Text Bundling: Statistics-Based Data Reduction

Lawrence Shih   KAI@AI.MIT.EDU
Jason D. M. Rennie   JRENNIE@AI.MIT.EDU
Yu-Han Chang   YCHANG@AI.MIT.EDU
David R. Karger   KARGER@LCS.MIT.EDU
Artificial Intelligence Laboratory; Massachusetts Institute of Technology; Cambridge, MA 02139

Abstract

As text corpora become larger, tradeoffs between speed and accuracy become critical: slow but accurate methods may not complete in a practical amount of time. In order to make the training data a manageable size, a data reduction technique may be necessary. Subsampling, for example, speeds up a classifier by randomly removing training points. In this paper, we describe an alternate method for reducing the number of training points by combining training points such that important statistical information is retained. Our algorithm keeps the same statistics that fast, linear-time text algorithms like Rocchio and Naive Bayes use. We provide empirical results that show our data reduction technique compares favorably to three other data reduction techniques on four standard text corpora.

1. Introduction

There is a great need to find fast, effective algorithms for text classification. A KDD panel headed by Domingos (2002) discussed the tradeoff of speed and accuracy in the context of very large (e.g. one-million record) databases. Most highly accurate text classifiers take a disproportionately large time to handle a large number of training examples. These classifiers may become impractical when faced with large data sets, like the Ohsumed data set with more than 170,000 training examples (Hersh et al., 1994).

The standard way to deal with a million point data set is to reduce the size of the data to a more manageable amount. Subsampling, for example, reduces the data set by randomly removes training points. Feature selection, another data reduction technique, tries to retain only the best features of a data set.

One way to understand a large body of data is to summarize them with statistics, which are functions over the data. Let $\vec{x} = \{x_1, x_2, \ldots, x_n\} \in \mathbb{R}^n$ be a set of data. Then a statistic, $s(\vec{x})$, is a function that maps the data to a single value in $\mathbb{R}$. For example the mean statistic is simply the average value of the data points.

Subsampling does not preserve the entire set of data, rather it preserves all statistics on a random subset of the data. In contrast, we propose to preserve certain statistics of *all* the data. For example, in text we preserve the mean statistic. The mean statistic has proven useful in Rocchio (Rocchio, 1971)—a standard text-classification algorithm—and we believe the mean statistic will be useful for other classifiers. Our "text bundling" algorithm fully preserves the mean statistics of all the data. Text bundling generates a new, smaller training set by averaging together small groups of points. This allows bundling to preserve some of the overall distribution information beyond simply the mean of the data. A classifier using this reduced data set should be able to learn a decision boundary that is at least as good as Rocchio's, since it can use additional information about the data.

We believe this focus on mean statistics will prove to be a superior data reduction technique than subsampling or feature selection. For example, in a binary problem where we reduce the data to one point per class or "maximal bundling," we are left with one "average" point in each class. In this average point, the count for word i is the average number of times word i appeared in documents of the class. When these points are given to most learning algorithms, the result will be a decision boundary equivalent to the Rocchio classifier: the perpendicular bisector of the two average points. Thus, due to our focus on statistics, even maximal bundling results in a good classifier. On the other hand, reducing to a single point per class via subsampling yields a single sample document. This gives almost no information about the nature of the

class. Similarly, text classification with the single best word-feature is often ineffective.

We provide empirical evidence that our bundling technique works well by comparing it with three other data reduction techniques: bagging, feature selection, and subsampling.

2. Related Work

The speed of machine learning algorithms is important in making learning practical and widely used. Google's results are good; but few would use it were it to take ten minutes to return results. In some fields like text, more training data usually corresponds to higher classification accuracy. In some problems, it is easy to amass a training set numbering in the hundreds of thousands or millions. The best text classification algorithms are "super-linear"—each additional training point takes more time to train on than the previous point. When the number of examples is very large, such algorithms effectively do not complete. Those algorithms run faster when given a smaller set of inputs. Our focus is on finding a better way to select the set of data points that we hand to such an algorithm.

One simple method of speeding up any classifier is to reduce the number of training points. A common approach is subsampling, which retains a random subset of the original training data. Subsampling has the advantage of being fast and easy implement. Our belief is that for text classification there are better ways to reduce the data set.

