UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

LEARNING RELATIONAL CONCEPTS WITH THE SPATIOTEMPORAL MULTIDIMENSIONAL RELATIONAL FRAMEWORK

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

MATTHEW BODENHAMER

Norman, Oklahoma

2014

LEARNING RELATIONAL CONCEPTS WITH THE SPATIOTEMPORAL MULTIDIMENSIONAL RELATIONAL FRAMEWORK

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dr. Andrew Fagg, Chair

Dr. Amy McGovern

Dr. Dean Hougen

Dr. Rickey Thomas

Dr. S. Lakshmivarahan

Acknowledgements

I would like to thank a number of individuals for their part in helping me to complete this dissertation. First, and foremost, I would like to thank Dr. Andrew Fagg, my advisor. He has provided an enormous amount of assistance in helping to guide and shape this dissertation into its final form, from providing many of the initial ideas for the SMRF project, to providing guidance and mentorship in the development of SMRF over the years, to providing thorough and helpful feedback on this document in its various stages of composition. It has been an excellent and educational experience to have worked in his lab over the past seven years. I would also like to thank my other committee members, Dr. Amy McGovern, Dr. Dean Hougen, Dr. Rickey Thomas, and Dr. S. Lakshmivarahan, for providing helpful suggestions and feedback over the course of this research.

Next, I would like to thank the following individuals who contributed to the development and implementation of SMRF in various degrees. Tom Palmer has provided many invaluable contributions to the present state of the learning algorithm, and SMRF would not be what it is today without his input. Dougal Sutherland worked in the summer of 2010 to implement and improve various aspects of SMRF, and his assistance was very helpful for transforming SMRF from a growing conceptual implementation into a powerful and fully-fledged experimental framework. Sam Bleckley worked in 2009 and 2010 to implement various aspects of the framework, and his assistance was very helpful for transforming SMRF from a set of theoretical ideas into a working implementation capable of providing results for a conference workshop paper. Daniel Fennelly also worked in the summer of 2009 to help create the first SMRF implementation.

Next, I would like thank all of my friends and family who provided much-needed support and encouragement for finishing this dissertation. And last, but not least, I would like to thank my wonderful wife, Jenny, for all of her love, support, and patience as I worked through this research and the composition of this document. You all have helped me much more than you know, and I count myself extremely blessed to have you all in my life.

Table of Contents

List of Tables

List of Figures

Abstract

The real world can be seen as containing sets of objects that have multidimensional properties and relations. Whether an agent is planning the next course of action in a task or making predictions about the future state of some object, useful task-oriented concepts are often encoded in terms of the complex interactions between the multi-dimensional attributes of subsets of these objects and of the relationships that exist between them. In this dissertation, I present the Spatiotemporal Multi-dimensional Relational Framework (SMRF), a data mining technique that extends the successful Spatiotemporal Relational Probability Tree models. From a set of labeled, multi-object examples of some target concept, the SMRF learning algorithm infers both the set of objects that participate in the concept, as well as the key object and relational attributes that characterize the concept. In contrast to other relational model approaches, SMRF trees do not require that categorical relations between objects be defined *a priori*. Instead, the learning algorithm infers these categories from the continuous attributes of the objects and relations in the training data. In addition, the SMRF approach explicitly acknowledges the covariant, multi-dimensional nature of attributes, such as position, orientation, and color, in the creation of these categories.

I demonstrate the effectiveness of the learning algorithm in three-dimensional domains that contain groups of objects related in various ways according to color, orientation, and spatial location. The learning algorithm is further shown to be robust to the addition of various kinds of noise to the data. I compare SMRF to other related algorithms and show that it outperforms each of them substantially on relational classification tasks, especially when noise is added to the data. I also show that SMRF handles the addition of extra objects to problem domains much more

efficiently than most of its competitors, which empirically exhibit polynomial and exponential increases in running time.

Chapter 1

Introduction

Some aspects of the world can be modeled as collections of objects, each with a set of associated attributes. Whether it is a robot preparing to perform the next step in a cooking sequence or an agent generating warnings of severe weather, only a specific subset of the observable objects is relevant to making decisions about what steps to take next. In particular, the relevance of an object is determined by its attributes and relations to other objects. These attributes are often continuous and multi-dimensional, such as Cartesian positions or colors in a red-green-blue (RGB) space. Given a set of training examples, the challenge is to discover the objects that play the crucial roles in the examples as well as the description of the key object attributes and relations.

For example, in the context of a stacking task, the goal might be to construct a tower of objects, such that a orange object is above a red object, which is above a green object. This concept of a "green-red-orange tower" is defined in terms of the color attributes of the three objects, as well as two spatial relations, defined over the first and second, and the second and third objects, respectively. Learning this concept however, involves identifying which objects play a role in the concept, and identifying the important attributes and relations between objects that define what specific roles they play. The problem is compounded if the relations between objects are not known *a priori*, and must be inferred from the attributes of the various objects. This is not an easy problem to solve, but it is nonetheless the problem that the approach described in this dissertation is designed to address.

In this dissertation, I present the Spatiotemporal Multidimensional Relational Framework (SMRF). SMRF is designed to solve problems in a unique sub-domain of machine learning that draws upon elements from both Multiple Instance Learning (MIL) and Relational Learning. This sub-domain has a number of distinguishing features.

First, the quantities under consideration are both continuous and multidimensional. For example, the location of an object can be expressed as a vector in $\mathbb{R}^3$. This is in contrast to a number of machine learning approaches that are generally limited to either categorical data, or one-dimensional continuous data. While a hypothesis in a one-dimensional continuous domain can be defined in terms of a threshold value, hypotheses in multidimensional continuous domains require the construction of a decision volume. This is particularly true when the data exhibit covariance in various dimensions.

Second, the data in this problem domain do not consist merely of sets of continuous multidimensional values, but involve attributed objects, whose attributes are defined in terms of continuous multidimensional values. In addition, there may be many different kinds of attributes, with different dimensionality. For example, an object may have a two-dimensional Cartesian position attribute, a one-dimensional orientation attribute, and a three-dimensional RGB color attribute.

Third, the data in this problem domain are defined in terms of possibly heterogeneous groups of attributed objects. The groups may have different numbers of objects. In addition, the objects within a group may have different kinds of attributes. For example, a group might contain a "circle" object and a "rectangle" object. The "circle" object would have a radius attribute, while the "rectangle" object would have height and width attributes.

Fourth, the target concepts in this domain are defined in terms of relations. If a certain combination of relations between objects is present in a particular group,

then that group is said to contain the target concept, and is labeled positive. If a certain combination of relations between objects is not present in a particular group, then that group is said to not contain the target concept, and is labeled negative. For example, a target concept, expressed in English, might be something like "there is a 'red' object 'near' a 'blue' object." Formally, this can be expressed as:

$$\exists A \; \exists B \left(Red(A) \wedge Blue(B) \wedge Near(A, B) \wedge A \neq B \right). \tag{1.1}$$

In such a concept, *Red*, *Blue*, and *Near* are the relations (unary, unary, and binary, respectively) that serve to define the concept. In addition, the target concepts may be disjunctive, in that there may be a set of such target concepts, one of which is true for a given positive group. This makes the problem more difficult, as this essentially increases the number of problems to be solved for a given dataset.

Fifth, relations are not pre-defined, but instead are inferred directly from the training data. In the above example (Eq. 1.1), the relations *Red*, *Blue*, and *Near* are not defined beforehand, but learned inductively from labeled training data. Given that these relations are learned in terms of training data that is defined in terms of continuous multidimensional values, the space of these relations is itself uncountable and multidimensional.

Sixth, instances are defined in terms of specific orderings of objects taken from each group.[1] As such, this leads to a combinatorial explosion in the number of instances as the number of objects in each group increases. In the above example, for instance, there are ${}_n\mathrm{P}_2$ possible instances[2] that can be evaluated on the concept defined by Eq. 1.1, for a group of size n. This illustrates a significant problem with this domain – namely that inferring relations from the entire space of instances becomes intractable for any substantial group size.

[1] Ordering is important because non-symmetric relations can be, and are often, present.

[2] ${}_n\mathrm{P}_k$ denotes the number of possible k-permutations of n items.

As such, this problem domain encompasses some of the features of both MIL and relational learning. The goal in this domain is to learn a target concept from labeled groups of entities. Like MIL, these labels are determined by the values and/or configuration of some subset of the entities within these groups, and this information is not available *a priori*. Rather, these values and/or configurations are what must be learned, in order to correctly classify novel groups.

The concepts in this domain are defined in terms of relations. Unlike many relation learning problems, however, the relations are not pre-defined, but rather inferred from the training data. For domains such as Inductive Logic Programming, relations are categorical in nature, and are generally drawn from a relational database. In contrast, the inferred relations in this domain are continuous and multidimensional in nature, being defined in terms of volumes in certain metric spaces. In addition, many decision-tree based relational learning approaches employ group-oriented reasoning, sorting the group of objects as a whole down to a specific leaf node of the tree. These approaches are appropriate in these contexts because the target concepts are defined in terms of the group. In contrast, the concepts in this domain are defined in terms of certain attribute values of specific combinations of objects (i.e., instances) drawn from the group.

Because of its combination of relational and multiple-instance elements, this problem domain is unique in the field of machine learning, and as such, standard MIL and relational learning approaches are not generally suited to solving problems in this domain.

In summary, the problem domain that SMRF is designed to address contains the following features:

1. Continuous multidimensional quantities,

2. Attributed objects,

3. Groups of possibly heterogeneous objects,

4. Relationally-defined target concepts, which may be disjunctive,

5. Relations that are inferred from training data, and

6. Instances defined in terms of distinct object orderings.

As such, SMRF occupies a unique place in the domain of machine learning, since, to my knowledge, it is the first approach designed to solve problems in this specific domain.

The problem of learning these attributes and relations is a form of *relational learning* (Getoor and Taskar, 2007; Kemp and Jern, 2009). Most relational learning approaches are formulated in terms of graphical models (Friedman et al., 1999a; Taskar et al., 2002; Neville and Jensen, 2007), logical formulae (Blockeel and Raedt, 1998), or some combination thereof (Richardson and Domingos, 2006; Wang and Domingos, 2008). The work presented here, however, focuses on a different method of solving the problem, utilizing augmented decision trees. In particular, this approach is inspired by the successful Relational Probability Tree (RPT) (Neville et al., 2003) and the Spatiotemporal Relational Probability Tree (SRPT) (McGovern et al., 2008) approaches, which create probability estimation trees (Provost and Domingos, 2000), a form of a decision tree with probabilities at the leaves. Splits in the decision trees can ask questions about the observed properties of the objects or their relationships. Given a novel graph, these decision trees estimate the probability that the graph contains a set of objects that corresponds to some target concept. Like Kubica et al. (Kubica et al., 2003b,a), these approaches build models using pre-specified categorical relations.

Extending the SRPT framework, Bodenhamer et al. (2009) first introduced the Spatiotemporal Multidimensional Relational Framework (SMRF), which was designed to address the continuous, multidimensional, and relational aspects of real-world data.

While motivated by many aspects of the SRPT approach, SMRF trees represent a break in terms of structural form, as there area a number of structural differences between SMRF Trees and probability trees. Later, Bodenhamer et al. (2012) presented an augmented version the SMRF learning algorithm that is significantly more powerful than that presented by Bodenhamer et al. (2009). Palmer et al. (2014) also presented further improvements to the SMRF learning algorithm. This prior work is further improved upon by the work described in this dissertation. As such, the details of the SMRF trees in their most mature form are discussed in Chapter 4, and the details of the SMRF learning algorithm in its most mature form are discussed in Chapter 5. For historical purposes, some of the details of the evolution of the SMRF learning algorithm are given in Appendix E.

Some work has also been done to apply SMRF to a number of real-world problem domains. Looking towards the application of SMRF to tasks involving Learning from Demonstrations (Argall et al., 2009), Sutherland (2011) discusses a version of the SMRF approach that can accept cues from a human teacher to aid in learning the target concept. Another application of the SMRF approach is presented by Palmer et al. (2012), where ensemble techniques and support vector machine classifiers (Cortes and Vapnik, 1995) are used to predict the outcomes of actions in various dynamical domains. The work of Palmer et al. (2012) is done with an eye to the eventual application of SMRF to planning tasks in various robotics domains.

By virtue of being designed to solve problems in this unique domain, SMRF extends prior relational probability tree approaches in a number of significant ways. The first extension is the ability to ask questions based on continuous, multi-dimensional attributes. For example, the color of a pixel can be represented as a RGB tuple. Capturing a concept such as "yellow" requires that the blue variable be low but the green and red variables can take on values almost within their full range so long as they vary together. While RPTs and SRPTs can split on individual continuous

variables (Jensen and Getoor, 2003; Friedman et al., 1999a; Getoor et al., 2002), it is desirable to explicitly acknowledge the fact that multiple dimensions can covary in interesting ways. To address this issue, SMRF trees construct decision surfaces within multi-dimensional metric spaces. The SMRF tree learning algorithm selects the location of the surface so as to produce splits with high utility.

A second key extension made by the SMRF approach is the ability to define relational categories dynamically. For example, objects may have a position attribute defined within some global coordinate frame. The decision tree splits can be made within a metric space that captures the position of one object *relative* to another. As with the multi-dimensional object attributes, splits on these relational attributes are made using decision surfaces within the metric space. In contrast, RPTs and SRPTs ask relational questions using categorical descriptions of the attributes.[3]

A third key extension made by the SMRF approach is the ability to reason explicitly about object instances. In graph-based approaches (such as RPTs and SRPTs), the objects/relations that satisfy a particular question are typically not represented in such a way that they can be referenced by questions deeper in the tree. This limits the types of concepts that can be easily represented and can make it difficult to address the question of *who* the actors are that play the key roles in determining the graph label. We refer to this as a graph-based method because the query (a graph representing objects and relations) descends as a single unit through the decision tree. In contrast, an instance-based method (such as SMRF trees) explicitly represents the acting objects through an instantiation process. A question at one level in the tree can then refer to the set of objects that have been instantiated to that point. This allows absolute questions about individual objects or relative questions about two or more objects to be asked.

[3]Later work done by McGovern et al. (2013) upgraded the SRPT framework to handle some multidimensional quantities and infer some relations dynamically. SMRF was designed in part to extend the original version of SRPTs, which did not have these capabilities.

In addition to the above extensions to previous work, the SMRF approach also makes a number of general contributions. First, the SMRF approach robustly handles noisy data. Sensors provide errors in both the small and larger scales, including failures lasting many time steps. Real-world data also often provides a variety of "distractors" that do not play a role in the concept to be learned. The SMRF approach handles both types of noise. In addition, the SMRF approach can learn with training sets of limited size. The ability to learn on small data sets translates well into the real world, where data is often scarce. Finally, the SMRF approach produces human-readable trees, which are useful for better understanding the learned concept.

In this dissertation, I introduce and discuss SMRF trees and discuss experimental results that show SMRF to be a viable learning approach. In particular, in Chapter 2, I review background literature that pertains to various aspects to the SMRF approach and the problem domain it is designed to address. In Chapter 3, I define the problem domain in explicit detail, and provide a notational framework for describing SMRF in relation to the problem domain. In Chapter 4, I introduce and formally define SMRF trees. In Chapter 5, I introduce and formally define the SMRF learning algorithm.

Moving from theory to implementation, I discuss the method used to generate data for experiments in Chapter 6. In Chapter 7, I describe the experiments performed to demonstrate the viability of SMRF as a learning approach, and present the results of these experiments. In Chapter 8, I discuss the significance and implications of these experimental results, as well as the relation of SMRF to related prior work. In Chapter 9, I present suggestions for future work. Finally, in Chapter 10, I present the conclusions of the dissertation.

Some supplemental material is also included in appendices, which may be skipped without loss of continuity. They are included however, as they will likely be of interest to some readers.

Chapter 2

Related Work

The work in this dissertation is related to prior work in the domains of tree-based learning and multiple instance learning (MIL), as well as to other graphical-model based methods in the field of statistical relational learning (Getoor and Taskar, 2007). In this chapter, I provide relevant background information on these approaches.

2.1 Tree-Based Learning

Tree-based learning methods have a long and venerable history in the field of machine learning. Tree-based learning methods are often utilized, in part, for their ability to represent concepts in a compact, expressive, and human-readable form. Furthermore, decision trees are robust to noisy data and are capable of naturally expressing disjunctive concepts (Mitchell, 1997). The tree-based learning approaches discussed in this section extend the basic decision tree framework in various ways. At the most basic level, the *decision tree* is a means of representing a discrete-valued function through the means of a binary decision structure. Decision trees can be used to classify instances by sorting them down through the tree. Each internal node of the decision tree poses a binary question regarding some aspect or attribute of the instance, and the instance is sorted down either the "yes" branch or the "no" branch, depending upon the answer of the binary question for this particular instance. This sorting process continues until the one of the leaf nodes is reached, which contains a class label. The basic decision tree, then, classifies the instance according to the class label of the leaf node into which it is sorted.

2.1.1 Basic Decision Trees

Basic decision trees (Mitchell, 1997) can contain internal nodes that split either on categorical or uni-dimensional continuous attributes of the data instance. For categorical data, a split is defined by enumerating the set of acceptable categorical values. For example, a categorical decision node might sort any instance x down the yes branch for which $\textsc{Shape}(x) \in \{\textsc{Square}, \textsc{Circle}\}$. Data instances for which $\textsc{Shape}(x) = \textsc{Trapezoid}$, for example, would be sorted down the no branch. For continuous values, a partition can be made in the continuous space by use of a threshold. For example, a continuous decision node might sort any instance x down the yes branch for which $\textsc{Age}(x) \leq 35$. A data instance, for example, for which $\textsc{Age}(x) = 42$, would be sorted down the no branch.

There are a number of methods for learning decision trees from a set of labeled examples. The most popular learning algorithms involve so-called *top-down induction*, wherein the tree is repeatedly expanded (by replacing leaf nodes with a decision node / leaf node sub-tree) in a greedy fashion until a maximal sorting of the training data instances is obtained at the leaf nodes. Maximal sorting is defined in terms of a metric, which is generally some form of minimal information entropy of the leaf-node sorting. Two of the most popular algorithms of this kind are **ID3** (Quinlan, 1986) and its successor **C4.5** (Quinlan, 1993).

For example, with **ID3**, the metric used to score candidate splits is the information gain of a candidate split. In order to define this formally, let S denote a set of instances, and let $C_\lambda(S)$ denote the set of instances which belong to label λ. Given this, the information gain provided by splitting over an attribute A is defined as:

$$\textsc{Entropy}(S) = \sum_{\lambda \in \Lambda} -\frac{|C_\lambda(S)|}{|S|} \log_2 \left(\frac{|C_\lambda(S)|}{|S|} \right), \text{and}$$

$$\textsc{InformationGain}(S) = \textsc{Entropy}(S) - \sum_{v \in \textsc{Values}(A)} \frac{|S_v|}{|S|} \textsc{Entropy}(S_v),$$

where S_v denotes the subset of instances in S for which attribute A has value v, and VALUES(A) denotes the set of possible values for attribute A. The **ID3** algorithm recursively evaluates new splits based on the information gain metric until either (1) all of the instances in set S have the same label, or (2) all possible attribute values have been used in splits already.

ID3 can only learn trees with categorical splits. **C4.5** extends **ID3** by enabling continuous splits to be learned as well, through the selection of a decision threshold that partitions the continuous attribute space. In addition, **C4.5** introduces a post-pruning step, which prunes away branches in a bottom-up fashion, replacing certain branches with leaf nodes, in accordance with some pruning criterion. Post-pruning results in smaller trees, which helps to prevent overfitting. Despite the popularity of top-down induction, algorithms that perform bottom-up induction have been developed as well (Barros et al., 2011).

Decision trees can be learned as individual models to solve a classification problem. However, there are many contexts in which using a collection of models to solve a particular problem may yield better results. This principle has given rise to a number of *ensemble learning* techniques, which seek to learn a collection of hypotheses and combine their predictions (Russell et al., 1995). As such, ensemble methods seek to learn a collection of diverse, accurate models which will collectively yield accurate classifications. A number of techniques have been developed for decision trees in this regard, the most popular and successful of which include *bootstrap aggregating* (also known as *bagging*), *adaptive boosting* (also known simply as *boosting*). These and other decision tree ensemble learning techniques are called *random forests* (Breiman, 2001).

Bagging (Breiman, 1996) is an ensemble method whereby each tree in the ensemble is trained from a "bootstrapped" version of the original training set. Each individual training set is acquired by sampling some fixed number of examples from

the original training set (with replacement). Boosting (Freund and Schapire, 1995) is a technique that maintains a set of weights for the examples in the training set. The weight of each example is increased if it is classified incorrectly by the learned tree, and decreased if it is classified correctly. A new training set can be derived from these weights, by sampling from the training set, where the probability of an example being selected is proportional to its weighting. This method, called *boosting by sampling* helps to ensure that different subsets of the training examples are covered by some of the members of the ensemble. An alternative method, called *boosting by weighting*, delivers the training set together with the weights if the learning algorithm being utilized is capable of learning from weighted data. Dietterich (2000) performs an empirical comparison of bagging and boosting applied to **C4.5**. This comparison suggests that in cases with little classification noise (where the example labels are largely accurate), boosting provides superior performance. However, in cases where classification noise is prevalent (where there are a substantial number of incorrect example labels), bagging appears to provide superior performance. Breiman (2001) unites these and other ensemble learning techniques in a common theoretical framework of random forests.

2.1.2 Probability Trees

Basic decision trees classify only according to a categorical class label. In the simplest case, this may simply be the set $\{True, False\}$. In more complex cases, there may be many possible class labels. However, given the uncertainty often present in real-world scenarios, it is often desirable to estimate the probabilities of the various classes, as opposed to simply returning a label itself. As a result, the decision tree framework has been extended to define so-called *probability trees* (also called *probability estimation trees*). Probability trees are similar to decision trees, with the difference being that probability tree leaf nodes contain a probability distribution defined over the classes,

as opposed to the label of one of the classes.

Probability trees are learned with top-down induction algorithms similar to those used to learn decision trees. Some approaches utilize post-pruning to avoid overfitting. In this regard, Provost and Domingos (2000) discuss the application of **C4.5** to probability trees. However, Provost and Domingos (2000) argue that post-pruning can be harmful to probability trees, given the fundamental differences between probability estimation and majority class estimation. As a result, they propose using **C4.4**, which is a version of **C4.5** without post-pruning. Alternative approaches include that of Friedman and Goldszmidt (1998), which utilize minimum description length (MDL) as a scoring metric. Drawing upon the MDL method, Fierens et al. (2005) apply the general Bayesian information criterion (BIC) (Schwarz, 1978) as a means of scoring candidate trees. Other approaches utilize statistical testing. For example, Neville et al. (2003) utilize a Chi-square test as a scoring metric for learning relational probability trees (Section 2.1.3).

Fierens et al. (2005) perform an experimental comparison of these learning algorithms, and determine that C4.4 is best on average. C4.5 performs nearly as well on average, but has the advantage of producing smaller trees. When the number of classes is small, the approach utilizing BIC performs nearly as well, and produces trees considerably smaller than either than those produced by C4.5 or C4.4.

Since probability trees define probability distributions over the classes present, they can be used to define conditional probability distributions within Bayesian networks (Friedman and Goldszmidt, 1998).

2.1.3 Relational Probability Tree Approaches

Standard probability tree learning techniques are not well-suited to handle relational data. As Neville et al. (2003) note, relational data violates two key assumptions of traditional propositional classification techniques. First, traditional propositional

techniques assume that the data are independent and identically distributed, while relational data often have dependencies. Second, traditional propositional techniques assume that the data are recorded in homogeneous structures, such as a database table with fixed set of fields. However, relational data instances are often structured in heterogeneous ways.

To address these issues, Neville et al. (2003) present Relational Probability Trees (RPTs), which extend the traditional probability tree approaches in order to handle relational data in a more natural manner. RPTs are structurally identical to probability trees. Like probability trees, RPTs define a probability distribution at the leaf nodes. Unlike probability trees, RPTs use decision nodes that can directly acknowledge certain relational aspects of the data. The relational aspect of the data is addressed through the use of aggregation functions (such as MEAN, MODE, and COUNT), which dynamically "propositionalize" the data. For example, in a movie classification tasks, different movies might have different numbers of actors. However, using the MEAN aggregation function, the tree can obtain an aggregate value for any numeric actor attribute, such as age. Such aggregation functions are then used to define features that split the data. With such features in hand, traditional top-down induction techniques can be used to learn trees to predict the target label.

Neville et al. (2003) apply RPTs to domains in which each training example is a graph of objects. Each graph contains a number of objects, and each object has some number of attributes. Additionally, edges (or relations) in the graph define relationships between the objects. In each graph, there is a target object, for which some attribute defines the label of the graph as a whole. For example, in one dataset, each graph contains a number of nodes representing web pages, and the label of the graph is determined by whether or not the target web page is the homepage of a university student. Although RPTs can handle graphs of arbitrary complexity, it is common to restrict examples to subgraphs where each object is at most k edges away

from the target. And if k is set to 0, the problem essentially becomes propositional, which illustrates the nature of RPTs as a relational extension of propositional tree learning approaches.

RPTs can handle both categorical and continuous real-valued attributes, through a variety of aggregation functions. For categorical attributes, RPTs can employ a function such as MODE, whereas for continuous attributes, RPTs could employ a function such as MEAN. In addition, RPTs can consider the attributes of other objects and relations in the graph, as well counts of certain objects and relations in the graph. To make use of these kinds of features, RPTs can use aggregation functions such as MAXIMUM, MINIMUM, and EXISTS, as well as functions such as COUNT, PROPORTION, and DEGREE.

The RPT learning algorithm of Neville et al. (2003) performs a form of top-down induction, searching the feature space in a greedy manner for binary splits that maximize the feature score, which is calculated using a chi-squared statistic, which is calculated from the contingency table of the question. Learning algorithm iteration continues until the p-value of the score associated with the best split exceeds a pre-defined threshold. To compensate for the multiple comparisons problem (Jensen and Cohen, 2000), a Bonferroni correction is used. In empirical results, Neville et al. (2003) report that the RPT learning algorithm is able to learn trees that are significantly and substantially smaller than trees produced by **C4.5** on a number of relational datasets.

Extending the RPT framework, McGovern et al. (2008) introduce Spatiotemporal Relational Probability Trees (SRPTs). SRPTs are designed to accurately predict the target label for relational data that can vary in both space and time, with the motivation of address problems in severe weather forecasting, such as cyclone formation and patterns of drought. SRPTs share the same formal structure as RPTs, but extend the latter framework by using different kinds of splits at the decision nodes. Also, due to

the temporal nature of the data used, data representation in the SRPT framework is quite different from that used by RPTs.

In the SRPT data representation, objects and their temporal qualities are represented differently, allowing for both a general Exists question (which asks if the object or relation is present in the graph) as well as a TemporalExists question (which asks is a particular object or relation is present for some period of time). The AttributeValue question can create splits over various aggregations of attribute values, and the TemporalGradient question can create splits based on the change of an attribute value with respect to time. In addition, there are three higher-level questions that can be defined in terms of other questions. The Count question looks at the number of objects that match a more basic question, and the Structural question looks at whether the type of a specific object matches the type of an object matched by another question. Finally, the TemporalOrdering question asks about the temporal relationships of the items matched by two different questions. These temporal relationships can be any one of the seven basic Allen relations (Allen, 1991).

The SRPT learning algorithm bears structural similarity to the RPT learning algorithm, in that both algorithms search for the best splits in a greedy top-down manner, scoring the splits with a chi-square statistic, and utilizing a p-value cutoff for question acceptance.

Supinie et al. (2009) apply random forest techniques to SRPTs, resulting in Spatiotemporal Relational Random Forests (SRRFs). For empirical comparison, Supinie et al. (2009) apply both SRPTs and SRRFs to data collected by a commercial airline, with the goal of predicting whether or not an aircraft will experience turbulence in certain weather conditions. SRRF forests are able to outperform the individual SRPTs on this dataset. Further work in this domain by McGovern et al. (2010) and McGovern et al. (2011) with the SRRF framework shows an increased ability

to predict turbulence, as well as applicability towards the prediction of drought and cyclone formation. McGovern et al. (2013) further improve the SRPT framework to allow important relationships between objects to be inferred from the data. This is accomplished by maintaining a representation of *fielded objects*, which contain a field of data points representing things such as wind speed and direction. Questions can then be formulated in terms of the field itself, looking at quantities such as gradient, curl, and divergence. In addition, shape-finding algorithms can detect formations of objects that approximate specific geometric shapes, such as circles, ellipses, spheres, and cylinders.

2.1.4 Logic-Based Probability Tree Approaches

Another set of tree-based learning approaches seeks to address problems within the domain of logic programming (Lloyd, 1987), which is concerned with the representation of concepts in terms of expressions in first-order logic. The field of *inductive logic programming* (ILP) (Muggleton, 1991) attempts to learn such representations from labeled relational data. Examples for ILP problems are commonly drawn from relational databases, where the relations are predefined in the various tables. A number of ILP systems, such as FOIL (Quinlan, 1990) and RIBL (Emde and Wettschereck, 1996), are not inherently tree-based. However, a number of tree-based learning systems have been successfully developed for the ILP domain. One well-known tree-based approach is TILDE, developed by Blockeel and Raedt (1998).

TILDE is an upgraded version of the **C4.5** algorithm, which applies top-down induction to the task of learning *first-order logical decision trees* (FOLDTs). The FOLDT is a tree-based decision structure that can perform classifications upon relational data. By contrast, basic decision trees can only perform classifications upon propositional data. FOLDTs are also called *relational decision trees* (Dzeroski, 2007), due to the fact that first-order logic is used to express relations between objects in

the problem domain. As such, relational decision trees can also be transformed into first-order decision lists (Bratko, 2001) for use in Prolog environments.

Operating in a first-order logical domain, each ILP example contains a set of objects, over which a number of attribute (unary predicates) and relations (n-ary predicates) are defined. At each decision node, such relational decision trees sort the entire example down either the yes branch or the no branch. Each decision node contains a logical formula. At a given decision node, the example is sorted down the yes branch if there exists a variable substitution that satisfies the formula. Otherwise, the example is sorted down the no branch. For example, suppose that a given decision node n_1 contains a formula $\phi(X, Y)$. If the example contains two objects, a and b, such that $\phi(a, b)$ is defined as true, then the example would be sorted down the yes branch. If, on the other hand, the example did not have any two such objects for which $\phi(X, Y)$ is defined as true, then it would be sorted down the no branch. Decision nodes down the yes branch also implicitly conjoin their formulae to those of the nodes preceding them. For example, if a decision node n_2 is on the yes branch of n_1 (which is also the root of the tree), and n_2 has the formula $\psi(Y, Z)$, then the example would be sorted down the yes branch of n_2 if and only if there were some combination of objects, that when mapped to the variables X, Y, and Z, satisfied the formula $\phi(X, Y) \wedge \psi(Y, Z)$. Thus, being sorted down the yes branch of n_2 is equivalent to stating that for such an example, it is the case that:

$$(\exists X)(\exists Y)(\exists Z)\left(\phi(X, Y) \wedge \psi(Y, Z)\right).$$

From this, it follows that sorting an example down the no branch of n_2 is equivalent to stating that it is the case that:

$$(\exists X)(\exists Y)\left(\phi(X, Y) \wedge \neg(\exists Z)(\psi(Y, Z))\right).$$

For this reason, decision nodes that are sorted down the no branch of a decision node n cannot make use of any variables over which the implicit compound formula at n is defined. No such use can be made because no such variables exist that satisfy that formula.

TILDE has a configurable learning bias, which can be set in terms of *refinement modes*, that allow the TILDE user to determine what kinds of variables can be utilized in forming decision nodes and in what contexts they can be used. As such, one must select refinement modes that are most appropriate for the problem one is attempting to solve in order for TILDE to perform optimally. In addition, TILDE allows the user to set parameters to control lookahead behavior. Lookahead allows multiple decision nodes to be added in one step. Lookahead may be necessary in some instances where a variable is required to produce further splits, but it's initial introduction does not produce any classification gain. As such, TILDE is able to perform multi-ply search over the space of possible splits within pre-defined boundaries. TILDE also provides support for numerical data, but handles such values by discretization. As such, TILDE can learn concepts in terms of discretized forms of numerical attributes and relations. However, due to the discretization process involved, any covariance present in the values of a multidimensional numerical attribute or relation cannot be directly acknowledged.

Logic-based tree learning approaches, such as TILDE, have been applied successfully to a number of problem domains. For example, Dzeroski et al. (1999) applies TILDE to the problem of predicting the biodegradability of a number of different chemicals, along with a number of other ILP approaches. TILDE, however, is outperformed by a number of other approaches in classification. In another instance, Dzeroski et al. (1998) apply TILDE and related ILP approaches to the problem of predicting the structure of diterpene compounds, given their nuclear magnetic resonance spectra. In these experiments, TILDE performs well, being outperformed only

by RIBL. However, unlike RIBL, the decision tree structures that TILDE learns are human-readable, which is advantageous in this context.

Like basic decision trees, the leaf nodes of relational decision trees contain a predictive class label. If the label is categorical, the problem is a classification problem, and can be solved by relational decision trees. If the label is continuous, then the problem is a regression problem, and must be solved by *relational regression trees*. With regression trees, each leaf node contains a continuous value, which serves as the class predictor. An application of regression trees to the ILP domain is found in the Structural Regression Tree approach (Kramer, 1996; Kramer and Widmer, 2001). Structural Regression Trees (SRTs) are learned using the SCART algorithm, which is an upgraded version of the CART algorithm (Breiman et al., 1984). Furthermore, relational regression trees that contain linear equations in the leaves are called *relational model trees* (Appice et al., 2008). Relational regression and model trees have the same kind of structure as decision trees, except for what is represented at the leaf nodes. These trees have been successfully applied to real-world applications such as predictions of real-valued quantities associated with mutagenic activity. These approaches, however, are not capable of directly predicting quantities in multidimensional spaces where the distribution of relevant values exhibits covariance, as might occur in certain physical domain problem settings.

2.2 Multiple Instance Learning

The multiple instance learning (MIL) problem (Dietterich et al., 1997) is a specific type of learning problem in which a target concept is learned to explain why various *bags* of instances are labeled in specific ways. The target concept must be learned without knowing which instances in each bag matches the target concept. In a binary MIL problem, the learning agent is given two groups of bags, one group in which the

bags are labeled positive, and another in which they are labeled negative. A bag is a set of instances.[1] In a positive bag, there must be at least one instance of the target concept, whereas in a negative bag, there can be no instances of the target concept. Solving an MIL problem is different from solving a standard supervised learning problem, since some information is given regarding the instances in the form of bag labels, but not enough information is given to identify exactly which instances in those bags match the target concept. The definition of the target concept, then, must be inferred from the data.

2.2.1 Axis-Parallel Rectangles

A number of methods have been devised for solving MIL problems. Dietterich et al. (1997) devised the first approach for solving MIL problems, which constructed axis-parallel rectangles (APRs) around the target concept in the instance space. A rectangle is defined in the instance space, such that instances "inside" the rectangle are classified as positive, and instances "outside" the rectangle are classified as negative. The rectangle axes are then shifted so as to maximize the correct classification of bags. This pioneering approach to solving MIL problems was applied to the pharmaceutical domain, in attempting to identify whether or not a given drug molecule will bind strongly to a target protein. Each molecule has a variety of shapes that it can take, many of which may not bind to the protein. If a molecule has at least one shape that will bind well, that collection of shapes (a bag) is considered positive. Otherwise, it is a negative bag. The datasets of this form provided by Dietterich et al. (1997), are called MUSK1, and MUSK2. Concerning axis-parallel rectangles, Dietterich et al. (1997) show that their approach significantly outperforms other supervised learning approaches, such as **C4.5** and neural networks with backpropagation, that do not take the MIL problem structure into consideration (i.e. learning from labelled bags

[1] In this context, the term *instance* denotes a vector of feature values.

instead of labelled instances).

The performance of the APR approach on the MUSK datasets has seen widespread use as a benchmark for later MIL approaches. However, despite such widespread acceptance as an MIL benchmark, Xu (2003) argues that there are good statistical reasons to question the validity of the MUSK datasets as real MIL problems. In addition, the MUTAGENESIS datasets (Debnath et al., 1991) have also been widely employed as benchmark datasets for various MIL approaches. While MUTAGENESIS is not fundamentally an MIL dataset, it can be transformed into an MIL-friendly format. However, Xu (2003) questions the validity of this dataset as an informative MIL benchmark. In addition, Lodhi and Muggleton (2005) question whether or not the dataset is sufficiently challenging for the broader domain of supervised learning. As such, while the classification performance of an algorithm on the MUSK and MUTAGENESIS has served as a *de facto* benchmarking standard for the MIL community, the work of Xu (2003) and Lodhi and Muggleton (2005) raise serious questions regarding the suitability of these datasets for this purpose. Nonetheless, as these datasets are used by the broader MIL community as benchmarks for assessing algorithm performance, the discussion of the following algorithms will include, in part, a discussion of their performance on these benchmarks in comparison to other well-known algorithms.

2.2.2 Diverse Density

Maron and Lozano-Pérez (1998) developed a more general approach to solving MIL problems that involves maximizing the *diverse density* (DD) of the instances from both positive and negative bags. Diverse density is a measure of the concentration of instances from positive and negative bags. DD is maximized as more instances from positive bags are concentrated in one area of the instance space, while instances from negative bags are separated from that group of instances. DD decreases as

either instances from positive bags are separated from that concentrated group, or as instances from negative bags are drawn closer to that group. Thus, the DD-based approach searches for the optimal point of high concentration of instances from positive bags concurrent with a low concentration of instances from negative bags. The point of maximum diverse density is considered to be the best description of the target concept.

As further explained by Maron (1998), diverse density is defined in terms of a conditional likelihood $p(t|B_i)$ of a hypothesized concept t, given a bag B_i. This conditional likelihood is defined separately for positive and negative bags. As it turns out, diverse density can be maximized with respect to t by maximizing $\hat{DD}(t)$, which is simply the product of the conditional likelihoods for all of the bags:

$$\hat{DD}(t) = \prod_{1 \leq i \leq n} p(t|B_i^+) \prod_{1 \leq i \leq m} p(t|B_i^-),$$

where n is the number of positive bags, m is the number of negative bags, and B_i^+ and B_i^- denote the ith positive and negative bags, respectively. For each hypothesized value t, a probability distribution $\Pr(B_{ij} \in c_t)$ is defined in terms of t and the individual points in each bag, which represents the probability that such particular member j or bag i belongs to the concept class c_t. As such, $\Pr(B_{ij} \in c_t)$ functions like a kernel defined around the point t. The conditional likelihood $p(t|B_i)$ is then defined in terms of this distribution. There are two kinds methods for defining this conditional likelihood: one that is smooth, and one that is non-smooth. The smooth approach defines the conditional likelihood in terms of Noisy-Or:

$$p(t|B_i^+) = 1 - \prod_j \left(1 - \Pr\left(B_{ij}^+ \in c_t\right)\right), \text{and}$$
$$p(t|B_i^-) = \prod_j \left(1 - \Pr\left(B_{ij}^- \in c_t\right)\right).$$

In this formulation, each member of the bag contributes towards the conditional likelihood. On the other hand, the non-smooth approach defines the conditional likelihood in terms of max:

$$p(t|B_i^+) = \max_j \left\{ \Pr\left(B_{ij}^+ \in c_t\right) \right\}, \text{and}$$
$$p(t|B_i^-) = 1 - \max_j \left\{ \Pr\left(B_{ij}^- \in c_t\right) \right\}.$$

In this formulation, only the "best" member of the bag (with respect to c_t) contributes towards the conditional likelihood. The difference in these two approaches is representative of the commitments that one must make in approaching an MIL problem. One must decide whether to treat bags in terms of their single "best" members, or to treat bags in terms of the contributions of each of their members.

The diverse density approach has also been modified to make use of the Expectation-Maximization (EM) algorithm (Dempster et al., 1977), leading to the development of the EM-DD algorithm (Zhang and Goldman, 2002), which maintains a set of hidden variables that denote whether or not each instance is target-conceptual. The core EM-DD algorithm starts with a point in the instance space as the target concept hypothesis (that is, the point believed to be the target concept). In the estimation step, using a generative model, every instance in each bag is assigned a value that represents the measure of responsibility that this instance has in determining the bag label. The maximally-responsible instances are selected from each bag. In the maximization step, these instances are used to compute a new hypothesis h', such that the diverse density of h' is maximal among all possible hypotheses. The algorithm then repeats at the estimation step, until convergence. Upon convergence, the current hypothesis h is considered to be the best description of the target concept.

Diverse density and its EM-DD variant have been applied to several problem domains. Maron and Lozano-Pérez (1998) apply diverse density to the MUSK datasets,

and conclude that while diverse density does not perform as well as some axis-parallel rectangle approaches (while it also performs comparably with others), they note that the DD approach involves no parameter tuning, unlike the highest performing APR approaches. On the other hand, (Zhang and Goldman, 2002) apply the EM-DD algorithm to the MUSK datasets, and report that their EM-DD approach outperforms all previous approaches, including the best axis-parallel rectangle approaches, on both MUSK1 and MUSK2.

Diverse-density has also been applied to the problem of Content-Based Image Retrieval (CBIR) (Datta et al., 2005). In this vein, Maron and Ratan (1998) apply diverse density to CBIR, using DD to classify natural scenes from the Corel Image Library into classes based upon the natural features they depict, such as fields, mountains, and waterfalls. Each image is transformed into a set of visual features, which comprise the bag corresponding to that image. Through empirical testing, Maron and Ratan (1998) demonstrate that diverse density can produce classification performance that is competitive with hand-crafted models. Yang and Lozano-Perez (2000) also apply diverse density to image classification, applying DD to both natural images, as well as images of objects such as cars, shoes, watches, and hammers. Yang and Lozano-Perez (2000) report that their approach performs similarly to the approach of Maron and Ratan (1998) on natural images. However, Yang and Lozano-Perez (2000) also report that their approach performs well on object images, which it was not designed to classify. Zhou et al. (2003) continue in this line of research applying diverse density to the classification of natural image scenes, while utilizing their own novel bag generator. They report that on their datasets, their approach outperforms the approach of Yang and Lozano-Perez (2000), but is worse than the approach of Maron and Ratan (1998).

Bringing EM-DD into the picture, Zhang et al. (2002) apply both DD and EM-DD to the task of classifying natural images, using a broader range of image processing

techniques than those used by Maron and Ratan (1998) or Yang and Lozano-Perez (2000). Zhang et al. (2002) report that the classification performance of DD and EM-DD are very similar, but that EM-DD can identify concepts approximately 15 times faster than diverse density itself. Given the reported classification results of (Zhang and Goldman, 2002) on the MUSK datasets, and the reported performance improvement in the CBIR domain (Zhang et al., 2002), it would appear that the combination of both diverse-density and expectation maximization, in the form of the EM-DD algorithm, provides a powerful means of solving certain kinds of MIL problems.

2.2.3 Clustering-Based Approaches

In addition to APR and DD-based approaches, a number of other methods have been proposed for solving MIL problems. Wang and Zucker (2000) apply the k-nearest neighbor (kNN) algorithm to the MIL domain, producing two algorithms: BAYESIAN-KNN and CITATION-KNN. In both algorithms, kNN is applied to the MIL domain through the use of the *Hausdorff Distance* metric (Edgar, 2008), which allows one to define the "distance" between two bags in a metric space. For two bags A and B, the Hausdorff distance between the two, denoted $H(A, B)$, is defined as:

$$H(A, B) = \max\left\{h(A, B), h(B, A)\right\},$$

where $h(A, B)$ is defined as:

$$h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\|.$$

As with the MAX approach to computing diverse density discussed above, the Hausdorff distance treats each bag in terms of the contribution of its most relevant member. The approaches developed by Wang and Zucker (2000) attempt to account for the

fact that the class of an unseen bag is not always the majority class of its k-nearest neighbor bags. In the BAYESIAN-KNN approach, the class label c_u of the "unseen" bag is chosen to maximize the posterior probability of the classes of its k-nearest neighbors, such that:

$$c_u = \arg\max_c \; p(c_1, c_2, \ldots, c_k|c)p(c).$$

Inspired by the notion of *citation* (Garfield and Merton, 1979) from the field of library science, the CITATION-KNN approach defines relationships between bags in terms of *reference* and *citation*. The R-nearest $referrers$ of the unseen bag are defined as the R-nearest neighbors (via Hausdorff distance), and the C-nearest *citers* of the unseen bag are defined C-closest bags in terms of similarity rank. The quantities R and C are then split according to the distribution of positive and negative bags that comprise them, such that $R = R_p + R_n$ and $C = C_p + C_n$. Then, if $(R_p + C_p) > (R_n + C_n)$, the bag is labeled positive, otherwise negative.

The approach of Wang and Zucker (2000) benefits from the fact that it can make use of an established supervised learning method in the kNN algorithm. However, unlike DD-based approaches, kNN-based approaches cannot provide label estimates for individual instances.

Wang and Zucker (2000) apply both BAYESIAN-KNN and CITATION-KNN to the MUSK datasets. On both musk datasets, CITATION-KNN outperforms BAYESIAN-KNN. On the MUSK1 dataset, CITATION-KNN performs as well as the best APR algorithm. On the MUSK2 dataset, CITATION-KNN is outperformed by the best APR algorithm.

2.2.4 Tree-Based Approaches

Applying decision tree techniques to the MIL domain, Zucker and Chevaleyre (2000) extend the ID3 algorithm to learn decision trees defined over multiple-instance datasets, resulting in an algorithm they call ID3-MI. The ID3-MI algorithm utilizes a modified form of decision tree, called a *multiple decision tree*. In mono-instance problem domains, in which basic decision trees are designed to function, each instance receives its own label. Thus, each instance is sorted down the decision tree into one leaf node. In multiple-instance problem domains, however, each bag receives its own label, and the instances within the bag do not. Thus, in the multiple decision-tree, multiple instances from the same bag may be sorted into different leaves. Structurally, the multiple-instance tree is identical to the standard decision tree, but in order to use such a tree to classify a bag, one must take the max over the values returned by the tree for each of the instances. The usage of max follows from the definition of the multiple-instance problem: if there is one instance that is classified as positive, then the whole bag must be classified as positive. If there are no instances that are classified as positive, then the whole bag must be classified as negative.

The ID3-MI algorithm extends the ID3 algorithm by re-defining entropy and information gain in ways that respect the multiple instance problem structure. As defined by Zucker and Chevaleyre (2000), given a set of instances S, where $\pi(S)$ and $n(S)$ denote the number of positive bags represented in S and the number of the negative bags represented in S, respectively,

$$\textsc{Entropy}(S) = \frac{-\pi(S)}{\pi(S)+n(S)}\log_2\left(\frac{\pi(S)}{\pi(S)+n(S)}\right) - \frac{n(S)}{\pi(S)+n(S)}\log_2\left(\frac{n(S)}{\pi(S)+n(S)}\right)$$

$$\textsc{InformationGain}(S,A) = \textsc{Entropy}(S) - \sum_{v\in\textsc{Values}(A)} \frac{\pi(S_v)+n(S_v)}{\pi(S)+n(S)}\textsc{Entropy}(S_v),$$

where S_v denotes the subset of instances in S for which attribute A has value v, and

VALUES(A) denotes the set of possible values for attribute A. As in ID3, the split is chosen that maximizes information gain. In addition, once ID3-MI correctly classifies an instance from a bag from the training set, it removes from consideration all other instances from that bag. The motivation behind this restriction is to avoid learning overly-complicated trees. Unfortunately, Zucker and Chevaleyre (2000), provide no empirical analysis of the ID3-MI algorithm.

Blockeel et al. (2005) take a similar approach and construct an algorithm for constructing multiple decision trees called MITI (Multiple-Instance Tree Inducer). MITI is a variant of the ID3 algorithm that recursively introduces splits until all leaf nodes contain only instances of one label. Similar to ID3-MI, when MITI encounters a leaf containing only positive instances, it removes all other instances from the bags that those instances belong to. Unlike most top-down induction algorithms, which build the tree in a depth-first order, MITI builds the tree in a best-first order, where "best" is defined in terms of one of three possible sorting metrics. One metric is the unbiased proportion of instances present. Let p denote the number of positive instances present, and n the number of negative instances present. The unbiased proportion heuristic sorts the nodes by order of $\frac{p}{n+p}$. A second metric is defined in terms of the Laplace estimate, which is $\frac{p+1}{n+p+2}$. This formula is also known as the *rule of succession*, which was utilized by Laplace to address the sunrise problem (Jaynes, 2003). The last metric is called the *to zero estimate*, and is defined as $\frac{p}{n+p+k}$, where k is a parameter that defines how strongly the estimate is pulled towards zero. These metrics are called *biased estimates for the proportion of positives (BEPPs).*

MITI can utilize one of five possible splitting metrics, two of which are defined in terms of the chosen sorting heuristic. The first metric is defined as the maximum BEPP of the child leaves. The second metric is defined in terms of the sum of squared BEPPs of the child leaves. The other three metrics include the prediction accuracy of the resultant tree, as well as the bag entropy metric used by Zucker and Chevaleyre

(2000) and the Gini index used by CART (Breiman et al., 1984). In addition, MITI can learn trees based upon weighted instances. Blockeel et al. (2005) discuss either weighting instances uniformly, or weighting each instance as the inverse of the size of the bag from which it was drawn.

Blockeel et al. (2005) compare MITI to both TILDE and a modified version of ID3-MI (called ID3-MI') on the MUSK and MUTAGENESIS datasets. The empirical comparison shows that MITI is able to out-perform both TILDE and ID3-MI' on the MUSK dataset. On these datasets, TILDE also out-performs ID3-MI'. However, on the MUTAGENESIS datasets, which do not fundamentally define multiple-instance problems (Xu, 2003), performance is relatively comparable between the algorithms. The authors note that TILDE is technically able to learn MIL problems given a particular learning bias. However, given that TILDE trees sort bags, while MITI trees sort instances, TILDE can end up learning concepts that are outside of the MIL domain, by defining concepts in terms of questions that are universally quantified over the instances in a bag. However, the instance oriented approach of MITI enables it to outperform the more general bag-oriented approach of TILDE.

Bjerring and Frank (2011) further extends this approach by applying ensemble techniques to the MITI algorithm, as well as modifying the MITI algorithm to utilize ensemble techniques to learn rule sets, as opposed to individual trees (resulting in an algorithm called MIRI). Both techniques perform comparably on classification tasks, though MIRI tends to produce more compact concept representations. A different approach to MIL utilizing ensemble techniques is presented by Leistner et al. (2010), who build forests of randomized trees (Shotton et al., 2008), with an eye towards application in the domain of computer vision. This technique, called MI-FOREST, builds a random forest of randomized trees, using deterministic annealing as an optimization procedure. The technique performs comparably to other SVM-based approaches on image classification tasks, as well as on the MUSK datasets.

2.2.5 Support Vector Machines

Taking a different approach to the MIL problem, Andrews et al. (2003) develop an approach in terms of support vector machines (SVMs) (Cortes and Vapnik, 1995). SVMs can be applied to the MIL domain by organizing the data as follows. All of the instances are grouped together in a set $X = \{x_1, x_2, \ldots, x_n\}$. There are then m bags $B_1, B_2, \ldots, B_m$ defined in terms of index sets I, where each $I \subseteq \{1, 2, \ldots, n\}$. A bag defined in terms of an index set I is denoted B_I. The label of instance x_i is denoted y_i, and label of bag B_I is denoted Y_I. For positive instances, $y_i = 1$, and for negative instances, $y_i = -1$. Given the formulation the label of a bag B_I can be defined as:

$$Y_I = \max_{i \in I} y_i. \tag{2.1}$$

Equation 2.1 can then be formulated as a set of two constraints, such that:

$$\sum_{i \in I} \frac{y_i + 1}{2} \geq 1, \forall I \text{ s.t. } Y_I = 1, \tag{2.2}$$

and

$$y_i = -1, \forall I \text{ s.t. } Y_I = -1. \tag{2.3}$$

Given these constraints, Andrews et al. (2003) develop a method called MI-SVM, which is defined in terms of a soft-margin SVM. This SVM is defined as:

$$\min_{\{y_i\}} \min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C \sum_i \xi_i,$$

such that $\forall i$:

$$y_i \left(w^T x_i + b \right) \geq 1 - \xi_i,$$

$$\xi_i \geq 0,$$

$$y_i \in \{-1, 1\},$$

Equation 2.2 holds, and

Equation 2.3 holds.

In this approach, a soft-margin criterion is jointly maximized over both possible hyperplanes, as well as possible label assignments. Unfortunately, this leads to a mixed-integer programming problem (Wolsey, 1998). In order to cope with the practical intractability of the problem, Andrews et al. (2003) devise an optimization heuristic that takes an iterative two-step approach of first finding a hyperplane, then finding labels that minimize the objective function.

In addition, Andrews et al. (2003) develop another technique, called MI-SVM,[2] which defines margins in terms of bags, as opposed to MI-SVM, which defines margins in terms of instances. The functional margin of a bag with respect to a hyperplane, γ_I, can be defined as

$$\gamma_I = Y_I \max_{i \in I} \left(w^T x_i + b \right). \tag{2.4}$$

As with diverse density above, one can commit to solving an MIL problem either through an instance-based formulation, or a bag-based formulation. Given Equation 2.4, MI-SVM is defined as follows:

$$\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_I \xi_I,$$

[2]MI-SVM is a distinct algorithm from MI-SVM

such that $\forall I$:

$$Y_I \max_{i \in I} \left(w^T x_i + b \right) \geq 1 - \xi_I, \text{ and}$$

$$\xi_I \geq 0.$$

After further re-formulation, MI-SVM also ends up being a mixed integer programming problem. As with MI-SVM above, Andrews et al. (2003) devise an optimization heuristic that takes an iterative two-step approach of first finding a hyperplane using quadratic programming (Nocedal and Wright, 2006), then maximizing a "selector function" that identifies the representative positive instance in the positive bags (called the *witness*).

Andrews et al. (2003) apply both SVM methods to the MUSK dataset, and compare against other approaches. They report that MI-SVM performs comparably to DD, and outperforms EM-DD on MUSK1. This is certainly an interesting result, as it contradicts Zhang and Goldman (2002), who report that EM-DD outperforms DD as well as the best APR approach. Andrews et al. (2003) also report that MI-SVM performs worse than MI-SVM on MUSK1, but better on MUSK2.

Mangasarian and Wild (2008) take a different approach to the bag-based SVM solution in formulating their algorithm, MICA. Instead of selecting a single "witness" from each positive bag, they maintain an arbitrary convex combination of points from each bag. In addition, their cost function utilizes the 1-norm, as opposed to the 2-norm, which allows the problem to be solved with a succession of linear programs, as opposed to quadratic programs. This allows a solution to be derived that does not rely upon heuristics, and provably converges to a local solution. Mangasarian and Wild (2008) report that MICA outperforms all other SVM-based MIL approaches, as well as APR, on the MUSK2 dataset. They also report that MICA is outperformed by MI-SVM on the MUSK1 dataset, but outperforms MI-SVM

Gehler and Chapelle (2006) extend the ideas of Andrews et al. (2003), developing two SVM-based algorithms that make use of deterministic annealing (Rose, 1998)

in their optimization algorithms. They develop an instance-based algorithm (AL-SVM) and a bag-based (or witness-based) algorithm (AW-SVM). In addition, they formulate a version of AL-SVM, called ALP-SVM, which utilizes a constraint controlling the number instances assigned to positive bags. For empirical comparison, they report that, with respect to instance-based algorithms, ALP-SVM outperforms AL-SVM on both MUSK datasets, as well as out-performing MI-SVM on the MUSK2 dataset. With respect to bag-based algorithms, AW-SVM outperforms both MICA and MI-SVM on MUSK1, but is greatly outperformed by MICA on MUSK2.

2.2.6 Other Approaches

Various other approaches have been devised to address the MIL domain. Zhou and Zhang (2002) apply neural networks to the MIL domain, extending the backpropagation algorithm with an error function capable of accounting for the multiple-instance nature of the training data. They report that their algorithm performs comparably to diverse density on MUSK2, but somewhat worse on MUSK1. Zhang and Zhou (2006) develop a new neural network approach, which utilizes both the Hausdorff distance and singular value decomposition (Golub and Kahan, 1965). They report that their algorithm performs comparably to the best APR approach, performing slightly better on MUSK2 and slightly worse on MUSK1.

Viola et al. (2006) apply boosting techniques (Mason et al., 1999) to the MIL domain and the problem of object detection in images, defining the data likelihood function in terms of Noisy-Or. Babenko et al. (2009) and Babenko et al. (2011) also employ boosting-based MIL techniques for object detection, that are defined in terms of Noisy-Or. Lin et al. (2009) and Zeisl et al. (2010) also employ boosting-based MIL techniques for object detection, but formulate the approach in terms of the geometric mean function, which they claim is more suitable than Noisy-Or in cases where there is a large number of samples. Kim and De la Torre (2010) apply Gaussian processes

(Rasmussen, 2004) to the MIL domain, demonstrating capability on text and image classification tasks, as well as the standard MUSK datasets.

2.3 Graphical Models

There are a number of graphical model-based approaches that are designed to learn concepts in relational data. *Graphical models* are a graph-based mathematical formalism that facilitate the modeling of a joint probability distribution over a set of random variables. In practice, modeling the joint probability distribution of n variables requires 2^n entries, which is intractable for problems of any substantial size. Rather, graphical models allow one to define relationships between the variables, and given an assumption of conditional independence, define the joint distribution in terms of a product of conditional and marginal distributions. There are two main forms of graphical models: Bayesian networks (BNs) and Markov random fields (RMFs). Bayesian networks are defined in terms of directed graphs, where the direction of an edge implies a causal or explanatory relationship. In BNs, each node defines a probability distribution. Markov random fields, by contrast, are defined in terms of undirected graphs, which allows for a simpler representation of the relationships between the variables. However, individual nodes do not define probability distributions, but rather *potential functions*, which are not limited to the domain $[0, 1]$. The network as a whole defines a probability function over the variables in terms of the product of the potentials, divided by a normalizing factor, called the *partition function*.

Graphical model-based approaches are more general in scope than tree-based and multiple instance learning approaches. The latter are constrained to simply predicting a class label. By contrast, graphical model-based approaches can be used to predict the value of any variable, given that the values of the other variables are known. The

values of the predicted variable implicitly define a class label, but this need not be defined explicitly. Graphical models define probability distributions, and thus are naturally probabilistic in classification prediction.

Graphical models were originally developed to learn concepts in propositional data. However, a number of approaches have been developed that extend traditional graphical model-based methods by upgrading them (Kersting and Raedt, 2001; Kersting et al., 2003) from the ability to represent propositional data to the ability to represent relational data.

A number of such approaches are based upon Bayesian networks. One such class of approaches is called Probabilistic Relational Models (PRMs) (Friedman et al., 1999b), which upgrade Bayesian networks to the representation of heterogeneous relational data. Traditional Bayesian networks are unsuited to learning from and performing inference over heterogeneous data, as well as representing probabilistic dependencies among different instances of the same data set. PRMs solve this problem by modeling the general probabilistic dependencies among the attributes and relations (called a model graph), and then applying these to each of the elements in the data set by unrolling the model graph (resulting in an inference graph). A number of PRM approaches have been developed. Relational Bayesian Networks (RBNs) (Friedman et al., 1999a) extend traditional Bayesian networks in the method just described. Relational Dependency Networks (RDNs) (Neville and Jensen, 2007) also extend Bayesian networks in a manner similar to RBMs, though they allow cycles in the model graph, in order to model autocorrelation in the data. This approach also makes use of RPTs, utilizing them to define conditional probability distributions at various nodes. The fact that learned probability trees are used to represent a single probability distribution within a wider graphical network should serve to illustrate the relationship and differences between tree learning and graphical model based approaches.

Other approaches upgrade MRFs to the representation of Relational Markov Networks (RMNs) (Taskar et al., 2002) extend Markov learning techniques to the relational domain, performing maximum *a posteriori* parameter estimation and using Gaussian priors. Other density estimation approaches extend first-order logic environments to allow probabilistic reasoning grounded in graphical models, such as Markov Logic Networks (MLNs) (Richardson and Domingos, 2006), which ground first-order formulae in an undirected Markov model. MLNs are extended by Hybrid Markov Logic Networks (HMLNs) (Wang and Domingos, 2008), which allow the modeling of both logical formulae and numeric quantities.

Chapter 3

Grounding Relational Spatiotemporal Concepts

In this chapter, I define the problem domain that SMRF is designed to address and explore the rationale for the design of the SMRF framework from a mathematical perspective. The principles developed in this chapter form the foundation for the framework and learning algorithm presented in Chapters 4 and 5, respectively.

Fundamentally, the work presented in this dissertation is motivated by the problem of learning target concepts defined in terms of attributed objects and the relations between them. Important continuous, multidimensional, spatiotemporal, relational concepts are defined in terms of object properties and relations. For instance, in the context of a stacking task, the goal might be to construct a tower of objects, such that a orange object is above a red object, which is above a green object. This concept of a "green-red-orange tower" is defined in terms of the color attributes of the three objects, as well as two spatial relations, defined over the first and second, and the second and third objects, respectively. In this context, a specific color represents a point in RGB space, while the spatial relations in this example represent points in a Euclidean space denoting relative location.

The problem that SMRF aims to solve is that of learning a minimal tree-structured representation of these spatio-temporal concepts in a manner consistent with the training data. The problem can be formally defined as follows.[1] First, let $\mathcal{O}$ be the set of objects that form the domain of discourse of the current task – that is, the

[1]The following ontology is not intended to be an exhaustive representation the real world. Rather, it is simply intended to represent the salient features of the learning domain that SMRF is designed to address.

set of all objects that are involved in defining or learning a particular target concept. The concepts with which SMRF is concerned are defined fundamentally in terms of objects, and $\mathcal{O}$ represents the set of such objects over which concepts can be defined.

Second, as these objects have attributes, let $\mathcal{A}$ be the set of attributes defined over the objects in $\mathcal{O}$. $\mathcal{A}$ is a set of the form $\{A_1, A_2, \ldots, A_{|\mathcal{A}|}\}$, where A_i is an individual attribute defined over the objects in $\mathcal{O}$. For example, A_1 might denote "color," A_2 might denote "orientation," etc. An attribute is an association of a specific kind or type of value to an object. For instance, a location attribute associates the physical position of an object to some point in a metric space. Thus, an individual attribute A_i is a function of the following form: $A_i : \mathcal{O} \rightarrow \mathbf{S}_{A_i}$. For spatiotemporal attributes, $\mathbf{S}_{A_i}$ is a metric space, typically (though not always) a Euclidean space $\mathbb{R}^n$. For future reference, let $\mathbf{S}_{\mathcal{A}}$ represent the union of all of the codomains $\mathbf{S}_{A_i}$ of the attributes $A_i \in \mathcal{A}$. The value of attribute A_i for some object o is denoted as $A_i(o)$. One can also make use of *dot notation* and represent this attribute value, equivalently, as $o.A_i$. For example, if A_i denotes the "color" attribute, then the color of an object o could be denoted $color(o)$ or $o.color$. In dynamic contexts, attribute values can be time-indexed to indicate a change in value as the result of a temporal process. In such contexts, the value of attribute A_i for an object o at time t is denoted as $A_i(o, t)$, or in dot notation, as $o.A_{i,t}$.

Third, as there are relations defined over these objects, let $\overline{\mathscr{R}}$ be the set of relations defined over the objects in $\mathcal{O}$. The set $\overline{\mathscr{R}}$ is of the form $\{\mathscr{R}_1, \mathscr{R}_2, \ldots, \mathscr{R}_{|\overline{\mathscr{R}}|}\}$, where $\mathscr{R}_i$ is an n-ary relation over $\mathcal{O}$, with $n \geq 1$ for any $\mathscr{R}_i$. Following Suppes (1972), each relation $\mathscr{R}_i$ is defined as a set of n-tuples, where each n-tuple is comprised of objects from $\mathcal{O}$. That is, if $\mathscr{R}_i$ is of arity n, then $\mathscr{R}_i \subseteq \mathcal{O}^n$. It should be noted that each $\mathscr{R}_i$ denotes the *extension* of each relation – that is, the ordered sequences of elements that are related in a certain way. For example, if there are only two objects in $\mathcal{O}$, x and y, that stand together in an "above" relation, such that x is "above"

y, then $\mathscr{R}_{\text{above}} = \{(x, y)\}$. At this level of description, no commitment is made as to why the objects in each element of $\mathscr{R}_i$ are related. Such a commitment would require the *intension* of $\mathscr{R}_i$ to be specified. As detailed in Chapter 4, the intensions of various $\mathscr{R}_i$ are implicitly defined within SMRF in terms of *relational functions* and *acceptance sets*. It is useful to assign truth values to n-tuples corresponding to whether or not they are members of the various relations $\mathscr{R}_i$. This assignment is performed by *relational predicates*.[2] Let $\mathcal{R}$ be the set of relational predicates defined over the objects in $\mathcal{O}$. $\mathcal{R}$ is a set of the form $\{R_1, R_2, \ldots, R_{|\mathcal{R}|}\}$. Each R_i is a truth-valued function[3] of the form $R_i : \mathcal{O}^n \rightarrow \{0, 1\}$, and is defined as:

$$R_i(o_1, o_2, \ldots, o_n) = \begin{cases} 1 & \text{if } (o_1, o_2, \ldots, o_n) \in \mathscr{R}_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

An example of a unary relational predicate is "object A is red", which is true just in case that a particular object A has the color "red" as defined by some observer. An example of a binary relational predicate is "objects A and B are far apart", which is true just in case that two particular objects A and B are located "far apart" in a position space, as defined by some observer. In this context, relations are defined in terms of spatiotemporal quantities, so "red" might consist of a region in RGB space, defined over an object's color attribute. The relational predicate "object A is red" would map all objects whose color attributes fall within the particular region to 1, and the rest to 0.

Using $\mathcal{O}$, one can define the notion of an *instance*. An instance I is an n-tuple of the form $(o_1, o_2, \ldots, o_n)$, where each o_i is a member of $\mathcal{O}$, and where each o_i is unique in I. Let $\textsc{Perm}(S, n)$ denote a function that maps a set S of objects to the set of all

[2]In many contexts, what is here defined as a *relational predicate* is termed a *relation*. Depending upon the context, the term *relation* can either refer to a set of tuples, or an assignment of truth values to tuples. In the interest of rigor, the term has been defined according to the former definition. However, as both terms are so closely related, they are interchangeable for most practical purposes.

[3]In this context, truth is represented by the numeric value '1' and falsity by '0'.

possible instances of objects in S of a particular size n. PERM is formally defined as follows:

$$\textsc{Perm}(S,n) = \{(o_1, o_2, \ldots, o_n) | \bigwedge_{i=1}^{n} \Big(o_i \in S \wedge \forall j [(j \in [1,n] \wedge i \neq j) \Rightarrow o_i \neq o_j] \Big) \}. \quad (3.2)$$

In essence, PERM produces all n-permutations of the elements of S. With PERM, one can define another function $\mathfrak{F}(S)$, which maps a set S of objects to all possible instances of objects in S. $\mathfrak{F}$ is defined as follows:

$$\mathfrak{F}(S) = \bigcup_{k=1}^{|S|} \textsc{Perm}(S,k). \quad (3.3)$$

With $\mathfrak{F}$, one can define an *instance space*, Υ, where $\Upsilon = \mathfrak{F}(\mathcal{O})$. This allows one to define a target concept. A *target concept*, in this context, is simply a binary function over an instance space. A target concept $c(I)$ is a function of the form $c : \Upsilon \rightarrow \{0,1\}$. If $c(I)$ maps an instance I to 1, then I is called an *instance of the target concept* c, or a *target conceptual instance* of c. Alternatively, I *matches*, *exemplifies*, or *satisfies* the target concept c. In concept learning, learning agents are given data in the form of $\langle I, c(I) \rangle$ pairs (Mitchell, 1997). This corresponds to providing the learning agent with some number of labeled instances, and asking it to learn a suitable representation of the target concept. However, the SMRF approach is designed to solve problems in which it is not known which objects play the key role. Given a set of objects, there are one or more specific instances that exemplify the target concept. However, for the problems that SMRF is designed to solve, such labeled instances are not available *a priori*. Rather, the sets of objects from which such instances are defined are the only data that are available. The sets of objects are called *groups* or *graphs*. Thus, for each graph, there are multiple instances, only one of which might satisfy the target concept. This existential property makes the SMRF learning problem a Multiple Instance Learning (MIL) problem (Maron and Lozano-Pérez, 1998) (see Section 2.2)

in terms of the instance space of a particular set of graphs.

A graph G is a set of objects,[4] such that $G \subseteq \mathcal{O}$. Let the set of all graphs comprising a dataset be denoted Γ. Let Υ_G denote a *graph instance space*, and be defined as $\mathfrak{J}(G)$. From this, it follows that $\Upsilon_G \subseteq \Upsilon$. Given a graph G, let B_G denote the *bag* corresponding to that graph, where $B_G = \Upsilon_G$. While a graph is a set containing some number of objects, the bag corresponding to that graph is a set containing all k-permutations of the objects in the graph, where k ranges from 1 to the number of objects in the graph. All elements of a bag are instances, whereas the objects in a graph are not instances. This notion of bag allows one to denote different combinations of objects as exemplifying the target concept. For instance, if a target concept is defined as "a red object next to a green object," and if a particular graph had two such pairs of objects, then the bag corresponding to that graph would contain two target-conceptual instances.

Given this notion of bag, the definition of the target concept can be extended to cover bags as well as instances. This allows one to denote whether or not a given bag contains target-conceptual instances. Let the target concept defined over bags be denoted $C(B)$, which is a function of the form $C : \wp(\Upsilon) \rightarrow \{0, 1\}$. $C(B_G)$ is defined as:

$$C(B_G) = \max_{I \in \Upsilon_G} c(I). \tag{3.4}$$

That is, $C(B_G) = 1$ if and only if there are one or more instances I of the target concept c in B_G. If $C(B_G) = 1$ for some bag B_G, then B_G is called a *positive bag*. If $C(B_G) = 0$ for some bag B_G, then B_G is called a *negative bag*. Let the set of positive

[4]The use of the term *graph* in this context diverges somewhat from the standard usage, which denotes a tuple (V, E), containing a set of vertices and a set of edges. In the context of SMRF, the set of objects can be thought of as V. Moreover, given that the objects are attributed and the graph is labeled, E can be thought of as being implicitly defined by the relations that participate in the definition of the target concept.

bags be denoted B^+ and the set of negative bags be denoted B^-. Furthermore, let the set of graphs corresponding to the positive bags be denoted G^+, and the set of graphs corresponding to the negative bags G^-, such that $G^+ \cup G^- = \Gamma$.

However, since each graph has a one-to-one correspondence to a bag, it makes sense to define target concepts in terms of the graph that corresponds to a bag, in addition to the bag itself. In a double usage of notation, let C be also defined over graphs, so that $C : \wp(\mathcal{O}) \rightarrow \{0, 1\}$, where C is defined in this case as $C(G) = C(B_G)$, where B_G is the bag corresponding to G. If $C(B_G) = 1$, then it is said that the graph G *contains* the target concept C.

The learning problem that the SMRF approach attempts to solve is as follows. As a form of concept learning, the algorithm is given a dataset that consists of $\langle graph, label \rangle$ pairs. In this context, the label is the evaluation of the target concept function C on the graph *graph*. Intuitively, the goal of the learning algorithm is to learn a function that approximates C as closely as possible, given the training data. Formally, given a particular dataset $\mathcal{D}$, where $\mathcal{D}$ can be formally denoted as $\{\langle G_1, C(G_1)\rangle, \langle G_2, C(G_2)\rangle, \ldots, \langle G_j, C(G_j)\rangle, \ldots\}$, the learning algorithm attempts to find a hypothesis H, where $H : \wp(\mathcal{O}) \rightarrow [0, 1]$, such that for every $\langle G_j, C(G_j)\rangle \in \mathcal{D}$, $H(G_j) = C(G_j)$. The degree to which H approximates C is the degree to which the algorithm correctly learns the target concept.

It is useful to be able to evaluate H with respect to the bag that is associated with a given graph. Since each bag maintains a one-to-one correspondence with a given graph, it makes sense to define H with respect to both bags and graphs, as was done with C above. By a similar double usage of notation, H can be defined with respect to bags, such that $H(B)$ is of the form $H : \wp(\Upsilon) \rightarrow [0, 1]$. $H(B_G)$ is defined as:

$$H(B_G) = \max_{I \in \Upsilon_G} h(I). \tag{3.5}$$

Just as the target concept c was defined with respect to instances, and was extended to be defined with respect to bags, so also can the hypothesis H, which is defined with respect to bags, also be contracted to be defined with respect to instances. In Equation 3.5 above, h represents this contraction from bags to instances. The function h is a learned approximation of c, and is a function of the form $h : \Upsilon \to [0, 1]$. Since $B_G = \mathfrak{S}(G)$, it follows that $H(G)$ is defined as

$$H(G) = \max_{I \in \mathfrak{S}(G)} h(I). \tag{3.6}$$

That is, that the value of H for a given graph G is the value of h evaluated on the instance in the bag corresponding to G that maximizes h.

For a given dataset $\mathcal{D}$, let the set of graphs contained by the dataset be denoted $\Gamma_{\mathcal{D}}$ (or just Γ, if $\mathcal{D}$ is implicit from context), and let the sets of positive and negative graphs be denoted $G^+_{\mathcal{D}}$ and $G^-_{\mathcal{D}}$, respectively (or just G^+ and G^-, respectively, if $\mathcal{D}$ is implicit from context).

