Appears in the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)

Using Text Classifiers for Numerical Classification

Sofus A. Macskassy, Haym Hirsh, Arunava Banerjee, Aynur A. Dayanik
{sofmac,arunava,aynur,hirsh}@cs.rutgers.edu
Department of Computer Science
Rutgers University
110 Frelinghuysen Rd
Piscataway, NJ 08854-8019

Abstract

Consider a supervised learning problem in which examples contain both numerical- and text-valued features. To use traditional feature-vector-based learning methods, one could treat the presence or absence of a word as a Boolean feature and use these binary-valued features together with the numerical features. However, the use of a text-classification system on this is a bit more problematic — in the most straight-forward approach each number would be considered a distinct token and treated as a word. This paper presents an alternative approach for the use of text classification methods for supervised learning problems with numerical-valued features in which the numerical features are converted into bag-of-words features, thereby making them directly usable by text classification methods. We show that even on purely numerical-valued data the results of text-classification on the derived text-like representation outperforms the more naive numbers-as-tokens representation and, more importantly, is competitive with mature numerical classification methods such as C4.5 and Ripper.

1 Introduction

The machine learning community has spent many years developing robust classifier-learning methods, with C4.5 [Quinlan, 1993] and Ripper [Cohen, 1995] two popular examples of such methods. Although for many years the focus has been on numerical and discrete-valued classification tasks, over the last decade there has also been considerable attention to text-classification problems [Sebastiani, 1999; Yang, 1999]. Typically such methods are applied by treating the presence or absence of each word as a separate Boolean feature. This is commonly performed either directly, by generating a large number of such features, one for each word, or indirectly, by the use of set-valued features [Cohen, 1996], in which each text-valued field of the examples is viewed as a single feature whose value for an example is the set of words that are present in that field for this example.

The information retrieval community has similarly spent many years developing robust retrieval methods applicable to many retrieval tasks concerning text-containing documents, with vector-space methods [Salton, 1991] being the best known examples of techniques in this area. Although for many years the focus has been primarily on retrieval tasks, here, too, the last decade has seen a significant increase in interest in the use of such methods for text-classification tasks. The most common techniques use the retrieval engine as the basis for a distance metric between examples, for either direct use with nearest-neighbor methods [Yang and Chute, 1994], or, based on the closely related Rocchio [1971] relevance feedback technique, for use after creating a summary "document" for each class and retrieving the nearest one [Schapire *et al.*, 1998].

The irony is that although we now have much experience on placing text-clasification problems in the realm of numerical-classification methods, little attention has been brought to the question of whether numerical-classification problems can be effectively brought into the realm of text-classification methods. Since the text-retrieval methods on which they are based have many decade's maturity, if done effectively they have the potential of broadening further our base of methods for numerical classification.

More importantly for us, however, is the fact that many real-world problems involve a combination of both text- and numerical-valued features. For example, we came to ask these questions by confronting the problem of email classification, where we wanted to explore instance-representations that considered not only the text of each message, but also the length of the message or the time of day at which it is received [Macskassy *et al.*, 1999]. Although the machine-learning-derived methods that we now know how to apply to pure text-classification problems could be directly applied to these "mixed-mode" problems, the application of information-retrieval-based classification methods was more problematic. The most straight-forward approach is to treat each number that a feature may take on as a distinct "word", and proceed with the use of a text-classification method using the combination of true words and tokens-for-numbers words. The problem is that this makes the numbers 1 and 2 as dissimilar as the numbers 1 and 1000 — all three values are unrelated tokens to the classification method. What we would like is an approach to applying text-classification methods to problems with numerical-valued features so that the distance between such numerical values is able to be discerned by the classification method.

This paper presents one way to do just this, converting numerical features into features to which information-retrieval-based text-classification methods can apply. Our approach presumes the use of text-classification methods that treat a piece of text as a "bag of words", representing a text object by an unordered set of tokens present in the text (most commonly words, but occasionally tokens of a more complex derivation).