Given a classification algorithm that is super-linear, another potential solution is bagging (Breiman, 1996). Bagging partitions the original data set and learns a classifier on each partition. A test document is labeled by a majority vote of the classifiers. This technique makes training faster (since each classifier has fewer examples), but slows down testing since it evaluates multiple classifiers for each test example. Hence the overall training and testing time does not always decrease.

We can also speed up classifiers by reducing the number of features. Feature selection has been the focus of much work (Yang & Pedersen, 1997; Mladenic, 1998). Note that all classifiers already perform a type of feature selection: if the classifier sees a feature as irrelevant, it simply ignores that feature by zeroing out a weight corresponding to that feature. Thus, an important contribution of feature selection is to have a fast pre-processing step that reduces the overall training time. It is unusual, however, to see empirical evidence comparing the empirical time reduction of feature selection with the resulting loss in accuracy.

2.1. The Rocchio Algorithm

Several times in this paper, we mention the Rocchio classification algorithm (Rocchio, 1971). For completeness we describe it in full detail here. Consider a binary classification problem. Simply put, Rocchio selects a decision boundary (plane) that is perpendicular to a vector connecting two class centroids. Let $\{\vec{x}_{11}, \ldots, \vec{x}_{1l_1}\}$ and $\{\vec{x}_{21}, \ldots, \vec{x}_{2l_2}\}$ be sets of training data for the positive and negative classes, respectively. Let $\vec{c}_1 = \frac{1}{l_1} \sum_i \vec{x}_{1i}$ and $\vec{c}_2 = \frac{1}{l_2} \sum_i \vec{x}_{2i}$ be the centroids for the positive and negative classes, respectively. Then, we define the Rocchio score of an example as

$$\text{RocchioScore}(\vec{x}) = \vec{x} \cdot (\vec{c}_1 - \vec{c}_2). \qquad (1)$$

One selects a threshold value, b, which may be used to make the decision boundary closer to the positive or negative class centroid. Then, an example is labeled according to the sign of the score minus the threshold value,

$$l_{\text{Rocchio}}(\vec{x}) = \text{sign}\left(\text{RocchioScore}(\vec{x}) - b\right). \qquad (2)$$

3. The Bundling Algorithm

We can think of the tradeoffs between speed and accuracy in an information sense: the less raw information we retain, generally the faster the classifier will run and the less accurate the results. Each data reduction technique operates by retaining some information and removing other information. By carefully selecting our statistics for a domain, we can optimize the information we retain.

Bundling preserves a set of k user-chosen statistics, $\vec{s} = (s_1, \ldots, s_k)$, where s_i is a function that maps a set of data to a single value. Bundling imposes a global constraint as follows.

global constraint Let $\vec{x}$ be a set of data. Let $\vec{x}'$ be a reduced set of training data, the "bundled" set. Bundling requires that $s_i(\vec{x}) = s_i(\vec{x}') \; \forall i$.

There are many possible reduced data sets, $\vec{x}'$, that can satisfy this constraint. But, we don't *only* want to preserve the global statistics. We also want to preserve additional information about the distribution. To get a reduced data set that satisfies the global constraint, we could generate several random points and then choose the remaining points to preserve the statistics. This does not retain any information about

our data except for the chosen statistics. We can retain some of the information besides the statistics by grouping together sets of points and preserving the statistics locally:

local constraint Assume that the data points, $\vec{x}$, and the reduced data, $\vec{x}'$ are partitioned into the same number of sets. Let $\vec{y}_{(j)}$ be the data points from the jth partition of $\vec{x}$. Let $\vec{y}'_{(j)}$ be the data points from the jth partition of $\vec{x}'$. Bundling requires that $s_i(\vec{y}_{(j)}) = s_i(\vec{y}'_{(j)}) \ \forall i, j$.

The bundling algorithm's local constraint is to maintain the same statistics between subsets of the training data.

The focus on statistics also usually implies that the bundled data will not have any examples in common with the original data. This is a necessary consequence of our wish to fully preserve certain statistics rather than the precise examples in the original training set. The bundling algorithm ensures that certain global statistics are maintained, while also maintaining a relationship between certain partitions of the data in the original and bundled training sets.

In the following section we will discuss how to implement bundling for text, where we preserve mean statistics.

