Context-sensitive learning methods for text categorization

William W. Cohen and Yoram Singer
AT&T Labs
180 Park Avenue
Florham Park, NJ 07932
{wcohen,singer}@research.att.com

April 21, 1998

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, 1992b; Lewis and Ringuette, 1994; Lewis and Gale, 1994; Apté *et al.*, 1994b; Yang and Chute, 1994; Hull *et al.*, 1995; 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, 1996b], and *sleeping-experts*, a new on-line learning method [Freund *et al.*, 1997].

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; in particular, 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. Many common text categorization schemes, such as "naive" Bayes and Rocchio's algorithm, produce linear classifiers, in which the features correspond to individual terms—typically words or word stems. In a linear classifier, the presence or absence of a word w in a document d will have the same influence on the predicted class of d regardless of other words in the document; thus a linear classifier effectively assumes that the "context" in which the word w appears (as encoded by the other words present in the document d) has no effect on the meaning of w. This assumption is obviously unrealistic, and one might hope that performance could be improved by relaxing it.

One way to relax this assumption is to build a linear classifier that uses more complex features—for example, features that test for the occurrence of a word w together with some enclosing context. The principle technical challenge here is finding useful complex features, given the potentially enormous space of potential features. One of the algorithms we investigate, sleeping-experts, is an algorithm of this type. A second way of introducing context is to learn a non-linear classifier. The second algorithm we investigate, RIPPER, learns a non-linear classifier in the form of a boolean combination of simple terms; alternatively, RIPPER can be thought of as learning a disjunction of "contexts", each context defined by a conjunction of simple terms. The principle technical challenge when this approach is followed is to learn these non-linear classifiers efficiently.

A final commonality of sleeping-experts and RIPPER is that 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 for RIPPER 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* to represent context. A sparse phrase is a sequence of nearby, but not necessarily consecutive words; hence the context of a word for sleeping-experts consists of *nearby* words, in a *fixed* order.[1]

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

[1]Of course, other types of context could be used in a learning algorithm; the notions of context adopted by RIPPER and sleeping-experts are just two among many alternatives.

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'.

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; Freund *et al.*, 1997] 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-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 describe the rule learning algorithm RIPPER. The classifier constructed by RIPPER is a set of rules, such as the one illustrated in Figure 1, which was learned for "ireland" category of the AP titles corpus (see Section 3.1 below.) This set of rules can be interpreted as a disjunction of conjunctions: for instance, a document d is considered to be in the category `ireland` if and only if

```
function IREP*(Data)
begin
  Data0 := copy(Data);
  RuleSet := an empty rule set
  while ∃ positive examples∈ Data0 do
    /* grow and prune a new rule */
    split Data into GrowData,PruneData
    Rule := GrowRule(GrowData)
    Rule := PruneRule(Rule,PruneData)
    add Rule to RuleSet
    remove examples covered by
      Rule from Data0
    /* check stopping condition */
    if DL(RuleSet)>DL(RuleSet_opt) + d
       where RuleSet_opt has lowest DL
       of any RuleSet constructed so far
    then
       RuleSet := Compress(RuleSet,Data0)
       return RuleSet
    endif
  endwhile
  RuleSet := Compress(RuleSet,Data0)
  return RuleSet
end
```

```
function Optimize(RuleSet,Data)
begin
    for each rule c ∈ RuleSet do
        split Data into GrowData,PruneData
        c' := GrowRule(GrowData)
        c' := PruneRule(c',Prunedata)
          guided by error of RuleSet-c+c'
        ĉ := GrowRuleFrom(c,GrowData)
        ĉ := PruneRule(ĉ,Prunedata);
          guided by error of RuleSet-c+ĉ
        replace c in RuleSet with best of c, c', ĉ
          guided by DL(Compress(RuleSet-c+x))
    endfor
    return RuleSet
end


function RIPPER(Data)
begin
    RuleSet := IREP*(Data)
    repeat 2 times:
        RuleSet := Optimize(Ruleset,Data);
        UncovData := examples in Data not
          covered by rules in RuleSet
        RuleSet := RuleSet + IREP*(UncovData)
    endrepeat
end
```

Figure 2: The RIPPER algorithm

(the word `ireland` appears in d) OR
(the word `ira` appears in d AND the word `killed` appears in d) OR
⋮
(the word `ira` appears in d AND the word `shot` appears in d) OR
(the word `ira` appears in d AND the 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, 1996b].

There are a number of subtleties involved in learning rulesets. In particular, relatively straightforward greedy algorithms often give rulesets with unnecessarily high error rates; furthermore, many algorithms that do find accurate rulesets tend to be relatively inefficient for large noisy datasets. Because of these problems, the algorithm used in RIPPER is relatively complex. A more detailed description of RIPPER and motivation for some of the details of the algorithm can be found elsewhere [Cohen, 1995a]; here, we will simply summarize the algorithm. We will restrict ourselves to problems with two classes. In this case all rules learned by RIPPER have an identical consequent (the name of the positive class).

The algorithm used by RIPPER is summarized in Figure 2, and consists of two main stages. The first stage is a greedy process which constructs an initial ruleset. This stage is based on an earlier rule-learning algorithm called *incremental reduced error pruning* (IREP) [Fürnkranz and Widmer, 1994], which in turn is based on earlier work due to Quinlan [1990], Cohen [1993], Brunk and Pazzani [1991], and Pagallo and Haussler [1990]. The second stage is an "optimization" phase which attempts to further improve the compactness and accuracy of the ruleset.

Stage 1: Building an initial ruleset. The first stage of RIPPER is a variant of IREP that we call IREP$*$. IREP$*$ is a "set-covering" algorithm: it constructs one rule at a time, and removes all examples covered by a new rule[2] as soon as the rule is constructed. The heuristics used in constructing a rule are intended to ensure that the rule covers many positive examples and few negative examples.

To construct a rule, the uncovered examples are randomly partitioned into two subsets, a *growing set* containing 2/3 of the examples and a *pruning set* containing the remaining 1/3. IREP$*$ will first *grow* a rule, and then simplify or *prune* the rule.

