Learning Prototypical Concept Descriptions

Piew Datta and Dennis Kibler
Department of Information and Computer Science
University of California
Irvine, CA 92717
{pdatta, kibler}@ics.uci.edu

Abstract

We describe a new representation for learning concepts that differs from the traditional decision tree and rule approach. This representation, called prototypical concept descriptions, can represent several prototypes for a concept. We also describe PL, our algorithm for learning these prototypes, and demonstrate that prototypical concept descriptions can, in some situations, classify more accurately than standard Machine Learning algorithms. More importantly, we show that they yield more stable descriptions when applied in noisy and dynamic situations.

1 INTRODUCTION

Many machine learning algorithms use the minimum description length (MDL) (Rissanen 1976) bias to choose between two or more theories that classify equally well. Examples include most decision tree methods such as ID3 (Quinlan 1986) and other rule based algorithms, such as CN2 (Clark & Boswell 1991). By choosing only those features that discriminate classes as part of the learned theory and preferring simpler concepts, these algorithms follow the MDL bias.

Bayesian classifiers (Duda & Hart 1973), neural networks (Rosenblatt 1958; Rumelhart, Hinton, & Williams 1986), and instance-based methods such as IB1 (Aha, Kibler, & Albert 1991) and PEBLS (Cost & Salzberg 1993), on the other hand, use all of the relevant features to represent concept theories, even those that are redundant, but informative. For example, suppose one is learning to distinguish 1's from 3's with a standard 7 bit LED representation. This example illustrates two types of redundant features. The vertical bars have a *common redundancy*, since they are present in both classes, and will not be considered as part of a concept theory by any machine learning algorithm based on the MDL principle. Second the horizontal bars have a *group redundancy* since any one horizontal bar is sufficient to discriminate the two digits. In this case, MDL based algorithms will randomly select one of the horizontal bars as the only relevant feature, leading to instability in the final concept description. Although standard Bayesian, neural network, and instance-based methods can represent redundant features, they do not yield concise, comprehensible concepts.

We introduce a representation, prototypical concept descriptions, that represents informative redundant features. The basis of this representation is a prototype which gives the characteristics of an ideal example of the concept. This ideal example allows focusing on relevant features and eliminates attention on irrelevant features. In addition, redundant features are also described as part of the concept. For example, prototypes for the digits, 1 and 3 would show all of the relevant features, whether redundant or not, but would not include the feature color, since it is irrelevant. In addition, prototypical concept descriptions have the advantage of being more comprehensible than other methods representing redundant features. Although some research has focused on finding prototypical examples by using typicality measures (Zhang 1992), Monte Carlo sampling and random mutation hill climbing (Skalak 1994), our research centers on creating a prototype not necessarily any member of the training examples. We follow the standard psychological terminology where a prototype is an ideal example.

Although prototypes have not been used much in machine learning research, psychologists have researched the existence of prototypes extensively in animals (Aydin & Pearce 1994) and in humans (Huttenlocher et al. 1991; Medin et al. 1984). On the other hand, some researchers contend that other mental processes occur which give the illusion of prototypical models (Richardson & Bhavnani 1984). Despite the current conflict in psychology, we contend that for machine learning, prototypes provide some benefits for learning.

Learning prototypes to represent concepts has advantages over learning discriminating descriptions. Prototypical concept descriptions describe all relevant features in a class, whether they are redundant or overlap with other classes. We demonstrate that learning concepts with common and group redundancy makes the algorithm more robust in noisy and changing domains. Ensuring that an algorithm can learn concepts with group redundancy increases the stability of the theories learned. Our notion of group redundancy is similar to weakly relevant features described by John, Kohavi, & Pfleger (1994).

Second, although redundant information may not always be critical in the classification task, it may be useful in other tasks. Unfortunately, neither the psychological nor the machine learning literature provides a complete measure of what it means to have learned a concept. A partial measure of knowing a concept is being able to identify it. A more complete measure could include making inferences about the concept, using the concept for communication, and reasoning with the concept in a problem solving task. We plan to research the task of inferencing where the algorithm predicts an examples' features given its class. It is in this task that algorithms that can represent redundant features will have a distinct advantage over those learning discriminating descriptions. However, in this paper we only address the classification properties of prototypical descriptions.

