Context-sensitive learning methods for text categorization

William W. Cohen and Yoram Singer
AT&T Research
600 Mountain Avenue
Murray Hill, NJ 07974
{wcohen,singer}@research.att.com
Phone: 908 582 2092 Fax: 908 582 7550

Abstract

Two recently implemented machine learning algorithms, RIPPER and sleeping experts for phrases, are evaluated on a number of large text categorization problems. These algorithms both construct classifiers that allow the "context" of a word w to affect how (or even whether) the presence or absence of w will contribute to a classification. However, RIPPER and sleeping experts differ radically in many other respects: differences include different notions as to what constitutes a context, different ways of combining contexts to construct a classifier, different methods to search for a combination of contexts, and different criteria as to what contexts should be included in such a combination. In spite of these differences, both RIPPER and sleeping experts perform extremely well across a wide variety of categorization problems, generally outperforming previously applied learning methods. We view this result as a confirmation of the usefulness of classifiers that represent contextual information.

1 Introduction

Learning methods are frequently used to automatically construct classifiers from labeled documents [Lewis, 1992; Lewis and Ringuette, 1994; Lewis and Gale, 1994; Apté *et al.*, 1994; Wiener *et al.*, 1995; Cohen, 1995b]. In this paper, we will investigate the performance of two recently implemented machine learning algorithms on a number of large text categorization problems. The two algorithms considered are set-valued RIPPER, a recent rule learning algorithm [Cohen, 1995a; Cohen, 1996], and *sleeping experts*, a new on-line learning method.

These algorithms share several features that make them attractive for large text categorization problems. First, both algorithms are efficient on large, noisy corpora, running in linear or nearly linear time. Second, both algorithms use what could be called a *direct* representation of text, in which a document is represented as an ordered list of tokens; *i.e.*, it is not necessary to extract from a corpus a small set of informative features.

Third, both algorithms allow the *context* of a word w to influence how the presence or absence of w will contribute to a classification. This contrasts with many common text categorization schemes, such as "naive" Bayes and Rocchio's algorithm, which produce linear classifiers. While linear classification schemes are efficient and easy to implement, they rely on the unrealistic assumption that the influence of any word w on classification will be insensitive to the presence or absence of other words in the document.

Finally, both algorithms have the aesthetic advantage that "contexts" are determined by the learning algorithm itself. There is no external process that selects likely combinations of words to be used as contexts, or that integrates the predictions of classifiers learned using different externally-imposed notions of context; instead contexts are created naturally, as a by-product of the process of finding a predictive classifier.

While sharing these important properties, however, RIPPER and sleeping experts differ radically in many other respects. For RIPPER, the context of a word w_1 is a conjunction of the form

$$w_1 \in \text{document and } w_2 \in \text{document} \ldots \text{and } w_k \in \text{document}$$

In other words, the context of a word w_1 consists of a (usually small) number of other words $w_2 \ldots w_k$ that must co-occur with w_1, but which may occur in any order, and in any location in the document. In contrast, sleeping experts uses sparse phrases (a sequence of not necessarily consecutive words) to represent context, and hence the context of a word consists simply of (some subset of) the *nearby* words, in a *fixed* order.

RIPPER attempts to find a simple hypothesis, in the form of a small disjunction of conjunctions (of the form shown above) that accurately classifies the training data. To solve this difficult optimization problem, a number of heuristic methods are used. In contrast, sleeping experts' hypothesis is a linear combination of *all* sparse phrases that occur in the training corpus—typically an enormous set. To construct this large linear combination, sleeping experts uses a multiplicative update algorithm that has strong theoretical justifications, being based on recent work in competitive analysis of learning algorithms [Cesa-Bianchi *et al.*, 1993] and weak-hypothesis boosting [Freund and Schapire, 1995].

Finally, the conjunctions that are included in RIPPER's hypotheses always represent "contexts" that are positively correlated with the class being learned. However, sleeping

From "Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR96), Zurich, Switzerland, ACM Press, 1996, pages 307–315.

ireland ← ireland∈document.
ireland ← ira∈document, killed∈document.
ireland ← ira∈document, kills∈document.
ireland ← ira∈document, belfast∈document.
ireland ← ira∈document, to∈document, calls∈document.
ireland ← irish∈document, abortion∈document.
ireland ← ira∈document, shot∈document.
ireland ← ira∈document, out∈document.
else not_ireland.

Figure 1: A learned ruleset for the category 'ireland'

experts makes use of phrases that are both positively and negatively correlated with the class of interest—negatively correlated phrases being taken as evidence against membership in the category.

One of the more interesting observations of this paper is that both RIPPER and sleeping experts for phrases perform extremely well across a wide variety of categorization problems, generally outperforming previously applied learning methods. This fact holds in spite of the fundamental differences listed above as to what constitutes a "context", how to combine contexts, how to search for a combination of contexts, and which contexts should be included in a such a combination. We view this result as a confirmation of the usefulness of learning classifiers that represent contextual information.

In the remainder of the paper, we present the learning algorithms, describe the benchmark problems used and the experimental results obtained with them, and finally present our conclusions.

2 Learning algorithms

2.1 RIPPER

2.1.1 How RIPPER learns rulesets

We will first briefly describe the rule learning algorithm RIPPER; for a more detailed description, see Cohen [1995a]. The classifier constructed by RIPPER is a set of rules, such as the one illustrated in Figure 1. This set of rules can be interpreted as a disjunction of conjunctions: a document d is considered to be in the category `ireland` if and only if

(word `ireland` appears in d) OR
(word `ira` appears in d AND word `killed` appears in d) OR
⋮
(word `ira` appears in d AND word `shot` appears in d) OR
(word `ira` appears in d AND word `out` appears in d)

Rulesets enjoy some properties that make them useful in certain contexts. If a ruleset is compact, it is relatively easy for people to understand; this may make it easier for users to accept a learned classifier as being reasonable. Rulesets can also be easily converted to queries for a boolean search engine [Cohen and Singer, 1996].

