Machine Learning 34, 107–130 (1999)

A Winnow-Based Approach to Context-Sensitive Spelling Correction*

ANDREW R. GOLDING golding@merl.com
MERL—A Mitsubishi Electric Research Laboratory, 201 Broadway, Cambridge, MA 02139

DAN ROTH danr@cs.uiuc.edu
Department of Computer Science, University of Illinois—Urbana/Champaign, 1304 W. Springfield Avenue, Urbana, IL 61801

Editors: Claire Cardie and Raymond Mooney

Abstract. A large class of machine-learning problems in natural language require the characterization of linguistic context. Two characteristic properties of such problems are that their feature space is of very high dimensionality, and their target concepts depend on only a small subset of the features in the space. Under such conditions, multiplicative weight-update algorithms such as Winnow have been shown to have exceptionally good theoretical properties. In the work reported here, we present an algorithm combining variants of Winnow and weighted-majority voting, and apply it to a problem in the aforementioned class: context-sensitive spelling correction. This is the task of fixing spelling errors that happen to result in valid words, such as substituting *to* for *too*, *casual* for *causal*, and so on. We evaluate our algorithm, WinSpell, by comparing it against BaySpell, a statistics-based method representing the state of the art for this task. We find: (1) When run with a full (unpruned) set of features, WinSpell achieves accuracies significantly higher than BaySpell was able to achieve in either the pruned or unpruned condition; (2) When compared with other systems in the literature, WinSpell exhibits the highest performance; (3) While several aspects of WinSpell's architecture contribute to its superiority over BaySpell, the primary factor is that it is able to learn a better linear separator than BaySpell learns; (4) When run on a test set drawn from a different corpus than the training set was drawn from, WinSpell is better able than BaySpell to adapt, using a strategy we will present that combines supervised learning on the training set with unsupervised learning on the (noisy) test set.

Keywords: Winnow, multiplicative weight-update algorithms, context-sensitive spelling correction, Bayesian classifiers

1. Introduction

A large class of machine-learning problems in natural language require the characterization of linguistic context. Such problems include lexical disambiguation tasks such as part-of-speech tagging and word-sense disambiguation; grammatical disambiguation tasks such as prepositional-phrase attachment; and document-processing tasks such as text classification (where the context is usually the whole document). Such problems have two distinctive properties. First, the richness of the linguistic structures that must be represented results in extremely high-dimensional feature spaces for the problems. Second, any given target

*An earlier version of this work appeared in ICML'96.

concept depends on only a small subset of the features, leaving a huge balance of features that are irrelevant to that particular concept. In this paper, we present a learning algorithm and an architecture with properties suitable for this class of problems.

The algorithm builds on recently introduced theories of multiplicative weight-update algorithms. It combines variants of Winnow (Littlestone, 1988) and Weighted Majority (Littlestone & Warmuth, 1994). Extensive analysis of these algorithms in the COLT literature has shown them to have exceptionally good behavior in the presence of irrelevant attributes, noise, and even a target function changing in time (Littlestone, 1988; Littlestone & Warmuth, 1994; Herbster & Warmuth, 1995). These properties make them particularly well-suited to the class of problems studied here.

While the theoretical properties of the Winnow family of algorithms are well known, it is only recently that people have started to test the claimed abilities of the algorithms in applications. We address the claims empirically by applying our Winnow-based algorithm to a large-scale real-world task in the aforementioned class of problems: context-sensitive spelling correction.

Context-sensitive spelling correction is the task of fixing spelling errors that result in valid words, such as *I'd like a peace of cake*, where *peace* was typed when *piece* was intended. These errors account for anywhere from 25 to over 50% of observed spelling errors (Kukich, 1992); yet they go undetected by conventional spell checkers, such as Unix *spell*, which only flag words that are not found in a word list. Context-sensitive spelling correction involves learning to characterize the linguistic contexts in which different words, such as *piece* and *peace*, tend to occur. The problem is that there is a multitude of features one might use to characterize these contexts: features that test for the presence of a particular word nearby the target word; features that test the pattern of parts of speech around the target word; and so on. For the tasks we will consider, the number of features ranges from a few hundred to over 10,000.[1] While the feature space is large, however, target concepts, such as "a context in which *piece* can occur", depend on only a small subset of the features, the vast majority being irrelevant to the concept at hand. Context-sensitive spelling correction therefore fits the characterization presented above, and provides an excellent testbed for studying the performance of multiplicative weight-update algorithms on a real-world task.