Lastly, we believe that prototypes provide a closer approximation to human concept representation than do discriminating descriptions. Certainly if a child learns about zebras and ducks, they would know that both animals have two eyes, a nose, a mouth, and many other traits in common. With discriminating descriptions these features would be lost; however, prototypical descriptions would capture this characteristic information. Discriminating representations were solely intended for classification purposes, and therefore may not perform well in other tasks. Since prototypical descriptions can represent redundant features they can be better adapted to a variety of tasks that people perform.

Our goals for this paper are to first describe a method for learning prototypical concept descriptions of classes. To this end, we have developed an algorithm, PL (Prototype Learner), that learns characteristic prototypes of classes. Secondly, we will use PL to demonstrate the advantages of using characteristic prototypes over discriminating descriptions. We will focus on aspects concerning accuracy, stability, and the robustness of the algorithms. In section 2, we describe our algorithm to learn prototypical concept descriptions and we use the Exclusive Or domain as an illustration. Section 3 details the experiments using data from the UCI data repository (Murphy & Aha 1994). Lastly, Section 4 summarizes our contributions and discusses future work.

2 LEARNING PROTOTYPICAL DESCRIPTIONS

This section defines PL's method for creating characteristic prototypes. It then describes how prototypes are used for classification.

2.1 Terminology

Before we describe PL in detail, we will define the terminology we use throughout the paper. In a domain, each of the examples, ex, have been classified into one of the C classes, cl_c, where $c = 1, \ldots, C$. An example is defined by a finite number features, $F_1, \ldots, F_n$, where each feature F_i can take on a value f_{ij} where $f_{ij} \in domain(F_i)$. In this paper, we restrict our algorithm to learn using only nominal features.

A *prototype* consists of two elements: a class, cl_c, for the prototype and a conjunction of some features, $\{F_i\}$, with their values, $\{f_{ij}\}$, that an example should possess to be a member of the prototype class. Each feature value listed in a prototype also has an associated strength level measuring the amount of belief the system has in the feature value. The strength value ranges from .5 to .99 . Each prototype has only one class, however a class can have any number of prototypes associated with it. A *prototypical concept description* is the list of prototypes used to describe a class of examples. For example, suppose examples are described by 3 boolean features and the concept is the exclusive or of F_1 and F_2. A prototypical concept description of the concept would consist of the two prototypes {(Class: positive; Features ($F_1 = T$ .8) ($F_2 = F$ .9)),(Class:positive; Features: ($F_1 = F$ .75) ($F_2 = T$ .95))}.

2.2 The Learning Algorithm

Our learning algorithm, PL, learns prototypical concept descriptions from a set of examples for one class. Although the final output of the system is a prototypical concept description of the examples, internally PL uses a top down splitting mechanism to partition examples from one class and then learns prototypes for each partition. This method can partition examples from the same class which results in learning more than one prototype for a class. Therefore, PL learns disjunctions within classes by first partitioning classes and then learning one prototype for each partition. When classifying test examples, a voting method is applied to find the best matching prototype to the example. The class of this prototype is the predicted class of the example. The remainder of this section describes PL's learning method, while the next section describes the voting method used for classification.

Let P={S}, where S = $\{exs : exs \in cl_c\}$
Repeat
    Choose set S' in P.
    Choose partition of S' with highest quality
    If the quality of the partition > S'
    Then replace S' by partition.
Until partitioning P does not increase quality

Figure 1: Pseudocode for partitioning examples

PL learns each class in a domain separately. The general learning method is described in Figure 1. To learn a class, cl_c, PL partitions the examples by feature values. PL tries to find partitions of the examples in the class, where the randomness of the feature values in the example partition is significantly less than that of the original group of examples. Currently, PL only separates examples by splitting on a single feature to create partitions, similar to ID3's method. First PL must find the feature that creates the partition with the highest quality. Suppose an example set, S, is partitioned by the feature values in feature, F_i. We define the quality of $S_{f_{ij}}$ (the partition created by using only those examples in S where F_i is the value f_{ij}), as

$$quality(S_{f_{ij}}) = \sum_{available F_i} confmax(F_i)$$

where $availableF_i$ is $\{all\ F_i : confmax(f_i) > \eta\ and\ F_i\ unused\}$. We use the parameter setting η as the minimum feature value strength required for all features to be included in the prototype. The method we applied for finding η is described in the evaluation section. Intuitively, the quality of $S_{f_{ij}}$ measures how characteristic the attribute values are. The confmax of a feature, F_i, is defined by

$$confmax(F_i) = Max\{binsum(ex_{f_{ij}}, ex_{cl_c}, \pi) : f_{ij} \in d(F_i)\}$$

where $d(F_i)$ is the $domain(F_i)$, $ex_{f_{ij}}$ is the number of examples that have feature F_i set to value f_{ij}, ex_{cl_c} is the number of examples in class cl_c, and π is $1/J$ (the probability of getting a value f_{ij} for feature F_i, assuming all values have equal probability and where J is the number of possible values for F_i). $binsum(m, n, p)$ is the probability of getting m or more successes in n binomial trials with probability p for each success. The binomial distribution[1] is used to estimate whether partitioning the examples according to f_{ij} will reduce the randomness of the other features' values in the set of examples, $S_{f_{ij}}$.