RIPPER builds a ruleset by repeatedly adding rules to an empty ruleset until all positive examples are covered. Rules are formed by first splitting the training data into two sets, a "growing set" and a "pruning set", and then greedily adding conditions to the antecedent of a rule with an empty antecedent until no negative examples are covered; after such a rule is found, the rule is simplified, by greedily deleting conditions so as to improve the rule's performance on the "pruning" examples. In this phase of learning, different *ad hoc* heuristic measures are used to guide the greedy search for new conditions, and the greedy search for simplifications.

After a ruleset is thus constructed, an "optimization" phase modifies the ruleset so as to reduce its size and improve its fit to the training data. Each pass of the optimization involves looping over each rule R in the constructed ruleset, and attempting to construct a replacement for R that improves performance of the entire ruleset. To construct candidate replacements, a strategy similar to the one used to construct rules in the covering phase is used: a rule is grown, and then simplified, with the goal of simplification being now to reduce the error of the total ruleset on another held-out "pruning" set.

2.1.2 Extensions to RIPPER motivated by text

Before running these experiments, RIPPER was modified so as to be more efficient for text categorization problems. In the initial implementation of RIPPER, examples were represented as feature vectors. This implementation could be used to learn rulesets such as the one shown above by constructing an appropriate set of boolean features; for example, one could construct a feature for each word appearing the corpus. A corpus with m documents and n words would thus be represented as a $m \times n$ matrix. However, this representation is inefficient in its usage of space, since even a moderately large corpus will usually contain many different words. To avoid this problem, RIPPER was extended to allow the value of an attribute to be a set of symbols. The primitive tests on a set-valued attribute a (*i.e.*, the tests which are allowed in rules) are of the form "$w_i \in a$". When constructing a rule, RIPPER finds the test that maximizes information gain for a set of examples S efficiently, making only a single pass over S for each attribute. All symbols w_i that appear as elements of attribute a for some training example are considered by RIPPER. This extension is described in more detail elsewhere [Cohen, 1996].

In our experiments, a document is generally represented with a single set-valued feature, the value of which is the set of all words appearing in the document. In the implementation, the set can include multiple occurrences of an element (extra occurrences are simply ignored), so one can represent a document with the list of words that appear in it.

Optionally, RIPPER can also include tests of the form "$w_i \notin$ *feature*" in its rules. Although extending RIPPER to find and use such tests is a simple matter, there are some reasons for being wary of using them. In this paper we will typically present results for two versions of RIPPER—one with negative word tests allowed, and one with negative word tests forbidden. Unless stated otherwise, RIPPER will be used with negative word tests forbidden.

A second extension to RIPPER motivated by text categorization problems allows the user to specify a *loss ratio* [Lewis and Catlett, 1994]. A loss ratio indicates the ratio of the cost of a false negative to the cost of a false positive; the goal of learning is to minimize misclassification cost on unseen data. By using an appropriate loss ratio RIPPER can trade off recall on a category for precision. Loss ratios in RIPPER are implemented by changing the weights given to false positive errors and false negative errors in the pruning and optimization stages of the learning algorithm.

2.2 Sleeping experts for phrases

Sleeping experts is based on a new framework for combining the "advice" of different "experts" (or in another words the predictions of several classifiers) which has been developed within the computational learning community over the last several years. Prediction algorithms in this framework are given a pool of fixed "experts"—each of which is usually a simple, fixed, classifier—and build a *master algorithm*, which combines the classifications of the experts in some manner. Typically, the master algorithm classifies an example by using a weighted combination of the predictions of the experts. Building a good master algorithm is thus a matter of finding an appropriate weight for each of the experts. The vast majority of the weight allocation algorithms are on-line algorithms. That is, the examples are fed one-by-one to the master algorithm, which updates the weight of the different experts based on their performance (prediction) on that example. Several weight updating methods have been examined and analysed. Formal results show that under some circumstances, very high dimensional weight allocation problems can be handled with moderate amounts of training data, if an appropriate multiplicative weight update algorithm is used [Littlestone and Warmuth, 1994; Kivinen and Warmuth, 1994]. Moreover, empirical evidence indicates that multiplicative update algorithms often outperform traditional learning techniques for linear classifiers [Blum, 1995; Lewis *et al.*, 1996].

The *sleeping experts* algorithm is a procedure of the above type. It is based on two recent advances in multiplicative update algorithms. The first is a weight allocation algorithm called Hedge [Freund and Schapire, 1995], which is applicable to a broad class of learning problems and loss functions. The second is the *infinite attribute model* [Blum, 1990]. In this setting, there may be any number of experts, but only a few actually post predictions on any given example; the remainder are said to be "sleeping" on that example.

In the context of document classification, an expert can be any lexical unit. Such an expert is "awake" and predicts if the unit appears in a given document. In order to test and compare simple context sensitive models we chose the pool of possible experts to be the set of all (sparse) phrases that appear in a document. Let ω_i be the ith word appearing in the document. Each expert is an ordered list of up to n words and is of the form $\omega_{i_1}\omega_{i_2}\ldots\omega_{i_j}$ where $i_1 < i_2 < \ldots i_{j-1} < i_j$ and $i_j - i_1 < n$. Put another way, each expert corresponds to a given phrase that may appear in *any* position in the text and are not bound to particular positions. Since we allow "holes" in each phrase, we will use the term "sparse" phrases for this generalization of the word n-gram model. Note that when $n = 1$ the pool contains all single words appearing in the document at all possible locations.