To evaluate the proposed Winnow-based algorithm, WinSpell, we compare it against BaySpell (Golding, 1995), a statistics-based method that is among the most successful tried for the problem. We first compare WinSpell and BaySpell using the heavily-pruned feature set that BaySpell normally uses (typically 10–1000 features). WinSpell is found to perform comparably to BaySpell under this condition. When the full, unpruned feature set is used, however, WinSpell comes into its own, achieving substantially higher accuracy than it achieved in the pruned condition, and better accuracy than BaySpell achieved in either condition.

To calibrate the observed performance of BaySpell and WinSpell, we compare them to other methods reported in the literature. WinSpell is found to significantly outperform all the other methods tried when using a comparable feature set.

At their core, WinSpell and BaySpell are both linear separators. Given this fundamental similarity between the algorithms, we ran a series of experiments to understand why WinSpell was nonetheless able to outperform BaySpell. While several aspects of the

WinSpell architecture were found to contribute to its superiority, the principal factor was that WinSpell simply learned a better linear separator than BaySpell did. We attribute this to the fact that the Bayesian linear separator was based on idealized assumptions about the domain, while Winnow was able to adapt, via its mistake-driven update rule, to whatever conditions held in practice.

We then address the issue of dealing with a test set that is dissimilar to the training set. This arises in context-sensitive spelling correction, as well as related disambiguation tasks, because patterns of word usage can vary widely across documents; thus the training and test documents may be quite different. After first confirming experimentally that performance does indeed degrade for unfamiliar test sets, we present a strategy for dealing with this situation. The strategy, called *sup/unsup*, combines supervised learning on the training set with unsupervised learning on the (noisy) test set. We find that, using this strategy, both BaySpell and WinSpell are able to improve their performance on an unfamiliar test set. WinSpell, however, is found to do particularly well, achieving comparable performance when using the strategy on an unfamiliar test set as it had achieved on a familiar test set.

The rest of the paper is organized as follows: the next section describes the task of context-sensitive spelling correction. We then present the Bayesian method that has been used for it. The Winnow-based approach to the problem is introduced. The experiments on WinSpell and BaySpell are presented. The final section concludes.

2. Context-sensitive spelling correction

With the widespread availability of spell checkers to fix errors that result in non-words, such as *teh*, the predominant type of spelling error has become the kind that results in a real, but unintended word; for example, typing *there* when *their* was intended. Fixing this kind of error requires a completely different technology from that used in conventional spell checkers: it requires analyzing the context to infer when some other word was more likely to have been intended. We call this the task of *context-sensitive spelling correction*. The task includes fixing not only “classic” types of spelling mistakes, such as homophone errors (e.g., *peace* and *piece*) and typographic errors (e.g., *form* and *from*), but also mistakes that are more commonly regarded as grammatical errors (e.g., *among* and *between*), and errors that cross word boundaries (e.g., *maybe* and *may be*).

The problem has started receiving attention in the literature only within about the last half-dozen years. A number of methods have been proposed, either for context-sensitive spelling correction directly, or for related lexical disambiguation tasks. The methods include word trigrams (Mays, Damerau, & Mercer, 1991), Bayesian classifiers (Gale, Church, & Yarowsky, 1993), decision lists (Yarowsky, 1994), Bayesian hybrids (Golding, 1995), a combination of part-of-speech trigrams and Bayesian hybrids (Golding & Schabes, 1996), and, more recently, transformation-based learning (Mangu & Brill, 1997), latent semantic analysis (Jones & Martin, 1997), and differential grammars (Powers, 1997). While these research systems have gradually been achieving higher levels of accuracy, we believe that a Winnow-based approach is particularly promising for this problem, due to the problem’s need for a very large number of features to characterize the context in which a word occurs, and Winnow’s theoretically-demonstrated ability to handle such large numbers of features.