The core of our approach is, roughly, to convert each number into a bag of tokens such that two numbers that are close together have more tokens in common in their respective bags than would two numbers that are farther apart. The high-level idea is to create a set of landmark values for each numerical feature, and assign two tokens to each such landmark. Every value of a feature for an example will be compared to each

landmark for that feature. If the example's value is less than or equal to the landmark, the first of that landmark's two tokens is placed in that example's "bag". If the value is more than the landmark the second token is instead used in the bag. The result is that every feature gets converted into a bag of tokens, with each bag containing the same number of entries, each differing only to reflect on which side of each landmark a value lies.[1]

The key question then becomes how to pick landmark values. Although our experiments show that even fairly naive approaches for selecting landmarks can perform quite well, we instead appeal to the body of work on feature discretization that has already been well-studied within machine learning [Catlett, 1991; Kerber, 1992; Fayyad and Irani, 1993; Dougherty *et al.*, 1995; Kohavi and Sahami, 1996; Frank and Witten, 1999]. Learning methods such as C4.5 would normally have to consider a large number of possible tests on each numerical feature, in the worst case one between each consecutive pair of values that a feature takes on. These discretization methods instead use a variety of heuristic means to identify a more modest — and hence more tractable — subset of tests to consider during the learning process. Our approach is thus to apply such methods to identify a set of landmark values for each numerical feature and create two possible tokens for each landmark value, exactly one of which is assigned to each example for each landmark. The result is a "bag of words" representation for each example that can then be used by text-classification methods.

In the remainder of this paper we first describe our approach for converting numerical features into bag-of-words features in more detail, including the landmark-selection methods that we used. We then describe our experimental evaluation of our approach: the learning methods, evaluation methods used, and our results — which show that the text-classification methods using our bag-of-words representation perform competitively with the well-used methods C4.5 and Ripper when applied to the original numerical data. We conclude the paper with some analysis of these results and some final remarks.

2 Converting Numbers to Bags of Tokens

The approach taken in this paper is to convert every number into a set of tokens such that if two values are close, these sets will be similar, and if the values are further apart the sets will be less similar. This is done for each feature by finding a set of "landmark values" or "split-points" within the feature's range of legitimate values by analyzing the values that the feature is observed to take on among the training examples. Given an example, its numerical value for a given feature is compared to each split-point for that feature generated by our approach, and for each such comparison a token will be added, representing either that the value is less than or equal to the particular split-point or greater than that split-point. This will result in exactly one token being added per split-point.

For example, consider a news-story classification task that includes a numerical feature representing the story's length. We can artificially invent a set of split-points to demonstrate our process, such as 500, 1500, and 4000. For each split-point we define two tokens, one for either side of the split-point a value may lie. Using the above split-points for the length of a news-story would result in the tokens "lengthunder500", "lengthover500", "lengthunder1500", "lengthover1500", "lengthunder4000", and "lengthover4000". A new message of length 3000 would thereby have its length feature converted into the set of tokens "lengthover500", "lengthover1500", and "lengthunder4000". These would be added to the bag-of-words representation of the example — whether the other words in the bag were created from other numerical features, or the result of pre-existing text-valued features. More abstractly, consider the hypothetical numerical feature plotted along a number line in Figure 1. If a training example is obtained whose value for this feature falls in bin2, the set {morethansplit1, lessthansplit2, lessthansplit3, lessthansplit4} would be the bag-of-words representation created for this value. Note that more than one value can be given the same representation, as long as they all fall between the same two consecutive split-points.

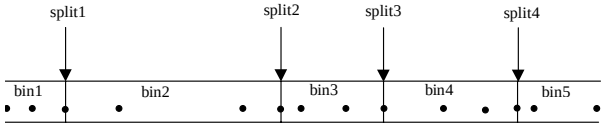

Figure 1: An example feature range and construction of bins. Dots represent values and rectangles represent bins.