In the sleeping experts framework the weight associated with each phrase is "learned" in an on-line manner so as to minimize a cost function, in our case the classification error. However, in order to have a fair comparison with other systems, we will update the weights of the different phrases only during the training phase, and keep them fixed during the test phase. We will now give a description of the sleeping experts procedure for phrases. Pseudo-code for the algorithm is shown in Fig. 2.

The master algorithm maintains a *pool*, which is a set recording the sparse phrases that have appeared in any of the previous documents, and a set $\mathbf{p}$ containing one weight for each sparse phrase in the pool. At all times, all weights in $\mathbf{p}$ will be nonnegative; however, the weights need not sum to one.

At each time step t, a new document is presented. Let the content of the t-th document be denoted by $\omega_1^t, \omega_2^t, \ldots, \omega_l^t$, where l is the length (number of words) of the document. Therefore, the set of active phrases, denoted by W^t, is

$$W^t = \left\{ \bar{w} \mid \bar{w} = \omega_{i_1}^t \omega_{i_2}^t \ldots \omega_{i_j}^t \right\}$$

where

$$1 \le i_1 < i_2 < \ldots i_{j-1} < i_j \le l, \; i_j - i_1 < n \;\; .$$

Thus a phrase is a sequence of words, possible containing some "gaps", appearing in a fixed order but at any position in a document. Given this set the first task of the master algorithm is to make a prediction based on the prediction of each of the active mini-experts. In our implementation, we associate with each phrase, $\bar{w}$, two mini-experts, denoted by $\bar{w}_1$ and $\bar{w}_0$. The first mini-expert consistently predicts that the document belongs to the class (whenever the sparse phrase $\bar{w}$ appears in a document), and the second consistently predicts that the document does not belong to the class. Clearly, only one of them is right given the correct classification of the document. To make a prediction, the master algorithm decides on a distribution $\tilde{\mathbf{p}}$ over the active mini-experts, which is determined by restricting the set of weights $\mathbf{p}$ to the set of active mini-experts and normalizing the weights. Therefore, the set of active mini-experts, denoted by E^t, is,

$$E^t = \left\{ \bar{w}_k \mid \bar{w} \in W^t, k \in \{0,1\} \right\} \;\; .$$

We denote the vector of normalized weights by $\tilde{\mathbf{p}}$, where

$$\tilde{p}_{\bar{w}_k}^t = p_{\bar{w}_k}^t \Big/ \left(\sum_{\bar{w}'_l \in E^t} p_{\bar{w}'_l}^t \right) \;\; .$$

The prediction of the master algorithm, denoted by y^t, depends on the weighted sum of the mini-experts. Since half of the mini-experts always predict 0 when they are active, this sum can be efficiently calculated by summing the weights[1] of mini-experts that predict 1

$$\begin{aligned} y^t &= \sum_{\bar{w}_k \in E^t} \tilde{p}_{\bar{w}_k}^t k \\ &= \sum_{\bar{w} \in W^t} \tilde{p}_{\bar{w}_1}^t \\ &= \frac{\sum_{\bar{w} \in W^t} p_{\bar{w}_1}^t}{\sum_{\bar{w} \in W^t} \sum_{k=0,1} p_{\bar{w}_k}^t} \;\; . \end{aligned}$$

Therefore, the master algorithm combines the predictions of all the experts into a single prediction which is a number between zero and one. In order to classify a document we set a threshold θ_c, and classify a document as positive (class 1) iff $y^t > \theta_C$. When minimizing error, we always use $\theta_C = 1/2$. However, to achieve a desired precision (or recall), the threshold can be adjusted; a low value usually results in higher recall, and a high value usually results in higher precision.

After receiving the predictions of all the mini-experts, the master algorithm receives the true classification of the

[1] If the goal is to minimize the absolute loss of the master's predictions, then a non-linear function, denoted by f_β, should be applied to the weighted sum of the experts' predictions. However, since f_β is monotonically increasing such that $f_\beta(0) = 0, f_\beta(1/2) = 1/2$, and $f_\beta(1) = 1$ (see [Vovk, 1990; Cesa-Bianchi *et al.*, 1993]), then for classification purposes we can simply use the weighted sum of the predictions itself.

document, and then updates the weights of the mini-experts appropriately. The goal is to update the weights of the active mini-experts so as to reflect their correlation with the correct classification of the document. To update the weights, the weight of each expert is multiplied by the factor $\beta^{l(\bar{w}_k)}$ where $l(\bar{w}_k)$ is the loss of the mini-expert $\bar{w}_k$. The parameter $\beta < 1$ is called the learning rate, and controls how quickly the weights are updated. We used values for β in the range $[0.1, 0.5]$.

In our implementation we used the absolute error as our loss function, *i.e.*, $l(\bar{w}_k) = 1$ if the mini-expert $\bar{w}_k$ predicted incorrectly, and $l(\bar{w}_k) = 0$ if the mini-expert is $\bar{w}_k$ predicted correctly. Thus the rule above can be summarized as follows: for each two mini-experts based on the same phrase, we multiply the weight of the incorrect mini-expert by β, and keep the weight of the correct mini-expert unchanged. Then, after updating weights for the mini-experts, weights of the active mini-experts are normalized so that the total weight of the active mini-experts does not change; the effect of this re-normalization is to actually increases the weights of the mini-experts which were correct.