[1] We use the normal distribution to estimate $binsum$ if $m * p > 5$.

PL then takes the weighted sum of the quality of each of the partitions, $S_{f_{i1}}, \ldots, S_{f_{iJ}}$ to find the $quality$ of splitting on F_i. PL weighs each of the partitions according to the percentage of examples that they contain. The weighted sum is applied to test the quality of splitting on all of the features, $F_1, \ldots, F_n$, and to choose the feature with the highest quality. If the quality of possibly splitting on a feature is greater than the quality of the original set, S, then the set S is partitioned, otherwise the partitioning process stops.

The motivation for the partitioning process is to create several good partitions of the examples. Once this process finishes, PL learns a prototype for each of the partitioned sets of examples. It generalizes the partitioned examples into an ideal example and stores it in the prototype. PL sets the class to cl_c and applies $binsum(ex_{f_{ij}}, cl_c, \pi)$ to calculate the strength for each feature value for each of the prototypes. The strength value corresponds to the randomness of the feature value distribution for the partitioned examples. High randomness corresponds to low strength. If a feature has low strength, it is not considered important for the prototype. The set of prototypes learned for class cl_c is a prototypical concept description for cl_c. These prototypes can be used for classification or inferencing.

2.3 Classification

Once the prototypes are learned, they are treated as if they were actual examples. When classifying, PL applies a distance metric to find the prototype which is the closest to the test example. The class that prototype represents is then predicted as the class of the example.

Like other instance-based methods, PL computes the distance between a test example and a prototype by summing the distance between each of the feature values. PL uses a modification of MVDM (Cost & Salzberg 1993) to compute the difference between two feature values. To use this metric, a table is created based on the training data that defines the difference δ between two feature values, $V1$ and $V2$. The original MVDM computed the tables based on the difference between the frequency of $V1$ and $V2$ in each of the classes. We modified this metric to take advantage of the partitions created when searching for prototypes. The examples in the partitions are used to calculate the difference table. More specifically, the distance δ between $V1$ and $V2$ is defined by

$$\delta(V1, V2) = \sum_{i=1}^{P} \left| \frac{C_{1_i}}{C1} - \frac{C_{2_i}}{C2} \right|^k$$

where P is the total number of prototypes learned, C_{1_i} is the number of times $V1$ occurred in the example partition used to create prototype i and C_1 is the total number of times $V1$ occurred in the training set, and k is set to 1. PL creates this table once after the

Table 1: Exclusive Or: feature value counts

Values	**Features**				
	F_1	F_2	F_3	F_4	F_5
0	8	8	8	8	8
1	8	8	8	8	8

prototypes are stored. To classify a test example, PL computes the distance between the test example and each of the prototypes by comparing their feature values. PL sums the distances between the test example and each prototype, and predicts the class of the prototype with the smallest distance to the test example. Our modification to the MVDM metric and use of prototypes allows PL to learn in the Exclusive or domain when other instance-based methods perform poorly. Although PL calculates strengths for each prototypical feature, they are not used for classification, but they could be used for inferencing.

2.4 An Illustration

We will use the Exclusive Or domain to illustrate the PL algorithm. The Exclusive Or domain consists of n binary features where only two of the features F_i and F_k $(i \neq k)$ determine the class of the example. Examples where $F_i \neq F_k$ are positive examples, and all others are negative examples.

Suppose $n = 5$ where F_1 and F_2 are the two relevant attributes. There are 32 examples, 16 positive and 16 negative, if all possible examples are used for training. Suppose that PL learns the set of positive examples, S, first. Table 1 gives the feature names and the number of examples that have each of the possible values before any partitioning of the examples occur.

