Mining for Features to Improve Classification

Sarah Zelikovitz
College of Staten Island of CUNY
2800 Victory Blvd
Staten Island, NY 10314

Abstract *When first faced with a learning task, it is often not clear what a satisfactory representation of the training data should be, and we are often forced to create some set of features that appear plausible, without any strong confidence that they will yield superior learning. Moreover, we often do not have any prior knowledge of what learning method is best to apply, and thus often try multiple methods in an attempt to find the one that performs best. This paper describes a method called Feature-mine, that takes a set of features and augments them with macro-features that test for the occurrence of combinations of values on the original features. Our approach uses associations that are mined from the data to create these new features. Importantly, the method is independent of any learning method, with the creation and selection of new features based simply on the relationship between the attributes and the class labels in the data.*

Keywords: constructive induction, data mining, classification

1 Introduction

Inductive learning problems are often formulated as a set of examples for which the class is known, where each example is a vector of feature-values and the class that it belongs to [1]. Different supervisory learning algorithms use the example set of vectors to create a concept that represents the known set of examples, as well as unknown ones in different ways. The bias of the learning algorithm, in part determines how closely it can approximate the true target concept from the set of examples. Distinct learning algorithms, therefore, are often useful for different classes of target concepts.

Often the set of original features that is given with the training examples is not the ideal one [2, 3]. In this work we are concerned with the *relationships* between features in conjunction with their relationships to the target concept. We exploit the relationships among features, or the partial redundancies between features to create new features. These features are closer to the target concept than the original set of feature. Therefore, with the addition of these features, learners can overcome initial biases and hypotheses are closer to the true target concept; hence error rates on unseen examples improve. This is a data driven method of constructive induction [4].

2 Feature-mining Algorithm

2.1 Notation

Our approach assumes a two-class classification problem, with boolean feature values. Each training and test instance is denoted by an n dimensional vector of 0s and 1s, where n is the number of features used to describe each example. Each position i in the vector gives the value of the feature number i for that instance. We will represent features by upper case letters (A,B,etc.) As is the case for many machine-learning algorithms, Feature-mine trains on only a portion of the training data, leaving the rest for a pruning set. Following the example of previous learners, only two thirds of the training set is used for deter-

mining relationships and the creation of new features. For the rest of this discussion we will refer to this two-thirds as the training set, and the remaining one-third as the pruning set.

Feature-mine determines the relationships among different features for the positive class, as well as the negative class. This is done by dividing the training examples into two groups, one containing all positive instances and one containing all negative instances. Relationships between features are then computed separately for each of these two groups of data.

2.2 Association-finding

Feature mine determines the relationships between features by borrowing the concept of association rules from data-mining [5, 6]. For every pair of features, say A and B , four association rules are formed:
$A = 0 \Longrightarrow B = 0$,
$A = 1 \Longrightarrow B = 0$,
$B = 0 \Longrightarrow A = 0$,
$B = 0 \Longrightarrow A = 1$.

These association rules have both a confidence factor and support associated with them, where the support refers to the number of times that the right-hand side of the association is true, and the confidence factor refers to the percent of time that the entire association is true (given that the right-hand side is true). If we set the support to be s, and the confidence factor to be c then the first rule in the sequence above can be interpreted as follows: In the positive set of examples (alternatively in the negative set, as these are done separately) A has the value of 0 s times, and out of these times c percent of them have B with a value of 0. Alternatively, we can just say simply that A = 0 *implies* that B = 0, in the positive corpus with support s and confidence c.

This data-mining is done on both the positive set of examples and on the negative set of examples. The set of associations that is mined on the positive set can be looked at in two ways. One is that it is a way of representing the target concept, or piece of the target concept. As a simple, extreme example if the target concept is XOR of A and B, we would have $A = 0 \Longrightarrow B = 1$, with a confidence factor of 100%. The second way that these association rules can be interpreted is by representing the level of redundancy between the two features. A simple example of this would be: if A = 1 *always* implies that $B = 1$, then feature B is redundant once feature A is known to be 1.