It should be noted that while the codomain of C is $\{0, 1\}$, the codomain of H is $[0, 1]$. In this context, this means that the SMRF approach attempts to perform probabilistic classification on graphs – that is, that:

$$H(G) = \Pr\left(C(G)\right).^5 \tag{3.7}$$

In the case of perfect classification,

$$\Pr\left(C(G)\right) = 0 \quad \text{if} \quad C(G) = 0$$

and

$$\Pr\left(C(G)\right) = 1 \quad \text{if} \quad C(G) = 1.$$

[5] More explicitly, this is $\Pr\left(C(G) = 1\right)$. In this paper, we use $\Pr\left(\bullet\right)$ to denote a probability (Kolmogorov, 1956), and $p(\bullet)$ to denote a density.

Hence, $H(G) = C(G)$ in the case of perfect classification. The more that classification deviates from the ideal, $H(G)$ will deviate from $C(G)$ by a greater amount.

The primary goal of the learning algorithm is to find an H that will perfectly classify a dataset.[6] To measure how well H classifies a particular dataset, a notion of perfect classification is needed. It follows from the definition of C that for a particular dataset $\mathcal{D}$,

$$\left(\bigwedge_{G \in G^+_{\mathcal{D}}} C(G) \right) \wedge \neg \left(\bigvee_{G \in G^-_{\mathcal{D}}} C(G) \right). \tag{3.8}$$

The target concept C assigns a value of 1 to each positive graph, and assigns a value of 0 to each negative graph. It is thus a property of the target concept that labels of the positive graphs satisfy an AND relation, and that the labels of the negative graphs satisfy a NOR relation. As such, the expression in Equation 3.8 denotes a property that is true of all target concepts.

Let $\mathfrak{C}_{\mathcal{D}}(f)$ represent a predicate that denotes whether the property defined in Equation 3.8 is true of some arbitrary function f, whose range is $\{0, 1\}$. Thus, $\mathfrak{C}_{\mathcal{D}}$ defines a necessary condition of target-conceptuality, with respect to $\mathcal{D}$, that can be applied to any binary-valued function. $\mathfrak{C}_{\mathcal{D}}(f)$ is defined as:

$$\mathfrak{C}_{\mathcal{D}}(f) \equiv \left(\bigwedge_{G \in G^+_{\mathcal{D}}} f(G) \right) \wedge \neg \left(\bigvee_{G \in G^-_{\mathcal{D}}} f(G) \right). \tag{3.9}$$

Of all possible hypotheses H, there is one hypothesis, denoted H_C, which perfectly mirrors the target concept C. $H_C(G)$ is defined as:

[6] There are other algorithmic desiderata, such as run-time efficiency and the simplicity of learned hypotheses. In this context, however, only classification accuracy is considered, as it is the only desideratum relevant in this context to the derivation of the scoring function L (Eq. 3.11).

$$H_C(G) = \begin{cases} 1 & \text{if } C(G) = 1, \\ 0 & \text{if } C(G) = 0. \end{cases} \tag{3.10}$$

It follows that $\mathfrak{C}_{\mathcal{D}}(H_C) = 1$. Thus, when applied to hypotheses, $\mathfrak{C}_{\mathcal{D}}$ defines the criteria of perfect classification. $\mathfrak{C}_{\mathcal{D}}(H)$ indicates whether or not the values of H for the positive graphs satisfy the AND relation and the values of H for the negative graphs satisfy the NOR relation.

However, while a particular hypothesis may have a range of $\{0, 1\}$, the codomain of all hypotheses is $[0, 1]$. In light of this, $\mathfrak{C}_{\mathcal{D}}$ can be converted from a predicate that defines perfect classification, to a real-valued function that defines the degree of successful classification. This conversion can be accomplished by converting the AND and NOR relations into Noisy-AND and Noisy-NOR operations, respectively. Let the result of this conversion be denoted $L(\mathcal{D}|H)$, and defined as follows:

$$L(\mathcal{D}|H) = \left(\prod_{G \in G^+_{\mathcal{D}}} H(G) \right) \times \left(\prod_{G \in G^-_{\mathcal{D}}} (1 - H(G)) \right). \tag{3.11}$$

L approaches one as H classifies positive graphs with high probability and negative graphs with low probability. Conversely, L approaches zero as H either classifies positive graphs with low probability, or negative graphs with high probability. Thus, $L(\mathcal{D}|H)$ can be interpreted as normalized measure of the classification quality of H over $\mathcal{D}$, as well as the likelihood of $\mathcal{D}$, given H.

With these concepts in mind, the SMRF learning problem can be formally stated, as follows: given some dataset $\mathcal{D}$, learn a hypothesis H that maximizes $L(\mathcal{D}|H)$.[7] The representation of learned hypotheses by SMRF trees is discussed in Chapter 4. The algorithm used to solve this problem is given in Chapter 5.

[7]The SMRF learning algorithm is designed to produce hypotheses of minimal complexity. Such a constraint, however, is not strictly required to solve problems of this kind.

Chapter 4

SMRF Trees

In this chapter, I describe the SMRF framework in a series of steps. First, in Section 4.1, I describe the various elements of a SMRF tree and describe the process of *instantiation*, whereby SMRF builds up a relevant subset of the graph instance space $\mathfrak{S}(G)$ for hypothesis evaluation. Next, in Section 4.2, I describe tree sorting mechanism in detail, in terms of the framework developed in Chapter 3. In Section 4.3, I take a short aside to show how concepts defined in terms of categorical variables can be easily represented by SMRF trees. Last, in Section 4.4, I relate the SMRF framework to the hypothesis framework developed in Chapter 3, in preparation for the discussion of the learning algorithm in Chapter 5.

4.1 Instantiation

SMRF trees are a means of representing a hypothesis of a target concept (Ch. 3). Specifically, a SMRF tree is an augmented decision tree that performs probabilistic classification upon a given group of objects. That is, given a group G of objects drawn from $\mathcal{O}$, a SMRF tree T defines a function $T : \wp(\mathcal{O}) \rightarrow [0, 1]$, where $T(G)$ represents the probability that G contains the target concept. In this way, SMRF trees are a means of representing a target concept hypothesis H (cf., Equation 3.10).

However, SMRF trees do more than simply define a hypothesis. They also provide a means of evaluating potentially-intractable hypotheses in a tractable manner. For groups of any substantial size, the group instance space over which a target concept is defined is prohibitively large. Since the instance space includes all k-permutations

of the objects in the group, its size is:

$$|\mathfrak{F}(G)| = \sum_{k=1}^{|G|} \frac{|G|!}{(|G| - k)!}. \tag{4.1}$$

The instance space is factorial in the number of objects present in a group. Thus, it is intractable to explicitly enumerate the instance space for all but simple problems. However, the definition of a target concept hypothesis specifies that the value of the maximal instance be returned as the value of a particular group (Equation 3.10). The SMRF approach is designed to locate representative instances from positive bags, while avoiding an intractable iteration over the instance space. It does this by pruning off subsets of the instance space that do not contain maximal or near-maximal instances, and evaluating some smaller subset of instances that might possibly be maximal. Ideally, the size of this subset is linear, instead of factorial, in the size of the input group.

SMRF trees accomplish this instance space pruning through the process of incremental *instantiation*. In a straightforward implementation of the theoretical problem formulation (Chapter 3), the group instance space is derived by evaluating $\mathfrak{F}(G)$, which entails an enumeration of all possible instances. A SMRF tree, on the other hand, builds up a set of instances incrementally, pruning off subsets of the instance space that do not contain maximal instances. Given an ordered sequence of unique objects drawn from a group G, an *instantiation* is the appending of a new object from G, not already in the sequence, to the end of the sequence. The process of instantiating a single object occurs at *instantiation nodes* in the SMRF tree.

The process of instantiation and group evaluation is best illustrated by example. Figure 4.1(a) shows a hand-crafted, example SMRF tree that is designed to recognize "yellow" objects. In this figure, an instantiation node is represented by a parallelogram. The sequence of instantiated objects is called an *instantiation sequence*. Ab-

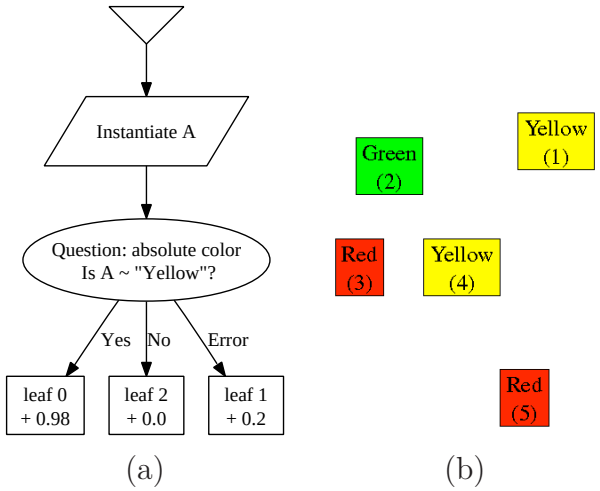


Figure 4.1: A hand-crafted SMRF tree (a) that identifies yellow objects. Although I describe the decision tree split with a categorical label ("yellow"), the concept is represented as a volume in RGB space. (b) A set of objects scattered on a plane.

stractly, an instantiation sequence I is a sequence of objects $(o_1, o_2, \ldots, o_i, \ldots, o_{|I|})$, where each object o_i corresponds to the object instantiated at the i^{th} instantiation node in a root-to-leaf path in the SMRF tree.[1] In this approach, instantiation sequences are distinguished according to the group from which they are drawn, and thus the i^{th} instantiation sequence drawn from a group G is denoted I_i^G. Every instantiation sequence in a tree corresponds to an instance in the group instance space, but due to the SMRF tree's pruning of the instance space, the converse is often not the case. The set of all instantiation sequences present in a tree from a group G is denoted $\mathcal{I}^G$, and is called an *instantiation sequence collection*. $\mathcal{I}^G$ is to be distinguished from $\mathfrak{S}(G)$, the instance space of G. $\mathcal{I}^G \subseteq \mathfrak{S}(G)$, and, ideally, $|\mathcal{I}^G| \ll |\mathfrak{S}(G)|$. In addition, the set of all instantiation sequences present in a tree from a set of groups Γ is denoted $\mathcal{I}^\Gamma$.

[1] The notion of an instantiation sequence is structurally equivalent to the notion of an instance in Chapter 3. However, the term *instance* is used to denote a possible ordering of objects that exists *in abstracto*, by virtue of the definition of the data set. In contrast, the term *instantiation sequence* is used to denote a sequence of objects that exists *in actu* (as much as any quantity produced by an algorithm can be said to be actual, as opposed to purely abstract), being generated by instantiation nodes in time.

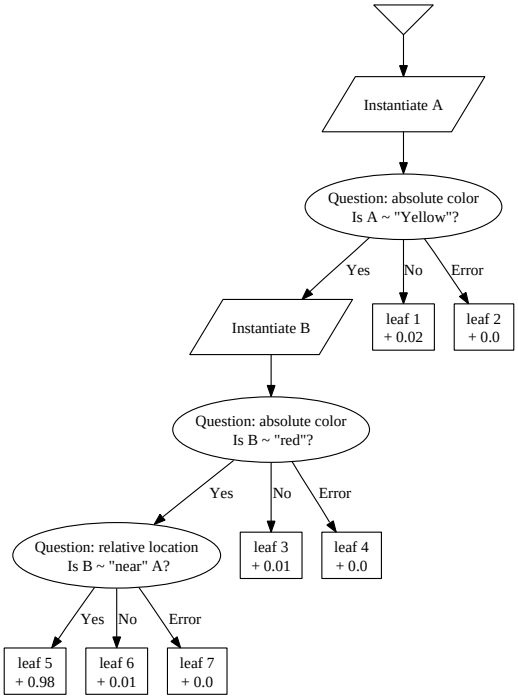


Figure 4.2: A tree that identifies yellow objects that are near red objects.

Figure 4.2 shows an example SMRF tree that identifies groups containing a "yellow" object next to a "red" object. Figure 4.1(b) shows an example group that this tree can probabilistically classify. The process of probabilistically classifying a group as to whether or not it contains a target concept is called a *query*.

The query process begins at the *root node*, represented by an inverse triangle. The instantiation sequence collection at this point contains only an empty sequence, and is represented as $\{()\}$, where "()" is the empty sequence. Next, the process proceeds to the first instantiation node (labeled "Instantiate A"), at which the first variable (denoted "A") is instantiated. Since there are five objects in the group, five instantiation sequences are created, each containing one object. At this point, the set of instantiation sequences is $\{(1), (2), (3), (4), (5)\}$. Next, the process proceeds to the first *question node*, denoted by an oval. This node queries each instantiation sequence, such that those that satisfy the criteria of the question node are sorted down the *yes branch*, those that don't are sorted down the *no branch*, and any questions that are

asked of attributes that a particular object does not possess are sorted down the *error branch*. In this example, the question node asks whether or not the first object in the instantiation sequence is "yellow." Objects 2, 3, and 5 are not "yellow," and are thus sorted into the no branch. Thus, leaf 1 contains $\{(2), (3), (5)\}$. The remaining two instantiation sequences ($\{(1), (4)\}$) is sorted down the yes branch because objects 1 and 4 are "yellow". No objects are sorted down the error branch, since all objects have a color attribute.

Next, the query procedure comes to the other instantiation node (labeled "Instantiate B"), at which a second variable (denoted "B") is instantiated in the sequences. The set of instantiation sequences coming into this node is $\{(1), (4)\}$. Since the process of instantiation creates all possible instantiation sequences, the resulting set of sequences is $\{(1,2), (1,3), (1,4), (1,5), (4,1), (4,2), (4,3), (4,5)\}$. Note that sequences such as $(1,1)$ and $(4,4)$ are not possible, since those sequences map the same object to multiple variables. Next, the query comes to another question node, which asks if the second object in the instantiation sequence is "red". The set of instantiation sequences whose second object is not "red" is $\{(1,2), (1,4), (4,1), (4,2)\}$, and thus these are sorted down the no branch, and into leaf 3. The remaining set $\{(1,3), (1,5), (4,3), (4,5)\}$ are sorted into the yes branch. Next, the query comes to the final question node, which asks if the two objects in the instantiation sequence are "near" to one another in Euclidean space. Only objects 3 and 4 meet this criterion, so $\{(3,4)\}$ is sorted into leaf 5, and $\{(1,5), (4,3), (4,5)\}$ is sorted into leaf 6. The query procedure returns the max probability over the leaves into which instantiation sequences from a group are sorted. The leaf node with the highest probability into which an instantiation sequence falls is leaf 5, with a probability of 0.98. Thus, the tree classifies the group to contain the target concept with a 98% probability. This example introduces the significant features of the SMRF approach – namely, that various target concepts can be defined in a tree-like manner, with only ordered sequences

of objects being considered that have proven relevant at higher levels of the tree. This gives the SMRF tree a good deal of power in determining if a group encodes a target concept, while allowing tractability to be preserved through the pruning of irrelevant instantiation sequences.

Formally, the i^{th} instantiation sequence from group G at edge e in the tree is denoted ${}^{e}I_i^G$. The set of instantiation sequences from group G at an edge e in the tree is denoted by ${}^{e}\mathcal{I}^G$, and the set of instantiation sequences from all groups at an edge e in the tree is denoted by ${}^{e}\mathcal{I}$. For the sake of convenience, sets of instantiation sequences arriving at a node from its parent edge are indexed by the index of the node itself. For instance, the set of instantiation sequences arriving at leaf node k is denoted ${}^{k}\mathcal{I}$.

For every instantiation node in the tree, there is a set of instantiation sequences entering the node from its parent edge, and a set leaving via its child edge. The set entering via the parent edge is denoted ${}^{p}\mathcal{I}$, and the set exiting via the child edge is denoted ${}^{c}\mathcal{I}$. The process of instantiation is formally described by the following expression which defines ${}^{c}\mathcal{I}$ in terms of ${}^{p}\mathcal{I}$:

$$ {}^{c}\mathcal{I}^G = \bigcup_{I \in {}^{p}\mathcal{I}^G} \bigcup_{o \in (G - \{I\})} \textsc{Append}(I, o), $$

where for any arbitrary sequence $S = (s_1, \ldots, s_{|S|})$, $\textsc{Append}(S,o) = (s_1, \ldots, s_{|S|}, o)$. The recursive definition of instantiation finds a base condition in the *root node* of the tree. The set of instantiation sequences from a group G at the root node, denoted ${}^{0}\mathcal{I}^G$, is a set containing a single empty sequence, such that ${}^{0}\mathcal{I}^G = \{()\}$.

4.2 Sorting

A question node evaluates a given instantiation sequence based upon some criterion, and then sorts it down the node's yes branch if it meets the criterion, or down the

no branch if it fails to meet the criterion. There is also a possible case that can arise if non-homogeneous groups are used, where the question node criterion makes use of an attribute that a particular object in the instantiation sequence doesn't possess. In such a case, the instantiation sequence cannot be properly evaluated according to the criterion, and it is sorted down the error branch. The error branch is considered a normal branch of the tree, just like the yes and no branches.

In general, the criterion that a question node q uses to sort instantiation sequences is whether or not a particular sequence contains a specific combination of objects that satisfy a relational predicate $R \in \mathcal{R}$ (Chapter 3). That is, the criterion that a question node uses is whether or not R would map a particular combination of objects in an instantiation sequence to 1 or 0. Those that R would map to 1 are sorted down the yes branch, those that R would map to 0 are sorted down the no branch, and those for which R is not defined are sorted down the error branch.

For example, a certain relation might be of the form "object A is near object B." A particular question node q might use this relation as its criterion for sorting instantiation sequences. However, this is too imprecise. To which objects do "A" and "B" correspond? After all, an instantiation sequence might contain more than two objects, thus producing a number of possible mappings between the objects in the sequence, and the variables "A" and "B." Furthermore, how near is "near"? More than that, how is "near" even defined? Is it defined over a Euclidean space? If so, which distance metric does it refer to? Euclidean distance? Manhattan distance? A more formal specification of relations is required.

A *model* fully specifies the criterion used by a question node to sort instantiation sequences, thus removing the above ambiguities. A particular model at a question node q is denoted M_q, and is a function of the form $M_q : \mathcal{I}^\Gamma \rightarrow \{1, 0, \mathbf{E}\}$, where $\mathcal{I}^\Gamma$ is the set of possible instantiation sequences of a set of groups Γ, and $\mathbf{E}$ denotes an error condition. A model has two components: a relational predicate R_q (where

$R_q \in \mathcal{R}$), and a specification that maps particular objects in an instantiation sequence to the particular parameters of R_q, called a *parameter mapping*. The necessity of the parameter mapping can be seen in that a given R_q is an n-ary predicate, but a given instantiation sequence may be of length m, where $m \neq n$. If $m < n$, then R_q cannot be applied directly to I. If $m > n$, then R_q cannot be applied to I itself – some subsequence of I must be used instead. Even if $m = n$, it may be desirable that R_q be applied to some permutation of I. For these reasons, it is necessary to have a mechanism that maps elements in certain positions of I to certain inputs of R_q, and the parameter mapping serves just such a function. For example, a parameter mapping would choose particular objects out of a given instantiation sequence to map to the variables "A" and "B" in the example above. A parameter mapping is formally denoted χ_q, and is a function of the form $\chi_q : \mathcal{O}^m \rightarrow \mathcal{O}^n$. If an instantiation sequence I is of length m, then $\chi_q(I)$ is an instantiation sequence of length n.

It is also desirable to have different parameter mappings defined for different models that make use of the same R. The parameter mapping can itself be defined in terms of a parameter that determines which elements in I get mapped to which inputs of R. This parameter defines the parameter mapping χ, and is called a *mapping template*. A mapping template π_χ is a function of the form $\pi_\chi : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$, and is used to define χ in the following way:

$$\chi(I) = \Big(I[\pi_\chi(k)]\Big)_{k \in [1, |I|]_\mathbb{Z}}. \tag{4.2}$$

The "sequence builder" notation $\Big(\varphi(k)\Big)_{k \in \Sigma}$ specifies a sequence S where, for all $k \in \Sigma$ and some function φ defined over the positive integers, each k-th element [2] of the sequence S is equal to $\varphi(k)$. The notation $S[k]$ denotes the k-th element of a sequence S. In addition, the notation $[a, b]_\mathbb{Z}$ denotes an integer-valued interval

[2] This representation assumes the first element of a sequence has an index of 1.

between endpoints a and b, inclusive. The following example shows how the parameter mapping functions: if $\pi_\chi(1) = 3$ and $\pi_\chi(2) = 1$, and if $I = (o_1, o_2, o_3)$, then $\chi(I) = (o_3, o_1)$.

Given R_q and χ_q, $M_q(I)$ is defined as follows:

$$M_q(I) = \begin{cases} 1 & \text{if } \neg\varepsilon_q(I) \wedge R_q(\chi_q(I)), \\ 0 & \text{if } \neg\varepsilon_q(I) \wedge \neg R_q(\chi_q(I)), \\ \mathbf{E} & \text{if } \varepsilon_q(I). \end{cases} \tag{4.3}$$

In the above equation, $\varepsilon_q(I)$ is a function of the form $\varepsilon_q : \mathcal{I}^\Gamma \rightarrow \{1, 0\}$ that encodes the error conditions over which the model cannot be properly evaluated for a given I. If an error condition exists, then $\varepsilon_q(I) = 1$. $\varepsilon_q(I) = 0$ otherwise. Also, in the case that I denotes a sequence $(o_1, o_2, \ldots, o_n)$, the above notation presumes that expressions such as $R_q(I)$ and $R_q(o_1, o_2, \ldots, o_n)$ are equivalent.

R_q is an n-ary truth-valued function of the form $R_q : \mathcal{O}^n \rightarrow \{1, 0\}$. However, in order to define the intension of the relation $\mathscr{R}_q$ over which the relational predicate R_q is defined (Chapter 3; Equation 3.1), $\mathscr{R}_q$ can be decomposed into two further components: a *relational function* ρ_q, and an *acceptance set* Δ_q. A relational function ρ_q maps an n-tuple of objects to a value in some metric space $\mathfrak{S}_q$, and is of the form $\rho_q : \mathcal{O}^n \rightarrow \mathfrak{S}_q$. The acceptance set Δ_q specifies some "acceptable" region in $\mathfrak{S}_q$. Hence, $\Delta_q \subseteq \mathfrak{S}_q$. Thus, the SMRF approach intensionally defines each n-ary relation $\mathscr{R}_q$ (where $\mathscr{R}_q \in \mathfrak{R}$), as follows:

$$\mathscr{R}_q = \{(o_1, o_2, \ldots, o_n) \,|\, \rho_q(o_1, o_2, \ldots, o_n) \in \Delta_q\} \tag{4.4}$$

From Equations 3.1 and 4.4, R_q can be defined as follows:

$$R_q(o_1, o_2, \ldots, o_n) = \begin{cases} 1 & \text{if } \rho_q(o_1, o_2, \ldots, o_n) \in \Delta_q, \\ 0 & \text{otherwise.} \end{cases} \tag{4.5}$$

Thus, the manner in which the question nodes of the SMRF tree sort a given instantiation sequence I is as follows. If $M_q(I) = 1$, then I is sorted down the yes branch of q. If $M_q(I) = 0$, then I is sorted down the no branch of q. If $M_q(I) = \mathbf{E}$, then I is sorted down the error branch of q.

The relational function ρ_q can be decomposed into two further components: a *model function* f_q, and an *attribute mapping*, that maps particular objects in the parameters of ρ_q to certain attributes defined over those objects. A relational function ρ_q maps some sequence of objects to a value in some metric space. However, it is not immediately clear how a given relational function should be defined. The SMRF approach asks questions in terms of the attributes of the objects present in a group, but this implies that certain object attributes must be specified, and that some function be applied to them. For instance, one question might concern the distance between the locations of two objects. In such a case, the relational function would need to specify the "location" attributes of two objects, and compute their difference. The operation of specifying attributes is independent of the function that is applied to those attributes, so it makes sense to decouple them in the definition of ρ_q. Hence, the differentiation between the attribute mapping, which specifies the object attributes of interest, and the model function, which performs some particular operation of interest (such as subtraction) on its inputs.

An attribute mapping is formally denoted $\mathfrak{A}$, and is a function of the form $\mathfrak{A} : \mathcal{O}^n \rightarrow \mathfrak{S}_{\mathcal{A}}^n$. Recall from Chapter 3 that $\mathcal{A}$ is the set of all attributes A_i, and that $\mathfrak{S}_{\mathcal{A}}$ is union of all of the the codomains of the attributes $A_i \in \mathcal{A}$. If X is a sequence of objects in $\mathcal{O}$ of length n, then $\mathfrak{A}(X)$ is a sequence of values in $\mathfrak{S}_{\mathcal{A}}$ of length n, where each element in $\mathfrak{A}(X)$ is a particular attribute of a particular element in X.

$\mathfrak{A}$ is defined in terms of an *attribute template* $\pi_{\mathfrak{A}}$, which is a function of the form $\pi_{\mathfrak{A}} : \mathbb{Z}^+ \to \mathcal{A}$, and is used to define $\mathfrak{A}$ in the following way:

$$\mathfrak{A}(X) = \Big(\pi_{\mathfrak{A}}(k)(X[k])\Big)_{k \in [1,|X|]_{\mathbb{Z}}}. \tag{4.6}$$

Alternatively, using dot notation,

$$\mathfrak{A}(X) = \Big(X[k].\pi_{\mathfrak{A}}(k)\Big)_{k \in [1,|X|]_{\mathbb{Z}}}. \tag{4.7}$$

For example, if $\pi_{\mathfrak{A}}(1) = A_i$ and $\pi_{\mathfrak{A}}(2) = A_j$, and if $X = (o_3, o_1)$, then $\mathfrak{A}(X) = (A_i(o_3), A_j(o_1))$. Alternatively, using dot notation, $\mathfrak{A}(X) = (o_3.A_i, o_1.A_j)$ in this example.

The model function f_q is a function of the form $f_q : \mathfrak{S}_{\mathcal{A}}^n \to \mathfrak{S}_q$. That is, f_q maps a sequence of attribute values to a point in the metric space of R_q. Some commonly-used model functions in the SMRF framework are:

- Identity: $f_q(\alpha) = \alpha[1]$,
- Difference: $f_q(\alpha) = \alpha[2] - \alpha[1]$, and
- Distance: $f_q(\alpha) = \sqrt{(\alpha[2] - \alpha[1])^T(\alpha[2] - \alpha[1])}$,

where α is a sequence of attribute values. "Identity" simply denotes the first (and generally only) attribute value in α. This is useful for asking questions about the attributes of objects, such as "Is the color of A is red?" "Difference" and "Distance" simply denote the difference and distance, respectively, of the first (and generally only) two attribute values in α. This is useful for asking questions about binary relations of attributes of objects, such as "Is A is near B?"

In addition, ternary model functions are also available that return the position of the third parameter in terms of a two-dimensional coordinate frame defined by

the first two parameters. For example, in the case of two-dimensional position, $\alpha[1]$ serves as the origin of the new coordinate frame, and the new x-axis is defined as the difference vector $\alpha[2] - \alpha[1]$. The new y-axis is then simply the vector orthogonal to the new x-axis. These axes are used to define a coordinate frame transformation matrix, which is then used to determine the position of $\alpha[3]$ in the new coordinate frame.

Several such functions are available. In addition to the function just described, there is another function which normalizes all values to the unit square. Additionally, higher-order procedures exist for defining coordinate frames in higher dimensions (where $k-1$ objects are required to define a coordinate frame in k dimensions). These functions are called *Reframe* functions, as they define a quantity in a new coordinate frame. These functions provide SMRF with the ability to represent k-ary relations between objects.[3]

Putting everything together, the relational function ρ_q is defined as follows:

$$\rho_q(o_1, o_2, \ldots, o_n) = f_q(\mathfrak{A}_q(o_1, o_2, \ldots, o_n)). \tag{4.8}$$

From Equations 4.3 and 4.8, it follows that the full formal theoretical definition of $M_q(I)$ is as follows:

$$M_q(I) = \begin{cases} 1 & \text{if } \neg\varepsilon_q(I) \wedge [\rho_q(\chi_q(I)) \in \Delta_q], \\ 0 & \text{if } \neg\varepsilon_q(I) \wedge [\rho_q(\chi_q(I)) \notin \Delta_q], \\ \mathbf{E} & \text{if } \varepsilon_q(I). \end{cases} \tag{4.9}$$

As defined above, $\mathfrak{S}_q$ is a metric space, and in practice, is often a Euclidean space $\mathbb{R}^n$. Since $\mathfrak{S}_q$ is often infinite in practice (given that SMRF relations are defined over continuous multidimensional values), a pdf-likelihood threshold pair is used to define

[3]In practice, however, only ternary relations have been used thus far in experiments.

Δ_q. The pdf is of the form $p(x|\theta_q)$, where θ_q are the distribution parameters. The threshold Θ_q defines a region in the likelihood space of p, such that:

$$(p(x|\theta_q) > \Theta_q) \text{ iff } (x \in \Delta_q). \tag{4.10}$$

In addition, it is convenient to refer to the combination of the relational function ρ_q and the parameter mapping χ_q. This allows one, for instance, to easily distinguish between "A is near B" and "A is near C," since both share the same basic relational form "(Object) near (Object)." This combination of the relational function and the parameter mapping is called a *mapping function*, and is denoted ϕ_q. A mapping function ϕ_q is a function of the form $\phi_q : \mathcal{I}^\Gamma \rightarrow \mathfrak{S}_q$, and is defined as:

$$\phi_q(I) = \rho_q(\chi_q(I)). \tag{4.11}$$

For example, if $I = (o_1, o_2, o_3)$, an arbitrary mapping function ϕ_q might map I to $(o_3.A_i, o_1.A_j)$. Putting Equations 4.9, 4.10, and 4.11 together, the full definition of $M_q(I)$, as utilized by SMRF, is as follows:

$$M_q(I) = \begin{cases} 1 & \text{if } \neg\varepsilon_q(I) \wedge [p(\phi_q(I)|\theta_q) > \Theta_q], \\ 0 & \text{if } \neg\varepsilon_q(I) \wedge [p(\phi_q(I)|\theta_q) \leq \Theta_q], \\ \mathbf{E} & \text{if } \varepsilon_q(I). \end{cases} \tag{4.12}$$

In summary, the components of a model M_q, as utilized by the SMRF approach, are as follows:

- An *error predicate* ε_q,
- A pdf $p(\bullet|\theta_q)$, which is currently one of the following distributions:
  - A multivariate Gaussian distribution (for color and location attributes),

  - A Von Mises distribution (for 2D orientation),
  - A Dimroth-Watson distribution (for 3D orientation),
  - A Girdle distribution (for 3D orientation of objects that display rotational symmetry),

- Distribution parameters θ_q,
- A *decision threshold* Θ_q,
- A mapping function ϕ_q, which is defined in terms of:
  - A parameter mapping χ_q, which is defined in terms of:
    - A mapping template π_χ,
  - A relational function ρ_q, which is defined in terms of:
    - An attribute mapping $\mathfrak{A}_q$, which is defined in terms of:
      - An attribute template $\pi_{\mathfrak{A}}$,
    - A model function f_q.

In the learning process (Chapter 5), each mapping function ϕ_q is associated with a particular pdf p, and θ_q and Θ_q are computed during learning. However, π_χ, $\pi_{\mathfrak{A}}$, and f_q can be changed independently to some degree, allowing one to define a large number of relations from a relatively small number of parameters. For instance, varying π_χ, one can define relations in terms of the color of the first object in the instantiation sequence, the color of the second object, the color of the third object, and so on. Varying $\pi_{\mathfrak{A}}$, one can define relations in terms of the color of the first object in the instantiation sequence, the location of the first object, the orientation of the first object, and so on. Varying f_q, one can define relations in terms of the difference vector between the location of the first and second objects in the instantiation sequence, the distance between those locations, and so on. Thus, from a small number of primitive

components, a large number of relations are can be defined. Moreover, once π_χ, $\pi_{\mathfrak{A}}$, and f_q are chosen, the parameters θ_q and Θ_q are learned from the training data. Thus, not only can SMRF consider a large number of relations defined in terms of a small number of primitive components, but it also defines these relations specifically in terms of the data that it is given. This is a significant contribution of the SMRF approach, as it eliminates the problems associated with using pre-defined relations. In the SMRF approach, every relation is relevant to the data at hand, as its parameters are learned from the data – there are no irrelevant pre-defined relations sitting around to clutter up the learning process.

4.3 Categorical Variables

While SMRF trees are primarily designed to represent target concepts defined over continuous and multi-dimensional variables, they also have the capacity to represent concepts defined in terms of categorical variables as well. A *categorical variable* is here defined as an attribute A_i whose codomain $\mathfrak{S}_{A_i}$ is a denumerable set[4] whose items have no intrinsic ordering.

For continuous multi-dimensional variables, the acceptance set Δ_q is uncountably infinite, which requires it to be represented in terms of a pdf-threshold pair. Since the codomains of categorical variables are finite, the acceptance sets can be represented directly. For categorical variables, the question node model M_q has the following form, per Eq. 4.9:

$$M_q(I) = \begin{cases} 1 & \text{if } [\rho_q(\chi_q(I)) \in \Delta_q] \wedge \neg\varepsilon_q(I), \\ 0 & \text{if } [\rho_q(\chi_q(I)) \notin \Delta_q] \wedge \neg\varepsilon_q(I), \\ \mathbf{E} & \text{if } \varepsilon_q(I). \end{cases}$$

[4]These sets are finite in practice

The relational function f_q that defines ρ_q (Eq. 4.8) is generally the 'Identity' function. Given Eq. 4.11, the question node model for mapping functions can be defined as:

$$M_q(I) = \begin{cases} 1 & \text{if } [\phi_q(I) \in \Delta_q] \wedge \neg\varepsilon_q(I), \\ 0 & \text{if } [\phi_q(I) \notin \Delta_q] \wedge \neg\varepsilon_q(I), \\ \mathbf{E} & \text{if } \varepsilon_q(I). \end{cases} \tag{4.13}$$

For example, suppose that there is defined a certain categorical variable A, called *Shape*, whose codomain $\mathfrak{S}_A = \{Square, Rectangle, Triangle, Pentagon, Circle, Oval\}$. Suppose that a specific question node defines the concept: "Is the first object in the instantiation sequence round?" This question could be defined by a mapping function ϕ_q that returns the shape of the first object in the instantiation sequence, and an acceptance set $\Delta_q = \{Circle, Oval\}$.

Suppose that a group G contains three objects, $\{o_1, o_2, o_3\}$, and that $o_1.Shape = Circle$, $o_2.Shape = Triangle$, and $o_3.Shape = Oval$. Suppose that $I_1 = (o_1)$, $I_2 = (o_2)$, and $I_3 = (o_3)$. $\phi_q(I_1) = Circle$, and since $Circle \in \Delta_q$, $M_q(I_1) = 1$. Likewise, $M_q(I_3) = 1$, since $Oval \in \Delta_q$. On the other hand, $\phi_q(I_2) = Triangle$, and since $Triangle \notin \Delta_q$, then $M_q(I_2) = 0$.

4.4 Hypothesis Representation

As described above, a SMRF tree T is a method of representing a group target concept hypothesis H. However, as discussed in Chapter 3, the definition of a group target concept hypothesis (Eq. 3.10) specifies that the value of a hypothesis $H(G)$ is the maximal $h_l(I)$, for some $I \in \mathfrak{J}(G)$, and instance target concept hypothesis $h_l \in \mathcal{H}$.[5] As previously stated, one of the innovations of the SMRF approach is that it prunes out irrelevant subsets of the instance space, and it does this by sorting them into low-

[5] An analysis of this space is provided in Appendix C.

probability leaf nodes. Each root-to-leaf path in the tree, for each leaf l, implicitly defines an instance target concept hypothesis h_l, which, for a given instantiation sequence, I, is explicitly defined as:

$$h_l(I) = \Pr(l), \tag{4.14}$$

if and only if $I \in {}^{l}\mathcal{I}$, where ${}^{l}\mathcal{I}$ denotes the set of instantiation sequences sorted into leaf l, and $\Pr(l)$ denotes the probability assigned to leaf l. Thus, $\Pr(l)$ denotes the probability that an instantiation sequence sorted into leaf k exemplifies the target concept. Let the set of leaf nodes in a tree be denoted $\mathcal{L}$, and the leaf node into which an instantiation sequence I is sorted be denoted $\mathcal{L}(I)$. Then, the set of all instantiation sequences that fall down into the leaves of a tree is denoted ${}^{\mathcal{L}}\mathcal{I}$. The quantity $T(G)$, which denotes the probability that group G contains the target concept, as determined by tree T, is defined as follows:

$$T(G) = \max_{I \in {}^{\mathcal{L}}\mathcal{I}^G} h_{\mathcal{L}(I)}(I). \tag{4.15}$$

By Eq. 4.14, this is equivalent to

$$T(G) = \max_{I \in {}^{\mathcal{L}}\mathcal{I}^G} \Pr(\mathcal{L}(I)). \tag{4.16}$$

The representation of this concept can be more clear by introducing the concept of a winning leaf. The *winning leaf* of a group G given a tree T is the highest-probability leaf in T into which an instantiation sequence from G is sorted. Denoted $\mathcal{W}_T(G)$, the winning leaf is formally defined as:

$$\mathcal{W}_T(G) = \mathcal{L}\left(\arg\max_{I \in {}^{\mathcal{L}}\mathcal{I}^G} \Pr(\mathcal{L}(I))\right). \tag{4.17}$$

By Equation 4.17, Equation 4.16 can be rewritten as:

$$T(G) = \Pr\left(\mathcal{W}_T(G)\right). \tag{4.18}$$

Thus, if a particular group has an instantiation sequence that is sorted into a high-probability leaf node, then the group is considered, with that probability, to contain the target concept.

Chapter 5

Learning Algorithm

The fundamental objective of the SMRF tree learning algorithm is to grow a tree that can accurately predict whether or not a particular group contains the target concept. A SMRF tree probabilistically classifies a group as containing the target concept based upon the probability of the highest-probability leaf node into which an instantiation from the group in question is sorted. Because groups are probabilistically classified, the algorithm seeks to build a tree that will maximize the likelihood of correct group classification over a training set. By Equation 3.11 and Section 4.4, the likelihood of correct classification L given a SMRF tree T and a dataset $\mathcal{D}$ is defined as:

$$L(\mathcal{D}|T) = \left(\prod_{G \in G^+_{\mathcal{D}}} T(G) \right) \times \left(\prod_{G \in G^-_{\mathcal{D}}} (1 - T(G)) \right). \tag{5.1}$$

By Equation 4.18, L can be re-written as:

$$L(\mathcal{D}|T) = \left(\prod_{G \in G^+_{\mathcal{D}}} \Pr\left(\mathcal{W}_T(G)\right) \right) \times \left(\prod_{G \in G^-_{\mathcal{D}}} \left(1 - \Pr\left(\mathcal{W}_T(G)\right)\right) \right). \tag{5.2}$$

The likelihood of the data given the tree is thus higher when instantiation sequences from positive groups are sorted into leaf nodes with a high probability, and lower when no instantiation sequences from a positive group are sorted into a high-probability leaf. The likelihood of the data is also higher when no instantiation sequences from negative groups are sorted into high-probability leaves, and lower when at least one such instantiation sequence is sorted into a high-probability leaf.

The objective of the learning algorithm is to build a SMRF tree T that maximizes

$L(\mathcal{D}|T)$ for a given dataset $\mathcal{D}$. Learning a tree that maximizes L for a given dataset is an iterative, multi-step process. On the initial iteration, the learning algorithm begins with a "stub" – a trivial tree comprised of only a root node and a single leaf node. The tree is then incrementally grown according to the following greedy algorithm:

1. A set of leaf nodes is chosen for possible expansion (Section 5.2).
2. For each leaf node to be expanded:
   (a) A set of possible expansions is sampled, resulting in a set of candidate trees (Section 5.2.1).
   (b) For each candidate tree:
      i. The parameters of each question node model are chosen so as to maximize L (Section 5.3).
      ii. The tree leaf node probabilities are re-computed (Section 5.4).
3. The candidate tree with the greatest improvement to L is identified. If the improvement is statistically significant (Section 5.5), it replaces the current tree and the algorithm begins again at step 1. Otherwise, the algorithm halts.

The details of the above operations are explained in the following sections. Before going into detail about the algorithm, however, ways of maximizing the grand metric L will be discussed in Section 5.1.

5.1 Maximizing the Grand Metric

This section contains various derivations that demonstrate various methods of maximizing the grand metric L (Eq. 5.2).

5.1.1 Logarithmic Conversion

Since log is a monotonically increasing function, the solution that maximizes an arbitrary function $f(\theta)$ will also maximize the function $\log(f(\theta))$. Certain aspects of the learning algorithm (Sec. 5.3) are formulated in terms of the log-likelihood LL, which is derived as follows:

$$\begin{aligned}
L(\mathcal{D}|T) &= \left(\prod_{G\in G^+_{\mathcal{D}}} \Pr\left(\mathcal{W}_T(G)\right)\right) \times \left(\prod_{G\in G^-_{\mathcal{D}}} \left(1-\Pr\left(\mathcal{W}_T(G)\right)\right)\right) \\
LL(\mathcal{D}|T) &= \log\left[\left(\prod_{G\in G^+_{\mathcal{D}}} \Pr\left(\mathcal{W}_T(G)\right)\right) \times \left(\prod_{G\in G^-_{\mathcal{D}}} \left(1-\Pr\left(\mathcal{W}_T(G)\right)\right)\right)\right] \\
&= \log\left[\left(\prod_{G\in G^+_{\mathcal{D}}} \Pr\left(\mathcal{W}_T(G)\right)\right)\right] + \log\left[\left(\prod_{G\in G^-_{\mathcal{D}}} \left(1-\Pr\left(\mathcal{W}_T(G)\right)\right)\right)\right] \\
&= \sum_{G\in G^+_{\mathcal{D}}} \log\left(\Pr\left(\mathcal{W}_T(G)\right)\right) + \sum_{G\in G^-_{\mathcal{D}}} \log\left(1-\Pr\left(\mathcal{W}_T(G)\right)\right) \qquad (5.3)
\end{aligned}$$

Log-Likelihood for a Two-Leaf Tree

In the case of a tree T_2 with only two leaves, l_1 and l_2, LL becomes:

$$\begin{aligned}
LL(\mathcal{D}|T_2) &= \sum_{G\in G^+_{\mathcal{D}}} \log\left(\Pr\left(\mathcal{W}_T(G)\right)\right) + \sum_{G\in G^-_{\mathcal{D}}} \log\left(1-\Pr\left(\mathcal{W}_T(G)\right)\right) \\
&= \sum_{G\in G^+_1} \log\left(\Pr\left(l_1\right)\right) + \sum_{G\in G^+_2} \log\left(\Pr\left(l_2\right)\right) \\
&\quad + \sum_{G\in G^-_1} \log\left(1-\Pr\left(l_1\right)\right) + \sum_{G\in G^-_2} \log\left(1-\Pr\left(l_2\right)\right) \\
&= \left|G^+_1\right| \log\left(\Pr\left(l_1\right)\right) + \left|G^+_2\right| \log\left(\Pr\left(l_2\right)\right) \\
&\quad + \left|G^-_1\right| \log\left(1-\Pr\left(l_1\right)\right) + \left|G^-_2\right| \log\left(1-\Pr\left(l_2\right)\right), \qquad (5.4)
\end{aligned}$$

where G_l^+ denotes the set of positive graphs that are "won" by l, and G_l^- denotes the set of negative graphs that are "won" by l.[1] This result is used in Section 5.3.

5.1.2 Optimal n-Leaf Probabilities

In the case of a general T with any set of leaves $\mathcal{L}$ whose cardinality is at least one, the optimal leaf node probabilities for each leaf $l \in \mathcal{L}$ can be determined by setting the derivative of of L, with respect to the probability of a given leaf l, $\Pr(l)$, equal to zero, and solving for $\Pr(l)$. First, L can be simplified into a form more conducive to differentiation:

$$\begin{aligned} L(\mathcal{D}|T) &= \left(\prod_{G \in G_{\mathcal{D}}^+} \Pr\left(\mathcal{W}_T(G)\right) \right) \times \left(\prod_{G \in G_{\mathcal{D}}^-} \left(1 - \Pr\left(\mathcal{W}_T(G)\right)\right) \right) \\ &= \prod_{l \in \mathcal{L}} \prod_{G \in G_l^+} \Pr(l) \prod_{G \in G_l^-} (1 - \Pr(l)) \\ &= \prod_{l \in \mathcal{L}} \Pr(l)^{|G_l^+|} \left(1 - \Pr(l)\right)^{|G_l^-|} \\ &= \prod_{l \in \mathcal{L}} p_l^{\pi_l} \left(1 - p_l\right)^{n_l}, \end{aligned} \tag{5.5}$$

where G_l^+ denotes the set of positive graphs "won" by leaf l, and G_l^- denotes the set of negative graphs "won" by leaf l. In addition, π_l denotes the number of positive graphs "won" by leaf l, n_l denotes the number of negative graphs "won" by leaf l, and p_l is a simpler way of writing $\Pr(l)$. Taking the derivative of Equation 5.5 with

[1] Formally, a graph G is "won" by the leaf $\mathcal{W}_T(G)$. This concept is covered in more detail in the discussion of leaf node probabilities, in Section 5.4

respect to p_l,

$$\begin{aligned}
\frac{\partial L}{\partial p_l} &= \frac{\partial}{\partial p_l} \prod_{l' \in \mathcal{L}} p_{l'}^{\pi_{l'}} \left(1 - p_{l'}\right)^{n_{l'}} \\
&= \frac{\partial}{\partial p_l} p_l^{\pi_l} \left(1 - p_l\right)^{n_l} \left[\prod_{l' \in \mathcal{L} \cap \overline{\{l\}}} p_{l'}^{\pi_{l'}} \left(1 - p_{l'}\right)^{n_{l'}} \right] \\
&= \left[\left(1 - p_l\right)^{n_l} \frac{\partial}{\partial p_l} p_l^{\pi_l} + p_l^{\pi_l} \frac{\partial}{\partial p_l} \left(1 - p_l\right)^{n_l} \right] \left[\prod_{l' \in \mathcal{L} \cap \overline{\{l\}}} p_{l'}^{\pi_{l'}} \left(1 - p_{l'}\right)^{n_{l'}} \right] \\
&= \left[\left(1 - p_l\right)^{n_l} \pi_l p_l^{\pi_l - 1} - p_l^{\pi_l} n_l \left(1 - p_l\right)^{n_l - 1} \right] \left[\prod_{l' \in \mathcal{L} \cap \overline{\{l\}}} p_{l'}^{\pi_{l'}} \left(1 - p_{l'}\right)^{n_{l'}} \right]. \qquad (5.6)
\end{aligned}$$

Setting $\frac{\partial L}{\partial p_l}$ (Eq. 5.6) equal to 0,

$$\begin{aligned}
0 &= \left[\left(1 - p_l\right)^{n_l} \pi_l p_l^{\pi_l - 1} - p_l^{\pi_l} n_l \left(1 - p_l\right)^{n_l - 1} \right] \left[\prod_{l' \in \mathcal{L} \cap \overline{\{l\}}} p_{l'}^{\pi_{l'}} \left(1 - p_{l'}\right)^{n_{l'}} \right] \\
0 &= \left(1 - p_l\right)^{n_l} \pi_l p_l^{\pi_l - 1} - p_l^{\pi_l} n_l \left(1 - p_l\right)^{n_l - 1} \\
p_l^{\pi_l} n_l \left(1 - p_l\right)^{n_l - 1} &= \left(1 - p_l\right)^{n_l} \pi_l p_l^{\pi_l - 1} \\
p_l n_l &= \pi_l \left(1 - p_l\right) \\
n_l p_l &= \pi_l - \pi_l p_l \\
\pi_l p_l + n_l p_l &= \pi_l \\
p_l \left(\pi_l + n_l\right) &= \pi_l \\
p_l &= \frac{\pi_l}{\pi_l + n_l}. \qquad (5.7)
\end{aligned}$$

This result matches the frequentist intuition of how leaf node probabilities should be assigned – particularly, that the probability that a graph "won" by leaf l is positive is approximated by the frequency with which positive graphs were "won" by l during the learning process. This frequency is defined as the number of positive graphs that were "won" by l, divided by the total number of graphs that were "won" by l. Or, in

mathematical notation, $\frac{\pi_l}{\pi_l + n_l}$.

5.2 Choosing Leaves for Expansion

In order to make the algorithm more efficient, only a subset of the available leaves are considered for expansion. To select the leaves to expand, the algorithm scores each of the leaves, and selects the n highest-scoring leaves above a minimum threshold. In this work, n was empirically set to three. Each leaf is scored according to how much L would improve if the leaf were to be replaced by a hypothetical beneficial expansion.

The results of a hypothetical beneficial expansion of a given leaf l are determined heuristically by separating the instantiation sequences drawn from groups for which l is of maximal probability from the instantiation sequences drawn from groups for which l is not of maximal probability. A leaf l is of maximal probability for a group G if an instantiation sequence drawn from G is present at l, and if the probability associated with leaf l is greater than the probability of any other leaf into which an instantiation sequence from G is present. Such a split is beneficial because it separates those instantiation sequences that are most strongly representative of the hypothesis defined by the root-to-leaf path to l from those that are not.

After making this hypothetical split, the leaf node probabilities that would result in each of the branches are computed (Section 5.4). Next, $L(\mathcal{D}|T_H)$ is computed for this new hypothetical tree T_H. This score provides a measure for distinguishing those leaves that could result in large possible classification improvement from those that could not result in any substantial improvement, and such a distinction allows computational resources to be allocated more efficiently in building the SMRF tree.

5.2.1 Generating Candidate Trees

The expansion process replaces a leaf node in the tree with a partial tree, called an *expansion*. All legal expansions contain one question node, which has three leaf nodes as children (on its Yes, No, and Error branches, respectively). Some legal expansions also include one or more instantiation nodes above the question node. The expansion process thus enables the tree to ask a new question about previously instantiated objects, about new objects, or incorporating both. The expansion process enables the tree to either ask a new question about already-instantiated objects, or to instantiate new objects and ask a question about them. To avoid the polynomial-time complexity of a multi-ply search, all legal expansions are restricted to containing one question node only.

Each candidate question node is determined by its mapping function. Each mapping function is associated with a particular pdf, such as a Gaussian or von Mises distribution. The pdf parameters and decision threshold are learned at a later stage in the algorithm (Section 5.3).

A number of mapping functions are defined by the experimenter, allowing the learning algorithm to ask a variety of types of questions relevant to the problem domain. "Absolute" mapping functions simply return the value of an attribute at a particular position in the instantiation sequence. For example, a mapping function might return the location of the third object in the instantiation sequence. "Relative" mapping functions return the (vector) difference between the values of a particular attribute for two different objects. For example, a mapping function might compute the vector difference in RGB space between the second and third object of the instantiation sequence. Mapping functions that compute the Euclidean distance between two attribute values are also defined. A complete list of the mapping functions used in this work is provided in Section 5.2.2.

For each leaf node selected for expansion, a number of candidate expansions are individually applied to the tree, resulting in a set of candidate trees. The parameters of each candidate tree are then optimized with respect to L, according to the procedures described in Sections 5.3 through 5.4.

5.2.2 Mapping Functions Employed

Below is a list of the mapping functions that were employed in this work, to produce the results discussed in Section 7. The pdf associated with each mapping function is also stated.

- **Identity Location**: Provides the spatial location of the object specified by the mapping template. Associated with a Gaussian pdf.
- **Difference Location**: Provides the difference vector between the spatial locations of the two objects specified by the mapping template. Associated with a Gaussian pdf.
- **Distance Location**: Provides the Euclidean distance between the spatial locations of the two objects specified by the mapping template. Associated with a uni-dimensional Gaussian pdf.
- **Identity Color**: Provides the RGB color value of the object specified by the mapping template. Associated with a Gaussian pdf.
- **Difference Color**: Provides the difference vector between the RGB color values of the two objects specified by the mapping template. Associated with a Gaussian pdf.
- **Identity 2D Orientation**: Provides the orientation, in polar coordinates, of the object specified by the mapping template. Associated with a von Mises pdf.

5.3 Optimizing Question Node Models

Question node decision volume parameters are optimized[2] with respect to L using a version of the covariant aggregation method developed by Palmer et al. (2014). This method optimizes pdf parameters by greedily including the instances from nearby bags, for which the maximum likelihood estimated model produces the best split according to the likelihood metric L (Eq. 5.2).

To provide context, some theoretical details pertaining to the optimization procedure are discussed first in Section 5.3.1. The details of the optimization procedure itself are then discussed in Section 5.3.2.

5.3.1 Theoretical Prolegomenon

Maximizing L is difficult, due to the presence of the max operator. One possible approach is to approximate L using softmax, and then differentiate L with respect to the various question node models. However, this becomes a very complex proposition for trees of any moderate degree of complexity. In order to avoid these complexities, the decision was made to optimize the parameters of each question node using a principled heuristic method. The optimization procedure avoids the complexities of maximizing L by solving a much easier problem: maximizing the data likelihood of the candidate expansion subtree.

Recall from Section 5.2.1 that a SMRF tree is expanded by replacing a leaf node with a question node subtree. That is, the leaf node is replaced by a question node (or possibly an instantiation node, followed by a question node), with three leaf node children. The optimization procedure works to select the parameters of this new

[2] I use the term *optimize* in this context to denote the application of an algorithm to a set of model parameters, wherein the aforementioned algorithm seeks to maximize the predictive or classificatory performance of the model with respect to some metric or scoring function, in either a greedy or non-greedy manner. As such, this use of the term does not discriminate between algorithms that can provably obtain the global optimum and those that only obtain an approximation of the global optimum in general.

question node so as to maximize the data likelihood of this local subtree. As such, this optimization procedure circumvents the the complexities of seeking to maximize the data likelihood L (Eq. 5.2) of the entire tree, and instead proceeds under the assumption that maximizing the data likelihood of the expansion subtree will lead to an increase in data likelihood for the tree as a whole.[3] This assumption works well in practice, as evidenced by the experimental results in Chapter 7.

While attempting to maximize the data likelihood of the entire tree is often a hard problem, maximizing the data likelihood of a tree containing only one question node is relatively easy. The local subtree being optimized will consist of one question node and three leaf nodes. One leaf node will correspond to the Error branch of the question node. Since instantiation sequences sorted down this branch cannot be evaluated by the question node model, the sorting of instantiations into this branch cannot be affected by changes to the question node model. As such, the Error branch leaf, being "constant" in this sense, is independent of the parameters of the question node pdf with respect to L, and thus has no role in the optimization process.

As a result, the likelihood of the candidate subtree T_s can be defined in terms of the likelihood of a two-leaf tree for the purposes of question node optimization. Moreover, given that maximizing $\log L$ is equivalent to maximizing L, the score to be maximized becomes:

$$LL_{T_s} = \pi_Y \log(p_Y) + n_Y \log(1 - p_Y) + \pi_N \log(p_N) + n_N \log(1 - p_N), \tag{5.8}$$

where π_Y denotes the number of positive graphs that are "won" by the Yes branch leaf,[4] n_Y denotes the number of negative graphs that are "won" by the Yes branch

[3] If a case should arise where this assumption is violated for all candidate expansion models, none of the models will be incorporated into the tree, due to the significance testing procedure described in Section 5.5. In the worst possible case, the learning algorithm will simply halt without expanding the tree. It will never expand the tree in a way that causes classification performance (on the training set) to decrease.

[4] That is, for a given graph G, the yes branch leaf is $\mathcal{W}_{T_s}(G)$.

leaf, π_N denotes the number of positive graphs that are "won" by the No branch leaf, and n_N denotes the number of negative graphs that are "won" by the No branch leaf. This equation is equivalent to Equation 5.4, and follows from the derivations in Section 5.1.1. In addition, p_Y denotes the probability of the Yes branch leaf, and p_N denotes the probability of the No branch leaf. As demonstrated in Section 5.1.2, these values can be set to the following quantities in order to maximize LL:

$$p_Y = \frac{\pi_Y}{\pi_Y + n_Y} \tag{5.9}$$

and

$$p_N = \frac{\pi_N}{\pi_N + n_N}. \tag{5.10}$$

These equations follow from Equation 5.7, which provides the optimal probabilities for an n-leaf tree.

There are many possible ways to sort the instantiation nodes at the candidate expansion question nodes into the Yes and No branches. Supposing that the leaf node probabilities are always set according to Equations 5.9 and 5.10, some sortings will produce higher values of LL_{T_s} than others. The goal of the optimization procedure is to find the question node model that produces the highest value of LL_{T_s}. Formally stated, then, the optimization problem is as follows: given a candidate expansion subtree, T_s, containing a single question node, find the question node model pdf parameters θ and threshold Θ that maximize LL_{T_s}, given that leaf node probabilities are assigned according to Equations 5.9 and 5.10. The optimization procedure that is designed to solve this problem is described in Section 5.3.2.

5.3.2 The Optimization Procedure

The question node model optimization procedure is described in Algorithm 1, under the name FindBestCollectionModel. The general idea of the algorithm is to it-

eratively build a collection of graphs,[5] and then estimate the parameters of a question node model from the graphs in that collection, using maximum likelihood estimation. The algorithm begins by building a decision volume around a sampled graph. Then, at each iteration, all of the graphs that fall within the decision volume of the current question node model are considered for inclusion into the collection. The graph is added to the collection that best improves LL_{T_s}, and a new question node model is computed from the new collection. This process continues until no more graphs can be added to improve LL_{T_s}. This process is restarted a specified number of times, and the best model from these runs is kept.

FindBestCollectionModel uses several sub-procedures. The SampleFrom procedure samples one item from a set of items. The Initialize procedure (Alg. 3) initializes the collection, and the GrowCollection procedure (Alg. 2) expands the collection as much as possible, and computes new question node model parameters from the graphs in the collection. The Score procedure (Alg. 5) effectively computes LL for the candidate subtree. These latter three procedures are discussed in more detail below.

For a given question node q being optimized, FindBestCollectionModel takes the following parameters:

- G_q^+, the set of all positive graphs for which at least one instantiation sequence has been sorted down to node q,
- G_q^-, the set of all negative graphs for which at least one instantiation sequence has been sorted down to node q,
- ϕ_q, the mapping function used by the question node model M_q,

[5]For the sake of introducing the algorithm without first going into all of the details, this description is somewhat generalized. As is explained later in this section, each graph is represented by an instantiation sequence, and these sequences are technically the members of collections in the actual algorithm implementation. However, it is valid to say that graphs are present in collections by proxy, and this serves to make the high-level summary easier to follow.

- n_{COLLS}, a parameter that determines how many collections to construct, and
- k_{SAMPLE}, a parameter that determines how many graphs to sample in each iteration of the outer loop of GROWCOLLECTION.

Algorithm 1 Finding the Best Collection-Based Model.

function FINDBESTCOLLECTIONMODEL(G_q^+, G_q^-, ϕ_q, n_{COLLS}, k_{SAMPLE})
  $ll \leftarrow -\infty$ ▷ Log-likelihood for the candidate expansion tree
  for $j = 1 \rightarrow \min(\left|G_q^+\right|, n_{\text{COLLS}})$ **do**
    $G_j \leftarrow \text{SAMPLEFROM}(G_q^+)$
    $\mathcal{C}, \theta', \Theta' \leftarrow \text{INITIALIZE}(\mathcal{I}^{G_j}, G_q^+, G_q^-, \phi_q)$
    $\theta', \Theta' \leftarrow \text{GROWCOLLECTION}(\mathcal{C}, G_q^+ \cap \overline{\{G_j\}}, G_q^+, G_q^-, \phi_q, \theta', \Theta', k_{\text{SAMPLE}})$
    $ll' \leftarrow \text{SCORE}(G_q^+, G_q^-, \phi, \theta', \Theta')$
    if $ll' > ll$ **then**
      $\theta \leftarrow \theta'$
      $\Theta \leftarrow \Theta'$
      $ll \leftarrow ll'$
    end if
  end for
  return θ, Θ
end function

This procedure contains a central loop. In each iteration, a graph is sampled without replacement from the set G_q^+ (denoted by the SAMPLEFROM procedure). An initial collection, as well as initial pdf parameters and an initial likelihood threshold, are obtained from this graph through the INITIALIZE procedure. This threshold and parameter set are then optimized through the GROWCOLLECTION procedure. This model is kept if it is better (in terms of LL_{T_s}) than any previous model, otherwise it is discarded. In the call to the GROWCOLLECTION procedure, the sampled graph is removed from the whole set of positive graphs represented at q. The whole set of positive graphs represented at q, with the exception of the sampled graph, forms the initial set of graphs used as potential collection membership candidates in GROWCOLLECTION. Since positive graphs encode the target concept, and negative graphs do not, negative graphs are not included in the set of graphs used as potential

collection membership candidates.

GrowCollection (Alg. 2) uses several sub-procedures that have not yet been mentioned. The SubSample procedure takes two parameters, a set S and an integer n, and returns a subset of n items sampled from S without replacement. The OptimizeThreshold (Alg. 4) procedure selects a threshold for the question node model in order to maximize LL_{T_s}. This procedure is described in more detail below.

The MLE procedure, which is also utilized by GrowCollection, estimates model parameters using maximum likelihood estimation (MLE).[6] This procedure takes a set of elements in $\mathbb{R}^n$ as input and returns pdf parameters θ. For notational convenience, a function $\mathcal{V}$ is defined which denotes the application of a mapping function ϕ to all of the instantiation sequences in a set. The function $\mathcal{V}$ is formally defined as follows:

$$\mathcal{V}(\mathcal{I}, \phi) = \{\phi(I) | I \in \mathcal{I}\}, \tag{5.11}$$

where $\mathcal{I}$ denotes a general set of instantiation sequences. The specific equations for the maximum likelihood estimators of various pdfs are given in Section D.

In order to fully explain the GrowCollection procedure, the notion of a *representative instantiation sequence* must be defined. The representative instantiation sequence of a graph G is the one for which the question node model returns the highest value. In formal terms, the model value of the representative instantiation

[6] To account for the tendency of maximum likelihood estimation to provide degenerate covariance estimates for small amounts of data, a maximum condition number equivalent to the square of the number of points used to compute the ML estimate is imposed upon Gaussian pdf covariance matrices. Thus, for the initial model estimated from one point, the covariance is constrained to be isotropic. But, as more data is incorporated into the ML estimate, the covariance is allowed to take on a more ellipsoidal shape. For one-dimensional metric spaces, no covariance is computed, and the variance parameter is set to unity. This is because in one-dimensional cases, the extent of the decision volume (and thus the sorting of instantiation sequences) can be controlled solely by the threshold Θ if the variance is set to a constant value. Likewise, for the von Mises distribution, which is also defined over a one-dimensional metric space, the concentration parameter κ is also set to unity.

sequence is denoted $V_\theta(G)$, and is defined as:

$$V_\theta(G) = \max_{I \in \mathcal{I}^G} p(\phi(I)|\theta), \tag{5.12}$$

where $\mathcal{I}^G$ denotes the set of instantiation sequences constructed from the objects in G that are present at the question node being optimized. The representative instantiation sequence of G is the I for which $p(\phi(I)|\theta) = V_\theta(G)$. With these concepts defined, the collection growth procedure can now be discussed in detail.

The collection growth procedure GrowCollection grows an initialized collection incrementally, evaluating all of the graphs whose representative instantiation sequences fall within the current model and have not previously been added to the collection, and selecting the graph whose addition best improves LL_{T_s}. The representative instantiation sequence of this graph is added to the collection, the model parameters are re-computed using the new collection, and the process is repeated. This procedure, as detailed in Algorithm 2, takes the following elements as input:

- An initialized collection $\mathcal{C}$,
- A set of graphs $\mathcal{G}$, for which the instantiation sequences of each member G that are present at the optimized node (which set is denoted $\mathcal{I}^G$) are candidates for membership in collection $\mathcal{C}$,
- The set of positive graphs G^+ which have at least one instantiation sequence present at the optimized node,
- The set of negative graphs G^- which have at least one instantiation sequence present at the optimized node,
- The question node mapping function ϕ,
- Initialized pdf parameters θ,

- An initialized likelihood threshold Θ, and
- A parameter k_{SAMPLE} which determines how many graphs to sample in each iteration of the algorithm.

Algorithm 2 Growing Collections.

function GROWCOLLECTION($\mathcal{C}$, $\mathcal{G}$, G^+, G^-, ϕ, θ, Θ, k_{SAMPLE})
  $ll \leftarrow \text{SCORE}(G^+, G^-, \phi, \theta, \Theta)$ ‘
  $\mathcal{G}' \leftarrow \left\{G | (G \in \mathcal{G}) \wedge (\exists I)(I \in \mathcal{I}^G) \wedge p(\phi(I)|\theta) \geq \Theta\right\}$
    ▷ The candidate graphs are those that fall within the initial model
  repeat
    $\hat{\mathcal{G}}' \leftarrow \text{SUBSAMPLE}(\mathcal{G}', \min(|\mathcal{G}'|, k_{\text{SAMPLE}}))$
      ▷ Subsample from the current set of candidate graphs
    $\overline{ll} \leftarrow -\infty$
    for all $G \in \hat{\mathcal{G}}'$ **do**
      $I' \leftarrow \arg\max_{I \in \mathcal{I}^G} p(\phi(I)|\theta)$
      $\mathcal{C}' \leftarrow \mathcal{C} \cup \{I'\}$ ▷ A candidate collection
      $\theta' \leftarrow \text{MLE}(\mathcal{V}(\mathcal{C}', \phi))$
      $\Theta' \leftarrow \text{OPTIMIZETHRESHOLD}(G^+, G^-, \phi, \theta')$
      $\overline{ll}' \leftarrow \text{SCORE}(G^+, G^-, \phi, \theta', \Theta')$
      if $\overline{ll}' > \overline{ll}$ **then**
        $\mathcal{C}'' \leftarrow \mathcal{C}'$ ▷ Keep the best candidate collection
        $\theta'' \leftarrow \theta'$ ▷ Keep the best candidate parameter set
        $\Theta'' \leftarrow \Theta'$
        $G' \leftarrow G$
        $\overline{ll} \leftarrow \overline{ll}'$
      end if
    end for
    if $\overline{ll} > ll$ **then**
      $\theta \leftarrow \theta''$
        ▷ If the best candidate collection improves performance, use the parameters
      $\Theta \leftarrow \Theta''$
      $ll \leftarrow \overline{ll}$
    end if
    $\mathcal{C} \leftarrow \mathcal{C}''$ ▷ The best candidate becomes the new collection
    $\mathcal{G} \leftarrow \mathcal{G} \cap \overline{\{G'\}}$ ▷ Remove the selected graph from the set of candidates
    $\mathcal{G}' \leftarrow \left\{G | (G \in \mathcal{G}) \wedge (\exists I)(I \in \mathcal{I}^G) \wedge p(\phi(I)|\theta) \geq \Theta\right\}$
      ▷ Only look at the subset of candidate graphs that fall within the new model
  until $|\mathcal{G}'| = 0$ ▷ Iterate until no more candidate graphs remain
  return θ, Θ
end function

This procedure contains a nested loop. The outer loop persists as long as there are some graphs that have representative instantiation sequences that fall within the model decision volume. The inner loop iterates over a subset of these graphs (the output of the SubSample function), the size of which is set equal to the minimum of k_{SAMPLE} and the size of the set of graphs being subsampled. For each such sampled graph, its representative instantiation sequence is added to the collection $\mathcal{C}$ to form a new collection $\mathcal{C}'$, and a new set of parameters θ' and a new threshold Θ' are computed from the members of $\mathcal{C}'$. After iterating through all sampled graphs, the best such model computed in the inner loop is then compared against the model maintained by the outer loop. If it is better, in terms of LL, it becomes the new standard maintained by the outer loop.

Regardless of whether or not the inner loop produces a better model than the one maintained by the outer loop, the representative instantiation sequence from the best inner loop model is added to the collection $\mathcal{C}$, and the graph from which that instantiation sequence was instantiated is removed from the list of eligible graphs $\mathcal{G}$. In this way, the collection grows greedily, adding as many points as it can, but only making use of the models that provide the best LL score. Since the algorithm only considers points that already fall within the collection-defined model for prospective membership, this provides a degree of robustness against outliers and allows clusters to be found even in noisy conditions.

With the details of the main optimization procedure discussed, details of the sub-procedures Initialize, Score, and OptimizeThreshold will now be given.

The collection initialization procedure Initialize begins by considering the set of instantiation sequences, $\mathcal{I}^G$, that are instantiated from a graph G and present at the question node being optimized. Each graph may only have one representative instantiation sequence present in a collection. As such, Initialize iterates through the set $\mathcal{I}^G$, building a model around each element. The element whose model has the

highest LL score is selected to be the first member of the collection $\mathcal{C}$. This procedure is detailed in Algorithm 3.

Algorithm 3 Initializing Collections.

```
function INITIALIZE(I^G, G^+, G^-, φ)
    C ← {}                                          ▷ The collection being initialized
    ll ← −∞
    for all I ∈ I^G do
        C' ← {I}                                    ▷ The candidate "new" collection
        θ' ← MLE(V(C', φ))
        Θ' ← OPTIMIZETHRESHOLD(G^+, G^-, φ, θ')
        ll' ← SCORE(G^+, G^-, φ, θ', Θ')
        if ll' > ll then                            ▷ If the new score is better ...
            C ← C'                                  ▷ Keep the new collection
            θ ← θ'                                  ▷ Keep the new pdf parameters
            Θ ← Θ'                                  ▷ Keep the new model threshold
        end if
    end for
    return C, θ, Θ
end function
```

Given a pdf defined by parameters θ, optimal thresholds are determined by the OPTIMIZETHRESHOLD procedure described in Algorithm 4. First, the model evaluations of the representative instantiation sequences present at the optimized node are computed and sorted in descending order. Next, each of these values is used as a candidate value for the likelihood threshold Θ, and the LL score of the data present is computed with respect to each candidate threshold. The candidate threshold that results in the highest LL score is chosen, and the best threshold is set at this value, plus half the difference between this value and the next highest candidate threshold. This is done to add some "padding" to the model, and avoid having member elements directly on the model boundary. It should also be noted that Algorithm 4 makes use of the "sequence builder" notation described in Section 4.2.

Given a mapping function ϕ, a parametric probability density defined by parameters θ, and a likelihood threshold Θ, the SCORE function computes the log likelihood

Algorithm 4 Optimizing Decision Volume Thresholds.

function OPTIMIZETHRESHOLD(G^+, G^-, ϕ, θ)
  $\mathbf{V}_\theta \leftarrow \{V_\theta(G) \,|G \in (G^+ \cup G^-)\}$
  $\mathbf{V}_{ll} \leftarrow (\text{SCORE}(G^+, G^-, \phi, \theta, \mathbf{V}_\theta[k]))_{k \in [1, |\mathbf{V}_\theta|]_\mathbb{Z}}$ ▷ The score for each value in $\mathbf{V}_\theta$
  $\mathbf{V}_{ll} \leftarrow \text{SORTDESCENDING}(\mathbf{V}_{ll})$
  $\Theta \leftarrow \mathbf{V}_{ll}[1] + \frac{\mathbf{V}_{ll}[1] - \mathbf{V}_{ll}[2]}{2}$
  return Θ
end function

of the single-question candidate expansion subtree whose question node model is defined by ϕ, θ, and Θ. In order to define leaf node probabilities, SCORE defines all graphs whose representative instantiation sequences are sorted down the Yes branch as being "won" by the Yes branch leaf. Likewise, SCORE defines all graphs whose representative instantiation sequences are sorted down the No branch as being "won" by the No branch leaf. With these numbers in hand, the score, LL_{T_s}, is computed according to Equation 5.8. This process is described in Algorithm 5.

Algorithm 5 Scoring PDFs.

function SCORE(G^+, G^-, ϕ, θ, Θ)
  $Y^+ \leftarrow \{G | G \in G^+ \wedge V_\theta(G) \geq \Theta\}$ ▷ Positive graphs "within the model"
  $Y^- \leftarrow \{G | G \in G^- \wedge V_\theta(G) \geq \Theta\}$ ▷ Positive graphs "outside the model"
  $N^+ \leftarrow G^+ \cap \overline{Y^+}$ ▷ Negative graphs "within the model"
  $N^- \leftarrow G^- \cap \overline{Y^-}$ ▷ Negative graphs "outside the model"
  $\pi_Y \leftarrow |Y^+|$
  $n_Y \leftarrow |Y^-|$
  $\pi_N \leftarrow |N^+|$
  $n_N \leftarrow |N^-|$
  $p_Y \leftarrow \frac{\pi_Y}{\pi_Y + n_Y}$
  $p_N \leftarrow \frac{\pi_N}{\pi_N + n_N}$
  $ll \leftarrow \pi_Y \log(p_Y) + n_Y(1 - \log(p_Y)) + \pi_N \log(p_N) + n_N(1 - \log(p_N))$
  return ll
end function

Upon completion of the FINDBESTCOLLECTIONMODEL procedure, the next step in the optimization process is to compute the probabilities of the leaf nodes in the candidate tree, as discussed in Section 5.4.