A rule is "grown" by repeatedly adding conditions to rule r_0 with an empty antecedent. This is done is a greedy fashion: at each stage i, a single condition is added to the rule r_i, producing a longer and more specialized rule r_{i+1}. The condition added is the one that yields the largest *information gain* for r_{i+1} relative to r_i [Quinlan, 1990]. Information gain is defined as

$$Gain(r_{i+1}, r_i) \equiv T^+_{i+1} \cdot \left(-\log_2 \frac{T^+_i}{T^+_i + T^-_i} \;+\; \log_2 \frac{T^+_{i+1}}{T^+_{i+1} + T^-_{i+1}} \right)$$

where T^+_j (respectively T^-_j) is the number of positive (negative) examples in the growing set covered by rule r_j. Information gain rewards rules r_{i+1} that increase the density of positive examples covered by the rule, without greatly reducing the total number of covered positive examples. The greedy addition of new literals continues until the clause covers no negative examples in the growing set, or until no condition has a positive information gain.

After growing a rule, the rule is pruned (*i.e.*, simplified). This is another greedy process. At each stage, IREP$*$ considers deleting any final sequence of conditions from the rule, and chooses the deletion that maximizes the function

$$f(r_i) = \frac{U^+_{i+1} - U^-_{i+1}}{U^+_{i+1} + U^-_{i+1}}$$

where U^+_{i+1} (respectively U^-_{i+1}) is the number of positive (negative) examples in the pruning set covered by the new rule. After pruning, the pruned clause is added to the rule set, and the examples covered by it are removed.

When to stop adding rules. The requirement that information gain is non-zero means that every rule must cover some positive example, guaranteeing that IREP$*$ will eventually terminate. However, if the data is noisy, it is possible for many rules to be constructed, each of which covers only a few examples; this can be computationally expensive for large datasets. IREP$*$ thus includes an additional heuristic which attempts to determine if the current ruleset is already "large enough" given the training data. This stopping criterion is based on a minimum description length (MDL) heuristic.

[2] An example is "covered by a rule" if it satisfies the rule's antecedent.

MDL heuristics are based on the assumption that the best model of a given set of data is the model that allows one to most succinctly encode the data. Generally, the encoding of data is done in two parts: first the model is encoded, and then the errors made on by the model on the data are encoded. The preferred model is one with the smallest *description length* (the number of bits required for this two-part encoding). To encode the errors made by a model, RIPPER uses an encoding scheme proposed by Quinlan [1995]. A variant of this scheme is also used to encode rules.[3]

In IREP$*$, after every rule is added to the ruleset, the total *description length* of the current ruleset and the examples is computed. IREP$*$ stops adding rules when this description length is more than d bits larger than the smallest description length obtained so far, or when there are no more positive examples.[4] The ruleset is then *compressed* by examining each rule in turn, starting with the last rule added, and deleting any rules that increase the description length.

Stage 2: "optimizing" a ruleset. After RIPPER stops adding rules, the ruleset is "optimized" so as to further reduce its size and improve its accuracy. Rules are considered in turn in the order in which they were added. For each rule r, two alternative rules are constructed. The *replacement for* r is formed by growing and then pruning a rule r', where pruning is guided so as to minimize error of the entire ruleset (with r' replacing r) on the pruning data. The *revision of* r is formed analogously, except that it is grown by greedily adding literals to r, instead of to the empty rule. Finally a decision is made as to whether the final theory should include the revised rule, the replacement rule, or the original rule. This decision is made using the description length heuristic—the definition with the smallest description length after compression is preferred.

After optimization, the definition may cover fewer positive examples; thus IREP$*$ is called again on the uncovered positive examples, and any additional rules that it generates are added.

This optimization step can be repeated, occasionally resulting in further improvements in a ruleset. Experiments with a large collection of propositional learning benchmark problems indicates that two rounds of optimization is usually sufficient, and so in RIPPER, this optimization step is by default repeated twice.

It should be emphasized that while RIPPER is a fairly complex algorithm, the algorithm and all parameters were fixed before any of these experiments were conducted. The primary set of benchmarks used in developing the algorithm and setting its parameters was a set of non-textual classification problems taken from the UC/Irvine Repository [Cohen, 1995a].

2.1.2 Extensions to RIPPER motivated by text

Before running these experiments, RIPPER was modified so as to be more appropriate for text categorization problems. One extension 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

[3]In Quinlan's scheme, $-\log_2 p \cdot e + -\log_2 (1-p) \cdot (n-e)$ bits are used to encode a subset of e elements from a set of size n, where p represents the expected value of the fraction e/n. We encode a rule by first encoding the length k of the rule's antecedent, and then encoding the antecedent as a k-element subset of the set of all possible conditions, with $p = k/n$.

[4]By default $d = 64$.

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 appropriately changing the weights given to false positive errors and false negative errors in the pruning and optimization stages of the learning algorithm.

A second extension to RIPPER was motivated by the large number of features typically available in 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 boolean feature b_i for each word w_i appearing in the corpus, letting b_i be true for example x iff w_i appears in the corresponding document. A corpus with m documents and n words would thus be represented as a $m \times n$ matrix.

For text classification, this representation is very inefficient in its usage of space; even a moderately large corpus will usually contain many different words, only a few of which appear in a any particular document. To avoid this problem, RIPPER was extended to allow the value of an attribute to be a set of symbols—for instance, a document can be represented with a single attribute, having as its value the set of words that appear in the document. The primitive tests on a set-valued attribute a (*i.e.*, the tests which are allowed in rules) are of the form "$w_i \in a$".

The implementation of this extension is explained briefly below, and more detail elsewhere [Cohen, 1996b]. When constructing a rule, RIPPER must find a single test that maximizes information gain; most of RIPPER's run-time is spent in this operation. For set-valued attributes that can be done in two steps. First, RIPPER iterates over the set of examples S that are covered by the current rule, recording the set of words W_S^a that appear as elements of attribute a for some example $x \in S$, and also recording for each word $w_i \in W_S^a$ two statistics: p_i, the number of times w_i appears in a positive example in S, and n_i, the number of times w_i appears in a negative example in S. Second, RIPPER iterates over the words in W_S^a, using these statistics to compute the gain for each possible test of the form $w_i \in a$. The entire process requires time linear in the size of S. Notice that all symbols w_i that appear as elements of attribute a for some training example will be considered by RIPPER.

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 formal reasons for being wary of using them [Cohen, 1996b]. In this paper we will typically present results for RIPPER with negative word tests forbidden.

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.

Alternatively, one could also use set-valued features to encode more sophisticated representations of a document, such as a set of word stems, or a set of words and a set of phrases. In this paper we have elected to use sets of unstemmed words, a word being any sequence of alphanumeric characters (ignoring case). This choice was made largely for the sake of simplicity, and is also consistent with our emphasis on using direct representations for text. However, it is likely that our results could be improved somewhat by using word stems instead of words; experimental results on retrieval problems show that stemming does lead to a slight gain in average performance [Hull and Grefenstette, 1996], and it is plausible to believe that this gain could also be achieved in classification.