The key question, of course, is how these split-points are selected. We use two methods in our main results. The first, called the *MDL* method, uses an existing entropy-based method for discretization to find good split-points [Fayyad and Irani, 1993]. This method is very similar to one that uses C4.5 on the training data, restricting it to ignore all but the single feature for which split-points are being selected, harvesting the decision points found within each of the internal nodes of the tree [Kohavi and Sahami, 1996]. The second method, which we call the *density* method, selects split-points that yield equal-density bins — where the number of values that fall between consecutive split-points stays roughly constant. The number of bins is selected using hill-climbing on error rate using a hold-out set. In the rest of this section we describe these two methods in more detail, concluding with a discussion of how we handle examples that are missing values for one or more of their features.

2.1 The MDL Method

The MDL method [Fayyad and Irani, 1993; Kohavi and Sahami, 1996] makes use of information-theoretic techniques to analyse the values of a numeric feature and create split-points that have high information gain. It does so in a recursive fashion, finding one split-point in the overall set of values, then finding another split-point in each of the two created subsets, until a stopping criteria based on Minimum Description Length priciples [Rissanen, 1987] is met. Each numerical feature is treated separately, yielding a different set of split-points for each feature. Due to space limitations we refer the reader to the given citations for further details about this popular discretization algorithm.

2.2 The Density Method

The density method (described in algorithmic form in Figure 2) begins by obtaining and sorting all values observed in the data for a feature f, yielding an ordered list S_f. Thus, for example, the result for some feature f might be $S_f = \langle 1, 2, 2, 2, 5, 8, 8, 8, 9 \rangle$. Given some desired number of splits k, the density method splits the feature set S_f into $k+1$ sets of equal size (except when rounding effects require being off by one) such that split-point s_j is the $(|S_f| \times \lfloor \frac{j}{k+1} \rfloor)$-th point in S_f. Using this split-point, two tokens are generated; one for when a numerical value is less than or equal to the split-point, and another for when the numerical value is greater than the

[1] However, as we will explain later, there may be fewer values in case of missing values.

Inputs: Sets of values for each numeric feature.
Algorithm:
$currError \leftarrow 100$ /* assume errors run from $0 - 100$ */
$lastError \leftarrow 100$
$numSplits \leftarrow 1$
$maxSplits \leftarrow \text{argmax}_{f \in \text{numerical features}}(|S_f|)$
while($numSplits < maxSplits$) do
  for each numerical feature f
    $n_f \leftarrow \min(numSplits, |S_f|)$
    /* Create n_f split-points for feature f. */
    Use as split-points for f the j-th element of S_f
      for $j = |S_f| \times \lfloor \frac{i}{n_f+1} \rfloor$ with i running from 1 to n_f
  end
  Divide data into 70% for $train$ and 30% for $test$.
  $C \leftarrow$ Run learner on $train$ with current split-points.
  $currError \leftarrow$ Evaluate C on $test$.
  if($currError = 0$) do
    $maxSplits \leftarrow 0$
    $lastError \leftarrow currError$
  else if($lastError < currError$) do
    $numSplits \leftarrow numSplits/2$
    $maxSplits \leftarrow 0$
  else
    $numSplits \leftarrow 2 \times numSplits$
    $lastError \leftarrow currError$
  end
end
Outputs: $lastError$

Figure 2: Density algorithm for finding split-points.

split-point. k — the final number of split-points — is found using a global hill-climbing search with the number of split-points growing geometrically by a factor of two until the observed error rate on a 30% hold-out set is observed to increase or reach 0.

One final detail of the density method is that the algorithm in Figure 2 is actually run twice. The first time is based on a "volumetric" density calculation, where duplicate values are included in the analysis. After doing this the algorithm is run a second time after duplicate values have been removed, yielding a second set of split points that have an (approximately) equal number of *distinct* values between each split-point. Thus, for example, for the feature f this second run would now use the list $S_f = \langle 1, 2, 5, 8, 9 \rangle$. Whichever method yielded a lower error rate gives the final set of split points. If they have the same performance, whichever had fewer split-points is selected.