5.4 Computing Leaf Node Probabilities

The last step of the optimization procedure is to assign probabilities to the leaf nodes of the tree that maximize L (Eq. 5.2) for the current sorting. This task is made difficult by the fact that L is defined in terms of max operators. However, as demonstrated in Section 5.1.2, if instantiation sequences instantiated from p positive graphs and n negative graphs are present at a leaf, then the leaf node probability would be set to $\frac{p}{n+p}$ (Eq. 5.7) in order to maximize the grand metric (Eq. 5.2). As noted in Section 5.1.2, the $\frac{p}{n+p}$ ratio gives the frequentist probability of positive graphs being "won" by the leaf under consideration. A graph is "won" by a leaf in the case that the leaf contains an instantiation sequence instantiated from that graph, and no other leaves containing instantiation sequences from that graph have a higher probability. Or in formal terms, a graph G is "won" by the leaf $\mathcal{W}_T(G)$, as defined by Equation 4.18.

Consider that a root-to-leaf path in a SMRF tree defines a specific hypothesis h_l (Sec. 4.4), and thus $\Pr(l)$ denotes the probability that h_l covers a target concept. Since positive graphs exemplify the target concept, it also makes intuitive sense that the probability of the correctness of h_l should be calculated in terms of the frequentist probability that positive graphs will be "won" by this leaf. After all, if 100% of the graphs won by the leaf are positive, then one would intuitively think h_l correct, and if 0% of the graphs won by the leaf are positive, then one would intuitively think h_l wrong. However, the $\frac{p}{n+p}$ ratio does not make sense only on frequentist grounds. As shown in Appendix B, the ratio also turns out to be the optimal value on a Bayesian analysis.

However, one cannot assign $\frac{p}{n+p}$ as the probability of each leaf, without first determining which leaf is the "winner" for each graph. However, according to Equation 4.17, the winning leaf is defined in terms of the leaf node probabilities. This

presents something of a "chicken and the egg" problem, as winning leaves are required to compute leaf probabilities, and leaf probabilities are required to determine winning leaves.

This problem is solved by an iterative procedure called AssignProbabilities, described in Algorithm 6. The idea is to iteratively compute the $\frac{p}{n+p}$ for all leaves and all graphs, assign the leaf with the highest ratio value as the "winner" for the graphs that have instantiation sequences at that leaf, and then to remove that leaf and that set of graphs from consideration in the next iteration of the algorithm. Additionally, the algorithm ensures that each leaf that receives a non-zero probability receives a *unique* non-zero probability. If the $\frac{p}{n+p}$ value for a given leaf l is not less than the probability p' assigned (to a different leaf l') in the previous iteration, then the leaf l is assigned a probability equal to p' minus very small amount ϵ.

Describing the process in more detail, the algorithm begins by considering all graphs in the dataset, and all leaf nodes in the tree. The dataset contains a set G^+ of positive graphs, and a set G^- of negative graphs. For notational convenience, let also G_l^+ denote the set of positive graphs for which at least one instantiation sequence is present at leaf l. Likewise, let G_l^- denote the set of negative graphs for which at least one instantiation sequence is present at leaf l.

The algorithm begins an iteration by computing the probability $\frac{|G_l^+|}{|G_l^+|+|G_l^-|}$ for each leaf l. The leaf l' with the highest such value is then assigned that value as its probability, and all of the graphs in $G_{l'}^+$ and $G_{l'}^-$ are then considered to be "won" by l. As such, those graphs are removed from the sets G^+ and G^-, respectively, for the next iteration. The leaf l' is also removed from consideration. Beginning with all of the tree leaves and all of the graphs in the dataset, this iteration proceeds as described until both G^+ and G^- are empty. Any remaining leaves are assigned a probability of zero.

Algorithm 6 Leaf probability assignment.

procedure AssignProbabilities($\mathcal{L}$, G^+, G^-)
  $p'' \leftarrow 1 + \epsilon$
  repeat
    $\mathbf{V}_{\mathcal{L}} \leftarrow \left\{ \frac{|G_l^+|}{|G_l^+|+|G_l^-|} | l \in \mathcal{L} \right\}$
    $l' \leftarrow \arg\max_{l \in \mathcal{L}} \mathbf{V}_{\mathcal{L}}$
    $p' \leftarrow \frac{|G_{l'}^+|}{|G_{l'}^+|+|G_{l'}^-|}$
    $\mathcal{L} \leftarrow \mathcal{L} \cap \overline{\{l'\}}$
    $G^+ \leftarrow G^+ \cap \overline{G_{l'}^+}$
    $G^- \leftarrow G^- \cap \overline{G_{l'}^-}$
    if $p' \geq p''$ **then**
      $\Pr(l') \leftarrow p'' - \epsilon$
    else
      $\Pr(l') \leftarrow p'$
    end if
    $p'' \leftarrow \Pr(l')$
  until $|G^+| + |G^-| = 0$
end procedure

5.5 Evaluating Statistical Significance

Given the set of optimized candidate expansions, the candidate tree T_c that best maximizes L is considered as a replacement for the current tree T. In particular, the successor tree that maximizes the difference in log-likelihood (see Eq. A.25) is selected as T_c. As demonstrated in Appendix A, choosing the tree that maximizes the difference in log-likelihood is equivalent to choosing the tree that maximizes information gain.

If this new tree performs significantly better than the current tree according to a likelihood ratio test, then this replacement is kept. According to Huelsenbeck and Crandall (1997), if the sample size is reasonably large, then the test statistic s, defined as:

$$s = -2 \log \frac{L(\mathcal{D}|T)}{L(\mathcal{D}|T_c)}, \tag{5.13}$$

is approximately χ^2 distributed. For computing the p-value, the degrees of freedom of the χ^2 distribution are considered to be the number of leaf and instantiation nodes, plus the sum of the number of distribution parameters in each question node. To compensate for the multiple comparisons problem (Jensen and Cohen, 2000), we employ a Bonferroni correction to obtain a collective cutoff of $\alpha = 0.1$.

As demonstrated in Appendix A (Eq. A.28), the test statistic s is proportional to the information gain induced by the new graph sorting of T_c.

Chapter 6

Data Generation

This Chapter describes the various procedures utilized for generating synthetic data for use in the experiments described in Chapter 7.

The development of a synthetic example generator was motivated by a desire to test the abilities of the SMRF algorithm in a precise and controlled fashion not generally afforded by the complexities and difficulties associated with real-world data collection. As such, the decision was made to generate example data for SMRF using computational means. However, as explained in Section 7.1.2 this does not mean that such generated data is entirely synthetic. Rather, such data also incorporates attribute values whose distributions match those found in the real world.

Another desideratum was the ability to specify example concepts in a subset of first-order logic, as SMRF trees naturally encode first-order existentially-quantified expressions, with instantiation nodes serving as quantifiers, question nodes serving to define predicates, and leaf node values along with the branching structure of the tree itself serving to determine the logical connectives between predicates.

An additional desideratum was a meaningful distribution of negative examples. An early attempt at writing an example generator simply assigned random values to the attributes of objects in negative examples, checking afterward to make sure that the negative examples did not contain the target concept. While such a method produces valid data, such data does not force SMRF to flesh out the target concept in the learning process. With such data, selecting only one element of the target concept is generally sufficient to distinguish positives from negatives. For example, given the

concept *Blue above Green*, it is unlikely that there will be more than a few negatives that have an object with a color attribute in the *Blue* region of RGB space, if the color attributes are randomly assigned from a uniform distribution over the RGB space. As such, the SMRF learning algorithm can simply learn a tree with one question picking out the color *Blue*, and it will rightly classify almost all of the examples. It will generally classify so many examples correctly, that further expansions of the tree are deemed statistically insignificant.

Thus, while such data are valid, they are not particularly interesting from an empirical standpoint. To test the abilities of the SMRF algorithm (along with its competitor algorithms), it is desirable to produce datasets whose negatives generally cover the "negation space" of the target concept. For example, if N negative examples of the *Blue above Green* concept were generated, a set covering the "negation space" of the concept would comprise some examples that have a *Blue* object, but not a *Green* object, some examples that have a *Green* object, but not a *Blue* object, some examples that have to objects in the *above* relation that are neither *Blue* nor *Green*, and so on. Having such a distribution of negatives prevents the learning algorithm from latching on to one element of the target concept, and forces the consideration of a much larger subset of the target concept elements.

To accomplish these goals, an algorithm was developed that facilitated the generation of data in the following stages.

6.1 Concept Definition

Before generating examples expressing a particular target concept, the target concept must first be defined in a subset of first order logic where only existential quantification is permitted, and where each quantification implies the unique identity of the object denoted by the variable being quantified. For example, $\exists x \exists y$ would imply that $x \neq$

y. Each predicate was semantically grounded by being associated with a particular question node model (Ch. 4). For example, a predicate $red(x)$ would be associated with a question node model that assigns 1 to an instantiation sequence whose i^{th} object has a *color* attribute in a "red" region of RGB space defined by the model. This association is done in terms of attribute and relation classes. For example, *red* is defined as a color attribute associated with one model, and *above* is defined as a location relation associated with another model. This association in terms of attribute and relation classes serves to define the problem domain.

In order to facilitate expressing concepts in this manner, as well as manipulating them in further stages of the process, a first-order symbolic logic system was implemented in Python.

After defining the logic expression that captures the concept, and defining the problem domain, the only remaining step is to define a few parameters that control the randomization of generated attribute values (Sec. 6.4), such as attribute value bounds. No further input is required from the user, as the symbolic logic and attribute randomization systems take care of everything else automatically.

It should be noted that real-world characteristics were injected into the datasets described in Section 7.1.2 by associating color predicates with a question node model defined in terms of Gaussian distributions modeled upon the RGB pixel distribution of an image of a "real world" object of that color. For example, the model associated with the *red* predicate was based upon the RGB distribution of an image of a "real world" red object.

6.2 Enumerating the Concept Space

The basic idea behind this example generator procedure is that *positive concept space* is equivalent to the set of mutually non-isomorphic models[1] that satisfy the target concept definition. Likewise, the *negative concept space* is equivalent to the set of mutually non-isomorphic models that satisfy the negation of the target concept definition. The set of positive examples is obtained by generating an equal number of examples that correspond to each model that satisfies the target concept. Likewise, the set of negative examples is obtained by generating an equal number of examples that correspond to each model that satisfies the negation of the target concept. The process of obtaining examples from models is described in more detail in Section 6.3.

The procedure of enumerating the concept space is performed as follows. First, the concept space definition is obtained. For the positive concept space, this is simply the target concept definition. For the negative concept space, this is the negation of the target concept definition.

Second, information from the problem domain is used to augment the target concept definition, using universal quantification. This augmentation serves to constrain the generation process to prevent bogus values from being included. For example, if the concept is $\exists x \exists y(blue(x) \wedge green(y) \wedge above(x, y)$ (where x and y are implicitly non-identical, as described above), then this step would conjunctively add clauses such as $\forall x(green(x) \supset \neg blue(x))$ and $\forall x \forall y(above(x, y) \supset \neg above(y, x))$ and $\forall x(\neg above(x, x))$.[2] The first clause is an example of the *exclusion* constraint: an object can only be predicated by one of the predicates in the problem domain corresponding to a particular attribute type (such as color). For example, if x is predicated by green (that is, $green(x)$ is true), then x could not be predicated by any other color predicate (that is, neither $blue(x)$, $red(x)$, etc. could be true). The second clause is an

[1] That is, a set of first-order models, of which no two are isomorphic to each other.

[2] The symbol $\supset$ denotes material implication.

example of the *anti-symmetric* constraint, which forbids symmetric binary relations. The third clause is an example of the *anti-reflexive* constraint, which forbids reflexive binary relations. The exclusion constraint is applied to all unary predicates (i.e. attribute values), and the anti-symmetric and anti-reflexive constraints are applied to all binary relations (i.e. attribute relations). This augmented expression is then simplified. The procedure is performed for both the target concept and its negation.

Third, a partial first-order model is constructed for the augmented target concept. As such, the model consists of a set of objects and a set of predicate definitions. The set of objects is populated by inserting an object for each existentially quantified variable. This procedure follows from the implicit non-identicality assumption, and greatly simplifies the model construction procedure. Each predicate present in the augmented target concept is used to populate the list of predicates, but the definitions of those predicates are left empty at this time. The procedure is performed for both the augmented target concept and its augmented negation.

Fourth, the augmented expression is propositionalized according to the partial first-order model constructed in the previous step. Propositionalization proceeds by enumerating the set of objects in the model at each quantifier, and producing a propositional variable whose name is the combination of the predicate and the object. For example, if a domain contained two objects, a and b, the expression $\exists x P(x)$ would be propositionalized as $Pa \vee Pb$, where Pa and Pb are propositional variables corresponding to the first order expressions $P(a)$ and $P(b)$, respectively. Likewise, the expression $\forall x P(x)$ would be propositionalized as $Pa \wedge Pb$. This procedure is performed both for the augmented target concept and its augmented negation. Each expression is then simplified and converted into conjunctive normal form (CNF).

Fifth, a truth table is constructed for the CNF expression. Given the truth table, the set of propositional models that satisfy (i.e. render True) the CNF expression are enumerated. In this context, the term *propositional model* is used to denote the

mapping of a set of propositional variables each to either True or False. This procedure is performed both for the CNF propositionalized augmented target concept and the CNF propositionalized augmented target concept negation.

It should be noted that the above process is simply one method of solving the propositional satisfiability (SAT) problem. For contexts with a large number of propositional variables, using truth tables becomes computationally intractable. As the expressions are already in conjunctive normal form, any standard SAT-solving technique could theoretically be used in lieu of a truth table for this step. However, for datasets generated for the work presented in this dissertation, the total number of propositional variables was small enough for truth tables to be used, and truth tables were a much simpler solution, in terms of implementation time, than other SAT-solving techniques.

Sixth, for each propositional model enumerated in the previous step, the equivalent first-order version is constructed. This is done by first converting each propositional variable into an equivalent first-order expression. For example, Rab would be converted into $R(a, b)$. The set of objects in the first order model is populated by the union of the parameters of all of the predicates and relations in the upgraded first-order expressions. Each predicate definition is then populated by the union of the parameter tuples for all expressions whose corresponding propositional variables were mapped to True by the propositional model. For example, a propositional model $\{Pa : True, Pb : False, Pc : True, Rab : True\}$ would produce an equivalent first-order model with a set of objects $\{a, b, c\}$ and predicate definitions $\{P : \{(a), (c)\}, R : \{(a, b)\}\}$.

Seventh, for both the target concept and its negation, each set of models is pruned of models that are isomorphic to one or more other models in the set. This pruning step ensures a set of models that are not elementarily equivalent to any other model.

These two sets of models represent the enumeration of the positive concept space,

and negative concept space, respectively. These sets are then repeatedly and equally sampled (with replacement) according to the number of positive and negative examples to be generated. For example, if there were two positive models, and four negative models, then 50 positive examples would be generated from each positive model, and 25 negative examples would be generated from each negative model. The process of generating an example from a model is discussed in the following section (Sec. 6.3).

6.3 Generating Examples from Models

Example generation is a function of a first-order logic model and a problem domain specification, which includes lists of attribute and relation types, and the bounds of the possible values of each attribute type. The general idea is that the combination of the model and the problem domain specification provides enough information to generate a group of objects that are valid participants in some part of the concept space. This procedure only generates "pure" examples – that is, examples that correspond directly to the model. Extra elements, such as distractors, are added in the procedure described in Section 6.4.

An example is generated by creating an example object (Ch. 3) for each object in the model, and then assigning object attributes according to the model predicate definitions. Attribute values are assigned in three stages. In the first stage, each unary predicate definition is enumerated. If the object corresponding to the example object is present in the predicate definition, then a value is sampled from the question node model associated with that predicate, and assigned to the object as the value of the attribute that the problem domain description associates with the predicate. For example, suppose the model contains two objects, a and b. The generated example would contain two example objects, o_1 and o_2, and where o_1 corresponds to a, and

o_2 corresponds to b. Suppose that the model contains two unary predicates, *red* and *green*, whose definitions are as follows: $red = [(a)]$, $green = [(b)]$. The 3-vector (corresponding to the three dimensions of RGB space) c_1 would be sampled from the question node model associated with *red*, and the 3-vector c_2 would be sampled from the question node model associated with *green*. These values would then be assigned as object attributes, such that $o_1.color = c_1$ and $o_2.color = c_2$.

The second stage of attribute assignment is to to assign binary relational values. This procedure begins like the preceding, enumerating the object tuples in each binary predicate definition in the first-order logic model. Likewise, a value is sampled from the question node model associated with the binary predicate. However, value assignment is not as straightforward as it is with unary predicates, since the relevant attribute values of both objects must combine a certain way to produce the sampled value. For example, suppose that values are being populated for the *above* relation. Suppose further that (a, b) (for the objects a and b defined above) is a member of predicate definition of *above*. If z denotes the value sampled from the *Difference Location* model associated with the *above* relation, then $o_1.location - o_2.location$ must equal z.

In some cases, only one of the two objects may have already been assigned a value for the attribute associated with the relation. In this case, the inverse of the relation is applied to produce the attribute value for the second object. For example, if $o_1.location$ has been assigned, but not $o_2.location$, then $o_2.location = o_1.location - z$. Likewise, if only $o_2.location$ has been assigned, then $o_1.location = z + o_2.location$. Only invertible relations were used in this work. However, in contexts where non-invertible relations are employed, the inverse function could be approximated through sampling-based methods.

If the attribute value has not been assigned for either object, then the attribute value of one object is randomly assigned, drawn uniformly from the space defined

by the bounds of that attribute (as specified in the problem domain description). The attribute value for the second object is then found using the aforementioned inverse method. If attribute values have been assigned for both objects, then the user is warned that the problem is over-constrained, and one of the attribute values is overwritten using the aforementioned inverse method. None of the problem sets generated for this work were over-constrained in this manner.

The third and final stage of attribute assignment is to assign random values to object attributes that were not assigned in the first two stages. This example generation procedure assumes that each example should contain homogeneous objects, in terms of attributes. As such, for each attribute in the problem domain definition, any object that has not yet been assigned a value for that attribute is assigned a random value drawn uniformly from the space defined by the attribute value bounds.

Given the group of objects generated by this procedure, the example is finalized using the method described in Section 6.4.

6.4 Randomizing Attribute Values

The idea behind the randomization procedure is that the group generated in the procedure described in Section 6.3 should fall down into a specific leaf of an "ideal" SMRF tree describing the section of the concept space that is covered by the first-order logic model from which the group was generated. Given this sorting, distractors (objects with random attribute values) can be added to the group as long as their addition does not change the sorting of the group. Likewise, in order to add more randomness to the example, the attribute values of each object can be randomly perturbed an arbitrary number of times, given that each perturbation does not change the sorting of the group.

This method is based upon two further principles: (1) that the first-order logic

model from which the group was generated can be converted into an equivalent existential first-order logic expression where the uniqueness of existentially quantified variables is assumed, and (2) that an existential first-order logic expression where the uniqueness of existentially quantified variables is assumed can be converted into an equivalent SMRF tree.

The randomization procedure can therefore be partitioned into four separate processes that occur sequentially. First, the first-order logic model from which the example group was generated is converted into an equivalent first-order logic expression where the uniqueness of existentially-quantified variables is assumed. Second, this first-order logic expression is converted into an equivalent SMRF tree, and the example group is sorted down the tree, and the "winning" leaf (Ch. 5) is noted. Third, the attribute values of each object in the group (including distractors) are randomly perturbed for some specified number of iterations. Fourth, some number of distractor objects (as specified by the user) are added to example group, such that the addition of each object does not change the "winning" leaf sorting. Each process will be described more fully in turn.

6.4.1 Generating Trees from Models

The general idea of the first process is that a first-order logic model can be converted into an equivalent first-order logic expression (with the implied uniqueness of existentially quantified variables) by conjunctively asserting that each predicate in the model holds for the tuples of objects in its definition and that each predicate does not hold for all possible tuples of objects not present in its definition, given the *anti-reflexive* constraint previously discussed in Section 6.2. For example, suppose a model has the set of objects $\{a, b, c\}$ and three predicates P, Q, and R, where $P = \{(a), (b)\}$,

$Q = \{(a, b), (b, c)\}$, and $R = \{(b), (c)\}$. The equivalent expression would be:[3]

$$\exists x \exists y \exists z \quad (P(x) \wedge P(y) \wedge \neg P(z) \wedge \neg Q(x) \wedge Q(y) \wedge Q(z) \wedge R(x, y) \wedge R(y, z) \wedge \neg R(y, x) \wedge \neg R(z, y) \wedge \neg R(x, z) \wedge \neg R(z, x)).$$

This expression can then be converted into an equivalent SMRF tree.

The general idea of the second process is that, for a tree with class labels at the leaf nodes, each element of the expression produced by the previous process is equivalent to some tree component, or operation upon a tree. Since the uniqueness of existentially quantified variables is assumed, each existential quantifier is equivalent to an instantiation node in a SMRF Tree. As such, given a tree T_ϕ representing an expression ϕ, $T_{\exists x(\phi)}$ can be obtained by adding an instantiation node under the root node of T_ϕ.

Likewise, a predicate assertion is equivalent to a subtree – specifically, a question node with two leaf nodes, where the Yes branch leaf node has the positive class label (represented by a probability of 1.0), and the No branch leaf node has the negative class label (represented by a probability of 0.0). For a predicate assertion $P(a, b)$, let this subtree be denoted $T_{P(a,b)}$.

Similarly, a negated predicate is equivalent to the same subtree, except with the leaf node probabilities swapped. As it turns out, given T_ϕ, $T_{\neg\phi}$ can be obtained by swapping the leaf node probabilities (0.0 for 1.0, and vice-versa) of all of the leaves in T_ϕ.

Lastly, for two expressions ϕ and ψ, their conjunction is equivalent to replacing every positive leaf node in the subtree T_ϕ with a copy of the subtree T_ψ.

To produce a tree, the expression is parsed, placing each existential quantifier

[3]Per the preceding description of how these expressions are generated, this concept contains predicate assertions corresponding to the tuples present in the predicate definitions in the model, as well as negated predicate assertions corresponding to all possible tuples not present in the predicate definitions in the model, given the *anti-reflexive* constraint.

on a stack. Next, a question node subtree is generated for each unique predicate assertion (e.g. $red(x)$ and $red(y)$ are two different predicate assertions). Next, the aforementioned transformations for each logical connective are recursively applied for the whole expression. Lastly, each existential quantifier is taken off the stack and inserted into the tree as an instantiation node.

After the tree is produced, the mapping template of the mapping function of each question node model is set so as to map the order of the instantiation nodes to the order of the variables of the predicate assertion upon which the question node model was based. For example given the expression $\exists x \exists y P(x) \wedge R(y, x)$, the first instantiation node would correspond to x, and the second to y. As such, the mapping template of the question node model corresponding to $P(x)$ would be (0), and the mapping template of the question node model corresponding go $R(y, x)$ would be $(1, 0)$.

6.4.2 Randomizing Attribute Values and Generating Distractors

Given the tree generated in the prior procedure, the example group is sorted down the tree, and the winning leaf is noted. The attributes values of the objects in the group are then randomly perturbed while preserving the group sorting. The randomization procedure is not strictly necessary, but it does add randomness to the group. It also enables the generation of examples if groups cannot be generated from first-order logic models.

Attributes are randomized according to a specified "random walk," that describes the randomization procedure for each attribute value. There are three possible randomization procedures. The first chooses a random value for the attribute uniformly from the space defined by the attribute bounds defined by the problem domain specification. This procedure is used for attributes whose values are not determined by the target concept description. The second method perturbs the attribute value by

a small random amount. Repeating this procedure results in a random walk akin to Brownian motion. This procedure is used for attributes whose values are determined by unary predicates in the target concept. The third method perturbs the values of the attributes of two separate objects by the same random amount. This produces the same kind of random motion as the previous method, except that the attribute values of the two objects move together throughout the space. This method is used for the attribute values of two objects that stand together in a binary relation in the target concept.

Each attribute value is randomized some specified number of times. At each step, the specific randomization procedure is applied, and the group is sorted down the tree. If the sorting is good, the change is kept. If not, the change is discarded and the randomization procedure is tried again. This process repeats a certain specified number of times, or until an acceptable sorting is achieved. If the repeated application of the randomization procedure cannot produce an acceptable sorting, the algorithm gives up and moves to the next attribute, where this same process is applied.

After randomizing the group object attribute values, a specified number of distractor objects is added to the group. The procedure is similar to the first attribute randomization procedure. Distractor objects are added one at a time. For each object, random values are selected for each attribute, drawn uniformly from the space defined by the attribute bounds defined by the problem domain specification. The object is tentatively added to the group, and the new group is sorted down the tree. If the sorting is good, the change is kept. If not, the new object is discarded and the procedure is tried again. This process repeats for a certain specified number of time, or until an acceptable sorting is achieved. If an acceptable distractor object cannot be produced, the graph remains unchanged and an error is thrown.

6.5 Summary

The data generation procedure described in this appendix provides a powerful method of generating examples that cover the full positive and negative concept spaces of a target concept. From a restricted first-order logic expression describing the concept, and a brief description of the problem domain, any number of positive and negative examples can be generated that collectively cover the target concept space. This is made possible by the fact that these first-order expressions can be propositionalized, and that these resulting propositional expressions can be applied to SAT-solving techniques to find the set of propositional models that satisfy each expression. These propositional models can then be upgraded into equivalent first-order models, which can then be further converted into groups of objects, and by way of restricted first-order logic expressions, SMRF trees. The ability to convert between these various kinds of representation allows the generation of the high-quality data utilized in the experiments reported in Chapter 7.

Chapter 7

Experiments and Results

The SMRF algorithm was tested on a number of datasets, and its performance was compared to the performance of a number of related learning algorithms on those same datasets. The goal of these experiments is to test the ability of the SMRF algorithm to learn concepts of varying complexity, at varying levels of noise. Additionally, these experiments were performed with the objective of obtaining a quantitative comparison of the SMRF algorithm and related algorithms on these same datasets. SMRF was compared to TILDE (Sec. 2.1.4), IAPR (Sec. 2.2.1), DD, EM-DD (Sec. 2.2.2), CITATION-KNN (Sec. 2.2.3), MI-SVM, MI-SVM, and MICA (Sec. 2.2.5). These datasets were chosen to provide a diverse assortment of different kinds of MIL algorithms to test SMRF against.

These experiments utilize datasets that contain target concepts defined in terms of the color, orientation, and spatial attributes and relationships of synthetic objects that are components of multi-object structures. The target concepts present in the data involve not only concepts of single attributes, but also complex concepts involving relations between object attributes. For example, one dataset encodes the relatively simple concept that either a red object or a green object is present. Another dataset encodes a more complex target concept (a blue object above a green object), which is defined over both the color attributes of two objects, as well as a relation between the three-dimensional locations of those objects. The data for these experiments were generated using the procedure described in Chapter 6. It should be noted that while these datasets were synthetically-generated, the color attributes were sampled from

color models built from images of real objects. As such, the color attributes in the datasets reflect the covariance properties of the colors of real-world objects.

The goal of this set of experiments was to demonstrate five important qualities of the SMRF learning algorithm, in relation to other comparable learning algorithms. First, that the algorithm can construct complex target concepts consisting of multiple questions. Second, that the algorithm is able to learn disjunctive concepts, including complex concepts that contain a mix of conjunctive and disjunctive elements. Third, that the algorithm can find an appropriate target concept even when "distractor" objects (which do not play a role in the true target concept definition) are present. Fourth, that the algorithm is robust to the presence of mislabeled training examples. And fifth, that the algorithm is capable of learning concepts that are defined in terms of real-world covariances.

7.1 Experimental Configuration

The experiments described in Section 7.1.5 used a number of datasets, involved a number of learning algorithms, and utilized a number of performance metrics. Each of these are discussed in turn.

7.1.1 SMRF Configuration

To enable the concepts defined in Section 7.1.2 to be learned, the following mapping functions were provided to the SMRF learning algorithm for all experiments:

- **Identity Location**: Provides the three-dimensional spatial location of a single object. Associated with a Gaussian pdf.
- **Difference Location**: Provides the difference vector (in three dimensions) between the spatial locations of two separate objects. Associated with a Gaussian pdf.

- **Distance Location**: Provides the Euclidean distance between the spatial locations of two separate objects specified by the mapping template. Associated with a Gaussian pdf.
- **Identity Color**: Provides the RGB color value of a single object. Associated with a Gaussian pdf.
- **Difference Color**: Provides the difference vector between the RGB color values of two separate objects. Associated with a Gaussian pdf.
- **Identity 2D Orientation**: Provides the two-dimensional orientation (in the XY plane), in polar coordinates, of a single object. Associated with a von Mises pdf.

The other relevant parameters of the SMRF algorithm are discussed in Chapter 5.

Additionally, the SMRF learning algorithm was implemented with an auto-pruning procedure, which prunes candidate expansions that would produce more than a certain large number[1] of instantiation sequences. This has the practical effect of including single-object questions early in the tree that prune away most of the instance space, enabling two-object questions to be asked further down the tree without causing a dramatic increase in the number of instantiation sequences present.

7.1.2 Datasets

The datasets used in these experiments each encoded one of the following target concepts:

- **Red or Green (R / G)**: A red object present in the scene, or a green object present in the scene.

[1] Any value that works well for a particular computing platform can be used. The value 10^5 was used in these experiments.

- **Red or Green or Blue or Yellow (R / G / B / Y)**: A red, or green, or blue, or yellow object present in the scene.
- **Blue above Green (BaG)**: A blue object positioned above a green object in three-dimensional space.
- **Blue above Green or Green above Blue (BaG / GaB)**: A blue object positioned above a green object (in three-dimensional space), or a green object positioned above a blue object.
- **Red above Green or Green above Blue (RaG / GaB)**: A red object positioned above a green object, or a green object positioned above a blue object.
- **Red above Green or Blue above Yellow (RaG / BaY)**: A red object positioned above a green object, or a blue object positioned above a yellow object.
- **Right Red right of Left Green [(RtR)ro(LtG)]**: A red object oriented to the right (in the two-dimensional orientation space of the XY plane) to the right of a green object oriented to the left.

These concepts are defined over three different kinds of attributes (color, orientation, and position), and span a range of complexity, including purely disjunctive concepts, purely conjunctive concepts, and concepts that are a mix of conjunctive and disjunctive. The simplest pure disjunctive concepts are **Red or Green** and **Left or Right**. The concept **Red or Green or Blue or Yellow** is harder for the algorithm, since there are essentially four separate concepts present which divide up the dataset, leaving only a fourth of the total amount of training data for each individual disjunct. The only pure conjunctive concept is **Blue above Green**. The last three concepts employ a mix of conjunctive and disjunctive, and grow progressively more complex,

being defined over 2, 3, and 4 separate colors, respectively. Some visualizations of some of these datasets are provided in Appendix F.[2]

In order to make these problems more difficult, as well as to simulate certain aspects of real-world data collection, two "dimensions" of noise were added to the datasets. First, so-called "distractor" objects were added to the examples. A "distractor" is defined as an object that plays no role in the target concept. Datasets were generated with zero, two, five, and ten distractor objects. The addition of distractors makes the concept more difficult to learn, as the learning algorithm must work to identify the key players in the target concept, in addition to learning what the concept itself is. This dimension of noise is also often present in real-world environments.

A second dimension of noise was also added to the datasets, to simulate mistakes made by those who determine the training set labels. For the various datasets, a percentage (from 0% to 18%) of the training set examples were randomly flipped (from positive to negative, or vice-versa). The trees learned on these corrupted datasets were then tested on test sets with uncorrupted example labels. In addition to making the problem more difficult, this dimension of noise simulates mistakes in example labeling, which is not a rare phenomenon in real-world data collection.

For all of the data sets, 100 positive example graphs and 100 negative example graphs were generated, according to the specific target concept involved (Chapter 6). For training and testing, 10-fold cross-validation was employed. For most experiments, the datasets were subsampled to some smaller number of positive and negative graphs.

Per the discussion of Section 2.2.1, a conscientious decision was made to compare the SMRF algorithm to other MIL and tree-based learning approaches, not on the conventional benchmark datasets such as MUSK and MUTAGENESIS, but on the re-

[2]These figures provide a "SMRF's-eye" view of the data, visualizing the distribution of the values produced when the dataset is processed through some of the various mapping functions listed in Section 7.1.1.

lational datasets described above. There are a number of reasons for this decision. First, as Xu (2003) and Lodhi and Muggleton (2005) contend, MUSK and MUTAGENESIS are not well-suited for use as benchmarks for evaluating and comparing MIL algorithms. As such, their historical use as benchmarks by the MIL community has generally been conventional, rather than principled. The relational datasets described above have been specifically designed to express *bona fide* MIL concepts. In contrast, Xu (2003) argues that the multiple-instance qualities of MUSK and MUTAGENESIS are questionable at best.

Second, using conventional benchmark datasets would not allow the addition of distractor objects, since generative models for those datasets do not exist. As such, there is no way to determine with certainty whether or not adding additional objects would invalidate or corrupt the examples. However, as generative models do exist for the relational datasets described above, distractor objects can be easily added without corrupting the generated examples. The ability to add distractor objects allows the robustness of the SMRF algorithm to distractors to be directly compared to the ability of other algorithms to deal with distractor objects.

Last, as discussed in Section 8.1.1, SMRF is not designed to solve problems where the data already exists in the form of labeled bags of feature vectors (i.e., instances). Rather, SMRF is designed to solve problems where the data exists in the form of labeled groups of attributed objects. As such, SMRF is designed to handle real-world problems (such as might be encountered in robotics), where the world is presented to the agent as a set of objects with attributes. This approach has the advantage of presenting the learning agent with a more natural and straightforward view of the world, and avoids the problem of having to select or fine-tune feature generation heuristics. Additionally, representing the world as a collection of attributed objects enables SMRF to build up a set of meaningful instantiation sequences incrementally, pruning out a large section of the instance space which is not target-conceptual. As

such, presenting SMRF with bags of pre-calculated feature vectors does not allow one to obtain a very accurate picture of the range of SMRF's learning capabilities, as compared to testing SMRF on data which represent physical domains in a more straightforward and natural way.

7.1.3 Comparison Algorithms

The following algorithms were used in experiments discussed in this chapter:

- TILDE: As discussed in Section 2.1.4, TILDE (Blockeel and Raedt, 1998) is an inductive logic programming approach that builds first-order logical decision trees. The implementation of TILDE used in these experiments was obtained by request from Blockeel (2012). The algorithm is implemented as a closed-source binary executable that accepts datasets and configuration information in the form of Prolog files.

- IAPR: *Iterated Discrimination Axis-Parallel Rectangles.* IAPR is the best-performing APR algorithm discussed by Dietterich et al. (1997), and is the *de facto* benchmark for comparison on the MUSK1 and MUSK2 datasets. The implementation of IAPR used in these experiments was obtained as a part of the MILL package (Yang, 2013). The algorithm is implemented in MATLAB.

- DD: *Diverse Density.* Introduced by Maron and Lozano-Pérez (1998), the diverse density algorithm is discussed in Section 2.2.2. The implementation of DD used in these experiments was obtained as part of the MILL package (Yang, 2013). The algorithm is implemented in MATLAB.

- EM-DD: Introduced by Zhang and Goldman (2002), EM-DD transforms the DD approach with the addition of an EM-like procedure (Sec. 2.2.2). The implementation of EM-DD used in these experiments was obtained as part of the MILL package (Yang, 2013). The algorithm is implemented in MATLAB.

- Citation-kNN: As discussed in Section 2.2.3, Citation-kNN (Wang and Zucker, 2000) is an MIL approach that extends the k-nearest neighbor algorithm. The implementation of Citation-kNN used in these experiments was obtained as part of the MILL package (Yang, 2013). The algorithm is implemented in MATLAB.

- mi-SVM: As discussed in Section 2.2.5, mi-SVM (Andrews et al., 2003) is an "instance-oriented" SVM approach to solving MIL problems. The implementation of mi-SVM used in these experiments was obtained as part of the MISVM package (Doran, 2013). The algorithm is implemented in Python.

- MI-SVM: As discussed in Section 2.2.5, MI-SVM (Andrews et al., 2003) is a "bag-oriented" SVM approach to solving MIL problems. The implementation of MI-SVM used in these experiments was obtained as part of the MISVM package (Doran, 2013). The algorithm is implemented in Python.

- MICA: As discussed in Section 2.2.5, MICA (Mangasarian and Wild, 2008) is another "bag-oriented" SVM approach to solving MIL problems. The implementation of MICA used in these experiments was obtained as part of the MISVM package (Doran, 2013). The algorithm is implemented in Python.

An optimal combination of parameters for each of the above algorithms was selected through a grid search. A set of combinations of parameters was generated for each algorithm. For each combination, a 10-fold experiment was performed on the *Red above Green or Blue above Yellow* dataset.[3] The parameter combination that performed the best[4] over its 10-fold run was selected for the experiments described in Section 7.1.5.

[3]This dataset was chosen, because it encodes the most complex target concept, and therefore is arguably the most difficult.

[4]That is, it had the highest mean PSS on the test set. See Section 7.1.4 for more details on the performance metrics used in this research.

Since these comparison algorithms are incapable of inferring relations between objects given only the sets of object attributes, each example graph in each dataset had to be preprocessed in order to encode pertinent relational information. To provide this encoding for a particular graph, a feature vector was created for each possible 2-tuple of distinct objects in the graph. For a given tuple (a, b), the feature vector was populated with the attributes of object a followed by the attributes of object b.[5] Following the object attributes, the vector was populated with the result of evaluating $\phi((a, b))$, for each binary mapping function ϕ listed in Section 7.1.1.[6] These feature vectors were then labeled according to the graph from which they were drawn. This procedure was used to pre-process data for all comparison algorithms except for TILDE. As such, each approach using this procedure required ${}_nP_2$ features to be generated for each graph, where there are n objects in the graph.

Preprocessing data for TILDE involved generating Prolog files. First, each of the mapping functions listed in Section 7.1.1 was declared as a Prolog predicate. Unary mapping functions, such as *Identity Location*, were defined as predicates of the form $\phi(o, x, y, z)$, where o denotes an object, and x, y, and z denote location coordinates. Likewise, binary mapping functions, such as *Difference Location*, were defined as predicates of the form $\phi(a, b, x, y, z)$, where a and b denote objects, and x, y, and z denote components of the difference location vector between a and b. TILDE accepts examples formatted as a collection of Prolog facts. One fact is the example (graph) label. The other facts represent various attributes and relations between objects in the example. To provide these facts, one fact was generated per each object in the graph, per each unary mapping function. Additionally, one fact was generated per every possible 2-tuple of unique objects in the graph, per each binary mapping function.

[5] Object attributes were provided by evaluating the unary (i.e. *Identity*) mapping functions listed in Section 7.1.1

[6] That is, *Distance Location*, *Difference Location*, and *Difference Color*.

As discussed above, generating features for the comparison algorithms effectively required generating ${}_nP_2$ features, for n objects in the graph. Generating data for TILDE effectively required generating $n + {}_nP_2$ Prolog facts. As such, running experiments became prohibitively expensive at higher distractor levels, especially for the algorithms drawn from the MISVM package. As such, experiments with comparison algorithms were generally only run up to 5 distractors. Only experiments involving TILDE and EM-DD were run at the 10 distractor level, because these algorithms were of special interest for comparison against SMRF. TILDE was of interest because of the dominant position it holds in the relational and tree-learning domains, and EM-DD, because it performed the best overall at the 5 distractor level and below.

7.1.4 Performance Metrics

In order to facilitate accurate comparisons with other learning approaches which provide classification labels (not probabilities), SMRF classification probabilities were rounded to 0 and 1 (cutoff at 0.5), respectively, to provide class labels. It should be noted that, per Section 5.4, rounding a leaf probability (with cutoff at 0.5) produces the majority class label for bags "won" by that leaf. Such rounding thus facilitates a direct comparison to approaches such as TILDE, which reports majority class labels during classification.

The Peirce Skill Score (PSS) was used to assess performance. Classification accuracy is a popular metric for assessing classification performance in the MIL literature. However, using only classification accuracy can produce misleading results, especially if the algorithm is biased towards choosing the most common label. Whereas classification accuracy only considers the relative number of correct classifications, PSS considers both correct and incorrect classification in its score value. PSS is defined

as the hit rate minus the false alarm rate, as follows:

$$PSS = \frac{TP}{TP + FN} - \frac{FP}{FP + TN},$$

where TP, FP, FN, and TN denote the number of true positives, false positives, false negatives, and true negatives, respectively. As such, PSS has a value of 1 for perfect classification, 0 for random classification, and -1 for perfectly incorrect classification.

Additionally, learning time was recorded for each algorithm on each run. Learning time is defined in this context as the time taken for the algorithm to complete the learning process for a given set of training data. This performance metric was collected simply to provide an impression of how SMRF compares to other comparable algorithms on these datasets, and to show that SMRF's benefits are not outweighed by a learning time that is disproportional to the rest of the field.

It should be noted that other implementations of these algorithms exist, and may provide better (or worse) performance than those used for these comparisons. However, these implementations do provide a good comparison with respect to the fact that all of the algorithms, TILDE alone excluded, are implemented in non-compiled interpreted programming languages. While better performance should be expected from implementations of these algorithms as compiled and optimized executables, it is unclear whether relative performance between the algorithms (TILDE excluded) would change significantly if all were implemented in this fashion. As such, the learning time comparisons presented here arguably provide a picture that demonstrates SMRF's general relationship to other algorithms in the field in terms of learning time.

In addition, the number of question nodes present in a learned tree provides a third metric of comparison between SMRF and TILDE, in addition to PSS and learning time. The number of question nodes present in a tree provides a general representation of the complexity of the description of the target concept hypothesis encoded by the

tree itself. All things being equal, simpler hypotheses are preferred to more complex hypotheses. As such, this metric facilitates the comparison between SMRF and TILDE as to the compactness of their target concept hypothesis representations.

7.1.5 Experimental Setup

Two main sets of experiments were performed. The first, described in Section 7.1.5, compares SMRF and the algorithms listed in Section 7.1.3 on the datasets described in Section 7.1.2, at a combination of different distractor and corruption levels. The goal of this experiment is to determine how well SMRF performs relative to other comparable algorithms on datasets of varying levels of complexity and at various levels of noise.

The second experiment, described in Section 7.3, compares SMRF to both TILDE and EM-DD on varying amounts of training data, at a fixed amount of noise (distractors and corruption). The goal of this experiment is to determine how the better-performing algorithms compare to SMRF with smaller amounts of training data, and then to determine how much training data is needed to achieve comparable performance.

Main Experiment

The goal of the main experiment is to test the performance of both the SMRF learning algorithm and the listed comparison algorithms on a combination of datasets, numbers of distractors, and percentages of examples corrupted.

A 10-fold cross-validation experiment was performed for all of the possible combinations of each of the following conditions:

- **Number of Distractors**: 0, 2, and 5,
- **Percentage of Examples Corrupted**: 0%, 3%, 6%, 9%, 12%, 15%, and 18%,

- **Data Sets**: *Red or Green*, *Red or Green or Blue or Yellow*, *Blue above Green*, *Blue above Green or Green above Blue*, *Red above Green or Green above Blue*, *Red above Green or Blue above Yellow*, *Right Red right of Left Green*.

The original experimental intent was to test all algorithms at 10 distractors as well. However, as discussed in Section 7.1.3, having to construct features from the explicit enumeration of all 2-permutations of objects in each graph rendered running times practically intractable for the algorithms in the MISVM package at 10 distractors. The MILL package was able to cope better, but it also faced problems with high running times. As such, only three algorithms were tested at the 10-distractor level: SMRF, TILDE, and EM-DD. Originally, only SMRF and TILDE were run at the 10-distractor level. After observing the performance of EM-DD on the 5-distractor datasets, the decision was made to test EM-DD at the 10-distractor level as well. The results of these experiments are presented in Section 7.2.

7.2 Results

The results are presented as follows. First, the behavior of SMRF as a function of distractors, corruption, and problem set type is discussed in Section 7.2.1. Next, a high-level classification performance analysis is presented for each algorithm, aggregated over all of the problem sets are presented in Section 7.2.2. Following from this high-level performance summary, more detailed comparisons of SMRF to both TILDE and EM-DD for each individual problem set are presented in Section 7.2.3. After this, results for each algorithm for other performance metrics, such as learning time and number of questions, are presented in Section 7.2.4.

7.2.1 SMRF Results

In this section, I discuss SMRF performance as a function of problem set type, distractors, and corruption. This performance is reported first by showing examples of learned trees at various conditions, to provide the reader a sense of how the SMRF learning algorithm functions under various conditions. After showing some illustrative examples in terms of trees, SMRF performance as a whole is analyzed in depth with an analysis of variance and *post hoc* tests.

Learned Trees

SMRF trees will be shown for the following datasets: *Blue above Green*, *Red or Green*, and *Red above Green or Blue above Yellow*. The first dataset is a purely conjunctive dataset, and is one of the easiest for SMRF to solve. The second is a purely disjunctive dataset. It is also relatively easy for SMRF to solve, though as one will see in Section 7.2.3, other algorithms tend of have difficulty with it. The last dataset is the most complex of all the datasets employed in this work, and is a disjunctive-conjunctive mixture of five distinct relations.[7]

Figure 7.1 shows a typical tree learned for *Blue above Green* at zero distractors and no corruption. In this figure, each instantiation node is depicted as a parallelogram with a single letter (such as "A" or "B"). This letter denotes the object that is instantiated at that node. Question nodes are denoted by rounded squares. In each question node, the mapping function is stated at the top, as a function of instantiated objects. The mean of the Gaussian pdf is listed, along with the "extent" of the covariance matrix. The "extent" is a matrix where each column is a scaled eigenvector of the covariance matrix. Each eigenvector is scaled by the product of the question node model threshold and the square root of the eigenvalue corresponding to that

[7] *Red*, *Green*, *Blue*, *Yellow*, and *Above*.

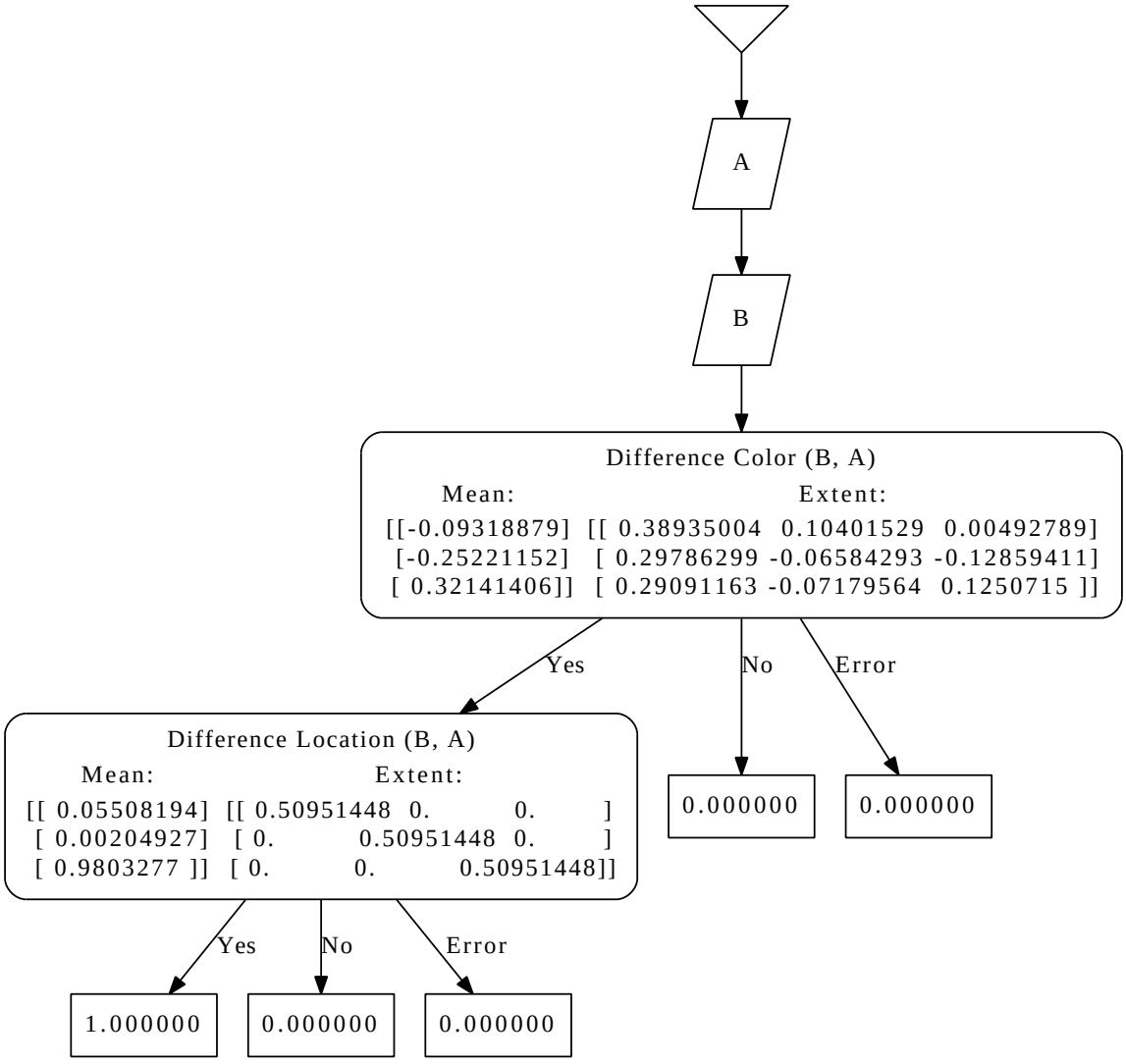


Figure 7.1: Typical *Blue Above Green* tree at zero distractors and no corruption.

eigenvector. As such, the "extent" matrix provides a set of principle axis vectors of the question node decision volume. This matrix is included to provide the reader with a sense of both the shape and the size of the decision volume. Lastly, each leaf node is represented as a rectangle, and the number in each rectangle is the probability that the tree assigns to examples that are "won" by that leaf.

This tree is able to learn the concept perfectly by asking two questions, one that confirms the existence of two objects whose difference in color is the difference between "blue" and "green" (such that the first object is blue), and another that confirms

that those two objects are located relative to each other in an "above" relationship (such that the first object is the high object). As such, this tree will sort at least one instantiation sequence from each positive example into the leftmost leaf, and all negative examples will be sorted into one of the other leaves. The leftmost leaf has a probability of 1, and the other leaves have a probability of 0. As such, this tree perfectly classifies the training data.

Note that though the concept seemingly requires three question nodes, SMRF is able to perfectly classify the data with only two. This is because there are no pairs of objects that exhibit the above relation, in addition to the "blue"-"green" color different relation, and are not "blue" and "green," respectively. Given that the learning algorithm is biased towards producing smaller trees, SMRF chooses to use two question nodes instead of three.

Figure 7.2 shows a typical tree learned for *Blue above Green* at two distractors and no corruption. As one can clearly see, the tree is nearly identical to the tree in Figure 7.1. Both question node models concern the same concepts: the "blue"-"green" difference and the above relationship. The question node model parameters, as one might expect, are also comparable, though not identical. However, in contrast to the zero distractor tree, the two distractor tree sorts a few positive examples down the No branch of the *Difference Location* question, which should contain negative examples only. As such, adding two distractors makes SMRF produce a few false negatives. However, the probability of the leftmost leaf is still 1, so no false positives are produced.

Figure 7.3 shows a typical tree learned for *Blue above Green* at five distractors and no corruption. This tree is larger than the previous two, as the tree asks a "blue" question, followed a "green" question, followed by an "above" question. Adding in 5 distractors (to only 2 actors) causes SMRF to abandon the difference color question, and ask two identity color questions instead. In doing so, it suffers no

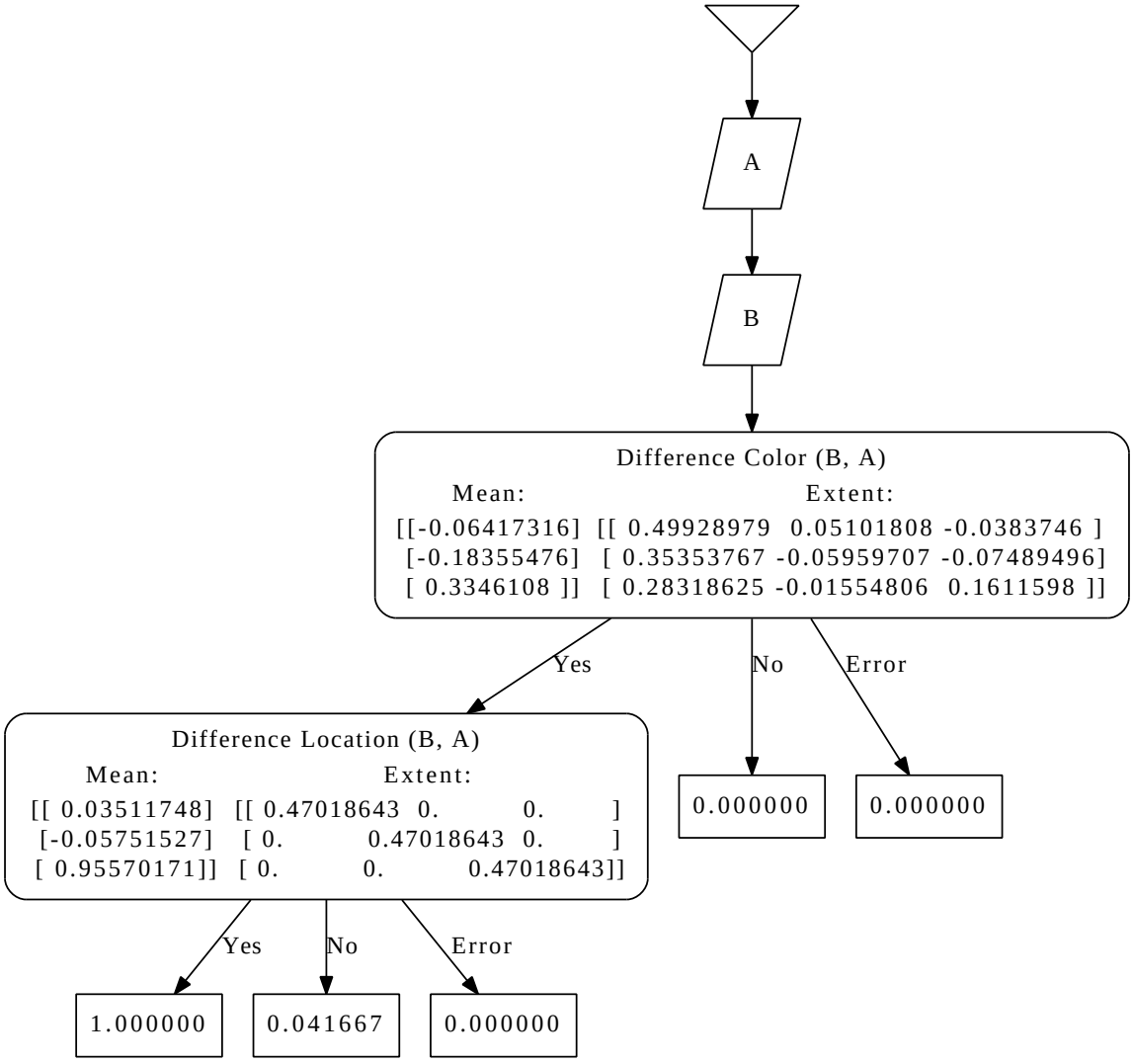


Figure 7.2: Typical *Blue Above Green* tree at two distractors and no corruption.

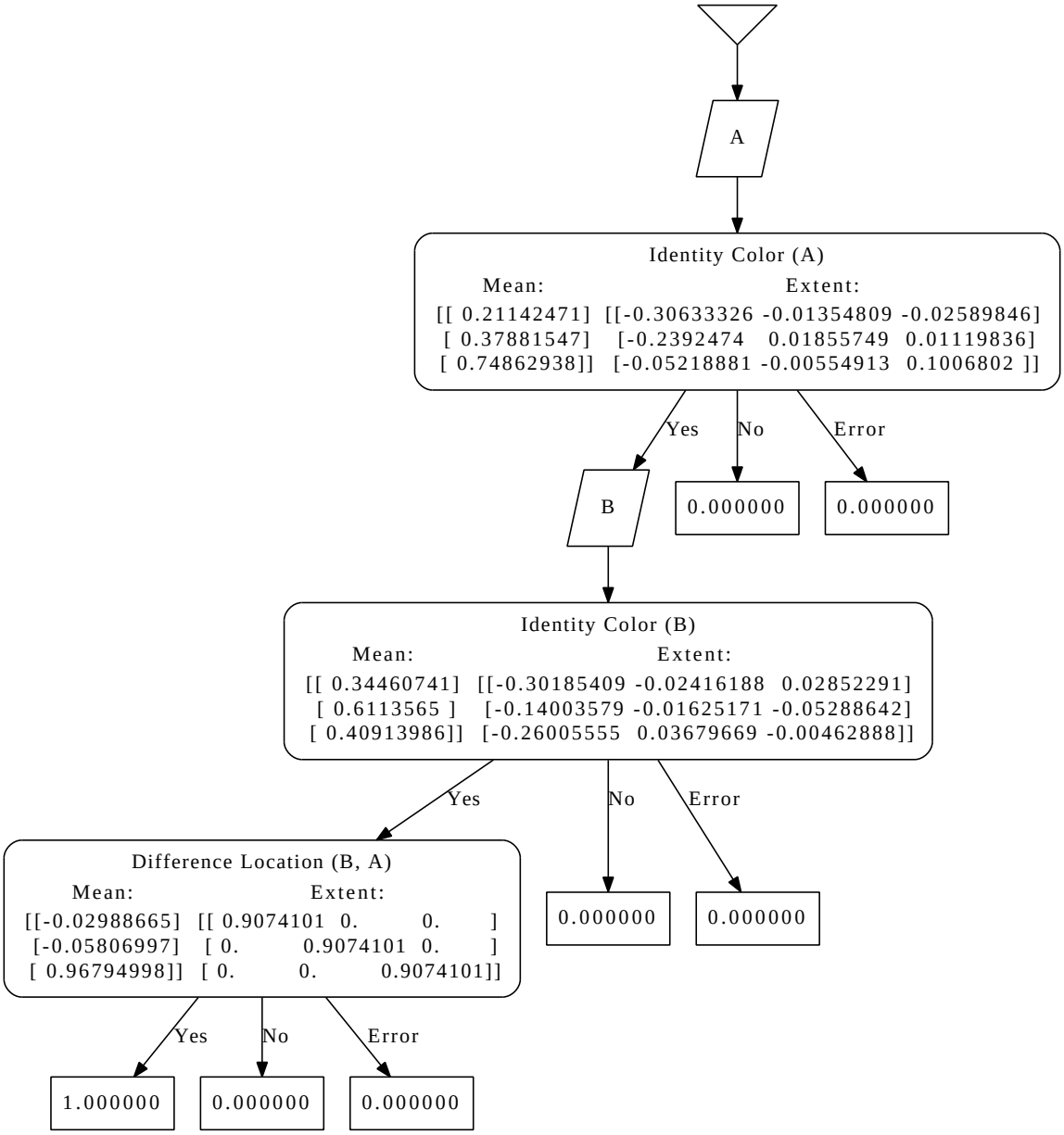


Figure 7.3: Typical *Blue Above Green* tree at five distractors and no corruption.

loss of classification performance. On the contrary, it classifies the training data perfectly. As such, it appears that adding a small amount of noise (for this dataset), forced SMRF to get more specific with its trees (at the cost of a larger tree), and thus improved classification performance slightly over the two distractor tree.

Figure 7.4 shows a typical tree learned for *Blue above Green* at ten distractors and

no corruption. This tree is almost the same as the ten distractor tree in Figure 7.4, except that this time, the tree asks the "green" question before the "blue" question. The question nodes are also different, in that they define tighter boundaries in their respective metric spaces, in order to prune out the larger number of non-target-conceptual instantiation sequences. However, the striking feature of this tree is that, even with the number of distractors doubled from five to ten, it still classifies the training data perfectly. As such, it appears that SMRF is very much robust to the addition of distractor objects.

Looking now at the other dimension of noise, Figure 7.5 shows a typical tree learned for *Blue above Green* at zero distractors and 6% corruption. This tree is structurally similar to the no noise tree in Figure 7.1, in that it asks a "blue"-"green" difference color question, followed by an "above" question. However, despite the corruption, there are no false positives in the leftmost leaf. On the other hand, about half of the examples sorted down the No branch of the *Difference Location* question appear to be false negatives, and there are some more false negatives sorted down the No branch of the *Difference Color* question.

Figure 7.6 shows a typical tree learned for *Blue above Green* at zero distractors and 12% corruption. This tree is almost the same as the 6% tree in Figure 7.5, as it asks a "blue"-"green" difference color question, followed by an "above" question. Unlike the 6% tree, the leftmost leaf is dilluted somewhat, as roughly 6% of the examples sorted there are false positives. As with the previous tree, there are also false negatives down the No branches of the two question nodes. However, given that over 10% of the example labels are incorrect, this tree still seems to represent the concept in a reasonable manner.

Figure 7.7 shows a typical tree learned for *Blue above Green* at zero distractors and 18% corruption. There is a striking difference between this tree and the 12% tree, in that this tree only has one question node. This tree asks a familiar first question:

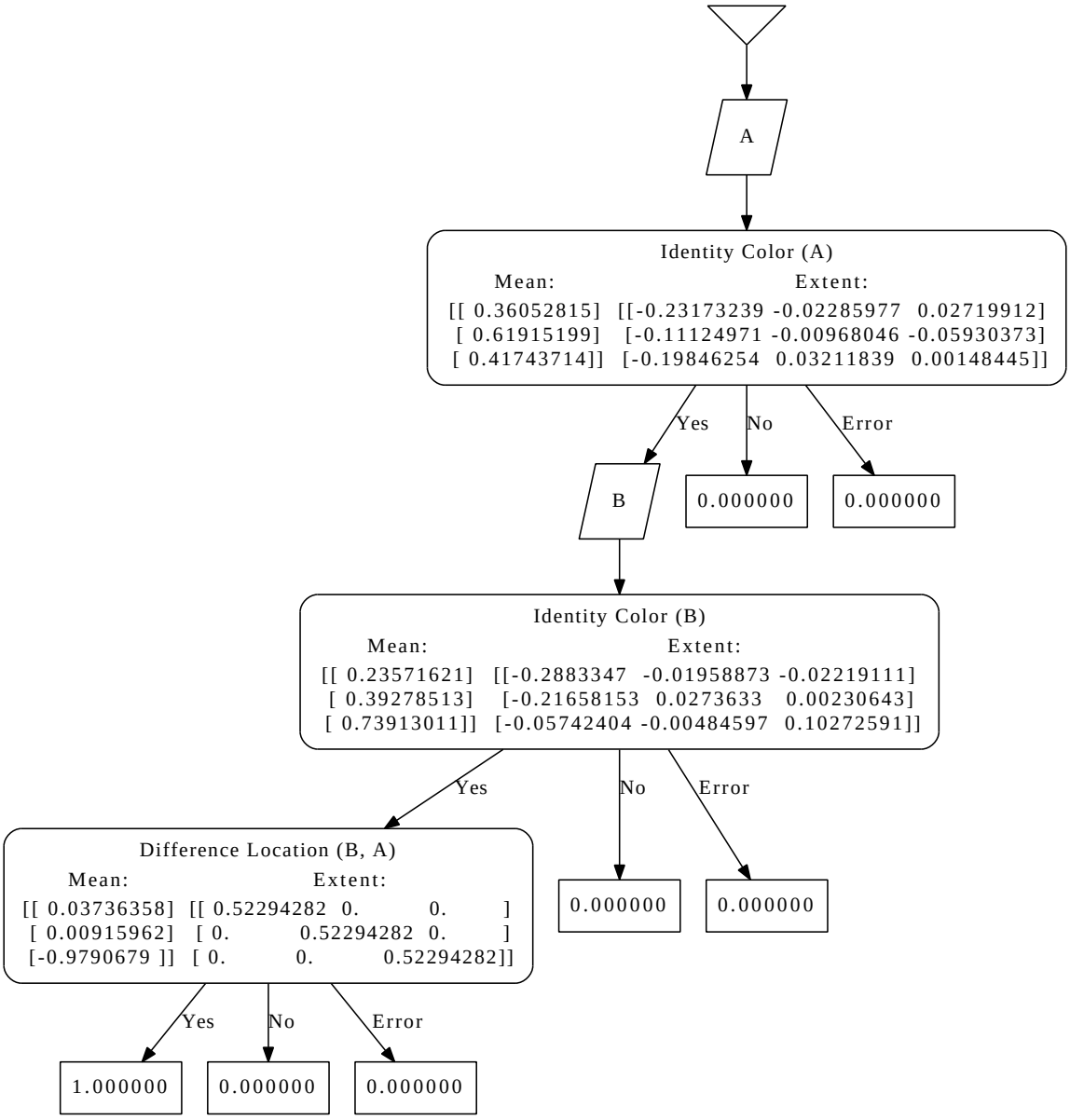


Figure 7.4: Typical *Blue Above Green* tree at ten distractors and no corruption.

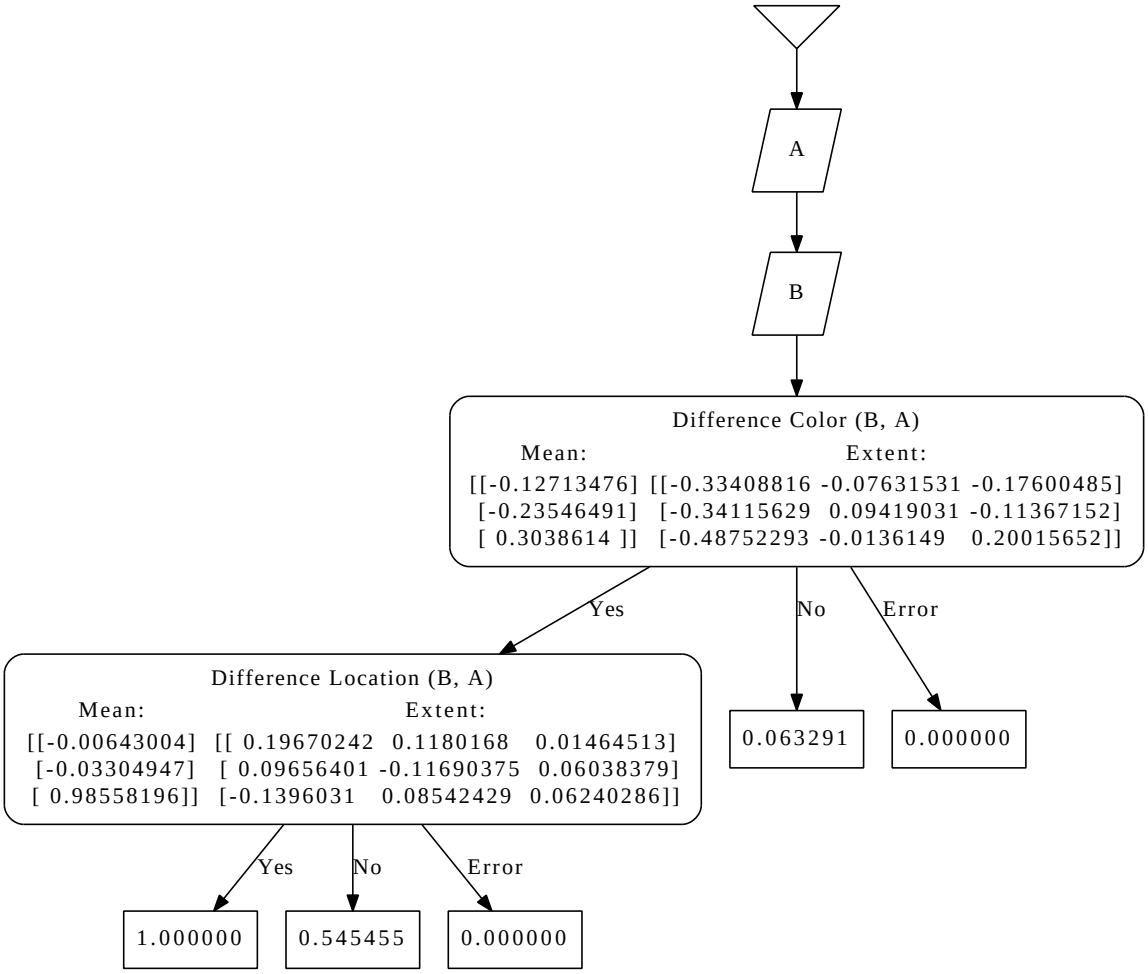


Figure 7.5: Typical *Blue Above Green* tree at zero distractors and 6% corruption.

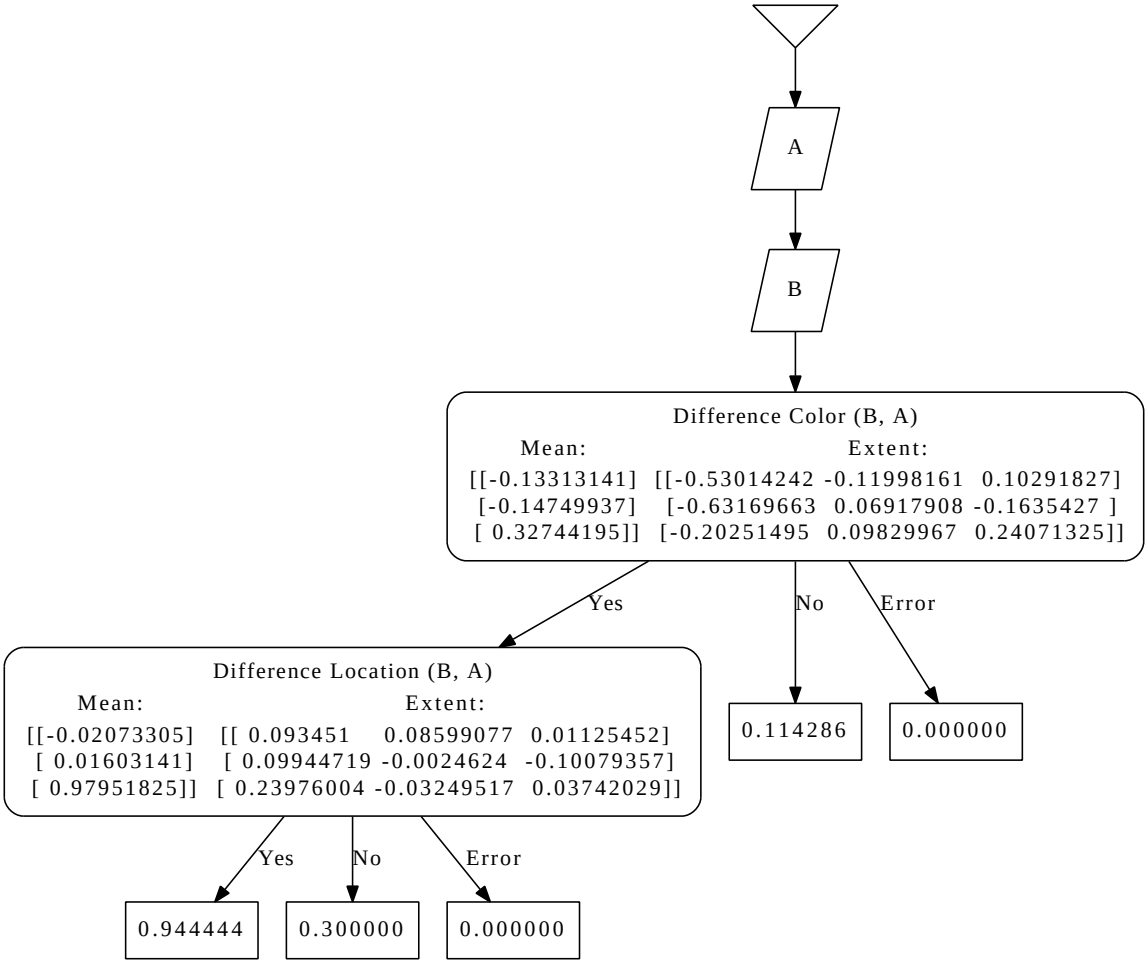


Figure 7.6: Typical *Blue Above Green* tree at zero distractors and 12% corruption.

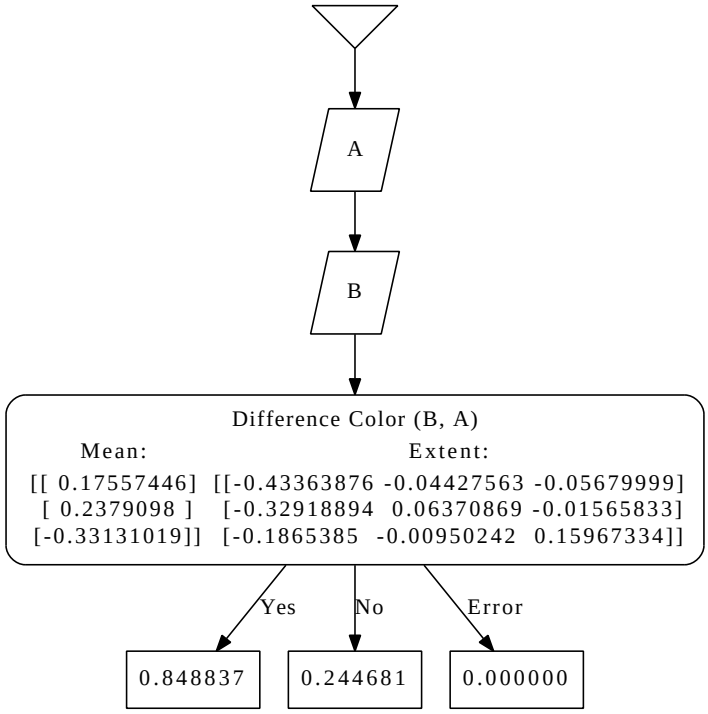


Figure 7.7: Typical *Blue Above Green* tree at zero distractors and 18% corruption.

do the two objects exhibit the "green"-"blue" color difference? However, no further questions are asked beyond this point. One will notice that the leaf nodes of this tree are substantially more dilluted than in the previous case, with No branch leaf containing nearly 25% false negatives, and the Yes branch leaf containing nearly 15% false positives. Given this large dillution at the leaf nodes, combined with the large number of corrupted labels (nearly a fifth!), SMRF is unable to produce any further question nodes that significantly improve sorting. What this shows is that SMRF gets more conservative as more errors are introduced into the training set labels. Rather than overfitting the data with more complex models, SMRF is content to learn a simple model and explain the features of the data that it can.