2.1. Problem formulation

We will cast context-sensitive spelling correction as a word disambiguation task. The ambiguity among words is modelled by *confusion sets*. A confusion set $C = \{W_1, \ldots, W_n\}$ means that each word W_i in the set is ambiguous with each other word. Thus if $C = \{hear, here\}$, then when we see an occurrence of either *hear* or *here* in the target document, we take it to be ambiguous between *hear* and *here*; the task is to decide from the context which one was actually intended. Acquiring confusion sets is an interesting problem in its own right; in the work reported here, however, we take our confusion sets largely from the list of "Words Commonly Confused" in the back of the Random House dictionary (Flexner, 1983), which includes mainly homophone errors. A few confusion sets not in Random House were added, representing grammatical and typographic errors.

The Bayesian and Winnow-based methods for context-sensitive spelling correction will be described below in terms of their operation on a single confusion set; that is, we will say how they disambiguate occurrences of words W_1 through W_n. The methods handle multiple confusion sets by applying the same technique to each confusion set independently.

2.2. Representation

A target problem in context-sensitive spelling correction consists of (i) a sentence, and (ii) a target word in that sentence to correct. Both the Bayesian and Winnow-based algorithms studied here represent the problem as a list of active features; each active feature indicates the presence of a particular linguistic pattern in the context of the target word. We use two types of features: *context words* and *collocations*. Context-word features test for the presence of a particular word within $\pm k$ words of the target word; collocations test for a pattern of up to ℓ contiguous words and/or part-of-speech tags[2] around the target word. In the experiments reported here, k was set to 10 and ℓ to 2. Examples of useful features for the confusion set $\{weather, whether\}$ include:

(1) *cloudy* within ± 10 words
(2) __ *to* VERB

Feature (1) is a context-word feature that tends to imply *weather*. Feature 2 is a collocation that checks for the pattern "*to* VERB" immediately after the target word, and tends to imply *whether* (as in *I don't know whether to laugh or cry*).

The intuition for using these two types of features is that they capture two important, but complementary aspects of context. Context words tell us what kind of words tend to appear in the vicinity of the target word—the "lexical atmosphere". They therefore capture aspects of the context with a wide-scope, semantic flavor, such as discourse topic and tense. Collocations, in contrast, capture the local syntax around the target word. Similar combinations of features have been used in related tasks, such as accent restoration (Yarowsky, 1994) and word sense disambiguation (Ng & Lee, 1996).

We use a *feature extractor* to convert from the initial text representation of a sentence to a list of active features. The feature extractor has a preprocessing phase in which it learns a set of features for the task. Thereafter, it can convert a sentence into a list of active features simply by matching its set of learned features against the sentence.

In the preprocessing phase, the feature extractor learns a set of features that characterize the contexts in which each word W_i in the confusion set tends to occur. This involves going through the training corpus, and, each time a word in the confusion set occurs, generating all possible features for the context—namely, one context-word feature for every distinct word within $\pm k$ words, and one collocation for every way of expressing a pattern of up to ℓ contiguous elements. This gives an exhaustive list of all features found in the training set. Statistics of occurrence of the features are collected in the process as well.

At this point, pruning criteria may be applied to eliminate unreliable or uninformative features. We use two criteria (which make use of the aforementioned statistics of occurrence): (1) the feature occurred in practically none or all of the training instances (specifically, it had fewer than 10 occurrences or fewer than 10 non-occurrences); or (2) the presence of the feature is not significantly correlated with the identity of the target word (determined by a chi-square test at the 0.05 significance level).