We would like to note that more 'sophisticated' experts can be constructed. For instance, an expert may take into account the number of appearances of the phrase in the current (and previous documents), or use an inverse document frequency or a mixture of Poisson models [Church and Gale, 1995] to predict the classification of the document. However, it is not clear whether an on-line (incremental) prediction scheme that is based on such information can be efficiently constructed.

Two properties of this algorithm require further explanation. First, if a phrase that appears in many documents is not relevant to the class considered, then the number of times each of the phrase's mini-expert is correct will be approximately equal to the number of times it is incorrect. This means that over time the weight of both mini-experts corresponding to the phrase will be demoted. Second, when many "relevant" phrases appear in a document, the total weight of the active mini-experts is large, and the promotion of the correct mini-experts due to the re-normalization is small; however, if there are only few "relevant" phrases, then the total weight of the active mini-experts is small, and hence the re-normalization results in a higher promotion of the correct mini-experts. Clearly, this is a desirable property in text categorization, since it allows a rare but highly correlated phrase to have a large influence on classification.

The result of the sleeping experts algorithm is a large pool of sparse phrases each associated with a weight. In testing the performance of the sleeping experts algorithm, whenever a new (unclassified) document is presented, the phrases which appear in the document are extracted from the pool and their weights are normalized so that they sum to 1. Following the same steps as in the training phase (but without updating the weights) the master's prediction is then computed from the weighted sum of the active mini-experts. The actual classification is done by comparing the master's prediction with the threshold θ_C.

Analysis of the general sleeping experts algorithm is beyond the scope of this paper and will be presented elsewhere. Briefly, if one defines the loss of the master algorithm to be the average loss with respect to the instantaneous distribution defined by the set of active mini-experts on each example, $\tilde{\mathbf{p}}$, the cumulative loss of the master algorithm over all t can be bounded relative to the loss suffered by the best possible fixed weight vector.

2.3 Rocchio

As a basis for comparison, we implemented a version of Rocchio's algorithm [Rocchio, 1971], as adapted to text categorization by Ittner *et al.* [1995]. We represent the data (both training and test documents) as vectors of numeric weights. The weight vector for the mth document is $\mathbf{v^m} = (v_1^m, v_2^m, \ldots, v_l^m)$, where l is the number of indexing terms used. We use single words as terms. We follow the TF-IDF weighting [Salton, 1991] and define the weight v_k^m to be:

$$v_k^m = \frac{f_k^m \log(N_D/n_k)}{\sum_{j=1}^{l} f_j^m \log(N_D/n_j)} \quad .$$

Here, N_D is the number of documents, n_k is the number of documents in which the indexing term k appears, and f_k^m is:

$$f_k^m = \begin{cases} 0 & l = 0 \\ log(l) + 1 & \text{otherwise} \end{cases}$$

where l is the number of occurrences of the indexing term k in document m. In Rocchio's algorithm a prototype is produced for each class c. This prototype is represented as a single vector $\tilde{\mathbf{v}}^{\mathbf{c}}$ of the same dimension as the original weight vectors $\mathbf{v}^{\mathbf{1}}, \ldots, \mathbf{v}^{\mathrm{N_D}}$. For class c, the k'th term in its prototype is defined to be

$$\tilde{v}_k^c \stackrel{\text{def}}{=} \max\left\{0,\ \frac{\beta}{|R_c|}\sum_{m\in R_c} v_k^m \ - \ \frac{\gamma}{|\bar{R}_c|}\sum_{m\in \bar{R}_c} v_k^m\right\}$$

where R_c is the set of all documents in class c and $\bar{R}_c$ is the set of all documents not in c. The parameters γ and β control the relative contribution of the positive and negative examples to the prototypes vector; we use the standard values $\beta = 16$ and $\gamma = 4$ [Buckley *et al.*, 1994]. Documents to be classified are first converted into weight vectors, and then compared against the prototype $\tilde{\mathbf{v}}^{\mathbf{c}}$ by computing the dot product. A novel document is classified as a member of class c if its distance (dot product with the prototype) is less than a threshold θ_c, which can be chosen so as to balance recall and precision in some set manner. In our experiments, θ_c was chosen to either minimize error on the training set, or to minimize total loss on the training set for a specific loss ratio.

3 Experimental results

3.1 The AP titles corpus

The first benchmark we will use is a corpus of AP newswire headlines, tagged as being relevant or irrelevant to topics like "federal budget" and "Nielsens ratings". This dataset is described in more detail elsewhere [Lewis and Gale, 1994; Lewis and Catlett, 1994]. The corpus contains 319,463 documents in the training set and 51,991 documents in the test set. The headlines are an average of nine words long, with a total vocabulary is 67,331 words. No preprocessing of the text was done, other than to convert all words to lower case and remove punctuation marks. We will use nine representative categories.[2]

In previous work, Lewis and Gale [1994] and Lewis and Catlett [1994] used this domain to evaluate the performance

[2] We have discarded one of the ten categories ("tickertalk"). This category seems to be very hard to learn, and hence learning algorithms often generate classifiers with extreme recall and precision values. This tends to distort averages across the categories.