2.3 Missing Values

A common occurrence for many learning problems is when some examples are missing values for some of the features. This can be a complicating factor both during the learning phase, when assessing the importance of features in forming some learning result, and in classification, when making a decision when values of some of the attributes are unavailable. Common approaches range from simply deleting data or features to remove such occurrences, to imputing some value for the feature — such as through learning or through something as simple as using the median, mean, or mode value in the training data, to more complex methods such as are used in learning algorithms such as C4.5. Our approach for creating bag-of-word features out of numerical features contributes another interesting way to handle missing values. The idea is, quite simply, to add no tokens for a feature of an example when the value for this feature is missing. By not committing to any of the tokens that this feature might otherwise have added it neither contributes nor detracts from the classification process, allowing it to rely on the remaining features in assigning a label to the example.

3 Learning Algorithms

In this section we briefly describe the learning algorithms that we use in our experiments. Recall that our goal is to demonstrate that, using our approach, text-classification methods can perform as credibly on numerical classification problems as more traditional methods that were explicitly crafted for such problems. To show this we use a sampling of four different approaches for text classification. Our first is based on the Rocchio-based vector-space method for text retrieval mentioned earlier, which we label TFIDF [Joachims, 1997; Schapire *et al.*, 1998; Sebastiani, 1999]. We also consider two probabilistic classification methods that have become popular for text classification, Naive Bayes [Domingos and Pazzani, 1996; Joachims, 1997; Mitchell, 1997] and Maximum Entropy [Nigam *et al.*, 1999]. Finally, we also use the Ripper rule learning system [Cohen, 1995; 1996], using its capability of handling set-valued features so as to handle text classification problems in a fairly direct fashion. (The first three methods used the implementations available as part of the *Rainbow* text-classification system [McCallum, 1996].) The baselines to which we compare the text-classification methods are two popular "off the shelf" learning methods, C4.5 (release 8) [Quinlan, 1993] and Ripper. Note that Ripper has been mentioned twice, once as a text-classification method using our transformed features encoded for Ripper as set-valued features, and the other using the original numerical features in the same fashion as they are used with C4.5. Thus Ripper is in the unique position of being both a text-classification method when used with one representation, and as a numerical classification method when used with the other. Missing values were handled for the text-classification methods as discussed earlier, by simply not generating any tokens for a feature of an example when it had no given value, and for C4.5 and Ripper (when used with numerical features) by their built-in techniques for handling missing values.

The *TFIDF* classifier is based on the relevance feedback algorithm by Rocchio [1971] using the vector space retrieval model. This algorithm represents documents as vectors so that documents with similar content have similar vectors. Each component of such a vector corresponds to a term in the document, typically a word. The weight of each component is computed using the TFIDF weighting scheme, which tries to reward words that occur many times but in few documents. In the learning phase, a prototype vector is formed for each class from the positive and negative examples of that class. To classify a new document d, the cosines of the prototype vectors with the corresponding document vector are calculated for each class. d is assigned to the class with which its document vector has the highest cosine.

Naive Bayes is a probabilistic approach to inductive learning. It estimates the a posteriori probability that an example belongs to a class given the observed feature values of the example, assuming independence of the features. The class with the maximum a posteriori probability is assigned to the example. The naive Bayes classifier used here is specifically designed for text classification problems.

The *Maximum Entropy* classifier (labeled MAXENT in our results) estimates the conditional distribution of the class label given a document, which is a set of word-count features. The high-level idea of this technique is, roughly, that uniform dis-

tributions should be prefered in the absence of external knowledge. A set of constraints for the model are derived from the labeled training data, which are expected values of the features. These constraints characterize the class-specific expectations for the model distribution and may lead to minimal non-uniform distributions. The solution to the maximum entropy formulation is found by the improved iterative scaling algorithm [Nigam *et al.*, 1999].

Ripper is a learning method that forms sets of rules, where each rule tests a conjunction of conditions on feature values. Rules are returned as an ordered list, and the first successful rule provides the prediction for the class label of a new example. Importantly, Ripper allows attributes that take on sets as values, in addition to numeric and nominal features, and a condition can test whether a particular item is part of the value that the attribute takes on for a given example. This was designed to make Ripper particularly convenient to use on text data, where rather than listing each word as a separate feature, a single set-valued feature that contains all of an instance's words is used instead. Rules are formed in a greedy fashion, with each rule being built by adding conditions one at a time, using an information-theoretic metric that rewards tests that cause a rule to exclude additional negative data while still hopefully covering many positive examples. New rules are formed until a sufficient amount of the data has been covered. A final pruning stage adjusts the rule set in light of the resulting performance of the full set of rules on the data.