3.1. Text Bundling

The first step in bundling is to select a statistic or statistics to preserve. For text, we choose the mean statistic of each feature. Rocchio and Multinomial Naive Bayes perform classification using only the mean statistics of the data. As these classifiers perform well, the mean statistics are very important for text classification.

We assume that there are a set of training documents for each class. We apply bundling separately to each class, so we will only discuss what needs to be done for a single class. Let $D = \{\vec{d}_1, \ldots, \vec{d}_n\}$ be a set of documents. It is standard to use the "bag-of-words" representation (McCallum & Nigam, 1998), where each word is a feature and a document is represented as a vector of word counts. We write $\vec{d}_i = \{d_{i1}, \ldots, d_{iV}\}$, where the second subscript indexes the words and V is the size of the vocabulary. We use the mean statistic for each feature as our text statistics. So, we define the jth statistic as

$$s_j(D) = \frac{1}{n} \sum_{i=1}^{n} d_{ij}. \tag{3}$$

procedure Randomized Bundling
1: Let n be the number of documents.
2: Let m be the chosen partition size (we assume n/m is an integer).
3: Let $s_1, \ldots, s_V$ be the mean statistics as defined in equation 3.
4: Randomly partition the set of documents D into m equal-sized partitions $P_1, \ldots, P_m$.
5: Compute $d'_{ij} = s_j(P_i)$, where $\vec{d}'_i$ is the ith reduced data point and d'_{ij} is the count of word j.
6: $D' = \{\vec{d}'_1, \ldots, \vec{d}'_m\}$ is the reduced data set.

As an example, consider what happens when we do "maximal bundling." We reduce to a single point with the mean statistics; the jth component of the single point is $s_j(D)$. Using a linear classifier on this "maximal bundling" will result in a decision boundary equivalent to Rocchio's decision boundary. Both decision boundaries are the perpendicular bisector of the sample means of the two classes. Thus, bundling degrades well; even when reducing to a single point, we expect the training set to provide enough information to create an effective text classifier. Other data reduction techniques do not degrade as nicely.

Bundling gives us the power to trade-off between the advantages of a mean statistic based classifier (e.g., Rocchio) and the advantages of one that uses the full data (e.g., an SVM). Thus bundling allows us to explicitly adjust the level of classification speed and accuracy.

For text bundling, we partition the full data set D into m equal-sized partitions $P_1, \ldots, P_m$. Each partition becomes a single data point in the bundled data set $D' = \{D'_1, \ldots, D'_m\}$. Each element D'_i is a vector of the mean statistics of the data in partition P_i. We calculate the elements of D'_i as $D'_{ij} = s_j(P_i)$. Note this method also satisfies the global constraint; the global mean is simultaneously conserved. We then feed D' to a classifier of our choosing.

The remaining question is determining how to partition the points. We present two algorithms, randomized bundling and Rocchio bundling. In randomized bundling, we simply partition points randomly. This takes very little time: one pass over the n training points—the same as subsampling.

Randomized bundling puts together random points, so it poorly preserves data point locations in feature space. Ultimately we would like to preserve as much location information as possible by bundling nearby points. We might try doing a bottom-up clustering where we combine elements closest to one another, but

procedure Rocchio Bundling

1: Let n be the number of documents.
2: Let m be the chosen partition size (we assume n/m is an integer).
3: Let $s_1, \ldots, s_V$ be the mean statistics as defined in equation 3.
4: Sort the document indices $\{1, \ldots, n\}$ according to RocchioScore($\vec{d_i}$). Let $r_1, \ldots, r_n$ be the sorted indices.
5: Partition the documents according to the sorted indices. Let $P_i = \{d_{r_{(i-1)m+1}}, \ldots, d_{r_{im}}\}$ be the ith partition.
6: Compute $d'_{ij} = s_j(P_i)$, where d'_i is the ith reduced data point and d'_{ij} is the count of word j.
7: $D' = \{\vec{d'_1}, \ldots, \vec{d'_m}\}$ is the reduced data set.

Table 1. This table summarizes the four text data sets used in our experiments. SVM timing and accuracy are based on SVMFu; question marks (?) indicate SVM runs that did not finish within a week. The Reuters "accuracy" is precision-recall break even; 20 news and Industry Sector is multi-class accuracy; Ohsumed is binary accuracy. Features refer to the number of features found in the training set.