Parameters: $\beta \in (0,1)$, $\theta_C \in (0,1)$, number of labeled documents T
Initialize: $Pool \leftarrow \phi$
Do for $t = 1, 2, \ldots, T$

1. Receive a new document $\omega_1^t, \omega_2^t, \ldots, \omega_l^t$ and it classification c^t.
2. Define the set of active phrases:
$$W^t = \left\{ \bar{w} \mid \bar{w} = \omega_{i_1}^t \omega_{i_2}^t \ldots \omega_{i_j}^t ,\ 1 \le i_1 < i_2 < \ldots i_{j-1} < i_j \le l,\ i_j - i_1 < n \right\}$$
3. Define the set of active mini-experts:
$$E^t = \left\{ \bar{w}_k \mid \bar{w} \in W^t,\ k \in \{0,1\} \right\}$$
4. Initialize the weights of new mini-experts:
$$\forall \bar{w}_k \in E^t \ s.t. \ \bar{w}_k \notin Pool: \ p_{\bar{w}_k}^t = 1 \ .$$
5. Classify the document as positive if
$$y^t = \frac{\sum_{\bar{w} \in W^t} p_{\bar{w}_1}^t}{\sum_{\bar{w} \in W^t} \sum_{k=0,1} p_{\bar{w}_k}^t} > \theta_C \ .$$
6. Update weights:
$$p_{\bar{w}_k}^{t+1} = p_{\bar{w}_k}^t \, \beta^{l(\bar{w}_k)} = \begin{cases} p_{\bar{w}_k}^t & c^t = k \\ \beta \, p_{\bar{w}_k}^t & c^t \neq k \end{cases}$$
7. Renormalize weights:
$$p_{\bar{w}_k}^{t+1} = p_{\bar{w}_k}^{t+1} \left(\sum_{\bar{w}'_k \in E^t} p_{\bar{w}'_k}^t \right) / \left(\sum_{\bar{w}'_k \in E^t} p_{\bar{w}'_k}^{t+1} \right) \ .$$
8. Update: $Pool \leftarrow Pool \bigcup E^t$.

Figure 2: The sleeping experts for phrases algorithm.

	# Errors			
			Experts	
Domain	Rocchio	Ripper	4-words	1-word
bonds	31	34	31	60
boxoffice	26	20	22	37
budget	170	159	161	171
burma	46	33	33	34
dukakis	107	112	98	107
hostages	212	206	185	214
ireland	106	97	86	87
nielsens	49	35	41	53
quayle	73	65	66	68
average	91.11	84.56	80.33	92.33

Table 1: Ripper, Sleeping Experts, and Rocchio's algorithm on AP titles with full sample

a new sampling method called *uncertainty sampling.* Following Lewis and Catlett, we will use error rate (as estimated by the number of errors made on the test set) as our principle evaluation metric. However, rather than using subsamples, we will train our learning systems on the entire training set.

Table 1 summarizes performance of the learning systems on this task. Performance for sleeping experts is given with both four word phrases and single word phrases; we provide numbers for single word phrases as a second example of a linear, context-insensitive classifier.

On average, both Ripper and sleeping experts with four word phrases achieve lower error rates than the linear classifiers. Ripper achieves a lower error rate than Rocchio, the better of the two linear classifiers, on seven of nine categories, and has a higher error rate only twice. Using four word phrases, sleeping experts achieves a lower error rate than Rocchio on eight of the nine categories, and achieves an identical error rate once.

Table 2 gives some additional points of reference on this benchmark. Lewis and Gale [1994] used the F-measure as the principle evaluation metric;[3] to compare with these results, we also record average F-measure in the table, as well as the widely used measurements of precision and recall.

The first rows of the table show the average performance of Rocchio's algorithm, the probabilistic classifier used by Lewis and Gale, the decision tree learner C4.5 using a sample of 10,000 examples (from Lewis and Catlett), and Ripper with negative word tests allowed (*i.e.*, Ripper, allowing tests of the form $e \notin S$.) Again, sleeping experts with single word phrases is also included as an additional example of a

[3] The F-measure is defined as [Van Rijsbergen, 1979, pages 168–176]
$$F_\beta \equiv \frac{(\beta^2+1) precision \cdot recall}{\beta^2 precision + recall}$$
where β is a parameter that controls the importance given to precision relative to recall; a value of $\beta = 1$ corresponds to equal weighting of precision and recall, with higher scores indicating better performance.

Learner	#errors	recall	precision	$F_{\beta=1}$
Rocchio	91.11	44.23	77.39	0.56
Probabilistic classifier				0.41
C4.5 (subsample)	100.9	42.75	66.94	0.50
RIPPER(negative tests)	86.00	60.12	72.26	0.64
RIPPER	84.56	51.43	74.46	0.59
Experts (4-words)	80.33	49.78	81.41	0.60
Experts (1-word)	92.33	42.61	67.32	0.52

Table 2: Additional comparisons for the full AP titles benchmark. All measurements are averages across nine categories.

	# Errors			
			Experts	
Domain	Rocchio	RIPPER	4-words	1-word
aparts	202	202	202	202
bonds	74	81	68	111
boxoffice	46	36	35	57
britx	874	864	819	859
budget	356	365	412	442
burma	43	37	37	37
bush	1361	1362	1230	1349
dollargold	204	52	56	141
dukakis	20	21	16	20
german	1021	962	880	942
gulf	2852	2804	2842	2843
hostages	299	350	345	348
ireland	117	123	111	111
israel	870	625	617	622
japan	1136	742	700	865
nielsens	63	62	70	71
quayle	51	44	48	52
tickertalk	35	35	34	34
weather	182	254	163	319
yugoslavia	155	111	108	111
average	498.1	456.6	439.65	476.8

Table 3: RIPPER, Sleeping Experts, and Rocchio's algorithm on AP titles with TREC-AP sample

linear classifier.

The results of this broader comparison are qualitatively the same. Notice that none of the algorithms considered here (including Lewis') explicitly attempt to maximize F-measure—instead all attempt to minimize error rate. However, RIPPER and sleeping experts for phrases also outperform all the linear classifiers with respect to the F-measure at $\beta = 1$. In fact, sleeping experts with four word phrases, the best of the new learning methods with respect to error rates, strictly dominates Rocchio on all four performance metrics.