Feature-mine then creates a set S+ that contains all the associations that are true for the positives, but are not true for the negative set. Since all associations are actually "true" for both sets, but at different levels of support and confidence, the levels of support and confidence are used to determine which associations to add to this set.

Feature-mine takes the two distinct sets of association rules from the positive set and negative set, and extracts those associations that are statistically more significant in the positive set than in the negative set. We use a chi-squared test of independence to determine the probability that each combination of two features comes from the same distribution. If this probability is lower than a threshold t, then we can say with some certainly that the association rule more accurately represents the positive corpus than the negative corpus. This association is then added to the set $S+$.

However, the discussion above fails to take an important point into account [7]. Suppose that the association $A = 0 \Longrightarrow B = 0$ is added to the set S+. This means that with high probability this association captures a property of the positive set, but does not capture a property of the negative set. However, this might be true only because of the right-hand side of the associations. For example, if B is always 0 in the positive corpus (or never 0 in the negative corpus) then the association above would capture a property of the positive set. This is not because of the redundancy relationship between A and B, or because of the relationship of both A and B to the target concept, but rather only because of the relationship of B to one of the classes. Before adding an association to the set S+ therefore, two more one-sided statistical tests must be performed. In the above

example, this would correspond to comparing the associations $A = 0 \Longrightarrow B = 0$ with $A = 1 \Longrightarrow B = 0$ in the positive set of associations. This would have to be checked in the negative set as well. Only if all three of these tests are passed, we can safely say that:

1. The association represents the positive set, but not the negative set.

2. The combinations of features is what causes the association to represent the positive set, not the value of the left hand side features in the positive corpus.

3. The combination of features is what causes the association to represent the positive set, not the value of the left hand side feature in the negative corpus.

If this is the case, then the association is added to S+.

2.3 Feature creation

Combinations of the original boolean features are then formed using this set S+ of association rules. Both conjunctions and disjunctions of features are created by combining associations that are proof of the utility of the combination of a specific set of features. a disjunction is added if both associations that imply this disjunction are in the set S+. For example: if
$A = 1 \Longrightarrow B = 0$,
and
$B = 1 \Longrightarrow A = 0$
are both in S+, both implications can be interpreted as $A = 0$ *or* $B = 0$, and therefore this feature would be added to the original set of features. It is important that both of these implications be present in S+ in order for the disjunctive feature to be added to the original set. As in all feature creation programs we must be concerned with adding too many features. Too many features can degrade the learners [3], so we want to be certain that the features we add are the correct ones and accurately describe the positive data. A way of looking at this set of associations is by thinking of them as experts describing the positive class, and a new feature is created only if there are two experts that suggest that it should be added.

Conjunctive features are formed in the same way as disjunctive ones. If both
$A = 1 \Longrightarrow B = 1$
and
$B = 1 \Longrightarrow A = 1$
we would add $A = 1$ *and* $B = 1$. Each one of these associations individually tells us that the combination $A = 1$, $B = 1$ is more prevalent in the positive set than in the negative, when both of them are in S+, the feature is actually added.

The original dataset is then re-described using the original set of features as well as this new set of features. The process is then iterated. Associations are formed between the set of new features and all other features, and more new features are created. This process continues until no new features are created, or a limit on the number of iterations is reached.

3 Empirical results

We wished to test the utility of the new features that were created using the feature-mining algorithm. Since this feature creation process did not tailor itself to any specific learner, it would be best if all learners were improved by the addition of these new features. If the new features do indeed capture properties of the positive data (which is actually a clue to the target concept) many different learning algorithms should be improved. This does indeed turn out to be the case.