3. Bayesian approach

Of the various approaches that have been tried for context-sensitive spelling correction, the Bayesian hybrid method, which we call BaySpell, has been among the most successful, and is thus the method we adopt here as the benchmark for comparison with WinSpell. BaySpell has been described elsewhere (Golding, 1995), and so will only be briefly reviewed here; however, the version here uses an improved smoothing technique, which is described below.

Given a sentence with a target word to correct, BaySpell starts by invoking the feature extractor (Section 2.2) to convert the sentence into a set $\mathcal{F}$ of active features. BaySpell normally runs the feature extractor with pruning enabled. To a first approximation, BaySpell then acts as a naive Bayesian classifier. Suppose for a moment that we really were applying Naive Bayes. We would then calculate the probability that each word W_i in the confusion set is the correct identity of the target word, given that features $\mathcal{F}$ have been observed, by using Bayes' rule with the conditional independence assumption:

$$P(W_i \mid \mathcal{F}) = \left(\prod_{f \in \mathcal{F}} P(f \mid W_i) \right) \frac{P(W_i)}{P(\mathcal{F})}$$

where each probability on the right-hand side is calculated by a maximum-likelihood estimate[3] (MLE) over the training set. We would then pick as our answer the W_i with the highest $P(W_i \mid \mathcal{F})$.

BaySpell differs from the naive approach in two respects: first, it does not assume conditional independence among features, but rather has heuristics for detecting strong dependencies, and resolving them by deleting features until it is left with a reduced set $\mathcal{F}'$ of (relatively) independent features, which are then used in place of $\mathcal{F}$ in the equation above. This procedure is called *dependency resolution*.

Second, to estimate the $P(f \mid W_i)$ terms, BaySpell does not use the simple MLE, as this tends to give likelihoods of 0.0 for rare features (which are abundant in the task at hand), thus yielding a useless answer of 0.0 for the posterior probability. Instead, BaySpell performs smoothing by interpolating between the MLE of $P(f \mid W_i)$ and the MLE of the unigram probability, $P(f)$. Some means of incorporating a lower-order model in this way is generally regarded as essential for good performance (Chen & Goodman, 1996). We use:

$$P_{\text{interp}}(f \mid W_i) = (1 - \lambda)P_{\text{ML}}(f \mid W_i) + \lambda P_{\text{ML}}(f)$$

We set λ to the probability that the presence of feature f is independent of the presence of word W_i; to the extent that this independence holds, $P(f)$ is an accurate (but more robust) estimate of $P(f \mid W_i)$. We calculate λ as the chi-square probability that the observed association between f and W_i is due to chance.

The enhancement of smoothing, and to a minor extent, dependency resolution, greatly improve the performance of BaySpell over the naive Bayesian approach. (The effect of these enhancements can be seen empirically in Section 5.4.)

4. Winnow-based approach

There are various ways to use a learning algorithm, such as Winnow (Littlestone, 1988), to do the task of context-sensitive spelling correction. A straightforward approach would be to learn, for each confusion set, a discriminator that distinguishes specifically among the words in that set. The drawback of this approach, however, is that the learning is then applicable only to one particular discrimination task. We pursue an alternative approach: that of learning the contextual characteristics of each word W_i individually. This learning can then be used to distinguish word W_i from any other word, as well as to perform a broad spectrum of other natural language tasks (Roth, 1998). In the following, we briefly present the general approach, and then concentrate on the task at hand, context-sensitive spelling correction.

The approach developed is influenced by the Neuroidal system suggested by Valiant (1994). The system consists of a very large number of items, in the range of 10^5. These correspond to high-level concepts, for which humans have words, as well as lower-level predicates from which the high-level ones are composed. The lower-level predicates encode aspects of the current state of the world, and are input to the architecture from the outside. The high-level concepts are learned as functions of the lower-level predicates; in particular, each high-level concept is learned by a *cloud* or ensemble of classifiers. All classifiers within the cloud learn the cloud's high-level concept autonomously, as a function of the same lower-level predicates, but with different values of the learning parameters. The outputs of the classifiers are combined into an output for the cloud using a variant of the Weighted Majority algorithm (Littlestone & Warmuth, 1994). Within each classifier, a variant of the Winnow algorithm (Littlestone, 1988) is used. Training occurs whenever the architecture interacts with the world, for example, by reading a sentence of text; the architecture thereby receives new values for its lower-level predicates, which in turn serve as an example for training the high-level ensembles of classifiers. Learning is thus an on-line process that is done on a continuous basis[4] (Valiant, 1995).