C4.5 is a widely used decision tree learning algorithm. It uses a fixed sets of attributes, and creates a decision tree to classify an instance into a fixed set of class-labels. At every step, if the remaining instances are all of the same class, it predicts that class, otherwise, it chooses the attribute with the highest *information gain* and creates a decision based on that attribute to split the training set into one subset per discrete value of the feature, or two subsets based on a threshold-comparison for continuous features. It recursively does this until all nodes are final, or a certain user-specified threshold is met. Once the decision tree is built, C4.5 prunes the tree to avoid overfitting, again based on a user-specified setting.

4 Evaluation Methodology

To compare our text-like encoding of numbers when used with text-classification systems to the use of C4.5 and Ripper on the original numerical features we used 23 data sets taken from the UCI repository [Blake and Merz, 1998]. Table 1 shows the characteristics of these datasets. The first 14 represent problems where all the features are numeric. The final 9 represent problems in which the designated number of features are numeric and the rest are discrete or binary-valued.

The accuracy of a learner was done through ten-fold stratified cross-validation [Kohavi, 1995]. Each dataset was represented in one of four ways for our experiments:

- The original feature encoding — using numbers — for use with C4.5 and Ripper.
- The bag-of-words encoding generated by the density method, for use with the four text-classifications.
- The bag-of-words encoding generated by the MDL method, for use with the four text-classifications.
- The bag-of-words encoding generated using the tokens-for-numbers approach, for use with the five text-classifications. This was accomplished by converting every number into its English words – for example, "5" becomes "five" and "2.3" becomes "twopointthree".

The first of these represents our baseline, using a machine learning method designed for numerical classification. The next two are the new approaches presented in this paper. The final one is the representation that simply treats each number as a distinct "word" without regard to its value.

Dataset	# of Instances	# of Features	# of Numeric Features	# of Classes	Base Accuracy (%)
bcancerw	699	10	10	2	66
diabetes	768	8	8	2	65
glass	214	9	9	6	36
hungarian	294	13	13	2	64
ionosphere	351	34	34	2	64
iris	150	4	4	3	33
liver	345	6	6	2	58
musk	476	166	166	2	57
new-thyroid	215	5	5	5	70
page-blocks	5473	10	10	5	90
segmentation	2310	19	19	7	14
sonar	208	60	60	2	53
vehicle	846	18	18	4	26
wine	178	13	13	2	40
arrhythmia	452	279	206	16	54
autos	205	26	16	2	88
cleveland	303	13	6	2	54
credit-app	690	15	6	2	56
cylinder-bands	512	39	20	2	61
echocardiogram[1]	132	13	9	2	44
horse	368	22	7	2	63
post-operative	90	8	1	3	71
sponge	76	45	3	3	92

[1] The MDL approach was unable to find any split-points for this data-set, so it is omitted in any comparisons on the MDL method.

Table 1: Properties of all datasets.

5 Results

The first question we ask is the key one: To what extent does our approach yield a competitive learning method on numerical classification problems? Figure 3 shows the results of comparing the four text-learning methods using our MDL-algorithm features to the two numerical classification methods. Each point represents a single data set, where the x-axis is the accuracy of either C4.5 or Ripper and the y-axis is the accuracy of one of the four text methods. Points above the y=x line represent cases where the numerical-classification method was inferior to our use of a text-classification method, and points below the line are cases where the numerical method was superior. The qualitative flavor of this graph is that the MDL-algorithm features allows text-classification methods to perform credibly in many cases, exceeding numerical methods in some cases, although performing less successfully in many cases as well. We plot in Figure 4 a similar graph comparing the four text methods using the density-algorithm features to the two numerical methods. (The "outlier" cases at the bottom of the graphs are for the post-operative data set, which has only 90 examples and only 1 of its 8 features being numeric.)