For similar reasons, we have adopted an unusual approach to handling the large number of features available in text categorization problems: while previous researchers have generally relied on preprocessing to reduce the number of possible features, particularly when applying symbolic learning systems to text categorization, we have elected to extend the algorithms to directly handle large sparse feature sets directly, and use little feature selection. There are two reasons for this choice. One reason is that the evidence as to the effectiveness of feature selection is mixed; while in some contexts use of appropriate feature selection methods has improved generalization performance [Almuallim and Dietterich, 1991; John *et al.*, 1994], in other cases, performance is only degraded by feature selection [Lewis and Ringuette, 1994; Cohen, 1996b]. Hence it is important to be able to be able to work efficiently with large feature sets, if only so that one can explore a wide range of levels of feature selection. The second reason is that a good learning system should not only be accurate in its generalization performance, but also be easy to use. Even when feature selection does improve generalization performance, it makes application of a learning system somewhat more complex, if only because it introduces another set of parameters that must be set.

2.2 Sleeping-experts for phrases

2.2.1 Background

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 (see for instance [Littlestone, 1988; Littlestone and Warmuth, 1994; Kivinen and Warmuth, 1994; Freund *et al.*, 1997]). Prediction algorithms in this framework are given a pool of fixed, yet possibly infinite, "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 analyzed. In this paper we will use a *multiplicative* update method, in which weights for the individual experts are modified by multiplying them by a constant. (The algorithm we use also includes certain normalization steps, which we will describe shortly.)

Empirical evidence indicates that multiplicative update algorithms often outperform traditional learning techniques for linear classifiers [Blum, 1995; Lewis *et al.*, 1996]. More intriguingly, 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; Freund *et al.*, 1997]. In particular, certain multiplicative up-

Phrase	log-weight $\notin$ ireland	log-weight $\in$ ireland	number of occurrences $\notin$ ireland	number of occurrences $\in$ ireland
belfast	-7.19	12.05	8	31
haughey	-6.35	11.10	2	10
ira says	-1.07	10.44	2	7
northern ireland	-7.20	10.17	18	38
catholic man	-0.87	6.03	0	3
ulster	-3.98	5.20	4	8
killed ? ira	-0.09	4.68	1	4
protestant extremists claim	-0.12	4.59	0	2
moderate catholic	-0.02	4.58	0	2
ira supporters	-3.20	3.68	0	3
sinn fein	-3.52	3.38	2	5
west belfast	-5.90	3.05	3	16

Table 1: Experts with large weights for the category `ireland`.

date procedures can learn quickly even if the set of experts is huge, as long as a few of these experts are accurate; in contrast, formal bounds on the performance of more traditional additive update rules suggest that the number of experts must be small in order to guarantee good generalization performance.

These results suggest that multiplicative update procedures might perform well on text categorization problems, if they are provided with some appropriate large set of experts. Below we will describe an application of multiplicative update algorithms where the experts correspond roughly to all length-k phrases that occur in a corpus.

2.2.2 The sleeping-experts algorithm

The *sleeping-experts* algorithm is an update algorithm that is based on two recent formal 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 set of sparse phrases is exactly the set of words appearing in the corpus. Table 1 gives some examples of sparse phrases that were found useful for the "ireland" category of Figure 1.

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.[5] 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. 3.

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 $\bar{\omega}$ 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

[5] Other cost functions that associate a different utility value with each possible pair of the correct outcome and the outcome predicted by the algorithm can be implemented in the sleeping-experts framework. Such cost functions might result in a non-symmetric weight update, and are beyond the scope of this paper.

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

1. Receive a new document $\omega_1^t, \omega_2^t, \ldots, \omega_l^t$ and its classification c^t.

2. Define the set of active phrases:

$$W^t = \left\{ \bar{w} \,|\, \bar{w} = \omega_{i_1}^t \omega_{i_2}^t \ldots \omega_{i_j}^t, \; 1 \leq i_1 < i_2 < \ldots i_{j-1} < i_j \leq l, \; i_j - i_1 < n \right\}$$

3. Define the set of active mini-experts:

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

4. Initialize the weights of new mini-experts:

$$\forall \bar{w}_k \in E^t \; s.t. \; \bar{w}_k \notin Pool : \; p_{\bar{w}_k}^t = 1 \quad .$$

5. Classify the document as positive if

$$y^t = \frac{\sum_{\bar{w} \in W^t} p_{\bar{w}_1}^t}{\sum_{\bar{w} \in W^t} \sum_{k=0,1} p_{\bar{w}_k}^t} > \theta_C \quad .$$

6. Update weights:

$$l(\bar{w}_k) = \left\{ \begin{array}{ll} 0 & c^t = k \\ 1 & c^t \neq k \end{array} \right. \quad \Rightarrow \quad p_{\bar{w}_k}^{t+1} = p_{\bar{w}_k}^t \, \beta^{l(\bar{w}_k)} = \left\{ \begin{array}{ll} p_{\bar{w}_k}^t & c^t = k \\ \beta \, p_{\bar{w}_k}^t & c^t \neq k \end{array} \right.$$

7. Renormalize weights:

   (a) $Z_t = \sum_{\bar{w}'_k \in E^t} p_{\bar{w}'_k}^t$

   (b) $Z_{t+1} = \sum_{\bar{w}'_k \in E^t} p_{\bar{w}'_k}^{t+1}$

   (c) $p_{\bar{w}_k}^{t+1} = \dfrac{Z_t}{Z_{t+1}} p_{\bar{w}_k}^{t+1}$

8. Update: $Pool \leftarrow Pool \bigcup E^t$.

Figure 3: The sleeping-experts for phrases algorithm.

can be efficiently calculated by summing the weights[6] of mini-experts that predict 1

$$y^t = \sum_{\bar{w}_k \in E^t} \tilde{p}^t_{\bar{w}_k} k = \sum_{\bar{w} \in W^t} \tilde{p}^t_{\bar{w}_1} = \frac{\sum_{\bar{w} \in W^t} p^t_{\bar{w}_1}}{\sum_{\bar{w} \in W^t} \sum_{k=0,1} p^t_{\bar{w}_k}} .$$

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 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 increase the weights of the mini-experts which were correct.

