Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Artificial Intelligence 143 (2003) 51–77

Artificial Intelligence

www.elsevier.com/locate/artint

Converting numerical classification into text classification ☆

Sofus A. Macskassy *, Haym Hirsh, Arunava Banerjee [1], Aynur A. Dayanik

Department of Computer Science, Rutgers University, 110 Frelinghuysen Rd, Piscataway, NJ 08854-8019, USA

Received 30 November 2001; received in revised form 29 July 2002

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, Ripper, and SVM. We further show that on mixed-mode data adding numerical features using our approach can improve performance over not adding those features.

Keywords: Machine learning; Text classification; Information retrieval

☆ This is an extended version of the paper presented at IJCAI-2001, Seattle, WA, 2001.

* Corresponding author.
E-mail addresses: sofmac@cs.rutgers.edu (S.A. Macskassy), hirsh@cs.rutgers.edu (H. Hirsh), arunavab@bcs.rochester.edu (A. Banerjee), aynur@cs.rutgers.edu (A.A. Dayanik).
[1] Now at: Brain and Cognitive Sciences, University of Rochester, Rochester, NY 14627-0268.

0004-3702/02/$ – see front matter
PII: S0004-3702(02)00359-4

1. Introduction

The machine learning community has spent many years developing robust classifier-learning methods, with C4.5 [1] and Ripper [2] 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 [3,4]. 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 [5], 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 [6] 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 [7], or, based on the closely related Rocchio [8] relevance feedback technique, for use after creating a summary "document" for each class and retrieving the nearest one [9].

The irony is that although we now have much experience on placing text-classification 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.

Moreover, 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 [10]. 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.

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 [11–16]. 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.

This approach is similar to the idea of "thermometer coding" used in the neural network community to convert numerical values into a set of Boolean variables [17]. Thermometer coding also uses a set of threshold or landmark values, only where each yields a Boolean variable representing on which side of the threshold a given value lies. Here, however, each of these Boolean variable can be viewed as being further converted into two tokens, one of which is added to the token-based representation of a number depending on which side of the threshold it lies. Although the thresholds for thermometer coding are typically selected in an *ad hoc* fashion, [18] uses entropy-based measures to build up a set of landmarks. Our approach uses similar ideas (selecting thresholds on a feature-by-feature basis, in contrast to the selection scheme used by [18]), only with an eye towards selecting landmarks that yield good sets of tokens for 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 next show that for an email classification task containing both text and numerical features, we improve performance of standard text-classification methods, and also that we outperform methods such as Ripper and SVM, which both are well-known learning

methods that can handle bags of words as well as numerical features. We conclude with 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 Fig. 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.

The key question, of course, is how these split-points are selected. We use two methods in our main results. The first, called the *entropy* method, uses an existing entropy-based method for discretization to find good split-points [13]. 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

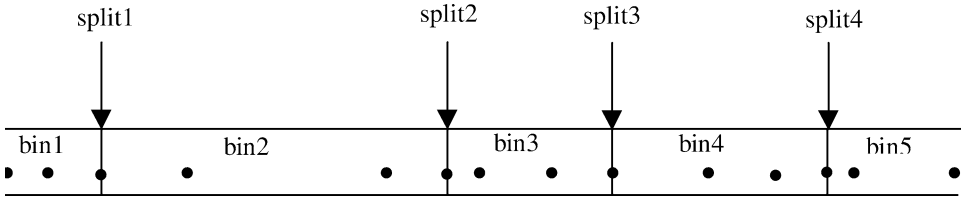

Fig. 1. An example feature range and construction of bins. Dots represent values and rectangles represent bins.

the internal nodes of the tree [15]. 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 have missing values for one or more of their features.

2.1. The entropy method

The entropy method [13,15] makes use of information-theoretic techniques to analyze 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 principles [19] is met. Each numerical feature is treated separately, yielding a different set of split-points for each feature. Further details about this popular discretization algorithm can be found in the given citations.

2.2. The density method

The density method (described in algorithmic form in Fig. 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 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 Fig. 2 is actually run twice. The first time is based on a "volumetric" density calculation, where duplicate values are included in the analysis.[2] 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] In fact, because of this, a particular split-point value might be selected more than once, having the tokens for that particular split also appear more than once in the resulting bag-of-tokens.

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 $\leftarrow$ argmax$_{f \in \text{numerical features}}(|S_f|)$
while(*numSplits* $<$ *maxSplits*) do
  for each numerical feature f
    $n_f \leftarrow \min(\textit{numSplits}, |S_f|)$
    /* Create n_f split-points for feature f. */
    Use as split-points for f the jth 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*

Fig. 2. Density algorithm for finding split-points.

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 [3,9,20]. We also consider two probabilistic classification methods that have become popular for text classification, Naive Bayes [20–22] and Maximum Entropy [23]. Finally, we also use the Ripper rule learning system [2,5], using its capability of handling set-valued features so as to handle text classification problems in a fairly direct fashion, as well as a Support Vector Machine (SVM) algorithm [24]. The first three methods used the implementations available as part of the *Rainbow* text-classification system [25]. The SVM method uses the SVM^{light} package available at http://svmlight.joachims.org [26]. The baselines to which we compare the text-classification methods are three popular "off the shelf" learning methods, C4.5 (release 8) [1], Ripper, and SVM. Note that Ripper and SVM have been mentioned twice, first as text-classification methods, and second as numerical classification methods (using the original numerical features for Ripper, or a normalized numerical value used with SVM). Thus Ripper and SVM are in the position of being run as both a text-classification method when used with one representation, and as a numerical classification method when used with another. 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 Ripper and SVM when used with numerical features and for C4.5 by their built-in techniques for handling missing values.

The *TFIDF* classifier is based on the relevance feedback algorithm by Rocchio [8] 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 distributions should be preferred 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 [23].

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.

Support Vector Machines (SVMs) are learning machines which are based on statistical learning theory. They perform binary classification and regression estimation tasks. SVMs non-linearly map their n-dimensional input space into a higher-dimensional feature space. In this high-dimensional feature space a linear classifier is then constructed using quadratic programming. This latter step could potentially be very costly. SVMs make use of various *kernel* methods to optimize the calculation of inner numerical products. The implementation we use, $\text{SVM}^{\text{light}}$, also performs optimizations to reduce-dimensionality of the space in order to make SVMs feasible in large-dimension domains.

4. Case study: Numerical data sets

To compare our text-like encoding of numbers when used with text-classification systems to the use of C4.5, Ripper, and SVM on the original numerical features we used 22 data sets taken from the UCI repository [27]. Table 1 shows the characteristics of these data sets. The first 14 represent problems where all the features are numeric. The final 8 represent problems in which the designated number of features are numeric and the rest are discrete or binary-valued.