Since the preceding graphs collapse eight different comparisons (four text methods versus two numerical methods)

	TFIDF	NB	MAXENT	Ripper
MDL/C4.5	9/13/0	11/10/1	12/10/0	7/14/1
MDL/Ripper	8/14/0	11/9/2	11/11/0	8/12/2
Density/C4.5	4/19/0	7/15/1	13/10/0	9/13/1
Density/Ripper	4/19/0	8/14/1	13/9/1	8/13/2

Table 2: Comparing the number of wins/losses/ties for each featurization (MDL first two rows, density second two rows) when coupled with one of the four text-classification methods labeling the columns, versus a numerical method.

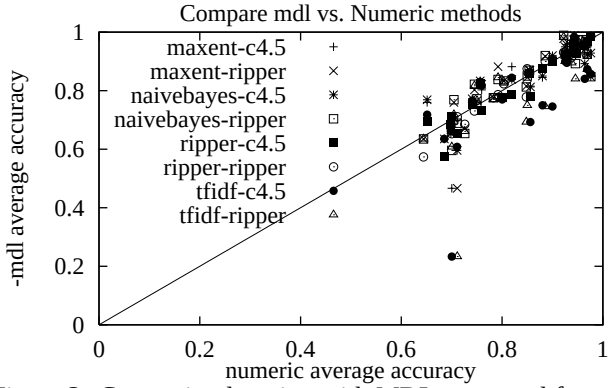

Figure 3: Comparing learning with MDL-generated features to numeric learning.

into a picture Table 2 also shows for how many data sets each text method beat a numerical method. Each entry in the table is the number of wins/losses/ties for the new featurization method used with a text method compared to a numerical classification method. The columns label the text method used. The first two rows are results when the MDL method is used, the next two are for the density method, in each case the first comparison is to numerical classification with C4.5 comes in the first of the two rows, followed by Ripper. These results show that in a non-trivial number of cases the use of our approach for converting numerical features into text-based data beats out the popular learning methods, with absolutle improvements in accuracy ranging as high as 12%. While these results do *not* show that the approach is unconditionally superior to numerical classification, they do show that the approach does merit consideration for use as a numerical classification method.

The next question we ask is whether the use of two different featurization methods is necessary: Does either dominate the other? Figure /reffig:mdlvsdens shows the results of such a comparison, where each point represents a single data set and a text learning method, with the x-axis representing the result of using the MDL method with that learning algorithm, and the y-axis representing the result of using the density method with that learning algorithm. Here, too, the results show that neither method is clearly superior, with perhaps a bit better performance in general by the MDL method, but with some cases still going to the density method.

6 Additional Analysis

We began the paper stating that the obvious approach to converting numbers into text-based feature is convert each number into a unique token, to be added as-is to the set of words

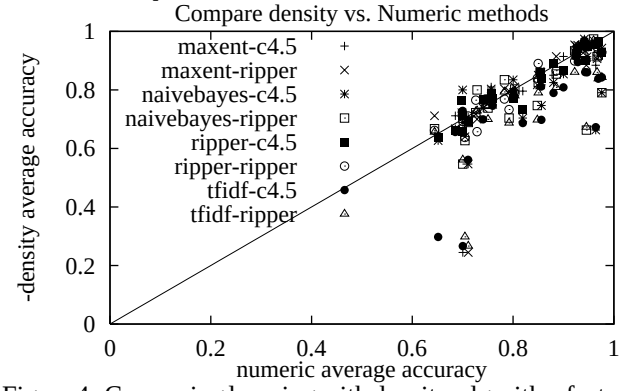

Figure 4: Comparing learning with density-algorithm features to numeric learning.