PL will test the quality[2] of each of the features F_1, F_2, F_3, F_4, F_5. Table 2 lists the counts for the feature values if F_1 is used to partition the examples. To find the characteristic quality of each of the partitions of F_1, we will sum the $confmax$ of the other features after splitting on f_{10}. Thus the $quality(S_{f_{10}}) = confmax(F_2) + confmax(F_3) + confmax(F_4) = .99 + 0 + 0$. The $confmax(F_2) = Maximum(binsum(0, 8, .5), binsum(8, 8, .5)) = .99$, since there are 0 examples with f_{20} and 8 examples with f_{21} and the probability of getting either a 0 or 1 value is .5. The quality of S_{f_11} is also equal to .99. Thus the quality of using F_1 as a splitter is .99 since each partition has the same number of elements. The quality of F_2 as a splitter is the same as F_1.

Next we need to find the quality for the remaining features F_3, F_4, and F_5. Table 3 lists the counts for the feature values if F_3 is used to partition the examples. The $characteristic\ quality(S_{f_{30}}) = confmax(F_1) + confmax(F_2) + confmax(F_4) = 0$. The quality of $S_{f_{31}}$, $S_{f_{40}}$, $S_{f_{41}}$, $S_{f_{50}}$ and $S_{f_{51}}$ are also 0. Thus F_1 and F_2 have the highest quality thus far; however, PL will need to see if their quality is greater than that of the entire set before partitioning. The $characteristic\ quality(S) = 0$ by looking at Table 1. Thus PL will choose either F_1 or F_2 as the feature used to partition the set of examples.

[2]We will set η to .5 for this illustration

Table 2: Feature value counts for examples partitioned by F_1

Set name	**Value** of F_1	F_2		F_3		F_4		F_5	
		0	**1**	**0**	**1**	**0**	**1**	**0**	**1**
$S_{f_{10}}$	0	0	8	4	4	4	4	4	4
$S_{f_{11}}$	1	8	0	4	4	4	4	4	4

Table 3: Feature value counts for examples partitioned by F_3

Set name	**Value** of F_3	F_1		F_2		F_4		F_5	
		0	**1**	**0**	**1**	**0**	**1**	**0**	**1**
$S_{f_{30}}$	0	4	4	4	4	4	4	4	4
$S_{f_{31}}$	1	4	4	4	4	4	4	4	4

Suppose F_1 is randomly chosen to create the first partition. The examples will be separated into two sets, $S_{f_{10}}$ and $S_{f_{11}}$. We will now focus on $S_{f_{10}}$ and see if it should be partitioned again. Since PL has already used F_1 to partition, it cannot be applied again. The quality of each of the remaining features F_2, F_3, F_4 is 0 and the quality of $S_{F_{10}}$ is .999. Thus the partitioning stops after creating the two sets $S_{f_{10}}$ and $S_{f_{11}}$. Note, that regardless of which feature (F_1 or F_2) PL chooses to create the partition, the resulting prototypes are the same.

Next PL learns prototypes for the two partitions, $S_{f_{10}}$ and S_{f11}. Both of the partitions will have high feature value strengths for the relevant features, F_1 and F_2, although the actual feature values for each feature in each of the partitions will be different. Thus the prototype for $S_{f_{10}}$ will look like {Class: positive; Features: $(F_1 = f_{10}\ .99)$ $(F_2 = f_{21}\ .99)$ }, and the prototype for $S_{f_{11}}$ is similar. A similar process will occur for learning prototypes for the negative examples. PL will also learn two prototypes for the negative examples.

3 EVALUATION

To evaluate the usefulness of prototypical concept descriptions, we will test PL on classification accuracy and compare it against ID3 and C4.5 (Quinlan 1993). To better understand the circumstances under which PL performs well, we describe experiments on two very different artificial domains. In addition, experiments

on natural domains are described to show empirically that prototypical concept descriptions can be applied in practical situations.

3.1 Experimental Evidence on Artificial Domains

To predict the domains that an algorithm works well in, the domains and the algorithms need to be carefully characterized. With real world data, little is known about the domain characteristics. Therefore to understand the characteristics of the algorithm, artificial domains should be used since their characteristics can be carefully controlled. To this end, we have created two different types of artificial domains: one domain where class membership is defined by necessary and sufficient conditions, and the other domain where membership is defined by the number of attributes present (k of n attributes must be present). We contend that as we add redundancy to both of these domains, the accuracy of PL should surpass that of discriminating learning methods.