	20 News	IS	Reut.	Ohsum.
Train Size	12,000	4797	7,700	179,215
Test Size	7,982	4,822	3,019	49,145
Features	62,060	55,194	18,621	266,901
SVM time	6464	2268	408	?
Accuracy	86.5%	92.8%	88.7%	?

simply computing all pairs of distances is too time consuming. An alternate, faster clustering algorithm is k-means, an algorithm that iteratively attracts points to k centroids. Empirically this method was not much more accurate than randomized bundling but it was much slower. Next, we describe a faster algorithm that preserves more location information than random, but runs faster than the two clustering algorithms.

It is difficult to do any fast clustering while considering all dimensions of the data. If we can project the points onto one important dimension, then we can cluster as fast as we can sort. Rocchio bundling projects points onto a vector and then partitions points that are near one another in the projected space. For binary classification, that vector is the one that connects the class centroids. For multi-class problems, we choose a vector for each class that connects the class centroid with the centroid of all the other classes' data.

Let $\vec{c}$ the centroid of one class, and $\vec{c'}$ the other centroid. Let $\vec{d_i}$ be the data point. Our score is the projection of $\vec{d_i}$ on to $\vec{c'} - \vec{c}$:

$$\text{RocchioScore}(\vec{d_i}) = \vec{d_i} \cdot (\vec{c} - \vec{c'}), \qquad (4)$$

By sorting documents by their score, then bundling consecutive sorted documents, we concatenate similar documents.

This provides a quick way to decide which points fall within a bundle. Further details on the algorithm are provided in the Rocchio bundling pseudo code. The pre-processing time for Rocchio bundling is $O(n \log(m))$.

3.2. Other Applications and Methods for Bundling

Here we describe how bundling might be used in domains where more statistics need to be preserved or where statistics other than the sample mean are important. This section is not relevant to text, but may be of interest to a wider audience interested in applying it to different domains.

Single, simple statistics like feature maxima, minima and means can be solved in a straightforward fashion like the text example. If each local bundle has the maximum of each feature, then the global maximum for each feature will also be conserved.

One can also bundle with two or more statistics simultaneously, though only in special cases. Instead of bundling a partition to one point, we bundle to two or more. One can preserve the minimum and maximum statistics by creating two points, one of which contains the minimum value for each feature, the other containing the maximum. One can preserve mean and variance statistics by converting each partition into two points that have the same mean and variance statistics as the partition. Both of these examples simultaneously satisfy the local and global constraints.

4. Results

4.1. Data Sets and Experimental Setup

Our experiments try to quantify the relationship between speed and accuracy for five different data reduction techniques at varying levels of speed-ups. In order to perform these comparisons, we made a test bed as broad and fair as possible. We compared the various reduction techniques on SvmFu, a fast, chunking, publicly available SVM implementation (Rifkin,

2000). We coded each pre-processing step in C++, and compared the total reported preprocessing, training and testing time reported by the UNIX time command, for user process time. We used a fast, relatively un-used machine (1GB RAM, 1.6GHz PIII processor) to perform all the experiments.

We use four well-known text data sets: 20 Newsgroups (McCallum & Nigam, 1998; Slonim & Tishby, 1999; Berger, 1999), Industry Sector (Ghani, 2000), Reuters-21578 (Yang & Liu, 1999; Joachims, 1997; Schapire & Singer, 2000), and Ohsumed(Hersh et al., 1994). The data sets are summarized in Table 1. We use Rainbow to pre-process the raw documents into feature vectors (McCallum, 1996); our pre-processing steps mimic those used by Rennie and Rifkin (2001) for 20 Newsgroups and Industry Sector; we use the Mod-Apte split for Reuters-21578. For Ohsumed, we used the top ten categories and split training and test by date; we compute accuracy as the average binary accuracy of those ten categories.

We chose to base our tests on the Support Vector Machine (SVM), a highly accurate, but slower (super-linear) algorithm for classification. In many text-classification tasks, it has consistently outperformed other algorithms (Yang & Liu, 1999; Joachims, 1997). The SVM takes the positive and negative training points, and tries to place a hyper-plane between them so that it optimizes a tradeoff between classification error and margin width. It is more sensitive to the exact location of the training points than algorithms that simply use the features' means. For more information about the SVM, see the Burges (1998) tutorial.