As an illustration of the effect of the weight update procedure, Table 1 lists 12 experts for which the weight of their mini-experts predicting that the document belongs to the category `ireland` (from the AP titles corpus) is large. (The symbol `?` is a place-holder which stands for any word in a sparse n-gram.) For each expert we give the logarithm of the weight of its two mini-experts, and the number of positive (respectively negative) examples in which the phrase appears. For this category, the set of all sparse 4-grams was used to construct the set of mini-experts. There are a total of 1,813,632 sparse phrases, and hence $2 \cdot 1{,}813{,}632 = 3{,}647{,}264$ mini-experts. Like RIPPER, sleeping-experts is applied separately to each category; thus we built a separate pool of experts for each different classification problem.

To classify a new document d using this pool, one first finds all sparse n-grams appearing in the document, and then computes the weights of the corresponding mini-experts. For instance, in classifying the documents "`prayers said for soldiers killed in ira bombing`" and "`taxi driver killed by ira`" the relevant set of phrases would include "`killed ? ira`" and "`bombing`". The documents above are classified correctly; among the

[6]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.

mini-experts associated with these phrases, the total weight of the mini-experts predicting $d \in$ `ireland` is larger than the total weight of the mini-experts predicting $d \notin$ `ireland`.

2.2.3 Discussion

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. It should be emphasized that both the update and prediction steps depend only the *active* experts—the phrases appearing in the document. This means that the run-time of these steps depend only the size of the document being processed, not the total number of experts.[7]

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.

An analysis of the general sleeping-experts algorithm for various settings is provided by Freund *et al* [1997]. This analysis technique can be directly applied to our setting. 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. These results hold only in a pure on-line model, in which the weights are updated after each round; when the weight update is 'turned off', the assumptions used in deriving the cumulative loss bounds no longer hold. Nevertheless, the experiments described below indicate that the algorithm also performs well in a batch setting, in which weights are updated during a training phase, and then held fixed while a set of test cases is classified.

Finally, we note that many other types of experts could be constructed. For instance, an expert may record the number of appearances of a phrase in the current document, or previous documents, and then use an inverse document frequency or a mixture of Poisson models [Church and Gale, 1995] to predict classification. The focus of this paper is methods for exploiting contextual information, rather than frequency statistics, and thus we will

[7] We assume here that the "pool" is implemented as a hash table, so that weights can be updated in constant time.

restrict ourselves here to experts which are based only on the presence and absence of phrases (sparse n-gram). This also enables a fairer comparison with Ripper, whose rules are based only the presence or absence of words in documents.

2.3 Rocchio

We also 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 $\tilde{\mathbf{v}}^c$ of the same dimension as the original weight vectors $\mathbf{v}^1, \ldots, \mathbf{v}^{N_D}$. 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; following Ittner *et al.* and others [Buckley *et al.*, 1994], we use the values $\beta = 16$ and $\gamma = 4$. Documents to be classified are first converted into weight vectors, and then compared against the prototype $\tilde{\mathbf{v}}^c$ 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.

In the experiments described below, we will compare our results to previously published results whenever possible. We will use Rocchio as an additional reference point, supplementing these previous results; this allows us compare the performance of Ripper and sleeping-experts with a well-understood existing algorithm under precisely the same experimental conditions.

Title	Class
george burns back on the best seller list	aparts
bond prices rise	bonds
war of the roses wins box office battle for first place	boxoffice
deutsche bank controls percent of morgan grenfell	britx
cheney gets billion to spend in	budget
burmese troops seize major rebel base on thai frontier	burma
park returns to playground status after bush leaves anti drug rally	bush
dollar little changed gold higher	dollargold
dukakis adopts bush strategy executive experience counts	dukakis
west german mobile phone license awarded to international consortium	german
paulson says bid for gulfstream to be ready soon	gulf
iran said advising friends in lebanon to release hostages	hostages
belfast police find explosives arrest six	ireland
palestinians mark second anniversary of rebellion	israel
superpower detente won't stop rise in japan's defense outlays	japan
bill cosby leads nbc to another ratings victory	nielsens
quayle tells soviets they may like star wars after all	quayle
stock analysts see not so nifty	tickertalk
yugoslavia consumed by crisis	yugoslavia
snow hits arkansas plains	weather

Table 2: Titles from the TREC-AP corpus.

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.[8] Examples of titles from this corpus is given in Table 2.

In previous work, Lewis and Gale [1994] and Lewis and Catlett [1994] used this domain to evaluate the performance 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 3 summarizes performance of the learning systems on this task. Performance for

[8]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.

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

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. In both cases the improvement in error rate is statistically significant using a two-tailed paired t test.[9]

Table 4 gives some additional points of reference on this benchmark. Lewis and Gale [1994] used the F-measure as the principle evaluation metric;[10] 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 (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 linear classifier. C4.5 contains no special mechanism to handle the large feature sets encountered in IR problems, and was therefore run on a relatively small subsample of 10,000 examples; this probably accounts for its relatively poor performance.

The results of this broader comparison are qualitatively the same. Notice that none of the

[9] For RIPPER, $t = 2.97$ and $p > 98\%$; For sleeping-experts, $t = 3.92$ and $p > 99\%$.

[10] 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 4: Additional comparisons for the full AP titles benchmark. All measurements are averages across nine categories.

algorithms considered here 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 (see for instance [Lewis and Gale, 1994; Lewis and Catlett, 1994]). 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 5. (The column labeled "frequency" will be discussed in Section 3.5.1.) The results are broadly similar, except that the performance of sleeping-experts is even stronger, relative to the other algorithms. Sleeping-experts is again statistically significantly better than Rocchio, using a two-tailed test; however, Ripper outperforms Rocchio with confidence of only 90% on a paired t-test.[11] A comparison to the more recent results of Lewis *et al.* can be found in Table 6, which indicates the average performance of Ripper, sleeping-experts, our implementation of Rocchio's algorithm, and also three additional linear classifiers (the first two tested by Lewis *et al.* [1996]): Widrow-Hoff; Exponentiated Gradient (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.)

[11] For sleeping-experts, $t = 2.25$, $p > 95\%$, and for Ripper, $t = 1.72$, $p > 90\%$.