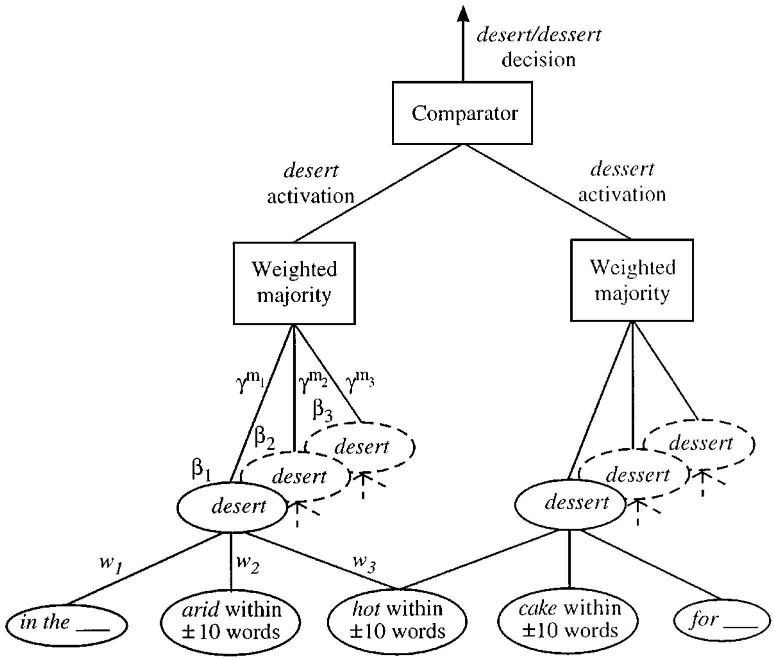

Figure 1. Example of WinSpell network for {*desert*, *dessert*}. The five nodes in the bottom tier of the network correspond to features. The two higher-level *clouds* of nodes (each shown as overlapping bubbles suspended from a box) correspond to the words in the confusion set. The nodes within a cloud each run the Winnow algorithm in parallel with a different setting of the demotion parameter, β, and with their own copy of the input arcs and the weights on those arcs. The overall activation level for each word in the confusion set is obtained by applying a weighted majority algorithm to the nodes in the word's cloud. The word with the highest activation level is selected.

Figure 1 shows the instantiation of the architecture for context-sensitive spelling correction, and in particular for correcting the words {*desert*, *dessert*}. The bottom tier of the network consists of nodes for lower-level predicates, which in this application correspond to features of the domain. For clarity, only five nodes are shown; thousands typically occur in practice. High-level concepts in this application correspond to words in the confusion set, here *desert* and *dessert*. Each high-level concept appears as a *cloud* of nodes, shown as a set of overlapping bubbles suspended from a box. The output of the clouds is an activation level for each word in the confusion set; a *comparator* selects the word with the highest activation as the final result for context-sensitive spelling correction.

The sections below elaborate on the use of Winnow and Weighted Majority in WinSpell, followed by a discussion of the properties of the architecture.

4.1. Winnow

The job of each classifier within a cloud of WinSpell is to decide whether a particular word W_i in the confusion set belongs in the target sentence. Each classifier runs the Winnow algorithm. It takes as input a representation of the target sentence as a set of active features, and returns a binary decision as to whether its word W_i belongs in the sentence. Let $\mathcal{F}$ be the set of active features; and for each feature $f \in \mathcal{F}$, let w_f be the weight on the arc connecting f to the classifier at hand. The Winnow algorithm then returns a classification of 1 (positive) iff:

$$\sum_{f \in \mathcal{F}} w_f > \theta,$$

where θ is a threshold parameter. In the experiments reported here, θ was set to 1.