Notice also that while adding distractors has the capacity to force SMRF to learn a larger and more expressive tree, adding corruption tends to have the oppositive effect. Adding corruption never seems to increase tree size, though it does definitely decrease it as the corruption percentage nears 18%.

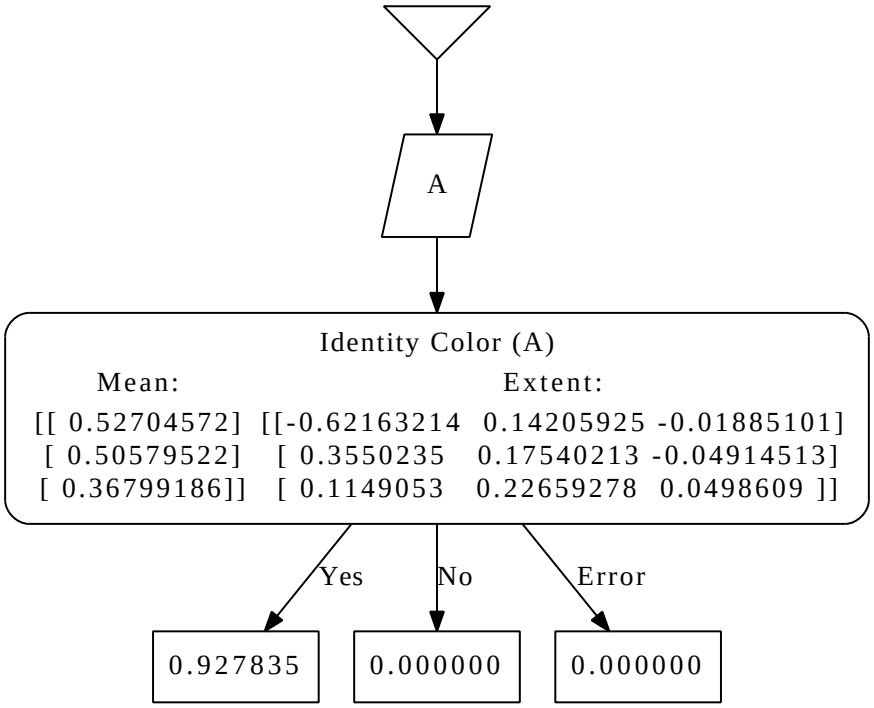


Figure 7.8: Typical *Red or Green* tree at zero distractors and no corruption.

Looking now at a different problem set type, Figure 7.8 shows a typical tree learned for *Red or Green* at zero distractors and no corruption. One might expect to see a tree with two questions, one for "red" and one for "green." However, since there is only one object in the scene with no additional noise, SMRF can ask a single identity color question that covers both the "red" and "green" areas of the RGB space. Of course, such a model will also cover areas in between the "red" and "green" regions. And as such, about 7% of the examples sorted into the Yes branch leaf are false positives. However, to the learning algorithm, this small amount of classificaion inaccuracy is preferable to learning a tree with two question nodes. The bias towards producing small trees sometimes forces SMRF to come up "creative" solutions to problems.

However, when a few distractors are added, the additional classification inaccuracy becomes large enough to force SMRF to include a second question node. Figure 7.9 shows a typical tree learned for *Red or Green* at two distractors and no corruption. This tree classifies the data perfectly, and exhibits disjunctive structure, with "green"

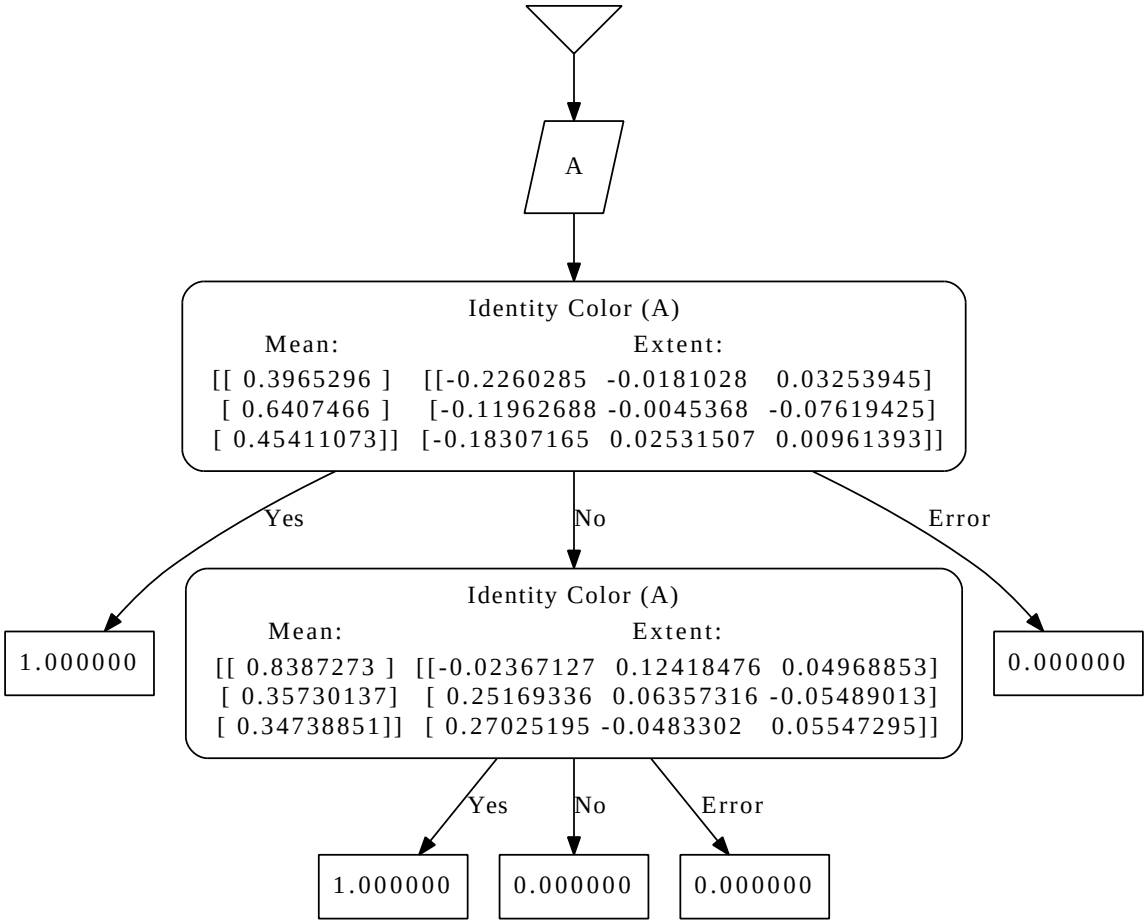


Figure 7.9: Typical *Red or Green* tree at two distractors and no corruption.

examples being sorted down the Yes branch of the first question node, "red" examples being sorted down the Yes branch of the second question node, and examples that are neither "red" nor "green" being sorted down the No branch of the second question node. As such, while purely conjunctive concepts tend to produce trees that are expanded down the Yes branches, purely disjunctive concepts tend to produce trees that are expanded down the No branches.

Looking at the third problem set type, Figure 7.10 shows a typical tree learned for *Red above Green or Blue above Yellow* at zero distractors and no corruption. This tree is a good example of how SMRF tries to find the smallest possible tree to reasonably represent a concept. This concept, if fully expressed, would require five question

nodes. However, this tree only includes three. Likewise, the tree displays Yes branch expansion, which is typical of a purely conjunctive concept, instead of the No branch expansion which one might expect to see out of a disjunctive concept. As it turns out, SMRF is actually being very "clever" in the way that it represents the concept, and doing so in a way that reduces the number of question nodes. The first question node attempts to capture "blue" and "green." Likewise, the third question node attempts to capture both "red" and "yellow." These question nodes will pick up colors that are "in between" the two intended regions, but the resulting classification inaccuracy is not strong enough to force SMRF to build a larger tree. Since "blue" is the high object in one disjunct, and "green" is the low object in the other disjunct, SMRF approximates the "above" relationship in terms of distance, and thus avoids having the create separate question nodes for when the object matched by the "blue"-"green" color model is either the high object or the low object. Of course, using distance location allows for other relations like "besides" to satisfy the model, in addition to "above." But as for the color questions, the resulting classification inaccuracy is not enough to force SMRF to learn a larger and more complex tree. As it is, the tree has no false negatives, and only 10% of the examples sorted into the leftmost leaf are false positives.

Adding distractors to more complex questions has the effect of causing SMRF to learn more conservative trees, similar to the effect of corrupting a relatively large percentage of the training set example labels. This effect can be seen in Figure 7.11, which shows a typical tree learned for *Red above Green or Blue above Yellow* at five distractors and no corruption. Unlike the zero distractor tree, this tree has only two questions, not three. The first question in this tree identifies a pair of objects that exhibit the "above" relationship. The second question identifies the color of the low object as being either green or yellow. Once again, this *Identity Color* model also captures objects whose color is located in RGB space "between" the "green" and

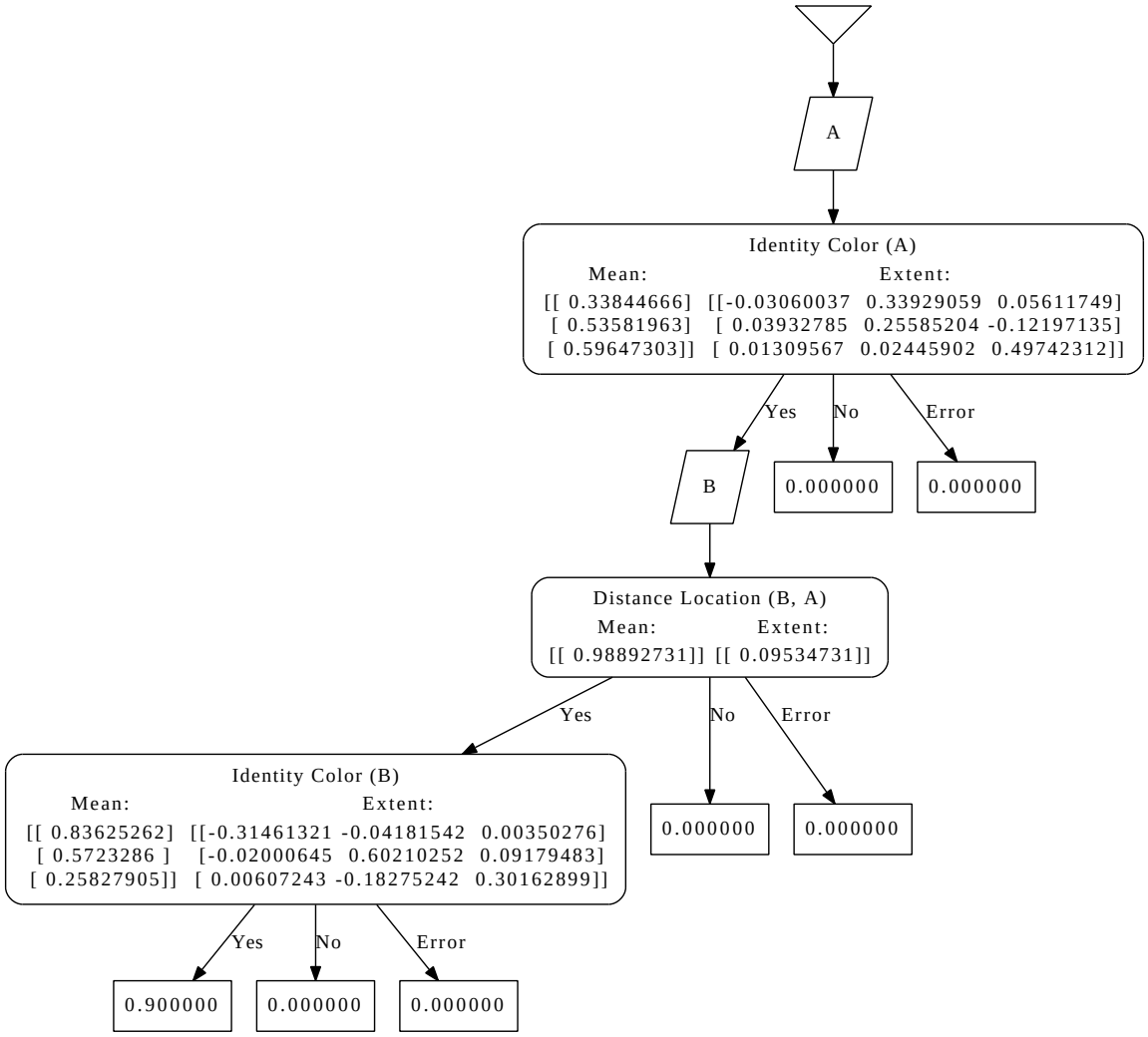


Figure 7.10: Typical *Red above Green or Blue above Yellow* tree at zero distractors and no corruption.

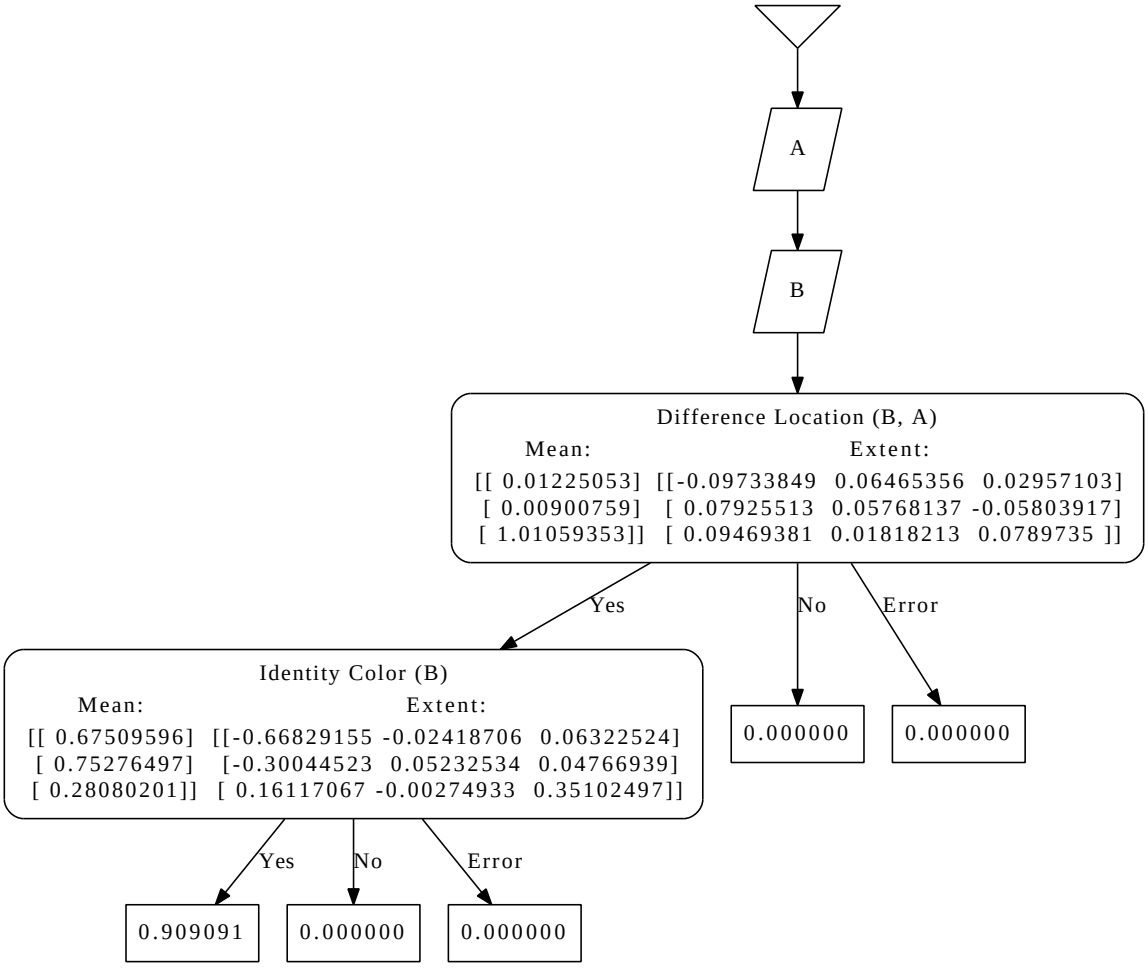


Figure 7.11: Typical *Red above Green or Blue above Yellow* tree at five distractors and no corruption.

"yellow" regions, and as such, about 9% of the examples sorted down the leftmost leaf are false positives. However, like the zero distractor tree, there are no false negatives down the other branches. As such, SMRF is able to learn compact representations that model complex target concepts to a reasonable degree of accuracy, even with the addition of distractor objects.

As such, these figures show that the SMRF learning algorithm has the capacity to learn interesting trees in a number of circumstances, to respresent various kinds of concepts, as well as to account for added numbers of distractors and the corruption of the labels in the training set. With these figures providing an intuition as to how

Factors	SS	df	MS	F	p-value
Problem Set	35.171	6	5.862	166.335	**~0**
Distractors	6.331	3	2.110	59.881	**~0**
Corruption	16.490	6	2.748	77.986	**~0**
Problem Set × Distractors	8.783	18	0.488	13.846	**~0**
Problem Set × Corruption	2.232	36	0.062	1.760	**0.004**
Distractors × Corruption	1.968	18	0.109	3.102	**~0**
Problem Set × Distractors × Corruption	4.056	108	0.038	1.066	0.309
Error	62.165	1764	0.035		
Total	137.196	1959			

Table 7.1: SMRF performance ANOVA (at 0-10 distractors).

SMRF works in various cases, I now turn to an in-depth analysis performed over all of the cases, using 3-way ANOVA and *post hoc* statistical tests.

ANOVA and *Post Hoc* Tests

There are three factors involved in determining the shape of the main experiment, when only the SMRF algorithm is considered: the data set (or *problem set*), the number of distractors present, and the percentage of examples corrupted. To gain an understanding of which factors (and which combinations of factors) have a significant effect on classification performance, a 3-way analysis of variance (ANOVA) was performed on these PSS data. The results of this ANOVA are given in Table 7.1.

In Table 7.1, significant p-values (at significance level $\alpha = 0.05$) are displayed using **boldface text**. Likewise, small p-values that are greater than zero, yet less than 10^{-3} are written as **~0**. Otherwise, p-values are written to four significant digits.

One would expect each of the individual factors to have a significant effect, and this is indeed the case. Each of the three two-factor combinations also has a significant effect, through the three-factor combination does not.

For each of the individual factors, *post hoc* tests were performed to determine which combinations of values were significant. The Bonferroni test was used (at significance level $\alpha = 0.05$), as it corrects for possible spurious significant results that might arise as a result of performing multiple comparisons. The Bonferroni test is a version of Fisher's Least Significance Difference test, where the resulting p-value is multiplied by the number of comparisons performed. In all *post hoc* test tables in this document, the adjusted p-values are displayed, and as such may exhibit values greater than 1.

First, the *post hoc* tests for the problem set factor are given in Table 7.2. These results can be interpreted as a measure of which problems are more difficult for SMRF than others. In particular, four concepts seem to easier for SMRF than the others: *Blue above Green (BaG)*, *Red or Green (R / G)*, *Blue above Green or Green above Blue (BaG / GaB)*, and *Right Red right of Left Green [(RtR)ro(LtG)]*. Both *BaG* and *(RtR)ro(LtG)* are purely conjunctive, and are the only such problem sets in this experiment. As disjunctive concepts have the effect of splitting the problem into multiple problems to be solved with fewer examples, it is not surprising that SMRF finds these purely conjunctive concepts to be easier than others. On the other hand, *R / G* is a purely disjunctive concept. However, it is a very simple concept, which appears to compensate for its disjunctive character (in terms of difficulty). The last "easy" concept for SMRF is *(BaG / GaB)*, which is a mixed disjunctive-conjunctive concept. However, both disjunctive clauses have the same combination of predicates, with the exception of the order of the parameters of the *above* predicate. As such, SMRF is able to take advantage of the shared information between disjuncts, which reduces the difficulty of the problem.

On the other hand, the other three concepts are harder for SMRF. This can be seen both in their mean values, as well as their small p-values when compared to any of the easy problems.

		Bonferroni Test	
Comparison	Means	F	p-value
BaG/GaB × R/G	0.819 × 0.831	0.657	17.549
BaG/GaB × RaG/BaY	0.819 × 0.607	178.186	**~0**
BaG/GaB × RaG/GaB	0.819 × 0.707	49.326	**~0**
BaG/GaB × (RtR)ro(LtG)	0.819 × 0.775	7.542	0.255
BaG/GaB × R/G/B/Y	0.819 × 0.448	544.907	**~0**
BaG/GaB × BaG	0.819 × 0.837	1.318	10.546
R/G × RaG/BaY	0.831 × 0.607	200.478	**~0**
R/G × RaG/GaB	0.831 × 0.707	61.365	**~0**
R/G × (RtR)ro(LtG)	0.831 × 0.775	12.650	**0.016**
R/G × R/G/B/Y	0.831 × 0.448	583.397	**~0**
R/G × BaG	0.831 × 0.837	0.114	30.898
RaG/BaY × RaG/GaB	0.607 × 0.707	40.011	**~0**
RaG/BaY × (RtR)ro(LtG)	0.607 × 0.775	112.410	**~0**
RaG/BaY × R/G/B/Y	0.607 × 0.448	99.892	**~0**
RaG/BaY × BaG	0.607 × 0.837	210.153	**~0**
RaG/GaB × (RtR)ro(LtG)	0.707 × 0.775	18.292	**~0**
RaG/GaB × R/G/B/Y	0.707 × 0.448	266.343	**~0**
RaG/GaB × BaG	0.707 × 0.837	66.769	**~0**
(RtR)ro(LtG) × R/G/B/Y	0.775 × 0.448	424.235	**~0**
(RtR)ro(LtG) × BaG	0.775 × 0.837	15.166	**0.004**
R/G/B/Y × BaG	0.448 × 0.837	599.822	**~0**

Table 7.2: Post hoc tests on SMRF performance for factor Problem Set (at 0-10 distractors)

		Bonferroni Test	
Comparison	Means	F	p-value
5 × 0	0.689 × 0.797	81.335	**~0**
5 × 10	0.689 × 0.645	13.260	**0.003**
5 × 2	0.689 × 0.741	18.828	**~0**
0 × 10	0.797 × 0.645	160.277	**~0**
0 × 2	0.797 × 0.741	21.897	**~0**
10 × 2	0.645 × 0.741	63.690	**~0**

Table 7.3: Post hoc tests on SMRF performance for factor Distractors (at 0-10 distractors)

Second, the *post hoc* tests for the distractors factor are given in Table 7.3. Each of the levels is significantly different than the others. This serves to demonstrate that the addition of distractors has a meaningful effect upon learning difficulty and algorithm performance.

Last, the *post hoc* tests for the corruption factor are given in Table 7.4. Not all level comparisons are statistically significant. However, all level comparisons involving a difference of 6% or greater are significant, with the exception of 0% × 6%. This serves to demonstrate that the addition of label corruption has a meaningful effect upon learning difficulty and algorithm performance.

7.2.2 High-Level Aggregate Results

Classification performance summaries for each algorithm, aggregated over all problem sets, are given in Figures 7.12- 7.14. The ordering of these figures is based upon the order in which the algorithms are listed in Section 7.1.3. These figures are grouped according to algorithm type. The tree-based algorithms are given first, followed by the non-SVM-based MIL algorithms, followed by the SVM-based MIL algorithms. These figures show the mean classification performance (in terms of PSS) for each algorithm across all datasets, across the range of possible distractor and corruption percentage values. Each figure is a heat map, with the number of distractors on the vertical axis

		Bonferroni Test	
Comparison	Means	F	p-value
0 × 3	0.819 × 0.820	0.005	39.739
0 × 6	0.819 × 0.771	9.099	0.109
0 × 9	0.819 × 0.736	27.273	**~0**
0 × 12	0.819 × 0.699	56.866	**~0**
0 × 15	0.819 × 0.618	160.043	**~0**
0 × 18	0.819 × 0.560	265.608	**~0**
3 × 6	0.820 × 0.771	9.511	0.087
3 × 9	0.820 × 0.736	27.983	**~0**
3 × 12	0.820 × 0.699	57.889	**~0**
3 × 15	0.820 × 0.618	161.756	**~0**
3 × 18	0.820 × 0.560	267.814	**~0**
6 × 9	0.771 × 0.736	4.867	1.155
6 × 12	0.771 × 0.699	20.472	**~0**
6 × 15	0.771 × 0.618	92.822	**~0**
6 × 18	0.771 × 0.560	176.388	**~0**
9 × 12	0.736 × 0.699	5.376	0.862
9 × 15	0.736 × 0.618	55.181	**~0**
9 × 18	0.736 × 0.560	122.658	**~0**
12 × 15	0.699 × 0.618	26.111	**~0**
12 × 18	0.699 × 0.560	76.677	**~0**
15 × 18	0.618 × 0.560	13.298	**0.011**

Table 7.4: Post hoc tests on SMRF performance for factor Corruption (at 0-10 distractors)

and the corruption percentage on the horizontal axis. Each cell in the heat map has a specific color, which corresponds to the value of mean PSS performance across all datasets for a given number of distractors and a given corruption percentage. The color values are given in a scale to the right of the figure, where the highest values start at the color red, and then follow the natural color spectrum to violet, which represents the smallest values. The color scale is the same for each figure, with the maximum and minimum values being selected as the global max and min, respectively, across all of the problem sets. For these figures, the maximum mean PSS value is 0.86, and the minimum value is -0.04.

In each cell of the heat map, there are two numbers. The first (top) number denotes the mean PSS value for this cell. The second (bottom) number, which is surrounded by brackets, is the standard deviation of the individual PSS values in this cell. The first number can be used to learn the precise mean value, and the second number can be used to gain a sense of the variation present within a cell.

Additionally, for each comparison algorithm, each row and column may be marked with an asterisk. Such markings are present when the aggregate row (or column) results are significantly different than the corresponding SMRF results. The Bonferroni test is used for testing significance, with aggregate $\alpha = 0.05$.

Figure 7.12a summarizes the classification performance of SMRF for this experiment. As one can see, the classification performance is quite good, with a mean PSS of 0.85 at 0 distractors and 0% corruption. SMRF is remarkably robust to distractors, maintaining a mean PSS of greater than 0.8 for all distractor values with no corruption. Likewise, SMRF is robust to corruption, maintaining a mean PSS of 0.8 or greater through 12% corruption at 0 distractors. Performance falls off slightly to around 0.7 PSS for corruption percentages of 15% and 18%. SMRF also seems reasonably robust to the combination of distractors and corruption, only falling below 0.7 PSS for the combinations of the higher values of distractors and corruption per-

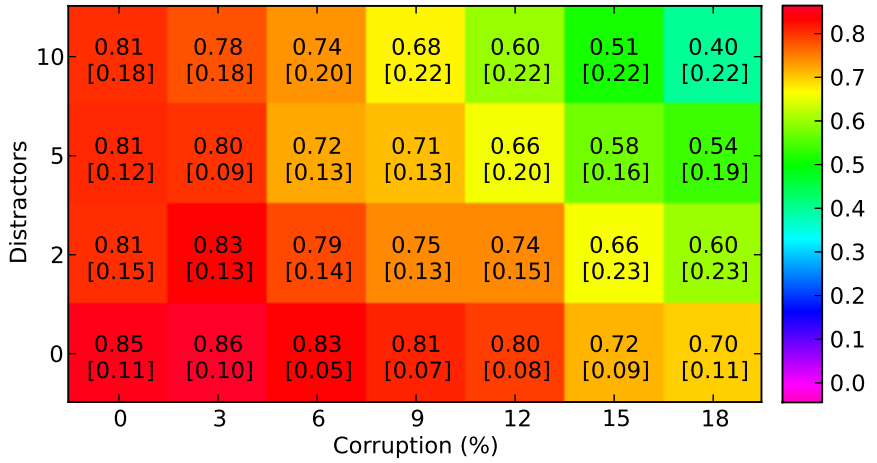
centage. As, such, SMRF appears to do quite well across a range of datasets (some of which are quite difficult) for substantial amounts of added noise.

Figure 7.12b summarizes the classification performance of TILDE for this experiment. TILDE appears to be reasonably robust to corruption, losing only 0.16 PSS when full corruption is added at 0 distractors. In comparison, SMRF also lost only 0.15 PSS in the same context. As such, TILDE appears to handle corruption, on its own, about as well as SMRF does well.

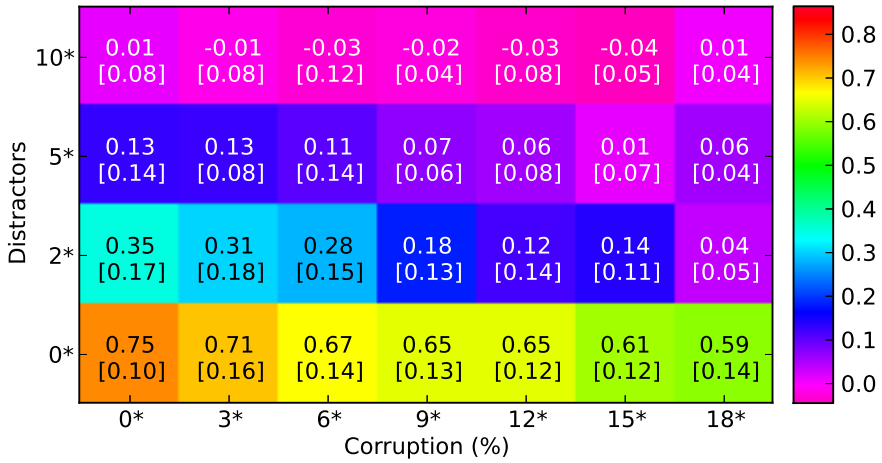
However, TILDE is *not* robust to the addition of distractor objects. Whereas SMRF stays relatively even through the addition of distractors, losing only 0.04 PSS, TILDE loses 0.4 PSS with the addition of the first two distractors. With no corruption, TILDE is at roughly random performance (0.01 mean PSS) at 10 distractors. Moreover, because TILDE handles distractors so poorly, it does not handle the combination of distractors and corruption well, either. SMRF performs significantly better than TILDE at all distractor levels and all corruption levels, as each row and column is significant.

Figure 7.12c summarizes the classification performance of IAPR for this experiment. With no added noise, IAPR performs very poorly, with a mean PSS of 0.06 and standard deviation 0.08, which indicates practically random classification. IAPR constructs rectangles along single dimensions to find target-conceptual regions in the instance space. However, since each of these datasets includes color data that exhibit real-world covariances, finding rectangular splits in individual dimensions will likely not allow an algorithm to classify these examples with a high degree of accuracy, and this appears to be the case here. Since IAPR classifies randomly without noise, the addition of noise does not change performance in any substantial way. Classification performance is poor overall, and as such, SMRF performas statistically better overall, as each row and column is significant.

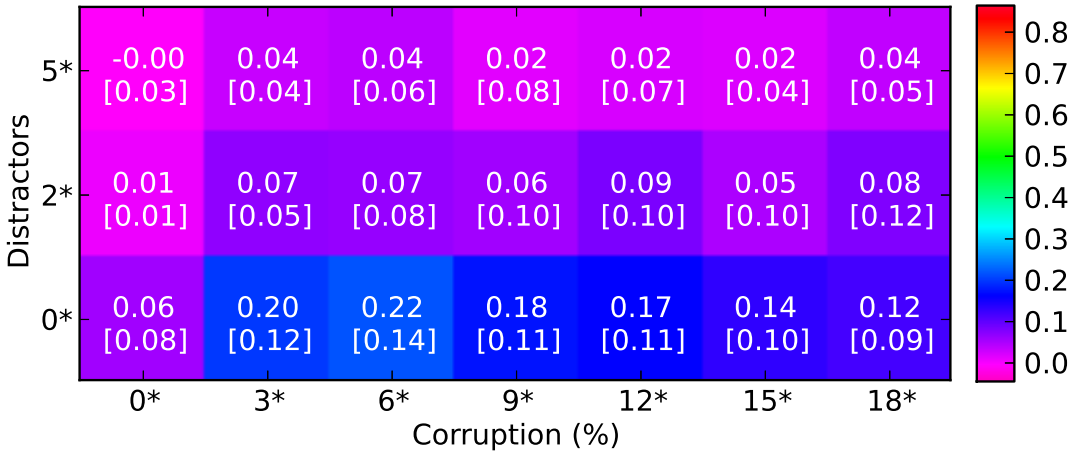
Figure 7.13a summarizes the classification performance of DD for this experiment.



(a) PSS results for SMRF.



(b) PSS results for TILDE.



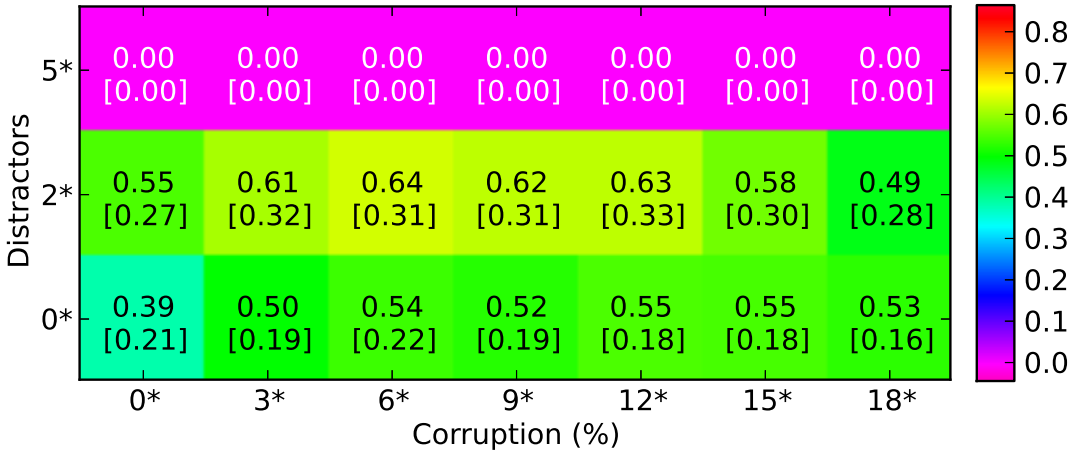
(c) PSS results for IAPR.

Figure 7.12: Mean PSS Summaries

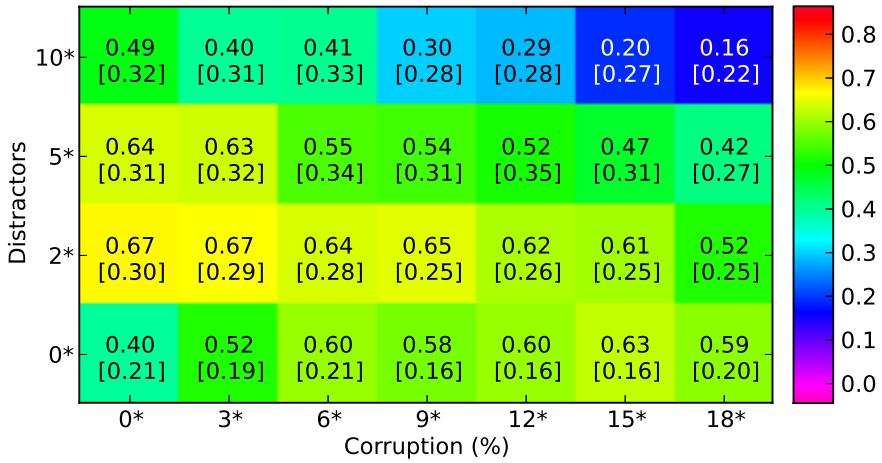
With no added noise, DD performs relatively poorly, with a mean PSS of 0.39. Adding a small amount of noise, however, causes performance to improve by roughly 0.10 PSS in the case of corruption, and 0.15 PSS in the case of distractors. DD appears to be nicely robust to corruption, as mean PSS does not fall below 0.5 for any corruption level greater than 3%. DD also appears to be robust to small numbers of distractors, as well as the combination of corruption and small numbers of distractors. However, at 5 distractors, it appears that something about the data caused the DD implementation to terminate its search early, and learn nothing. None of these 5 distractor results are the result of run termination due to errors. As such, this result might be due to a peculiarity present in this implementation of DD that might not be present in other implementations of the algorithm. SMRF performs significantly better than DD at all distractor levels and all corruption levels, as each row and column is significant.

Figure 7.13b summarizes the classification performance of EM-DD for this experiment. After SMRF, EM-DD performs the best of any of the algorithms used in this experiment. Like DD, EM-DD performs relatively poorly with no noise, at 0.4 mean PSS. However, like, DD, it does better with the addition of small amounts of noise, improving by 0.12 PSS in the case of corruption, and 0.27 PSS in the case of distractors. Judging from the other PSS summary figures, this appears to be a phenomenon limited to only DD and EM-DD. This phenomenon can be explained by the fact that features are constructed for these algorithms by enumerated all 2-permutations of the objects in each example, and constructing an instance[8] for each permutation. With two objects in an example and no distractors, this results in two instances per example. With two objects in an example and two distractors, this results in 12 instances per examples, which is a 6-fold increase in the amount of training data. The increase

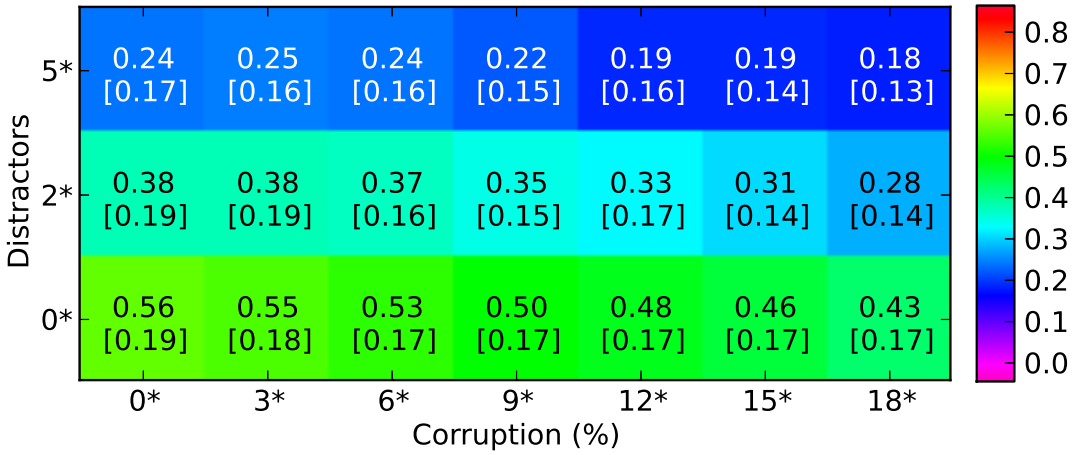
[8]In this context, the term *instance* diverges slightly from the sense in which it was defined in Chapter 3, in order to permit the discussion of standard MIL data representation in a natural manner. In this context, the term *instance* still denotes a member of a bag, but is represented as a vector of feature values, as opposed to an ordered list of objects.



(a) PSS results for DD.



(b) PSS results for EM-DD.



(c) PSS results for kNN.

Figure 7.13: Mean PSS Summaries

in performance with the addition of distractor objects (from zero to two distractors) can thus be explained as the result of the addition of distractors effectively providing these algorithms with roughly six times more training data, and thus making it easier for them to learn the target concept.

Like DD, EM-DD is robust to corruption at 0 distractors, not falling below the 0.52 PSS of 3% corruption. Like DD, it also is relatively robust to smaller numbers of distractors, as well as combinations of corruption and smaller numbers of distractors. However, going from 5 to 10 distractors, mean performance drops by 0.15 PSS, and EM-DD does not do well as 10 distractors with the addition of corruption. SMRF performs significantly better than EM-DD at all distractor levels and all corruption levels, as each row and column is significant.

Figure 7.13c summarizes the classification performance of Citation-kNN for this experiment. Citation-kNN performs moderately with no noise, with a mean PSS of 0.56. Unlike DD and EM-DD, Citation-kNN does not benefit from small amounts of noise. However, Citation-kNN appears to be reasonably robust to corruption, losing only around 0.10 PSS on average when corruption is increased from 0% to 18%. However, it is not very robust to distractors, losing around 0.15 PSS on average when the number of distractors is increased from 0 to 2, and from 2 to 5. SMRF performs significantly better than Citation-kNN at all distractor levels and all corruption levels, as each row and column is significant.

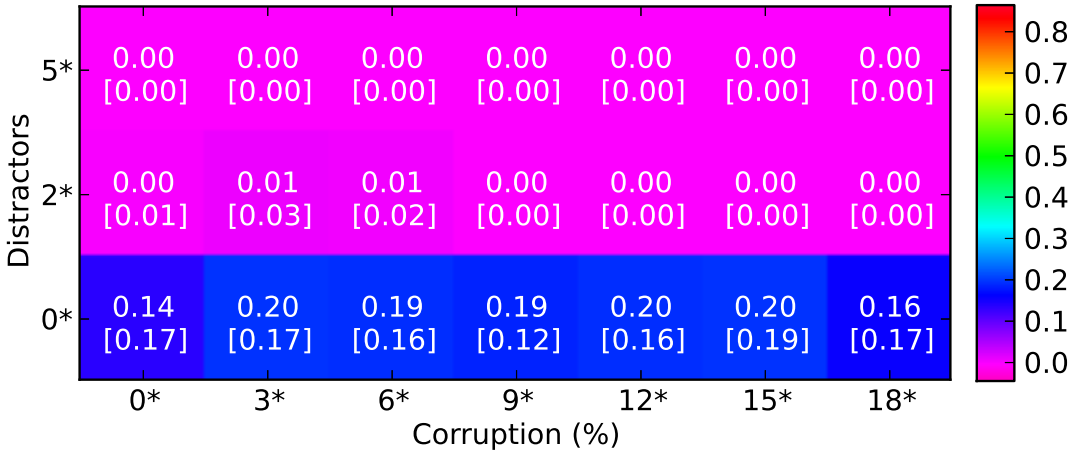
Figure 7.14a summarizes the classification performance of mi-SVM for this experiment. mi-SVM is the worst-performing SVM-based algorithm, and along with IAPR, one of the two worst-performing algorithms overall. With no noise, mi-SVM can only manage a mean PSS of 0.14. Adding distractors drives mi-SVM down to random classification. Since performance is at or close to random to begin with, the addition of corruption does not produce any meaningful effects. As performance is so poor, it is no surprise that each row and column are significant, with SMRF

outperforming MI-SVM at ever distractor and corruption level.

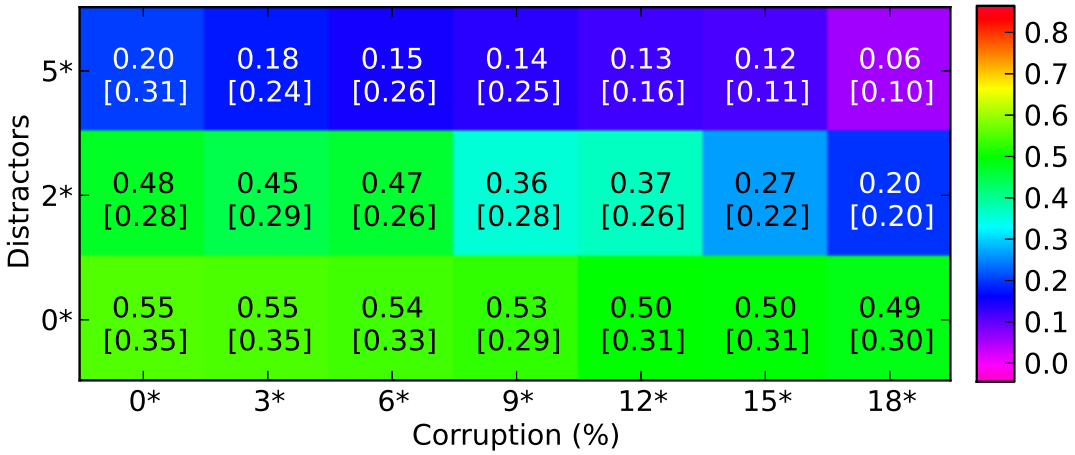
Figure 7.14b summarizes the classification performance of MI-SVM for this experiment. MI-SVM is the best-performing SVM algorithm in this experiment. With no noise, MI-SVM performs moderately, with a mean PSS of 0.55. However, the standard deviation is rather large, at 0.35. MI-SVM seems relatively robust to corruption by itself, losing only 0.06 PSS from 0% to 18% with no distractors. MI-SVM is somewhat robust to the addition of 2 distractors, but pretty much falls apart when 5 distractors are present. SMRF performs significantly better than MI-SVM at all distractor levels and all corruption levels, as each row and column is significant.

Figure 7.14c summarizes the classification performance of MICA for this experiment. With no noise present, MICA performs poorly, with a mean PSS of 0.33. MICA seems to be relatively robust to corruption, only losing 0.09 PSS as the corruption percentage is increased from 0% to 18%, at no distractors. However, MICA is not robust to distractors, as it loses 0.21 PSS at 2 distractors, and is down to random performance at 5 distractors. SMRF performs significantly better than MICA at all distractor levels and all corruption levels, as each row and column is significant.

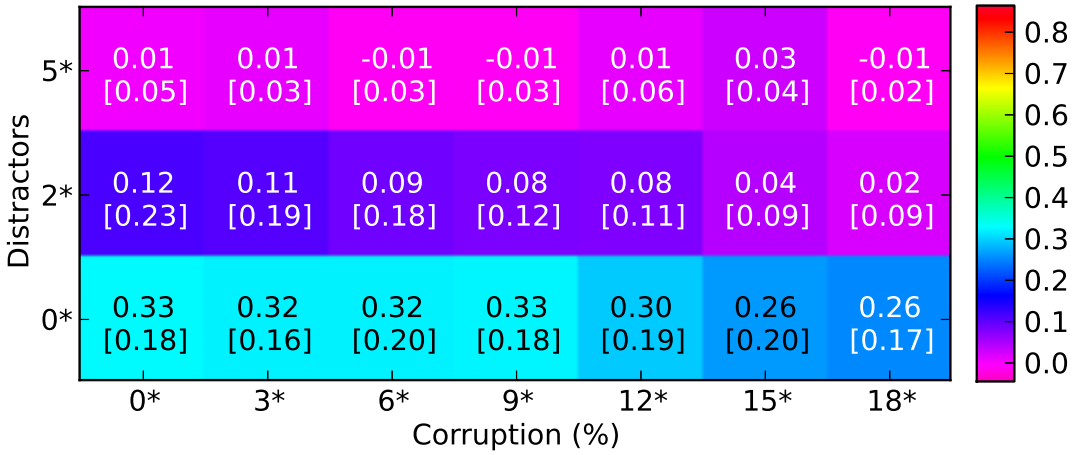
As such, from these figures, it is clear that SMRF is the best-performing algorithm overall, and that it is robust to both distractors and corruption, as well as being reasonably robust to the combination of the two. EM-DD is the next-best algorithm, followed by MI-SVM, Citation-kNN, and Tilde. Tilde is by far the best of this group at 0 distractors, but its performance is either worse than or comparable to the other algorithms of this group at 2 and 5 distractors. The other algorithms do not perform well overall, with the exception of DD, which performs comparably to EM-DD at 0 and 2 distractors, but completely fails to learn anything at 5 distractors.



(a) PSS results for mi-SVM.



(b) PSS results for MI-SVM.



(c) PSS results for MICA.

Figure 7.14: Mean PSS Summaries

7.2.3 Detailed Aggregate Results

There are four factors involved in determining the shape of the main experiment, when all algorithms are considered: the data set (or *problem set*), the algorithm, the number of distractors present, and the percentage of examples corrupted. To gain an understanding of which factors (and which combinations of factors) have a significant effect on classification performance, a 4-way analysis of variance (ANOVA) was performed on the PSS data for the main experiment, for 0-5 distractors. The results of this ANOVA are given in Table 7.5.

As with the SMRF-only ANOVA, one would expect that each of the individual factors would be significant, and this is indeed the case. Additionally, every combination of two factors is also significant, except for problem set $\times$ corruption percentage.

For each of the individual factors, *post hoc* tests were performed to determine which combinations of values were significant. The Bonferroni test was used at significance level $\alpha = 0.05$.

The primary result to take away from the 0-5 distractor analysis is that SMRF significantly outperforms all other algorithms. This can be seen in the results for the algorithm factor *post hoc* tests, given in Table 7.6. It should be noted that Table 7.6 does not contain all pairwise comparisons for the algorithm factor, but only those involving SMRF. The mean SMRF performance (across all other factors) is 0.742 PSS. In comparison, all other algorithms, with the exception of EM-DD, have a mean performance of less than 0.4 PSS. EM-DD performs better than the others, with a mean of 0.574 PSS. However, the mean SMRF performance is still significantly better than the mean performance of EM-DD. Even TILDE only has a mean PSS of 0.315. As such, it is clear that SMRF is a far superior algorithm on these problem sets than any of its competitors.

In addition, a 4-way ANOVA was performed on the PSS data for the main ex-

Factors	SS	df	MS	F	p-value
Problem Set	165.188	6	27.531	640.268	~**0**
Algorithm	597.317	8	74.665	1736.401	~**0**
Distractors	169.359	2	84.679	1969.305	~**0**
Corruption	12.197	6	2.033	47.275	~**0**
Problem Set × Algorithm	105.998	48	2.208	51.356	~**0**
Problem Set × Distractors	15.268	12	1.272	29.589	~**0**
Problem Set × Corruption	1.704	36	0.047	1.101	0.311
Algorithm × Distractors	121.157	16	7.572	176.103	~**0**
Algorithm × Corruption	12.533	48	0.261	6.072	~**0**
Distractors × Corruption	2.927	12	0.244	5.672	~**0**
Problem Set × Algorithm × Distractors	87.613	96	0.913	21.224	~**0**
Problem Set × Algorithm × Corruption	14.096	288	0.049	1.138	0.055
Problem Set × Distractors × Corruption	3.594	72	0.050	1.161	0.166
Algorithm × Distractors × Corruption	8.757	96	0.091	2.121	~**0**
Problem Set × Algorithm × Distractors × Corruption	23.572	576	0.041	0.952	0.787
Error	511.996	11907	0.043		
Total	1853.276	13229			

Table 7.5: Factorial experiment ANOVA (at 0-5 distractors).

		Bonferroni Test	
Comparison	Means	F	p-value
SMRF × TILDE	0.742×0.315	3122.240	~**0**
SMRF × kNN	0.742×0.354	2567.323	~**0**
SMRF × MI-SVM	0.742×0.344	2700.598	~**0**
SMRF × IAPR	0.742×0.080	7487.288	~**0**
SMRF × EM-DD	0.742×0.574	481.813	~**0**
SMRF × DD	0.742×0.366	2417.265	~**0**
SMRF × MICA	0.742×0.129	6412.968	~**0**
SMRF × mi-SVM	0.742×0.062	7913.376	~**0**

Table 7.6: Post hoc tests for factor Algorithm (at 0-5 distractors)

periment, for 0-10 distractors. This analysis only considers the subset of algorithms which were run at the 10 distractor level: SMRF, TILDE, and EM-DD. The results of this ANOVA are given in Table 7.7.

As expected, each of the four individual factors are significant. Additionally, all two factor combinations are significant, except for problem set × corruption percentage.

For each of the individual factors, *post hoc* tests were performed to determine which combinations of values were significant. The Bonferroni test was used at significance level $\alpha = 0.05$.

The primary result to take away from the 0-10 distractor analysis is that SMRF significantly outperforms both TILDE and EM-DD. This can be seen in the results for the algorithm factor *post hoc* tests, given in Table 7.8. It should be noted that Table 7.8 does not contain all pairwise comparisons for the algorithm factor, but only those involving SMRF. The mean SMRF performance (across all other factors) is 0.718 PSS. In contrast, the mean TILDE performance is 0.232 PSS. EM-DD does much better than TILDE, with a mean performance of 0.510 PSS. In both cases, SMRF performs significantly better, with p-value less than 10^{-3}.

Next, looking at the *post hoc* tests for the distractor factor, given in Table 7.9, one can see that each pairwise comparison is significant. As with the SMRF-only

Factors	SS	df	MS	F	p-value
Problem Set	102.982	6	17.164	340.844	~**0**
Algorithm	233.291	2	116.645	2316.388	~**0**
Distractors	98.941	3	32.980	654.935	~**0**
Corruption	21.692	6	3.615	71.795	~**0**
Problem Set × Algorithm	48.015	12	4.001	79.458	~**0**
Problem Set × Distractors	7.422	18	0.412	8.188	~**0**
Problem Set × Corruption	1.746	36	0.049	0.963	0.532
Algorithm × Distractors	64.572	6	10.762	213.715	~**0**
Algorithm × Corruption	3.545	12	0.295	5.866	~**0**
Distractors × Corruption	4.509	18	0.251	4.975	~**0**
Problem Set × Algorithm × Distractors	33.598	36	0.933	18.533	~**0**
Problem Set × Algorithm × Corruption	6.085	72	0.085	1.678	~**0**
Problem Set × Distractors × Corruption	6.381	108	0.059	1.173	0.108
Algorithm × Distractors × Corruption	8.867	36	0.246	4.891	~**0**
Problem Set × Algorithm × Distractors × Corruption	13.789	216	0.064	1.268	**0.006**
Error	266.487	5292	0.050		
Total	921.922	5879			

Table 7.7: Factorial experiment ANOVA (at 0-10 distractors).

		Bonferroni Test	
Comparison	Means	F	p-value
EM-DD × SMRF	0.510 × 0.718	835.867	~**0**
SMRF × TILDE	0.718 × 0.232	4599.465	~**0**

Table 7.8: Post hoc tests for factor Algorithm (at 0-10 distractors)

		Bonferroni Test	
Comparison	Means	F	p-value
5 × 0	0.436 × 0.672	808.485	**~0**
5 × 10	0.436 × 0.316	212.641	**~0**
5 × 2	0.436 × 0.523	108.527	**~0**
0 × 10	0.672 × 0.316	1850.383	**~0**
0 × 2	0.672 × 0.523	324.585	**~0**
10 × 2	0.316 × 0.523	624.992	**~0**

Table 7.9: Post hoc tests for factor Distractors (at 0-10 distractors)

analysis, this serves to show that the addition of distractors has a meaningful effect upon learning difficulty, even for other algorithms in addition to SMRF.

The *post hoc* tests for the corruption factor are given in Table 7.10. As with the SMRF-only analysis, not all level comparisons are statistically significant. However, as with the SMRF-only analysis, all level comparisons involving a difference of 6% or greater are significant, with the exception of 0% × 6%. This serves to demonstrate that the addition of label corruption has a meaningful effect upon learning difficulty, and thus upon algorithm performance.

Next, the *post hoc* tests for the problem set factor are given in Table 7.11. As with the SMRF-only analysis, these results can be interpreted as a measure of which problems are more difficult than others for SMRF, Tilde and EM-DD. As with the SMRF-only analysis, *Blue above Green (BaG)*, *Blue above Green or Green above Blue (BaG / GaB)*, and *Right Red right of Left Green [(RtR)ro(LtG)]* are the easier problems. This can be seen in the higher mean PSS, along with the statistically insignificant pairwise comparisons. The other four problem sets are harder for the group of three algorithms to solve. *Red or Green or Blue or Yellow (R / G / B / Y)* is by far the hardest, with a mean PSS of only 0.219. Surprisingly, *Red or Green (R / G)*, which was an "easy" problem for SMRF, is one of the "hard" problems for the group. Additionally, *Red above Green or Green above Blue (RaG / GaB)* and *Red above Green or Blue above Yellow (RaG / BaY)* are also among the harder

		Bonferroni Test	
Comparison	Means	F	p-value
0 × 3	0.560 × 0.552	0.490	20.321
0 × 6	0.560 × 0.526	9.595	0.082
0 × 9	0.560 × 0.491	39.378	~**0**
0 × 12	0.560 × 0.467	71.367	~**0**
0 × 15	0.560 × 0.424	153.984	~**0**
0 × 18	0.560 × 0.385	254.622	~**0**
3 × 6	0.552 × 0.526	5.748	0.695
3 × 9	0.552 × 0.491	31.081	~**0**
3 × 12	0.552 × 0.467	60.027	~**0**
3 × 15	0.552 × 0.424	137.098	~**0**
3 × 18	0.552 × 0.385	232.767	~**0**
6 × 9	0.526 × 0.491	10.097	0.063
6 × 12	0.526 × 0.467	28.626	~**0**
6 × 15	0.526 × 0.424	86.703	~**0**
6 × 18	0.526 × 0.385	165.362	~**0**
9 × 12	0.491 × 0.467	4.721	1.253
9 × 15	0.491 × 0.424	37.624	~**0**
9 × 18	0.491 × 0.385	93.736	~**0**
12 × 15	0.467 × 0.424	15.691	**0.003**
12 × 18	0.467 × 0.385	56.385	~**0**
15 × 18	0.424 × 0.385	12.587	**0.016**

Table 7.10: Post hoc tests for factor Corruption (at 0-10 distractors)

		Bonferroni Test	
Comparison	Means	F	p-value
BaG/GaB × R/G	0.596 × 0.467	138.592	~**0**
BaG/GaB × RaG/BaY	0.596 × 0.434	219.637	~**0**
BaG/GaB × RaG/GaB	0.596 × 0.454	167.804	~**0**
BaG/GaB × (RtR)ro(LtG)	0.596 × 0.631	10.045	0.064
BaG/GaB × R/G/B/Y	0.596 × 0.219	1182.242	~**0**
BaG/GaB × BaG	0.596 × 0.606	0.862	14.833
R/G × RaG/BaY	0.467 × 0.434	9.288	0.097
R/G × RaG/GaB	0.467 × 0.454	1.396	9.975
R/G × (RtR)ro(LtG)	0.467 × 0.631	223.261	~**0**
R/G × R/G/B/Y	0.467 × 0.219	511.268	~**0**
R/G × BaG	0.467 × 0.606	161.317	~**0**
RaG/BaY × RaG/GaB	0.434 × 0.454	3.483	2.606
RaG/BaY × (RtR)ro(LtG)	0.434 × 0.631	323.626	~**0**
RaG/BaY × R/G/B/Y	0.434 × 0.219	382.734	~**0**
RaG/BaY × BaG	0.434 × 0.606	248.022	~**0**
RaG/GaB × (RtR)ro(LtG)	0.454 × 0.631	259.962	~**0**
RaG/GaB × R/G/B/Y	0.454 × 0.219	459.238	~**0**
RaG/GaB × BaG	0.454 × 0.606	192.722	~**0**
(RtR)ro(LtG) × R/G/B/Y	0.631 × 0.219	1410.241	~**0**
(RtR)ro(LtG) × BaG	0.631 × 0.606	5.022	1.053
R/G/B/Y × BaG	0.219 × 0.606	1246.958	~**0**

Table 7.11: Post hoc tests for factor Problem Set (at 0-10 distractors)

problems to solve. These three problems sets are pairwise statistically insignificant among themselves, but are statistically significant when compared to any of the "easy" group. Likewise, *R / G / B / Y* is statistically significant when compared to any other problem set.

The order of problem set difficulty implied by Table 7.11 is reflected in the ordering of Figures 7.15 - 7.21, which are given in order from easiest to hardest. These figures show how much better (or occasionally, worse) SMRF performs relative to either Tilde or EM-DD, for each problem set, across the range of possible distractor and corruption percentage values. Each figure contains two subfigures, one for the difference in performance between SMRF and Tilde, and the other for the difference in performance between SMRF and EM-DD. Each subfigure is a heat map, with the

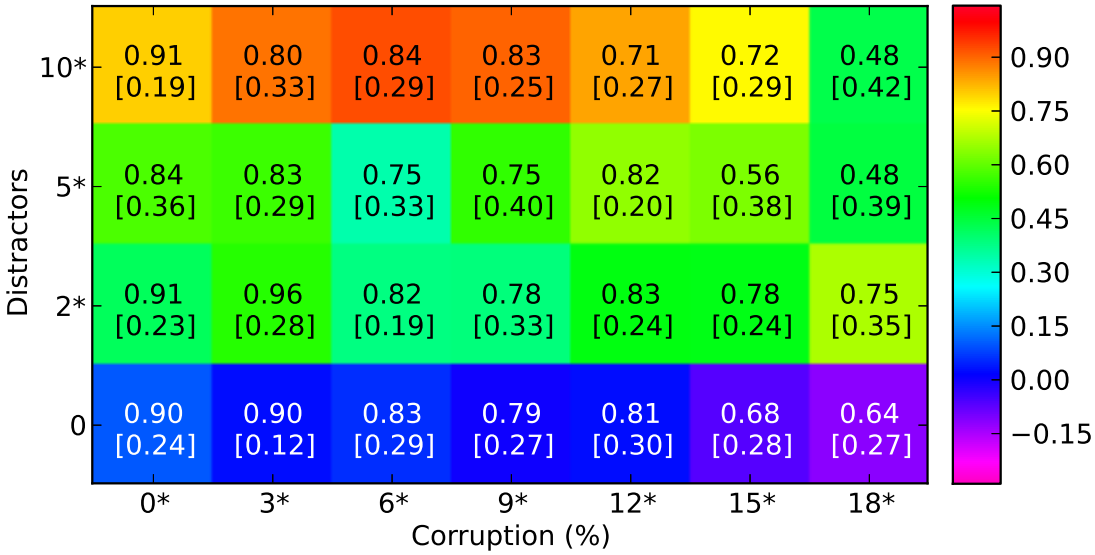
number of distractors on the vertical axis and the corruption percentage on the horizontal axis. Each cell in the heat map has a specific color, which corresponds to the value of mean SMRF PSS performance minus the value of mean TILDE or EM-DD PSS performance. For these figures, the maximum PSS difference is 1.04, and the minimum difference is -0.29.

In each cell of the heat map, there are two numbers. The first (top) number denotes the mean PSS of SMRF for this cell. The second (bottom) number, which is surrounded by brackets, is the standard deviation of the difference in PSS between SMRF and either TILDE or EM-DD, for this cell. The first number of the cell, together with the cell color, can be used to determine the mean performance of both SMRF and either TILDE or EM-DD, respectively, for that cell. Likewise, the second number of the cell can be used to gain a sense of how meaningful the difference between SMRF and either TILDE or EM-DD is.

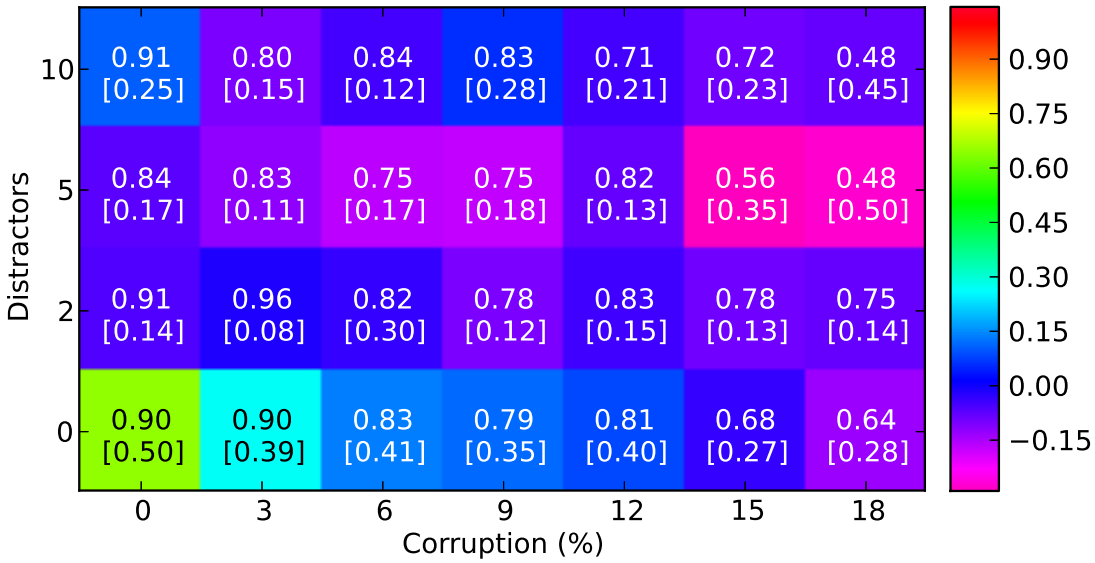
Additionally, each value on the vertical and horizontal axes may be marked with an asterisk. If a row or column is marked with an asterisk, then the difference in performance between SMRF and either TILDE or EM-DD, respectively, is significantly different than zero for the collective values in that row or column. The Bonferroni test is used here, with aggregate $\alpha = 0.05$. These row and column significance tests can be used to gain a sense of what trends are present in the data, particularly as the number of distractors and the corruption percentage change.

Each individual figure is discussed in the following pages, followed by a summary of the trends observed in these results.

Figure 7.15 shows the performance differences for the *Right Red right of Left Green* dataset. This is the easiest dataset in terms of mean PSS. As TILDE does not handle distractors well, the rows for 2, 5, and 10 distractors are all statistically significant. Likewise, each column is statistically significant, which is likely also a result of the fact that TILDE does not handle distractors well.



(a) SMRF - TILDE



(b) SMRF - EM-DD

Figure 7.15: Mean difference PSS for the *Right Red right of Left Green* dataset.

For EM-DD, no rows are significant. Likewise, no columns are significant. At 0 distractors EM-DD catches up with SMRF as the corruption percentage increases, and slightly outperforms SMRF at 18% corruption. At 2 and 10 distractors, performance is roughly equal between SMRF and EM-DD, with perhaps a slight edge to EM-DD. At 5 distractors, EM-DD seems to outperform SMRF overall, but not significantly so.

Figure 7.16 shows the performance differences for the *Blue above Green* dataset. As with the previous dataset, TILDE handles distractors poorly, and so all rows are significant except for 0 distractors, and all columns are significant.

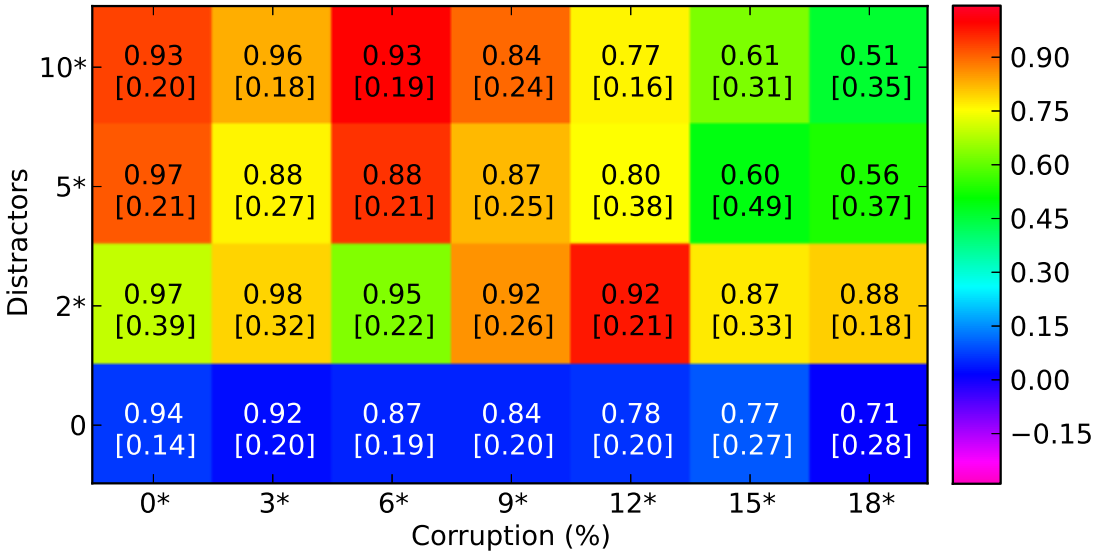
As this is problem is roughly equivalent to the previous, the performance of SMRF and EM-DD is comparable across all distractor and corruption values. As such, no rows or columns are significant for this problem set.

Figure 7.17 shows the performance differences for the *Blue above Green or Green above Blue* dataset. As with the previous two datasets, TILDE handles distractors poorly, and so all rows are significant except for 0 distractors, and all columns are significant.

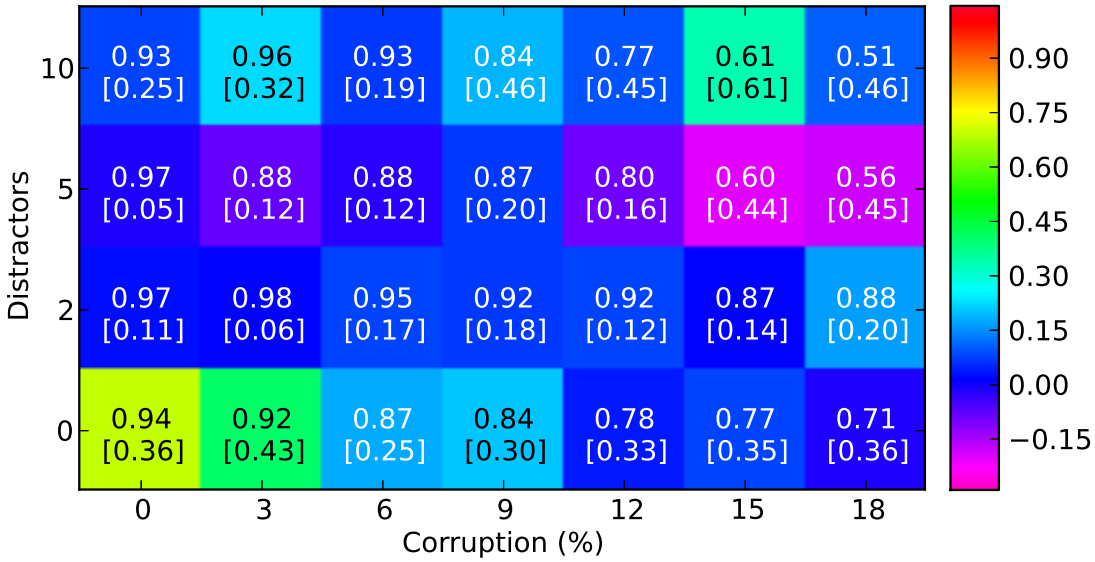
As this is a slightly harder problem, SMRF performs better relative to EM-DD than with the previous problem set. SMRF enjoys no significant advantage at 0, 2, or 5 distractors. However, at 10 distractors, SMRF enjoys a clear advantage with a significant row. For this problem set, SMRF enjoys an overall advantage at each corruption percentage level, as each column is significant.

Figure 7.18 shows the performance differences for the *Red above Green or Green above Blue* dataset. With respect to TILDE, all rows and columns are significant, as SMRF enjoys a clear all-around performance advantage. Likewise, with respect to EM-DD, SMRF also enjoys a clear all-around performance advantage, as columns are significant, and rows are significant except for 0 distractors.

Even though SMRF finds this problem set to be relatively easy, both TILDE and

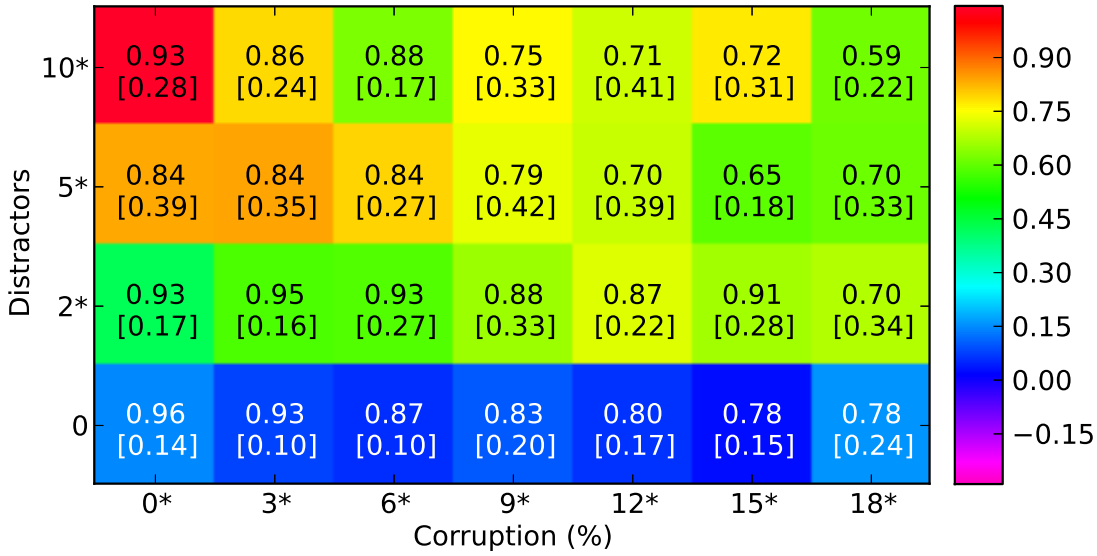


(a) SMRF - TILDE

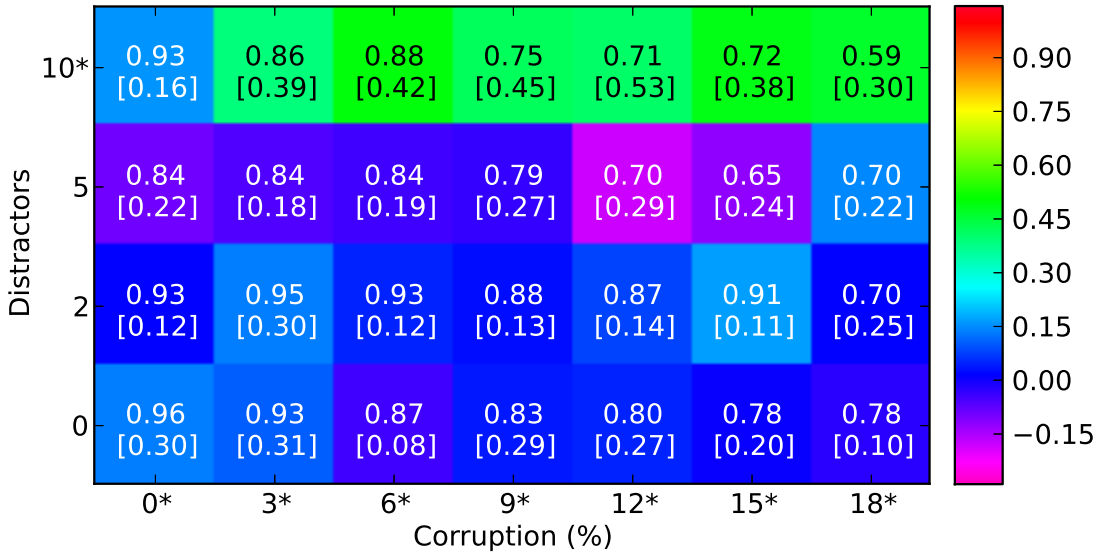


(b) SMRF - EM-DD

Figure 7.16: Mean difference PSS for the *Blue above Green* dataset.