3.1.1 Experiment with Domain Defined by Necessary and Sufficient Conditions

The first domain is defined by conditions that need to be satisfied for class membership. The conditions were represented using randomly created conjunctions and disjunctions. Examples that satisfy the conditions are considered a member of the class. More specifically, we used 8 binary attributes named A ...H, and 2 classes. To create the conditions for the two classes, we created all possible patterns assuming two relevant attributes (eg. 11, 10, 01, 00). These were the disjunctions in the classes. Then we randomly picked attributes to be associated with each bit in each pattern. An attribute could not be used more than once for each pattern. Lastly, we randomly choose classes for each of the disjunctions (eg. Class2 $\leftarrow$ (C=1 & E=1), Class1 $\leftarrow$ (B=1 & F=0)).

To test our hypothesis that redundancy aids classification accuracy, we incrementally added redundancy to the domain. Initially we ran PL, ID3, and C4.5 on the domain with 20% attribute noise and 256 examples. To increase redundancy in the domain, we added two extra attributes to the domain such that they took on the values used to define the classes (eg. Class1 $\leftarrow$ (B=1 & F=0 & I=1 & J=0)). Note that adding these two attributes to the domain does not increase the number of examples in the domain, since they can only have fixed values. We continued this technique of increasing the redundancy in the domain and tested the three algorithms on accuracy.

The results of 30 runs of this experiment are shown in Figure 2. We used 66% of the data for training and the remaining data as a test set. As expected, adding the redundancy to the domain increases the accuracy of all the learning algorithms. However, PL's accuracy does not surpass C4.5 or ID3's accuracy. PL's accuracy is only statistically different with 90% confidence than C4.5 when there is no redundancy, and not statistically different than ID3 or C4.5 at the other levels of redundancy. C4.5 and ID3 are able to take advantage of the redundant information in the domain mainly because the noise forces the discriminating algorithms to learn the redundant attributes. If this domain was not noisy, the redundant attributes would not be learned simply because they do not convey any additional information, however if the test examples are noisy, discriminating algorithms will classify less accurately than those learning redundant features. Section 3.3 describes an experiment where we trained the algorithms on ideal data and tested with noisy data.

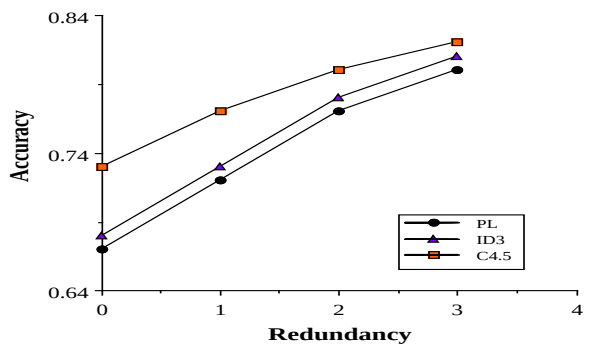

Figure 2: Accuracy vs. Redundancy in conditional domain.

3.1.2 Experimental Evidence on k of n Domain

Unlike the first domain, in the second artificial domain, a predefined number of attributes need to be present for class membership. We create the class using 8 binary attributes and 2 classes. All 256 examples were created, but only those that belonged to either of the classes were kept. The classes were defined such that any examples that had 3 or more features *on* were in Class1 and any examples that had 3 or more features *off* were in Class2. Examples that satisfied both of these classes were placed in Class1.

To see how the amount of redundancy affects the accuracy of the algorithms, we increased the number of attributes that needed to be *on* or *off* from 3 to 6, but also checked intermediate points. Figure 3 shows the results of this experiment after 30 runs using 66% of the data for training and 33% for testing. PL takes better advantage of the redundancy in this domain than both ID3 and C4.5. PL's accuracy steadily in-

creases as the redundancy in the domain increases. ID3 and C4.5's accuracy fairly reliable at point 3, but at points 4 and 5 neither the conditions for the classes nor their negation is easy for the discriminating algorithms to learn. At point 6, the discrimination algorithms can use negation to learn again. This type of domain is more easily represented by prototypes than necessary and sufficient conditions on particular attributes.

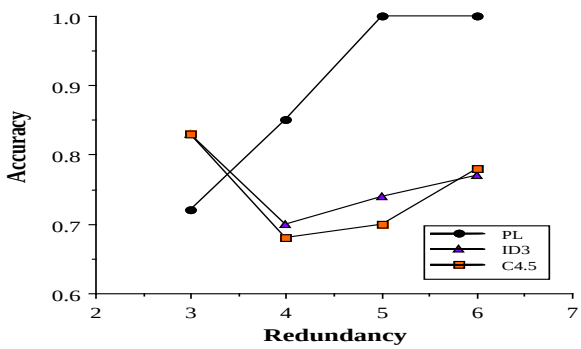

Figure 3: Accuracy vs. Redundancy in k of n domain.