		# errors			
				Experts	
Domain	Frequency	Rocchio	RIPPER	4-words	1-word
aparts	790	202	202	202	202
bonds	387	74	81	68	111
boxoffice	170	46	36	35	57
britx	3515	874	864	819	859
budget	863	356	365	412	442
burma	434	43	37	37	37
bush	3921	1361	1362	1230	1349
dollargold	1614	204	52	56	141
dukakis	736	20	21	16	20
german	2392	1021	962	880	942
gulf	3471	2852	2804	2842	2843
hostages	916	299	350	345	348
ireland	475	117	123	111	111
israel	3659	870	625	617	622
japan	4337	1136	742	700	865
nielsens	248	63	62	70	71
quayle	513	51	44	48	52
tickertalk	123	35	35	34	34
weather	1824	182	254	163	319
yugoslavia	576	155	111	108	111
average	1548.2	498.1	456.6	439.65	476.8

Table 5: RIPPER, Sleeping-Experts, and Rocchio's algorithm on AP titles with TREC-AP sample

3.3 The Reuters-22173 Corpus

3.3.1 Experiments

Another set of experiments were conducted on the Reuters-22173 dataset [Lewis, 1992b], 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 [1992b], discarding 723 stories used as test data in an earlier experiment. This resulted in a corpus of 14704 training cases and 6746 test cases. All words were converted to lower case, punctuation marks were removed, and "function words" from a standard stop-list were removed.[12] Example of an article from the Reuters-22173 corpus before and after pre-processing is given in Figure 4.

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

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

```
<DATE>19-MAR-1987 06:21:45.34</DATE>
<TOPICS><D>earn</D></TOPICS>
<PLACES><D>uk</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;F
&#22;&#22;&#1;f0370&#31;reute
b f BC-BRITOIL-PLC-&lt;BTOL.L>   03-19 0045</UNKNOWN>
<TEXT>&#2;
<TITLE>BRITOIL PLC &lt;BTOL.L> 1986 YR</TITLE>
<DATELINE>    LONDON, March 19 -
    </DATELINE><BODY>Shr 6.56p vs 50.31p
    Final div 6p, making 8p vs 13p.
    Pre-tax profit 134 mln stg vs 759 mln.
    Net profit 33 mln vs 253 mln.
    Turnover 978 mln stg vs 1.80 billion.
    Extraordinary debit 50 mln vs nil.
    Operating profit 149 mln stg vs 756 mln.
    Exceptional debit on rationalisation programme 12 mln vs
nil
    Petroleum Revenue Taxes 77 mln vs 319 mln,
    U.K. Corporation tax and overseas tax 24 mln vs 187 mln,
    Note - The net effect of accounting changes in 1986 was to
reduce after tax profits by 47 mln stg. Retained earnings for
prior years were increased by 209 mln.
    Extraordinary debit of 50 mln stg related to the decision
to seek a buyer for the company's U.S. Assets.
 REUTER
&#3;</BODY></TEXT>
```

```
britoil plc btol yr london march shr vs final div making vs pre tax profit
mln stg vs mln net profit mln vs mln turnover mln stg vs billion
extraordinary debit mln vs nil operating profit mln stg vs mln exceptional
debit rationalisation programme mln vs nil petroleum revenue taxes mln vs
mln corporation tax overseas tax mln vs mln note net effect accounting
changes reduce tax profits mln stg retained earnings prior years increased
mln extraordinary debit mln stg related decision seek buyer company assets
```

Figure 4: Example of an article from Reuters-22173 before and after pre-processing.

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

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 7 summarizes these "micro-averaged break-even" points for sleeping-experts, with three word phrases, two word phrases, and single word phrases; RIPPER, with and without negative tests; Rocchio; a simple decision tree learning system; and a Bayesian classifier. (Using four word phrases with sleeping-experts provided no additional improvement over three word phrases on these problems.) 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.* [1994b] 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 10,667 training cases and 3,680 test cases.[13] Additionally, only 93 classes were considered—those appearing in at least one document in the reduced corpus. 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.

3.3.2 Discussion

Previous work. On this corpus, a number of the learning algorithms of Table 7 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 negative tests.) As shown in Table 7, the learning algorithms that use context are uniformly better than those which

[13]To be precise, we discarded from the dataset all documents which were tagged with no topics, legal or otherwise. In particular, documents tagged with the topic word "bypass" were included. Note that in Lewis's experiments, these documents were treated as negative examples of all categories.

Dataset	Learner	Options	Micro-Averaged Break-Even
Lewis dataset	Experts	3-words	0.753
	Experts	2-words	0.737
	Ripper		0.683
	Ripper	negative tests	0.677
	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 + headlines	0.811
	SWAP-1	80-100 freq. feat. + headlines	0.805
	Ripper	negative tests	0.798
	Ripper		0.795
	SWAP-1	80-100 freq. feat.	0.789
	SWAP-1	80-100 boolean feat.	0.785
	Experts	3-words	0.759
	Experts	2-words	0.753
	Rocchio		0.748
	Experts	1-word	0.647

Table 7: Break-even summaries for Reuters-22173

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.* [1995], in a study in which neural networks were used on the Reuters-22173 corpus to learn nonlinear classifiers using a variety of representations, noted 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.

There is also some evidence that reducing dimensionality of the examples 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: the learning time for both Ripper and sleeping-experts with four word phrases averages a little over a ninety seconds[14] per category on the larger Lewis dataset.[15]

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 for SWAP-1 was improved 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 0.798 to 0.811.

3.4 The Reuters-21578 collection

We note that Table 7 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 0.748 for the Apte *et al.* datasets, but only 0.635 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.

To address this problem, a second version of the Reuters corpus, the Reuters-21578 corpus, has recently been made available.[16] The main differences are first, that a small number of duplicate documents have been removed, and second, that the corpus has been formatted and documented so as to facilitate systematic and reproducible experimentation. Two different training and testing splits of this corpus are proposed: the *ModApte* split,

[14] These times are on a MIPS Irix 6.3 with 200 MHz R10000 processors.

[15] 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, during classification, we only keep information on experts that correspond to phrases that appear at least twice in the corpus. Using hash tables we can access each expert in constant time using the phrase it represents as a key. Using this scheme the amount of memory used for maintaining a pool grows linearly with the total size of the text (for a fixed n-gram size).

[16] On http://www.research.att.com/~lewis/reuters21578.html.