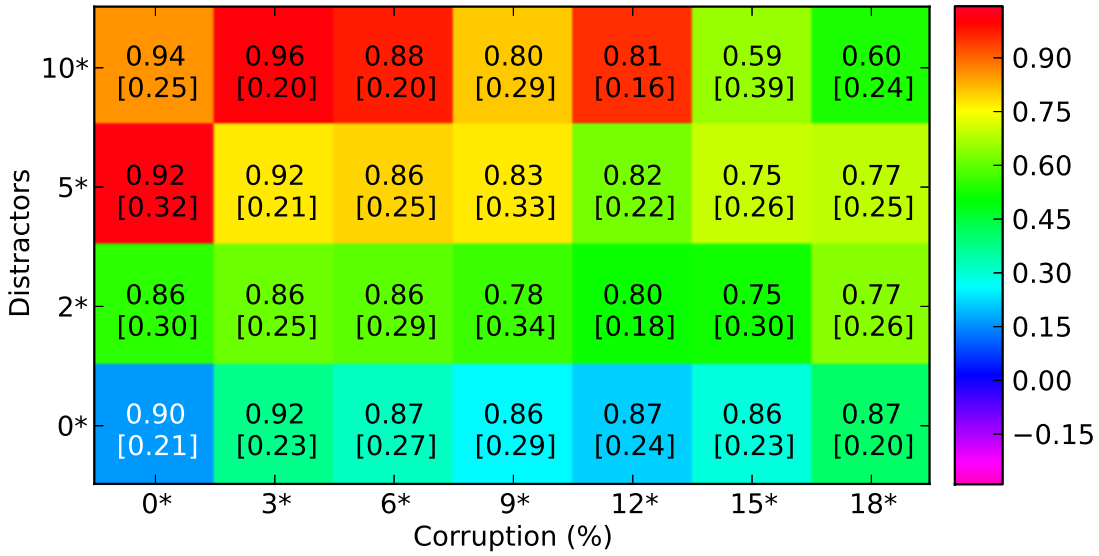


(a) SMRF - TILDE

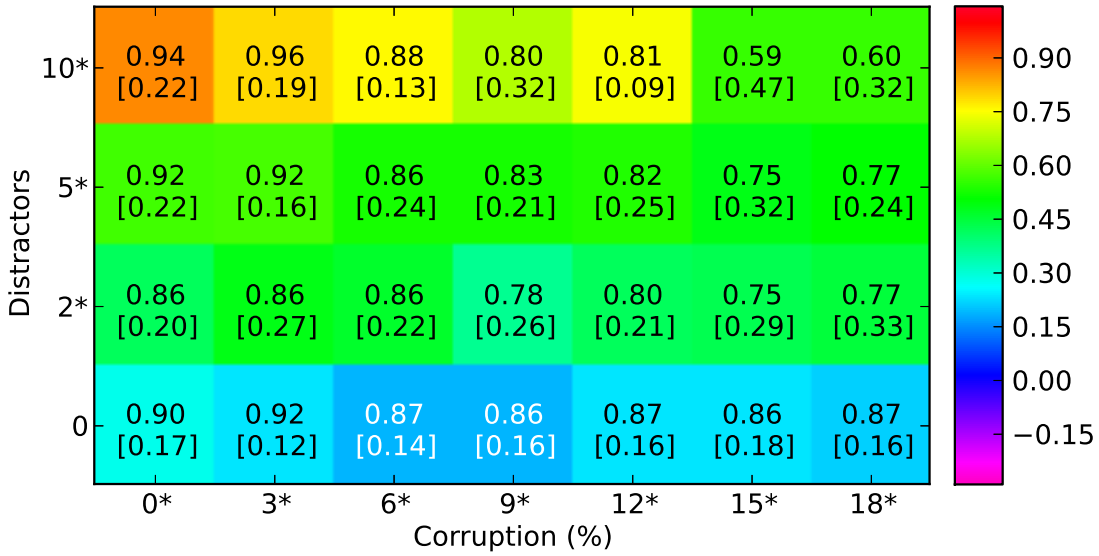


(b) SMRF - EM-DD

Figure 7.17: Mean difference PSS for the *Blue above Green or Green above Blue* dataset.



(a) SMRF - TILDE



(b) SMRF - EM-DD

Figure 7.18: Mean difference PSS for the *Red or Green* dataset.

EM-DD struggle quite a bit to perform well, especially with the addition of distractors. This problem set produces the greatest overall disparity in classification performance between SMRF and the other algorithms.

Figure 7.19 shows the performance differences for the *Red above Green or Green above Blue* dataset. As with the previous dataset, SMRF enjoys a significant overall performance advantage over TILDE, as all columns are significant, and all rows are significant except for 0 distractors.

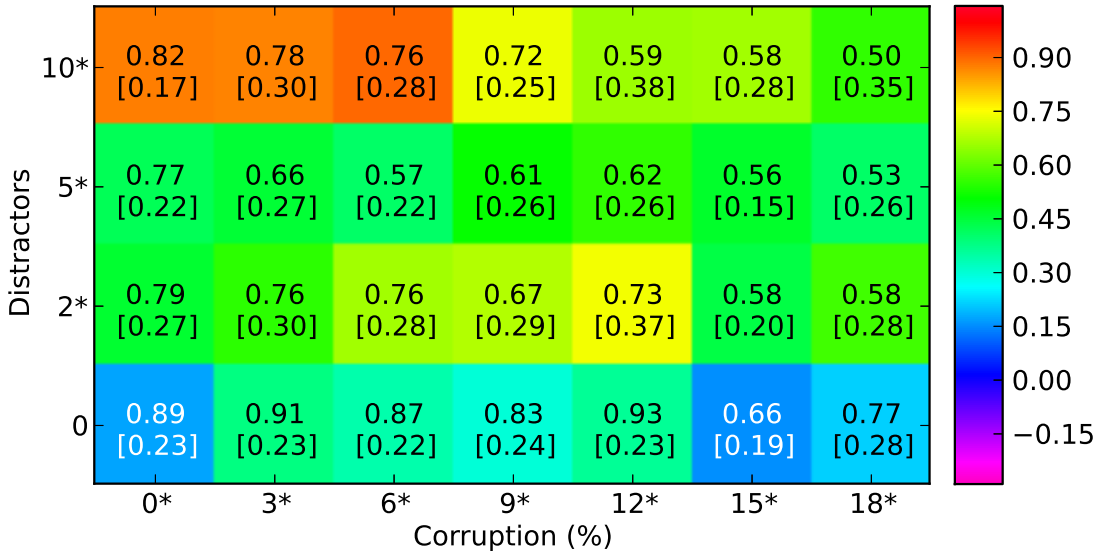
With respect to EM-DD, SMRF performs better overall at 0 and 10 distractors, as these rows are significant. However, SMRF and EM-DD performance is roughly comparable at 2 and 5 distractors. However, SMRF enjoys an overall advantage at each corruption percentage, as each column is significant.

Figure 7.20 shows the performance differences for the *Red above Green or Blue above Yellow* dataset. As with previous datasets, TILDE handles distractors poorly, and so all rows are significant except for 0 distractors, and all columns are significant.

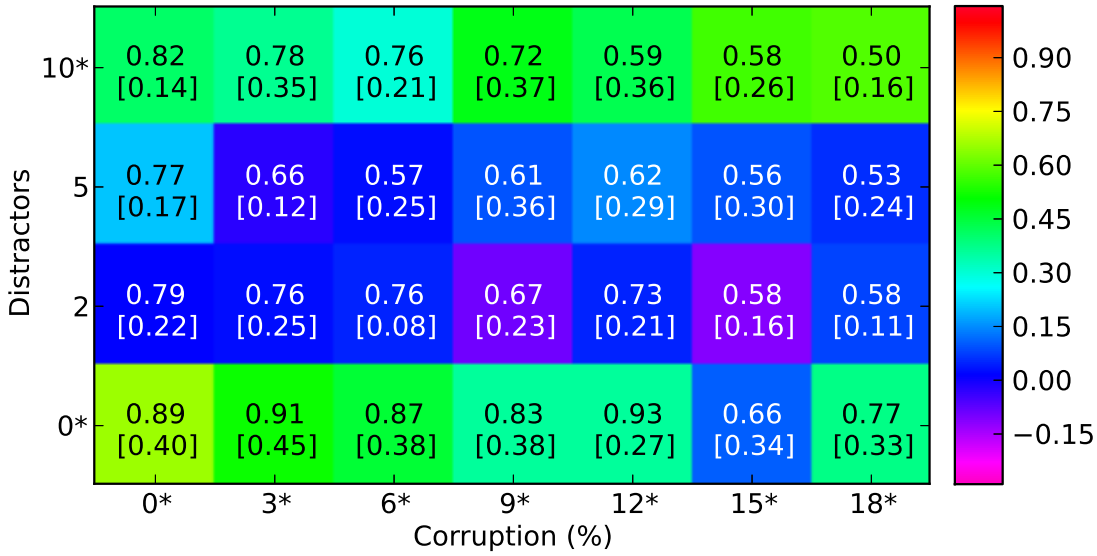
SMRF performs better than EM-DD overall at 5 and 10 distractors, doing particularly better at higher levels of corruption. Performance is roughly comparable overall at 0 and 2 distractors, with EM-DD slightly outperforming SMRF at the highest corruption percentages. However, SMRF still enjoys an overall advantage at each corruption level, as each column is significant.

Figure 7.21 shows the performance differences for the *Red or Green or Blue or Yellow* dataset. This dataset is by far the most difficult for all algorithms involved, and as such, each performs relatively poorly, particularly at higher distractor and corruption levels. As such, no columns are significant. However, for both TILDE and EM-DD, the 0, 2, and 5 distractor rows are significant. At 10 distractors, performance dips down to roughly 0 PSS for all algorithms, and as such, this row is not significant for either TILDE or EM-DD.

There are several overall trends to be observed from these results. First, there

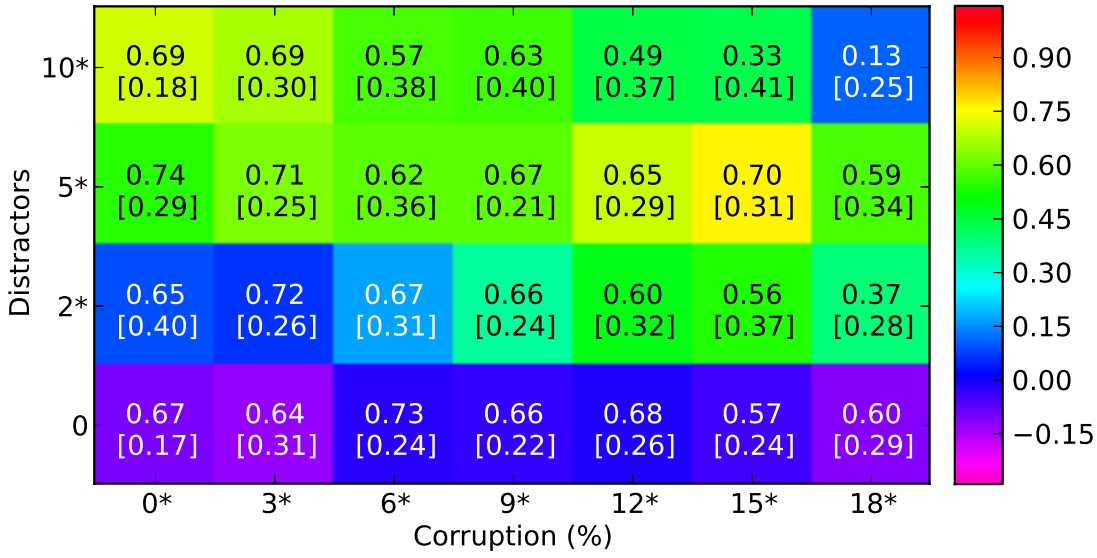


(a) SMRF - TILDE

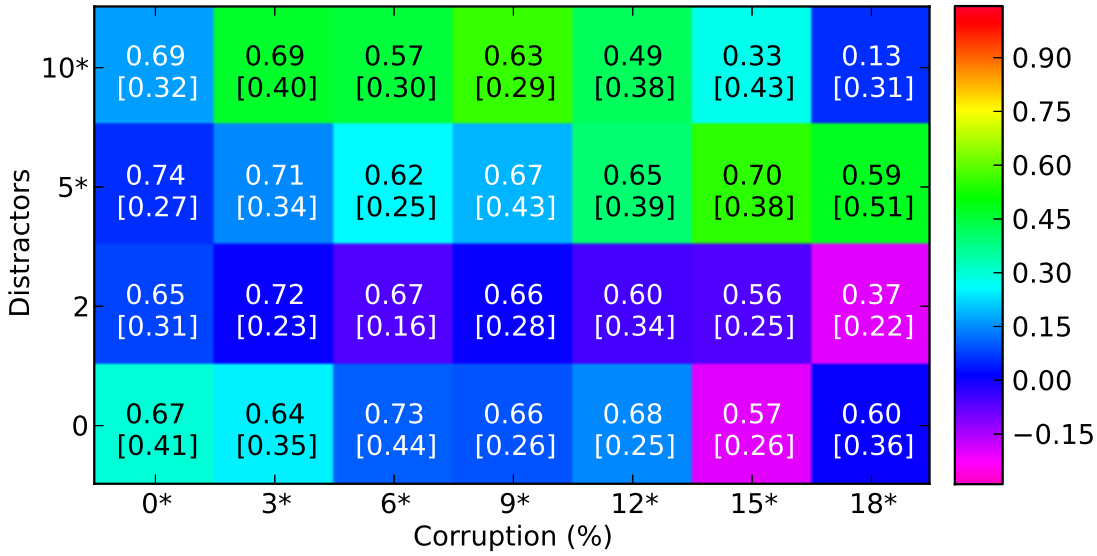


(b) SMRF - EM-DD

Figure 7.19: Mean difference PSS for the *Red above Green or Green above Blue* dataset.

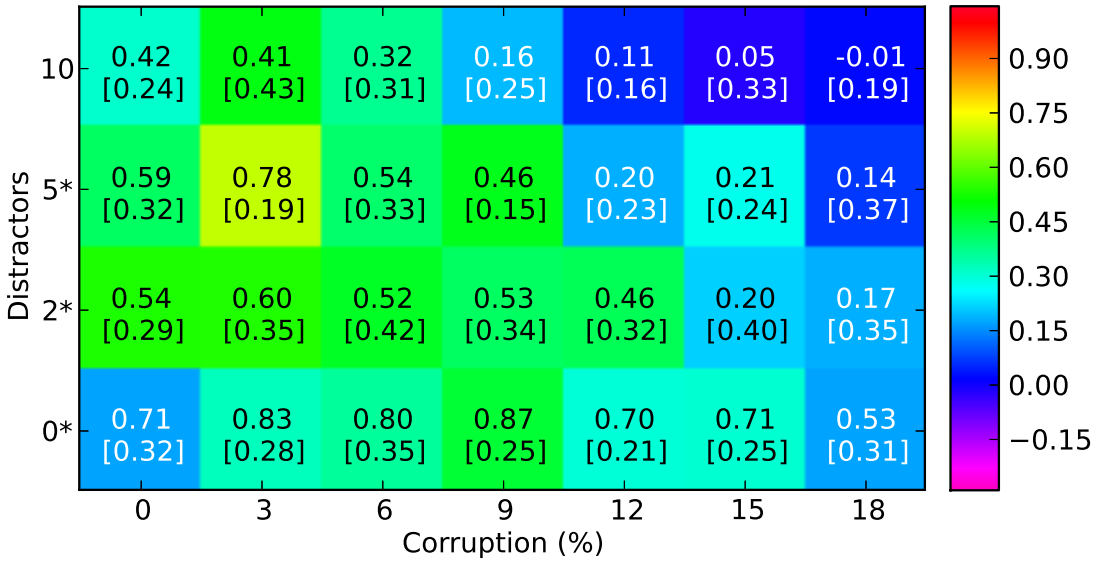


(a) SMRF - TILDE

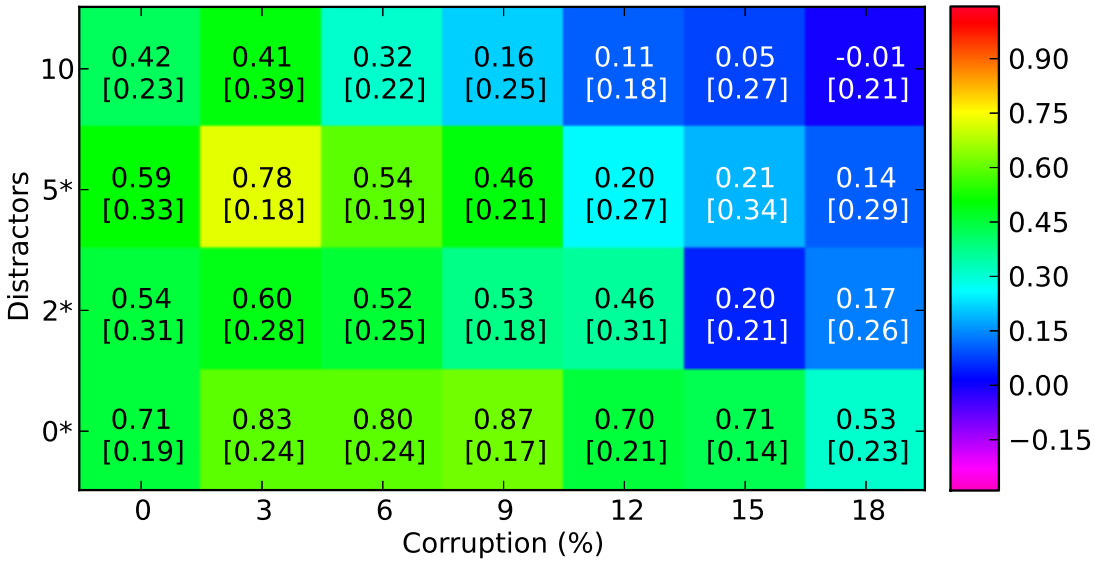


(b) SMRF - EM-DD

Figure 7.20: Mean difference PSS for the *Red above Green or Blue above Yellow* dataset.



(a) SMRF - TILDE



(b) SMRF - EM-DD

Figure 7.21: Mean difference PSS for the *Red or Green or Blue or Yellow* dataset.

are no instances where a significant row or column difference arises because another algorithm performs significantly *better* than SMRF overall. While there are a few individual cells where SMRF seems to perform worse than either TILDE or EM-DD, there are no trends which indicate that SMRF systematically performs worse for a given number of distractors or percentage of examples corrupted. On the contrary, the significance of the rows and columns in many of the figures shows that SMRF often performs better than its competitors overall.

Second, TILDE has significant trouble coping with larger numbers of distractors, relative to SMRF. While TILDE is able to perform comparably to SMRF without any distractors, it is no match for SMRF once distractors are added. Moreover, for the *Red or Green* problem set, TILDE does not even perform comparably to SMRF overall at 0 distractors. The only exception to this trend is the *Red or Green or Blue or Yellow* dataset, in which the 10 distractor row is insignificant. However, this dataset is so difficult, that all algorithms perform relatively poorly, especially with distractors and a moderate amount of corruption. As such, the results presented in Figure 7.21 seem to be influenced to a large degree by floor effects. Even so, SMRF still outperforms TILDE overall at 0, 2, and 5 distractors.

Third, with respect to EM-DD, using column significance as an indicator, there is difference between the first two datasets and the next four. All of the columns are significant in each of these four datasets, which are more difficult, and none of the columns are significant in any of the first two datasets, which are the easiest, as well as the simplest. Each of the first two datasets is a purely conjunctive concepts, while the next four contain purely disjunctice and mixed complex disjunctive concepts. As such, SMRF outperforms EM-DD overall at all levels of corruption, for both *Red or Green* and all three of the complex datasets.

Fourth, in the five disjunctive datasets, when SMRF does not outperform EM-DD at all distractor levels, it tends to do so at the higher distractor levels, such as

at 10 distractors, and sometimes also at 5 distractors. This suggests that SMRF has a superior ability to deal with distractors than EM-DD (and also TILDE), even on datasets where the algorithms perform comparably for lower numbers of distractors.

These trends, taken together, suggest that SMRF generally performs much better on these datasets than its competitor approaches. SMRF generally outperforms TILDE whenever distractors are introduced. Likewise, SMRF generally outperforms EM-DD on mixed disjunctive concepts when there are a relatively large number of distractors. Moreover, SMRF never suffers a significant performance disadvantage. As such, SMRF, at worst, exhibits performance comparable to TILDE and EM-DD in a few cases, but also exhibits performance superior to these algorithms in many cases.

7.2.4 Other Aggregate Results

Learning Time

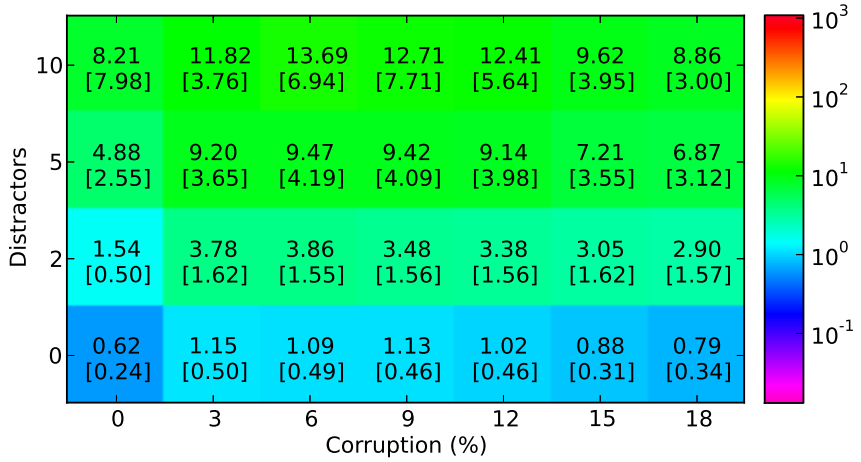
As with the figures in Section 7.2.2, the figures in this section are grouped according to algorithm type. The tree-based algorithms are given first, followed by the non-SVM-based MIL algorithms, followed by the SVM-based MIL algorithms. These figures show the mean learning time (in terms of minutes) for each algorithm across all datasets, across the range of possible distractor and corruption percentage values. Like the figures in Section 7.2.2, each figure is a heat map, with the number of distractors on the vertical axis and the corruption percentage on the horizontal axis. The first (top) number in each cell denotes the mean learning time value for that cell. The second (bottom) number, which is surrounded by brackets, is the standard deviation of the individual learning time values in this cell. For these figures, the maximum mean learning time is 1,085 minutes, and the minimum value is 0.013 minutes.

Figure 7.22a summarizes the learning time performance of SMRF for this experi-

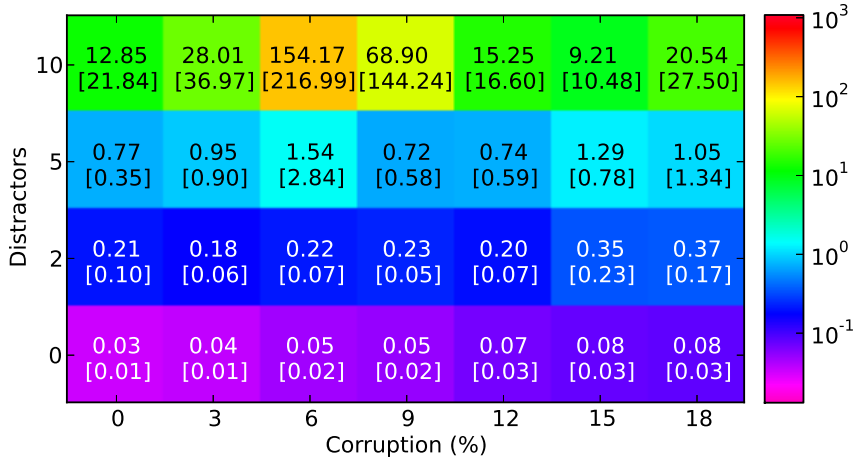
ment. SMRF learns in a relatively reasonable time frame, as there are no mean values greater than 14 minutes. With no noise, SMRF is able to learn concepts on average in a little over half a minute. The addition of distractors produces an increase in learning time that roughly follows a trend of one extra minute per distractor. As such, the increase in learning time due to additional distractors for these experiments is roughly linear. This makes sense, given that SMRF avoids the complexities of multi-ply search, and prunes away irrelevant subsets of the instance space after each instantiation. As such, SMRF generally only directly examines a very small subset of the instance space for higher numbers of distractors, and this results in a linear-time effect of the addition of distractors.

Introducing corruption (going from 0% to 3%) generally tends to increase learning time by a factor of 50%-100%, but further increases in corruption do not tend to increase learning time. Rather, as the corruption percentage grows beyond 10%, the learning time tends to decrease. This is not unexpected, as datasets with high amounts of corruption will tend to produce lower likelihood values (cf., Eq. 5.2), which will tend to make expansions less significant, which will tend to shorten the learning process. As the data become more corrupted, SMRF generally takes less time to learn what it can.

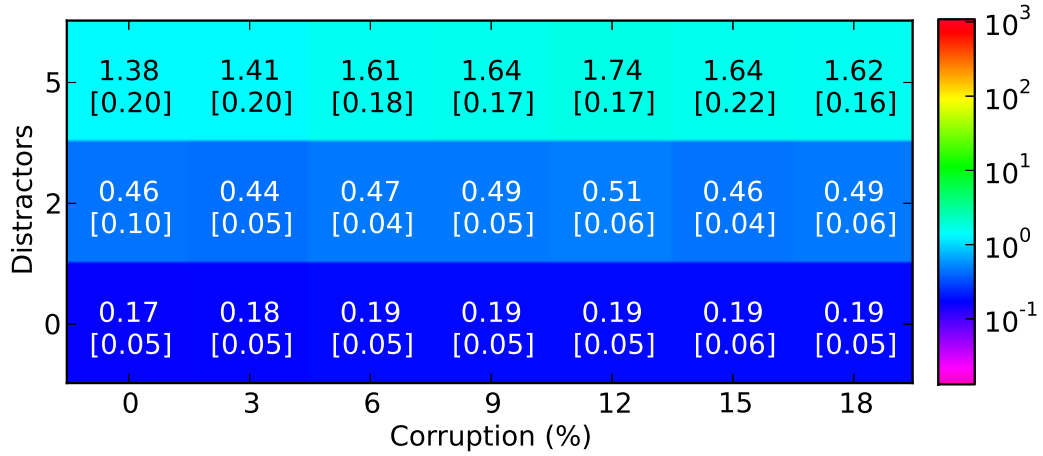
Figure 7.22b summarizes the learning time performance of TILDE for this experiment. With no distractors, TILDE is very fast, learning concepts in a matter of a few seconds, and adding corruption does not cause a substantial increase in learning time. At 2 distractors, TILDE seems to take 3-4 times as long, averaging around half a minute to learn a concept. At 5 distractors, TILDE to be yet another 3-4 times slower, averaging around a minute to learn a concept. At 10 distractors, however, the TILDE learning time increases by a factor of 10-100, depending upon the corruption level. This trend indicates that TILDE learning time increases polynomially with the number of distractors.



(a) Learning Time (m) results for SMRF.



(b) Learning Time (m) results for TILDE.



(c) Learning Time (m) results for IAPR.

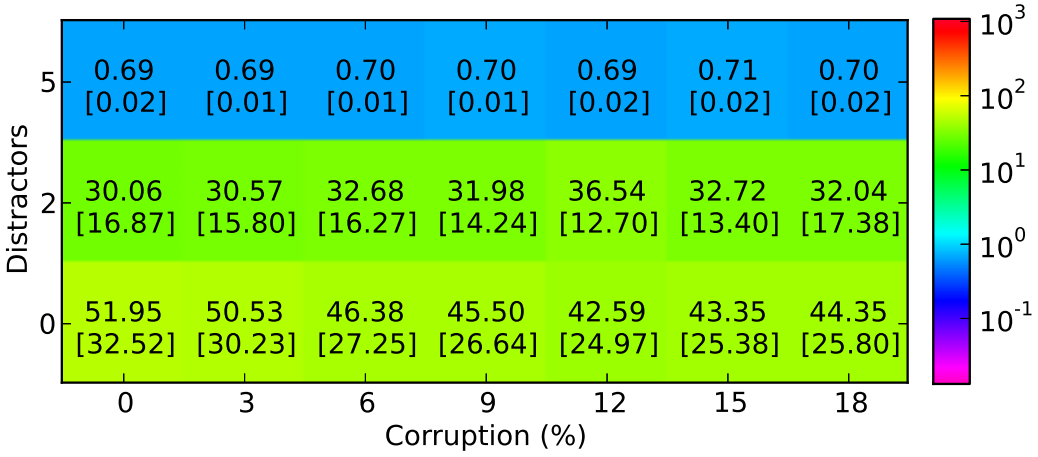
Figure 7.22: Mean Learning Time (m) Summaries

At 10 distractors, TILDE seems particularly vulnerable to moderate amounts of corruption, with a mean learning time of 154 minutes at 6% corruption, and a mean learning time of 69 minutes at 9% corruption (both with very large standard deviations).

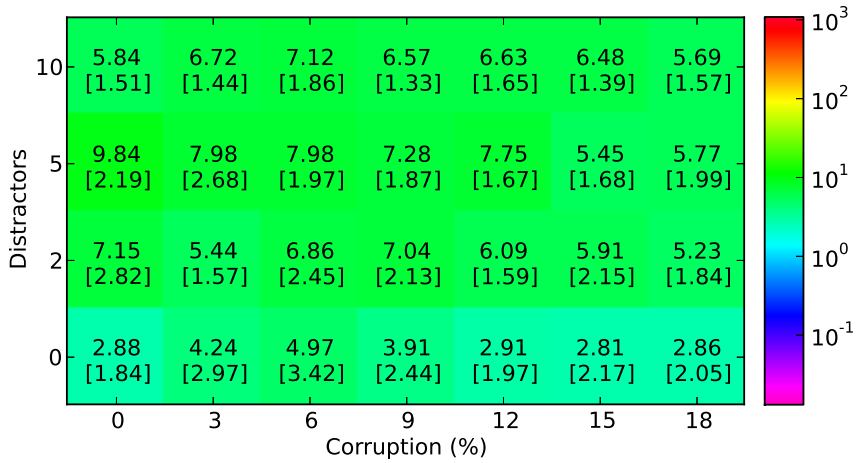
Figure 7.22c summarizes the learning time performance of IAPR for this experiment. IAPR runs very quickly, never taking more than 2 minutes on average to finish the learning process. IAPR also seems to be relatively unaffected by corruption, at all distractor levels. At no distractors, IAPR takes around 12 seconds to complete the learning process. At 2 distractors, IAPR, takes around 30 seconds, and at 5 distractors, IAPR takes around a minute and a half on average. This also suggests that learning time increases polynomially with the number of distractors. However, IAPR is still quite fast, though its quick learning time apparently does not have much of a positive effect upon its ability to learn the kinds of concepts present in this experiment.

Figure 7.23a summarizes the learning time performance of DD for this experiment. DD exhibits an interesting trends: adding distractors reduces learning time. DD takes around 45 minutes on average to learn a concept at 0 distractors, while it only takes around 32 minutes to learn a concept at 2 distractors. At 5 distractors, DD does not learn anything, and as such, takes less than a minute to run.

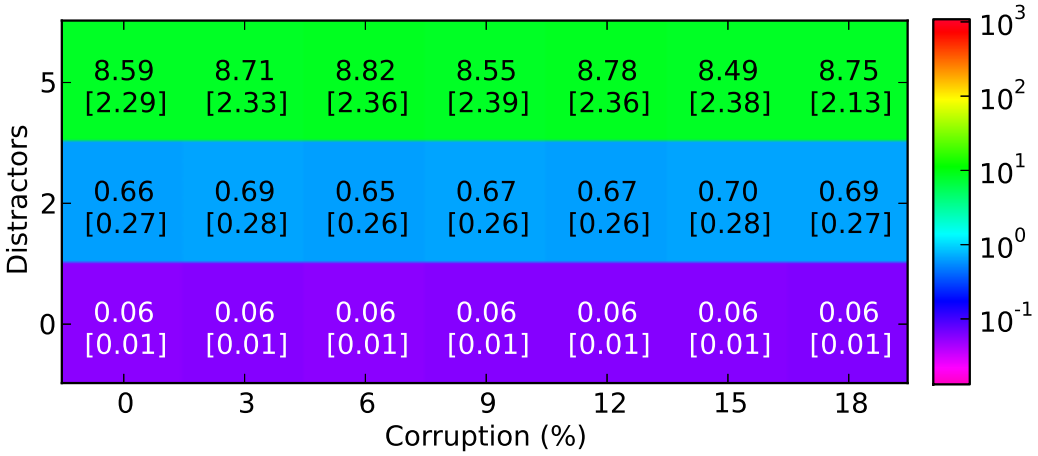
Figure 7.23b summarizes the learning time performance of EM-DD for this experiment. EM-DD tends to learn relatively quickly, never taking more than 10 minutes on average to finish the learning process. EM-DD takes around 3 and a half minutes on average at 0 distractors. This increases to 6 and a half minutes at distractors, and further to around 8 minutes at 5 distractors. However, at 10 distractors, learning time decreases back down to around 6 and a half minutes on average. Given that EM-DD experiences a large drop in classification performance in moving from 5 to 10 distractors, this drop in learning time is likely correlated.



(a) Learning Time (m) results for DD.



(b) Learning Time (m) results for EM-DD.



(c) Learning Time (m) results for kNN.

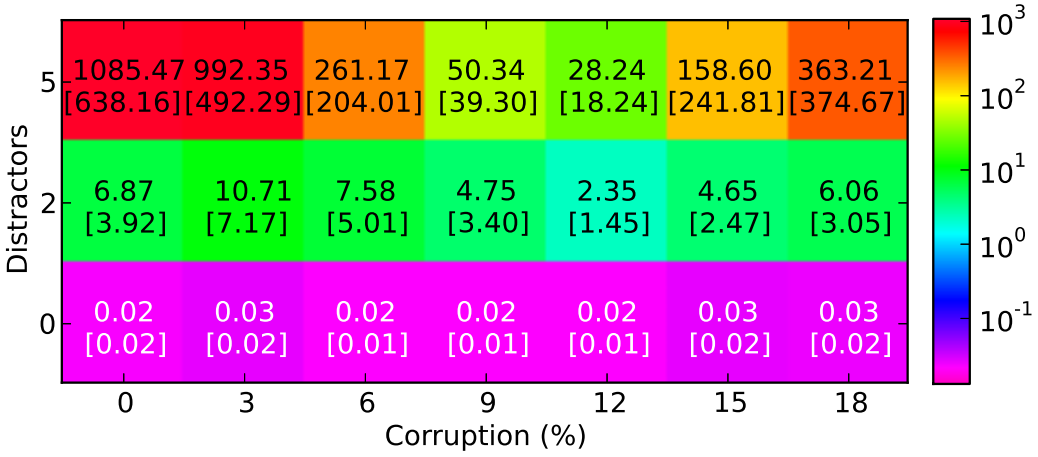
Figure 7.23: Mean Learning Time (m) Summaries

Figure 7.23c summarizes the learning time performance of CITATION-KNN for this experiment. CITATION-KNN runs very quickly at 0 distractors, taking only a few seconds on average to finish. The learning time of CITATION-KNN does not seem to be affected by corruption. Learning time increases 10-fold at 2 distractors, and increases again 10-fold at 5 distractors. This trend indicates a polynomial increase in learning time as a function of distractors. This increase is correlated, however, with a substantial decrease in classification performance.

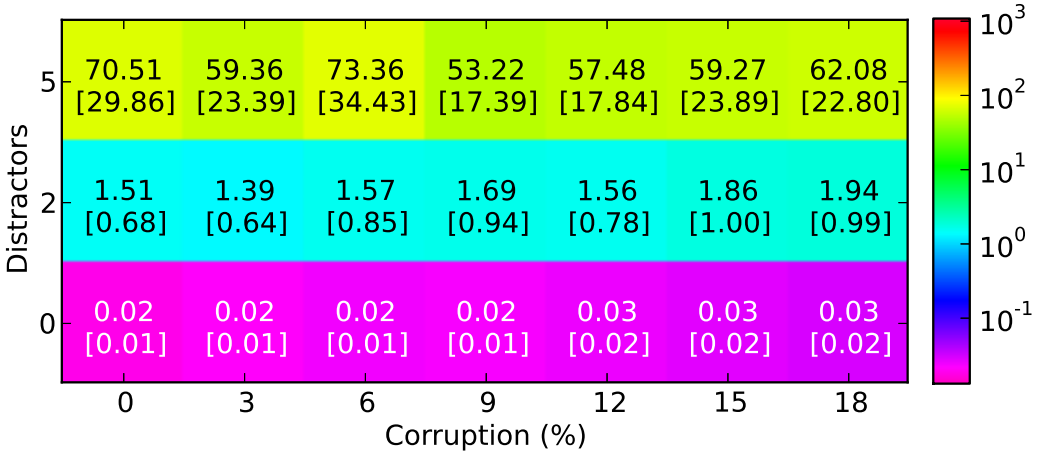
Figure 7.24a summarizes the learning time performance of MI-SVM for this experiment. MI-SVM learns very quickly in the no distractor case, taking only a few seconds to finish. At 2 distractors, however, learning time increases by around 100-fold. At 5 distractors, mean learning time again increases by around 100-fold. At 5 distractors and no corruption, MI-SVM takes on average 1100 minutes to complete, which is around 18 hours. Despite taking such a long learn at higher distractors levels, MI-SVM is one of the worst-performing algorithms in this experiment.

Figure 7.24b summarizes the learning time performance of MI-SVM for this experiment. Like MI-SVM, MI-SVM finishes within a few seconds at 0 distractors. Also like MI-SVM, MI-SVM exhibits an exponential increase in learning time as a function of the number of distractors, but this effect is not as extreme as it is for MI-SVM. At 5 distractors, MI-SVM only takes a little over an hour on average to finish. This difference can be explained somewhat by the fact that MI-SVM is more of an instance-based approach, while MI-SVM is more of a bag-based approach, and as such ends up doing fewer calculations involving instances in the course of the learning process. Like a number of other algorithms in this experiment, MI-SVM's increase in learning time with increased distractors correlates with a substantial decrease in classification performance.

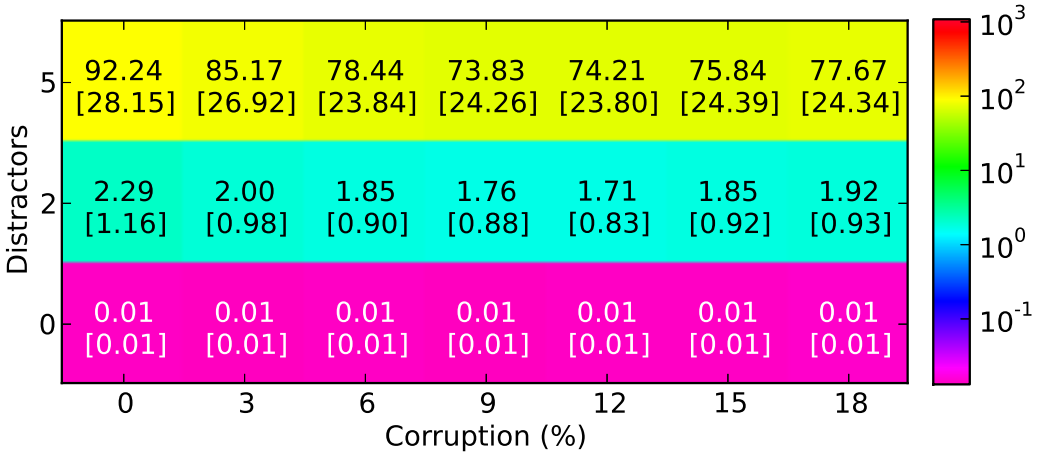
Figure 7.24c. summarizes the learning time performance of MICA for this experiment. MICA performs quite similarly to MI-SVM, finishing very quickly at 0



(a) Learning Time (m) results for mi-SVM.



(b) Learning Time (m) results for MI-SVM.



(c) Learning Time (m) results for MICA.

Figure 7.24: Mean Learning Time (m) Summaries

distractors, and exhibiting an exponential increase in learning time as the number of distractors is increased. At 5 distractors, MICA takes around 15 minutes longer than MI-SVM to finish, on average. Like with MI-SVM, the increase in learning time correlates to an substantial decrease in classification performance.

From these figures, it appears that only the learning times of SMRF and EM-DD are affected linearly by the number of distractors. Other approaches are either affected polynomially or exponentially. A number of these approaches have learning times on the order of only a few seconds at 0 distractors, while SMRF and EM-DD have learning times on the order of a minute and a couple of minutes, respectively, at 0 distractors. As such, it seems that algorithms that perform orders of magnitude more quickly than SMRF at 0 distractors generally have a trade-off with a polynomial or exponential increase in learning time as the number of distractors increases. As such, SMRF outperforms the majority of other approaches in this regard at higher distractor levels. The three approaches that perform better than SMRF (with respect to learning time) at the 5 distractor level (TILDE, IAPR, and CITATION-KNN) have terrible classification performance at that level as well. It is interesting that the two approaches with the best overall classification performance (SMRF and EM-DD) are also the only two algorithms that exhibit a linear-time increase in learning time as the number of distractors increases.

As such, with its linear-time increase in the number of distractors, SMRF is one of the most stable algorithms in the experiment. Where it is beaten by other algorithms in terms of learning time, it usually loses by amounts on the order of seconds or minutes. Where it beats other algorithms, it usually wins by amounts on the order of minutes or hours. As such, SMRF is a powerful algorithm relative to the majority of its competitors, in terms of efficiency given large numbers of distractor objects.

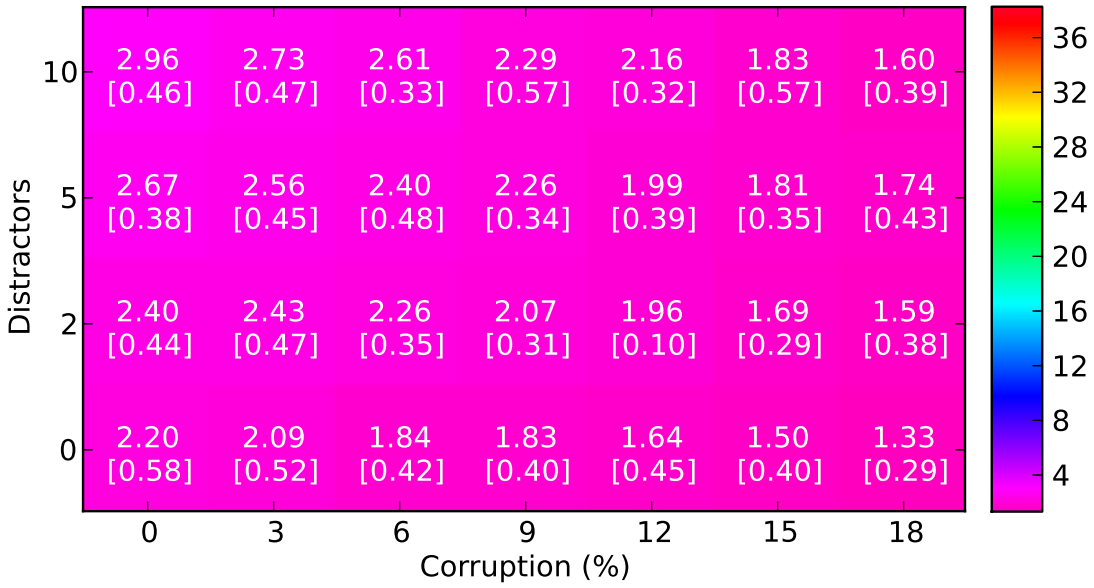
Number of Questions

Looking now at the number of questions in learned trees for both SMRF and Tilde, Figure 7.25 shows heat maps similar to the learning time heat maps discussed above. As SMRF and Tilde are the only tree-based learning algorithms, there are no figures for any other algorithms in this section. For a tree-based concept representation, size is important. The fewer splits (or question nodes, in this context) that a tree contains, the less likely it is to have overfit the training data. Likewise, smaller trees are more likely to generalize to other contexts. As such, given equal classification performance, the smaller tree is generally preferable. These figures show trends in tree size, as a function of the number of distractors and the amount of corruption present in a dataset, and allow the evaluation of the effectiveness of both SMRF and Tilde in learning compact representations under these conditions.

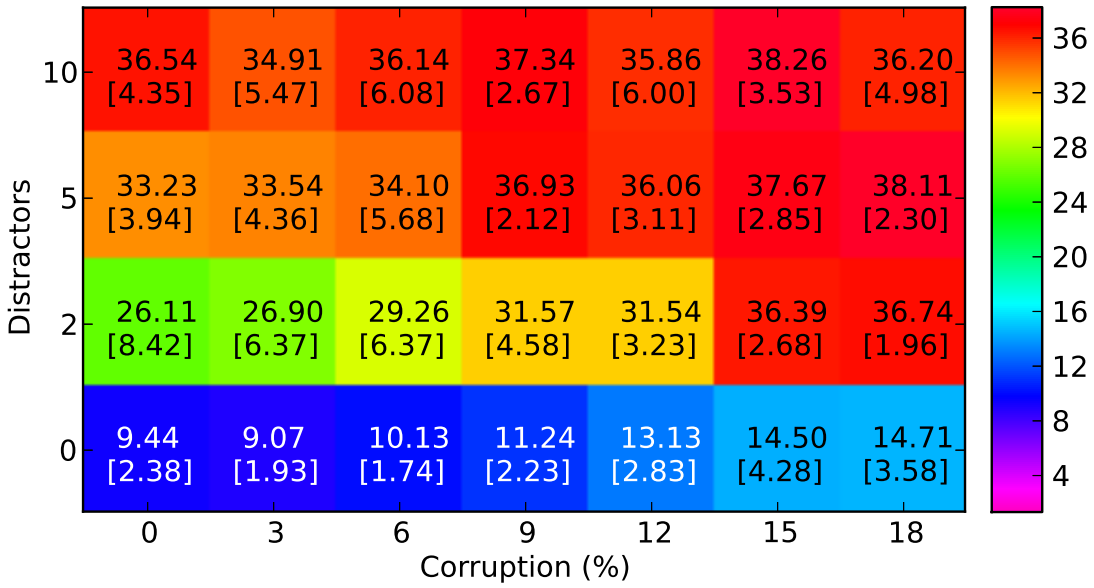
These heat maps are structured in the same way as the learning time summary heatmaps, except that information regarding the number of questions is displayed in each cell. For these figures, the maximum mean number of questions is 38.26, and the minimum value is 1.33.

Figure 7.25a summarizes the number of questions produced by SMRF in this experiment. The number of questions nodes produced by SMRF is quite low, never going about a mean of 3. In general, increasing the number of distractors forces SMRF to learn trees with more question nodes, so as to better distinguish objects in a graph that participate in a concept from the distractor objects which do not.

Increasing corruption generally tends to drive SMRF to produce smaller trees. This is not unexpected, as datasets with high amounts of corruption will tend to produce lower likelihood values (cf., Eq. 5.2), which will tend to make expansions less significant, which will tend to produce smaller trees. As the data become more corrupted, SMRF tends to lose the ability to identify as many aspects of the concept



(a) Number of Questions results for SMRF.



(b) Number of Questions results for TILDE.

Figure 7.25: Mean Number of Question Summaries

as it could in a situation with less corruption. This also makes intuitive sense, for the more that the example labels are inconsistent, the fewer warranted conclusions one can draw about the data.

Figure 7.25b summarizes the number of questions produced by TILDE in this experiment. One can immediately see that TILDE produces much larger trees than SMRF. The smallest trees that TILDE produces have 9.44 leaves on average (no distractors and no corruption), and largest trees that TILDE produces have 38.26 leaves on average (10 distractors and 15% corruption).

Unlike SMRF, TILDE tends to produce more question nodes, not fewer, in response to increased amounts of corruption. Like SMRF, however, TILDE does tend to produce more question nodes as the number of distractors increases. Unlike SMRF, however, which increases the mean number of question nodes by an amount less than 1, TILDE increases the number of question nodes by around 15-20 as it moves from no distractors to 2 distractors. The classification performance of TILDE is inversely correlated to the number of question nodes, as the number of distractors increases. This suggests that TILDE has a problem with overfitting when distractor objects are introduced into the data.

As such, from these figures, it is clear that SMRF is by far the best-performing tree-based algorithm, in terms of producing compact concept representations. At 0 distractors, where TILDE mean classification performance is not far behind SMRF, SMRF still produces trees that are an order of magnitude smaller than the trees that TILDE produces. This behavior is explained by the fact that TILDE only looks at individual dimensions. As such, many more questions would be required to represent the multidimensional covariant concepts employed in these experiments. These results show that the ability to deal directly with continuous multidimensional data is important for enabling an approach to learn compact and powerfully predictive target concept hypotheses in the context of multidimensional relational data.

7.3 Varying Training Set Size

The goal of this experiment was to determine how the size of the training set affected performance on the test set for the SMRF algorithm, as well as for TILDE and EM-DD. TILDE was chosen as a comparison algorithm because it is a well-known tree-based algorithm, and bears a number of similarities (as well as differences) to the SMRF framework. EM-DD was chosen because it was the overall best-performing algorithm, other than SMRF, in the main experiment (Sec. 7.1.5).

A 10-fold cross-validation experiment was performed for all of the possible combinations of each of the following conditions:

- **Number of Distractors**: 0, 2, 5, 10,
- **Percentage of Examples Corrupted**: 0%,
- **Dataset**: *Red or Green*, *Red or Green or Blue or Yellow*, *Blue above Green*, *Blue above Green or Green above Blue*, *Red above Green or Green above Blue*, *Red above Green or Blue above Yellow*, *Right Red right of Left Green*,
- **Total Training Set Size**: 60, 80, 100, 120, 140, 160, 180.

The condition *Total Training Set Size* (TTSS) refers to the number of graphs present in the training set. For the main experiments (Sec 7.1.5), a training set of 90 positive and 90 negative graphs were used for each cross-validation run, with a test set of 10 positive and 10 negative graphs. In this experiment, a test set of 10 positive and 10 negative graphs were also used for each experimental run. However, for runs where TTSS was set to something less than 180, the 90 positive and 90 negative training set graphs were subsampled, respectively. For example, in runs with a TTSS of 60, the training set included 30 positive and 30 negative graphs. In runs with a TTSS of 80, 40 positive and 40 negative graphs were used. This pattern was preserved, where the

training set for each run was composed of an equal number of positive and negative graphs. The results of these experiments are as follows.

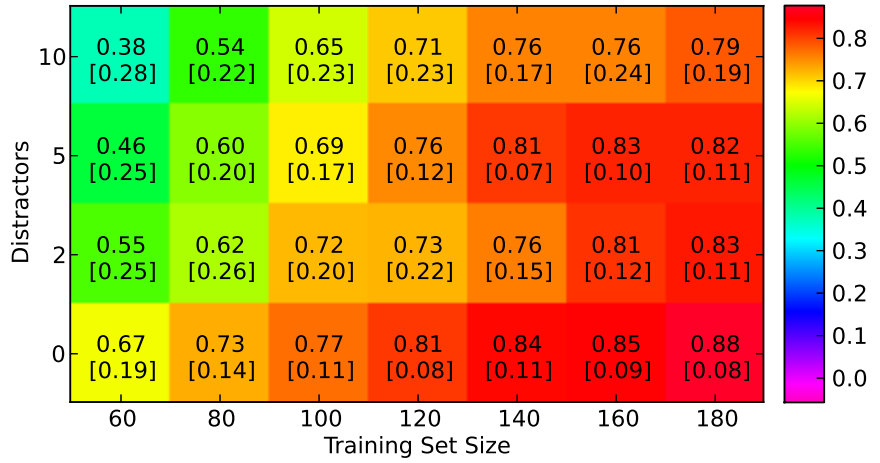
The following figures show the mean classification performance (in terms of PSS) for SMRF, TILDE, and EM-DD across all datasets, across the range of possible distractor and training set size values. Like the figures in Section 7.2.2, each figure is a heat map, with the number of distractors on the vertical axis and the training set size on the horizontal axis. The first (top) number in each cell denotes the mean PSS value for this cell. The second (bottom) number, which is surrounded by brackets, is the standard deviation of the individual PSS values in this cell. For these figures, the maximum mean PSS value is 0.88, and the minimum value is -0.06.

Additionally, for each comparison algorithm, each row and column may be marked with an asterisk. Such markings are present when the aggregate row (or column) results are significantly different than the corresponding SMRF results. The Bonferroni test is used, with aggregate $\alpha = 0.05$.

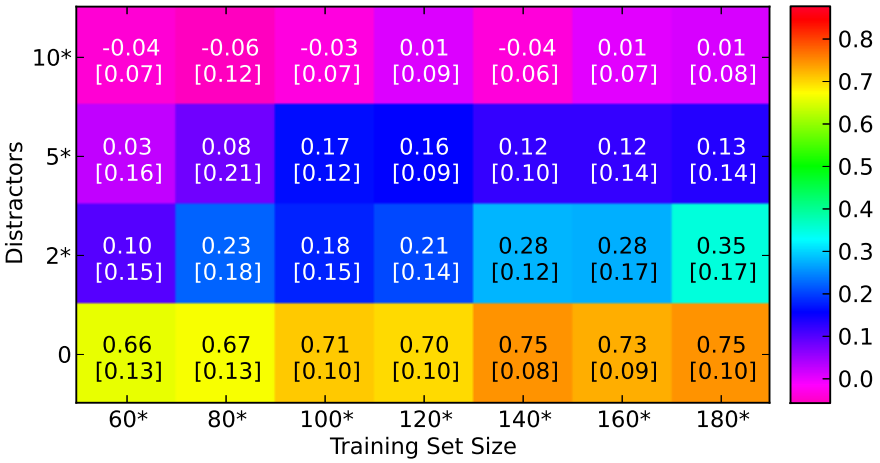
Figure 7.26a summarizes the classification performance of SMRF for this experiment. At training set size 60, classification performance is markedly reduced from the corresponding PSS values at training set size 180. However, SMRF gets better quickly, and at a training set size of 120, values are almost up to their maximum values at a training set size of 180.

Figure 7.26b summarizes the classification performance of TILDE for this experiment. TILDE performs comparably to SMRF at 0 distractors, though SMRF pulls ahead around training set size 120. At 2, 5, and 10 distractors, TILDE performs poorly, and SMRF performs significantly better. Likewise, SMRF performs significantly better overall for each training set size.

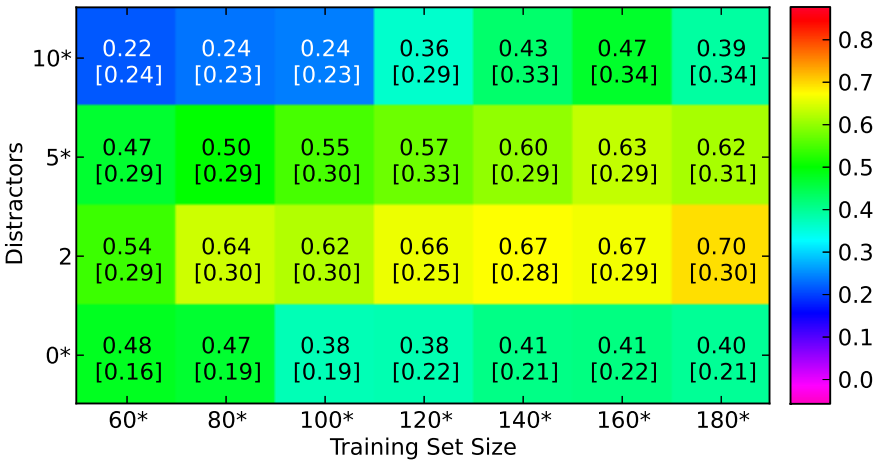
Figure 7.26c summarizes the classification performance of EM-DD for this experiment. EM-DD performs comparably to SMRF at training size 60 at 2 and 5 distractors, though SMRF performs better at 0 and 10 distractors. SMRF quickly



(a) PSS results for SMRF.



(b) PSS results for TILDE.



(c) PSS results for EM-DD.

Figure 7.26: Mean PSS Summaries

pulls away, however, and is better than EM-DD at training set size 100.

SMRF significantly outperforms EM-DD at each training set size level. Likewise, SMRF significantly outperforms EM-DD at 0, 5, and 10 distractors. The 2 distractor row is likely not significant because EM-DD, like DD, tends to perform better with a slight number of distractors added to the data.

As such, from these figures, it is clear that SMRF, on average, suffers no loss of performance, relative to the other algorithms, for small training set sizes at least as large as 60 examples. On the contrary, SMRF outperforms the other algorithms for the majority of cells.

In order to analyze these results in more detail, consider that there are four factors involved in determining the shape of the training set size experiment: the problem set, the algorithm, the number of distractors, and the training set size. To gain an understanding of which factors (and which combinations of factors) have a significant effect on classification performance, a 4-way analysis of variance (ANOVA) was performed on the PSS data. The results of this ANOVA are given in Table 7.12.

As one might expect, each of the individual factors are significant. Additionally, each two factor combination is significant except for problem set $\times$ training set size.

For each of the individual factors, *post hoc* tests were performed to determine which combinations of values were significant. The Bonferroni test was used at significance level $\alpha = 0.05$.

One of the primary results to take away from this analysis is that SMRF significantly and substantially outperforms both Tilde and EM-DD across the range of factor values. This can be seen in the results for the algorithm factor *post hoc* tests, given in Table 7.13. It should be noted that Table 7.8 does not contain all pairwise comparisons for the algorithm factor, but only those involving SMRF. The mean SMRF performance (across all other factors) is 0.719 PSS. In contrast, the mean Tilde performance is 0.260 PSS. EM-DD does much better than Tilde, with

Factors	SS	df	MS	F	p-value
Problem Set	95.852	6	15.975	261.474	~**0**
Algorithm	206.427	2	103.213	1689.332	~**0**
Distractors	77.539	3	25.846	423.039	~**0**
Training Set Size	21.483	6	3.580	58.603	~**0**
Problem Set × Algorithm	42.374	12	3.531	57.796	~**0**
Problem Set × Distractors	21.388	18	1.188	19.448	~**0**
Problem Set × Training Set Size	1.754	36	0.049	0.798	0.800
Algorithm × Distractors	103.176	6	17.196	281.452	~**0**
Algorithm × Training Set Size	6.676	12	0.556	9.106	~**0**
Distractors × Training Set Size	3.755	18	0.209	3.414	~**0**
Problem Set × Algorithm × Distractors	46.654	36	1.296	21.211	~**0**
Problem Set × Algorithm × Training Set Size	8.417	72	0.117	1.913	~**0**
Problem Set × Distractors × Training Set Size	7.121	108	0.066	1.079	0.273
Algorithm × Distractors × Training Set Size	4.085	36	0.113	1.857	**0.001**
Problem Set × Algorithm × Distractors × Training Set Size	13.712	216	0.063	1.039	0.336
Error	323.326	5292	0.061		
Total	983.740	5879			

Table 7.12: Size sweep experiment ANOVA.

		Bonferroni Test	
Comparison	Means	F	p-value
EM-DD × SMRF	0.491 × 0.719	827.938	~**0**
SMRF × TILDE	0.719 × 0.260	3378.553	~**0**

Table 7.13: Post hoc tests for factor Algorithm

		Bonferroni Test	
Comparison	Means	F	p-value
5 × 0	0.463 × 0.641	378.759	~**0**
5 × 10	0.463 × 0.324	233.646	~**0**
5 × 2	0.463 × 0.532	56.229	~**0**
0 × 10	0.641 × 0.324	1207.368	~**0**
0 × 2	0.641 × 0.532	143.116	~**0**
10 × 2	0.324 × 0.532	519.114	~**0**

Table 7.14: Post hoc tests for factor Distractors

a mean performance of 0.491 PSS. In both cases, SMRF performs significantly better, with p-value less than 10^{-3}.

Next, looking at the *post hoc* tests for the distractor factor, given in Table 7.14, one can see that each pairwise comparison is significant. As with the main experiment analysis, this serves to show that the addition of distractors has a meaningful effect upon learning difficulty, even for other algorithms in addition to SMRF.

The *post hoc* tests for the training set size factor are given in Table 7.15. Not all level comparisons are statistically significant. Below 140, each pairwise comparison with a difference of greater than 20 is significant. Also, the difference between 60 and 80 is significant. This serves to demonstrate that for training set sizes less than 140, change in training set size has a real effect on algorithm difficulty. At 140 and greater, no pairwise tests are significant, which indicates that performance generally ceases to meaningfully improve after 140 examples are included in the training set. This result matches intuition, as one would expect increases in training set size to be significant for small numbers, and then to effectively level out after reaching a certain size.

		Bonferroni Test	
Comparison	Means	F	p-value
60 × 80	0.377 × 0.438	25.474	**~0**
60 × 100	0.377 × 0.471	60.441	**~0**
60 × 120	0.377 × 0.505	112.945	**~0**
60 × 140	0.377 × 0.534	168.837	**~0**
60 × 160	0.377 × 0.547	199.281	**~0**
60 × 180	0.377 × 0.556	219.193	**~0**
80 × 100	0.438 × 0.471	7.437	0.269
80 × 120	0.438 × 0.505	31.141	**~0**
80 × 140	0.438 × 0.534	63.148	**~0**
80 × 160	0.438 × 0.547	82.256	**~0**
80 × 180	0.438 × 0.556	95.218	**~0**
100 × 120	0.471 × 0.505	8.141	0.182
100 × 140	0.471 × 0.534	27.242	**~0**
100 × 160	0.471 × 0.547	40.225	**~0**
100 × 180	0.471 × 0.556	49.433	**~0**
120 × 140	0.505 × 0.534	5.599	0.756
120 × 160	0.505 × 0.547	12.174	**0.021**
120 × 180	0.505 × 0.556	17.453	**0.001**
140 × 160	0.534 × 0.547	1.261	10.983
140 × 180	0.534 × 0.556	3.281	2.945
160 × 180	0.547 × 0.556	0.474	20.630

Table 7.15: Post hoc tests for factor Training Set Size

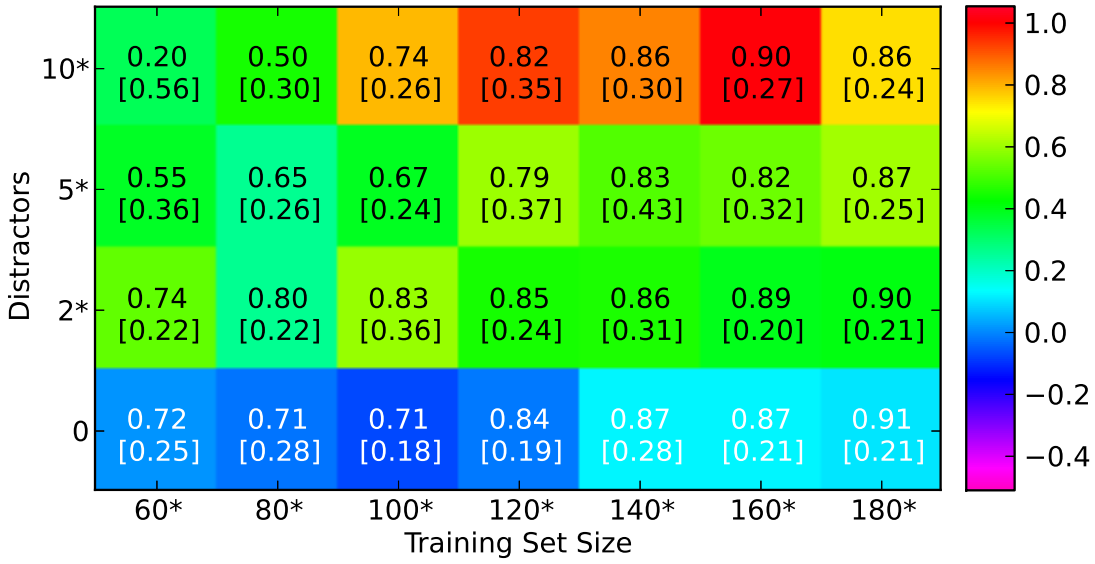
Next, the *post hoc* tests for the problem set factor are given in Table 7.11. Table 7.16. As with the main experiment analysis, these results can be interpreted as a measure of which problems are more difficult than others for the combination of SMRF, Tilde, and EM-DD. The results are similar to those obtained in the main experiment (Sec. 7.2.3). As with the main experiment, *Blue above Green (BaG)*, *Blue above Green or Green above Blue (BaG / GaB)*, and *Right Red right of Left Green [(RtR)ro(LtG)]* are the easier problems. This can be seen in the higher mean PSS, along with the statistically insignificant pairwise comparisons. The other four problem sets are harder for the group of three algorithms to solve. *Red or Green or Blue or Yellow (R / G / B / Y)* is by far the hardest, with a mean PSS of only 0.219. Unlike in Table 7.11, there is only one pair with an insignificant PSS difference: *Red or Green (R / G)* and *Red above Green or Green above Blue (RaG / GaB)*. All other pairs are statistically insignificant.

The order of problem set difficulty implied by Table 7.16 is reflected in the ordering of Figures 7.27 - 7.33, which are given in order from easiest to hardest. These figures, like their counterparts in Section 7.2.3, show how much better (or occasionally, worse) SMRF performs relative to either Tilde or EM-DD, for each problem set, across the range of possible distractor and training set size values. Each figure contains two subfigures, one for the difference in performance between SMRF and Tilde, and the other for the difference in performance between SMRF and EM-DD. Each subfigure is a heat map, with the number of distractors on the vertical axis and the training set size on the horizontal axis. In each cell, the first (top) number denotes the mean PSS of SMRF for this cell. The second (bottom) number, which is surrounded by brackets, is the standard deviation of the difference in PSS between SMRF and either Tilde or EM-DD, for this cell. For these figures, the maximum PSS difference is 1.05, and the minimum difference is -0.51.

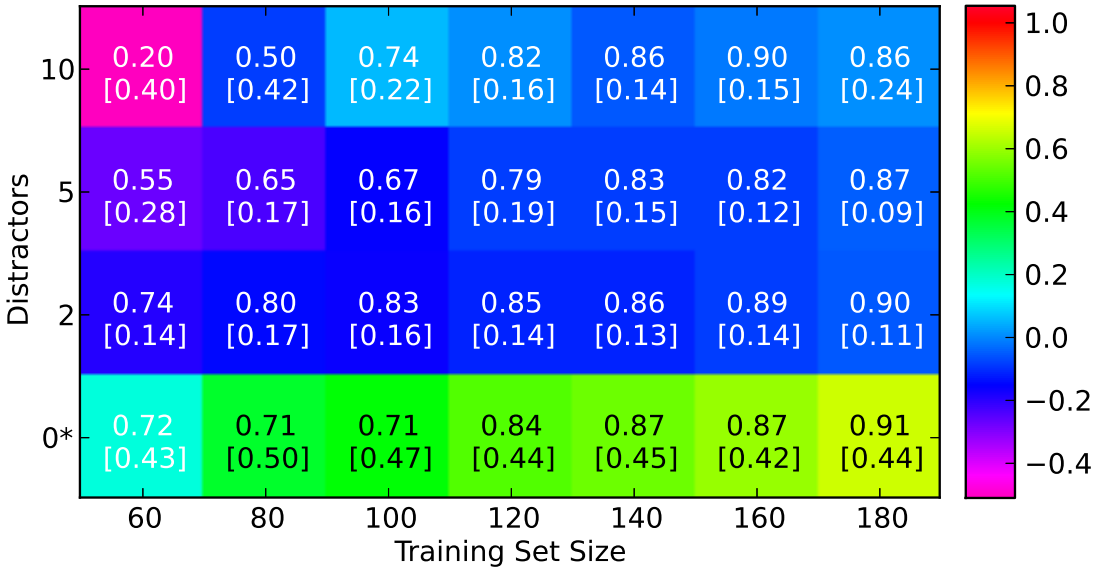
Each individual figure is discussed in the following pages, followed by a summary of

		Bonferroni Test	
Comparison	Means	F	p-value
BaG/GaB × R/G	0.601 × 0.479	102.723	~**0**
BaG/GaB × RaG/BaY	0.601 × 0.418	230.706	~**0**
BaG/GaB × RaG/GaB	0.601 × 0.482	96.940	~**0**
BaG/GaB × (RtR)ro(LtG)	0.601 × 0.617	1.781	7.645
BaG/GaB × R/G/B/Y	0.601 × 0.230	948.117	~**0**
BaG/GaB × BaG	0.601 × 0.602	0.002	40.524
R/G × RaG/BaY	0.479 × 0.418	25.541	~**0**
R/G × RaG/GaB	0.479 × 0.482	0.084	32.435
R/G × (RtR)ro(LtG)	0.479 × 0.617	131.560	~**0**
R/G × R/G/B/Y	0.479 × 0.230	426.681	~**0**
R/G × BaG	0.479 × 0.602	103.618	~**0**
RaG/BaY × RaG/GaB	0.418 × 0.482	28.550	~**0**
RaG/BaY × (RtR)ro(LtG)	0.418 × 0.617	273.034	~**0**
RaG/BaY × R/G/B/Y	0.418 × 0.230	243.438	~**0**
RaG/BaY × BaG	0.418 × 0.602	232.047	~**0**
RaG/GaB × (RtR)ro(LtG)	0.482 × 0.617	125.004	~**0**
RaG/GaB × R/G/B/Y	0.482 × 0.230	438.722	~**0**
RaG/GaB × BaG	0.482 × 0.602	97.809	~**0**
(RtR)ro(LtG) × R/G/B/Y	0.617 × 0.230	1032.095	~**0**
(RtR)ro(LtG) × BaG	0.617 × 0.602	1.666	8.268
R/G/B/Y × BaG	0.230 × 0.602	950.832	~**0**

Table 7.16: Post hoc tests for factor Problem Set



(a) SMRF - TILDE



(b) SMRF - EM-DD

Figure 7.27: Mean difference PSS for the *Right Red right of Left Green* dataset.

the trends observed in these results. Figure 7.27 shows the performance differences for the *Right Red right of Left Green* dataset. This is the easiest dataset in terms of mean PSS. As TILDE does not handle distractors well, the rows for 2, 5, and 10 distractors are all statistically significant. Likewise, each column is statistically significant, which is likely also a result of the fact that TILDE does not handle distractors well.

For EM-DD, no rows are significant. SMRF seems to perform better than EM-DD on this dataset at 0 distractors, and slightly pull away from EM-DD as the training set size is increased (though this trend is not significant). However, as the number of distractors is increased, SMRF seems to perform more poorly, and requires more training data to catch up with EM-DD. However, as the number of distractors increases, SMRF is able to improve classification performance faster relative to EM-DD, requiring less training data to do so. No columns are significant.