3.2 The TREC-AP corpus

In a recent paper Lewis *et al.* [1996] describe experiments with a slightly different version of the AP titles corpus. This dataset is somewhat smaller, has more categories, and is more readily available to other researchers, being a subset of the AP stories used for the TREC conferences. A final difference is that the split into training and testing sets was done chronologically, rather than randomly. Thus, the test set is drawn from a somewhat different distribution than the training set.

We ran RIPPER, Rocchio's algorithm, and sleeping experts on these problems as well. The results are shown in Table 3. The results are broadly similar, except that sleeping experts performance is even stronger, relative to the other algorithms. We conjecture that this difference is due to the fact that sleeping experts, an on-line algorithm, can adapt to distributions that vary over time, whereas the batch algorithms Rocchio and RIPPER cannot adapt in this manner. We hope to investigate this issue more fully in the near future.

Learner	#errors	recall	precision	$F_{\beta=1}$
Rocchio (for error)	498.1	28.7	72.0	0.41
Rocchio (for $F_{\beta=1}$)				0.52
Widrow-Hoff				0.58
EG				0.55
RIPPER	456.6	56.4	78.5	0.66
Experts (4-words)	439.7	57.1	80.2	0.67
Experts (1-word)	476.8	30.0	72.1	0.43

Table 4: Additional comparisons for the TREC AP titles benchmark. All measurements are averages across 20 categories.

A comparison to the more recent results of Lewis *et al.* can be found in Table 4, which indicates the average performance of RIPPER, sleeping experts, our implementation of Rocchio's algorithm, and also three additional linear classifiers: Widrow-Hoff; EG, another on-line scheme that uses multiplicative updates; and a version of Rocchio that uses a threshold chosen to optimize the F measure at $\beta = 1$. (Recall that our implementation of Rocchio chooses a threshold so as to minimize error rate.)

3.3 The Reuters-22173 corpus

3.3.1 Experiments

Another set of experiments were conducted on the Reuters-22173 dataset [Lewis, 1992], a corpus of 22,173 news stories which have been tagged with 135 different topics. Here we largely followed the methodology of Lewis and Ringuette [1994]. The documents were first split into training and test sets as described by Lewis [1992], discarding 723 stories used as test data in an earlier experiment. This resulted in a corpus of 15674 training cases and 7096 test cases. All words were converted to lower case, punctuation marks were removed, and "function words" from a standard stop-list were removed.[4]

Compared to the AP titles corpus, this corpus has fewer examples, more categories, and longer documents. The stories in the Reuters-22173 corpus average some 78 words in length, not including stopwords. The vocabulary size is roughly comparable, with 37,141 words appearing in the training corpus.

To evaluate performance, precision and recall were used. These measurements were *micro-averaged*; in micro-averaging, the total number of false positive, false negative, true positive, and true negative predictions across all categories is computed, and these totals are used to compute recall and precision. Performance was further summarized by a *break-even point*—a hypothetical point, obtained by interpolation, at which precision equals recall.

Table 5 summarizes these "micro-averaged break-even" points for sleeping experts, with three word phrases, two word phrases, and single word phrases; RIPPER, with and

[4] "Function words" include high frequency by contentless words like "of" and "the". We used the stop-list given by Lewis [Lewis, 1992].

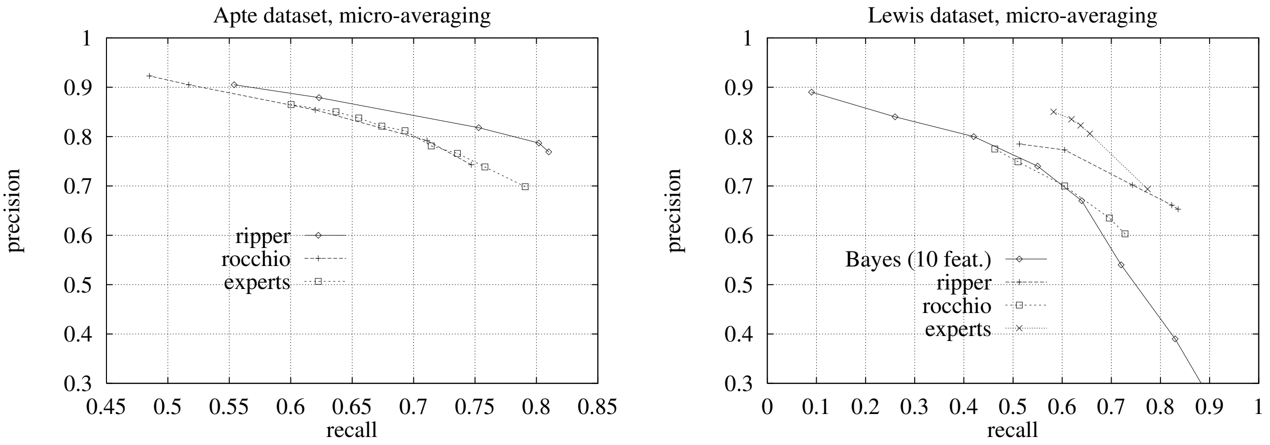

Figure 3: Micro-averaged performance on Reuters-22173

Dataset	Learner	Options	Micro-Averaged Break-Even
Lewis dataset	Experts	3-words	0.753
	Experts	2-words	0.737
	Ripper	(negative tests)	0.719
	Ripper		0.716
	decision tree	90 boolean feat.	0.670
	Rocchio		0.660
	Experts	1-word	0.656
	prop. Bayes	10 boolean feat.	0.650
Apte *et al.* dataset	Ripper	(negative tests)	0.796
	Ripper		0.793
	SWAP-1	80-100 frequency feat.	0.789
	SWAP-1	80-100 boolean feat.	0.785
	Experts	3-words	0.759
	Experts	2-words	0.753
	Rocchio		0.745
	Experts	1-word	0.647

Table 5: Break-even summaries for Reuters-22173

without negative tests; Rocchio; a simple decision tree learning system; and a Bayesian classifier. The last two figures are from Lewis and Ringuette [1994].