Dataset	Learner	Options	Micro-Averaged Break-Even
ModLewis dataset	Experts	3-words	0.769
	Experts	2-words	0.753
	RIPPER	(negative tests)	0.696
	RIPPER		0.689
	Experts	1-word	0.677
	Rocchio		0.668
ModApte dataset	Experts	3-words	0.827
	Experts	2-words	0.823
	RIPPER	(negative tests)	0.820
	RIPPER		0.819
	Experts	1-word	0.798
	Rocchio		0.776

Table 8: Break-even summaries for Reuters-21578

which contains 9,603 training examples and 3,299 test examples, and the *ModLewis* split, which contains 13,625 training examples and 6,188 test cases. (The *ModApte* split is perhaps misnamed; it appears to be more similar to the partition used by Weiner *et al* [1995].)

Table 8 presents our results on this variant of the corpora, which also confirm the hypothesis that context is important.

3.5 Further analysis of the experimental results

3.5.1 Effect of class frequency on performance

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 datasets, 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 this section we will analyze the data is somewhat more detail, in order to determine which circumstances are most favorable for which learning algorithms.

In reviewing the experimental results, one property which appears to greatly affect the relative performance of the three learning algorithms considered in this paper is the relative frequency of class to be learned. By *relative frequency* we mean simply the fraction of times that the class appears in the training data. In both the TREC-AP dataset and the Reuters datasets, there is a broad range of class frequencies. In general, Rocchio seems to perform better (relative to the other learning algorithms) on the low-frequency classes—those classes with fewest positive examples in the data.

For instance, in the TREC-AP dataset, there are 20 categories, the least frequent of which occurs only 123 times in the entire corpus, and the most frequent of which occurs

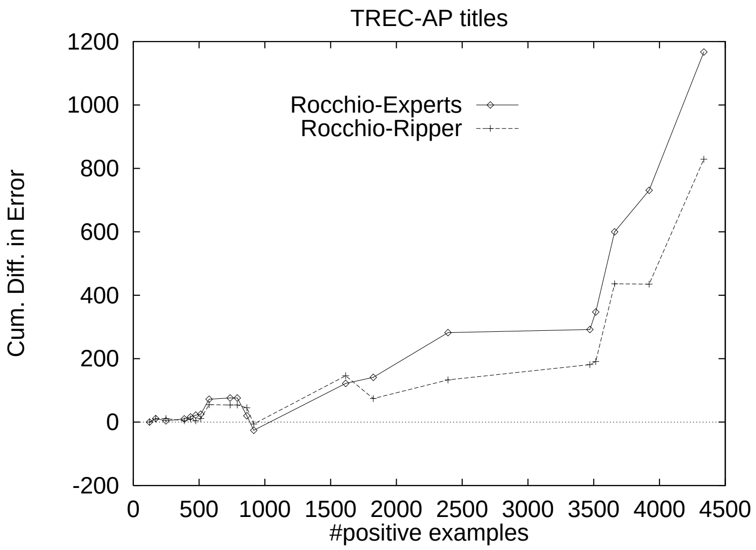

Figure 5: Cumulative difference in errors: TREC-AP

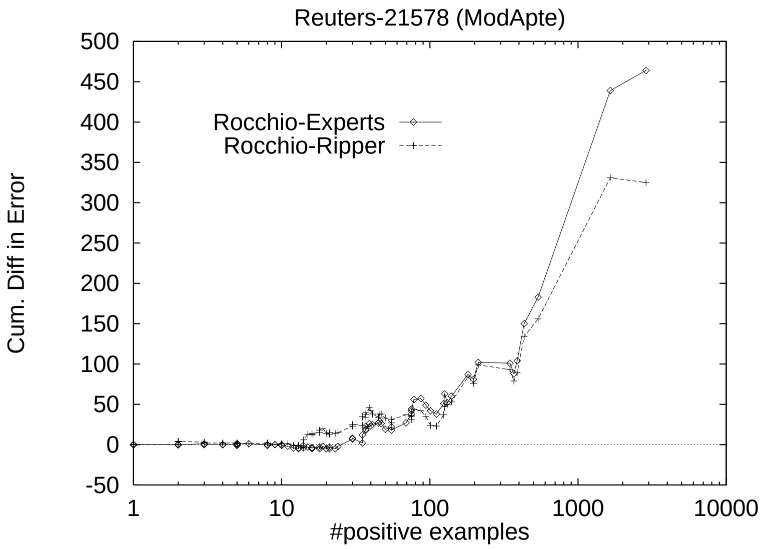

Figure 6: Cumulative difference in errors: ModApte

4337 times. (See Table 5.) Both RIPPER and sleeping-experts (with 4-word phrases) actually have a slightly higher error rate than Rocchio, on average, on the 12 least frequent classes. However, both algorithms dominate Rocchio on the more frequent classes: sleeping-experts achieves a lower error than Rocchio on all of the eight of most frequent classes, and RIPPER outperforms Rocchio on five of the six most frequent classes, and obtains essentially equivalent performance (only one additional error) on the remaining class.

We also analyzed the effect of frequency on the 93 classes on the Reuters-21578 ModApte dataset, and observed a similar effect.[17] Here, there is no apparent systematic difference between RIPPER and Rocchio for categories with less than 15 positive examples in the training set. For more frequent classes, however, RIPPER tends to outperform Rocchio. A similar behavior is observed for the sleeping-experts algorithm; here, Rocchio and sleeping-experts achieved similar error rates unless there are at least 30 positive examples in the training set.

The effect of class frequency on the learners can be easily seen in the graphs of Figures 5 and 6. Given a category c with an associated training set and test set, let us define the *edge* of a learner A relative to a second learner A' on category c to be the number of errors made by A' on the test set minus the number of errors made by A on the test set; in other words, the edge of A relative to A' is simply the number of errors avoided by using A instead of A'. The graphs of Figure 5-6 show, for each frequency n, the *cumulative* edge of RIPPER and sleeping-experts, relative to Rocchio, for all problems of frequency less than or equal to n.[18] Note that when this cumulative edge is close to zero, then performance is comparable to that of Rocchio (for problems of this frequency or less); also when the cumulative edge curve tends upwards (respectively downwards) this indicates that the learning algorithm is generally outperforming Rocchio (respectively being outperformed by Rocchio) with respect to error rate.

The graphs in the Figure illustrate the trends we have discussed above. In both cases, the cumulative edge is small for relatively low-frequency classes, and then begins to drift upwards. The points at which this upward drift begins are different; however in both domains, it is clearly present by the time the relative frequency reaches the range of 1% of the dataset (*i.e.*, 96 examples for Reuters-21578, 2098 examples for TREC-AP.) The graphs also clearly indicate that, although performance is at least competitive throughout the frequency range, most of the cumulative edge of the novel algorithms is due to a relatively small number of relatively frequent classes.