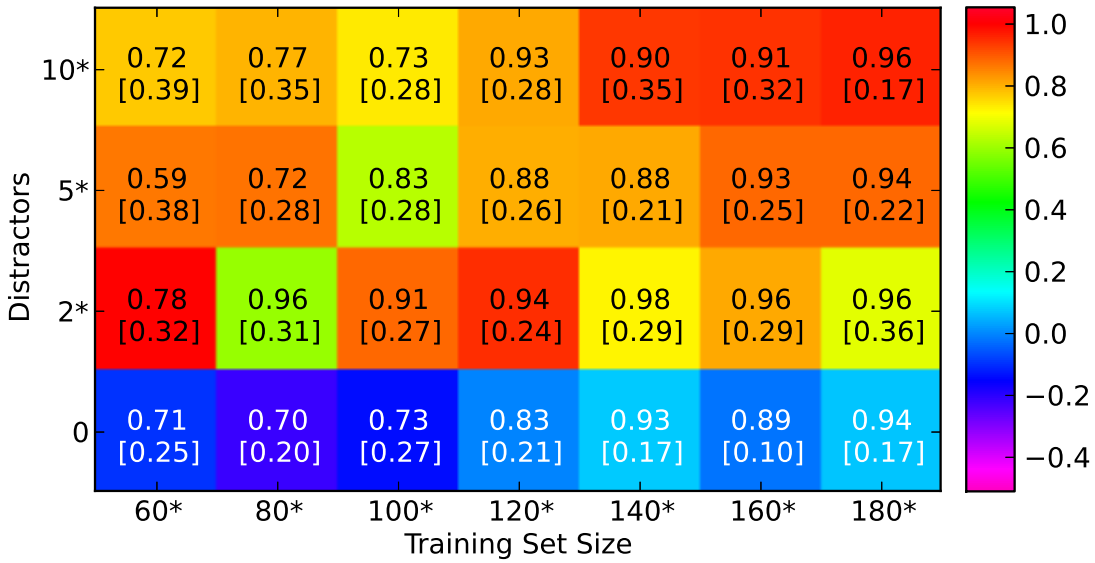
Figure 7.28 shows the performance differences for the *Blue above Green* dataset. As with the previous dataset, TILDE handles distractors poorly, and so all rows are significant except for 0 distractors, and all columns are significant.

As this is a slightly harder problem, SMRF performs better relative to EM-DD than with the previous problem set. The first row is significant, as SMRF outperforms EM-DD all around, pulling away substantially as the training set size approaches 180. At 2 and 5 distractors, SMRF starts out worse than EM-DD, but then pulls ahead slightly with more data. At 10 distractors, SMRF starts out with a sizable advantage at training set size 60, but then EM-DD gradually catches up. As such, no columns are significant.

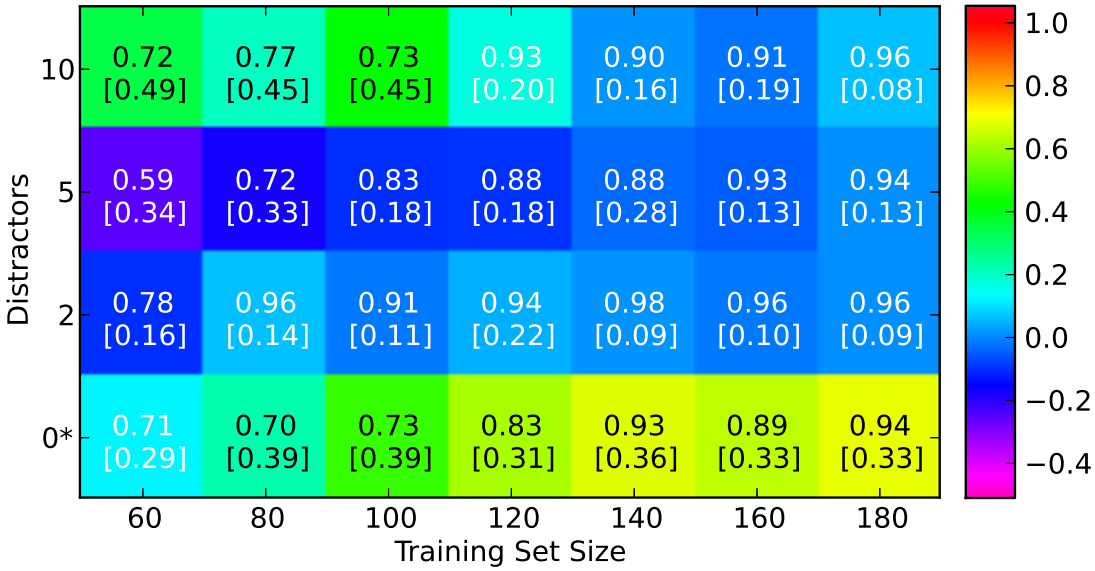
Figure 7.29 shows the performance differences for the *Blue above Green or Green above Blue* dataset. As with the previous two datasets, TILDE handles distractors poorly, and so all rows are significant except for 0 distractors, and all columns are significant.

As this is a slightly harder problem, SMRF performs better relative to EM-DD than with the previous problem set. SMRF enjoys no significant advantage at 0, 2, or 5 distractors. However, at 10 distractors, SMRF enjoys a clear advantage with a significant row. For this problem set, SMRF enjoys a clear overall advantage at each training set size, as each column is significant.

Figure 7.30 shows the performance differences for the *Red above Green or Green*

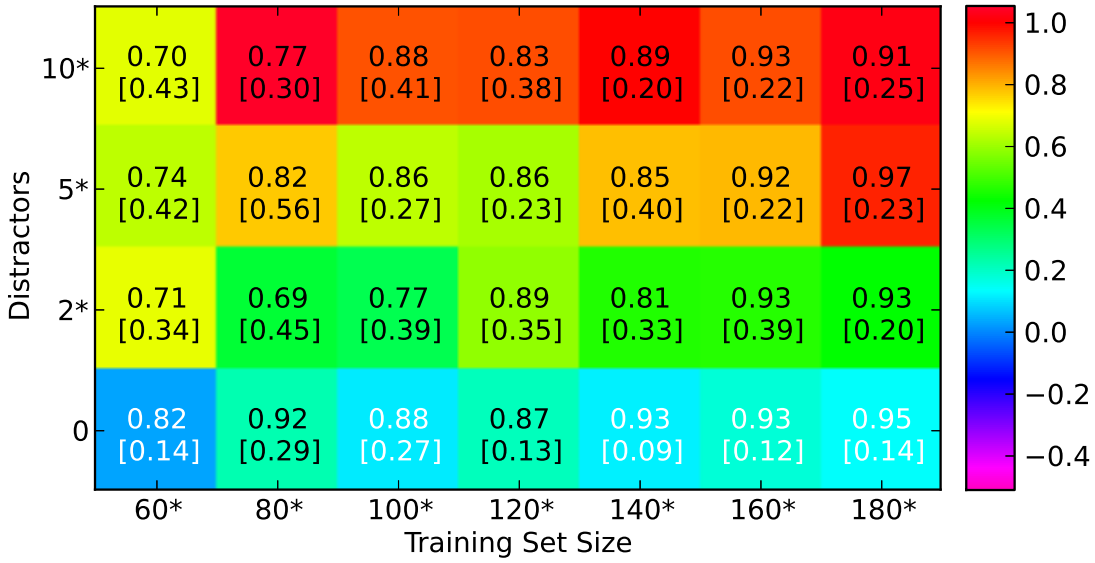


(a) SMRF - TILDE

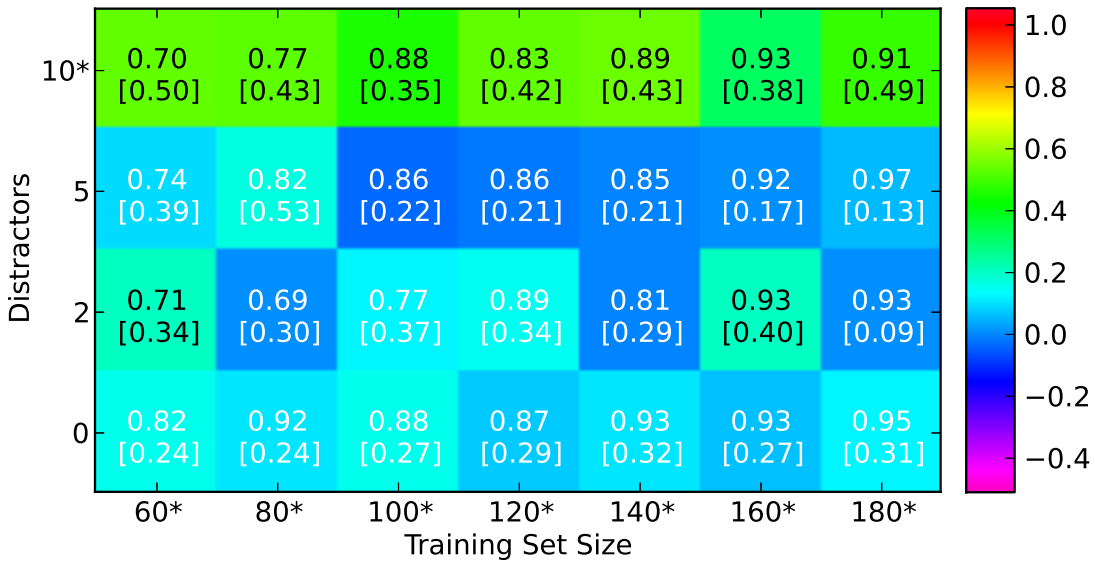


(b) SMRF - EM-DD

Figure 7.28: Mean difference PSS for the *Blue above Green* dataset.

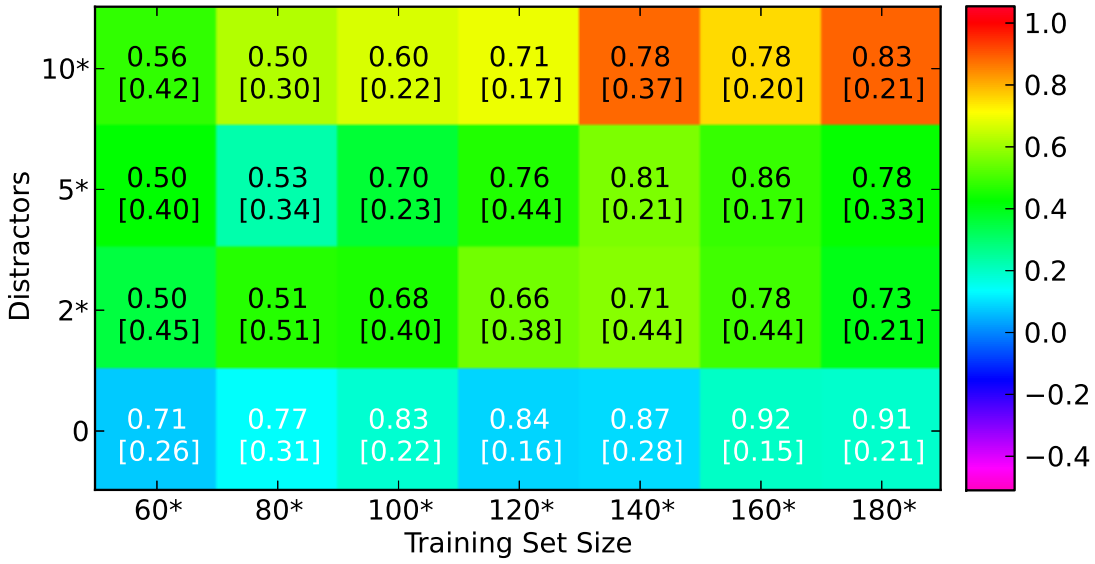


(a) SMRF - TILDE

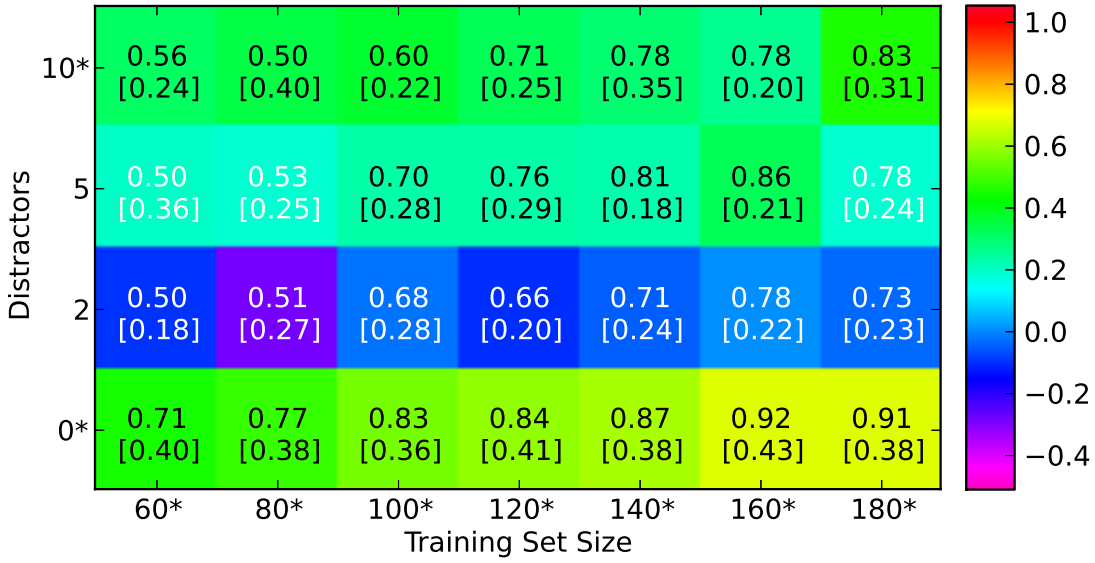


(b) SMRF - EM-DD

Figure 7.29: Mean difference PSS for the *Blue above Green or Green above Blue* dataset.



(a) SMRF - TILDE



(b) SMRF - EM-DD

Figure 7.30: Mean difference PSS for the *Red above Green or Green above Blue* dataset.

above Blue dataset. As with the previous three datasets, TILDE handles distractors poorly, and so all rows are significant except for 0 distractors, and all columns are significant.

This is a harder problem than the previous dataset, and as such, SMRF enjoys

better performance at 0 and 10 distractors, which are significant. However, EM-DD performs better than SMRF at 2 distractors for smaller training set sizes. At 5 distractors, SMRF seems to enjoy an advantage of EM-DD, but not enough to be significant with the Bonferroni correction. As with the previous dataset, SMRF enjoys an overall advantage at each training set size, as each column is significant.

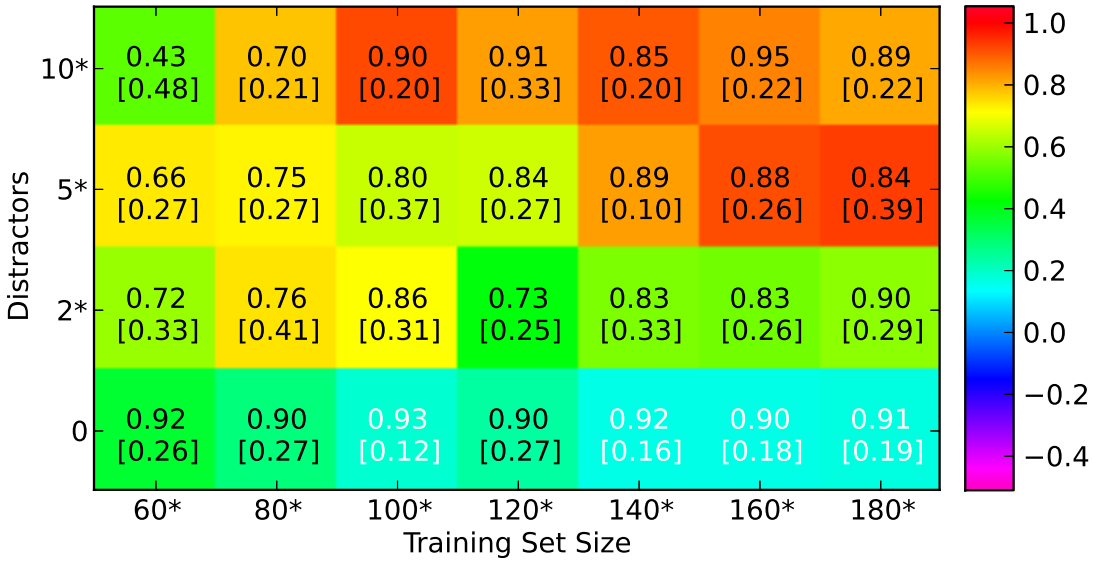
Figure 7.31 shows the performance differences for the *Red or Green* dataset. As with the previous four datasets, TILDE handles distractors poorly, and so all rows are significant except for 0 distractors, and all columns are significant.

This is the hardest problem presented yet, and as such, SMRF enjoys the best overall performance yet relative to EM-DD. SMRF enjoys a clear overall performance advantage at all distractor levels except one, as the 2, 5, and 10 distractor rows are significant. SMRF also enjoys a clear overall advantage at each training set size, as each column is significant.

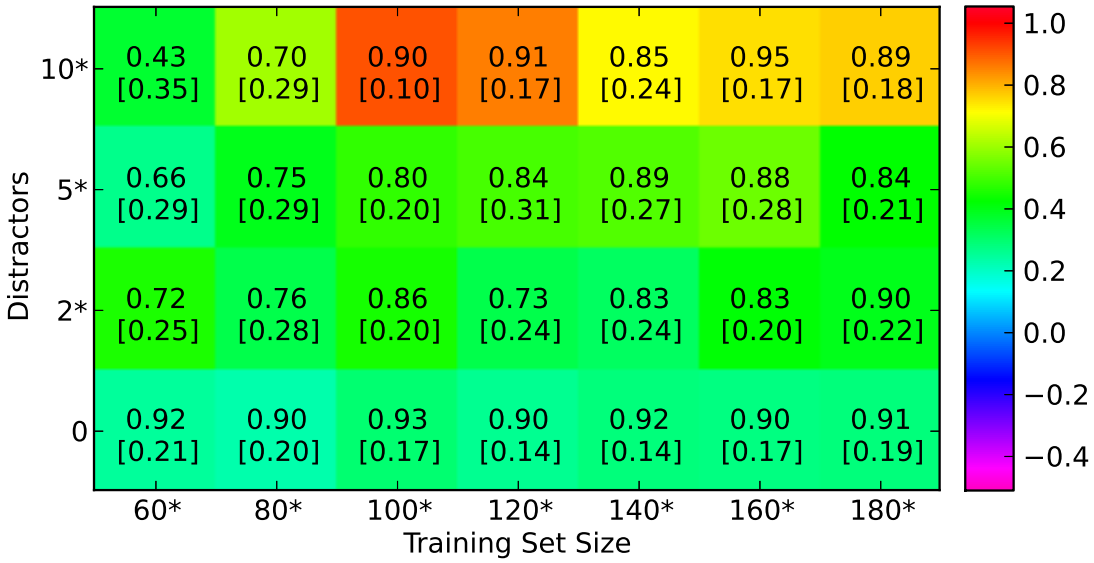
Figure 7.32 shows the performance differences for the *Red above Green or Blue above Yellow* dataset. As with the previous five datasets, TILDE handles distractors poorly, but no as poorly relevant to SMRF as on the previous datasets. As such, the 5 and 10 distractor rows are significant, though not the 0 and 2 distractor rows.

This is a harder problem than the previous dataset, and at this point, SMRF performance begins to decline relative to EM-DD performance, due to the difficulty of the problem. However, at 10 distractors, SMRF still enjoys an overall significant advantage. SMRF seems to enjoy an advantage at 0 distractors, but not enough of an advantage to be significant. At 2 distractors, SMRF struggles to stay even with EM-DD. At 5 distractors, SMRF seems to enjoy a slight advantage for training set sizes 80 and larger. However, the overall effect is not large enough to be significant, given the Bonferroni adjustment. However, SMRF enjoys an overall advantage at each training set size, as each column is significant.

Figure 7.33 shows the performance differences for the *Red or Green or Blue or*

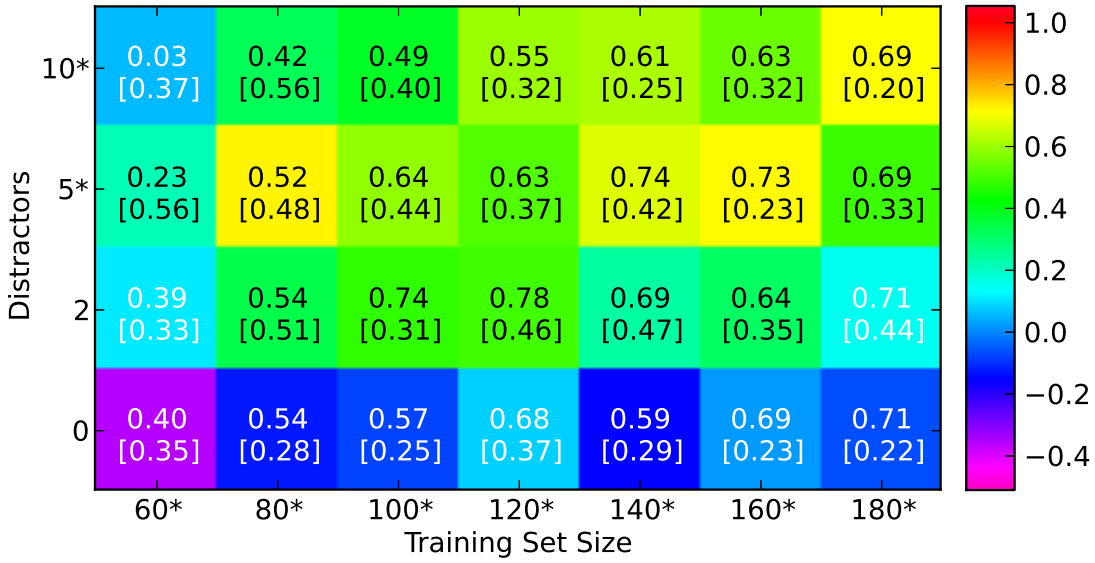


(a) SMRF - TILDE

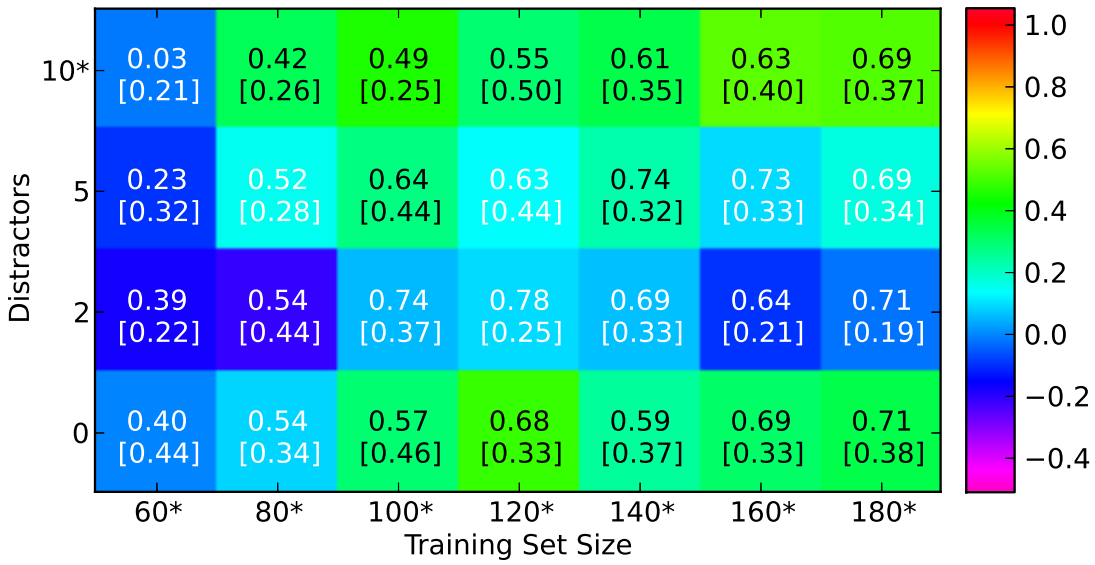


(b) SMRF - EM-DD

Figure 7.31: Mean difference PSS for the *Red or Green* dataset.



(a) SMRF - TILDE



(b) SMRF - EM-DD

Figure 7.32: Mean difference PSS for the *Red above Green or Blue above Yellow* dataset.

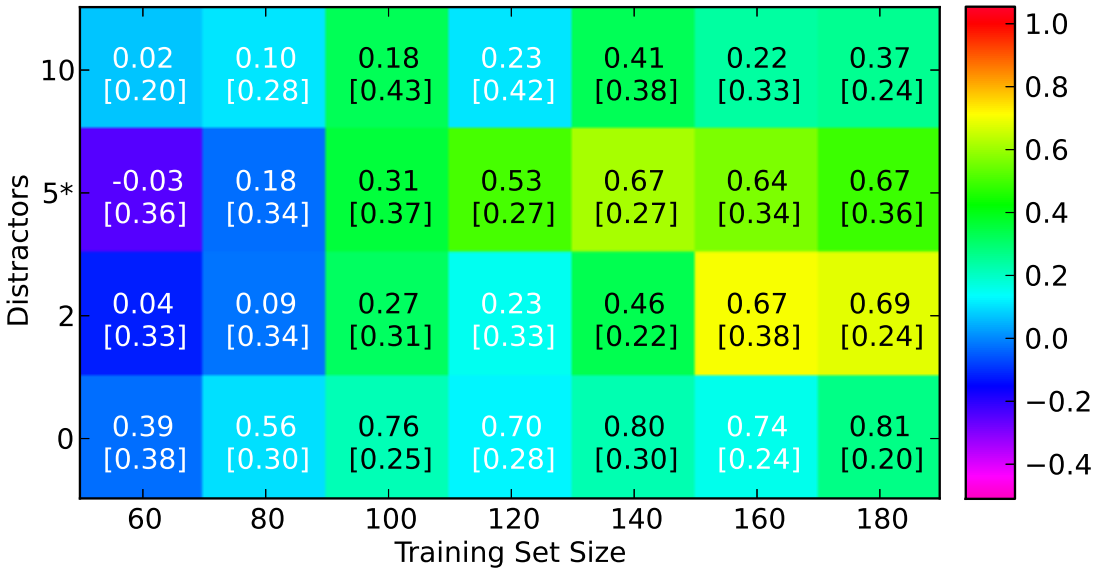
Yellow dataset. This dataset is by far the most difficult for all algorithms involved, and as such, each performs relatively poorly, especially at lower training set sizes. With respect to TILDE, SMRF seems to enjoy an overall advantage at each distractor level at training set sizes greater than 80. However, only the row corresponding to 5 distractors is significant, given the Bonferroni correction. No columns are significant in this case.

Likewise, with respect to EM-DD, SMRF seems to enjoy an overall advantage at each distractor level at training sizes greater than 80. However, as with TILDE, only the row corresponding to 5 distractors is significant. Additionally, no columns are significant in this case.

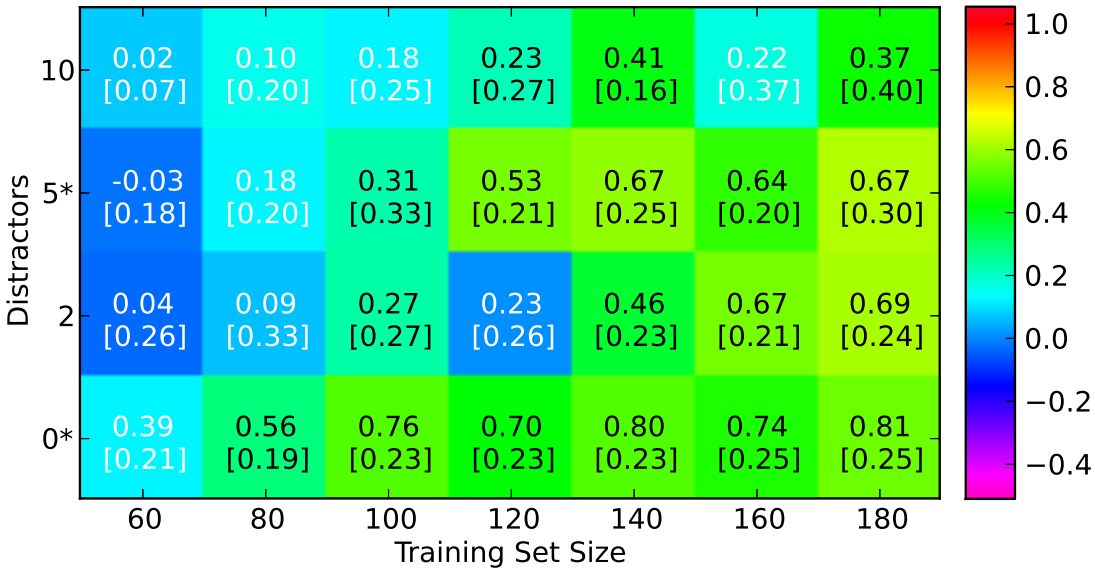
There are several overall trends to be observed from these results. First, there are no instances where a significant row or column difference arises because another algorithm performs significantly *better* than SMRF overall. While there are a few places where SMRF performs worse than either TILDE or EM-DD with a smaller training set size, there are no trends that indicate SMRF systematically performs worse for a given training set size level, or for a given distractor level. On the contrary, the significance of the rows and columns in many of the figures shows that SMRF often performs better than its competitors overall.

Second, TILDE has significant trouble coping with larger numbers of distractors, relative to SMRF. While TILDE is able to perform comparably to SMRF without any distractors, it is no match for SMRF once distractors are added. The only exception to this rule is the *Red or Green or Blue or Yellow* dataset. However, this dataset is so difficult, that all algorithms perform relatively poorly, especially with distractors and a small amount of training data. As such, the results presented in Figure 7.33 seem to be influenced to a large degree by floor effects. Even so, however, SMRF still outperforms the other approaches at multiple distractor levels.

Third, using column significance as an indicator, there is a "plateau" of sorts in



(a) SMRF - TILDE



(b) SMRF - EM-DD

Figure 7.33: Mean difference PSS for the *Red or Green or Blue or Yellow* dataset.

dataset difficulty, in which SMRF significantly outperforms EM-DD overall at each training set size level. This "plateau" occurs after the first two datasets (which are the easiest), and the last dataset (which is the hardest). Within the plateau, performance seems to reach a peak at the *Red or Green* dataset (Fig. 7.31), in which SMRF significantly outperforms EM-DD both for all rows and all columns. This suggests that for datasets that have at least a moderate amount of difficulty, but are not too difficult, SMRF outperforms both competitor algorithms overall for any amount of training data greater than 30 positive and 30 negative examples.

Fourth, for these "middle of the road" datasets – that is, all but the easiest two datasets and the most difficult dataset, SMRF performs significantly better than EM-DD at 10 distractors, and sometimes also at 0 distractors, but not at 2 and 5 distractors. The only exception is *Red or Green* (Fig. 7.31), for which SMRF significantly outperforms EM-DD at all distractor levels. This performance pattern seems to suggest that for "middle of the road" datasets in which SMRF performance does not completely dominate the performance of the other approaches, SMRF tends to struggle more than EM-DD at small training set sizes for intermediate numbers of distractors. However, at 10 distractors, SMRF performs better than EM-DD, even with small training set sizes considered, which suggests that SMRF is better than EM-DD at coping with large numbers of distractors at training set sizes both small and large.

These four trends, taken together, suggest that SMRF does not suffer any significant performance disadvantage for smaller amounts of training data, beginning at 30 positive and 30 negative examples, particularly when various amounts of distractors are factored in. On the contrary, SMRF generally outperforms TILDE for any amount of training data when distractors are present. Likewise, SMRF also outperforms EM-DD at 10 distractors, as well as at each training set size for "middle of the road" datasets. As such, not only does SMRF not suffer any such significant perfor-

mance disadvantage for smaller amounts of training data, but it often outperforms its competitors overall.

Chapter 8

Discussion

In this chapter, the relationship of the SMRF approach to some of the approaches described in Chapter 2 is discussed, as well as the the implications of the experimental results presented in Chapter 7, in light of those relationships.

8.1 Relationship to Other Approaches

The SMRF approach, as described in Chapters 4 and 5, bears various similarities and differences to the approaches described in Chapter 2. These similarities and differences render the SMRF approach a unique contributor the field. SMRF distinguishes itself by uniquely being a relational tree-based learning framework that can directly model covariant continuous multidimensional quantities, and infer significant relations from a set of attributed objects. By utilizing incremental instantiation, SMRF trees effectively prune away a large section of the instance space, enabling the framework to learn concepts defined in terms of a larger number of objects than traditional feature-based learning framework are capable of straightforwardly dealing with.

8.1.1 Tree-based Learning

SMRF bears a number of similarities to the approaches discussed in Chapter 2.1, not least of which, because SMRF is itself a tree-based learning approach. Tree-based concept representation has the advantage of being human readable, and also provides a compact means of representing concepts that are defined in terms of multiple distinctions. The SMRF approach was designed to leverage these qualities and apply

them to the domain of spatiotemporal multidimensional relational concepts. As discussed below, the problem domain for which SMRF is designed provides for many of the differences between SMRF and other tree-based learning approaches.

As discussed in Section 2.1.1, decision trees are able to provide class-label classification for individually labeled instances. Each instance is a set of attribute-value pairs. More rudimentary decision tree learning algorithms, such as **ID3**, can only build decision trees over categorical variables. More advanced decision tree learning algorithms, such as **C4.5**, are able to utilize continuous variables as well. However, continuous values are generally utilized by partitioning the space of continuous values according to a single real-valued threshold. More complex partitions of the space can only be accomplished by employing multiple such distinctions. Additionally, only single-dimensional continuous values are supported, and as such, any multidimensional values must be divided into their component dimensions.

Like **ID3** and **C4.5**, the SMRF algorithm is able to learn representations in terms of categorical variables. However, unlike **C4.5**, the SMRF algorithm is able to learn decision volumes over multidimensional spaces (for single distinctions). Besides providing a cleaner, more compact, and more intuitive representation in higher-dimensional spaces, this also allows for covariance to be accounted for, which generally cannot be accomplished simply by accumulating univariate distinctions over each of the individual dimensions of the space.

Decision tree learning algorithms, such as **ID3**, utilize information gain as a means of scoring candidate tree expansions. As demonstrated in Appendix A, the SMRF learning algorithm uses essentially the same criterion for scoring candidate tree expansions. Though both approaches use the same underlying metric, the SMRF learning algorithm was not designed *a priori* around information gain, but rather around the grand metric L (Eq. 5.2), the motivations for which are explained in Chapter 3.

Another difference between SMRF trees and traditional decision trees is that the

former do not take instances as learning algorithm input, but rather construct instances from objects *via* instantiation (cf., Section 4.1). Unlike decision tree problem domains, which consist of a set of homogeneous instances, SMRF problems consists of possibly-heterogeneous group of objects (i.e., different groups may have different numbers of objects), which themselves contain possibly-heterogeneous attributed objects (i.e., different objects might have different numbers and types of attributes). In many real-world scenarios, sensor failure, object occlusion, or other factors might prevent some object attributes from being observed during data collection. The possible heterogeneity of objects in SMRF datasets allows for such factors to be accounted for in a straightforward manner.[1] Given the possibly-heterogeneous nature of the attributed-object data, SMRF trees construct the equivalent of decision tree instances through a combination of object instantiation in instantiation nodes (Sec. 4.1), and object attribute selection in mapping functions (Sec. 4.2). This incremental construction of instances allows for uninformative attributes to be ignored, allowing the learning algorithm to focus on building decision volumes around the values of a particular attribute, or subset of attributes. Moreover, the implicit construction of instances allows the SMRF learning algorithm to focus in on those particular instances which exemplify the target concept, and build decision volumes to exclude those that do not. This allows SMRF to solve relational problems with multiple-instance aspects that are defined in terms of groups of attributed objects.

As discussed in Section 2.1.2, probability trees (or probability estimation trees) are an extension of decision trees. Whereas decision tree leaf nodes provide a single class

[1]None of the experiments described in Chapter 7 utilized heterogeneous data in this sense. As such, the argument being made here is that the SMRF learning algorithm can accept heterogeneous data without any modifications being made to the algorithm or the data. How well the algorithm would perform given such data still remains to be experimentally verified. However, Palmer et al. (2012) performed experiments with datasets that contained examples with varying numbers of objects. The SMRF trees in that context made heavy use of error branches, as would trees in a heterogeneous attribute context. Despite using error branches, the learned trees were still able to perform well in testing. Such results suggest that data with heterogeneous attributes would not be an insurmountable challenge for the learning algorithm to overcome.

label, probability trees provide a probability distribution over the possible class labels. SMRF trees are similar to probability trees in that SMRF leaf nodes also contain an implicit probability distribution. Each SMRF leaf node contains the probability that graphs "won" by that leaf (cf. Section 5.3) are positive examples of the target concept. Since the problem domain SMRF is designed to address only concerns binary class labels, and since the SMRF node leaf probability represents the probability of the positive class, the negative class probability can be easily obtained by subtracting the leaf node probability from one. Being restricted to binary labels means that SMRF lacks the ability to learn some problems in the decision tree / probability tree domain. However, a substantial number (if not an overwhelming majority) of interesting problems can be defined in terms of a binary class label, so this proves to be of little disadvantage in practice.

Since both decision trees and probability trees can only learn from instances, any relations to be learned must first be propositionalized into an appropriate set of instances. As discussed in Section 2.1.3 Relational Probability Trees (RPTs) overcome this constraint, enabling the representation of relational data within the probability tree structure. Spatiotemporal Relational Probability Trees (SRPTs) extend the RPT framework with the ability to represent spatiotemporal relations.

SMRF trees have a lot in common with SRPTs, as the SMRF approach was developed in order to solve some of the same problems as SRPTs, but in a more general and representationally-powerful way. In particular, it is much easier to collect object-attribute data in various physical real-world domains (such as robotics, or meteorology) than it is to collect object-relation data. In most cases, relations, unlike attributes, cannot be directly observed, and must be inferred. Such inference is often performed by a human judge, who either labels relations directly, or creates specific definitions for relations based upon the available data. Many useful and interesting relations exist implicitly between the attributes of various objects in the real world.

Such relations play an important role in various target concept definitions. However, since such implicit relations are not directly observable in the physical domain, it makes sense for learning algorithms to be designed to learn to infer these relations directly from the attributes of the objects involved. Such inference of relations represents a cleaner and more direct path from observation to learning, and minimizes the role of a human teacher in the process.

As such, the SMRF learning algorithm is designed to infer useful properties[2] and relations from the attributes of the objects themselves. This distinguishes the SMRF approach from RPT and SRPT approaches, which are designed to learn concepts in terms of pre-defined relations.

The ability to infer useful and relevant properties and relations from object attributes is an advantage for learning concepts that are defined in terms of physical real-world quantities. On the other hand, SMRF is not designed to learn concepts in terms of datasets that do have pre-defined relations. Such datasets most naturally arise where relational databases are involved, whose records often include a mixture of categorical and uni-dimensional continuous variables. For such problem domains, approaches such as TILDE work quite well.

As discussed in Section 2.1.4, TILDE is a top-down induction learning approach for constructing first-order logical decision trees (FOLDTs). Like decision trees and probability trees, FOLDTs can sort data based upon both categorical and uni-dimensional variables. Like decision trees, multi-dimensional variables cannot be utilized directly, but must be broken into their component dimensions. As with decision trees, this limitation prevents covariance in multidimensional data from being properly taken into account.

Approaches such as TILDE are designed for a different (though related) problem

[2]In this sense, "property" is defined as an arity-1 relation, which is defined along a certain range of values for a particular attribute.

domain than that for which SMRF is designed. Relational databases often do not contain data that would require a multidimensional representation. Datasets drawn from the observation of physical (or simulated) real-world quantities, on the other hand, do require a multidimensional representation. TILDE is designed for problems of the former kind, not the latter, while SMRF is vice-versa. As such, comparisons between the two have some usefulness, but the two approaches are not entirely commensurate.

TILDE and SMRF are also similar in the fact that both are "existential" approaches – that is, both define question node distinctions in terms of variables that correspond to various objects in the data. TILDE explicitly represents variables, while SMRF indirectly represents variables through the instantiation sequences and mapping functions (cf. Chapter 4).

Additionally, TILDE is a group-sorting approach, while SMRF is an instance-sorting approach. SMRF trees sort each individual instantiation sequence, regardless of bag membership, solely on the basis of its evaluation on the question node models. In contrast, TILDE sorts each group of objects (as a whole) through the tree, on the basis of whether or not there exists a mapping of objects to variables that satisfies the question node query. For TILDE, each group is represented in one leaf, while in SMRF, a group may be represented in a number of leaves. Because a group may be represented in many SMRF leaf nodes, SMRF trees define a "winning leaf" (cf., Section 5.4) for each group, so that each group can be associated with a single probability.

In summary, SMRF is a tree-based learning approach, and is similar to various prior tree-based learning approaches in a number of different ways. However, SMRF is distinguished from these approaches by the following qualities. First, SMRF can directly utilize multidimensional variables, allowing covariance in the data to be modeled. Second, SMRF can infer useful relations directly from the attributes of the objects themselves, which is important for problem domains that involve the collection

of data from the observation of physical (or simulated) real-world quantities. Third, SMRF does not require a set of pre-processed instances as input, but rather constructs its own instances through instantiation, allowing it to solve relational problems that are defined in terms of groups of attributed objects.

8.1.2 Multiple Instance Learning

SMRF is designed to solve problems that have multiple instance characteristics, and as such is related to various multiple-instance learning (MIL) approaches discussed in Chapter 2.2 in a variety of ways.

As discussed in Section 8.1.1, while SMRF is capable of solving problems with multiple instance characteristics, it is not designed to be an MIL algorithm, *per se*. Rather, it is designed to solve classification problems where each bag is implicitly defined in terms of a labeled group of attributed objects, and since the bag members are neither explicitly enumerated nor labelled, the process of determining which instantiation sequences satisfy the target concept is multiple-instance in essence.

The traditional MIL explicitly enumerated instance vector representation is a special case of the SMRF attributed-object-group representation, as an instance vector (which is an ordered list of attribute values) can be represented as an object with multiple attributes. Care must be taken in such cases, however, as such objects do not generally represent "real-world" objects, and relationships between such objects have meaning and significance that is questionable at best. For example, if SMRF were to be tested on the Musk dataset, each instance could be represented as an object with 163 attributes, but SMRF would not be allowed to infer relations between objects, since the objects do not actually exist together in the physical domain in a way such that relationships between them could legitimately inferred.

As such, SMRF is designed to solve a different set of problems, with a different set of data collection and processing procedures, than those generally addressed by

traditional MIL approaches. Traditional MIL approaches, when applied to real-world problem, are often paired with domain-specific feature extraction procedures. For example, the famous MUSK dataset is provided as the result of an esoteric feature-extraction procedure. On the other hand, various image-classification approaches discussed in Section 2.2.2 focused on using the same underlying diverse-density approach on the same or a similar set of images, but with different feature extraction algorithms. In traditional MIL approaches, feature extraction plays a critical role in the success of the learning algorithm.

In contrast, SMRF is designed to learn concepts in terms of data where the features are the values of the observable physical quantities themselves, such as position, orientation, and color. This provides a cleaner and more compact route from observation to classification. Additionally, defining features in terms of observable physical quantities avoids possible biases introduced by various feature algorithms. As such, the SMRF algorithm is more naturally suited towards solving problems that are defined in terms of physical (or simulated) real-world objects, as opposed to problems that are defined in terms of datasets with pre-generated features (such as MUSK). As a result, one might naturally expect SMRF to perform better on real-world-object-based datasets than traditional MIL algorithms, and for traditional MIL algorithms to perform better on feature vector-based datasets than SMRF. However, if SMRF were provided with the data used to construct those feature vectors in the first place, it is not unlikely that SMRF could identify important properties and relationships in the data, and perform comparably to traditional MIL algorithms on those problems. SMRF is designed to identify important attributes and relations in the data, and can do this much more effectively if it given the raw data itself, as opposed to a set of relational values that have been drawn from the data by a separate heuristic. This makes the SMRF algorithm more powerful than many traditional MIL-algorithms in this respect, but it also makes a fair comparison to traditional algorithms on standard

benchmark datasets next to impossible to perform, as the original data from which the feature vectors were heuristically generated is not publicly available for most of the standard benchmark datasets.

Furthermore, since the SMRF algorithm constructs instances through instantiation, irrelevant regions of the instance space can be pruned away by sorting the instantiation sequences which represent those regions into low-probability leaf nodes. This allows the relevant regions of the instance space to be identified and isolated, and potentially makes the search for the target concept more efficient.

In other areas, SMRF is similar to traditional MIL approaches, as they generally are capable of dealing directly with multidimensional values. Most traditional approaches, however, do not attempt to model or account for covariance in the training data. This is not entirely unexpected, however, as most traditional MIL problems are not defined in terms of covariant properties or features.

In addition, most traditional MIL approaches are not designed to utilize relational data, or infer relations from object attributes. Once again, this is not unexpected, as traditional MIL problems are generally not explicitly relational in nature. Problems that have relational aspects are generally dealt with at the level of feature extraction, not at the level of the learning algorithm itself. As a result, for the experiments discussed in Chapter 7, the relations between all pairs of objects had to be explicitly enumerated to enable an appropriate comparison between SMRF and traditional MIL algorithms on the datasets described in Section 7.1.2. Simply providing these algorithms the same kind of data that is supplied to SMRF[3] would not have resulted in a fair comparison, as the other algorithms would have had no way of inferring relational values on their own. However, as discussed in Section 7.1.3, explicitly enumerating all pairwise relations greatly increased the size of the datasets for these

[3]This could be accomplished to some degree by providing one feature vector per object, where each feature vector simply contains the appended attribute values of that object.

algorithms, especially at higher numbers of distractor objects.

In addition to general similarities and differences, SMRF can also be related to specific approaches discussed in Section 2.2. The Axis-Parallel Rectangle (APR) method (Sec. 2.2.1) builds representations by enclosing areas in the feature space that distinguish positive bags from negative bags. The SMRF algorithm does something related, as it builds decision volumes in object property and relation spaces. Being a tree structure approach, these volumes can be configured in the tree to represent both conjunctive and disjunctive concepts. As such, a purely-conjunctive SMRF tree[4] that asks questions only about individual object properties (and not relations) functions something like the APR method. However, even in such a non-relational case, the use of probability density functions in the definition of the decision volumes allows for property and relation covariance to be modeled explicitly, which is a significant difference. Moreover, the tree-structured nature of SMRF models allows easy representation of disjunctive concepts, which the APR models are not designed to do.

The SMRF approach is related to diverse-density approaches (Sec. 2.2.2) in that both employ Gaussian functions to define the target concept. SMRF utilizes Gaussian density function to define decision volumes, while DD and EM-DD use a Gaussian function to define a model which is used for the computation of diverse density. On the other hand, while SMRF makes use of Gaussian densities for modeling various properties and relations, SMRF is not restricted to using Gaussian densities. Quantities such as planar orientation, for example, can be modeled by SMRF using a von Mises distribution. Moreover, while DD and EM-DD are limited to finding a single volume in the feature space, SMRF trees, through multiple question nodes, can construct multiple decision volumes to facilitate the representation of disjunctive target concepts.

The approaches discussed in Section 2.2.3 are based upon the k-nearest neigh-

[4]That is, a SMRF tree with no question nodes on a No branch.

bor (kNN) algorithm. These approaches find clusters of points in the feature space through the use of the Hausdorff Distance metric, which defines the distance between two bags in terms of the distance between representative instances from those bags. The volume-building procedure described in Section 5.3 also utilizes a measure of "bag distance." In contrast to the kNN-based approaches, however, the volume-building procedure effectively computes the Mahalanobis distance of the representative point of a bag to the center of a model of a set of bags that is greedily grown, as opposed to computing distances between pairs of bags. The greedy volume-building algorithm has the effect of building a cluster of points in some property or attribute space. By building multiple question nodes, the SMRF learning algorithm can construct a set of clusters that are related conjunctively and/or disjunctively in defining the target concept.

Some MIL approaches, as discussed in Section 2.2.5 utilize support vector machines (SVMs). These approaches attempt to construct a hyperplane to separate positive bags from negative bags. Some approaches, like MI-SVM, are instance-based, and attempt to separate positive instances from negative instances. Other approaches, such as MI-SVM and MICA, choose representative instances from each bag and build a hyperplane to separate those. SMRF is similar to these latter approaches in that representative instances are chosen both for calculating the grand metric score L (Eq. 5.2), as well as for building decision volumes in the method described in Section 5.3. Where SVM-based approaches are limited to building one hyperplane to separate the data, SMRF trees can build multiple decision models, making it easier to represent disjunctive concepts.

Other MIL approaches, as discussed in Section 2.2.4, are tree-based learning approaches. SMRF is most closely related to the approach called MITI. In MITI, nodes are expanded in order of "best" to "worst" in terms of a sorting metric. There are three possible sorting metrics, the most simple of which is $\frac{p}{n+p}$ (where p denotes

the number of positive bags present at a node, and n denotes the number of negative bags present). One will note that, from Equation 5.7 in Appendix 5.1, this ratio also happens to be the value of SMRF leaf node probabilities that maximizes the grand metric L. As discussed in Section 5.2 chooses leaves to expand based on how well an optimal hypothetical split improves L. In this way, SMRF functions in a related way to MITI in choosing which nodes to expand.

Additionally, in choosing which candidate split to accept, MITI can utilize one of five possible splitting metrics, three of which are defined in terms of the ratio $\frac{p}{n+p+k}$, where k is a constant value arbitrarily assigned beforehand. In the case where k is zero, this reduces to $\frac{p}{n+p}$. This is also related to the SMRF optimization procedure, which chooses the candidate optimization that best significantly increases L, which is maximized by setting leaf node probabilities to $\frac{p}{n+p}$.

However, like the tree-based learning approaches discussed in Section 8.1.1, there is no indication that MITI can deal directly with multidimensional variables or infer relations directly from the attributes of objects.

In summary, SMRF is related to various MIL approaches in a variety of ways. However, SMRF is generally distinguished from these approaches by the following qualities. First, SMRF allows for observed physical quantities of real-world objects to be represented directly in an attributed-object group representation, that avoids the need for feature extraction. Second, the SMRF algorithm constructs instances through instantiation, whereas instances are typically enumerated prior to learning in traditional MIL approaches. Third, SMRF is capable of modeling the covariance of multidimensional properties, which is generally not true of traditional MIL approaches. Fourth, SMRF is capable of directly utilizing relations between objects in building target concept representations, which is generally not true of traditional MIL approaches. Those approaches that do utilize relations generally do so in the process of feature extraction, not the process of concept learning. This principle was

illustrated in the experiments described in Chapter 7, as the relations between all possible pairs of objects had to be explicitly enumerated in order to enable an appropriate comparison between SMRF and the other algorithms. Fifth, SMRF, by its tree-structured representation, can learn disjunctive concepts, which many traditional MIL approaches are incapable of doing in most cases.

8.2 Experimental Results

The experimental results in Chapter 7 demonstrate that SMRF is an effective framework for solving the kinds of problems represented by the datasets described in Section 7.1.2. While these problems have a synthetic aspect to their generation, they make use of real-world covariances in their color attribute values. These datasets represent the kinds of problems that a robotic system might be tasked with solving, in learning to perform a certain kind of task in the real world. With appropriate abstraction mechanisms in place, complex tasks, such as assembling a bookshelf, can reduce to repeated applications of relational concepts such as "this board above that board," "this board to the right of that board," and "this board oriented 90 degrees to the right of the orientation of that board." With these experimental results, SMRF has demonstrated the ability to learn these kinds of concepts, learning concepts successfully on datasets that include a mix of conjunctive and disjunctive concepts. With appropriate abstraction, planning, and execution mechanisms in place, SMRF could conceivably be utilized to learn to perform these kinds of tasks. Indeed, work in this regard has already been done in simulated reinforcement learning domains (Palmer et al., 2012).

Moreover, these results demonstrate that SMRF is robust to real-world kinds of noise. In performing real-world tasks, there will likely be some number of objects in the workspace or general vicinity of the agent that have nothing to do with the

current task at hand. Going back to the bookshelf example, if the robot is working on putting together a specific shelf, there might be other objects in the workspace, such as boards and nails, that will be used to assemble the other shelves. The ability to prune out distractor objects is important for learning agents operating in physical domains. The SMRF algorithm has shown that it is robust to high numbers and proportions of distractors. For the results presented in Chapter 7, SMRF was able to learn concepts well even when there were 5 times as many distractors as relevant objects in the scene.

Likewise, many real-world problems will face some amount of labeling noise. Sometimes, human teachers make mistakes, and mislabel examples. Or, an error in a perception system might lead a robotic agent to misinterpret the intended label of an example. However it occurs, corruption of the training labels is a phenomenon that must be accounted for by a real-world learning agent. As with distractors, SMRF has proven its robustness to corruption of the training labels. Even at 12% corruption (which is more than would likely occur in normal situation), SMRF still maintains a mean PSS of 0.8 across all of the datasets, at the zero-distractor level.

In comparison to other algorithms, SMRF is much better at handling the combination of complex relational concepts, covariant attributes, and significant amounts of noise, in terms of distractors and corruption. The most surprising result in this regard is the performance of EM-DD, which outperforms the other comparison algorithms in general. IAPR, whose performance on the MUSK datasets sets a benchmark for MIL literature, seems to be unable to cope with the covariant nature of the color attributes. Even with no noise, it is unable to get beyond a mean PSS of 0.06 across all datasets.

Likewise, TILDE performs more poorly than expected, as it is unable to cope with an increased number of distractor objects. At 10 distractors, it struggles to maintain a positive mean PSS, and its performance at 2 and 5 distractors is not much better

than random.

Additionally, TILDE tends to produce trees that have five to ten times as many questions nodes as are present in the comparable SMRF trees. As part of its optimal parameter set (Sec. 7.1.3), TILDE was run with C4.5 post-pruning enabled. Yet, even with post-pruning enabled, TILDE produces trees that are substantially larger than their SMRF counterparts. As more compact hypothesis representations are more preferably, all things being equal, SMRF has a distinct advantage in this regard. However, all things are not equal in this case, as SMRF substantially (and significantly) outperforms TILDE in many cases with respect to test set classification. As such, it would appear that TILDE has significant issues with overfitting on these kinds of data, especially as distractor objects are added and corruption is increased.

In general, the comparison algorithms had trouble coping with larger amounts of distractors, in both runtime and performance. As discussed in Section 7.1.3, this was due in part to the fact that these algorithms are not designed to infer relations from a set of objects and attribute values. This limitation forced the creation of feature vectors that correspond to the attributes and relational values for all 2-permutations of the objects in the set. Such a procedure poses a significant drawback for the use of these approaches in contexts where data is presented to the learning agent in object-attribute form. The enumeration of such features quickly becomes intractable as ternary relationships are introduced (requiring the enumeration of all 3-permutations in addition to the enumerated 2-permutations) and as the number of objects in each example grows.

SMRF is able to easily handle data with groups containing upwards of 100 objects each in a computationally tractable manner, learning trees that model the target concept in only a few minutes in some cases.[5] SMRF has this ability in part due to

[5]This claim follows from the results of exploratory tests performed outside of the context of the experiments discussed in Chapter 7. In these tests, SMRF was able to learn trees containing 2-3 question nodes in around 10 minutes when trained on the *Blue above Green* dataset with 100

the auto-pruning procedure discussed in Section 7.1.1.

In contrast, approaches such as TILDE were unable to learn concepts with 20 distractors in a reasonable time frame using a reasonable amount of system resources. For a two-object concept with 90 positive and 90 negative training examples, TILDE requires the generation of $(90+90)*(20+2+_{(20+2)}P_2)$, or over 87,000 Prolog facts at 20 distractors. In this case, the large number of features and facts required to represent the instance space proved to be too large to facilitate learning in a reasonable amount of time. Likewise, for the same concept, the other algorithms require the generation of $(90 + 90) * {}_{(20+2)}P_2$, or over 83,000, instances.

Moreover, for a 2-object concept with 100 distractors, and 180 training examples, the other algorithms would require the generation of approximately 1.9 million features or Prolog facts, respectively, which is computationally infeasible for these systems. On the contrary, SMRF would prune out all but single-object questions for the first expansion, and, as such, would only generate $180*(100+2)$, or 18,360, instantiation sequences for the first expansion, most of which would be pruned away into a low-probability leaf. A much smaller subset would then be used to build two-object instantiation sequences, which would then be further pruned by two-object questions, and so forth. As such, SMRF enjoys a distinct advantage to these other algorithms, given its ability to prune away the majority of irrelevant object permutations before they are ever explicitly enumerated. This allows SMRF to learn representations of target concepts in instance spaces that are much too large to fully enumerate in practice.

distractors.

Chapter 9

Future Work

In this chapter, some possible directions of future work are presented. These suggestions fall into four main categories: extending the representational capabilities of SMRF trees, adding regression to the SMRF framework, extending SMRF for use in human-robot interaction, and applying SMRF to novel problem domains.

9.1 Extending the Representational Capabilities of SMRF Trees

As described in Chapter 4, SMRF Trees provide a powerful framework for representing relational multidimensional concepts. However, this framework can theoretically be extended in a number of ways to allow its application to more kinds of problems.

Prior to discussion specific extensions to the SMRF algorithm, it should be noted that it is relatively easy to extend the representational capabilities of SMRF with respect to new types of attributes and relations. To represent a new attribute, one simply has to define a new identity mapping function defined over that attribute. Additionally, representing relations in terms of new attribute types can be accomplished by adding new mapping functions. For relations defined in terms of a difference vector (such as "Above") or in terms of Euclidean distance (such as "Near"), creating a new mapping function would simply entail combining an attribute template for the new attribute type with the existing model functions that compute the difference vector and Euclidean distance, respectively. More complex mapping functions can be

defined through the addition of custom model functions. As such, a large number of possible future extensions to SMRF would not involve any changes to the present existing framework, as they would only require the addition of new mapping functions. Extending SMRF to handle temporal data, for example, would only require changes of this type, as discussed below.

First, SMRF could be extended by representing temporal data. Technically, this requires no extension to SMRF *per se*. Rather, temporal data could be represented by modifying data representation conventions. A "static" graph describes a set of objects and their attributes at a given point in time. Such a representation can be upgraded to a "dynamic" mode by representing each object as a set of multiple objects, one for each period of time. Each graph object would contain a set of attributes describing its physical properties, as well as a temporal attribute denoting the time at which those properties obtained. To ground a notion of identity through time, each object would also need an "identity" or "identifier" attribute, that would allow the system to associate all of the different states (or configurations) of the same object together.

Given this upgraded representation (which simply adds two attributes to each object in the graph), SMRF would simply need to be provided with appropriate mapping functions that allow temporal relationships to be represented. Given appropriate mapping functions, the learning algorithm should not need to be modified in order to learn temporal concepts represented in this way. Such a representation would allow SMRF trees to represent events, if one defines an event as a specific change in an object (or configuration of objects) over a specified course of time.

Second, SMRF could be extended by introducing a notion of "meta-instantiation," where a certain number of objects are instantiated as a group. Providing such functionality would allow SMRF to do things such as count the number of objects which have a certain property. As such, this could help SMRF be able to solve more complex problems whose definitions rely upon a notion of grouping.

Such an extension faces a number of difficulties, not the least of which is determining which objects should belong to which group. One solution would be to couple the functions of an instantiation node and a question node together, such that the meta-instantiation node chooses all objects in the graph that satisfy a particular question node model. However, meta-instantiations would need to be followed by question nodes that make distinctions based upon group properties. The selection of meta-instantiation-question expansions would thus require something of a 2-ply search of the space of questions, which could greatly increase the learning time of the algorithm. As such, meta-instantiations remain an extension for which it is not clear if the pros outweigh the cons.

Third, SMRF could be extended by adding universal quantification. As it stands SMRF trees can be converted into existentially-quantified first-order logic expressions (with identity). However, universals (or negative existentials) are beyond the scope of its current representational capabilities. As such, it is not clear exactly how such an extension should be implemented. One solution would be to use a meta-instantiation that pulls in all of the objects in the graph. But such a solution runs into the problems associated with meta-instantiations discussed above.

Fourth, SMRF could be extended by adding mereological relations, allowing objects to be "composed of" or "a part of" other objects. As many physical objects are wholes composed of parts, which are themselves physical objects, such a representational capability seems to be well-suited to the kinds of problems SMRF is designed to solve.

Fifth, SMRF could be extended by allowing SMRF trees to be grown in the question nodes. Such use of trees could be useful for problems which contain multiple layers of abstraction, such as those in which mereological relations are significant. It is unclear at present, however, how to decide when using sub-trees would be better than simply growing a larger "flat" tree. Likewise, learning sub-trees could greatly increase

SMRF learning time (for the top-level tree), and could be liable to representational inefficiencies, such as duplicate questions in separate sub-trees.

Last, SMRF could be extended to provide regression capabilities, in addition to its current classification capabilities. This could be accomplished by augmenting leaf nodes with regression models. Regression models would themselves be a modified form of question node models. A regression model would take an instantiation sequence as input, and provide a real-valued number or vector as output. Such a value would denote the value of an attribute of interest with respect to the current graph. For example, in a graph that features two objects in a stack, the predicted value might denote the location at which a third object should be placed in order to vertically extend the stack.

In order to provide regression capabilities, graphs would need to encode this additional target information. Specifically, each graph would need to supply a target value, and target attribute type. Theoretically, multiple target value-type pairs could be supplied, and multiple regression models could be learned at each leaf, where each model corresponds to a value-type pair.

Adding regression would require a significant modification to the SMRF learning algorithm. First, the likelihood function L would need to be extended to include elements corresponding to the prediction accuracy of the regression nodes. Depending on how this accuracy is represented, this could drive L negative, which could be somewhat counter-intuitive. Second, the expansion process of the learning algorithm would need to be updated so as to pick regression nodes at the leaves, in addition to question nodes in the questions. Doing these two processes separately would lead to a 2-ply search, which could slow down the learning algorithm. Third, a regression node model optimization algorithm would need to be developed in order to select the best parameters for the regression node models. As such, adding regression capabilities would not be a trivial process, but could pay dividends in the long run.

9.2 Human-Robot Interaction

One field in which SMRF could be applied for future work is that of Human-Robot Interaction (HRI). Robots are becoming an ever more ubiquitous part of human life. With the ability to perform a number of useful tasks, robots have the potential to help humans in a number of ways. From performing mundane, common, tasks, to helping with very specialized procedures, robots have the capability to increase the human quality of life. However, a highly-capable robot is not very helpful if it cannot be easily used and configured by its human owners. Thus, the development of a means by which robots can effectively interact with humans becomes important. The emerging field of *Human-Robot Interaction* (HRI) (c.f., Burke et al., 2004) attempts to address these issues by studying how humans understand, develop, and use robotic systems.

HRI is a field that addresses the interactions between humans and robots. The field is relatively broad, addressing concepts such as various degrees of robot autonomy (from human teleoperation to complete autonomy) (Parasuraman et al., 2000), information exchange between robots and their operators, and the development of methods by which humans can train robots. HRI is utilized in a wide variety of domains, including search and rescue (Blitch, 1996), assistive robotics (Roy et al., 2000), educational robotics (Cooper et al., 1999), entertainment (Jonathan et al., 2000), military applications (Bruemmer and Walton, 2003), and even space exploration (Leger et al., 2005). This wide variety of applications demonstrates the growth of interest among researchers in studying, utilizing, and improving human-robot interaction as the level of robotic technological sophistication has increased. While the application of HRI to each of these domains has the potential to improve human quality of life, marked improvements can be made in the area of assistive robotics. In particular, some progress has been made in developing robotic technologies to assist the elderly

(e.g., Montemerlo et al., 2002; Pollack, 2005; Roy et al., 2000). Particular examples include approaches to provide navigation and mobility assistance (Yanco, 2001), cognitive assistance (Dieter et al., 2002), and even emotional welfare (Wada et al., 2002). Assistive technologies for the non-elderly include developments such as navigational assistance for the visually impaired (Kulyukin et al., 2006; Shoval et al., 1994), as well as navigational assistance in certain structured environments, such as grocery stores (Kulyukin and Gharpure, 2006). As this research demonstrates, HRI has a potential to develop technologies that will improve human quality of life.

Possible future applications of SMRF are concerned with the particular emphasis within HRI of how robots can be taught by humans to perform various tasks. This ability to teach robots to perform various tasks is important if robots are to be widely used to perform a number of everyday tasks. This is because humans are not, in general, computer programmers capable of programming a robot to perform specific tasks of the kind that HRI has already produced. Rather, teaching others to perform various tasks is a skill that comes naturally to humans.

9.2.1 Socially-Guided Machine Learning

A generalization of this idea of teaching robots to perform tasks is discussed by Thomaz (2006), and termed *Socially-Guided Machine Learning* (SG-ML). In contrast to the traditional machine learning (ML) paradigm, in which a computational agent learns a policy or concept purely from the data itself and without any human interaction, the SG-ML paradigm includes human input in the learning process. Specifically, the paradigm accepts human input to guide the learning process, and provides feedback to the human teacher, enabling the human to observe the state of the learning agent and to appropriately adjust the teaching input.

Thomaz discusses findings from developmental psychology and notes that three significant factors are present in situations where parents successfully teach their chil-

dren new skills and concepts. The first key factor in the guided learning process is *attention direction*, in which parents situate themselves and any objects involved in their teaching procedures to be within the gaze and reaching distance of the child. In addition, they actively guide the child's gaze by gestures and their own gaze direction. This has the effect of cuing certain objects, and signaling them to the child as significant, over and above other distractor elements in the environment.

The second key factor in the guided learning process is the use of *dynamic scaffolding*, which refers to the teacher's maintenance of the learning situation at a level of complexity that is appropriate for the child at a given point in time. The idea is that presenting the learner with situations that are too simple doesn't provide the learner with anything new to learn, but at the same time, providing situations that are too complex can make it too difficult for the learner to identify the important things to be learned. Thus, a proper pedagogical approach is essential to challenging children to advance in learning while not getting bogged down in overly complex situations before they are ready for them. In the same way, it seems intuitive that such an approach might be useful, if not critical, to the implementation of SG-ML agents, especially if the agent makes use of hierarchical knowledge. In such a case, the most basic behaviors/concepts need to be learned, and then more complex behaviors/concepts can be learned that make use of them. Thus, it is conceivable that certain complex behaviors or concepts may exist in hierarchical domains, such that a proper pedagogical approach may be necessary for an SG-ML agent to learn them efficiently, if at all.

The third key factor in the guided learning process is the linkage of old information to new information. Specifically, this refers to showing the learner how previously learned knowledge/skills can be used to understand/learn new knowledge/skills. In an SG-ML agent, utilizing this feature of guided learning entails the presence of knowledge transfer capabilities, as well as some mechanism by which certain elements of

existing knowledge can be denoted by the human teacher as relevant to the current task or concept being learned by the agent. In more complex and hierarchical domains, it makes sense that the capability to make use of previously-learned concepts and abilities, as well as the capability to be shown how those apply to the current learning problem, would be helpful in making the learning of certain concepts more efficient. Such capabilities would also enable the learning of more difficult concepts that might not otherwise be tractably learnable.

Another detail of interest from the work of Thomaz (2006) is that the feedback given to the human, by the agent, is found to be rather significant in how well the human can teach the agent certain tasks. This feedback, called *transparency*, displays some aspect of the agent's current knowledge state to the human, which allows the human to more accurately assess what next pedagogical steps need to be taken in order to better teach the concept to the agent. The more information given by the agent, the more that the human teacher can pinpoint the source of a problem that is inhibiting the learning of a concept, and thus provide feedback specifically designed to overcome that problem. Moreover, for SG-ML agents, transparency may be necessary to enable humans to properly teach skills and concepts to the agents. In experimental findings, Thomaz reports a roughly 50-100% reduction in accuracy and efficiency on one particular set of tasks when the robot stopped providing transparency to the teacher.

9.2.2 Learning from Humans

Various approaches have been proposed for developing agents that learn from human input. Thomaz (2006) identifies three broad classes of such approaches. The first is agents that learn from observing human behavior. One well-established set of approaches in this regard is called *learning from demonstration* (LfD) (Atkeson and Schaal, 1997), in which the agent learns a control policy that encodes a task, from

human demonstrations of the task. Since demonstrating new tasks to learners is an intuitive, natural way that humans teach tasks to others, it follows that an effective SG-ML agent must be able to learn from demonstrations to some degree.

The second class of approaches that Thomaz identifies is agents that learn by being explicitly directed in the task that they are to perform. This can take the form of directly controlling the agent during the learning process (Smart and Kaelbling, 2002), or of scaffolding the learning process by carefully controlling what kinds of tasks the agent is given at each step (Lin, 1992). Other approaches of this form involve teaching the robot the details of the task through natural language processing (Lauria et al., 2002; Kuhlmann et al., 2004). While there are many situations in which simply demonstrating the task to the robot would be sufficient, there are many complex, real-world situations in which the robot would have great difficulty learning a demonstrated task, especially if there are many environmental distractors present. In such cases, it stands to reasons that a more explicit approach to either teaching the task, or explicitly communicating some key details of the task to the agent, would be required.

The third class of approaches identified by Thomaz is agents that learn by receiving high-level feedback from humans. This can be as detailed as having the human directly control the reward in a reinforcement learning approach, or more generally, having the agent ask for human feedback if it is unsure of its performance on a task. An example of the latter agent is the robot Leonardo (Breazeal et al., 2004) This class of approaches is in some sense a compromise between the first two. If the agent can learn a task from demonstration and its own exploration only in certain situations, then having the extra overhead of a teaching model which explicitly directs learning would be inefficient, as well as unnecessary. However, if the agent is unable to learn a concept properly on its own, apart from explicitly-directed learning, then it makes sense that it should be able to request this information as needed.

These three classes encompass the various approaches to human-directed learning of ML agents, and demonstrate an awareness on the part of researchers that various degrees of human interaction are necessary and/or useful for the learning of social ML agents in various circumstances. The future work proposed here is concerned primarily with the development of an approach that can enable robot learning from demonstration in possibly complex and noisy environments, with the capability to utilize explicit direction in the learning process through the use of various cues, if necessary.

9.2.3 A Proposed Framework for Human-Guided Learning

To this end, the SMRF framework could be extended to allow for concepts to be learned in part from human interaction. Some work in this regard has already been done by Sutherland (2011), utilizing rudimentary *cues* for identifying important components of the target concept. The concept of cues comes into play when we realize that real-world target concepts often involve a number of objects, working together in intricately-related ways. Real-world data also often contain a significant number of distractors, and these must also be accounted for. The complex net of relations between objects found in real-world problems will be difficult to learn if only a linear search over possible sampled relations is used. Thus, what is needed is a way to guide the exploration of the learning algorithm by specifying which aspects and/or regions of the state space are important. Social cues are one such means to providing this extra information so that the learning algorithm can more easily identify the objects and relations that are important.

The cues to be utilized in this proposed future research come from two sources: gestures and linguistic utterances. *Gestural cues* involve gesturing or pointing at a particular object or group of objects. *Linguistic cues* involve providing a natural language utterance to the learning algorithm. Linguistic and gestural cues can also

be mixed. For example, the teacher could point to an object and say "This object must be red."

Linguistic cues can be subdivided into two types: *deictic cues* and *semantic cues*. A deictic cue is defined here as a linguistic utterance that exhibits *deixis*. According to Levinson (1983), *deixis* is the phenomenon whereby utterances encode meaning that is context-dependent – that is, the context of the utterance must be known in order to make sense of or disambiguate the utterance. A semantic cue is defined here as a linguistic utterance that does not exhibit deixis – namely an utterance which expresses some propositional content concerning the current task, and does not require knowledge of context to make sense of or disambiguate the utterance.

According to Levinson (1983), deictic usage can be subdivided into *symbolic deixis* and *gestural deixis*. Symbolic deixis refers to expressions that only require the general spatio-temporal context to be known. An example of symbolic deixis is "This city is beautiful." Gestural deixis, on the other hand, requires audio-visual information concerning the speech event in order to properly understand the utterance. An example of gestural deixis is "This finger is broken." The deictic cues utilized in this research exhibit gestural deixis only, and thus require some gesture or identifying audio-visual information to make sense of the deictic utterance. For example, acceptable deictic cues would be utterances of the form "This object must be red." However, to understand that utterance, the agent must know which object the phrase "this object" denotes. For this, a gestural cue is required, to physically point out which object is being referred to in the utterance. Thus, whenever the term "deictic cue" is used, it is assumed that a corresponding gestural cue is given alongside the linguistic utterance, in order to disambiguate the utterance for the learning agent.

The processing of linguistic cues poses a special challenge for computational approaches. The problem with many natural language processing (NLP) approaches is that words are ungrounded – that is, that they are defined in terms of other words.

This has a limited degree of usefulness, as the words used to communicate about things in the physical world have no inherent association to what the robot has observed in its interactions with the real world. If the language used to interact with a robot is to have any intrinsic meaning to the robot, then the words and syntax of that language must be inherently defined in – that is grounded, in sensory perception. The notion of grounding language in sensory perception, called the symbol grounding problem, was first explicitly explored by Harnad (1990), and since then, a various number of approaches have arisen to attempt to ground some aspects of natural language in sensory perception.

Cues do not necessarily provide a complete description of the target concept, but rather provide some information that can be used to provide direction for the learning algorithm. Thus, the proposed cue integration methods focus on transforming cues into information that "seeds" the initial conditions of the learning algorithm. The proposed methods by which cues will be integrated into the SMRF learning algorithm are as follows.

As discussed in Appendix E.1, the learning algorithm proposed by Sutherland (2011) relies upon maintaining a hidden variable representation h for each instantiation sequence $h(I)$. Since this algorithm already attempts to determine which instantiation sequences are correct by assigning them each a hidden variable $h(I)$, it seems natural that gestural cues could be taken into account as a prior probability over the objects in the graph, which influences the value of $h(I)$. There are a couple of options in this regard. First, the h values for objects that are cued can be "pinned" – that is, that we can require that they be instantiated at a particular point in any sequence, and that any sequence that contains only pinned objects receive an h value of 1.0. This ensures that cued objects are given a maximal weight when estimating model parameters. A second method of handling gestural cues is to use an instantiation sequence containing a cued object as a reference point for initializing the h's of

the other instantiation sequences. Given a mapping function, a kernel can be centered at the point in the mapping function codomain to which the sequence with the cued object is mapped. The h's of the other instantiation sequences can be calculated in proportion to the value assigned by the kernel to the points in the codomain to which they are mapped by the mapping function.