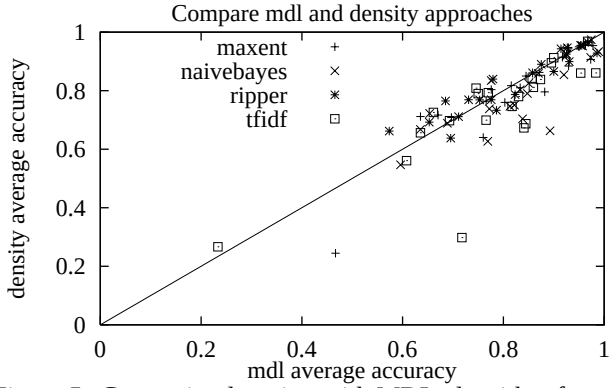

Figure 5: Comparing learning with MDL-algorithm features to density-algorithm features.

for a given example. One additional question we can ask is whether the complexity of our methods are necessary: Perhaps this simple tokenization approach performs as effectively? Figure 6 shows an analysis of this question. Each point is a data set, with the x-axis value representing the accuracy of the tokenization approach with a particular text-classification method, and the y-axis represents the accuracy with one of our two featurization methods using the same learning method. As is clearly evident from the figure, the tokenization approach is not as effective in general as our more sophisticated approach.

We conclude this section by noting that Kohavi and Sahami [1996] discuss a different discretization method that is very similar to the MDL method. This method simply runs C4.5 on the data, ignoring all features except the one for which split-points are being created. Kohavi and Sahami show that this method is slightly inferior to the MDL approach. However, just because it is inferior for discretization for decision-tree learning does not imply that it must be the case here, too. To test this we compared the four text classification methods using the MDL method to the C4.5 method. Figure 7 shows the results of this experiment. Each point represents a data set and a learning method. The x-axis represents the accuracy of the C4.5 approach, and the y-axis represents the accuracy of the MDL approach. As is clear, although there is some difference between the performance of the methods, they are somewhat similar in behavior.

7 Final Remarks

This paper has described an approach for converting numeric features into a representation enabling the application of text-classification methods to problems that have traditionally been solved solely using numerical-classification meth-

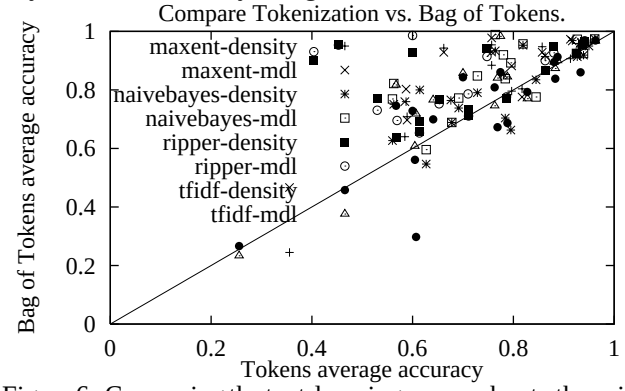

Figure 6: Comparing the text-learning approaches to the naive tokenization approach.

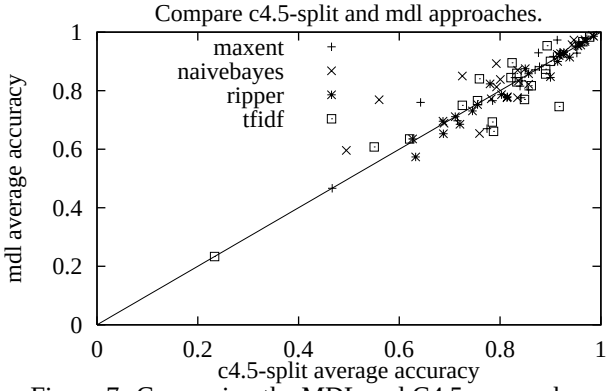

Figure 7: Comparing the MDL and C4.5 approaches.

ods. In addition to opening up the use of text-methods to problems that involve "mixed-mode" data — both numerical- and text-valued features — it yields a new approach to numerical-classification method in its own right. Our experiments show that in a non-trivial number of cases the resulting methods outperform highly optimized numerical-classification methods. Also importantly, our experiments show that our approach yields a vast improvement over the naive method of converting a numeric into its equivalent textual token.