As an additional point of comparison we also show the results of duplicating the experiments conducted by Apte *et al.* [1994] on the same corpus. The main difference is that Apte *et al.* discarded stories that were tagged with no categories, resulting in a smaller corpus of 10,668 training cases and 3,685 test cases. (In Lewis's experiments, these documents were treated as negative examples of all categories.) Additionally, only 93 of the more frequent classes were considered. For this dataset, we show results for Ripper; SWAP-1, the learning system used by Apte *et al.*; sleeping experts; and Rocchio. The different numbers for SWAP-1 reflect different methods for representing text. In each of these methods a document was represented by a relatively small number of features (between 80-100, depending on the category), each of which corresponds to a specific word. Generally, features corresponded to words that have high mutual information with a category, and features were either numbers indicating the frequency of a word in a document, or boolean variables indicating presence of a word.

Figure 3 displays the same results graphically for both some of the learning systems. For the Lewis dataset we show results for sleeping experts with two word phrases, Ripper with negative tests, Rocchio, and Lewis' Bayesian classifier. For the Apte *et al* dataset, we show a micro-averaged precision *vs.* recall curve for Ripper with negated tests (the best performing version); SWAP-1, using the best representation that did not treat words in the headline specially (see below); sleeping experts with two word phrases; and Rocchio.

3.3.2 Discussion

Previous work. On this corpora, a number of the learning algorithms of Table 5 make use of context. The hypotheses of SWAP-1, for instance, are extremely similar to those of Ripper, and decision trees also use the same notion of context as Ripper. (In fact, the decision trees used by Lewis and Ringuette can be converted into a ruleset with negation.) Neural networks were also used on the Reuters-22173 corpus to learn nonlinear classifiers using a variety of representations by Wiener *et al.* [1995]. As shown in Table 5, the learning algorithms that use context are uniformly better than those which do not.

While there has been a long history of applying learning algorithms that use context to this dataset, one contribution of the experiments of this paper is that unlike Ripper and sleeping experts, none of these earlier systems used a direct representation for text—*i.e.*, rather than representing documents internally as a list of tokens, these algorithms used some non-trivial process to convert the original documents into feature vectors. In some cases this process was quite expensive in terms of storage—for example, Wiener *et al.* note that one of their representation schemes required a different 200-dimensional vector representations for each document and for each topic. Clearly using a direct representation is more efficient for corpora with many categories.

Reducing dimensionality of the examples also may degrade generalization performance. For instance, although Lewis and Ringuette report that on the Reuters-22173 problem, performance improved monotonically as more features were added to the decision tree learner, both Lewis and Ringuette's decision tree learner and SWAP-1 limited the number of features to around 100, presumably for efficiency reasons. We conjecture that the ability of RIPPER and sleeping experts to use a much larger vocabulary contributes their improved performance. We note that both RIPPER and sleeping experts are reasonably efficient, even with large vocabularies: learning time for RIPPER averages a little over five minutes per category on the larger Lewis dataset, and learning time for sleeping experts averages less than one minute when single word phrases are used as experts and twenty five minutes with four word phrases.[5] There are 37,141 different experts representing single word phrases, more than half a million of two word phrases, and more than 10 million three word phrases. However, most (over 99%) of the three word phrases occur only once in the corpus and do not have any contribution to the classification. Therefore, we only keep information of experts that correspond to phrases that appear at least twice in the corpus. Using this simple technique, the memory requirements for storing the experts are kept modest and the amount of memory used for maintaining a pool of n word phrases is only two times bigger than the amount needed for a pool of $n-1$ word phrases.

Special notions of context. SWAP-1's performance was slightly improved by using a special representation in which the frequency associated with a word was adjusted according to where the word appeared in the document; specifically, the frequency counts of words appearing in a title were increased by double-counting each occurrence in the title. Using this representation, the micro-averaged break-even point was improved to from 0.789 to 0.805.

This representational trick can be viewed as making use of a domain-dependent notion of context, namely that the "context" represented by the headline of a document is of special importance. We repeated experiments with RIPPER (with negative tests) on the Apte *et al.* sample using a representation in which every document was represented by two set-valued features: the set of all words appearing in the story, and the set of words appearing in the headline. This representation allows RIPPER to construct rulesets which explicitly require a word to appear in a headline context. Not surprisingly (in light of the previous results) this representational trick improves performance: RIPPER's micro-averaged break-even improves from 0.796 to 0.809.

Methodology. We note that Table 5 indicates that there is a substantial difference between the difficulty of the categories in the datasets used by Apte *at al.* and those used by Lewis and Ringuette; for instance, the micro-averaged break-even point of Rocchio's algorithm is 0.745 for the Apte *et al.* datasets, but only 0.660 for the Lewis and Ringuette datasets. These differences are unsurprising in light of the differences in the number of examples and the number of categories, particularly since the categories and examples were restricted systematically, rather than by random sampling. Wiener *et al.* [1995] describe another series of experiments using the same corpus but (apparently) a third split into training and testing data, resulting in a corpus of 9,610 training examples and 3,662 test examples. While it is by no means clear which of these datasets contains more representative learning problems, it seems certain that performance comparisons on different datasets are at best approximate.