Table 1
Properties of the UCI datasets

Data set	# Instances	# Features	# Numeric Features	# Classes	Base Accuracy (%)
bcancerw	699	9	9	2	65.52
diabetes	768	8	8	2	65.10
echocardiogram	131	11	11	2	67.18
glass	214	9	9	2	76.17
hungarian	294	13	13	2	63.95
ionosphere	351	34	34	2	64.10
iris	150	4	4	3	33.33
liver	345	6	6	2	57.97
musk	476	166	166	2	56.51
new-thyroid	215	5	5	3	69.77
page-blocks	5473	10	10	5	89.77
segmentation	2310	19	19	7	14.29
sonar	208	60	60	2	53.37
vehicle[a]	846	18	18	4	25.77
wine	178	13	13	3	39.89
arrhythmia	452	279	272	16	54.20
autos	205	25	15	2	56.16
cleveland	303	13	13	2	54.13
credit-app	690	15	6	2	55.51
cylinder-bands	540	39	20	2	57.78
horse	368	23	7	2	66.30
sponge[b]	76	45	3	3	92.11

[a] This dataset was donated to the UCI repository from the Turing Institute, Glasgow, Scotland.
[b] The Entropy approach was unable to find any split-points for this data-set, so it is omitted in any comparisons on the Entropy method.

4.1. Evaluation methodology

The accuracy of a learner was done through ten-fold stratified cross-validation [28]. Each data set was represented in one of four ways for our experiments:

- The original feature encoding—using numbers—for use with C4.5, SVM, and Ripper.
- The bag-of-words encoding generated by the density method, for use with the five text-classification methods.
- The bag-of-words encoding generated by the Entropy method, for use with the five text-classification methods.
- The tokenization encoding generated using the tokens-for-numbers approach, for use with the five text-classification methods. 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

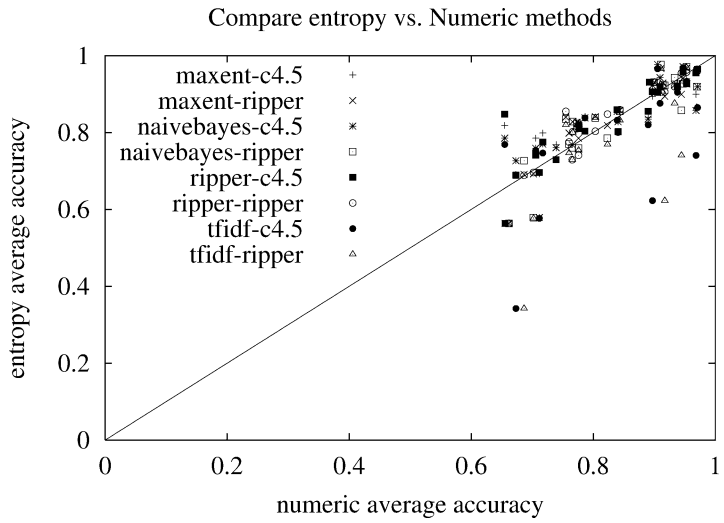

Fig. 3. Comparing learning with Entropy-generated features to numeric learning. Each of the four text-classifiers were compared against Ripper and C4.5. Points below the diagonal means the numeric methods had better accuracy, points above the diagonal means the Entropy method had better accuracy.

final one is the representation that simply treats each number as a distinct "word" without regard to its value.

4.2. Results

The first question we ask is a key one: To what extent does our approach yield a competitive learning method on numerical classification problems? Fig. 3 shows the results of comparing four of our five text-learning methods using our Entropy-algorithm features to the three numerical classification methods.[3] 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 Entropy-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 Fig. 4 a similar graph comparing the four text methods using the density-algorithm features to the two numerical methods. (The "outlier" cases showing at the bottom of both figures is for the arrhythmia dataset.)

[3] Comparisons to SVM are not shown on these graphs for readability. In fact, as is shown in Table 2, SVM had a slightly worse performance to the other numerical methods, and will therefore not be shown in any comparison graphs.

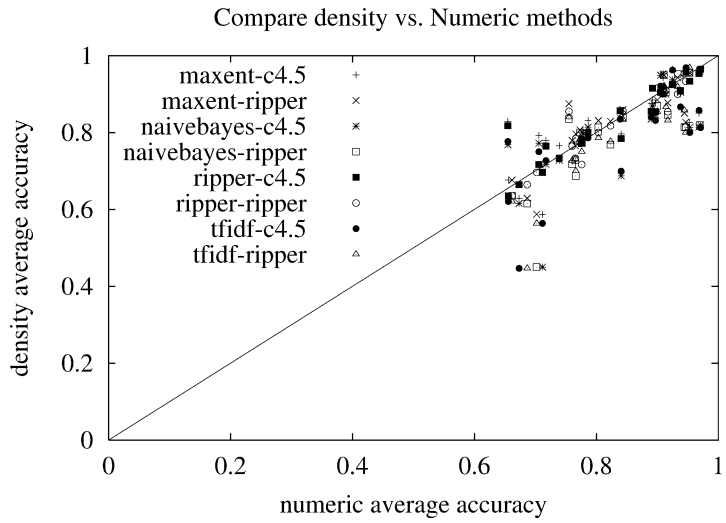

Fig. 4. Comparing learning with Density algorithm features to numeric learning.

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

	MAXENT	Naive Bayes	TFIDF	Ripper
Entropy/C4.5	12/9/0	13/8/0	8/13/0	11/10/0
Entropy/Ripper	13/8/0	13/8/0	7/14/0	11/10/0
Entropy/SVM-l	9/5/0	10/4/0	7/7/0	6/7/1
Entropy/SVM-q	10/4/0	9/5/0	5/9/0	6/7/1
Density/C4.5	12/10/0	8/13/1	8/14/0	7/14/1
Density/Ripper	14/8/0	6/15/1	5/17/0	8/13/1
Density/SVM-l	9/5/0	4/10/0	5/9/0	4/9/1
Density/SVM-q	8/6/0	3/10/1	4/9/1	4/9/1

Since the preceding graphs collapse eight different comparisons (four text methods versus two numerical methods) 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 four rows are results when the Entropy method is used, the next four are for the density method, in each case the first comparison is to numerical classification with C4.5, then with Ripper, then with SVM using linear (SVM-l) and quadratic (SVM-q) kernels.[4] We

[4] As SVM only handles two-class problems, we tested only on the two-class data sets. We decided not to test it on the n-class problems as its performance was not as strong as other numerical methods in these initial two-class problems. The table therefore only shows the comparisons from those initial fourteen runs.

Table 3
How often each type of classifier and encoding was significantly better at the 95% confidence level, using paired t-tests

	Density		Entropy splits		Totals	
	C4.5	Ripper	C4.5	Ripper	C4.5	Ripper
Splitting wins	11	8	27	19	19	46
Numeric wins	30	27	19	21	51	46
Insignificants	47	53	38	44	85	97

will therefore not include SVM in any comparison or discussion of performance on these data sets. The 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. 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.