However, it is unclear if it is possible (or desirable) to integrate this kind of a hidden variable representation into the learning algorithm described in Chapter 5. The hidden variable approach is instance-oriented, while the current SMRF learning algorithm is bag-oriented. To what degree these two could be integrated is a problem that needs to be addressed if these methods are to be utilized in the current SMRF configuration.

Gestural cues can be collected in real-time by our object tracking system, which uses a camera, glove, and position sensor to track the position and orientation of objects on a table. The system also tracks grasp and ungrasp events that are triggered when the teacher grasps and ungrasps an object with the glove, respectively. A grasp-ungrasp event that does not change the state of an object can be interpreted as a gestural cue. In addition, a human operator can specify that a gestural cue occurs at a specific point in time on the system control interface, which prompts the system to associate the cue to the object that is closest to the data collection glove at that point in time.

Linguistic cues can also provide information for initializing the learning process. Instead of providing information for initializing h values, linguistic cues can provide information to initialize learned trees, by "seeding" them with concepts that have been previously learned. A linguistic cue can be associated with a SMRF tree. For instance, the cue "red" specifies that a red object is important, and corresponds to a tree with an instantiation node and a question node that asks if the color attribute of a particular object falls within a "red" region of RGB space. This tree can be used as

an initial starting point from which to learn further tree expansions in order to capture the complete target concept. The cue tree can either be used "as-is," or its question node model parameters can be re-optimized to account for unique features of the data present in the training set. In addition to "seeding" the tree, linguistic cues can also be utilized by being treated as a possible expansion in addition to the standard tree expansions. To take advantage of the information provided by the cues, the expansions could be ordered by the amount of their expected relevance to the target concept. Cue expansions would have the highest relevance, followed by models that ask questions about attributes and relations that the cue expansions also utilize. The expansions could be evaluated in "chunks," such that if a particular expansion brings a substantial and significant improvement, it can be selected without evaluating the other expansions in the later chunks. This allows for the cue expansions to be utilized when they are relevant, and to prune the space of possible expansions in such cases.

Linguistic cues can also be collected in real-time by our object tracking system. The simplest method of collecting cues is to pre-define a set of linguistic utterances, and have them selectable from the system control interface. Then, during the data collection process, the operator signals a cue from the interface, and the system records the utterance as occurring at that specific point in time. Another option is to use a speech recognition system, such as ViaVoice (TM) or Sphinx. Given a linguistic utterance, translation into a SMRF cue expansion (or initial tree) is relatively straightforward. One approach is to explicitly map each possible utterance to a specific tree. This is feasible only if the cue language is small, but the approach is a simple one. A more powerful and complex approach is to define a Feature Context-Free Grammar (FCFG) for the cue language, using a tool such as the Natural Language Toolkit. The FCFG takes linguistic utterances and parses them into a first-order logic (FOL) expression. The second step would then be to translate the first-order logic expression into a corresponding SMRF tree. This can be done recursively, as

existential quantifiers and the variables they bind correspond to instantiation nodes, logical connectives correspond to different tree structures (involving question and leaf nodes), and predicates correspond to specific question node models.

Deictic cues can be collected by a mixture of the above methods. Deictic words (such as "this") can be made selectable from the system control interface. Then, during the data collection process, the operator can signal a deictic cue in correspondence with a gesture. For instance, to express the cue "this near that," the teacher would place his hand near one object, at which time the system operator would signal "this." The teacher would then move his hand near a second object, at which point the operator would signal "near that." Once the objects to which the deictic words refer are identified, the deictic cue can be processed like any other linguistic cue by the system.

9.3 Novel Problem Domains

SMRF, at its present state, could be applied to solving a number of other kinds of learning problems (with the caveat that appropriate mapping functions are provided to enable learning on the dataset). As has been stated numerous times, SMRF is most naturally suited to learning concepts in datasets which have some natural correspondence to attributed objects in real-world physical domains.

One natural application of SMRF is to the domain of learning from demonstrations. Given the temporal representation encoding discussed in Section 9.1, the movement of the objects during the demonstration could be naturally encoded. Apart from that means of representing data, each state of the demonstration could be represented as a distinct graph. Distinct states of the process would then be learned by extracting all graphs of state t from the dataset and labeling them positive, and the extract all graphs of state $t' < t$ and labeling them negative.

SMRF would not be able to perform a learned task, however, without an associated planning mechanism. Such a mechanism could be augmented with regression capabilities, as discussed in Section 9.1. However, other methods could be employed, such as those developed by Palmer et al. (2012).

SMRF could be used to model concepts from other domains as well, such as meteorology. For example, when predicting whether or not a given thunderstorm will produce a tornado, various features of the storm can be represented as objects with attributes. For example updrafts and downdrafts have a location within the storm (or a location relative to one another), as well as a velocity of the associated air mass. With the addition of temporal object representation, SMRF theoretically could be used to learn concepts in this domain as well, to enable the prediction of the occurrence of certain meteorological events.

Chapter 10

Conclusion

This dissertation has presented SMRF, a tree-based relational machine learning framework capable of learning target concepts from labeled groups of attributed objects. In particular, SMRF is designed to solve problems where these attribute values are both continuous and multidimensional, though it designed to account for categorical variables as well. SMRF is particularly suited to solve problems where these multidimensional and continuous attribute values also exhibit interesting covariances. Furthermore, SMRF is designed to solve problems where the target concept does not consist merely in a particular set of values for a particular object attribute, but where target concepts are defined in terms of relationships between objects. These relationships do not need to be specified *a priori*, as SMRF is capable of identifying salient relationships from the data itself.

SMRF is capable of learning a variety of different kinds of concepts. For this work, SMRF was tested upon purely conjunctive concepts (such as "Blue above Green"), purely disjunctive concepts (such as "Red or Green"), and a mix of the two (such as "Red above Green or Blue above Yellow"). In each case, SMRF proved itself capable of producing valid concept representations. And in many cases, especially for the harder concepts, SMRF produced substantially better representations than its competitor algorithms (such as EM-DD and TILDE). Over all of the different concepts combined, SMRF significantly and substantially outperforms the other competitor algorithms.

As such, SMRF is capable of learning and representing relational concepts with

varying degrees of complexity. Moreover, SMRF is capable of learning concepts that are corrupted by various amounts of noise. SMRF is reasonably robust to both added distractor objects, as well as the corruption of the labels of the training data examples. And as such, it proves itself to be much more robust to the injection of such noise than competing approaches, such as TILDE.

Lastly, SMRF is able to cope with problems entailing large instance spaces much better than other competing approaches. Whereas other approaches require the explicit enumeration of the instance space *a priori*, SMRF is able to prune away large and irrelevant portions of the instance space through incremental instantiation. As such, SMRF is able to learn relational concepts over data containing examples with 100 distractor objects each, while the other competing approaches are unable to learn concepts over data containing examples with even 10 or 20 distractor objects in a reasonable amount of time.

As such, SMRF makes a unique contribution to the field of machine learning, as it outperforms other well-known algorithms on the kinds of problems it is designed to solve. Moreover, the attributed object representation utilized by SMRF renders it particularly suitable for problems in the physical domain where a computational or robotic agent must learn from or interact with a set of objects in a particular configuration in the real world. As such, SMRF has the potential to be used in a number of novel, interesting, and useful ways as the field of machine learning is used to solve more and more problems in everyday life.

Bibliography

Allen, J. F. (1991). Time and time again: The many ways to represent time. *International Journal of Intelligent Systems*, 6(4):341–355.

Andrews, S., Tsochantaridis, I., and Hofmann, T. (2003). Support vector machines for multiple-instance learning. In S. Becker, S. T. and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 561–568, Cambridge, MA. MIT Press.

Appice, A., Ceci, M., and Malerba, D. (2008). Top-down induction of relational model trees in multi-instance learning. In *ILP '08: Proceedings of the 18th International Conference on Inductive Logic Programming*, pages 24–41, Berlin, Heidelberg. Springer-Verlag.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483.

Atkeson, C. G. and Schaal, S. (1997). Robot learning from demonstration. In *the Proceedings of the International Confernce on Machine Learning*, pages 12–20. Morgan Kaufmann.

Babenko, B., Yang, M.-H., and Belongie, S. (2009). Visual tracking with online multiple instance learning. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 983–990. IEEE.

Babenko, B., Yang, M.-H., and Belongie, S. (2011). Robust object tracking with online multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1619–1632.

Barros, R. C., Cerri, R., Jaskowiak, P. A., and de Carvalho, A. (2011). A bottom-up oblique decision tree induction algorithm. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 450–456. IEEE.

Bjerring, L. and Frank, E. (2011). Beyond trees: adopting miti to learn rules and ensemble classifiers for multi-instance data. In *AI 2011: Advances in Artificial Intelligence*, pages 41–50. Springer.

Blitch, J. G. (1996). Artifical intelligence technologies for robot assisted urban search and rescue. *Expert Systems with Applications*, 11(2):109–124.

Blockeel, H. (2012). The ACE datamining system, https://dtai.cs.kuleuven.be/ace/. accessed october, 2012.

Blockeel, H., Page, D., and Srinivasan, A. (2005). Multi-instance tree learning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 57–64. ACM.

Blockeel, H. and Raedt, L. D. (1998). Top-down induction of logical decision trees. *Artificial Intelligence*, pages 285–297.

Bodenhamer, M., Bleckley, S., Fennelly, D., Fagg, A. H., and McGovern, A. (2009). Spatio-temporal multi-dimensional relational framework trees. In *Proceedings of the International Workshop on Spatial and Spatiotemporal Data Mining, IEEE Conference on Data Mining*. electronically published.

Bodenhamer, M., Palmer, T., Sutherland, D., and Fagg, A. H. (2012). Grounding conceptual knowledge with spatio-temporal multi-dimensional relational framework trees. *School of Computer Science, University of Oklahoma, Tech. Rep. TR-AIR-1138*.

Bratko, I. (2001). *Prolog: Programming for Artificial Intelligence*. Addison-Wesley.

Breazeal, C., Hoffman, G., and Lockerd, A. (2004). Teaching and working with robots as a collaboration. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1030–1037, Washington, DC, USA. IEEE Computer Society.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth: Belmont, California.

Bruemmer, D. J. and Walton, M. C. (2003). Collaborative tools for mixed teams of humans and robots. In *Proceedings of the Workshop on Multi-Robot Systems*, pages 219–229.

Burke, J. L., Murphy, R. R., Rogers, E., Lumelsky, V. J., and Scholtz, J. (2004). Final report for the darpa/nsf interdisciplinary study on human-robot interaction. In *IEEE Transactions on Systems, Man, and Cybernetics: Part C – Applications and Reviews*, volume 34, pages 103–112.

Cooper, M., Keating, D., Harwin, W., and Dautenhan, K. (1999). Robots in the classroom – tools for accessible education. In *The Fifth European Conference of the Advancement of Assistive Technology*, pages 448–452.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms, Second Edition*. MIT Press.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Datta, R., Li, J., and Wang, J. Z. (2005). Content-based image retrieval: approaches and trends of the new age. In *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 253–262. ACM.

Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Eedicinal Chemistry*, 34(2):786–797.

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood estimation from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

Dieter, H. K., Fox, D., Etzioni, O., Borriello, G., and Arnstein, L. (2002). An overview of the assisted cognition project. In *In AAAI-2002 Workshop on Automation as Caregiver: The Role of Intelligent Technology in Elder*, pages 60–65.

Dietterich, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157.

Dietterich, T. G., Lathrop, R. H., and Lozano-Perez, T. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71.

Doran, G. (2013). MISVM: Multiple-instance support vector machines, https://github.com/garydoranjr/misvm. accessed august, 2013.

Dzeroski, S. (2007). Inductive logic programming in a nutshell. In Getoor, L. and Taskar, B., editors, *Introduction to Statistical Relational Learning*. MIT Press.

Dzeroski, S., Blockeel, H., Kompare, B., Kramer, S., Pfahringer, B., and Laer, W. (1999). Experiments in predicting biodegradability. In Dzeroski, S. and Flach, P., editors, *Inductive Logic Programming*, volume 1634 of *Lecture Notes in Computer Science*, pages 80–91. Springer Berlin Heidelberg.

Dzeroski, S., Schulze-Kremer, S., Heidtke, K. R., Siems, K., Wettschereck, D., and Blockeel, H. (1998). Diterpene structure elucidation from 13cnmr spectra with inductive logic programming. *Applied Artificial Intelligence*, 12(5):363–383.

Edgar, G. A. (2008). *Measure, topology, and fractal geometry*. Springer-Verlag.

Emde, W. and Wettschereck, D. (1996). Relational instance-based learning. In *Proceedings of the Thirteenth International Conference on Machine Learning*, volume 96, pages 122–130.

Fierens, D., Ramon, J., Blockeel, H., and Bruynooghe, M. (2005). A comparison of approaches for learning probability trees. In *Machine Learning: ECML 2005*, pages 556–563. Springer.

Freund, Y. and Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer.

Friedman, J. H. (1977a). A recursive partitioning decision rule for nonparametric classification. *IEEE Trans. Comput.*, 26(4):404–408.

Friedman, J. H. (1977b). A recursive partitioning decision rule for nonparametric classification. *IEEE Transactions on Computers*, C-26(4):404–408.

Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. (1999a). Learning probabilistic relational models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1300–1309.

Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. (1999b). Learning probabilistic relational models. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1300–1309.

Friedman, N. and Goldszmidt, M. (1998). Learning Bayesian networks with local structure. In *Learning in Graphical Models*, pages 421–459. Springer.

Garfield, E. and Merton, R. K. (1979). *Citation Indexing: Its Theory and Application in Science, Technology, and Humanities*, volume 8. Wiley New York.

Gehler, P. and Chapelle, O. (2006). Deterministic annealing for multiple-instance learning. *AI and Statistics (AISTATS)*.

Getoor, L., Friedman, N., Koller, D., and Taskar, B. (2002). Learning probabilistic models of link structure. *Journal of Machine Learning Research*, 3:679–707.

Getoor, L. and Taskar, B., editors (2007). *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, Massachusetts.

Golub, G. and Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224.

Harnad, S. (1990). The symbol grounding problem. *Journal of Physics D*, 42(1-3):335–346.

Haskell, R. E. (1993). Neuro-fuzzy classification and regression trees. In *Proceedings of the Third International Conference on Applications of Fuzzy Systems and Soft Computing*, pages 5–7.

Huelsenbeck, J. P. and Crandall, K. A. (1997). Phylogeny estimation and hypothesis testing using maximum likelihood. *Annual Review of Ecology and Systematics*, 28:437–466.

Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge university press.

Jensen, D. and Cohen, P. (2000). Multiple comparisons in induction algorithms. *Machine Learning*, 38(3):309–338.

Jensen, D. and Getoor, L. (2003). IJCAI 2003 workshop on learning statistical models from relational data. http://kdl.cs.umass.edu/srl2003/.

Jonathan, A. B., Knight, J., Listopad, S., Magerko, B., and Nourbakhsh, I. R. (2000). Robot improv: Using drama to create believable agents. In *In AAAI Workshop Technical Report WS-99-15 of the 8th Mobile Robot Competition and Exhibition*, pages 27–33. AAAI Press, Menlo.

Kemp, C. and Jern, A. (2009). Abstraction and relational learning. *Advances in Neural Information Processing Systems*, 22:934–942.

Kersting, K. and Raedt, L. D. (2001). Towards combining inductive logic programming with bayesian networks. In *Proceedings of the 11th International Conference on Inductive Logic Programming*, pages 118–131. Springer.

Kersting, K., Raiko, T., Kramer, S., and De Raedt, L. (2003). Towards discovering structural signatures of protein folds based on logical hidden markov models. In *Pacific Symposium on Biocomputing*, volume 8, pages 192–203.

Kim, M. and De la Torre, F. (2010). Gaussian processes multiple instance learning. In *Proceedings of the 27th International Conference on Machine Learning*, pages 535–542.

Kolmogorov, A. N. (1956). *Foundations of the Theory of Probability*. Chelsea, New York, NY.

Kramer, S. (1996). Structural regression trees. In *Proceedings of the National Conference on Artificial Intelligence*, pages 812–819.

Kramer, S. and Widmer, G. (2001). Inducing classification and regression trees in first order logic. In *Relational Data Mining*, pages 140–156. Springer-Verlag New York, Inc.

Kubica, J., Moore, A., Cohn, D., and Schneider, J. (2003a). Finding underlying connections: A fast graph-based method for link analysis and collaboration queries. In *Proceedings of the International Conference on Machine Learning*, pages 392–399.

Kubica, J., Moore, A., and Schneider, J. (2003b). Tractable group detection on large link data sets. In Wu, X., Tuzhilin, A., and Shavlik, J., editors, *The Third IEEE International Conference on Data Mining*, pages 573–576. IEEE Computer Society.

Kuhlmann, G., Stone, P., Mooney, R., and Shavlik, J. (2004). Guiding a reinforcement learner with natural language advice:. In *In Proc. of the AAAI-04 Workshop on Supervisory Control of Learning and Adaptive Systems*.

Kulyukin, V. and Gharpure, C. (2006). A robotic shopping assistant for the blind. In *Proceedings of the 29th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society of North America*.

Kulyukin, V., Gharpure, C., Nicholson, J., and Osborne, G. (2006). Robot-assisted wayfinding for the visually impaired in structured indoor environments. *Autonomous Robots*, 21(1):29–41.

Lauria, S., Bugmann, G., Kyriacou, T., and Klein, E. (2002). Mobile robot programming using natural language. *Robotics and Autonomous Systems*, 38(3–4):171–181.

Leger, P. C., Trebi-Ollennu, A., Wright, J. R., Maxwell, S. A., Bonitz, R. G., Biesiadecki, J. J., Hartman, F. R., Cooper, B. K., Baumgartner, E. T., and Maimone, M. W. (2005). Mars exploration rover surface operations: driving spirit at gusev crater. In *2005 IEEE Internation Conference on Systems, Man, and Cybernetics*, volume 2, pages 1815–1822.

Leistner, C., Saffari, A., and Bischof, H. (2010). Miforests: multiple-instance learning with randomized trees. In *Computer Vision–ECCV 2010*, pages 29–42. Springer.

Levinson, S. C. (1983). *Pragmatics*. Cambridge University Press.

Lin, L. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. In *Machine Learning*, pages 293–321.

Lin, Z., Hua, G., and Davis, L. S. (2009). Multiple instance ffeature for robust part-based object detection. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 405–412. IEEE.

Lloyd, J. W. (1987). *Foundations of Logic Programming*. Springer.

Lodhi, H. and Muggleton, S. (2005). Is mutagenesis still challenging. In *International Conference on Inductive Logic Programming (ILP'05), late-breaking papers*, pages 35–40.

Mangasarian, O. and Wild, E. W. (2008). Multiple instance classification via successive linear programming. *Journal of Optimization Theory and Applications*, 137(3):555–568.

Maron, O. (1998). *Learning from Ambiguity*. Phd thesis, Massachusetts Institute of Technology.

Maron, O. and Lozano-Pérez, T. (1998). A framework for multiple-instance learning. In Jordan, M. I., Kearns, M. J., and Solla, S. A., editors, *Advances in Neural Information Processing Systems 10*, pages 570–576, Cambridge, Massachusetts. MIT Press.

Maron, O. and Ratan, A. L. (1998). Multiple-instance learning for natural scene classification. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 341–349. Morgan Kaufmann, San Francisco, CA.

Mason, L., Baxter, J., Bartlett, P., and Frean, M. (1999). Functional gradient techniques for combining hypotheses. *Advances in Neural Information Processing Systems (NIPS)*, pages 221–246.

McGovern, A., Hiers, N., Collier, M., Gagne II, D. J., and Brown, R. A. (2008). Spatiotemporal relational probability trees. In *Proceedings of the 2008 IEEE International Conference on Data Mining*, pages 935–940, Pisa, Italy.

McGovern, A., John Gagne, D., Troutman, N., Brown, R. A., Basara, J., and Williams, J. K. (2011). Using spatiotemporal relational random forests to improve our understanding of severe weather processes. *Statistical Analysis and Data Mining*, 4(4):407–429.

McGovern, A., Supinie, T., Gagne, I., Collier, M., Brown, R., Basara, J., and Williams, J. (2010). Understanding severe weather processes through spatiotemporal relational random forests. In *2010 NASA Conference on Intelligent Data Understanding*.

McGovern, A., Troutman, N., Brown, R., Williams, J., and Abernethy, J. (2013). Enhanced spatiotemporal relational probability trees and forests. *Data Mining and Knowledge Discovery*, 26(2):398–433.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Montemerlo, M., Pineau, J., Roy, N., Thrun, S., and Verma, V. (2002). Experiences with a mobile robotic guide for the elderly.

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8(4):295–318.

Neville, J. and Jensen, D. (2007). Relational dependency networks. *Journal of Machine Learning Research*, 8:653–692.

Neville, J., Jensen, D., Friedland, L., and Hay, M. (2003). Learning relational probability trees. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 625–630.

Nocedal, J. and Wright, S. J. (2006). *Numerical optimization*. Springer Science+ Business Media.

Palmer, T. J., Bodenhamer, M., and Fagg, A. H. (2012). Learning to predict action outcomes in continuous, relational environments. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pages 1–7. IEEE.

Palmer, T. J., Bodenhamer, M., and Fagg, A. H. (2014). Multiple instance learning via covariant expansion. *School of Computer Science, University of Oklahoma, Tech. Rep. TR-AIR-1139*.

Parasuraman, R., Sheridan, T. B., and Wickens, C. D. (2000). A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 30(3):286–297.

Pollack, M. E. (2005). Intelligent technology for an aging population: The use of AI to assist elders with cognitive impairment. *AI Magazine*, 26(2):9–24.

Provost, F. and Domingos, P. (2000). Well-trained pets: Improving probability estimation trees. Technical report, Stern School of Business, NYU.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine learning*, 5(3):239–266.

Quinlan, J. R. (1993). *C4. 5: Programs for Machine Learning*, volume 1. Morgan kaufmann.

Rasmussen, C. E. (2004). Gaussian processes in machine learning. In *Advanced Lectures on Machine Learning*, pages 63–71. Springer.

Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine Learning*, 62(1–2):107–136.

Rose, K. (1998). Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 86(11):2210–2239.

Roy, N., Baltus, G., Fox, D., Gemperle, F., Goetz, J., Hirsch, T., Margaritis, D., Montemerlo, M., Pineau, J., Schulte, J., et al. (2000). Towards personal service robots for the elderly. In *Workshop on Interactive Robots and Entertainment (WIRE 2000)*, volume 25, page 184.

Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., and Edwards, D. D. (1995). *Artificial intelligence: A Modern Approach*, volume 74. Prentice Hall Englewood Cliffs.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464.

Shotton, J., Johnson, M., and Cipolla, R. (2008). Semantic texton forests for image categorization and segmentation. In *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE.

Shoval, S., Borenstein, J., and Koren, Y. (1994). Mobile robot obstacle avoidance in a computerized travel aid for the blind. In *IEEE International Conference on Robotics and Automation*, pages 8–13.

Smart, W. D. and Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Supinie, T. A., McGovern, A., Williams, J., and Abernathy, J. (2009). Spatiotemporal relational random forests. In *IEEE International Conference on Data Mining Workshops*, pages 630–635. IEEE.

Suppes, P. (1972). *Axiomatic Set Theory*. Dover.

Sutherland, D. (2011). *Integrating Human Knowledge into a Relational Learning System*. Undergraduate honors thesis, Swarthmore College.

Taskar, B., Abbeel, P., and Koller, D. (2002). Discriminative probabilistic models for relational data. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 485–492.

Thomaz, A. L. (2006). *Socially Guided Machine Learning*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Viola, P., Platt, J., Zhang, C., et al. (2006). Multiple instance boosting for object detection. *Advances in Neural Information Processing Systems*, 18:1417.

Wada, K., Shibata, T., Saito, T., and Tanie, K. (2002). Analysis of factors that bring mental effects to elderly people in robot assisted activity. In *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, volume 2, pages 1152–1157 vol.2.

Wang, J. and Domingos, P. (2008). Hybrid markov logic networks. In *Proceedings of the Twenty-third AAAI Conference on Artificial Intelligence*, volume 8, pages 1106–1111.

Wang, J. and Zucker, J.-D. (2000). Solving the multiple-instance problem: A lazy learning approach. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 1119–1125. Morgan Kaufmann, San Francisco, CA.

Wang, S. X. (2001). *Maximum Weighted Likelihood Estimation*. PhD thesis, University of British Columbia.

Wolsey, L. A. (1998). *Integer Programming*. Wiley, New York.

Xu, X. (2003). *Statistical Learning in Multiple Instance Problems*. PhD thesis, the University of Waikato.

Yanco, H. A. (2001). Development and testing of a robotic wheelchair system for outdoor navigation. In *Proceedings of the 2001 Conference of the Rehabilitation Engineering and Assistive Technology Society of North America*, pages 588–603. RESNA Press.

Yang, C. and Lozano-Perez, T. (2000). Image database retrieval with multiple-instance learning techniques. In *Proceedings of the 16th IEEE International Conference on Data Engineering*, pages 233–243. IEEE.

Yang, J. (2013). MILL: A multiple instance learning library, http://www.cs.cmu.edu/ juny/mill. accessed august, 2013.

Zeisl, B., Leistner, C., Saffari, A., and Bischof, H. (2010). On-line semi-supervised multiple-instance boosting. In *Proceedings of the 2010 IEEE Conference onComputer Vision and Pattern Recognition (CVPR)*, pages 1879–1879. IEEE.

Zhang, M.-L. and Zhou, Z.-H. (2006). Adapting rbf neural networks to multi-instance learning. *Neural Processing Letters*, 23(1):1–26.

Zhang, Q. and Goldman, S. (2001). EM-DD: An improved multiple-instance learning technique. In *Proceedings of Neural Information Processing Systems*, volume 14, pages 1073–1080. MIT Press.

Zhang, Q. and Goldman, S. A. (2002). EM-DD: An improved multiple-instance learning technique. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.

Zhang, Q., Goldman, S. A., Yu, W., and Fritts, J. E. (2002). Content-based image retrieval using multiple-instance learning. In *International Conference on Machine Learning (ICML)*, pages 682–689.

Zhou, Z.-H. and Zhang, M.-L. (2002). Neural networks for multi-instance learning. In *Proceedings of the International Conference on Intelligent Information Technology, Beijing, China*, pages 455–459.

Zhou, Z.-H., Zhang, M.-L., and Chen, K.-J. (2003). A novel bag generator for image database retrieval with multi-instance learning techniques. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, pages 565–569. IEEE.

Zucker, J.-D. and Chevaleyre, Y. (2000). Solving multiple-instance and multiple-part learning problems with decision trees and decision rules. Application to the mutagenesis problem. Technical Report 6, University of Paris.

Appendix A

An Information-Theoretic Analysis of the Grand Metric

In this appendix, I demonstrate that the learning algorithm described in Chapter 5 is directly related to learning algorithms based upon information gain.

Consider a set of objects S, and a set of classes C to which the objects in S belong. The information entropy of S, denoted $\textsc{Entropy}(S)$, is defined as

$$\textsc{Entropy}(S) = -\sum_{i \in C} p_i \log_2 (p_i), \tag{A.1}$$

where p_i denotes the proportion of members of class i in S.

Likewise, suppose that an entity (such as a question node) $\mathcal{L}$ partitions S into a set of subsets, $\mathbf{S}_{\mathcal{L}}$. The information gain given the partition induced by $\mathcal{L}$, denoted $\textsc{Gain}(S, \mathcal{L})$, is defined as

$$\textsc{Gain}(S, \mathcal{L}) = \textsc{Entropy}(S) - \sum_{S_l \in \mathbf{S}_{\mathcal{L}}} \frac{|S_{\mathcal{L}}|}{|S|} \textsc{Entropy}(S_l), \tag{A.2}$$

where each S_l denotes a subset of S.

Applying Equations A.1 and A.2 to the SMRF domain, let the set S denote the set of graphs "won" by a leaf node (Sec. 5.4). In the SMRF domain, the set of classes $C = \{+, -\}$, denoting positive and negative graphs, respectively. From this, it follows

that the information entropy for a leaf node is defined as follows:

$$\begin{aligned}
\textsc{Entropy}(S) &= -\sum_{i\in\{+,-\}} p_i \log_2(p_i) \\
&= -\left[p_+ \log_2(p_+) + p_- \log_2(p_-)\right] \\
&= -\left[\frac{\pi}{\pi+n}\log_2\left(\frac{\pi}{\pi+n}\right) + \frac{n}{\pi+n}\log_2\left(\frac{n}{\pi+n}\right)\right] \\
&= \frac{-1}{\pi+n}\left[\pi\log_2\left(\frac{\pi}{\pi+n}\right) + n\log_2\left(\frac{n}{\pi+n}\right)\right] \\
&= \frac{-1}{\pi+n}\left[\pi\log_2(\pi) - \pi\log_2(\pi+n) + n\log_2(n) - n\log_2(\pi+n)\right] \\
&= \frac{-1}{\pi+n}\left[\pi\log_2(\pi) + n\log_2(n) - (\pi+n)\log_2(\pi+n)\right], \qquad \text{(A.3)}
\end{aligned}$$

where π denotes the number of positive graphs in S, and n denotes the number of negative graphs in S, such that:

$$|S| = \pi + n.$$

To make the derivations more compact, let $\widetilde{LL}_b(x,y)$ be defined as follows:

$$\widetilde{LL}_b(x,y) = x\log_b x + y\log_b y - (x+y)log_b(x+y). \qquad \text{(A.4)}$$

In cases where the logarithmic base $b = e$, the b subscript parameter will be omitted. With this definition of $\widetilde{LL}_b$, the information gain in Equation A.3 becomes:

$$\textsc{Entropy}(S) = \frac{-1}{\pi+n}\widetilde{LL}_2(\pi,n). \qquad \text{(A.5)}$$

A.1 Information Gain over a One-Leaf Tree

Suppose that, through one or more iterations of the learning algorithm, the original one-leaf tree for which S is defined is expanded into a multi-leaf tree, whose leaf nodes

comprise the set $\mathcal{L}$. The information gain, given the partition of S induced by $\mathcal{L}$, is defined as follows:

$$\begin{aligned}
\textsc{Gain}(S, \mathcal{L}) &= \textsc{Entropy}(S) - \sum_{S_l \in \mathbf{S}_{\mathcal{L}}} \frac{|S_l|}{|S|} \textsc{Entropy}(S_l) \\
&= \textsc{Entropy}(S) - \sum_{l \in \mathcal{L}} \frac{\pi_l + n_l}{\pi + n} \textsc{Entropy}(S_l) \\
&= \frac{-1}{\pi + n} \widetilde{LL}_2(\pi, n) - \sum_{l \in \mathcal{L}} \frac{\pi_l + n_l}{\pi + n} \textsc{Entropy}(S_l) \\
&= \left(\frac{1}{\pi + n}\right) \left(- \sum_{l \in \mathcal{L}} (\pi_l + n_l) \, \textsc{Entropy}(S_l) - \widetilde{LL}_2(\pi, n) \right) \\
&= \left(\frac{1}{\pi + n}\right) \left(- \sum_{l \in \mathcal{L}} (\pi_l + n_l) \, \frac{-1}{\pi_l + n_l} \widetilde{LL}_2(\pi_l, n_l) - \widetilde{LL}_2(\pi, n) \right) \\
&= \left(\frac{1}{\pi + n}\right) \left(\sum_{l \in \mathcal{L}} \widetilde{LL}_2(\pi_l, n_l) - \widetilde{LL}_2(\pi, n) \right), \qquad \text{(A.6)}
\end{aligned}$$

where

$$\pi = \sum_{l \in \mathcal{L}} \pi_l, \qquad \text{(A.7)}$$

and

$$n = \sum_{l \in \mathcal{L}} n_l, \qquad \text{(A.8)}$$

such that π_l and n_l denote the number of positive and negative graphs at a given leaf l, respectively. Now, consider that for any $x \in \mathbb{R}$,

$$\log_2(x) = \frac{\log(x)}{\log(2)}, \qquad \text{(A.9)}$$

where $\log(x)$ denotes the natural logarithm of x, which is alternatively written as $\log_e(x)$. From Equations A.9 and A.4, it follows that

$$\widetilde{LL}_2(x, y) = \frac{\widetilde{LL}(x, y)}{\log(2)}. \qquad \text{(A.10)}$$

From Equation A.10, the definition of information gain (Eq. A.6) can be written as:

$$\textsc{Gain}(S, \mathcal{L}) = \left(\frac{1}{\log(2)(\pi + n)}\right)\left(\sum_{l \in \mathcal{L}} \widetilde{LL}(\pi_l, n_l) - \widetilde{LL}(\pi, n)\right). \tag{A.11}$$

Now, consider that for a single leaf l_0, with the set S of graphs "won" by that leaf, the grand metric likelihood $L(S|l_0)$ (Eq. 5.2) can be re-written as follows:

$$\begin{aligned} L(S|l_0) &= \left(\prod_{G \in S^+} \Pr\left(\mathcal{W}_{l_0}(G)\right)\right) \times \left(\prod_{G \in S^-} \left(1 - \Pr\left(\mathcal{W}_{l_0}(G)\right)\right)\right) \\ &= \left(\prod_{G \in S^+} \Pr\left(l_0\right)\right) \times \left(\prod_{G \in S^-} \left(1 - \Pr\left(l_0\right)\right)\right) \\ &= \Pr\left(l_0\right)^{|S^+|} \left(1 - \Pr\left(l_0\right)\right)^{|S^-|} \\ &= p_0^{\pi} \left(1 - p_0\right)^n, \end{aligned} \tag{A.12}$$

where S^+ and S^- denote the set of positive and negative graphs in S, respectively. Given the definition of p given in Section 5.1.2 (Eq. 5.7), Equation A.12 can be further re-written as:

$$\begin{aligned} L(S|l_0) &= p_0^{\pi} \left(1 - p_0\right)^n \\ &= \left(\frac{\pi}{\pi + n}\right)^{\pi} \left(1 - \frac{\pi}{\pi + n}\right)^n \\ &= \left(\frac{\pi}{\pi + n}\right)^{\pi} \left(\frac{\pi + n}{\pi + n} - \frac{\pi}{\pi + n}\right)^n \\ &= \left(\frac{\pi}{\pi + n}\right)^{\pi} \left(\frac{n}{\pi + n}\right)^n. \end{aligned} \tag{A.13}$$

The logarithm of this likelihood, $\log L(S|l_0)$, also denoted $LL(S|l_0)$, can be written as follows:

$$\begin{aligned}
LL(S|l_0) &= \log L(S|l_0) \\
&= \log\left[\left(\frac{\pi}{\pi+n}\right)^{\pi}\left(\frac{n}{\pi+n}\right)^{n}\right] \\
&= \log\left[\left(\frac{\pi}{\pi+n}\right)^{\pi}\right] + \log\left[\left(\frac{n}{\pi+n}\right)^{n}\right] \\
&= \pi\log\left(\frac{\pi}{\pi+n}\right) + n\log\left(\frac{n}{\pi+n}\right) \\
&= \pi\log(\pi) - \pi\log(\pi+n) + n\log(n) - n\log(\pi+n) \\
&= \pi\log(\pi) + n\log(n) - (\pi+n)\log(\pi+n) \\
&= \widetilde{LL}(\pi, n). \qquad \text{(A.14)}
\end{aligned}$$

Consider next that the tree comprising the single leaf l_0 is expanded through one or more iterations of the learning algorithm, to result in a tree whose set of leaf nodes is denoted $\mathcal{L}$. The grand metric likelihood $L(S|\mathcal{L})$ of this new tree can be written as follows:

$$\begin{aligned}
L(S|\mathcal{L}) &= \left(\prod_{G\in S^+}\Pr\left(\mathcal{W}_{\mathcal{L}}(G)\right)\right) \times \left(\prod_{G\in S^-}\left(1-\Pr\left(\mathcal{W}_{\mathcal{L}}(G)\right)\right)\right) \\
&= \prod_{l\in\mathcal{L}}\left(\prod_{G\in S_l^+}\Pr\left(l\right)\right)\left(\prod_{G\in S_l^-}\left(1-\Pr\left(l\right)\right)\right) \\
&= \prod_{l\in\mathcal{L}}\Pr\left(l\right)^{|S_l^+|}\left(1-\Pr\left(l\right)\right)^{|S_l^-|} \\
&= \prod_{l\in\mathcal{L}}\Pr\left(l\right)^{\pi_l}\left(1-\Pr\left(l\right)\right)^{n_l} \\
&= \prod_{l\in\mathcal{L}} p_l^{\pi_l}\left(1-p_l\right)^{n_l}, \qquad \text{(A.15)}
\end{aligned}$$

where S_l^+ and S_l^- denote the subset of S^+ and S^-, respectively, that is "won" by leaf l. Given the definition of p_l given in Appendix 5.1.2 (Eq. 5.7), Equation A.15 can be

further re-written as:

$$\begin{aligned} L(S|\mathcal{L}) &= \prod_{l\in\mathcal{L}} p_l^{\pi_l}\,(1-p_l)^{n_l} \\ &= \prod_{l\in\mathcal{L}} \left(\frac{\pi_l}{\pi_l+n_l}\right)^{\pi_l} \left(1-\frac{\pi_l}{\pi_l+n_l}\right)^{n_l} \\ &= \prod_{l\in\mathcal{L}} \left(\frac{\pi_l}{\pi_l+n_l}\right)^{\pi_l} \left(\frac{\pi_l+n_l}{\pi_l+n_l}-\frac{\pi_l}{\pi_l+n_l}\right)^{n_l} \\ &= \prod_{l\in\mathcal{L}} \left(\frac{\pi_l}{\pi_l+n_l}\right)^{\pi_l} \left(\frac{n_l}{\pi_l+n_l}\right)^{n_l}. \end{aligned} \tag{A.16}$$

The logarithm of this likelihood, $\log L(S|\mathcal{L})$, also denoted $LL(S|\mathcal{L})$, can be written as follows:

$$\begin{aligned} LL(S|\mathcal{L}) &= \log L(S|\mathcal{L}) \\ &= \log \prod_{l\in\mathcal{L}} \left(\frac{\pi_l}{\pi_l+n_l}\right)^{\pi_l} \left(\frac{n_l}{\pi_l+n_l}\right)^{n_l} \\ &= \sum_{l\in\mathcal{L}} \log\left[\left(\frac{\pi_l}{\pi_l+n_l}\right)^{\pi_l}\right] + \log\left[\left(\frac{n_l}{\pi_l+n_l}\right)^{n_l}\right] \\ &= \sum_{l\in\mathcal{L}} \pi_l \log\left(\frac{\pi_l}{\pi_l+n_l}\right) + n_l \log\left(\frac{n_l}{\pi_l+n_l}\right) \\ &= \sum_{l\in\mathcal{L}} \pi_l \log(\pi_l) - \pi_l \log(\pi_l+n_l) + n_l \log(n_l) - n_l \log(\pi_l+n_l) \\ &= \sum_{l\in\mathcal{L}} \pi_l \log(\pi_l) + n_l \log(n_l) - (\pi_l+n_l) \log(\pi_l+n_l) \\ &= \sum_{l\in\mathcal{L}} \widetilde{LL}(\pi_l, n_l). \end{aligned} \tag{A.17}$$

Next, consider the difference in log-likelihood induced by expanding the tree comprising the single leaf l_0 into the tree whose set of leaf nodes is denoted by $\mathcal{L}$. Let this difference be denoted $\Delta LL(S, \mathcal{L}, l_0)$, and be defined as follows:

$$\Delta LL(S, \mathcal{L}, l_0) = LL(S|\mathcal{L}) - LL(S|l_0). \tag{A.18}$$

From Equations A.14 and A.17, it follows that:

$$\Delta LL(S, \mathcal{L}, l_0) = \sum_{l \in \mathcal{L}} \widetilde{LL}(\pi_l, n_l) - \widetilde{LL}(\pi, n), \tag{A.19}$$

where Equations A.7 and A.8 also hold true. Comparing Equations A.11 and A.19, it can be easily seen that:

$$\begin{aligned}
\textsc{Gain}(S, \mathcal{L}) &= \left(\frac{1}{\log(2)(\pi + n)}\right) \left(\sum_{l \in \mathcal{L}} \widetilde{LL}(\pi_l, n_l) - \widetilde{LL}(\pi, n)\right), \\
\textsc{Gain}(S, \mathcal{L}) &= \left(\frac{1}{\log(2)(\pi + n)}\right) \Delta LL(S, \mathcal{L}, l_0), \\
\Delta LL(S, \mathcal{L}, l_0) &= \log(2)(\pi + n)\textsc{Gain}(S, \mathcal{L}) \\
&= \log(2)\,|S|\,\textsc{Gain}(S, \mathcal{L}).
\end{aligned} \tag{A.20}$$

As shown in Equation A.20, the difference log-likelihood induced by expanding the leaf is proportional to the information gain of the split in the set S induced by the leaf node set $\mathcal{L}$. Apart from the logarithm base conversion, the information gain is simply scaled by the number of graphs present at (and "won" by) the original leaf. In this way, the difference in log-likelihood encodes the information gain scaled by the number of graphs present. This additional scaling has the effect of inducing larger likelihood gains for expansions of single-leaf trees that have more graphs present. Intuitively, this makes sense. Expanding two separate single-leaf trees, T_i and T_j, may each provide the same information gain. However, if T_i has more graphs than T_j, it would make sense to say that the expansion of T_i provided a better overall sorting than the expansions of T_j, since more graphs are explained by the expansion of T_i, for the same amount of information gain.

A.2 Information Gain over a Multi-Leaf Tree

One can also consider the collective information entropy of a tree as being the sum of the information entropy of the leaves. Let $G_{l,T}$ denote the set of graphs "won" by leaf l of tree T. In addition, for a given tree T, let $\mathcal{L}_T$ denote the set of leafs nodes in T. Also let G_T denote the set of graphs "won" by the leaf nodes in T, such that:

$$G_T = \bigcup_{l \in \mathcal{L}_T} G_{l,T}.$$

Given these definitions, the information entropy of a given tree T, denoted by the term $\textsc{TreeEntropy}_T(G_T)$, can be defined as:

$$\textsc{TreeEntropy}_T(G_T) = \sum_{l \in \mathcal{L}_T} \frac{\pi_l + n_l}{|G_T|} \textsc{Entropy}(G_{l,T}), \tag{A.21}$$

where Entropy is defined in Equation A.5. With a notion of collective tree entropy, one can define a notion of tree information gain. Given a tree T, and a tree T' which is produced by one or more learning algorithm expansions of T, the information gain induced by T', relative to T, over the set G_T, is denoted $\textsc{TreeGain}_T(G_T, T')$, and is defined as the decrease in information entropy, such that:

$$\begin{aligned}
\textsc{TreeGain}_T(G_T, T') &= \textsc{TreeEntropy}_T(G_T) - \textsc{TreeEntropy}_{T'}(G_T) \\
&= \sum_{l \in \mathcal{L}_T} \frac{\pi_l + n_l}{|G_T|} \textsc{Entropy}(G_{l,T}) - \sum_{l \in \mathcal{L}_{T'}} \frac{\pi_l + n_l}{|G_T|} \textsc{Entropy}(G_{l,T'}) \\
&= \sum_{l \in \mathcal{L}_T} \frac{\pi_l + n_l}{|G_T|} \left(\frac{-\widetilde{LL}_2(\pi_l, n_l)}{\pi_l + n_l} \right) - \sum_{l \in \mathcal{L}_{T'}} \frac{\pi_l + n_l}{|G_T|} \left(\frac{-\widetilde{LL}_2(\pi_l, n_l)}{\pi_l + n_l} \right) \\
&= \left(\frac{1}{|G_T|} \right) \left(\sum_{l \in \mathcal{L}_{T'}} \widetilde{LL}_2(\pi_l, n_l) - \sum_{l \in \mathcal{L}_T} \widetilde{LL}_2(\pi_l, n_l) \right) \\
&= \left(\frac{1}{\log(2)\,|G_T|} \right) \left(\sum_{l \in \mathcal{L}_{T'}} \widetilde{LL}(\pi_l, n_l) - \sum_{l \in \mathcal{L}_T} \widetilde{LL}(\pi_l, n_l) \right).
\end{aligned} \tag{A.22}$$

Likewise, one can consider the difference in log-likelihood for a dataset and a tree T' that is the result of the learning algorithm expansion of a tree T. Let such a log-likelihood difference be denoted $\Delta LL(G_T, T', T)$, which in the vein of the notational conventions developed above, can be defined as:

$$\Delta LL(G_T, T', T) = LL(G_T|\mathcal{L}_{T'}) - LL(G_T|\mathcal{L}_T). \tag{A.23}$$

Given Equations A.17 and A.22, Equation A.23 can be rewritten as:

$$\begin{aligned}\Delta LL(G_T, T', T) &= LL(G_T|\mathcal{L}_{T'}) - LL(G_T|\mathcal{L}_T) \\ &= \sum_{l \in \mathcal{L}_{T'}} \widetilde{LL}(\pi_l, n_l) - \sum_{l \in \mathcal{L}_T} \widetilde{LL}(\pi_l, n_l) \\ &= \log(2)\, |G_T|\, \textsc{TreeGain}_T(G_T, T'). \end{aligned} \tag{A.24}$$

Thus, it is evident that the difference in log-likelihood resulting from learning algorithm tree expansion is equivalent to the information gain of the successor tree, scaled by the number of example graphs present, converted to the natural logarithmic base.

Moreover, consider that $|G_T|$ is a constant, since the number of graphs present in the predecessor tree T will be the same number of of graphs present in a successor tree T', as the learning algorithm expansion process only affects the sorting of the dataset, not the composition of the dataset. And since whole trees are being considered, not sub-trees, G_T thus remains constant across expansions.[1]

The learning procedure is designed to select the successor tree for which ΔLL is maximized. This "candidate" successor tree, denoted T_c, is defined as:

$$T_c = \max_{T' \in \textsc{Successors(T)}} \Delta LL(G_T, T', T), \tag{A.25}$$

[1] If T denoted a subtree of a larger tree $\overline{T}$, then G_T could vary across expansions if certain graphs were "won" by other leafs outside of $\mathcal{L}_T$.

where T is the tree being expanded, and $\textsc{Successors}(\mathrm{T})$ denotes the set of successor trees that the learning algorithm can derive from T. However, consider that by Equation A.24, Equation A.25 can be rewritten as:

$$T_c = \max_{T' \in \textsc{Successors}(\mathrm{T})} \log(2)\, |G_T|\, \textsc{TreeGain}_T(G_T, T'). \tag{A.26}$$

And since both $|G_T|$ and $\log(2)$ are constants in this context, Equation A.26 is equivalent to:

$$T_c = \max_{T' \in \textsc{Successors}(\mathrm{T})} \textsc{TreeGain}_T(G_T, T'). \tag{A.27}$$

Thus, as Equation A.27 demonstrates, the selection of the successor tree that maximizes the difference in log-likelihood is equivalent to the selection of the successor tree that maximizes information gain.

A.3 The Likelihood Ratio Test Statistic

Moreover, since whole trees are being considered, the set G_T and the dataset $\mathcal{D}$ are equivalent in this context, as every graph in the dataset is "won" by a leaf in any given tree. Given this, the test statistic given by Equation 5.13 can be rewritten as

follows:

$$
\begin{aligned}
s &= -2 \log \frac{L(\mathcal{D}|T)}{L(\mathcal{D}|T_c)} \\
&= -2 \left(\log L(\mathcal{D}|T) - \log L(\mathcal{D}|T_c)\right) \\
&= 2 \left(\log L(\mathcal{D}|T_c) - \log L(\mathcal{D}|T)\right) \\
&= 2 \left(LL(G_T|\mathcal{L}_{T_c}) - LL(G_T|\mathcal{L}_T)\right) \\
&= 2\Delta LL(G_T, T_c, T) \\
&= 2 \log(2) \left|G_T\right| \textsc{TreeGain}_T(G_T, T_c) \\
&= 2 \log(2) \left|\mathcal{D}\right| \textsc{TreeGain}_T(\mathcal{D}, T_c) \qquad \text{(A.28)}
\end{aligned}
$$

Thus, as Equation A.28 indicates, the test statistic is also related to information gain.

A.4 Conclusion

As Equations A.27 and A.28 demonstrate, the likelihood-based learning algorithm approach described in Chapter 5 turns out to be an approach concerned with the maximization of information gain. This is an unintended result, as the fundamental character of the learning algorithm, as discussed in Chapter 3 (in terms of maximizing the grand metric L), is derived from a different set of stated principles. Nonetheless, such a result is welcome, as it places the SMRF approach in a family of venerable and time-honored tree-based learning approaches that are also concerned with the maximization of information gain.

Appendix B

A Bayesian Approach to Selecting Leaf Node Probabilities

In this appendix, I demonstrate that the leaf node probability ratio described in Chapter 5, which is based upon frequentist assumptions, also makes sense on Bayesian grounds.

Taking a different approach from the frequentist arguments made in Section 5.4, one can view a leaf node probability $\Pr(l)$ as drawn from a binomial likelihood distribution. A binomial distribution $B(k|n,\theta)$ is defined as:

$$B(k|n,\theta) = \binom{n}{k}\theta^k(1-\theta)^{n-k}, \tag{B.1}$$

where k denotes a number of discrete successful independent yes/no trials, n denotes the total number of trials, and θ denotes the probability of success. Since the graphs in $\mathcal{D}$ are assumed to be independently generated, and are given a binary label of positive/negative, the binomial likelihood function $p(\theta|n,k)$ seems an appropriate choice for modeling the distribution of leaf node probabilities, where k represents the number of positive graphs represented at the leaf in question, and n represents the total number of graphs represented at the leaf in question. The binomial likelihood

function can be derived as follows:

$$\begin{aligned} p(\theta|n,k) &= \frac{p(\theta,n,k)}{p(n,k)} \\ &= \frac{p(k|\theta,n)p(\theta|n)p(n)}{p(k|n)p(n)} \\ &= \frac{p(k|\theta,n)p(\theta|n)}{p(k|n)} \\ &= \frac{p(k|\theta,n)p(\theta|n)}{\int_0^1 p(k,\theta|n)d\theta} \\ &= \frac{p(k|\theta,n)p(\theta|n)}{\int_0^1 p(k|\theta,n)p(\theta|n)d\theta}. \end{aligned} \tag{B.2}$$

Assuming that the $p(\theta|n)$ is uniform across all possible values of θ for a given n,[1] it can be treated as a constant, and thus Equation B.2 becomes:

$$\begin{aligned} p(\theta|n,k) &= \frac{p(\theta|n)p(k|\theta,n)}{p(\theta|n)\int_0^1 p(k|\theta,n)d\theta} \\ &= \frac{p(k|\theta,n)}{\int_0^1 p(k|\theta,n)d\theta} \end{aligned} \tag{B.3}$$

Given the binomial likelihood function, one can select leaf node probabilities by maximizing the likelihood function directly (Sec. B.1), or one can incorporate a prior and maximize the posterior distribution (Sec. B.2).

[1]Such is arguably the case is in this context, since the total number of graphs has no defined *a priori* relationship to the chosen leaf node probability.

B.1 Maximum Likelihood Estimation

The leaf node probability $\Pr(l)$ can be selected by finding the maximum likelihood estimate. The maximum likelihood estimate can be derived as follows:

$$
\begin{aligned}
\Pr(l) &= \arg\max_{\theta} \; p(\theta|n,k) \\
&= \arg\max_{\theta} \; \frac{p(k|\theta,n)}{\int_0^1 p(k|\theta,n)d\theta} \\
&= \arg\max_{\theta} \; \frac{B(k|n,\theta)}{\int_0^1 B(k|n,\theta)d\theta} \\
&= \arg\max_{\theta} \; \frac{\binom{n}{k}\theta^k(1-\theta)^{n-k}}{\int_0^1 \binom{n}{k}\theta^k(1-\theta)^{n-k}d\theta} \\
&= \arg\max_{\theta} \; \frac{\binom{n}{k}\theta^k(1-\theta)^{n-k}}{\binom{n}{k}\int_0^1 \theta^k(1-\theta)^{n-k}d\theta} \\
&= \arg\max_{\theta} \; \frac{\theta^k(1-\theta)^{n-k}}{\int_0^1 \theta^k(1-\theta)^{n-k}d\theta}
\end{aligned}
\tag{B.4}
$$

In order to find the value of θ that maximizes the expression, one can set the derivative with respect to θ equal to zero, and solve for θ:

$$\begin{aligned}
\frac{\partial}{\partial\theta}\frac{\theta^k(1-\theta)^{n-k}}{\int_0^1 \theta^k(1-\theta)^{n-k}d\theta} &= 0 \\
\frac{1}{\int_0^1 \theta^k(1-\theta)^{n-k}d\theta}\frac{\partial}{\partial\theta}\theta^k(1-\theta)^{n-k} &= 0 \\
\frac{\partial}{\partial\theta}\theta^k(1-\theta)^{n-k} &= 0 \\
(1-\theta)^{n-k}\frac{\partial}{\partial\theta}\theta^k + \theta^k\frac{\partial}{\partial\theta}(1-\theta)^{n-k} &= 0 \\
k\theta^{k-1}(1-\theta)^{n-k} + \theta^k(n-k)(1-\theta)^{n-k-1}(-1) &= 0 \\
k\theta^{k-1}(1-\theta)^{n-k} &= \theta^k(n-k)(1-\theta)^{n-k-1} \\
k(1-\theta) &= \theta(n-k) \\
k-k\theta &= n\theta - k\theta \\
k &= n\theta \\
\theta &= \frac{k}{n} \qquad \text{(B.5)}
\end{aligned}$$

It should be noted that Equation B.5 is equivalent to Equation 5.7, given the meaning of k and n in this context. Thus, the Bayesian maximum-likelihood approach provides the same solution as the approach taken in Section 5.1.2.

B.2 Maximum *A Posteriori* Estimation

The leaf node probability $\Pr(l)$ can also be selected by finding the maximum *a posteriori* estimate. The beta distribution is the conjugate prior of the binomial distribution, and is defined as:

$$\textsc{Beta}(\theta|\alpha,\beta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{\mathrm{B}(\alpha,\beta)}, \qquad \text{(B.6)}$$

where α and β are positive shape parameters, and B denotes the beta function, defined as:

$$\mathrm{B}(\alpha, \beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx. \tag{B.7}$$

With the beta distribution defining a prior distribution $p(\theta|\alpha, \beta)$ defined over the possible leaf node probabilities, the posterior distribution is:

$$p(\theta|n, k)\textsc{Beta}(\theta|\alpha, \beta). \tag{B.8}$$

Given this posterior distribution, the maximum *a posteriori* estimate can be derived as follows:

$$\begin{aligned}
\Pr(l) &= \arg\max_\theta \; p(\theta|n,k)\textsc{Beta}(\theta|\alpha,\beta) \\
&= \arg\max_\theta \; \frac{p(k|\theta,n)}{\int_0^1 p(k|\theta,n)d\theta}\textsc{Beta}(\theta|\alpha,\beta) \\
&= \arg\max_\theta \; \frac{B(k|n,\theta)\textsc{Beta}(\theta|\alpha,\beta)}{\int_0^1 B(k|n,\theta)d\theta} \\
&= \arg\max_\theta \; \frac{\binom{n}{k}\theta^k(1-\theta)^{n-k}\theta^{\alpha-1}(1-\theta)^{\beta-1}}{\left(\int_0^1 \binom{n}{k}\theta^k(1-\theta)^{n-k}d\theta\right)\mathrm{B}(\alpha,\beta)} \\
&= \arg\max_\theta \; \frac{\binom{n}{k}\theta^k(1-\theta)^{n-k}\theta^{\alpha-1}(1-\theta)^{\beta-1}}{\binom{n}{k}\mathrm{B}(\alpha,\beta)\int_0^1 \theta^k(1-\theta)^{n-k}d\theta} \\
&= \arg\max_\theta \; \frac{\theta^k(1-\theta)^{n-k}\theta^{\alpha-1}(1-\theta)^{\beta-1}}{\mathrm{B}(\alpha,\beta)\int_0^1 \theta^k(1-\theta)^{n-k}d\theta}
\end{aligned} \tag{B.9}$$

In order to find the value of θ that maximizes the expression, one can set the derivative with respect to θ equal to zero, and solve for θ:

$$\begin{aligned}
\frac{\partial}{\partial\theta}\frac{\theta^k(1-\theta)^{n-k}\theta^{\alpha-1}(1-\theta)^{\beta-1}}{\mathrm{B}(\alpha,\beta)\int_0^1\theta^k(1-\theta)^{n-k}d\theta} &= 0 \\
\frac{\frac{\partial}{\partial\theta}\theta^k(1-\theta)^{n-k}\theta^{\alpha-1}(1-\theta)^{\beta-1}}{\mathrm{B}(\alpha,\beta)\int_0^1\theta^k(1-\theta)^{n-k}d\theta} &= 0 \\
\frac{\partial}{\partial\theta}\theta^k(1-\theta)^{n-k}\theta^{\alpha-1}(1-\theta)^{\beta-1} &= 0 \\
\frac{\partial}{\partial\theta}\theta^{k+\alpha-1}(1-\theta)^{n-k+\beta-1} &= 0 \\
(1-\theta)^{n-k+\beta-1}\frac{\partial}{\partial\theta}\theta^{k+\alpha-1} &= -\theta^{k+\alpha-1}\frac{\partial}{\partial\theta}(1-\theta)^{n-k+\beta-1} \\
(k+\alpha-1)\theta^{k+\alpha-2}(1-\theta)^{n-k+\beta-1} &= \theta^{k+\alpha-1}(n-k+\beta-1)(1-\theta)^{n+k+\beta-2} \\
(k+\alpha-1)(1-\theta)^{n-k+\beta-1} &= \theta(n-k+\beta-1)(1-\theta)^{n-k+\beta-2} \\
(k+\alpha-1)(1-\theta) &= \theta(n-k+\beta-1) \\
k+\alpha-1-k\theta-\alpha\theta+\theta &= \theta n-\theta k+\theta\beta-\theta \\
k+\alpha-1-\alpha\theta+\theta &= \theta n+\theta\beta-\theta \\
(k+\alpha-1)+\theta(1-\alpha) &= \theta(n+\beta-1) \\
k+\alpha-1 &= \theta(n+\beta-1)-\theta(1-\alpha) \\
k+\alpha-1 &= \theta(n+\beta+\alpha-2) \\
\theta &= \frac{k+\alpha-1}{n+\beta+\alpha-2}
\end{aligned} \tag{B.10}$$

It should be noted that in the case of a uniform prior, where $\alpha = \beta = 1$, Equation B.10 reduces to Equation B.5. Thus, as might be expected, the maximum *a posteriori* solution is equivalent to the maximum likelihood solution given, a uniform prior.

B.3 Conclusion

A Bayesian approach to selecting leaf node probabilities provides the same solution as approach taken in Section 5.1.2. In this approach, the maximum likelihood estimate of leaf node probability (Eq. B.5) is equivalent to Equation 5.7. Likewise, the maximum *a posteriori* estimate of leaf node probability (Eq. B.10) is also equivalent to Equation 5.7 given a uniform prior. Such a result is welcome, as it demonstrates that a key aspect of the SMRF learning algorithm can be justified on Bayesian grounds.

Appendix C

An Analysis of the Hypothesis Space

In this appendix, I provide a short analysis of the space of hypotheses that SMRF trees represent. Recall from Section 4.4 that a single root-to-leaf path, represents a single hypothesis h_l. Let the set of all possible hypotheses be denoted $\mathcal{H}$. However, if the domain of discourse contains real-valued attributes, it can easily be seen that $\mathcal{H}$ has an uncountably infinite number of elements. Each h_l is defined in terms of question nodes, and the question nodes that concern real-valued attributes and relations contain models that are defined in terms of real-valued parameters. As there are an uncountably infinite number of such parameters, there are an uncountably infinite number of such models, and thus an uncountably infinite number of hypotheses. As such, $\mathcal{H}$ affords no further analysis where the domain of discourse concerns real-valued data.

In addition to considering the number of possible hypotheses, one can also consider the number of possible *types* of hypotheses. That is, one can consider what kinds of questions are asked, without considering the specific parameters of the associated models. As such, let $\overline{h_l}$ denote the *abstract hypothesis* which corresponds to the hypothesis h_l. The abstract hypothesis $\overline{h_l}$ represents an ordered sequence of mapping functions, where each element is the mapping function of the corresponding question node in the root-to-leaf path h_l. Let the set of all possible abstract hypotheses be denoted $\overline{\mathcal{H}}$. It can be easily seen that $\overline{\mathcal{H}}$ contains a countably infinite number of elements, if there are no *a priori* constraints restricting what kinds of questions can appear in $\overline{h_l}$. As such, any particular kind of question can potentially be repeated

indefinitely, giving rise to a countably infinite set of possibilities.

However, suppose that a constraint is placed upon an abstract hypothesis, which forbids any specific mapping function from being present more than once. Let such a *constrained abstract hypothesis* be denoted $\overline{h_l}'$, and let the set of all such possible hypotheses be denoted $\overline{\mathcal{H}}'$. Furthermore, suppose that for a given dataset, each group contains at most n objects. In addition, let k_m denote the number of available arity-m *abstract mapping functions*, where an abstract mapping function is defined as a mapping function whose mapping template is undefined. For example, **IdentityLocation** with a mapping template of (1) is a concrete mapping function that returns the location attribute value of the first instantiated object. On the other hand, **IdentityLocation** without a mapping template is an abstract mapping function, as there is no specification of which instantiated object should be examined *vis-à-vis* its location attribute value. If $m \leq n$, then it can be easily verified that there are $k_m \times {}_n\mathrm{P}_m$ possible arity-m (concrete) mapping functions that can be applied to the data, when instantiated m times.[1]

Now, consider the case where $n = 1$. As such, there are $k_1 \times {}_1\mathrm{P}_1 = k_1$ possible concrete mapping functions. All or any of these mapping functions may be used. Suppose that t mapping functions are used in a particular constrained hypothesis, where $t \leq k_1$. In this case, the t mapping functions may appear in any order. As such, there are ${}_t\mathrm{P}_t = t!$ possible root-to-leaf paths using these mapping functions. Given this result, it follows that the total number of possible abstract hypotheses is:

$$\left|\overline{\mathcal{H}}'\right| = \sum_{t=1}^{k_1} t!. \tag{C.1}$$

Using Stirling's approximation (Cormen et al., 2001), $t!$ is equivalent to $o(t^t)$.[2] Given

[1] This follows from the fact that there are ${}_n\mathrm{P}_m$ possible arity-m mapping templates.

[2] "Little-o" notation is used to denote a bound that is not guaranteed to be asymptotically tight.

this result, Equation C.1 becomes:

$$\left|\overline{\mathcal{H}}'\right| = \sum_{t=1}^{k_1} o(t^t), \tag{C.2}$$

which is equivalent to:

$$\left|\overline{\mathcal{H}}'\right| = o\left({k_1}^{k_1}\right). \tag{C.3}$$

As such, even in this very simple case, the result is exponential in the number of mapping function types.

Next, consider the case where $n > 1$, but where only arity-1 mapping functions are permitted. In this case, there are at most $k_1 \times {}_n\mathrm{P}_1 = nk_1$ possible concrete mapping functions. Using the same procedure which was used to derive Equation C.3, the number of possible abstract hypotheses in this case is:

$$\left|\overline{\mathcal{H}}'\right| = o\left({nk_1}^{nk_1}\right). \tag{C.4}$$

Now, consider that both arity-1 and arity-2 mapping functions are permitted. We have already seen that there can be at most nk_1 arity-1 mapping functions in a given constrained abstract hypothesis. There are at most $k_2 \times {}_n\mathrm{P}_2 < k_2 n^2$ distinct arity-2 mapping functions present in the hypothesis. As such, the total number of arity-1 and arity-2 mapping functions is bounded by $k_1 n + k_2 n^2$. Using the same procedure which was used to derive Equation C.3, the number of possible abstract hypotheses in this case is:

$$\left|\overline{\mathcal{H}}'\right| = o\left(\left(k_1 n + k_2 n^2\right)^{\left(k_1 n + k_2 n^2\right)}\right). \tag{C.5}$$

Extrapolating from Equation C.5 to the general case where mapping functions of up

to arity-m may be used, the number of possible abstract hypotheses becomes:

$$\begin{aligned} \left|\overline{\mathcal{H}}'\right| &= o\left(\left(\sum_{i=1}^{m} k_i n^i\right)^{\left(\sum_{i=1}^{m} k_i n^i\right)}\right), \\ &= o\left((n^m)^{(n^m)}\right). \end{aligned} \tag{C.6}$$

Finally, if $m = n$, then Equation C.6 becomes:

$$\begin{aligned} \left|\overline{\mathcal{H}}'\right| &= o\left((n^n)^{(n^n)}\right), \\ &= o\left(n^{n^{n^n}}\right). \end{aligned} \tag{C.7}$$

As Equations C.6 and C.7 indicate, the possible space of constrained abstract hypotheses is immense, for any substantial values of m and n. However, as the learning time results in Section 7.2.4 suggest, the SMRF learning algorithm explores only a small portion of this space in practice.

Appendix D

Maximum Likelihood Estimation of Density Parameters

Maximum likelihood estimation (MLE) is utilized in order to determine the parameters of the question node models (Sec. 5.3). As demonstrated in Appendix 5.1, the grand metric L (Eq. 5.2) can be maximized with respect to leaf node probabilities, in terms of graphs present. However, given the max operators inherent in the "winner" function (Eq. 4.17) in terms of which L is defined, no such straightforward closed-form solution exists for maximizing L with respect to question node model parameters, in terms of instances. To cope with this problem, the assumption is made that well-defined concepts will have concentrations of values in the metric space $\mathfrak{S}_q$ of pertinent question nodes. For example, a concept that specifies a blue object above a red object will have clusters of instance values in the "blue" and "red" regions of RGB space, if the concepts "blue" and "red" are well-defined.

Proceeding from this assumption, it makes sense to select density parameters θ_q that maximize the likelihood function $p(\mathbf{x}|\theta_q)$, where $\mathbf{x}$ denotes a set of the form $\{x_1, x_2, \ldots, x_n\}$, where each x_i is defined as

$$x_i = \phi(I_i),$$

for $I_i \in \widehat{{}^q\mathcal{I}}$, where $\widehat{{}^q\mathcal{I}}$ denotes a set of instantiation sequences deemed suitable for use as data in the optimization process, such that $\widehat{{}^q\mathcal{I}} \subseteq {}^q\mathcal{I}$. That is, density parameters should be selected to maximize the likelihood of the mapping function evaluations of some subset of the instantiation sequences present at a given question node q.

The likelihood function $p(\mathbf{x}|\theta_q)$ can be further simplified given the assumption that all of the values of $\mathbf{x}$ are independent. In the procedure described in Section 5.3, at most one instantiation sequence drawn from each graph is utilized as data in the optimization process. If each of the graphs are independent (as is the case for the datasets in Chapter 7), then it follows that the instantiation sequences in $\widehat{{}^{q}\mathcal{I}}$ will also be independent, since that set contains no more than one instantiation sequence drawn from each graph represented at q. Under these independence conditions, the likelihood function can be rewritten as:

$$p(\mathbf{x}|\theta) = \prod_{i=1}^{n} p(x_i|\theta). \tag{D.1}$$

One may also formulate solutions which take prior information into account regarding the importance or suitability of each of the individual x_i. The historical methods described in Appendix E took such an approach. The method of incorporating such prior information was to employ *maximum weighted likelihood estimation* (MWLE), as detailed by Wang (2001). Maximum weighted likelihood estimation is a means of incorporating prior information of the $\mathbf{x}$ in the form of real-valued weights $\mathbf{w}$. As such, the weighted likelihood function becomes:

$$p(\mathbf{x}|\theta, \mathbf{w}) = \prod_{i=1}^{n} p(x_i|\theta)^{w_i}. \tag{D.2}$$

In the cases where the weights $\mathbf{w}$ are uniform, the weighted maximum likelihood estimate is equivalent to the maximum likelihood estimate. And in the case that all $w_i = 1$, the weighted likelihood is equivalent to the unweighted likelihood, for a given θ.

As such, the weighted maximum likelihood estimators of various densities will be derived in this appendix, as they are also easily converted into maximum likeli-

hood estimators by selecting uniform weights. For the sake of clarity, the maximum likelihood estimators will also be given.

D.1 Multivariate Normal Distributions

The multivariate normal distribution has a probability density function defined as follows:

$$p(x|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^k\,|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}, \tag{D.3}$$

where x and μ are vectors of length k, and Σ is a $k \times k$ covariance matrix. The weighted likelihood becomes:

$$\begin{aligned} p(\mathbf{x}|\theta,\mathbf{w}) &= p(\mathbf{x}|\mu,\Sigma,\mathbf{w}) \\ p(\mathbf{x}|\theta,\mathbf{w}) &= \prod_{i=1}^{n} p(x_i|\mu,\Sigma)^{w_i}. \end{aligned} \tag{D.4}$$

Since log is a monotonically increasing function, the values of θ that maximize $p(\mathbf{x}|\theta, \mathbf{w})$ will also maximize $\log p(\mathbf{x}|\theta, \mathbf{w})$. As such,

$$\begin{aligned}
\log p(\mathbf{x}|\theta, \mathbf{w}) &= \log \left(\prod_{i=1}^{n} p(x_i|\mu, \Sigma)^{w_i} \right) \\
&= \sum_{i=1}^{n} \log p(x_i|\mu, \Sigma)^{w_i} \\
&= \sum_{i=1}^{n} w_i \log p(x_i|\mu, \Sigma) \\
&= \sum_{i=1}^{n} w_i \log \left(\frac{1}{\sqrt{(2\pi)^k |\Sigma|}} e^{-\frac{1}{2}(x_i-\mu)^T \Sigma^{-1}(x_i-\mu)} \right) \\
&= \sum_{i=1}^{n} w_i \left[\log \left(\frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \right) + \log \left(e^{-\frac{1}{2}(x_i-\mu)^T \Sigma^{-1}(x_i-\mu)} \right) \right] \\
&= \sum_{i=1}^{n} w_i \left[\log(1) - \log \left(\sqrt{(2\pi)^k |\Sigma|} \right) - \frac{1}{2}(x_i-\mu)^T \Sigma^{-1}(x_i-\mu) \right] \\
&= -\sum_{i=1}^{n} w_i \left[\log \left(\sqrt{(2\pi)^k |\Sigma|} \right) + \frac{1}{2}(x_i-\mu)^T \Sigma^{-1}(x_i-\mu) \right] \\
&= -\sum_{i=1}^{n} w_i \left[\frac{1}{2} \log \left((2\pi)^k |\Sigma| \right) + \frac{1}{2}(x_i-\mu)^T \Sigma^{-1}(x_i-\mu) \right] \\
&= -\frac{1}{2} \sum_{i=1}^{n} w_i \left[k \log(2\pi) + \log(|\Sigma|) + (x_i-\mu)^T \Sigma^{-1}(x_i-\mu) \right]. \qquad \text{(D.5)}
\end{aligned}$$

The weighted maximum likelihood estimate for μ is found by first taking the derivative of Equation D.5 with respect to μ:

$$\begin{aligned}
\frac{\partial \log p}{\partial \mu} &= \frac{\partial}{\partial \mu} \left(-\frac{1}{2} \sum_{i=1}^{n} w_i \left[k \log(2\pi) + \log(|\Sigma|) + (x_i-\mu)^T \Sigma^{-1}(x_i-\mu) \right] \right) \\
&= -\frac{1}{2} \sum_{i=1}^{n} w_i \frac{\partial}{\partial \mu} \left(k \log(2\pi) + \log(|\Sigma|) + (x_i-\mu)^T \Sigma^{-1}(x_i-\mu) \right) \\
&= -\frac{1}{2} \sum_{i=1}^{n} w_i \frac{\partial}{\partial \mu} (x_i-\mu)^T \Sigma^{-1}(x_i-\mu) \\
&= -\frac{1}{2} \sum_{i=1}^{n} w_i \left(-2\Sigma^{-1}(x_i-\mu) \right) \\
&= \Sigma^{-1} \sum_{i=1}^{n} w_i (x_i-\mu). \qquad \text{(D.6)}
\end{aligned}$$

Setting $\frac{\partial \log p}{\partial \mu}$ (Eq D.6) equal to zero,

$$\begin{aligned}
0 &= \Sigma^{-1} \sum_{i=1}^{n} w_i (x_i - \mu) \\
0 &= \sum_{i=1}^{n} w_i (x_i - \mu) \\
0 &= \sum_{i=1}^{n} w_i x_i - \sum_{i=1}^{n} w_i \mu \\
\sum_{i=1}^{n} w_i \mu &= \sum_{i=1}^{n} w_i x_i \\
\mu \sum_{i=1}^{n} w_i &= \sum_{i=1}^{n} w_i x_i \\
\mu &= \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}.
\end{aligned} \tag{D.7}$$

As one might expect, Equation D.7 is equivalent to the sample mean in the case where the weights $\mathbf{w}$ are uniform. In this case, which corresponds to the maximum likelihood estimate, the value of μ becomes:

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n}. \tag{D.8}$$

Likewise, the weighted maximum likelihood estimate for Σ is found by first taking the derivative of Equation D.5 with respect to Σ:

$$\begin{aligned}
\frac{\partial \log p}{\partial \Sigma} &= \frac{\partial}{\partial \Sigma}\left(-\frac{1}{2}\sum_{i=1}^{n} w_i\left[k\log(2\pi) + \log\left(|\Sigma|\right) + (x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\right]\right) \\
&= -\frac{1}{2}\sum_{i=1}^{n} w_i \frac{\partial}{\partial \Sigma}\left(k\log(2\pi) + \log\left(|\Sigma|\right) + (x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\right) \\
&= -\frac{1}{2}\sum_{i=1}^{n} w_i \left[\frac{\partial}{\partial \Sigma}\log\left(|\Sigma|\right) + \frac{\partial}{\partial \Sigma}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\right] \\
&= -\frac{1}{2}\sum_{i=1}^{n} w_i \left[\frac{1}{|\Sigma|}\frac{\partial}{\partial \Sigma}|\Sigma| - \left(\Sigma^{-1}\right)^T(x_i-\mu)(x_i-\mu)^T\left(\Sigma^{-1}\right)^T\right] \\
&= -\frac{1}{2}\sum_{i=1}^{n} w_i \left[\frac{1}{|\Sigma|}\left(|\Sigma|\left(\Sigma^{-1}\right)^T\right) - \left(\Sigma^{-1}\right)(x_i-\mu)(x_i-\mu)^T\left(\Sigma^{-1}\right)\right] \\
&= -\frac{1}{2}\sum_{i=1}^{n} w_i \left[\Sigma^{-1} - \Sigma^{-1}(x_i-\mu)(x_i-\mu)^T\Sigma^{-1}\right]. \qquad \text{(D.9)}
\end{aligned}$$

Setting $\frac{\partial \log p}{\partial \Sigma}$ (Eq D.9) equal to zero,

$$\begin{aligned}
0 &= -\frac{1}{2}\sum_{i=1}^{n} w_i \left[\Sigma^{-1} - \Sigma^{-1}(x_i-\mu)(x_i-\mu)^T\Sigma^{-1}\right] \\
0 &= \left(\sum_{i=1}^{n} w_i \left[1 - \Sigma^{-1}(x_i-\mu)(x_i-\mu)^T\right]\right)\Sigma^{-1} \\
0 &= \sum_{i=1}^{n} w_i \left[1 - \Sigma^{-1}(x_i-\mu)(x_i-\mu)^T\right] \\
0 &= \sum_{i=1}^{n} w_i - \sum_{i=1}^{n} w_i\Sigma^{-1}(x_i-\mu)(x_i-\mu)^T \\
\sum_{i=1}^{n} w_i &= \sum_{i=1}^{n} w_i\Sigma^{-1}(x_i-\mu)(x_i-\mu)^T \\
1 &= \Sigma^{-1}\frac{\sum_{i=1}^{n} w_i(x_i-\mu)(x_i-\mu)^T}{\sum_{i=1}^{n} w_i} \\
\Sigma &= \frac{\sum_{i=1}^{n} w_i(x_i-\mu)(x_i-\mu)^T}{\sum_{i=1}^{n} w_i}. \qquad \text{(D.10)}
\end{aligned}$$

In this case of uniform weights, which corresponds to the maximum likelihood estimate, the value of Σ becomes:

$$\Sigma = \frac{\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T}{n}. \tag{D.11}$$

The value of Σ in Equation D.11 approaches the value of the sample covariance as n becomes large.