In retrospect, the very competitive performance of Rocchio on rare classes should not be surprising. Because Rocchio uses an inverse document frequency (IDF) weighting scheme, its classifier will weight rare words heavily: this bias is clearly most appropriate in learning a rare class. Also, historically, Rocchio has been used most successfully in improving performance on *ad hoc* queries against large database; here typically only a small fraction of the documents in a database would be relevant. Of course, there are other text categorization domains in which accurate classification of frequent classes is important—one example of such a domain might be categorization of a user's incoming e-mail [Cohen, 1996a].

[17] In this analysis, we measured the frequency of classes using only the training data.

[18] In the graph for the Reuters dataset, we have plotted the log of frequency, as there is a very wide range of frequencies in the data.

In practical settings there may be *a priori* reasons to prefer one learning algorithm to another. For instance, Rocchio has the advantage that its hypotheses can be easily adapted to retrieval in many statistical IR systems; RIPPER has the advantage that its hypotheses can be easily incorporated into a boolean retrieval system; and sleeping-experts has the advantage of being an on-line algorithm with strong performance guarantees. The experiments in this section suggest that an additional reason to prefer Rocchio might be an expectation that most classes are rare; conversely, one might prefer to use RIPPER or sleeping-experts if most classes were frequent.

The experiments of this section also suggest that the average case performance of RIPPER and sleeping-experts might be improved by focusing on their performance on rare classes. One natural extension to the algorithms described in this paper that might improve performance on rare classes would be to augment the features indicating the presence or absence of words with additional information such as inverse document frequency. For instance, in the sleeping-experts framework one could modify the experts so that their predictions are weighted according to the IDF weight of the corresponding phase. Similarly, RIPPER could be extended to prefer tests on words with high IDF weights. We plan to explore these extensions in future research.

3.5.2 Sensitivity of Rocchio to parameter settings

In the experiments reported above, the parameters β and γ for Rocchio were chosen based on experiments performed by different researchers on a different classification task [Ittner *et al.*, 1995]. We also performed some smaller-scale experiments to explore the sensitivity of Rocchio to these parameters, and the degree to which performance might be improved by parameter tuning. We choose eight different categories[19] from the ModApte split of the Reuters-21578 dataset, and ran Rocchio using 31 different combinations[20] of β and γ.

The results indicate that Rocchio is indeed sensitive to β and γ. If one chooses for each category the optimal parameter values for that category (*i.e.*, the parameter settings that do best on the test set) then the average error rate is reduced by more than a factor of two relative to the default settings (from 2.6% to 1.2%). This suggests that the performance of Rocchio might be improved substantially by appropriate parameter tuning.

However, the experiments also show finding these optimal parameter settings is not trivial. If, for instance, one adopts the simple procedure of selecting for each category the parameter settings that give the lowest *training* set error,[21] the improvement in error rate is quite small (from 2.6% to 2.5%); furthermore, the error rate is much higher on the most frequent category, which is weighted heavily in the micro-averaged crossover measure. (The error rate for this category, "acq", is raised from 11.9% to 15.0%.)

It is also clear that different categories can behave quite differently for different variations

[19]The categories were "acq", "corn", "grain", "mony-fx", "trade", "ship", "gnp" and "pet-chem".

[20]We selected β and γ from the set $\{0, 1, 2, 4, 8, 16, 32\}$, and then discarded certain combinations that seemed unreasonable or redundant: combinations where $\beta = \gamma$ and $\beta \neq 1$, and combinations where $\gamma = 0$ and $\beta \neq 1$.

[21]In this case, choosing the parameter setting with lowest training set error is a simple but plausible choice: for linear models, such as that produced by Rocchio, training set error is often closely correlated with test set error.

of Rocchio. We also implemented a variation of Rocchio in which two prototype vectors are built, one for the positive examples, and one for the negative examples, and new instances are assigned to a category depending on the distance to the closest of the two prototypes [Armstrong *et al.*, 1995; Pazzani *et al.*, 1995]. This method was generally not competitive with the more traditional version of Rocchio, obtaining an average error rate of more than 20%; however, it obtained a much lower error rate on the frequent category "acq" (4.2% versus 11.9% for the default version of Rocchio.)

In summary, we suspect that automatic tuning methods might well improve Rocchio's performance; this might perhaps be done by automatic means on a category by category basis, using cross-validation or some other method to estimate test set error for different parameter settings, and perhaps incorporating recent extensions to Rocchio's algorithm such as dynamic feedback optimization [Buckley and Salton, 1995]. Development of such automatic methods is an intriguing direction for future research, and one that we plan to pursue [Schapire *et al.*, 1998].

We will conclude this section with a methodological point. The sensitivity of Rocchio to its parameter settings represents not only an opportunity, but also a danger. In the absence of a clearly defined (and preferably automatic) parameter tuning mechanism, it is quite possible for an experimenter to unwittingly "overfit" a benchmark problem by finding parameter settings that perform well on the test set for that problem, but do not perform well in general, and which moreover could not have been discovered without using the test set itself for reference. This phenomenon is also possible with a small set of benchmark problems, such as those associated with the AP headline corpus or the Reuters corpora. (For Reuters, while there are many categories, performance is dominated by a few frequent ones.)

In evaluating a learning system, it is thus methodologically safer to use parameter settings (or tuning mechanisms) that have been developed on problems different from the ones being used as benchmarks. In this paper, we have followed this procedure as much as possible; the parameters for Rocchio were set based on previous work in text categorization and retrieval [Buckley *et al.*, 1994; Ittner *et al.*, 1995], the Ripper algorithm was developed and tuned using a set of non-textual classification problems taken from the UC/Irvine Repository [Cohen, 1995a], and the design of sleeping-experts was driven almost entirely by theoretical results.

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 representations for a classifier, different search methods to find a classifier, and different notions of context.

For Ripper, predictions are based on rules which test for the simultaneous presence or absence of several words; for sleeping-experts, predictions are based on sparse phrases. Thus for both learners, a word's effect on prediction is "context sensitive" in the sense that this effect depends on remainder of the document containing that word. For Ripper, for instance, a word w_1 that appears in a conjunction "$w_1 \in D$ and $w_2 \in D$ ... and $w_k \in D$" has no effect on predicting the class of the document D unless the words $w_2, \ldots, w_k$ also appear somewhere in D; note however that $w_2, \ldots, w_k$ may occur anywhere in D, in any relative order. For sleeping-experts, the effect of a word w in predicting a class for a document D depends on the sparse phrases in which w appears—in other words, the context of w depends on the words that appear close to w in D, and the relative order of these nearby words.