We performed a statistical analysis, comparing each text-classifier (MAXENT, Naive Bayes, Ripper and TFIDF) using the two split-methods (Entropy and Density) with each of the two numerical learners (Ripper and C4.5). As we had run the experiments using ten-fold cross-validation, we used the errors from each of the runs to perform paired t-tests. A little less than half of the comparisons (162 out of 352 possible) had a significant difference between the splitting method and the numerical methods, all above the 95% confidence level. Of these, 65 times the split-type methods won, 49 times C4.5 won, and 48 times Ripper won. All data sets had differing ratios of winners, the only exception being the sponge data set with no difference as would be expected based on its characteristics. Table 3 shows a matrix counting for each type of split-method and numerical classifier pairing how often the split-method and how often the numerical classifier had a significant win at the 95% confidence level or above.[5]

The next question we ask is whether the use of two different featurization methods is necessary: Does either dominate the other? Fig. 5 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 Entropy method with that learning algorithm, and the y-axis representing the result of using the density method with that learning algorithm. Here, the results show that the Entropy method is clearly superior, though some cases still go to the density method. In particular, it seems that the TFIDF classifier consistently has problems with the Entropy featurization.

Again we performed a statistical analysis to compare the performances between the two types of splitting methods. We found a few interesting patterns. For all classifiers, the Entropy split method was the winner in the majority of cases where either of the methods had a significantly better performance, with Naive Bayes having only significant wins by the Entropy method. For the other methods, this was less so, though the Entropy split

[5] The comparisons add up to 344 as 8 of the comparisons could not be made. This was due to the fact that the split-methods were only able to find threshold values for one or none of the cross-validation training sets. We needed at least two runs to make a statistical significance test.

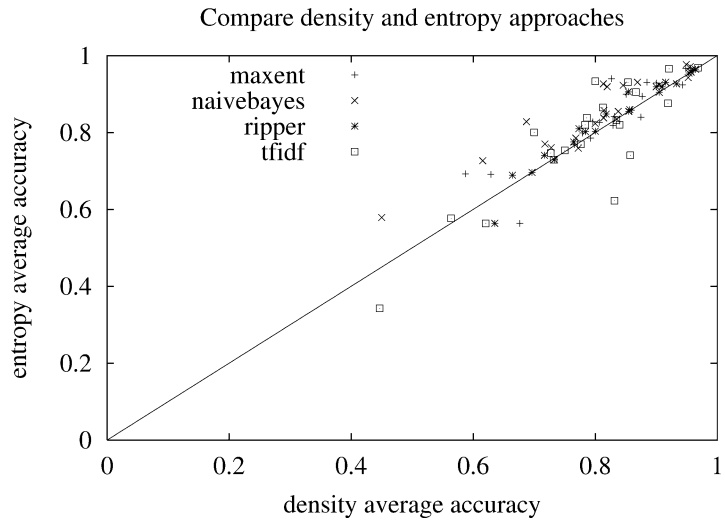

Fig. 5. Comparing learning with Entropy-algorithm features to density-algorithm features.

Table 4
How often, across the four text-classifiers, did each split-method have a significant win (at or above the 95% confidence level) over the other split-method

	MAXENT	Naive Bayes	TFIDF	Ripper
Density wins	1	0	3	1
Entropy-split wins	5	10	6	3
Insignificants	15	11	12	17

method was still generally better. Using Ripper, only a few significant wins was had by either split method. In 7 of the 10 cases where more than one classifier had a significant win for a particular dataset, if one split method outperformed the other, it did so regardless of which classifier is being used. However, we also found that in 8 of the 22 datasets, no method had a significant win at or above the 95% confidence level. Table 4 shows the actual number of wins for each type of split-method using each of the text-classifiers.

4.3. Additional analysis

In this section we explore the behavior of our methods in greater detail. First, we study whether the added complexity of our bag-of-words approach is necessary to exceed the performance of the naive tokenization method that converts each number into a unique token. Second, we attempt to characterize the behavior of split-point generation for each of the methods.

4.3.1. Comparison to simple tokenization

Fig. 6 shows an analysis of how our method compares to the naive tokenization approach. Each point is a data set, with the x-axis value representing the accuracy of

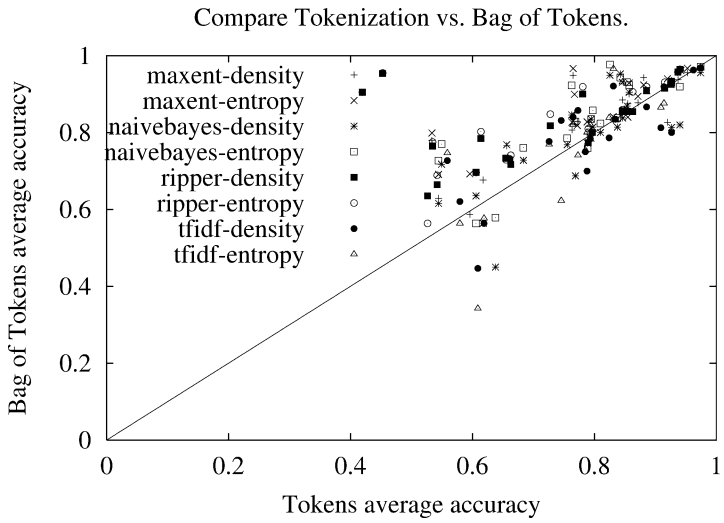

Fig. 6. Comparing the text-learning approaches to the naive tokenization approach.

Table 5
How often, across the four text-classifiers, did the Entropy split-method have a significant win (at or above the 95% confidence level) over Tokenization, and vice versa

	MAXENT	Naive Bayes	TFIDF	Ripper
Token wins	0	3	4	0
Entropy-split wins	13	10	6	13
Insignificants	8	8	11	8

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.

The statistical analysis comparing the Entropy split method with the tokenization verified this assessment. In fact, when we look at significant differences above the 95% level, the tokenization only outperformed the Entropy split method 7 times out of 86, while the Entropy split method outperformed the tokenization method 42 times, as is shown in Table 5.