3.2 Experimental Evidence on Accuracy and Stability

The usefulness of prototypical concept descriptions will depend on how accurately they classify and the stability of the theories they generate. In some domains, prototypical concept descriptions may classify as accurately as ID3 or C4.5 while in other domains they may classify poorly; however, the stability of the theories generated should consistently exceed that of ID3. In our second experiment, we test the robustness of prototypical concept descriptions when learning with pure training data and noisy test data. We first define the metrics used for the experiments, then describe the methodology we applied, and conclude with a discussion of the results.

3.2.1 Metrics

The metrics we use to evaluate prototypical concept descriptions are accuracy and stability. The accuracy of an algorithm shows the percentage of unseen test examples the algorithm classifies correctly, that is, the standard generalization accuracy. The stability of an algorithm measures how much the theory changes when given different training sets. We report on two types of stability: semantic and syntactic. An algorithm's ability to learn concepts with group redundancy reduces the variance of the concept descriptions, making them more stable. Learning common redundancy increases an algorithm's robustness in noisy situations.

Semantic stability measures how much the induction algorithm's classification varies with different training samples. We estimate the semantic stability by measuring the percentage of generated theories that consistently choose the same class for examples in a separate test set. The method we used to compute the semantic stability first separates a fraction of the data from all of the data available. In our experiments we used 10% of the available data as the semantic test set. We used the remaining data for 5 fold cross validation tests. We then used each of the 5 theories generated from each fold to classify each of the examples in the semantic test set. The semantic consistency reports the percentage of semantic test examples all of the theories agreed upon.

Syntactic stability measures the amount of change among the theories generated by different training sets. The syntactic stability is measured by noting the average number of modifications (edit distance) needed to change one theory into another (Wogulis & Pazzani 1993). The syntactic stability is computed by using the theories generated from each fold of the cross validation experiment. We compare each of the theories to each other and report the average number of modifications necessary to change one theory syntacticly into another. A modification is the addition or deletion of a feature value or the change in value of a feature. We use a simple hill climbing search to estimate the least number of modifications that need to be made to a theory to transform it into another.

3.2.2 Methodology

In both experiments, we report the accuracy and stability of the algorithms. The data for the domains is randomly separated into three sets, the training set, the test set, and the semantic test set, as mentioned earlier. The semantic set (10% of the all available data) is used to test the semantic stability of the theories generated. We use the training and test data for 5 fold cross validation experiments. The training data is obtained from 4 of the folds and the remaining fold is used for testing. The testing fold is changed each time to generate 5 different test sets.

When we run PL, the training data is again separated into a subtraining set and a subtest set for finding a good strength level. This method is not used when running ID3 or C4.5. The training data is broken down again into 5 folds and 4 folds are used for actual training data and the last fold is used as a strength level test set to find a value for η. In an attempt to find a reasonable strength level setting, we iterate through different settings using 0.7, 0.8, 0.9, and 0.99. We test each of the theories generated from the different strength level settings and choose the setting

that produces the most accurate theory. We set the strength level and learn a theory with the entire training set. We then use the test set to obtain accuracy and stability results.

3.2.3 Results

To demonstrate our claim, we tested PL on six domains and compared it to ID3 and C4.5 using 30 5-fold cross validation runs. These domains were chosen because they provide a mixture of artificial and natural data and they only contain nominal features. The domains and their characteristics are listed in Table 4. All of these databases are located in the UCI Machine Learning Data repository (Murphy & Aha 1994).

The averaged accuracy results are listed in Table 4. An asterisk beside the an algorithm's accuracy indicates that the algorithm and PL are significantly different to a .98% confidence level using a two-sided t-test. In these domains, PL tends to perform better than or comparable to ID3 and C4.5 with respect to accuracy. PL appears to work well in natural domains, but performs poorly on domains where concepts contain functional definitions. In domains with noisy data such as LED, PL's ability to learn features with common redundancy makes it more robust than ID3 and C4.5.