3.1 Evaluation Methodology

To evaluate Feature-mine we use three diverse learning methods that have been often used for classification. C4.5 [8] is a decision tree learner that uses a greedy divide-and-conquer approach. Ripper [9] is a rule induction system that heuristically builds a set of rules to pre-

dict class. A nearest-neighbor learner, PEBLS [10], classifies an example by finding which examples in the training set are close to it, learning weights on the features to reflect their relative importance in the classification task. C4.5 and Ripper themselves bottom out at an information-theoretic metric for selecting features, whereas PEBLS uses features "holistically", allowing all to weigh in when classifying an example.

3.2 Pruning Set

As mentioned in the introduction, different heuristics and/or biases are used in different machine learning algorithms. Hence, on the original feature set the three different algorithms that we used often chose features sets that were distinct from each other, to be included in the final target concept. This has ramifications in our constructive induction process. Since our feature creation is an iterative process, it is sometimes unclear when to end the iterations. We placed an external limit on the number of iterations; using empirical tests to determine when to stop the iterative process, we have set the limit to be ten iterations. Most often, in the domains that we have studied the process ends before the tenth iteration is reached. This is because no new features are created, and the process of feature creation is halted.

However, it is unclear which set of features to choose for the final evaluation of the test set. It would seem at first glance that the final set of features from the last iteration should be chosen; but this is not always the best choice. Even if it *is* the best choice for one learner, it is often not the best for another. We use the pruning set to determine which set of features to use in redescribing the test set of the unseen examples. One third of the original training set is not used for training, but at each iteration is redescribed with the new set of features. For each loop l through the iterative process, we then have a set of examples with known classes that represents the algorithm halting at iteration l. Let us call this set of pruning examples $P(l)$. The error rate at iteration l can then be estimated for any of the three learners described above, by running the learner with the set of examples $P(l)$. If this is done for l varying from 0 to the final number of iterations, and the l that provides the minimum value is chosen, then we are using the pruning set to determine which iteration to use to describe the test set of examples.

The pruning set is used separately with each learner; it turns out that often a different l is chosen for the different learners. This does not contradict our original assertion about our algorithm that it is learner independent. The actual creation of the new set of features is independent of any learning algorithm; the learning algorithm is only used in the analysis of where to stop the feature creation process. Hopefully, this combines two advantages in the following sense. Our features are representative of the data itself, as opposed to massaging the data into useful form for a specific algorithm. At the same time, the algorithm that will eventually be used for testing can provide input to determine how much feature creation that algorithm finds necessary for specific problem.

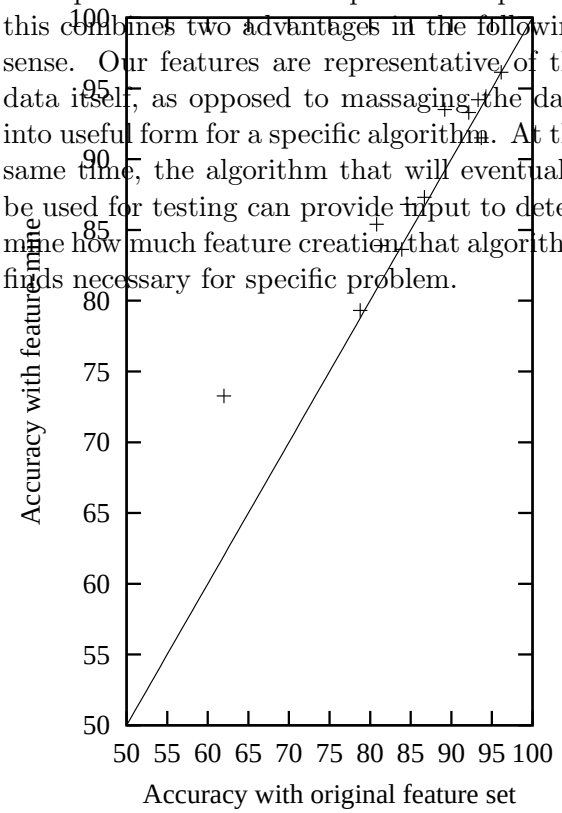

Figure 1: C4.5 vs C4.5 with Feature-mine

3.3 Data Sets

To explore the performance of Feature-mine with these three algorithms we use 16 datasets