We conclude this section by noting that Kohavi and Sahami [15] discuss a different discretization method that is very similar to the Entropy 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 Entropy 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 Entropy method to the C4.5 method. Fig. 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

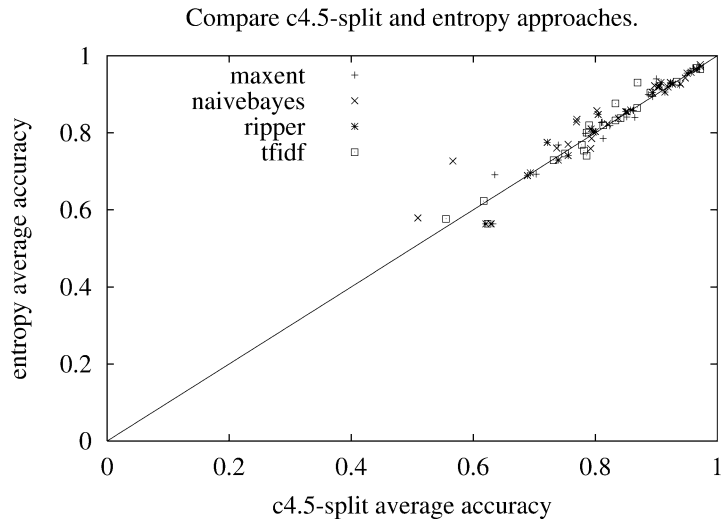

Fig. 7. Comparing the Entropy and C4.5 approaches.

Table 6
How often, across the four text-classifiers, did the Entropy split-method have a significant win (at or above the 95% confidence level) over using split-points found by C4.5, and vice versa

	MAXENT	Naive Bayes	TFIDF	Ripper
C4.5-split wins	2	1	2	0
Entropy-split wins	6	8	3	1
Insignificant	13	12	16	20

accuracy of the Entropy approach. A statistical analysis shows that the two methods are quite evenly matched, although Naive Bayes and MAXENT both seemed to favor the split-points created by the Entropy method and no classifier seemed to favor the split-points created by the C4.5 method. Table 6 shows the wins on each text-classifier.

4.3.2. Characteristics of splitpoint methods

Results and analyses so far have shown that all three splitpoint methods perform comparably. In order to get a better insight into how these methods work, we can look at the actual split points they generate. Because of their differences, we discuss the Density Method separately from the Entropy and C4.5 approaches. The Density Method does, for a particular learning algorithm, a geometric search through the maximum number of split-points (based on the numerical feature with the most candidate values). It would be helpful to understand how far along this exponential curve it normally needs to get before finding the best split-size. As the Density Method uses two different search variants, one based on unique values and one based on all values, it would be interesting to see whether one variant is used predominantly over the other.

The analysis of the Density Method for each the three text-classification algorithms used: Maximum Entropy, Naive Bayes, and TFIDF is shown in Tables 7, 8 and 9,

Table 7
Analysis of the Density Method using MAXENT

Data set	% unique	# feat	max bin	Number of bins		
				Min	Max	Mean ± stddev
arrhythmia	60%	272	8	0.0000	0.6250	0.1500 ± 0.2151
autos	100%	15	7	0.0000	0.0000	0.0000 ± 0.0000
bcancerw	50%	9	3	0.0000	1.6666	0.6666 ± 0.6831
cleveland	20%	13	7	0.0000	0.5714	0.2571 ± 0.2100
credit-app	90%	6	8	0.0000	0.5000	0.1625 ± 0.2019
cylinder-bands	50%	20	8	0.0000	0.3750	0.1125 ± 0.1179
diabetes	0%	8	9	0.2222	0.6666	0.4333 ± 0.1889
echocardiogram	100%	11	6	0.0000	0.0000	0.0000 ± 0.0000
glass	100%	9	7	0.0000	0.0000	0.0000 ± 0.0000
horse	70%	7	6	0.0000	0.8333	0.3500 ± 0.3452
hungarian	40%	13	7	0.0000	0.2857	0.1285 ± 0.1187
ionosphere	60%	34	8	0.0000	0.8750	0.2750 ± 0.2727
iris	100%	4	5	0.0000	0.0000	0.0000 ± 0.0000
liver	40%	6	6	0.0000	1.0000	0.3000 ± 0.3232
musk	80%	166	7	0.0000	1.0000	0.1571 ± 0.3286
new-thyroid	70%	5	6	0.0000	0.8333	0.3166 ± 0.2930
page-blocks	50%	10	10	0.0000	1.1000	0.6000 ± 0.2933
segmentation	40%	19	10	0.0000	0.8000	0.2800 ± 0.2638
sonar	100%	60	7	0.0000	0.0000	0.0000 ± 0.0000
sponge	100%	3	2	0.0000	0.0000	0.0000 ± 0.0000
vehicle	80%	18	8	0.0000	0.0000	0.0000 ± 0.0000
wine	100%	13	7	0.0000	0.0000	0.0000 ± 0.0000
Means	68.2%			0.0101	0.5060	0.19043
stddev	28.9%			0.0463	0.4663	0.19425

respectively. Each line in each table shows a statistic across the ten cross-validation runs for a particular data set. The first three columns shown after the data set name are the percentage of times that the best search was based on the unique values rather than all values, how many features in the data set were numerical in nature and what the maximum level of geometric search could be, where the level is the $\log_2$ of the largest number of values across all the numerical features in that data set. The last columns show the min, max, mean and standard deviation of how deep a search went with respect to the maximum depth. The last two lines of each table show the mean and standard deviation of key values across all the data sets.

Not surprisingly, looking at each table it is hard to find any correlation between any two features among the number of split points, the number of features, the maximum depth and the amount of the splits that were based on unique values. Some interesting patterns that do show up are that TFIDF in general created more split points than any of the other methods, it had higher variance in the amount of split points that it did create. While other methods seemed to favor the splits based on unique values, Maximum Entropy seemed to have less of a bias.

Analyzing the Entropy and C4.5 splitting methods require a different methodology as they work differently than the Density Method. In both cases, the methods find out the

Table 8
Analysis of the Density Method using Naive Bayes

Data set	% unique	# feat	max bin	Number of bins		
				Min	Max	Mean ± stddev
arrhythmia	70%	272	8	0.0000	0.8750	0.1625±0.2798
autos	100%	15	7	0.0000	0.0000	0.0000±0.0000
bcancerw	50%	9	3	0.0000	2.6666	0.8666±0.9568
cleveland	60%	13	7	0.0000	0.7142	0.2571±0.2843
credit-app	60%	6	8	0.0000	0.3750	0.0375±0.1125
cylinder-bands	70%	20	8	0.0000	0.6250	0.1250±0.2016
diabetes	80%	8	9	0.0000	0.5555	0.2555±0.2280
echocardiogram	100%	11	6	0.0000	0.0000	0.0000±0.0000
glass	100%	9	7	0.0000	0.0000	0.0000±0.0000
horse	90%	7	6	0.0000	0.8333	0.3833±0.3578
hungarian	50%	13	7	0.0000	0.5714	0.1285±0.1623
ionosphere	20%	34	8	0.2500	0.8750	0.5875±0.1941
iris	100%	4	5	0.0000	0.0000	0.0000±0.0000
liver	80%	6	6	0.0000	1.0000	0.5333±0.3399
musk	100%	166	7	0.0000	0.0000	0.0000±0.0000
new-thyroid	50%	5	6	0.0000	0.8333	0.3666±0.2963
page-blocks	60%	10	10	0.0000	0.8000	0.4500±0.2941
segmentation	60%	19	10	0.0000	0.9000	0.3600±0.3441
sonar	100%	60	7	0.0000	0.0000	0.0000±0.0000
sponge	100%	3	2	0.0000	0.0000	0.0000±0.0000
vehicle	70%	18	8	0.0000	0.0000	0.0000±0.0000
wine	100%	13	7	0.0000	0.0000	0.0000±0.0000
Means	75.9%			0.0114	0.5284	0.2052
stddev	22.5%			0.0521	0.6036	0.2393

best split points *per numerical feature*, regardless of the learning algorithm used. This intuitively leads us to ask the following questions:

(1) For each feature, what is the ratio of number of split-points found with respect to the maximum number of split-points possible?
(2) Is there any correlation between the number of split-points generated and either the possible maximum or the number of numerical features?
(3) How many of the features are not used at all? (e.g., no split-points were found)
(4) How do the two methods compare with respect to the previous answers.

Tables 10 and 11 try to answer these questions. The first columns of the tables, following the data set name, are again the number of numerical features for that data set, followed by the minimum and maximum number of values across all the numerical features. The next three columns show the range of splits that were actually found by the splitting method. The values are normalized for each feature. The last three columns show the range of numerical features that were *not* used by the splitting method, due to no points being found. These values have been normalized with respect to the number of numerical features in that data set.

Table 9
Analysis of the Density Method using TFIDF

Data set	% unique	# feat	max bin	Number of bins		
				Min	Max	Mean ± stddev
arrhythmia	20%	272	8	0.3750	0.5000	0.4750±0.0500
autos	100%	15	7	0.0000	0.0000	0.0000±0.0000
bcancerw	50%	9	3	0.3333	1.0000	0.7000±0.3145
cleveland	70%	13	7	0.0000	0.2857	0.1571±0.1187
credit-app	80%	6	8	0.1250	0.1250	0.1250±0.0000
cylinder-bands	50%	20	8	0.1250	0.6250	0.2625±0.1420
diabetes	80%	8	9	0.0000	0.6666	0.2222±0.2383
echocardiogram	100%	11	6	0.0000	0.0000	0.0000±0.0000
glass	100%	9	7	0.0000	0.0000	0.0000±0.0000
horse	40%	7	6	0.1666	0.6666	0.4000±0.1856
hungarian	100%	13	7	0.0000	0.2857	0.1857±0.0915
ionosphere	60%	34	8	0.1250	0.7500	0.5125±0.2050
iris	100%	4	5	0.0000	0.0000	0.0000±0.0000
liver	70%	6	6	0.0000	1.0000	0.4000±0.3091
musk	90%	166	7	0.0000	0.4285	0.1857±0.1696
new-thyroid	90%	5	6	0.0000	0.6666	0.3666±0.1795
page-blocks	100%	10	10	0.1000	0.2000	0.1600±0.0490
segmentation	100%	19	10	0.4000	0.6000	0.5400±0.0800
sonar	100%	60	7	0.0000	0.0000	0.0000±0.0000
sponge	100%	3	2	0.0000	0.0000	0.0000±0.0000
vehicle	80%	18	8	0.1250	0.3750	0.3250±0.0829
wine	100%	13	7	0.0000	0.0000	0.0000±0.0000
Means	80.9%			0.0852	0.3716	0.2281
stddev	23.1%			0.1265	0.3304	0.2066

As with the Density Method, there did not seem to be any correlation between the number of split points found and any other characteristic of the data set. However, one interesting pattern did seem to emerge when looking at the number of features that were not used: Both methods were able to disregard at least one numerical feature in the majority of cases. Further, when comparing the two methods, two observations can immediately be made: Entropy generates far fewer split-points on average and disregards a much larger amount of features than C4.5. While doing so, it still manages to perform comparably, and qualitatively outperform C4.5. Finally, it is interesting to note that the sponge data set had only one numerical feature, and it was ignored by both Entropy and C4.5.

5. Case study: Mixed-mode data set

The previous section demonstrated the credible performance of our approach on purely numerical data sets. We now turn to the mixed-mode example that we mentioned in the Introduction, classifying email using both numerical and text features. This allows us to evaluate whether adding numerical features to what we had previously treated as a text-only data set [10] yields any gain in performance. To explore this issue, we consider not only

Table 10
Analysis of the Entropy Splitting Method

Data set	# feat	min / max	Number of split-points			Number of no-splits		
		binsize	Min	Max	Mean ± stddev	Min	Max	Mean ± stddev
arrhythmia	272	1/384	0.000	0.114	0.008±0.018	0.710	0.761	0.737±0.015
autos	15	23/185	0.000	0.026	0.002±0.008	0.400	0.400	0.400±0.000
bcancerw	9	9/11	0.111	0.240	0.200±0.035	0.000	0.000	0.000±0.000
cleveland	13	2/152	0.000	0.500	0.161±0.184	0.308	0.308	0.308±0.000
credit-app	6	23/350	0.001	0.073	0.017±0.026	0.000	0.167	0.100±0.082
cylinder-bands	20	4/296	0.000	0.031	0.005±0.008	0.700	0.900	0.825±0.051
diabetes	8	17/517	0.000	0.058	0.015±0.018	0.250	0.375	0.263±0.038
echocardiogram	11	2/106	0.000	0.333	0.034±0.095	0.455	0.818	0.664±0.115
glass	9	32/178	0.006	0.032	0.016±0.011	0.000	0.000	0.000±0.000
horse	7	25/89	0.000	0.042	0.019±0.016	0.286	0.286	0.286±0.000
hungarian	13	2/154	0.000	0.500	0.115±0.158	0.462	0.539	0.523±0.031
ionosphere	34	1/281	0.000	0.500	0.027±0.083	0.029	0.118	0.059±0.023
iris	4	22/43	0.043	0.091	0.062±0.018	0.000	0.000	0.000±0.000
liver	6	16/94	0.000	0.006	0.001±0.002	0.833	1.000	0.900±0.082
musk	166	29/217	0.000	0.030	0.007±0.005	0.235	0.325	0.281±0.031
new-thyroid	5	47/100	0.023	0.073	0.042±0.018	0.000	0.000	0.000±0.000
page-blocks	10	104/1718	0.004	0.048	0.010±0.013	0.000	0.000	0.000±0.000
segmentation	19	1/1937	0.000	0.333	0.039±0.079	0.053	0.105	0.068±0.024
sonar	60	109/208	0.000	0.008	0.002±0.002	0.617	0.700	0.653±0.023
sponge	3	4/5	0.000	0.000	0.000±0.000	1.000	1.000	1.000±0.000
vehicle	18	13/424	0.012	0.308	0.072±0.068	0.000	0.000	0.000±0.000
wine	13	39/133	0.012	0.026	0.018±0.004	0.000	0.000	0.000±0.000
Means			0.010	0.153	0.040	0.289	0.355	0.321
stddev			0.024	0.172	0.053	0.279	0.265	0.286