We also report the semantic and syntactic performance of each of the algorithms for the same 30 runs. Judging from the stability results in Table 5, prototypical concept descriptions are less likely to change given different training sets. The semantic stability states the average percentage of examples that the theories from different folds agreed upon. A higher percent of semantic stability shows that the meanings of the different theories do not change as easily given different training sets. The syntactic stability shows the minimum number of modifications that were needed on average to transform one theory into another. The number of syntactic modifications for prototypic concept descriptions is much less than that needed for ID3. When ID3 cannot find a feature with information gain it chooses randomly. PL does not separate the examples if a good partitioning of the examples cannot be found, resulting in theories that are structurally similar. In addition, PL learns classes with group redundancy which stabilizes the learned concept descriptions.

Lastly, we wanted to mention the settings used for the feature value strength parameter, η. As described in the methodology section, the training set is separated into 5 folds, 4 of which are used for subtraining and the last is used as a subtest set for determining the feature value strength level for the run. Table 4 also shows the strength levels and the standard deviations for each domain. Although PL tested several different strength levels, the table shows that the actual values used within each domain remained steady. We originally intended to find only one strength level for each domain, but for methodological reasons concerning the amount of data available, we decided to find a strength level for each run. The small standard deviations for each domain show that only one strength level is really necessary for a domain.

3.3 Experimental Evidence on Robustness

In the previous section we showed that PL learns stable theories with noisy training data. In this section we show that when given perfect training data and testing on noisy data, PL performs more accurately than ID3. This is a non-standard form of evaluation as the training set is not chosen from the same population as the testing set. However, such situations actually occur rather naturally. For example, we can easily create perfect training data for typed character recognition and would want the classifier to work on noisy data without creating or modeling all of the different types of noise that might occur. In addition, situations exist where the theory of the domain is known, but classification time features could be missing or incorrect. For example, although plant classification can be done using a decision tree, some of the plant data could be missing due to varying seasons or the age of the plant.

For these occlusion experiments, we used two different forms of noise. In the first experiment, some of the values in the test examples were changed whereas in the second experiment, some of the values in the test examples were missing. In each experiment, the noisy test data was generated with a 10% chance that a particular example value would be either modified or eliminated. These different types of noise were not mixed together, but tested separately. Each experiment also had a training size of 200 examples and a test size of 1000 examples. We used the LED domain without irrelevant features to examine our hypothesis and ran PL with the average strength level found for the LED domain in the previous section (Table 4). If a test example has a missing value, PL does not use that attribute to compute its distance metric for the test example. ID3 and C4.5 use the most common value of the class to fill in missing data values for examples.

Table 7 shows the results of this experiment using 30 runs. PL performs more accurately in both situations in which noise is introduced into the test data only. PL's prototypic representation and classification metric aid classification in changing environments. ID3, on the other hand, uses a brittle tree representation that cannot easily adapt to different environments. Certainly ID3 could perform better in this domain by changing the perfect data into noisy data that models the noise in the test domain. However this is a problem since, first, modeling the noisy test data may be difficult and time consuming, and second, a different tree would need to be created for every different noisy

Table 4: Domain Characteristics

Domain	Examples	Features	Classes	Comments
Exclusive Or	64	6	2	2 irrel. feat.
LED	200	7	10	10% class noise
LED w/irr. feat.	500	24	10	10% class noise, 17 irrel. feat.
Lymphography	148	18	4	
Promoters	106	57	2	
Soybean-large	290	15	35	first 15 classes
Zoo	84	16	4	classes with 10 or more examples

Table 5: Accuracy

Domain	PL			ID3	C4.5
	Accuracy	η (Strength)	η Sd.	Accuracy	Accuracy
Exclusive Or	97.09%	0.93	0.06	85.11%*	98.81%
LED	67.70%	0.94	0.08	58.26%*	59.51%*
LED w/irrel. features	67.46%	0.98	0.05	55.45%*	67.43%
Lymphography	81.03%	0.91	0.09	74.69%*	78.15%*
Promoters	84.67%*	0.60	0.13	74.77% (76.04%)	
Soybean-large	73.42%	0.89	0.10	83.37%*	85.52%*
Zoo	99.56%	0.94	0.08	94.62*%	98.64%*

environment.

4 DISCUSSION AND FUTURE DIRECTIONS

This paper explores the application of prototypical concept descriptions for classification. We have described an alternative representation that can learn and represent redundant information within classes. We have implemented an algorithm to this end and experimented with it in several domains. In addition to showing that prototypical concept descriptions can in some domains classify more accurately than ID3, we have shown that they are more stable with respect to the structure and content of the theories generated. The robustness experiments illustrate that prototypical descriptions could be more accurate in noisy and changing environments when an ideal classification is known. Lastly, we have introduced the idea of common redundancy and group redundancy, and shown how they affect classification performance.