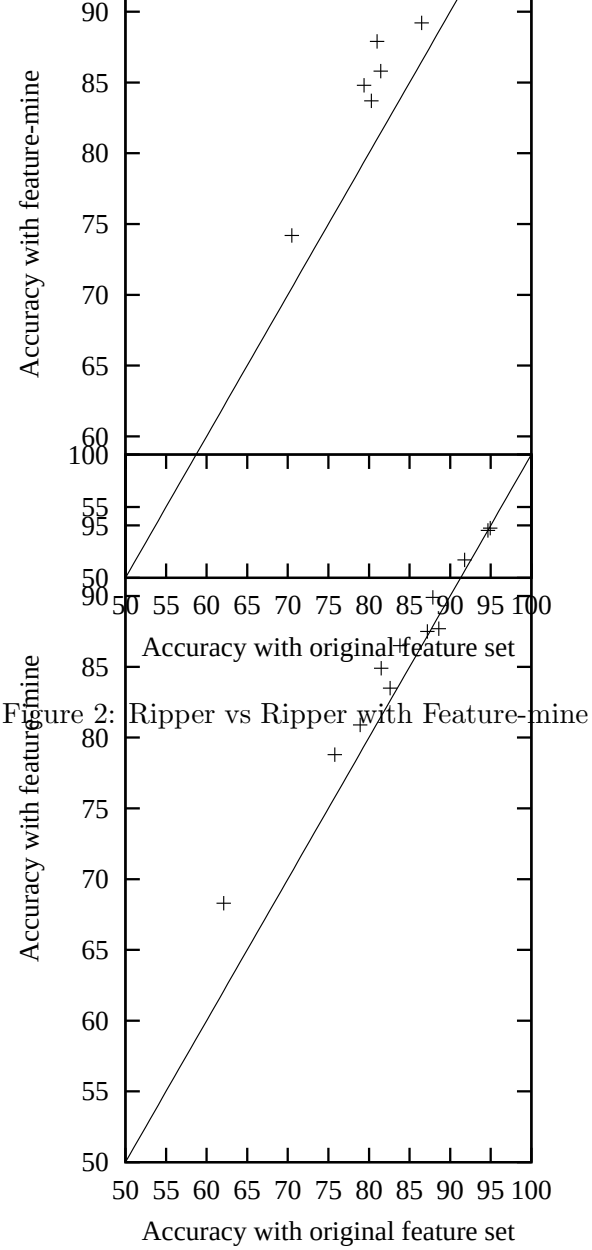

Figure 2: Ripper vs Ripper with Feature-mine

Figure 3: PEBLS vs PEBLS with Feature-mine

arising from three domains that were specifically selected because of their use in previous constructive-induction research [11, 12].

3.3.1 DNF Functions

This domain consisted of four functions taken directly from [11]. The datasets are generated randomly, and vary with regard to the number of features in the data set, number of features in the target concept, and monotonicity. However, we diverge from Pagallo and Haussler in the use of a training set that is one third the size that they considered. The reason for this is that Pagallo and Haussler were evaluating the ability of their learning method to eventually converge to the correct target concept. Here, instead, we want to have more limited amounts of data, to leave room for a range of result quality for learning — indeed, Ripper was able to learning the target functions with 100% accuracy on datasets of that larger size.

3.3.2 English text

Our second domain concerns problems in text categorization, where features correspond to the presence or absence of words in a document. Since such a representation ignores co-occurrences of words, it is often thought to be a domain where combinations of words, if correctly identified, should aid in the learning task. This domain had two classes of problems. The first is Email topic spotting, and the second is newsgroup topic spotting, using datasets taken from [12]. There are four different sets of email data, from four different users, where the target class is to recognize that a message is a talk announcement. There are four different newsgroup datasets. All correspond to the same set of data, selected from four newsgroups. In each case the postings to one newsgroup is considered positive and the others are collectively labeled negative. Data are generated by keeping all words in the header sections of each example (e.g., To: and From: in email), but only the one hundred most common words in the body of the text of the email messages or news postings were kept.