Finally, although micro-averaged break-even has been a widely used performance test on this dataset, it has some well-known drawbacks as a metric—in particular, micro-averaged measurements tend to be dominated by the most frequent categories. Macro-averaged measurements have a complementary disadvantage for datasets like Reuters-22173, which contain many very rare categories: they tend to be dominated by performance on very rare categories, where performance is hardest to estimate from test data. In later work we hope to examine performance on individual categories more closely, in the hope of clarifying the question of which circumstances are most favorable for which learning algorithms.

[5] These times are on a MIPS Irix 5 with 150 Mhz R4400 processors.

4 Concluding remarks

To summarize, we have evaluated two new text categorization algorithms. Both algorithms allow the "context" of a word w to influence how the presence or absence of w will contribute to a classification. However, the algorithms use different notions of context, different representations for a classifier, and different search methods to find a classifier.

We performed experiments with two large text categorization benchmarks: a corpus of AP titles, and a corpus of news stories. We found that the context sensitive learning algorithms generally performed better algorithms that learn context insensitive linear classifiers. Our rule-learning results improve on previous methods that output boolean text classifiers, both in quality of the results (on the Reuters corpus) and in the scale of problems that are demonstrated to be practically solvable (on the AP titles corpus).

On the AP titles, both algorithms achieved lower error rates than Rocchio's algorithm; in one case, the error rates were almost uniformly lower. On the more complete version of the Reuters corpus, the two algorithms performed better than any comparable algorithms previously applied to the corpus. On a restricted version of the corpus, the algorithms again performed better than any previously applied linear classifier. We view this result as a confirmation of the usefulness and practicality of learning classifiers that represent contextual information.

Acknowledgements

The authors would like to thank David Lewis for comments on a draft of the paper, and also for help in preparing data for the experiments.

References

(Apté *et al.*, 1994) Chidanand Apté, Fred Damerau, and Sholom M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12(3):233–251, 1994.

(Blum, 1990) Avrim Blum. Learning boolean functions in a infinite attribute space. In *22nd Annual Symposium on the Theory of Computing*. ACM Press, 1990.

(Blum, 1995) Avrim Blum. Empirical support for WINNOW and weighted majority algorithms: results on a calendar scheduling domain. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

(Buckley *et al.*, 1994) C. Buckley, G. Salton, and J. Allan. The effect of adding relevance information in a relevance feedback environment. In *17'th Int'l ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994.

(Cesa-Bianchi *et al.*, 1993) Nicolò Cesa-Bianchi, Yoav Freund, David P. Helmbold, David Haussler, Robert E.

Schapire, and Manfred K. Warmuth. How to use expert advice. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 382–391, May 1993. Submitted to the Journal of the ACM.

(Church and Gale, 1995) K. W. Church and W. A. Gale. Poisson mixtures. *Natural Language Engineering*, 1(2):163–190, 1995.

(Cohen and Singer, 1996) William W. Cohen and Yoram Singer. Learning to query the Web. In *Proceedings of AAAI-96 Workshop on Internet-Based Information Systems*, 1996.

(Cohen, 1995a) William W. Cohen. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

(Cohen, 1995b) William W. Cohen. Text categorization and relational learning. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

(Cohen, 1996) William W. Cohen. Learning with set-valued features. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, 1996.

(Freund and Schapire, 1995) Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37. Springer-Verlag, 1995. A long version will appear in JCSS.

(Ittner *et al.*, 1995) David J. Ittner, David D. Lewis, and David D. Ahn. Text categorization of low quality images. In *Symposium on Document Analysis and Information Retrieval*, pages 301–315, Las Vegas, NV, 1995. ISRI; Univ. of Nevada, Las Vegas.

(Kivinen and Warmuth, 1994) J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. Technical Report UCSC-CRL-94-16, University of California, Santa Cruz, Computer Research Laboratory, June 1994. Revised December 7, 1995. An extended abstract to appeared in the STOC 95, pp. 209-218.

(Lewis and Catlett, 1994) David Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Machine Learning: Proceedings of the Eleventh Annual Conference*, New Brunswick, New Jersey, 1994. Morgan Kaufmann.

(Lewis and Gale, 1994) David Lewis and William Gale. Training text classifiers by uncertainty sampling. In *Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994.

(Lewis and Ringuette, 1994) David Lewis and Mark Ringuette. A comparison of two learning algorithms for text categorization. In *Symposium on Document Analysis and Information Retrieval*, Las Vegas, Nevada, 1994.

(Lewis *et al.*, 1996) David Lewis, Robert Schapire, James P. Callan, and Ron Papka. Training algorithms for linear classifiers. In *Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Zurich, Switzerland, 1996.

(Lewis, 1992) David Lewis. Representation and learning in information retrieval. Technical Report 91-93, Computer Science Dept., University of Massachusetts at Amherst, 1992. PhD Thesis.

(Littlestone and Warmuth, 1994) Nick Littlestone and Manfred Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

(Rocchio, 1971) J. Rocchio. Relevance feedback information retrieval. In Gerard Salton, editor, *The Smart retrieval system—experiments in automatic document processing*, pages 313–323. Prentice-Hall, Englewood Cliffs, NJ, 1971.

(Salton, 1991) Gerard Salton. Developments in automatic text retrieval. *Science*, 253:974–980, 1991.

(Van Rijsbergen, 1979) C. J. Van Rijsbergen. *Information Retrieval*. Butterworth, London, second edition, 1979.

(Vovk, 1990) V. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383. Morgan Kaufmann, 1990.

(Wiener *et al.*, 1995) E. Wiener, J. O. Pederson, and A. S. Wiegend. A neural network approach to topic spotting. In *Symposium on Document Analysis and Information Retrieval*, pages 317–332, Las Vegas, Nevada, 1995.