D.2 Von Mises Distributions

The von Mises distribution has a probability density function defined as follows:

$$p(x|\mu, \kappa) = \frac{e^{\kappa \cos(x-\mu)}}{2\pi I_0(\kappa)}, \tag{D.12}$$

where μ and x denote angles, $\kappa \in \mathbb{R}$ is a concentration parameter, and I_0 denotes a modified Bessel function of order 0. The weighted likelihood becomes:

$$\begin{aligned} p(\mathbf{x}|\theta, \mathbf{w}) &= p(\mathbf{x}|\mu, \kappa, \mathbf{w}) \\ p(\mathbf{x}|\theta, \mathbf{w}) &= \prod_{i=1}^{n} p(x_i|\mu, \kappa)^{w_i}. \end{aligned} \tag{D.13}$$

As in Section D.1 above, the logarithm of the density function is derived, as follows:

$$\begin{aligned}
\log p(\mathbf{x}|\theta, \mathbf{w}) &= \log \left(\prod_{i=1}^{n} p(x_i|\mu, \kappa)^{w_i} \right) \\
&= \sum_{i=1}^{n} \log p(x_i|\mu, \kappa)^{w_i} \\
&= \sum_{i=1}^{n} w_i \log p(x_i|\mu, \kappa) \\
&= \sum_{i=1}^{n} w_i \log \left(\frac{e^{\kappa \cos(x_i - \mu)}}{2\pi I_0(\kappa)} \right) \\
&= \sum_{i=1}^{n} w_i \left[\log \left(e^{\kappa \cos(x_i - \mu)} \right) - \log \left(2\pi I_0(\kappa) \right) \right] \\
&= \sum_{i=1}^{n} w_i \left[\kappa \cos(x_i - \mu) - \log(2\pi) - \log I_0(\kappa) \right]. \qquad \text{(D.14)}
\end{aligned}$$

The weighted maximum likelihood estimate for μ is found by first taking the derivative of Equation D.14 with respect to μ:

$$\begin{aligned}
\frac{\partial \log p}{\partial \mu} &= \frac{\partial}{\partial \mu} \sum_{i=1}^{n} w_i \left[\kappa \cos(x_i - \mu) - \log(2\pi) - \log I_0(\kappa) \right] \\
&= \sum_{i=1}^{n} w_i \frac{\partial}{\partial \mu} \left[\kappa \cos(x_i - \mu) - \log(2\pi) - \log I_0(\kappa) \right] \\
&= \sum_{i=1}^{n} w_i \frac{\partial}{\partial \mu} \kappa \cos(x_i - \mu) \\
&= \sum_{i=1}^{n} w_i \kappa \sin(x_i - \mu) \\
&= \kappa \sum_{i=1}^{n} w_i \sin(x_i - \mu). \qquad \text{(D.15)}
\end{aligned}$$

Setting $\frac{\partial \log p}{\partial \mu}$ (Eq D.15) equal to zero,

$$
\begin{aligned}
0 &= \kappa \sum_{i=1}^{n} w_i \sin(x_i - \mu) \\
0 &= \sum_{i=1}^{n} w_i \sin(x_i - \mu) \\
0 &= \sum_{i=1}^{n} w_i \left[\sin(x_i)\cos(\mu) - \cos(x_i)\sin(\mu)\right] \\
0 &= \sum_{i=1}^{n} w_i \sin(x_i)\cos(\mu) - \sum_{i=1}^{n} w_i \cos(x_i)\sin(\mu) \\
\sin(\mu)\sum_{i=1}^{n} w_i \cos(x_i) &= \cos(\mu)\sum_{i=1}^{n} w_i \sin(x_i) \\
\frac{\sin(\mu)}{\cos(\mu)} &= \frac{\sum_{i=1}^{n} w_i \sin(x_i)}{\sum_{i=1}^{n} w_i \cos(x_i)} \\
\tan(\mu) &= \frac{\sum_{i=1}^{n} w_i \sin(x_i)}{\sum_{i=1}^{n} w_i \cos(x_i)} \\
\mu &= \tan^{-1}\left(\frac{\sum_{i=1}^{n} w_i \sin(x_i)}{\sum_{i=1}^{n} w_i \cos(x_i)}\right). \qquad \text{(D.16)}
\end{aligned}
$$

In the case of uniform weights, which corresponds to the maximum likelihood estimate, the value of μ becomes:

$$
\mu = \tan^{-1}\left(\frac{\sum_{i=1}^{n} \sin(x_i)}{\sum_{i=1}^{n} \cos(x_i)}\right). \qquad \text{(D.17)}
$$

Likewise, the weighted maximum likelihood estimate for κ is approximated by first taking the derivative of Equation D.14 with respect to κ:

$$\begin{aligned}
\frac{\partial \log p}{\partial \kappa} &= \frac{\partial}{\partial \kappa} \sum_{i=1}^{n} w_i \left[\kappa \cos(x_i - \mu) - \log(2\pi) - \log I_0(\kappa)\right] \\
&= \sum_{i=1}^{n} w_i \frac{\partial}{\partial \kappa} \left[\kappa \cos(x_i - \mu) - \log(2\pi) - \log I_0(\kappa)\right] \\
&= \sum_{i=1}^{n} w_i \left[\frac{\partial}{\partial \kappa} \kappa \cos(x_i - \mu) - \frac{\partial}{\partial \kappa} \log I_0(\kappa)\right] \\
&= \sum_{i=1}^{n} w_i \left[\cos(x_i - \mu) - \frac{1}{I_0(\kappa)} \frac{\partial}{\partial \kappa} I_0(\kappa)\right] \\
&= \sum_{i=1}^{n} w_i \left[\cos(x_i - \mu) - \frac{I_1(\kappa)}{I_0(\kappa)}\right]. \qquad \text{(D.18)}
\end{aligned}$$

Unfortunately, setting $\frac{\partial \log p}{\partial \kappa}$ (Eq. D.18) equal to zero does not provide a nice closed-form solution, as is the case for $\frac{\partial \log p}{\partial \mu}$. However, one can employ an iterative optimization algorithm, such as gradient descent, or Newton's method, to find a good value for κ, given the gradient $\frac{\partial \log p}{\partial \kappa}$.

Appendix E

Historical Model Optimization Methods

Prior published work on SMRF has reported results that were obtained from question node optimization methods that differ from the one described in Chapter 5. The differences in question mainly concern the construction of decision volumes, including both the pdf and the decision threshold. This appendix provides a brief description of these alternate methods. This material is included purely for historical purposes, and as such is independent of the broader argument being made throughout the rest of this document.

Earlier prior work (Bodenhamer et al., 2009; Sutherland, 2011) utilized an instance-oriented method of building volumes that made use of hidden variables. Later work (Bodenhamer et al., 2012; Palmer et al., 2012) utilized an improved method of building volumes that was group-oriented, and did not make use of hidden variables.

E.1 Optimizing Models With Hidden Variables

This section describes the volume-building approach utilized by Bodenhamer et al. (2009) and Sutherland (2011). This method, while capable of learning concepts, was hampered by a problem that Sutherland (2011) called *hidden variable dilution*. As detailed by Sutherland (2011), the process was overly-sensitive to initial hidden variable values, and often failed to optimize volumes properly as a result. In addition, this method utilized two additional likelihood metrics in addition to the grand metric (Eq. 5.1). Due to these factors, this method was abandoned for the improved method described in Section E.2.

Sutherland (2011) utilized this volume-building method in conjunction with the leaf node probability computation method described in Section 5.4. In contrast, Bodenhamer et al. (2009) utilized this volume-building method in conjunction with a different leaf node probability computation method, which is described in this section. Other aspects of this method, such as the method for choosing leaves to expand, differ from the method described in Chapter 5. The method description is as follows.

The fundamental objective of the SMRF tree learning algorithm is to grow a tree that can accurately predict whether or not a particular graph contains the target concept. However, since the label of each instantiation sequence is unknown, this is an MIL problem. In order to solve this problem, we follow (Zhang and Goldman, 2001) and maintain a set of hidden variables, each of which represents the probability with which a certain instantiation sequence matches the target concept. These hidden variables are denoted h. Specifically, the probability that a specific instantiation sequence I_i^j (drawn from graph G_j) is an instance of the target concept is denoted $h(I_i^j)$. However, this is not quite accurate – for a particular instantiation sequence at the leaves of a tree, h represents the probability that the sequence is an instance of the target concept. More generally, however, for instantiation sequences not a leaf nodes, h denotes the expected number of child instantiation sequences that match the target concept.

The possible values of the $h(I)$'s are constrained in order to reflect the labels of the different graphs. Since negative bags cannot contain the target concept, no instantiation sequences from a negative graph can ever match the target concept. Thus, $h(I_i^j) = 0$ for all $G_j \in G^-$. On the other hand, positive bags must contain the target concept, so it follows that $h(I_i^j)$ must sum to a quantity greater than or equal to one for all $G_j \in G^+$. Furthermore, the $h(I)$'s corresponding to a collection of child instantiation sequences must sum to the $h(I)$ of the parent instantiation sequence. Expressed formally, the constraints are as follows:

- $0 \leq h(I_i^j) \leq |(G_i^j)|$ for all I_i^j
- $0 \leq h(I) \leq 1$ for all $I \in^{\mathcal{L}} \mathcal{I}$
- $h(I) = 0$ for each $I \in {}^0\mathcal{I}^j$ and $G_j \in G^-$
- $h(I) \geq 1$ for each $I \in {}^0\mathcal{I}^j$ and $G_j \in G^+$
- $h({}^pI_i^j) = \sum_{I \in {}^c\mathcal{I}_i^j} h(I)$. for all I arriving at an instantiation node.

The SMRF tree probabilistically classifies a graph as containing the target concept based upon the probability of the highest-probability leaf node into which an instantiation from the graph in question is sorted. The learning algorithm determines these values from the $h(I)$'s of the instantiation sequences from the training set that are sorted down into each leaf k. The leaf node probabilities themselves are estimated by "soft" counting, and this is formally expressed as:

$$Pr(k) \leftarrow \sum_{I \in {}^k\mathcal{I}} h(I) / \left|{}^k\mathcal{I}\right|. \tag{E.1}$$

Since the algorithm is designed to grow a tree that can accurately probabilistically classify a given graph, it seeks to build a tree that will maximize the likelihood of correct graph classification. The likelihood of the correct classification (L) is defined as follows:

$$L = \prod_{G_j \in G^+} \max_{I \in {}^{\mathcal{L}}\mathcal{I}^j} \Pr(\mathcal{L}(I)) \prod_{G_j \in G^-} \max_{I \in {}^{\mathcal{L}}\mathcal{I}^j} (1 - \Pr(\mathcal{L}(I))) \tag{E.2}$$

The intuition behind Equation E.2 is that the likelihood of the data given the tree is higher when instantiation sequences from positive graphs are sorted into leaf nodes with a high probability, and lower when no instantiation sequences from a positive graph are sorted into a high-probability leaf. Conversely, the likelihood of the data is also higher when no instantiation sequences from negative graphs are sorted into high-probability leaves, and lower when at least one such instantiation sequence is sorted into a high-probability leaf.

In contrast to the method described in Section 5.2, the leaf selection criterion used in this approach is the increase in likelihood gained by expanding that leaf by a question node, assuming that the question node sorts the instantiation sequences at that node optimally. The optimal hypothetical split point is determined using the Kolmogorov-Smirnov (K-S) distance between the cumulative distributions of the $h(I)$'s and the $(1 - h(I))$'s (cf. Friedman, 1977a). The point of maximum K-S distance is chosen as the threshold, and all I's whose $h(I)$ are less than or equal to the threshold are sorted into one hypothetical leaf, and the others are sorted into the other hypothetical leaf. The hypothetical leaf probabilities are computing using the soft count of the instantiation sequences at those leaves (cf. Equation E.1), and the likelihood L of the new hypothetical tree is computed, taking the hypothetical leaves into account. The m nodes with the highest such increase of hypothetical likelihood are chosen as the set of leaves to be expanded. In prior work utilizing this method, m was empirically set to 3, as this allowed the algorithm to explore a good number of expansion paths, without exploring more than necessary.

Once a leaf has been selected for expansion, its expansion method chosen, and the mapping function for the candidate question node decided, the algorithm then proceeds to maximize the model parameters of the candidate question node q. First, an initial guess is made of the values of the $h({}^{q}I_i^j)$'s. Second, parameters θ_q are chosen to maximize the *model data likelihood* $\hat{L}$. The model data likelihood represents the likelihood of the $h({}^{q}I_i^j)'s$ and the values of the model pdf when it evaluates the given instantiation sequences. The intuition is that instantiation sequences with a higher $h({}^{q}I_i^j)$ value that have a higher model likelihood should increase the value of $\hat{L}$, and that sequences with a high $h({}^{q}I_i^j)$ value that have a lower model likelihood should decrease the value of $\hat{L}$. Formally, the model data likelihood is defined as follows:

$$\hat{L}(q) = \prod_{I \in {}^{q}\mathcal{I}} p\left(\phi_q(I)|\theta_q\right)^{h(I)} . \tag{E.3}$$

The values of θ_q that maximize $\hat{L}$ are in derived in Appendix D.

The model threshold Θ_q is determined from the $h({}^qI_i^j)$'s, using the Kolmogorov-Smirnov (K-S) distance between the cumulative distribution of the $h({}^qI_i^j)'s$ and the $(1 - h({}^qI_i^j))$'s. The intuition is that the algorithm should find the point where the most-probably-positive instantiation sequences are best separated from the most-probably-negative instantiation sequences. Or in other words, the threshold which maximizes the K-S threshold between the cumulative distributions of positive and negative examples is the threshold that should be chosen (Friedman, 1977a; Haskell, 1993). The cumulative distribution of the $h({}^qI_i^j)'s$ is denoted $F_p(x, q)$, and is defined as:

$$\frac{\sum_{I \in {}^q\mathcal{I}:\ p(\phi_q(I)|\theta_q) \geq \Theta_q} h(I)}{\sum_{I \in {}^q\mathcal{I}} h(I)} \tag{E.4}$$

Similarly, the cumulative distribution of the $(1 - h({}^qI_i^j))$'s is denoted $F_n(x, q)$, and is defined as:

$$\frac{\sum_{I \in {}^q\mathcal{I}:\ p(\phi_q(I)|\theta_q) \geq \Theta_q} (1 - h(I))}{\sum_{I \in {}^q\mathcal{I}} (1 - h(I))} \tag{E.5}$$

Then model threshold, then, is calculated as follows:

$$\Theta_q = \arg\max_x \left| F_p(x, q) - F_n(x, q) \right|. \tag{E.6}$$

Next, the instantiation sequences are sorted into the appropriate leaves and the leaf node probabilities are updated using Equation E.1. Given new leaf node probabilities, new $h(I_i^j)$ are chosen to maximize the instance-sorting likelihood $\overline{L}$:

$$\bar{L} = \prod_{I \in \mathcal{LI}} \Pr(\mathcal{L}(I))^{h(I)} \times (1 - \Pr(\mathcal{L}(I)))^{1-h(I)}. \tag{E.7}$$

Such maximization is subject to the aforementioned constraints upon the possible

values of h. It should be noted that Bodenhamer et al. (2009) used $\bar{L}$ in lieu of L for the grand metric, resulting in two metrics used ($\bar{L}$ and $\hat{L}$), whereas Sutherland (2011) used L for the grand metric, while also using $\bar{L}$ for the selection of new values of h, resulting in three total metrics used (L, $\hat{L}$, and $\bar{L}$).

After computing new values of h, a new value of L computed with the new $h(I_i^j)$'s is compared to the L computed with the old $h(I_i^j)$'s. If the difference is small enough, then iteration stops. After the candidate trees with the sampled models have been learned, the tree T with the highest likelihood L_T is chosen. Per the procedure described in Section 5.5, if the likelihood L_T of the best candidate tree is statistically significantly better than the likelihood L of the current tree, then the current tree is replaced with the candidate tree.

E.2 Optimizing Models Without Hidden Variables

This section describes the volume-building approach utilized by Bodenhamer et al. (2012) and Palmer et al. (2012). This method, while more effective than the method described in Section E.1, was ultimately abandoned for the method described in Section 5.3, as that method is more compact, unified approach that is based solely upon the grand metric (Eq. 5.1), and does not rely upon a second likelihood $\hat{L}$. This method is essentially an alternative to the procedure described in Section 5.3, and as such can be used in conjunction with the methods described in the other sections of Chapter 5. The description of the method is as follows.

For a given candidate tree, the parameters of the pdf at the expansion question node are optimized so as to capture the output of the mapping function for the "best" instantiation sequences. Each instantiation sequence is given a weighting, and the maximally-weighted instantiation sequences from each positive group are used to estimate the model parameters.

Models are learned using an iterative process. The question node model q being optimized is initialized with pdf parameters θ_q^1, which are selected based on a point of maximal diverse density (Maron, 1998). This procedure is described in more detail in Section E.2.1.

Based on the current model, each instantiation sequence I at iteration t is assigned a weight $w_t(I)$,[1] which is defined such that

$$w_t(I) = p(\phi_q(I)|\theta_q^t). \tag{E.8}$$

New model parameters θ_q^{t+1} are estimated to maximize the *weighted model likelihood*, which reflects the degree to which the new pdf captures the weighted output of the mapping function when evaluated on instantiation sequences from positive groups. Put another way, the weighted model likelihood expresses how well the new question node model captures the "important" instances at that node that are drawn from positive bags, where the instantiation sequence weight can be interpreted as a measure of "importance." The weighted model likelihood, denoted $\hat{L}\left(\theta_q^{t+1}|G^+\right)$, is defined as

$$\hat{L}(\theta_q^{t+1}|G^+) = \prod_{G \in G^+} p(\phi_q(I_G^q)|\theta_q^{t+1}))^{w_t(I_G)}, \tag{E.9}$$

where I_G^q denotes the highest-weighted instantiation sequence from G at the question node q being optimized. In order to formally define I_G^q, let $\mathfrak{W}_q(G)$ be a function of the form $\mathfrak{W}_q : \wp(\mathcal{O}) \rightarrow \Upsilon$, that maps groups to maximal instantiation sequences, as defined by the model at question node q. $\mathfrak{W}_q$ is formally defined as

$$\mathfrak{W}_q(G) = \underset{I \in {}^q\mathcal{I}^G}{\arg\max}\ p\left(\phi_q(I)|\theta_q\right), \tag{E.10}$$

where ${}^q\mathcal{I}^G$ denotes the set of instantiation sequences at the question node q drawn

[1] For the model resulting from the final iteration, this is simply denoted $w(I)$.

from group G. Making iterations explicit, let $\mathfrak{W}_q^t(G)$ be defined as follows, in accordance with Eq. E.8:

$$\mathfrak{W}_q^t(G) = \arg\max_{I \in {}^q\mathcal{I}^G} w_t(I). \tag{E.11}$$

Making use of Eq. E.11, I_G^q is formally defined as

$$I_G^q = \mathfrak{W}_q^t(G). \tag{E.12}$$

This process of assigning weights to instantiation sequences and estimating model parameters accordingly is repeated until $\hat{L}$ converges. The process of maximizing $\hat{L}$ is described in more detail in Section E.2.2.

E.2.1 Parameter Initialization Using Diverse Density

We utilize Diverse Density (DD) (Maron, 1998) in order to provide an intelligent parameter initialization for the question node model. Intuitively, the point of diverse density represents the optimal balance between the separation of positive instances from negatives and the concentration of positive instances, in the codomain of a given mapping function. Thus, DD gives us a good measure for determining a start point for our models, as we want to find models that separate as many positive instances from negative instances as possible.

Formally, we use the most-likely-cause estimator (Maron, 1998), which, for a point t in the codomain of ϕ_q, is defined as

$$\hat{DD}_q(t, \mathcal{D}) = \left(\prod_{G \in G_{\mathcal{D}}^+} \max_{I \in {}^q\mathcal{I}^G} \Pr\left(\phi_q(I) \in c_t\right) \right) \times \left(\prod_{G \in G_{\mathcal{D}}^-} 1 - \max_{I \in {}^q\mathcal{I}^G} \Pr\left(\phi_q(I) \in c_t\right) \right), \tag{E.13}$$

where $\Pr\left(\phi_q(I) \in c_t\right)$ represents the probability that the concept defined by I matches

the concept defined by t, with respect to ϕ_q.

We follow Maron (1998) in calculating $\Pr(\phi_q(I) \in c_t)$ through a Gaussian-like probability calculation (that is, a multivariate Gaussian normalized to lie in the range [0,1] rather than to have total probability mass of 1). The covariance matrix of this Gaussian is calculated by computing the variance of $\phi_q(I)$ for all of the positive instantiation sequences at q along each dimension of codomain of ϕ_q, and then multiplying by a "kernel factor" which serves to scale the volume defined by the covariance matrix to an appropriate size. For this work, the kernel factor was empirically set to 0.2.

To initialize the model parameters, we first find the point t that maximizes $\hat{DD}_q(t, \mathcal{D})$. This is done by evaluating all of the instantiation sequences present at q from m positive groups. The rationale here is that if the group labeling is correct, at least one instantiation sequence will have a point near the true point of maximal diverse density. In order to provide robustness to corruption in the group labeling, multiple bags are considered. In this work, m was empirically set to 2.

Given an approximate point of maximal diverse density, we then set the mean of the initial pdf to this point. The initial standard deviation is computed, as above, by computing the variance of $\phi_q(I)$ for all of the positive instantiation sequences at q along each dimension of codomain of ϕ_q. In order to remain robust to potential cases where the point of greatest diverse density does not produce good initial parameters, models are derived for the n highest points of diverse density. In this work, n was empirically set to 2.

E.2.2 Maximizing the Model Likelihood

The model parameters θ_q are estimated using the maximum-likelihood (ML) estimation of $\hat{L}$ (Eq. E.9). For a Gaussian pdf, it can be shown that $\hat{\mu}_q^{t+1}$ and $\hat{\Sigma}_q^{t+1}$, which maximize $\hat{L}$, are

$$\hat{\mu}_q^{t+1} = \frac{\sum_{I \in {}^q\mathcal{I}} w_t(I)\phi_q(I)}{\sum_{I \in {}^q\mathcal{I}} w_t(I)}, \tag{E.14}$$

and

$$\hat{\Sigma}_q^{t+1} = \frac{\sum_{I \in {}^q\mathcal{I}} w_t(I)\left(\phi_q(I) - \hat{\mu}_q^{t+1}\right)\left(\phi_q(I) - \hat{\mu}_q^{t+1}\right)^T}{\sum_{I \in {}^q\mathcal{I}} w_t(I)}, \tag{E.15}$$

where ${}^q\mathcal{I}$ denotes the total set of instantiation sequences present at question node q. Equations E.14 and E.15 follow from Equations D.8 and D.11, respectively, in Appendix D.

E.2.3 Computing Decision Thresholds

Once the final parameters θ_q of the pdf model have been determined, a decision threshold is computed to determine how the model will sort instantiation sequences. Making use of the approach of Friedman (1977b), we use the point of maximum Kolmogorov-Smirnov (K-S) distance between $p(\phi(I_G^q)|\theta_q)$'s of the positive groups and the $p(\phi(I_G^q)|\theta_q)$'s of the negative groups as the decision boundary, where I_G^q is defined in accordance with Eq. E.12. After determining the parameters of the question node model, the instantiation sequences are sorted into the appropriate child leaf nodes.

The K-S distance is determined by finding the point that maximizes the absolute difference of two empirical distribution functions (EDFs). In this context, the EDF is denoted $F_q(x, S)$, and is a function of the form $F_q : \mathbb{R} \times \wp(\wp(\mathcal{O})) \to \mathbb{R}$, where $x \in \mathbb{R}$ and S is a set of groups, such that $S \subseteq \Gamma_{\mathcal{D}}$. In order to formally define the EDF F_q, let $\mathfrak{L}_q(x, S)$ denote a function of the form $\mathfrak{L}_q : \mathbb{R} \times \wp(\wp(\mathcal{O})) \to \wp(\Upsilon)$, which maps a proper subset of $\Gamma_{\mathcal{D}}$ to the maximum instantiation sequence from each group as evaluated by the model of the question node q. Making use of Eq. E.10, $\mathfrak{L}_q(x, S)$ is formally defined as

$$\mathfrak{L}_q(x, S) = \{I \mid (\exists G)\ [G \in S \wedge I = \mathfrak{W}_q(G) \wedge p\left(\phi_q(I)|\theta_q\right) \leq x]\}\,. \tag{E.16}$$

Making use of Eq. E.16, the EDF F_q is defined as

$$F_q(x, S) = \sum_{I \in \mathfrak{L}_q(x,S)} p\left(\phi_q(I)|\theta_q\right). \tag{E.17}$$

Given Eq. E.17, the decision threshold Θ_q is determined as follows:

$$\Theta_q = \arg\max_x \left|F_q(x, G^+) - F_q(x, G^-)\right| \tag{E.18}$$

In practice, Θ_q is computed by sweeping across the range of values produced by evaluating the question node model on the set of instantiation sequences present at q.

Appendix F

Dataset Visualizations

To provide a sense of what the data look like, a couple of problem sets are visualized here, at a couple of different distractor levels. Rather than trying to visualize a set of individual examples, the following visualizations are designed to give the reader a "SMRF's-eye" view of the data. That is, the entire dataset is processed through the various mapping functions listed in Section 7.1.1.

Three datasets are visualized in this Appendix: *Blue above Green*, *Red or Green*, and *Red above Green or Blue above Yellow*. These datasets were chosen because they are representative of the three different kinds of problems that were presented to SMRF in the course of the experimental work described in Chapter 7: purely conjunctive concepts, purely disjunctive concepts, and complex mixed concepts.

For each of the datasets, visualizations are provided for both *Identity Color* and *Identity Location*. In addition, where examples contain more than one object, *Difference Color* and *Difference Location* visualizations are also provided. For the first problem set (*Blue above Green*), visualizations at two distractors are also given, to provide the reader with a sense of how the addition of distractors affects the data.

Each visualization is presented as 2×2 grid of figures. The top-left figure depicts the mapping function codomain in three dimensions, and is followed by three figures depicting the xy, xz, and yz planes, respectively. These figures were generated by instantiating the entire training set, once in the case of *Identity* mapping functions, and twice in the case of *Difference* mapping functions. The figures show the values given to each instantiation sequence by the mapping function in question. Values

corresponding to instantiation sequences from positive examples are marked as red triangles, and values corresponding to instantiation sequences from negative examples are marked as blue circles. Please note that, as such, the color of the marker does **not** correspond in any way to the color attribute value of any particular object.

F.1 Blue above Green

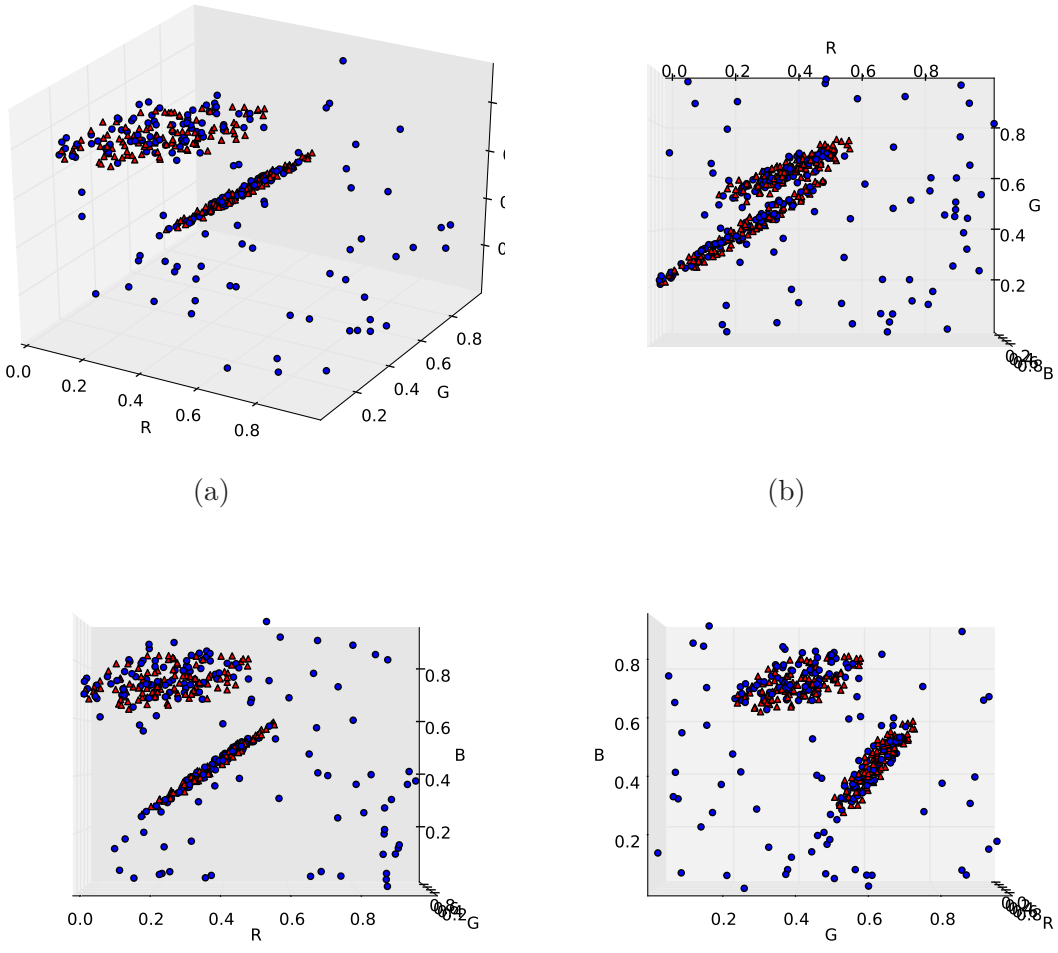
F.1.1 Zero Distractors

Figure F.1 shows the distribution of *Identity Color* values of all of the examples in the *Blue above Green* dataset, in RGB space. Figure F.1a shows a three-dimensional view of this space, while Figure F.1b shows the red-green plane. Likewise, Figure F.1c shows the red-blue plane, and Figure F.1d shows the green-blue plane. As one can see, there are two distinct clusters of points, one in the green region of the space, and the other in the blue region of the space. Red points are only found in these two clusters, which shows that no positive example has an object with a color other than "green" or "blue." Likewise, there are blue points in both clusters, which shows that there are negative examples that have "green" and "blue" objects.

Figure F.2 shows the distribution of *Difference Location* values of the *Blue above Green* dataset, in RGB space. There are two symmetric clusters of red points indicating a "blue"-"green" color difference, which is most clearly seen in Figure F.2d. This indicates that each positive example has two objects: one "blue" and the other "green." There are few blue points in these clusters, showing that there are a few negative examples that have both a "blue" object and a "green" object.[1]

Figure F.3 shows the distribution of *Identity Location* values of the *Blue above Green* dataset, in three-dimensional position space. Figure F.3a shows a three-dimensional view of this space, while Figure F.3b shows the xy plane. Likewise,

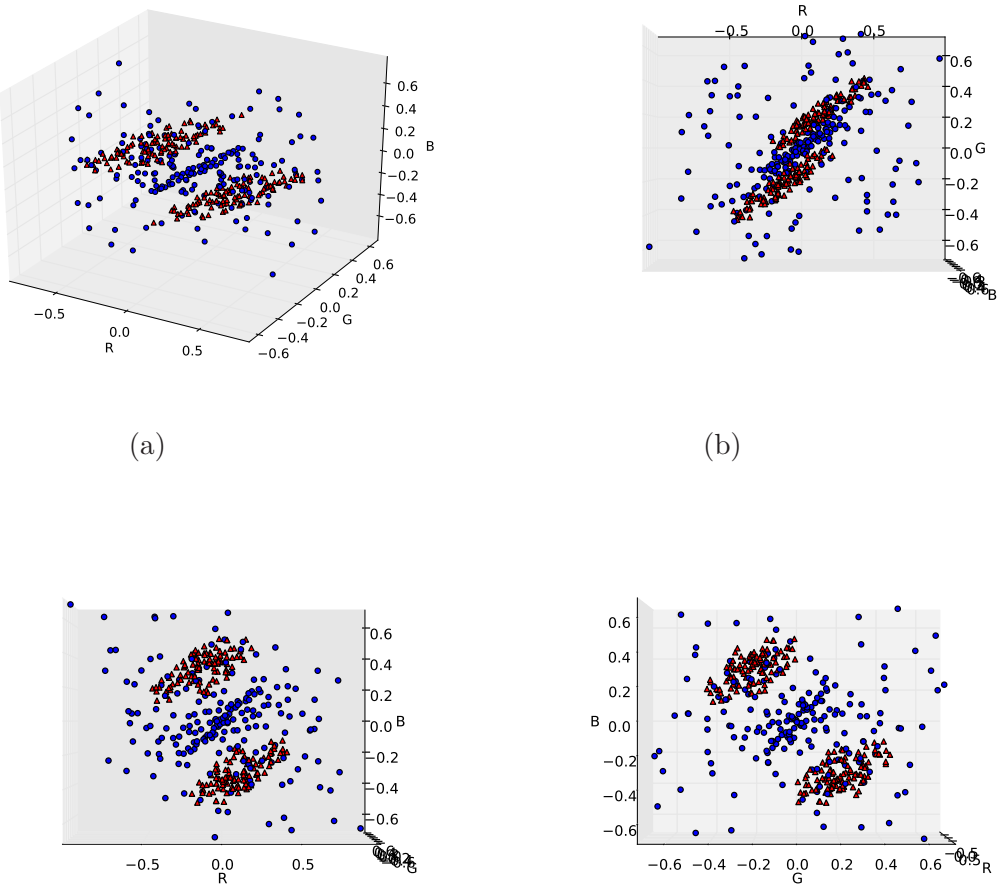
[1] In such cases, however, the objects do not satisfy the *above* relation.



(a) (b)

(c) (d)

Figure F.1: *Blue above Green* at zero distractors, viewed according to *Identity Color*.



(a) (b)

(c) (d)

Figure F.2: *Blue above Green* at zero distractors, viewed according to *Difference Color*.

Figure F.3c shows the xz plane, and Figure F.3d shows the yz plane. As one can see, both red and blue points are randomly distributed throughout the space, which indicates that the absolute position of objects plays no role in the target concept.

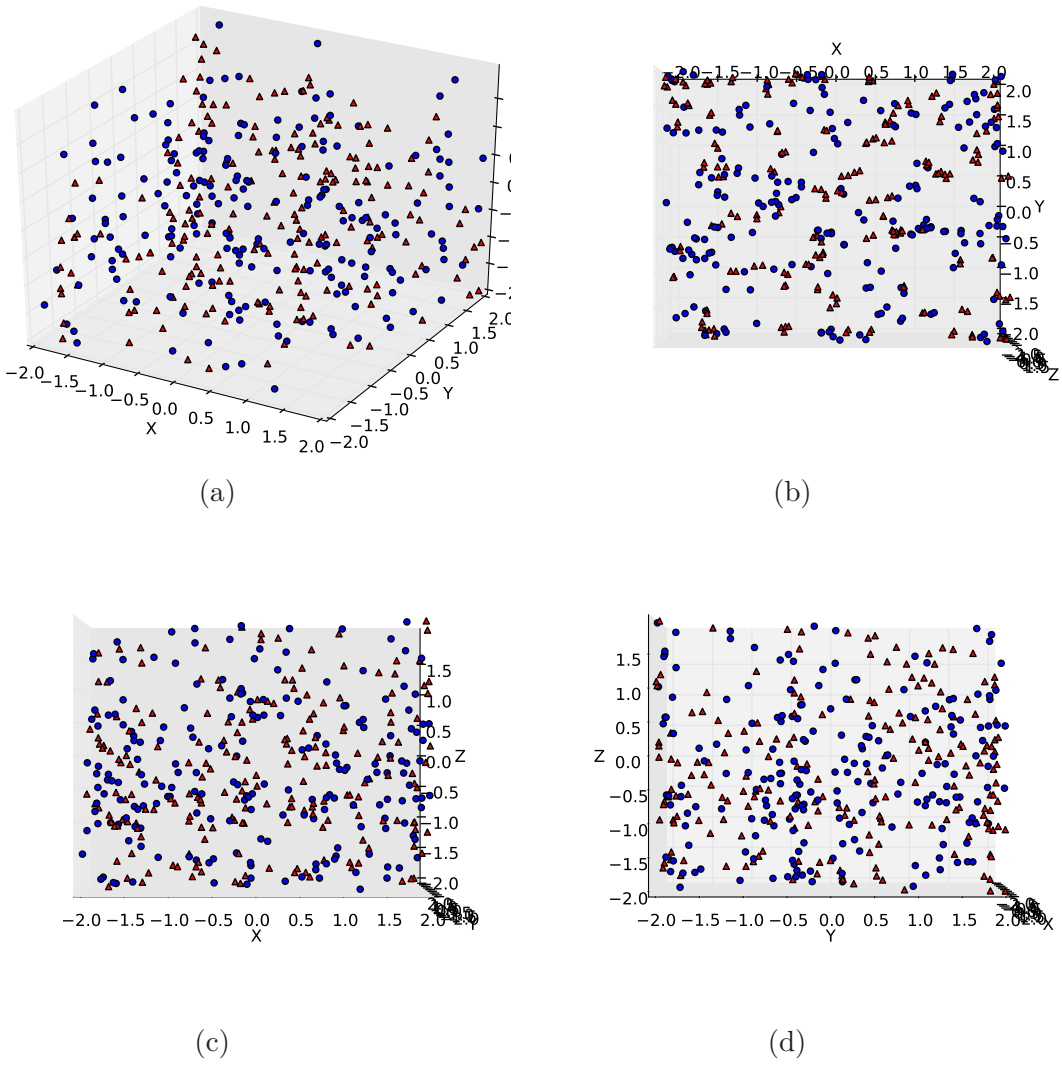
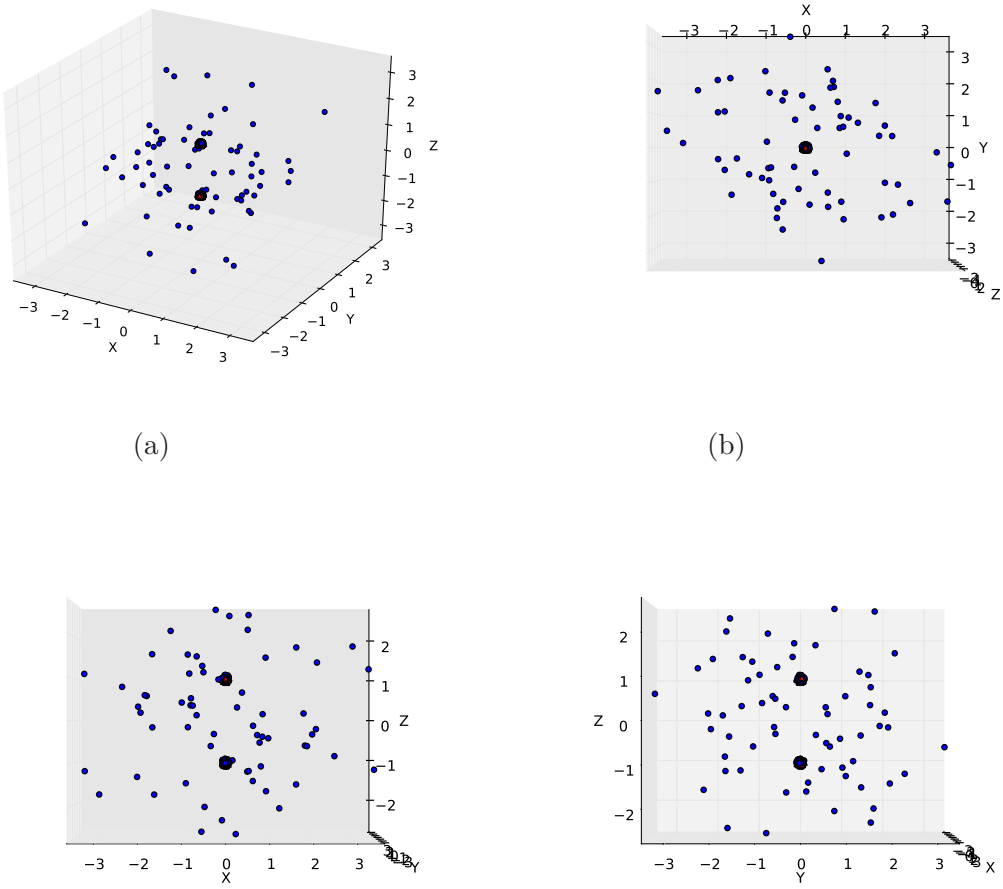


Figure F.3: *Blue above Green* at zero distractors, viewed according to *Identity Location*.

Figure F.4 shows the distribution of *Difference Location* values of the *Blue above Green* dataset, in three-dimensional difference location space. There are two clusters centered at $(0, 0, 1)$ and $(0, 0, -1)$ respectively, which correspond to the difference location values of pairs of objects that satisfy the *above* relation. All of the red points

are in one of these two clusters, indicating that all positive examples have exactly two points, which together satisfy the *above* relation. There are a few blue points in these clusters, which correspond to negative examples containing a pair of points satisfying the *above* relation.[2]



(a) (b)

(c) (d)

Figure F.4: *Blue above Green* at zero distractors, viewed according to *Difference Location*.

[2]In such cases, however, one of the objects is either not "blue" or not "green".

F.1.2 Two Distractors

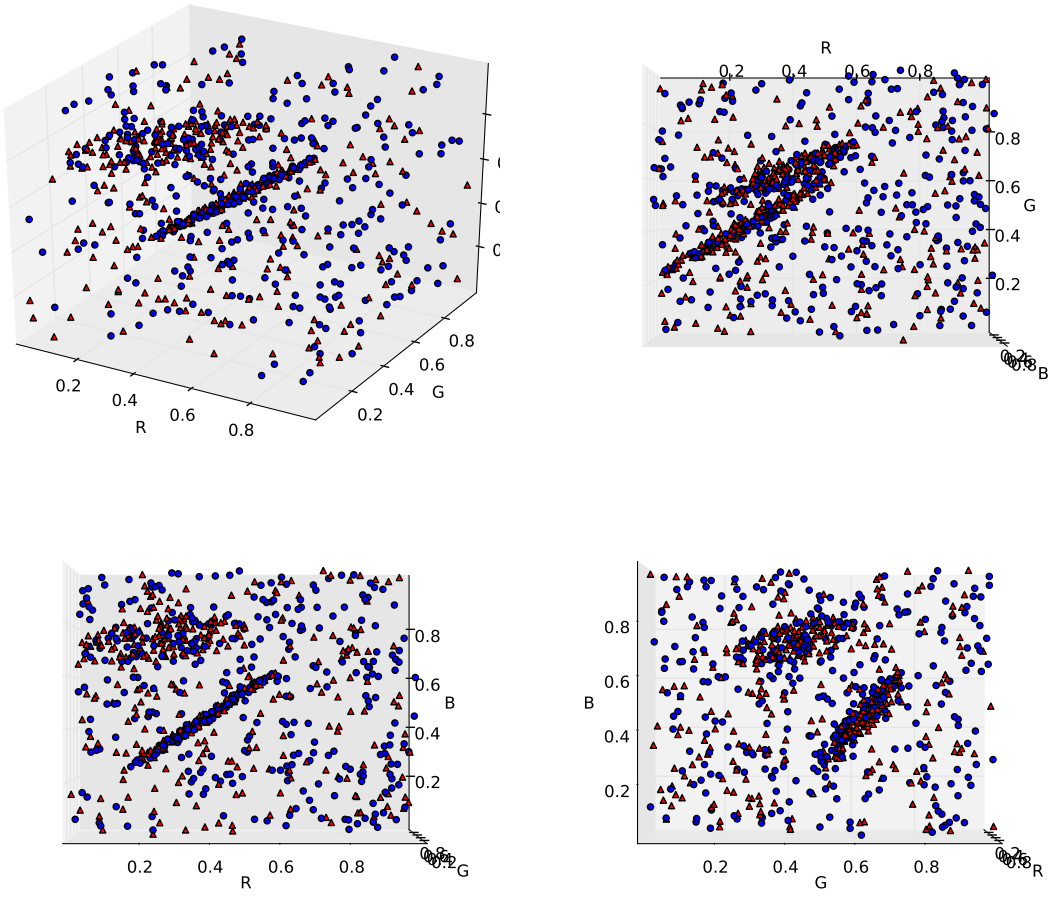


Figure F.5: *Blue above Green* at two distractors, viewed according to the mapping function *Identity Color*.

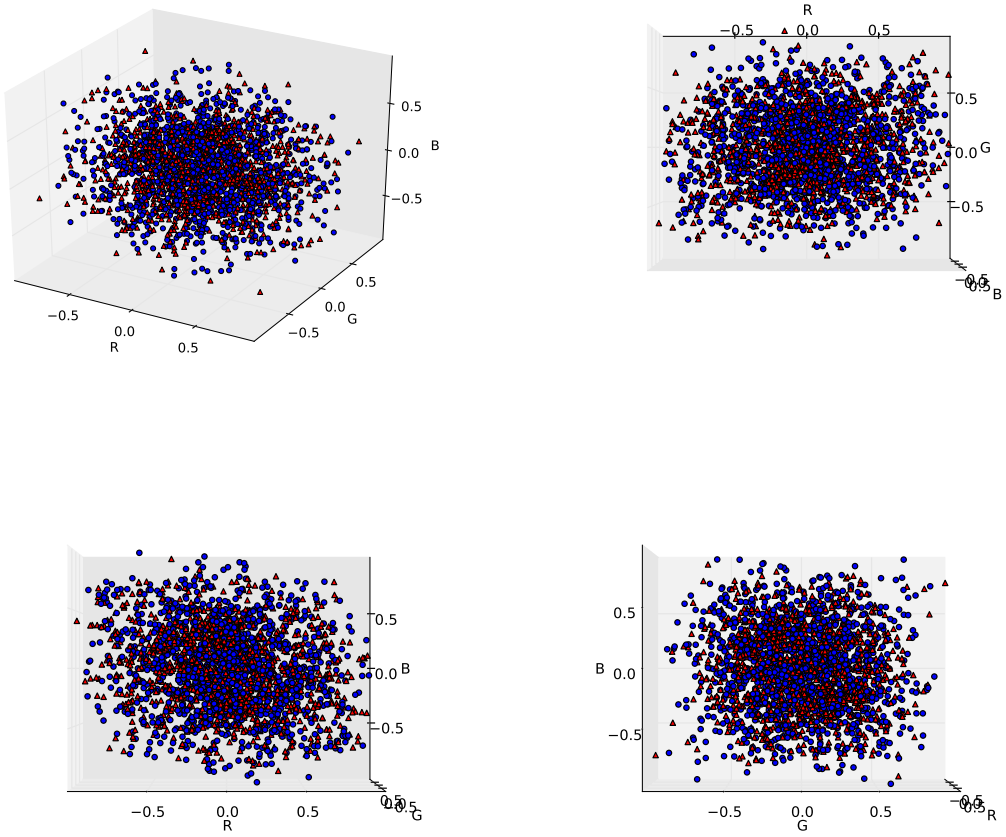


Figure F.6: *Blue above Green* at two distractors, viewed according to *Difference Color*.

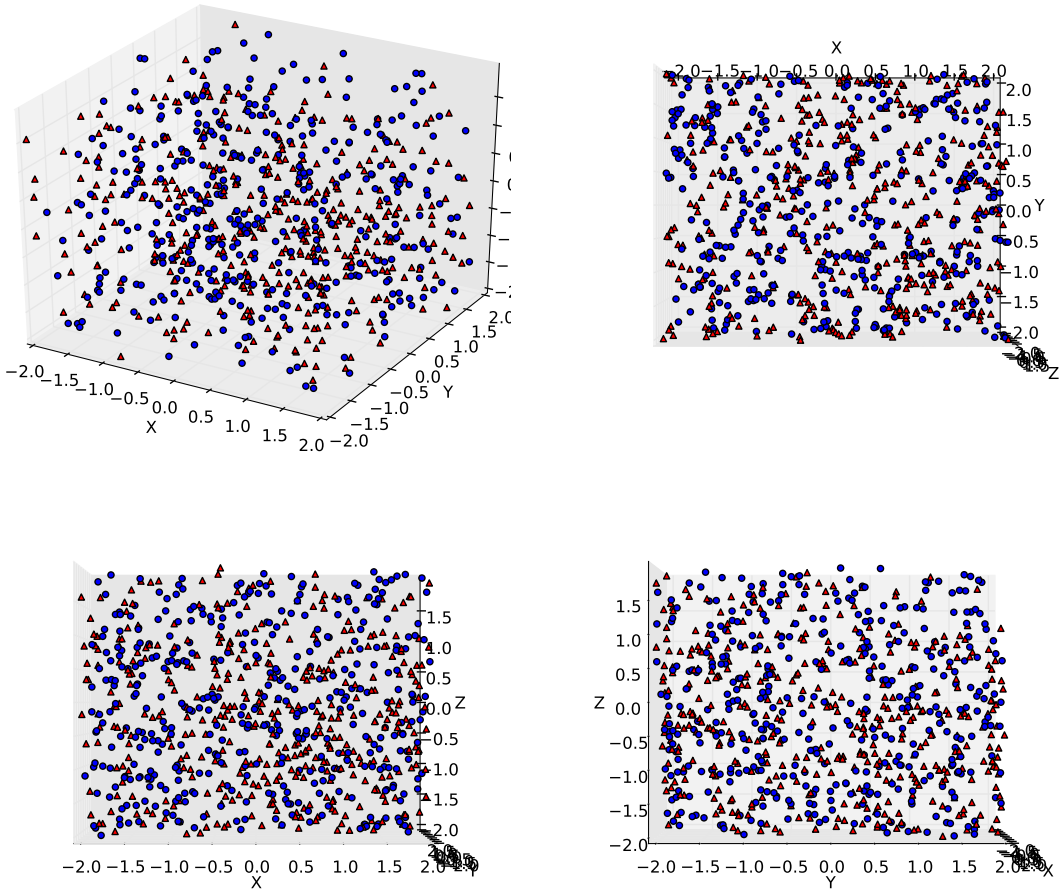


Figure F.7: *Blue above Green* at two distractors, viewed according to *Identity Location*.

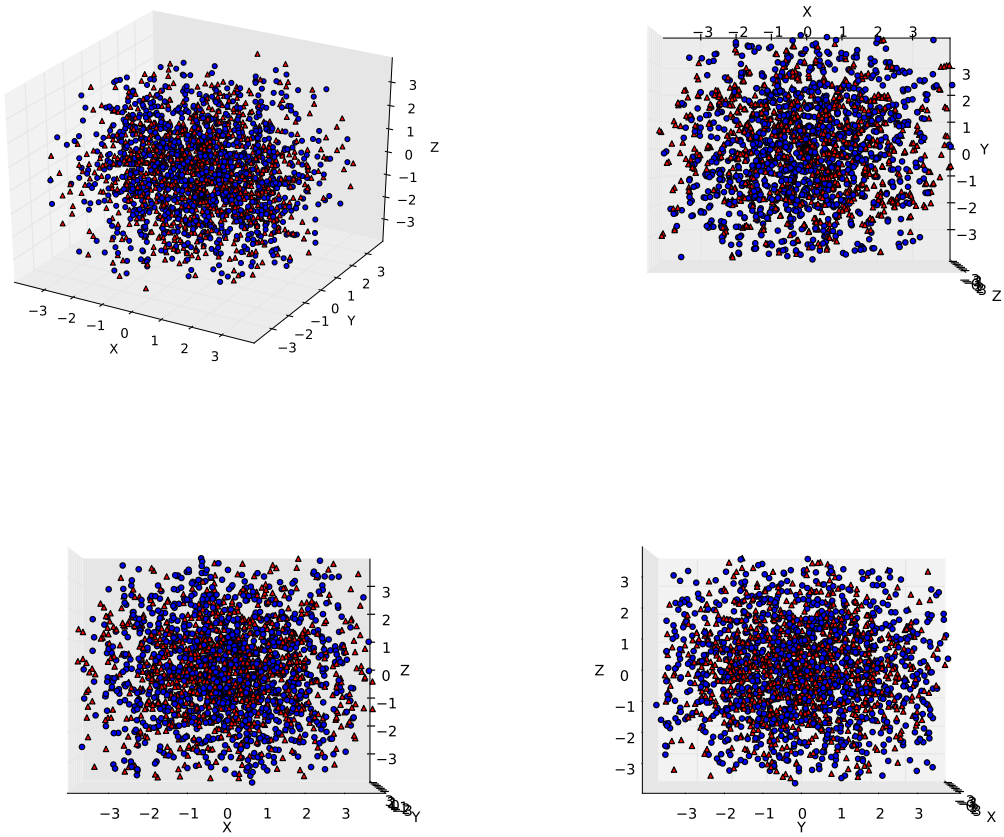


Figure F.8: *Blue above Green* at two distractors, viewed according to *Difference Location*.

F.2 Red or Green

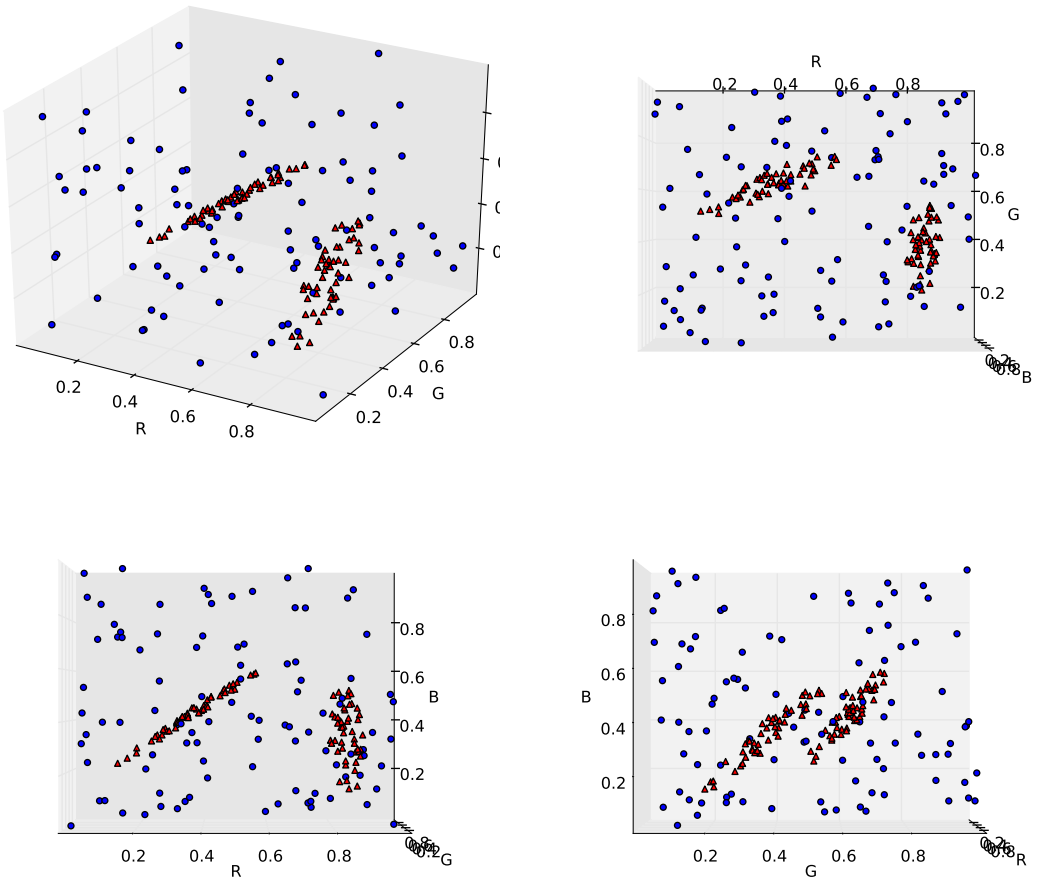


Figure F.9: *Red or Green* at zero distractors, viewed according to *Identity Color*.

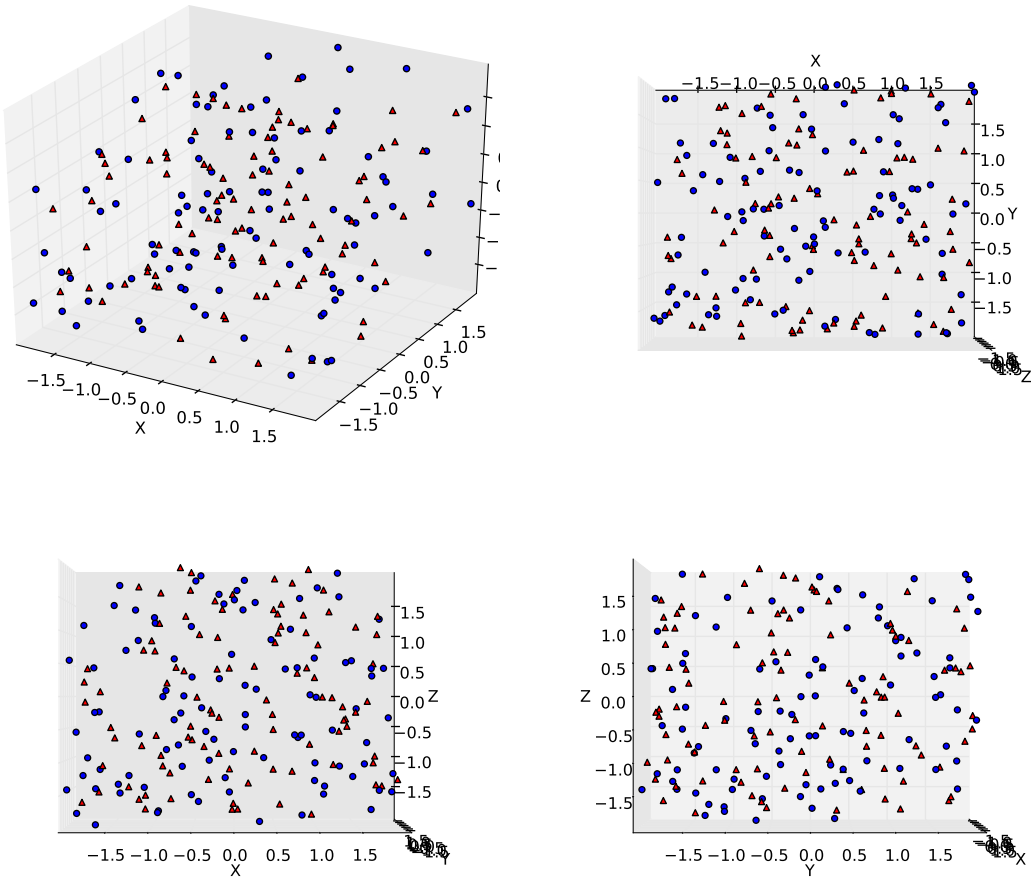


Figure F.10: *Red or Green* at zero distractors, viewed according to *Identity Location*.

F.3 Red above Green or Blue above Yellow

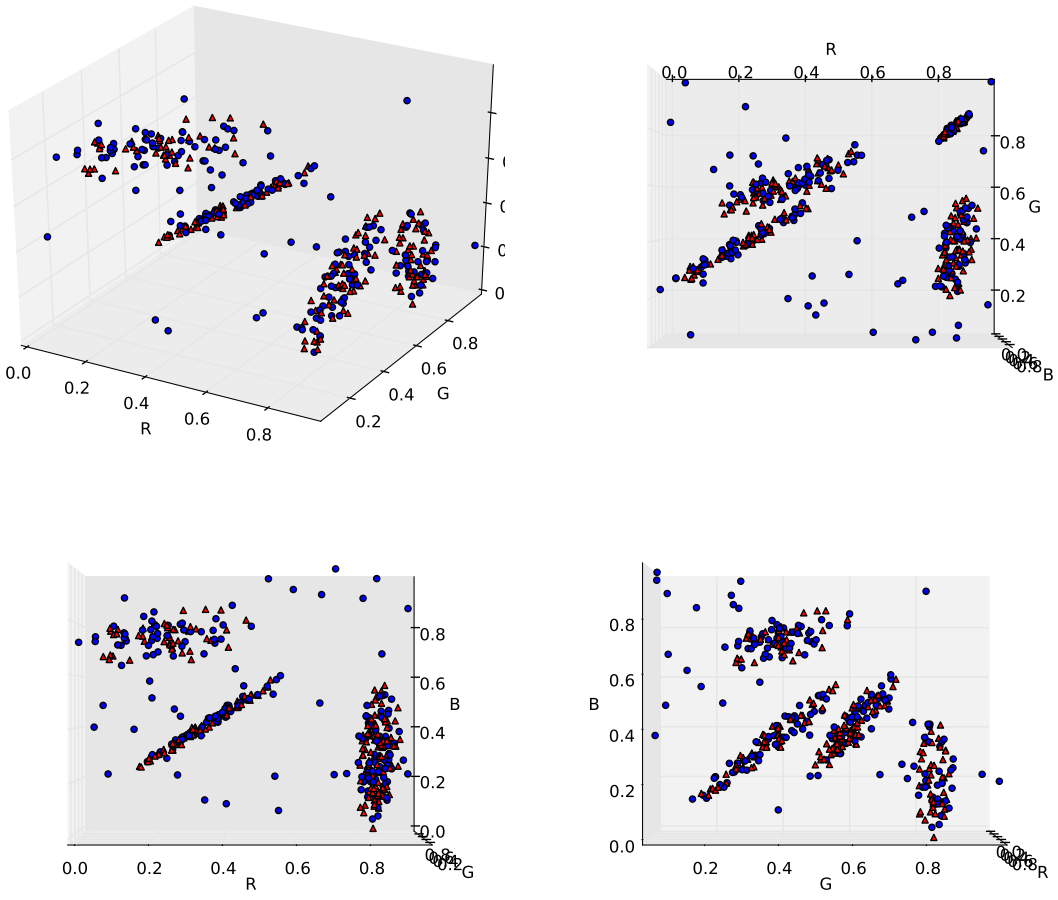


Figure F.11: *Red above Green or Blue above Yellow* at zero distractors, viewed according to *Identity Color*.

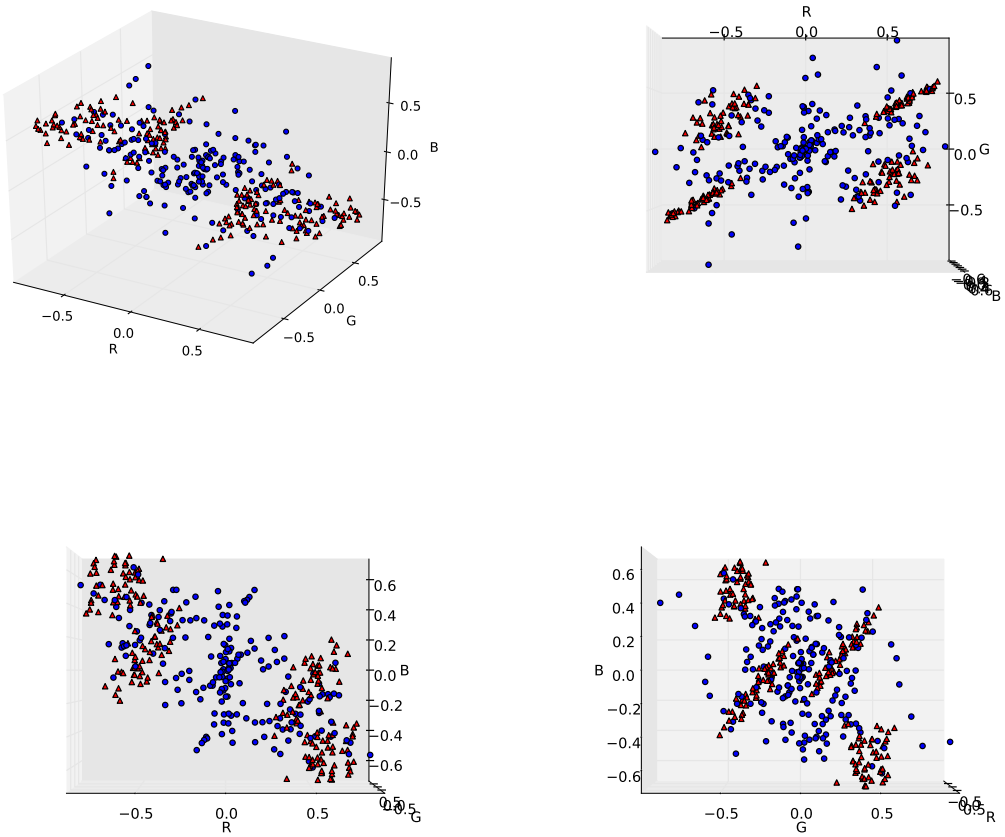


Figure F.12: *Red above Green or Blue above Yellow* at zero distractors, viewed according to *Difference Color*.

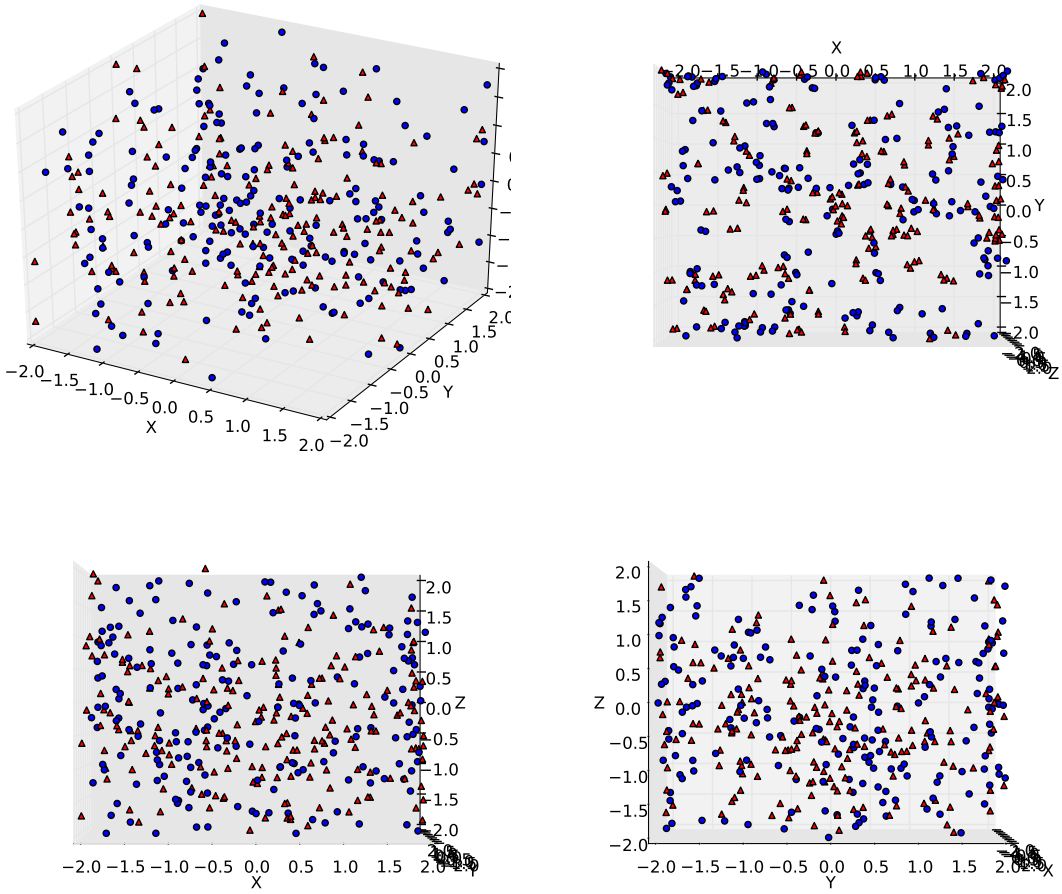


Figure F.13: *Red above Green or Blue above Yellow* at zero distractors, viewed according to *Identity Location*.

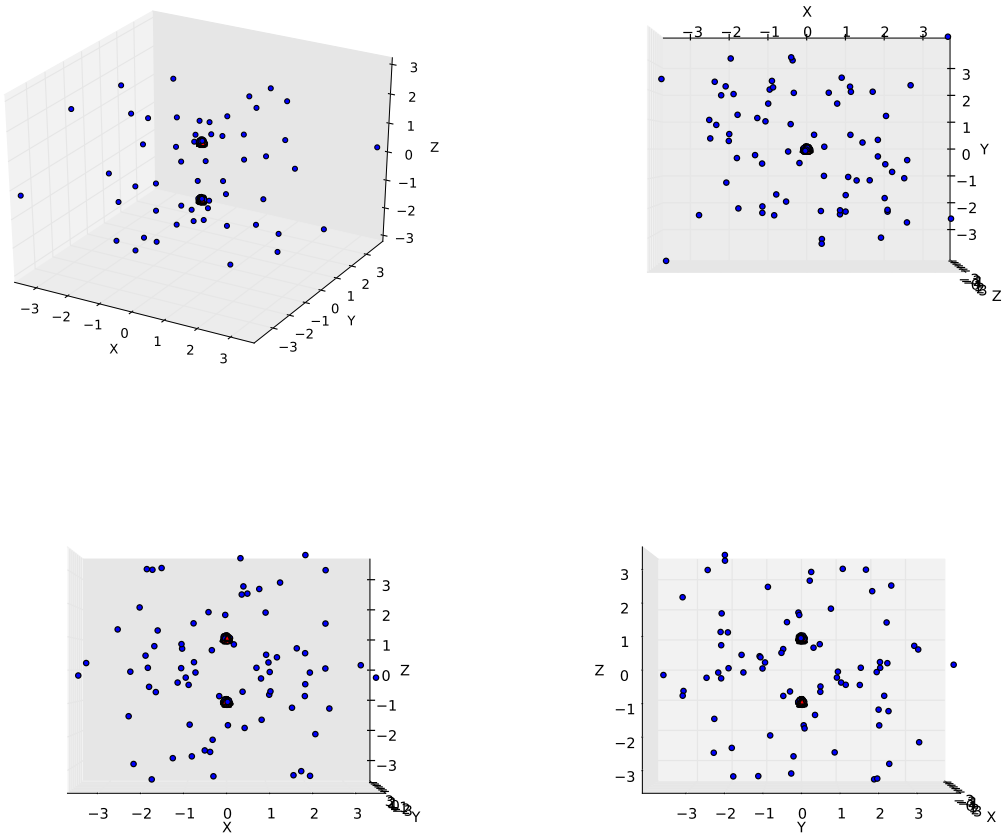


Figure F.14: *Red above Green or Blue above Yellow* at zero distractors, viewed according to *Difference Location*.