For our experiments, we use the SMART ltc transform; the SvmFu package is used for running experiments (Rifkin, 2000). We set C, the penalty for misclassifying training points, at 10. We produce multi-class labels by training a one-versus-all SVM for each class and assigning the label of the most confident SVM. We use the linear kernel for the SVM since after applying the SMART ltc transform, the linear kernel performs as well as non-linear kernels in text classification (Yang & Liu, 1999).

We use one of the best performing feature selection algorithms used in (Mladenic, 1998), which ranks features according to $|p(f_i|+) - p(f_i|-)|$, where $p(f_i|c)$ is empirical frequency of f_i in class c training documents. Subsampling is done by randomly removing training documents, and bagging is done as outlined in the related work section. The remaining two algorithms, Randomized bundling and Rocchio bundling are the focus of this paper.

4.2. Results

In this section, we analyze the results from our empirical work found in Table 2 and in Figure 1 showing the exact tradeoffs between speed and accuracy on the various data sets. We discuss how each of the five speed-up techniques worked.

Our results on Ohsumed explain how different data reduction techniques work on truly large data sets. Neither bagging nor feature selection were useful on the data set. No bagging runs completed within the allotted time (8 hours per run) and feature selection required reducing the feature set to 50 features, which yielded very poor results.

In general, feature selection was a disappointment on both speed and accuracy grounds. The actual feature selection algorithm is empirically (though not algorithmically) a small factor slower than the other algorithms: one must perform calculations for every feature, and the number of features tends to be larger than the number of data-points. This effect is relatively minor. More importantly, the speedups in training times for our SVM were relatively minor. In our tests, reducing the number of training points sped up the algorithm more than reducing the number of features.

Our results on Ohsumed help explain why feature selection does so poorly in our empirical results. At the fast, inaccurate end, we choose the top 50 or so features. However, those 50 features are distributed among 170,000 training points, so most documents end up consisting of a single feature. If a document's single feature is a somewhat common word, it will tend to appear at least once in both the class and its opposite class. So a common feature will result in duplicate documents in opposite classes, generally making that feature useless. If the document's single feature is somewhat rare, and only appears in one class, it generally can only help classify a small percentage of the test documents. So when feature selecting to only a few points, classification accuracy sometimes becomes near random. When we choose more features for Ohsumed (even 100 total features) we find that the SVM takes much longer than our eight hour time limit.

Bagging was generally more accurate than feature selection, but was slow. As discussed before, splitting the training set into more bags reduces the training time, but increases the test time. The amount of training time per number of bags is shaped like a "U": early speed increases are due to less training time, later speed decreases are due to more testing time. This means that bagging can only speed up an algorithm to

Table 2. This table summarizes results for various text corpora (columns) against various data reduction algorithms (rows). Results are expressed as empirical time-accuracy pairs. The first set of numbers are the time and accuracy for the algorithms resulting in the slowest classification times; the second set of numbers represent the fastest classification times. The maximally bundled data is functionally similar to the Rocchio algorithm. Question marks (bagging) indicate runs that did not complete in the allotted time. Only one run of feature selection completed within the allotted time (*).