There are many directions in which we are now taking this work. Our original motivation for performing this work was to broaden the class of learning methods that can be applied to mixed-mode data. Now that we have done so, we can return to some of the work that motivated this, performing additional evaluations of this work on mixed-mode data. Doing so, however, requires a set of benchmark problems, something that does not presently exist. We are therefore in the process of creating such data sets so we can perform this evaluation process. We also noted that our approach yields an intriguing way to deal with data with missing values, and understanding its benefits and liabilities compared to other approaches remains a question that we hope to explore. Finally, as is usually the case when comparing any two learning methods that are successful in competing cases, it is difficult to make any definitive statements about when they each may be successful. Various conjectures include differences in the amount of missing values in the different data sets, the number of numeric versus non-numeric features, etc. For the given data sets we were unable to discern any such pattern in our results. This remains an important question that we also plan to study.

Acknowledgments

We would like to thank Foster Provost, Lyle Ungar, and members of the Rutgers Machine Learning Research Group for helpful comments and discussions.

References

[Blake and Merz, 1998] C. L. Blake and C. J. Merz. Uci repository of machine learning databases, 1998.

[Catlett, 1991] J. Catlett. On changing continuous attributes into ordered discrete attributes. In Y. Kodratoff, editor, *Proceedings of the European Working Session on Learning*, pages 164–178. Berling: Springer-Verlag, 1991.

[Cohen, 1995] W. W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, Lake Tahoe, California, 1995.

[Cohen, 1996] W. W. Cohen. Learning trees and rules with set–valued features. In *AAAI96*, 1996.

[Domingos and Pazzani, 1996] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of simple bayesian classifier. In *Proceedings of the 13th International Conference on Machine Learning*, pages 105–112, 1996.

[Dougherty *et al.*, 1995] K. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of the 12th International Conference on Machine Learning*, pages 194–202. Morgan Kaufmann, 1995.

[Fayyad and Irani, 1993] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on AI*, pages 1022–1027. Morgan Kaufmann, 1993.

[Frank and Witten, 1999] E. Frank and I. H. Witten. Making better use of global discretization. In *Proceedings of the 17th International Conference on Machine Learning*, Slovenia, 1999.

[Joachims, 1997] T. Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, 1997.

[Kerber, 1992] R. Kerber. Discretization of numeric attributes. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 123–128, Menlo Park, CA, 1992. AAAI Press/MIT Press.

[Kohavi and Sahami, 1996] R. Kohavi and M. Sahami. Error-based and entropy-based discretization of continuous features. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 114–119, Menlo Park, CA, 1996. AAAI Press/MIT Press.

[Kohavi, 1995] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1137–1143, San Francisco, CA, 1995. Morgan Kaufmann.

[Macskassy *et al.*, 1999] S. A. Macskassy, A. A. Dayanik, and H. Hirsh. Emailvalet: Learning user preferences for wireless email. In *Proceedings of Learning about Users Workshop, IJCAI'99*, Stockholm, Sweden, 1999.

[McCallum, 1996] A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. `http://www.cs.cmu.edu/~mccallum/bow`, 1996.

[Mitchell, 1997] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[Nigam *et al.*, 1999] K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. In *Proceedings of Machine Learning for Information Filtering Workshop, IJCAI'99*, Stockholm, Sweden, 1999.

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

[Rissanen, 1987] I. Rissanen. Minimum description length principle. *Encyclopedia of Statistical Sciences*, 5:523–527, 1987.

[Rocchio, 1971] J. Rocchio. Relevance feedback in information retrieval. In Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter 14, pages 313–323. Prentice–Hall, 1971.

[Salton, 1991] G. Salton. Developments in automatic text retrieval. *Science*, 253:974–979, 1991.

[Schapire *et al.*, 1998] R. Schapire, Y. Singer, and A. Singal. Boosting and rocchio applied to text filtering. In *Proceedings of ACM SIGIR*, pages 215–223, 1998.

[Sebastiani, 1999] F. Sebastiani. Machine learning in automated text categorisation: a survey. Technical Report IEI-B4-31-1999, Istituto di Elaborazione dell'Informazione, 1999.

[Yang and Chute, 1994] Y. Yang and C. Chute. An example-based mapping method for text classification and retrieval. *ACM Transactions on Information Systems*, 12(3):252–277, 1994.

[Yang, 1999] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1/2):67–88, 1999.