the text of each email message, but also any numerical features available [10]. Continuing with our earlier work using information-retrieval-based classification methods, we turned to the bags-of-tokens approach to inject these numerical features into learning. This data concerns the use of two-way pagers by three users for over 14 months, each user choosing whether or not to read a new email message via the pager. Keeping track of which emails the users chose to read, we were able to label each email as either "Forward", that it was desirable to read this email on the pager, or "NotForward", that it was not desirable to forward this email. The historical use of the pager for each user was treated as a separate dataset, giving us three unique datasets. We use 23 features for this study, 16 of them being numerical. Tables 12 and 13 show the features that were used in this experiment, while Table 14 shows the size and ratio of messages read by each respective user.

5.1. Evaluation methodology

To evaluate our methods on this data we randomly pick 500 test instances, keeping the ratio of class-distributions. For each test email, we learn a model based on all emails whose timestamp is less than the test email, then use that learned model to predict whether to forward that particular test email. In order to guarantee that a training set is sufficiently

Table 11
Analysis of the C4.5 Splitting Method

Data set	# feat	min / max	Number of split-points			Number of no-splits		
		binsize	Min	Max	Mean ± stddev	Min	Max	Mean ± stddev
arrhythmia	272	1/384	0.000	0.507	0.160±0.129	0.287	0.338	0.313±0.013
autos	15	23/185	0.000	0.113	0.015±0.041	0.267	0.333	0.300±0.033
bcancerw	9	9/11	0.100	0.280	0.161±0.073	0.000	0.000	0.000±0.000
cleveland	13	2/152	0.000	0.500	0.196±0.175	0.154	0.231	0.185±0.038
credit-app	6	23/350	0.003	0.061	0.015±0.021	0.000	0.000	0.000±0.000
cylinder-bands	20	4/296	0.000	0.117	0.022±0.034	0.550	0.700	0.655±0.061
diabetes	8	17/517	0.000	0.059	0.014±0.019	0.250	0.500	0.425±0.083
echocardiogram	11	2/106	0.000	0.333	0.063±0.102	0.364	0.546	0.436±0.055
glass	9	32/178	0.000	0.040	0.014±0.012	0.111	0.222	0.122±0.033
horse	7	25/89	0.000	0.081	0.018±0.029	0.571	0.714	0.686±0.057
hungarian	13	2/154	0.000	0.333	0.118±0.136	0.385	0.462	0.431±0.038
ionosphere	34	1/281	0.000	0.500	0.029±0.082	0.029	0.029	0.029±0.000
iris	4	22/43	0.047	0.104	0.086±0.024	0.000	0.000	0.000±0.000
liver	6	16/94	0.000	0.131	0.028±0.047	0.500	0.667	0.617±0.076
musk	166	29/217	0.000	0.122	0.014±0.015	0.307	0.361	0.340±0.017
new-thyroid	5	47/100	0.022	0.089	0.048±0.025	0.000	0.000	0.000±0.000
page-blocks	10	104/1718	0.007	0.196	0.034±0.055	0.000	0.000	0.000±0.000
segmentation	19	1/1937	0.000	0.709	0.189±0.201	0.053	0.053	0.053±0.000
sonar	60	109/208	0.000	0.007	0.003±0.002	0.417	0.533	0.468±0.037
sponge	3	4/5	0.000	0.000	0.000±0.000	1.000	1.000	1.000±0.000
vehicle	18	13/424	0.039	0.510	0.280±0.139	0.000	0.000	0.000±0.000
wine	13	39/133	0.020	0.095	0.039±0.021	0.000	0.000	0.000±0.000
Means			0.011	0.222	0.070	0.238	0.304	0.276
stddev			0.023	0.199	0.077	0.228	0.216	0.219

Table 12
Numerical Features used in the Email data set

minutes	minutes since midnight
length	length in bytes
textlength	length in bytes of textual parts
pager_cmd	seconds since pager last used
recv	seconds since last mail from sender
recv_subj	seconds since last mail from sender on subject
sent	seconds since last mail to sender
sent_subj	seconds since last mail to sender on subject
logged_in	seconds since user was last logged in online
last_cmd	seconds since last online command
mail_read	seconds since last time mail was read online
num_recpt	number of recipients (to and cc)
num_qstn	number of question marks in message
num_excl	number of exclamation marks in message
num_pro	number of pronouns in message
num_upper	number of all upper-case words in message

Table 13
Textual Features used in the Email data set

dayname	name of day
daytype	type of day (weekday or weekend)
from	tokenized list of the from-field[1]
to	tokenized list of the to-field[1]
tocc	combined tokenized list of the to- and cc-field[1]
subject	words in subject
body	words in body

1) email addresses of the form:
`user@machine.domain`
were split into:
`user user@domain user@dmachine.domain domain machine.domain`

Table 14
Email data sets from three user's extended use of an EmailValet

	Size	Ratio of messages read on pager
AD	6048	20.11%
HH	23673	22.12%
SM	3987	15.98%

large to learn a discriminatory model, we constrain all test emails to have at least 500 emails in their respective training sets.

Each data set was represented in one of three ways for our experiments:

- **txt**: The original feature encoding—limited to only the textual features, for use with our five text-classifiers.
- **num**: The original feature encoding—using both text and numbers—for use with SVM and Ripper.
- **entropy**: The bag-of-words encoding generated by the Entropy method, for use with the five text-classifications.
- **human**: The bag-of-words encoding using hand-generated land-mark values, for use with the five text-classifications.

The first of these represents our baseline, using a machine learning method designed for text classification. The next one is used to gauge whether adding numerical features and using methods that can handle set-valued features as well as numerical features will improve performance. Notice that C4.5 has been dropped as a method as C4.5 does not deal easily with set-valued features. The last two use the new approach presented in this paper, one using the Entropy splitting method and one using hand-picked splitting values to compare how well a human would perform in picking landmark values. We chose to

use only the Entropy method here due to its faster performance and fewer splitting points. Table 15 shows the values that were manually picked.

SVM was actually run twice for each of the four split methods given above. The first run was done using a binary vector-space model, the second using a TFIDF vector-space model. In both models, each numerical feature was normalized based on the minimum and maximum values of the feature in the training set. In the binary model, henceforth referred to as *B-SVM*, each word was represented as either a 1 or a 0, based on whether that particular word was present in the file. The TFIDF model, henceforth referred to as *T-SVM*, represented words as a TFIDF value, using the current weighting schemes used by the information retrieval community [29]. These weighting schemes are defined in Table 16.