|	20 News		IS		Reuters		Ohsumed	
	Slowest Results (time,accuracy)							
Bagging	4051	.843	1849	.863	346	**.886**	?	?
Feature Selection	5870	.853	2186	.896	507	**.884**	11010*	.647*
Subsample	2025	.842	926	.858	173	.859	39340	.808
Random Bundling	2613	**.862**	1205	.909	390	.863	25100	**.830**
Rocchio Bundling	2657	**.864**	1244	**.914**	404	**.882**	26710	.804
	Fastest Results (time,accuracy)							
Bagging	2795	.812	1590	.173	295	.878	?	?
Feature Selection	4601	.649	1738	.407	167	.423	11010*	.647*
Subsample	22	.261	59	.170	9.6	.213	13	.603
Random Bundling	117	.730	177	.9	105	.603	74	.731
Rocchio Bundling	173	.730	248	.9	129	.603	134	.731
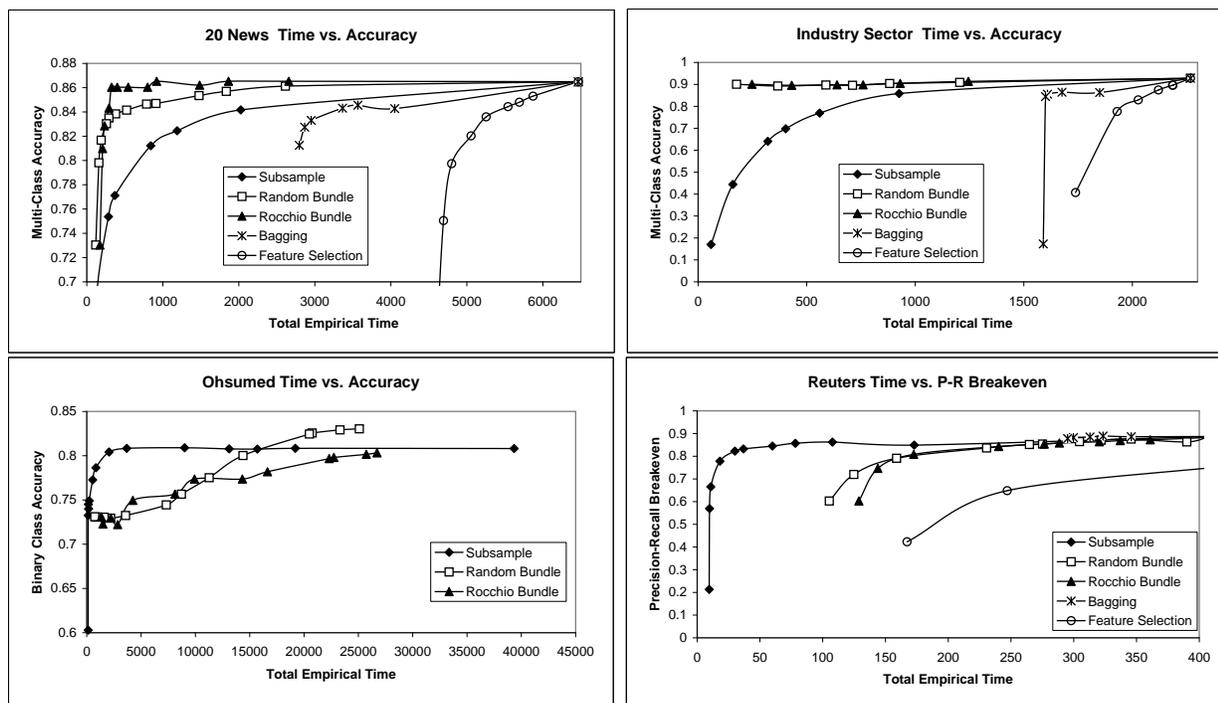
Figure 1. Speed (measured in seconds) against accuracy plotted for the four data sets, and the five data reduction techniques included in this study. On most graphs, the far right point corresponds to using all the training data, so the accuracies converge. In the Ohsumed data set, running the SVM on such large amounts of data took too long (didn't finish after a week). We show a close-up on Reuters because the larger times are not interesting.

a certain point that is slower than some of the other algorithms. On the Ohsumed database, there were no bagging sizes that came close to completion within our eight hour time limit.

Bagging yielded good accuracies for Reuters, and acceptable accuracies for 20 news and industry sector. Its accuracy is tied to subsampling's accuracy: the sharp drop off visible in the Industry Sector graph is due to the fact that subsampling down to a certain level yields extremely low (nearly random) accuracies; and combining a series of almost random classifiers does not really help create a better classifier.

Subsampling worked overall better than expected. Relative to the other algorithms, the process of randomly selecting documents is fast. For a given reduced training set size, subsampling pre-processing runs faster than any of the other data reduction algorithms. More importantly, the data points that it produces are more sparse (more zero entries) than bundling, and hence an algorithm like the SVM will run faster with subsampled points than with denser bundled points. For example, on Ohsumed we could run the SVM with 18,000 subsampled points, but with only 12,000 bundled points.

Subsampling also leads to surprisingly accurate classification on the two binary problems, Reuters and Ohsumed. On Reuters, it appears that a small number of documents can adequately define a class; so for a given amount of time, subsampling will often perform better than the other algorithms. On Ohsumed, the accuracy seems to level off in certain ranges, performing worse than bundling for higher amounts of time, but better than bundling for intermediate amounts of time. Subsampling did not work well on the multi-class problems. We believe this is because the multi-class problems are more difficult and require more training points to generate a good classifier.