3.3.3 Biological Sequences

Our final domain concerns tasks of DNA-sequence classification [12]. Of the four datasets used, one is a collection of data labeled by whether they are promoters. A second consists of data labeled as to whether it is a fragment of human or phage DNA. The final two arise from DNA structure classification. In one case the task labels DNA fragments as positive if they are in the A-DNA structural class, and negative if they are in the B-DNA or Z-DNA class. The second case labels B-DNA as

positive and the rest as negative. (We don't consider the third case of Z-DNA as positive due to the severe sparsity of such data.) Since Feature-mine deals with Boolean data, each feature corresponds to the presence or absence of some particular n-gram, for all possible n-grams of size 1, 2, or 3.

4 Results

Our results for all datasets are averages of 10 runs. For the DNF datasets, ten different random sets of training and test data were generated. For the text and DNA datasets, 10-fold-cross-validation was used.

This can be seen by plotting these results in scatter plots, as is done in Figures 1-3. Each point corresponds to the error on a dataset when used with Feature-mine compared to the method without Feature-mine. Points above the $y = x$ line thus represent datasets for which the use of Feature-mine yielded improved learning results. Our original motivation for building Feature-mine was to find multi-attribute features that overcome the greedy nature of learners such as C4.5 and Ripper, and indeed for both learners, Feature-mine creates features that often help learning. Notably, even though C4.5 consistently finds classifiers with low error rates, Feature-mine's features still help learning. Finally, the results with PEBLS show that Feature-mine can be of help even with learning algorithms not based on the greedy use of an information-theoretic measure.

5 Final Remarks

We presented a method of preprocessing a collection of training data before passing it on to a learning method in order to form new features that can improve the results of learning. Our method determines new features by manipulating both the positive and negative examples in the training set to determine co-occurences of feature values. In ongoing work we are exploring other methods for recognizing patterns in data that can be used to form macro-features that help across a range of learning methods.

In other work we have explored using unlabeled examples and background knowledge to aid in the classification task [13]. We plan to incorporate some of these ideas of external knowledge into the feature creation task as well.

References

[1] Tom Mitchell. *Machine Learning.* McGraw-Hill, New York, 1997.

[2] R. S. Michalski. A theory an methodology of inductive learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann, 1983.

[3] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[4] E. Bloedorn and R. S. Michalski. Data-driven constructive induction. *IEEE Intelligent Systems, Special Issue on Feature Transformation and Subset Selection*, 1998.

[5] T. Imielinski and A. Virmani. MSQL: A query language for database mining. *Data Mining and Knowledge Discovery*, 3:373–408, 1999.

[6] T. Imielinski, A. Virmani, and A. Abdulghani. DataMine: Application programming interface and query language for database mining. In *In proceedings of KDD*, 1996.

[7] Sergey Brin, Rajeev Motwani, and Craig Silverstein. Beyond market baskets: generalizing association rules to correlations. In *In proceedings of SIGMOD*, pages 265–276, 1997.

[8] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kauffmann, San Mateo, CA, 1993.

[9] William Cohen. Fast effective rule induction. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 1992.

[10] Scott Cost and Steven Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 1:57–78, 1993.

[11] Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–100, 1990.

[12] Daniel Kudenko and Haym Hirsh. Feature generation for sequence categorization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

[13] S. Zelikovitz and H. Hirsh. Integrating background knowledge into nearest-Neighbor text classification. In *Advances in Case-Based Reasoning, 6th European Conference, ECCBR 2002 Aberdeen, Scotland, UK, September 4-7, 2002, Proceedings*, volume 2416 of *Lecture Notes in Computer Science*. Springer, 2002.