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 than algorithms which learn context insensitive 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 these results as a confirmation of the usefulness and practicality of learning classifiers that represent contextual information.

When possible, we have compared our results to previous results on the same tasks. There is a large number of relevant studies (see for instance [Lewis, 1992b; Lewis, 1992a; Yang, 1994; Yang and Chute, 1994; Apte *et al.*, 1994a; Hull *et al.*, 1995] [Wiener *et al.*, 1995; Cohen, 1995b; Schutze *et al.*, 1996; Ng *et al.*, 1997] and the references therein) for which a direct comparison is impossible due to the diversity of the different datasets used in the experiments, the different methods used to pre-process and partition the data, and the different measures used to evaluate performance. It is difficult to predict how RIPPER and sleeping-experts will perform relative to these previously published algorithms. However, we would like to note that both RIPPER and sleeping-experts have certain properties that no previously published text categorization algorithms have; RIPPER constructs a boolean classifier given a direct representation of the training corpus, and sleeping-experts has strong formal guarantees on performance [Freund *et al.*, 1997] in a strict on-line model of learning. Furthermore, RIPPER and sleeping-experts make only a few statistical assumptions about the data (in contrast to other approaches such as [Schutze *et al.*, 1996]), and both RIPPER and sleeping-experts attempt to minimize error directly (as opposed to some related measure, such as squared loss used in [Yang and Chute, 1994; Lewis *et al.*, 1996]).

While this paper has compared and evaluated two context-sensitive learning methods, we note in closing that a potentially valuable research topic is to combine these two different learning methods, so as to exploit both of their stronger features. For instance, the output of RIPPER could be used to derive an additional set of experts, corresponding to the rules that RIPPER builds. This set could then be added to the existing set of sparse phrases that is used by sleeping-experts, allowing the learning system to model both short range contextual information (using the sparse n-grams) and long range correlations (using the conjunctions in the rule set built by RIPPER).

Acknowledgments

The authors would like to thank David Lewis for comments on a draft of the paper and for help in preparing data for the experiments, and the reviewers for SIGIR-96 for a number of useful suggestions. We also would like to thank Marti Hearst, Mehran Sahami, and the anonymous reviewers of TOIS for many constructive comments and suggestions.

References

(Almuallim and Dietterich, 1991) H. Almuallim and T. Dietterich. Learning with many irrelevant features. In *Proceedings of the Tenth National Conference on Artificial Intelligence*. AAAI Press, 1991.

(Apte *et al.*, 1994a) Chidanand Apte, Fred Damerau, and Sholom Weiss. Towards language independent automated learning of text categorisation methods. In *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 23–30, 1994.

(Apté *et al.*, 1994b) 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.

(Armstrong *et al.*, 1995) R. Armstrong, D. Frietag, T. Joachims, and T. M. Mitchell. WebWatcher: a learning apprentice for the world wide web. In *Proceedings of the 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, Stanford, CA, 1995. AAAI Press.

(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.

(Brunk and Pazzani, 1991) Clifford Brunk and Michael Pazzani. Noise-tolerant relational concept learning algorithms. In *Proceedings of the Eighth International Workshop on Machine Learning*, Ithaca, New York, 1991. Morgan Kaufmann.

(Buckley and Salton, 1995) Chris Buckley and Gerard Salton. Optimization of relevance feedback weights. In *Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 351–357, New York, 1995.

(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, 1996a) William W. Cohen and Yoram Singer. Context-sensitive learning methods for text categorization. In *Proceedings of the 19th Annual International ACM Conference on Research and Development in Information Retrieval*, pages 307–315, Zurich, Switzerland, 1996. ACM Press.

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

(Cohen, 1993) William W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993.

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

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

(Cohen, 1996a) William W. Cohen. Learning rules that classify e-mail. In *Proceedings of the 1996 AAAI Spring Symposium on Machine Learning and Information Access*, Palo Alto, California, 1996.

(Cohen, 1996b) 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.

(Freund *et al.*, 1997) Y. Freund, R.E. Schapire, Y. Singer, and M.K. Warmuth. Using and combining predictors that specialize. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, pages 334–343, 1997.

(Fürnkranz and Widmer, 1994) Johannes Fürnkranz and Gerhard Widmer. Incremental reduced error pruning. In *Machine Learning: Proceedings of the Eleventh Annual Conference*, New Brunswick, New Jersey, 1994. Morgan Kaufmann.

(Hull and Grefenstette, 1996) David Hull and Grefenstette. Stemming algorithms: a case study for detailed evaluation. *Journal of the American Society for Information Science*, 47(1):70–84, 1996.

(Hull *et al.*, 1995) S.A. Hull, J.O. Pedersen, and H. Schutze. Method combination for document filtering. In *Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, 1995.

(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.

(John *et al.*, 1994) George John, Ron Kohavi, , and Karl Pfeger. Irrelevant features and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh Annual Conference*, New Brunswick, New Jersey, 1994. Morgan Kaufmann.

(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, 1992a) David Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 37–50, 1992.

(Lewis, 1992b) 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.

(Littlestone, 1988) Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 1988.

(Ng *et al.*, 1997) H.T. Ng, W.B. Gog, and K.L. Low. Feature selection, perceptron learning, and a usability case study for text categorization. In *Proceedings of the Twentieth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 67–73, 1997.

(Pagallo and Haussler, 1990) Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5(1), 1990.

(Pazzani *et al.*, 1995) M. Pazzani, L. Nguyen, and S. Mantik. Learning from hotlists and coldlists: towards a WWW information filtering and seeking agent. In *Proceedings of AI Tools Conference*, Washington, DC, 1995.

(Quinlan, 1990) J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3), 1990.

(Quinlan, 1995) J. Ross Quinlan. MDL and categorical theories (continued). In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

(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.

(Schapire *et al.*, 1998) R.E. Schapire, Y. Singer, and A. Singhal. Bosoting and Rocchio applied to text filtering. in preparation, 1998.

(Schutze *et al.*, 1996) H. Schutze, D.A. Hull, and J.O. Pedersen. A comparison of classifiers and document representations for teh routing problem. In *Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Zurich, Switzerland, 1996.

(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.

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

(Yang, 1994) Yiming Yang. Expert network: Effective and efficient learning from human decisions in text categorization and retrieval. In *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 13–22, 1994.