Table 15
Manually chosen splitpoints for email data

Feature	Split-points
minutes	Every full and half hour of the day
length **textlength**	512, 1024, 2048, 4096, 8192 and 16384 bytes
pager_cmd **recv** **recv_subj** **sent** **sent_subj** **logged_in** **last_cmd** **mail_read**	1, 5, 10, 15 and 30 minutes 1, 2, 4, 8, 12, 16 and 24 hours
num_recpt **num_qstn** **num_excl** **num_pro** **num_upper**	0, 1, 2, 3, 4, 5, 10 and 20

Table 16
TFIDF Encoding Scheme for T-SVM Classifier

tf factor:
$d = 1 + \ln(1 + \ln(TF))$ [0 if $tf = 0$]
idf factor:
$t = \log\left(\frac{(N+1)}{df}\right)$
pivoted byte length normalization factor:
$b = \frac{1}{(0.8+0.2*(\frac{doc\_length}{avg\_doc\_length}))}$
tf = term frequency
N = number of documents in (training) collection
df = number of documents containing word
dnb weight $= d * b$
dtb weight $= d * t * b$
dtn weight $= d * t$

We use the *dtb* weighting scheme for both the training and the test sets, as the pivoted byte length normalization has been shown to improve performance [29].

Initially SVM was also run using both linear and quadratic kernels. The linear kernels outperformed the quadratic kernels in most runs, and were rarely beaten. For this reason, we only report the performance using linear kernels.

5.2. Results

The first question we ask here is key: Does adding numerical features using *any* method improve performance to using only the text of the emails? To answer this, we performed a straight comparison across all methods, using the splitting methods mentioned. Tables 17, 18, and 19 show the results of these first comparisons. Because it is hard to gauge which type of measure to use in this type of data set, we show four types of measures used often, that of error, precision, recall and the F1-measure. Precision is defined as: of all the messages that were predicted to be forwarded, what percentage of those were truly messages that should have been forwarded. Recall is defined as: of all messages that should have been forwarded, what percentage of these were predicted to be forwarded. The F1 measure is then defined as $2 * \text{Recall} * \text{Precision}/(\text{Recall} + \text{Precision})$.

Running SVM$^{\text{light}}$ with its default parameters yielded a surprisingly bad performance with respect to recall as well as the F1-measure, leading us to use a cost-factor of 5 between

Table 17
Comparison of Methods on the AD data set

Classifier	Method	Error	Precision	Recall	F1
B-SVM	Text	21.000	49.708	81.731	0.618
B-SVM	Numeric	22.000	48.352	84.615	0.615
B-SVM	Entropy	20.800	50.000	81.731	0.620
B-SVM	Human	21.000	49.708	81.731	0.618
T-SVM	Text	22.609	46.795	77.660	0.584
T-SVM	Numeric	26.000	38.732	76.389	0.514
T-SVM	Entropy	22.560	46.753	76.596	0.581
T-SVM	Human	23.011	46.250	77.895	0.580
Ripper	Text	36.419	33.040	72.115	0.453
Ripper	Numeric	36.419	33.040	72.115	0.453
Ripper	Entropy	31.187	38.140	78.846	0.514
Ripper	Human	35.614	35.102	82.692	0.493
MAXENT	Text	19.400	53.608	50.000	0.517
MAXENT	Entropy	17.800	57.426	55.769	0.566
MAXENT	Human	18.600	56.627	45.192	0.503
Naive Bayes	Text	21.400	49.068	75.962	0.596
Naive Bayes	Entropy	21.400	49.057	75.000	0.593
Naive Bayes	Human	24.400	45.455	86.538	0.596
TFIDF	Text	23.447	45.977	77.670	0.578
TFIDF	Entropy	23.800	45.562	74.038	0.564
TFIDF	Human	25.000	44.560	82.692	0.579

Table 18
Comparison of Methods on the HH data set

Classifier	Method	Error	Precision	Recall	F1
B-SVM	Text	34.800	34.959	86.000	0.497
B-SVM	Numeric	32.465	36.000	81.818	0.500
B-SVM	Entropy	34.600	35.102	86.000	0.499
B-SVM	Human	34.800	34.959	86.000	0.497
T-SVM	Text	31.600	36.058	75.000	0.487
T-SVM	Numeric	34.000	34.234	76.000	0.472
T-SVM	Entropy	31.600	36.058	75.000	0.487
T-SVM	Human	31.600	36.058	75.000	0.487
Ripper	Text	35.849	36.410	88.750	0.516
Ripper	Numeric	48.200	26.421	79.000	0.396
Ripper	Entropy	38.353	33.210	90.000	0.485
Ripper	Human	38.600	32.967	90.000	0.483
MAXENT	Text	18.437	55.738	34.343	0.425
MAXENT	Entropy	18.145	56.579	43.000	0.489
MAXENT	Human	18.200	56.716	38.000	0.455
Naive Bayes	Text	34.800	35.200	88.000	0.503
Naive Bayes	Entropy	30.364	37.313	75.758	0.500
Naive Bayes	Human	45.800	28.571	86.000	0.429
TFIDF	Text	34.400	35.124	85.000	0.497
TFIDF	Entropy	33.737	34.703	76.000	0.476
TFIDF	Human	41.400	30.545	84.000	0.448

misclassifying messages that should be forwarded versus those that should not. The factor of 5 was chosen deliberately to closely resemble the difference in class-ratios on all three data sets. We realize this does make use of test-data for parameter selection, but have no reason to believe this factor would in any way favor our methods. This change lowered the precision (by 1–5%) and error (by 10–15%), but increased the recall and F1-measures dramatically (by as much as 71% for recall and 0.41 for F1). The same problem was encountered with Ripper, where changing the cost-ratio to be similar to that of the class-ratio had the same large effect, though not quite as dramatically as for the SVM runs. Because of this, we show only the latter runs in our results.

The immediate qualitative observations that can be made from these three tables are: In the majority of the cases, using the Entropy method outperforms the other methods, and when it looses it is generally not by much. Interestingly, the human-generated split points generally did not perform very well, nor did the pure numerical methods. Comparing learners, it is obvious that using Maximum Entropy, while slow, is the clear winner on the error, while it did not perform so well on recall. In fact, SVM, Naive Bayes and TFIDF all did quite well on recall, with the tradeoff of worse precision.