In all of our data sets, subsampling has a steep drop off in accuracy; eventually at 1 point per class, it will obviously do poorly. The difficulty with subsampling is knowing when that drop off will occur. One might get lucky, like with Reuters, where we found the heavy drop off doesn't occur until you remove 19 of every 20 documents. Or one might get unlucky, like with 20 News, where removing half of the documents causes an immediate drop in accuracy.

The most consistent and often best performing algorithms were the two bundling algorithms. They had the best performances for 20 news and industry sector, and alternated the lead on Ohsumed with subsampling. For most data sets, they had the highest

scores at the slow/high accuracy end (bundling pairs of points for most; for Ohsumed, combining every 14 points into 1); and also did not drop as sharply as subsampling on the fast/low accuracy end. As mentioned before, full bundling combined with the SVM acts like the Rocchio classifier.

The Rocchio and random bundling methods have different strengths and weaknesses. As Table 2 shows, with minimal amounts of bundling (two points per bundle), Rocchio usually outperforms random and most other algorithms. This is because Rocchio has lots of freedom to choose the points that go in each bundle. At the other end of the spectrum, Rocchio has very few choices. For example, when bundling to one point Rocchio has no choices–it bundles identically to random. However, Rocchio takes more time to complete. Thus, Rocchio works well when bundling to more points, but suffers from higher preprocessing times when less points are retained.

5. Conclusions and Future Work

We present a new data reduction technique, called bundling, that tries to maintain user-chosen statistics of the data. We focused on text bundling, where the chosen statistic is the mean of each word feature in the training documents. We gave empirical evidence that bundling performs well on a variety of text data sets.

In the future, we would like to extend bundling in both a theoretical and empirical sense. It may be possible to analyze or provide bounds on the loss in accuracy due to bundling. We would like to construct general methods for bundling sets of statistics. We are also interested in extending bundling to other machine learning domains.

References

Berger, A. (1999). Error-correcting output coding for text classification. *Proceedings of IJCAI-99 Workshop on Machine Learning for Information Filtering.* Stockholm, Sweeden.

Breiman, L. (1996). *Bias, variance, and arcing classifiers* (Technical Report 460). Statistics Department, University of California.

Burges, C. J. C. (1998). A tutorial on Support Vector

Machines for pattern recognition. *Data Mining and Knowledge Discovery, 2*, 121–167.

Domingos, P. (2002). When and how to subsample: Report on the KDD-2001 panel. *SIGKDD Explorations, 3.*

Ghani, R. (2000). Using error-correcting codes for text classification. *Machine Learning: Proceedings of the Seventeenth International Conference.*

Hersh, W., Buckley, C., Leone, T., & Hickam, D. (1994). OHSUMED: An interactive retrieval evaluation and new large test collection for research. *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 192–201).

Joachims, T. (1997). A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. *Proceedings of the Fourteenth International Conference on Machine Learning.*

McCallum, A., & Nigam, K. (1998). A comparison of event models for naive Bayes text classification. *Proceedings of the AAAI-98 workshop on Learning for Text Categorization.*

McCallum, A. K. (1996). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/∼mccallum/bow.

Mladenic, D. (1998). Feature subset selection in text-learning. *Proceedings of the Tenth European Conference on Machine Learning.*

Rennie, J. D. M., & Rifkin, R. (2001). *Improving multiclass text classification with the Support Vector Machine* (Technical Report AIM-2001-026). Massachusetts Insititute of Technology, Artificial Intelligence Laboratory.

Rifkin, R. (2000). Svmfu. http://five-percent-nation.mit.edu/SvmFu/.

Rocchio, J. J. (1971). Relevance feedback in information retrieval. In G. Salton (Ed.), *The SMART retrieval system: Experiments in automatic document processing*, 313–323. Prentice-Hall.

Schapire, R. E., & Singer, Y. (2000). Boostexter: A boosting-based system for text categorization. *Machine Learning, 39*, 135–168.

Slonim, N., & Tishby, N. (1999). Agglomerative information bottleneck. *Neural Information Processing Systems 12 (NIPS-99).*

Yang, Y., & Liu, X. (1999). A re-examination of text categorization methods. *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval.*

Yang, Y., & Pedersen, J. O. (1997). A comparitive study on feature selection in text categorization. *Proceedings of the Fourteenth International Conference on Machine Learning.*