Although PL currently works on discrete data, we intend to extend the approach to include real-valued features. The partitioning algorithm will be modified by replacing the binomial calculations by the another calculation such as the normal, t-test, or F-ratio, and using class means and standard deviations. The prototype structures will be extended for real-valued data by replacing single feature values with a range of values, possibly represented by the mean and standard deviation of the real-valued feature. Although these extensions affect both the algorithm and representation, the main structure of the prototypes and PL will remain consistent with the descriptions in this paper.

We also intend to explore the task of inferencing with prototypical concept descriptions. The task of inferencing is to learn a theory given training examples and then predict a test example's features and values, given the example's class. The ability to predict features of a class gives a learning agent information about a particular example that can be used for problem solving, reasoning, communication, knowledge discovery, or other tasks. To add an inferencing ability to PL the partitioning algorithm and prototype representations will remain the same, but a new matching metric will replace the matching metric used for classification. This metric will need to match against classes instead of feature values. Since prototypical concept descriptions can represent redundant feature information, it will be better suited for this task than traditional discriminating learning methods.

References

Aha, D., Kibler, D. and Albert M. (1991). Instance-based learning algorithms. *Machine learning*, volume 6, pp 37-66. Boston, MA: Kluwer Publishers.

Aydin, A. and Pearce, J. (1994). Prototype effects in categorization by pigeons. *Journal of Experimental Psychology: Animal Behavior Processes*, volume 20 (n3) pp.264-277.

Clark, P. and Boswell, R. (1991). Rule Induction with CN2: some recent improvements. *European Working Session on Learning.*

Table 6: Stability

Domain	PL		ID3	
	Semantic	**Syntactic**	**Semantic**	**Syntactic**
Exclusive Or	85.71%	1.03	39.52%*	22.73*
LED	90.17%	1.12	63.83%*	9.80*
LED w/irrel. features	62.76%	1.9	32.21%*	56.76*
Lymphography	79.33%	4.13	59.56%*	34.01*
Promoters	73.64%*	31.00	38.48%	37.85
Soybean-large	69.2%	4.96	72.41%	11.68*
Zoo	98.52%	1.39	87.41%*	2.20*

Table 7: Accuracy with Perfect Training and Noisy Test Data

Domain	PL		ID3		C4.5	
	Accuracy	**Sd.**	**Accuracy**	**Sd.**	**Accuracy**	**Sd.**
LED with incorrect values	73.45%	0.01	69.15%*	0.01	69.67%*	0.01
LED with missing values	91.69%	0.00	85.60%*	0.01	85.87%*	0.01

Cost, S. and Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, volume 10, pp. 57-78. Boston, MA: Kluwer Publishers.

Duda, R., and Hart P. (1973). *Pattern classification and scene analysis.* New York: John Wiley & Sons.

Huttenlocher, J., Hedges, L. and Duncan, S. (1991). Categories and particulars: prototype effects in estimating spatial location. *Psychological Review*, volume 98 (n3) pp.352-376.

John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning*, New Brunswick, NJ. Morgan Kaufmann.

Medin, D., Altom, M. and Murphy, T. (1984). Given versus induced category representations: Use of prototype and exemplar information in classification. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, volume 10 (n3) pp. 333-352.

Murphy, P. and Aha, D. (1994). UCI repository of machine learning databases [machine readable data repository]. Tech. Rep., University of California, Irvine.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, volume 1. Boston, MA: Kluwer Publishers.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA.

Richardson, K. and Bhavnani, K. (1984). How a concept is formed: Prototype or contingency abstraction? *British Journal of Psychology*, volume 75 (n4) pp. 507-519.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, volume 14 pp. 465-471.

Rosenblatt, F. (1958). The perception: a probabilistic model for information storage and organization in the brain. *Psychological Review*, volume 65 (n6) pp.386-408.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Leaning internal representations by error propagation. *Parallel Distributed Processing - Explorations in the Microstructure of Cognition* chapter 8 pp. 318-362. MIT Press.

Skalak, D. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning*, New Brunswick, NJ. Morgan Kaufmann.

Wogulis, J. and Pazzani, M. (1993). A Methodology for Evaluating Theory Revision Systems: Results with Audrey II. *Proceeding of International Joint Conference for Artificial Intelligence*, pp. 1128-1134.

Zhang, J. (1992). Selecting typical instances in instance-based learning. In em Proceedings of the Ninth International Conference on Machine Learning, New Brunswick, NJ. Morgan Kaufmann.