Performing statistical significance testing, we compared the performance of each learner with numerical features added to the performance of the same learner using only textual features. We calculated the significance in difference between each of the four measurements given. There were a few cases where there was any significant differences

Table 19
Comparison of Methods on the SM data set

Classifier	Method	Error	Precision	Recall	F1
B-SVM	Text	30.200	29.508	71.053	0.417
B-SVM	Numeric	29.600	29.775	69.737	0.417
B-SVM	Entropy	30.200	29.121	70.667	0.412
B-SVM	Human	30.200	29.508	71.053	0.417
T-SVM	Text	26.600	30.070	56.579	0.393
T-SVM	Numeric	27.200	30.769	63.158	0.414
T-SVM	Entropy	26.600	29.577	56.000	0.387
T-SVM	Human	26.600	30.070	56.579	0.393
Ripper	Text	38.200	25.108	76.316	0.378
Ripper	Numeric	38.200	25.108	76.316	0.378
Ripper	Entropy	39.200	25.207	80.263	0.384
Ripper	Human	27.400	32.370	73.684	0.450
MAXENT	Text	19.400	24.390	13.158	0.171
MAXENT	Entropy	20.200	29.508	23.684	0.263
MAXENT	Human	15.200	50.000	19.737	0.283
Naive Bayes	Text	43.000	23.574	81.579	0.366
Naive Bayes	Entropy	42.600	23.954	82.895	0.372
Naive Bayes	Human	44.000	23.723	85.526	0.371
TFIDF	Text	44.400	22.963	81.579	0.358
TFIDF	Entropy	43.800	23.420	82.895	0.365
TFIDF	Human	43.200	23.684	82.895	0.368

(above 95%) between the values. In the AD data set, in only 2 of these runs did the pure textual data sets win over the numerically enhanced data sets, in the HH data set, most of the significant differences were in favor of the purely textual data set, and in the SM data set all significant differences were found for the numerically augmented runs. One interesting consistent pattern across all of these tests is that the Maximum Entropy method augmented with the numerical features using the Entropy splitting method has significant improvement above 95% in recall, and dropping to the 90% confidence level this also holds true for the F1-measure.

6. 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 methods. 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.

Our original motivation for performing this work was to broaden the class of learning methods that can be applied to mixed-mode data. Our understanding of numerical classification methods has been helped by the availability of a large number of benchmark classification problems. Obtaining similar insights into problems with mixed-mode data would similarly be helped by the availability of mixed-mode benchmark problems, a task we are currently performing. Our method for converting numbers to bags of tokens also opens up new questions for feature generation and selection, one of the key sources of improvement in applied classifier learning. Finally, 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.

Acknowledgements

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

References

[1] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, CA, 1993.
[2] W.W. Cohen, Fast effective rule induction, in: Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, CA, 1995, pp. 115–123.
[3] F. Sebastiani, Machine learning in automated text categorization, ACM Comput. Surveys 34 (2002) 1–47, http://faure.iei.pi.cnr.it/~fabrizio/Publications/ACMCS02.pdf.
[4] Y. Yang, An evaluation of statistical approaches to text categorization, Inform. Retrieval 1 (1/2) (1999) 67–88.
[5] W.W. Cohen, Learning trees and rules with set-valued features, in: Proceedings AAAI-96, Portland, OR, 1996, pp. 709–716.
[6] G. Salton, Developments in automatic text retrieval, Science 253 (1991) 974–979.
[7] Y. Yang, C. Chute, An example-based mapping method for text classification and retrieval, ACM Trans. Inform. Syst. 12 (3) (1994) 252–277.
[8] J. Rocchio, Relevance feedback in information retrieval, in: Salton (Ed.), The SMART Retrieval System: Experiments in Automatic Document Processing, Prentice-Hall, Englewood Cliffs, NJ, 1971, pp. 313–323, Chapter 14.
[9] R. Schapire, Y. Singer, A. Singhal, Boosting and Rocchio applied to text filtering, in: Proceedings of ACM SIGIR, Melbourne, Australia, 1998, pp. 215–223.
[10] S.A. Macskassy, A.A. Dayanik, H. Hirsh, Emailvalet: Learning user preferences for wireless email, 1999.
[11] J. Catlett, On changing continuous attributes into ordered discrete attributes, in: Y. Kodratoff (Ed.), Proceedings of the European Working Session on Learning, Springer-Verlag, Berlin, 1991, pp. 164–178.
[12] R. Kerber, Discretization of numeric attributes, in: Proceedings AAAI-92, San Jose, CA, AAAI Press/MIT Press, Menlo Park, CA, 1992, pp. 123–128.
[13] U.M. Fayyad, K.B. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, in: Proceedings IJCAI-93, Chambéry, France, Morgan Kaufmann, San Mateo, CA, 1993, pp. 1022–1027.
[14] K. Dougherty, R. Kohavi, M. Sahami, Supervised and unsupervised discretization of continuous features, in: Proceedings of the 12th International Conference on Machine Learning, Tahoe City, CA, Morgan Kaufmann, San Mateo, CA, 1995, pp. 194–202.

[15] R. Kohavi, M. Sahami, Error-based and entropy-based discretization of continuous features, in: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, AAAI Press/MIT Press, Menlo Park, CA, 1996, pp. 114–119.
[16] E. Frank, I.H. Witten, Making better use of global discretization, in: Proceedings of the 17th International Conference on Machine Learning, Bled, Slovenia, 1999, pp. 115–123.
[17] S.I. Gallant, Neural Network Learning and Expert Systems, MIT Press, Cambridge, MA, 1993.
[18] J. Yang, V. Honavar, A simple randomized quantization algorithm for neural network pattern classifiers, in: Proceedings of the World Congress on Neural Networks, San Diego, CA, 1996, pp. 223–228.
[19] I. Rissanen, Minimum description length principle, Encyclopedia Statist. Sci. 5 (1987) 523–527.
[20] T. Joachims, A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization, in: Proceedings of the Fourteenth International Conference on Machine Learning, Nashville, TN, 1997, pp. 143–151.
[21] P. Domingos, M. Pazzani, Beyond independence: Conditions for the optimality of the simple Bayesian classifier, in: Proceedings of the 13th International Conference on Machine Learning, Bari, Italy, 1996, pp. 105–112.
[22] T. Mitchell, Machine Learning, McGraw-Hill, New York, 1997.
[23] K. Nigam, J. Lafferty, A. McCallum, Using maximum entropy for text classification, 1999.
[24] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag, Berlin, 1995.
[25] A.K. McCallum, Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering, 1996, http://www.cs.cmu.edu/~mccallum/bow.
[26] T. Joachims, Advances in kernel methods—Support vector learning, 1999.
[27] C.L. Blake, C.J. Merz, UCI repository of machine learning databases, 1998.
[28] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings IJCAI-95, Montreal, QB, Morgan Kaufmann, San Francisco, CA, 1995, pp. 1137–1143.
[29] A. Singhal, J. Choi, D. Hindle, D. Lewis, F. Pereira, AT&T at TREC-7, in: E. Voorhees, D. Harman (Eds.), Proceedings of the Seventh Text REtrieval Conference (TREC-7), Gaithersburg, MD, 1998, pp. 186–198.