Initially, the classifier has no connection to any feature in the network. Through training, however, it establishes appropriate connections, and learns weights for these connections. A training example consists of a sentence, represented as a set of active features, together with the word W_c in the confusion set that is correct for that sentence. The example is treated as a positive example for the classifiers for W_c, and as a negative example for the classifiers for the other words in the confusion set.

Training proceeds in an on-line fashion: an example is presented to the system, the representation of the classifiers is updated, and the example is then discarded. The first step of training a classifier on an example is to establish appropriate connections between the classifier and the active features $\mathcal{F}$ of the example. If an active feature $f \in \mathcal{F}$ is not already connected to the classifier, and the sentence is a *positive* example for the classifier (that is, the classifier corresponds to the target word W_c that occurs in the sentence), then we add a connection between the feature and the classifier, with a default weight of 0.1. This policy of building connections on an as-needed basis results in a *sparse* network with only those connections that have been demonstrated to occur in real examples. Note that we do not build any new connections if the sentence is a *negative* example for the classifier; [5] one consequence is that different words in a confusion set may have links to different subsets of the possible features, as seen in Figure 1.

The second step of training is to update the weights on the connections. This is done using the Winnow update rule, which updates the weights only when a mistake is made. If the classifier predicts 0 for a positive example (i.e., where 1 is the correct classification), then the weights are promoted:

$$\forall f \in \mathcal{F},\ w_f \leftarrow \alpha \cdot w_f,$$

where $\alpha > 1$ is a promotion parameter. If the classifier predicts 1 for a negative example (i.e., where 0 is the correct classification), then the weights are demoted:

$$\forall f \in \mathcal{F},\ w_f \leftarrow \beta \cdot w_f,$$

where $0 < \beta < 1$ is a demotion parameter. In the experiments reported here, α was set to 1.5, and β was varied from 0.5 to 0.9 (see also Section 4.2.). In this way, weights on non-active features remain unchanged, and the update time of the algorithm depends on the number of *active* features in the current example, and not the total number of features in the domain. The use of a sparse architecture, as described above, coupled with the representation of each example as a list of *active* features is reminiscent of the infinite attribute models of Winnow (Blum, 1992).

4.2. Weighted Majority

Rather than evaluating the evidence for a given word W_i using a single classifier, WinSpell combines evidence from multiple classifiers; the motivation for doing so is discussed below. Weighted Majority (Littlestone & Warmuth, 1994) is used to do the combination. The basic approach is to run several classifiers in parallel within each cloud to try to predict whether W_i belongs in the sentence. Each classifier uses different values of the learning parameters, and therefore makes slightly different predictions. The performance of each classifier is monitored, and a weight is derived reflecting its observed prediction accuracy. The final activation level output by the cloud is a sum of the predictions of its member classifiers, weighted by the abovementioned weights.

More specifically, we used clouds composed of five classifiers, differing only in their values for the Winnow demotion parameter β; values of 0.5, 0.6, 0.7, 0.8, and 0.9 were used. The weighting scheme assigns to the jth classifier a weight γ^{m_j}, where $0 < \gamma < 1$ is a constant, and m_j is the total number of mistakes made by the classifier so far. The essential property is that the weight of a classifier that makes many mistakes rapidly disappears. We start with $\gamma = 1.0$ and decrease its value with the number of examples seen, to avoid weighing mistakes of the initial hypotheses too heavily.[6] The total activation returned by the cloud is then:

$$\frac{\sum_j \gamma^{m_j} C_j}{\sum_j \gamma^{m_j}},$$

where C_j is the classification, either 1 or 0, returned by the jth classifier in the cloud, and the denominator is a normalization term.

The rationale for combining evidence from multiple classifiers is twofold. First, when running a mistake-driven algorithm, even when it is known to have good behavior asymptotically, there is no guarantee that the current hypothesis, at any point in time, is any better than the previous one. It is common practice, therefore, to predict using an average of the last several hypotheses, weighting each hypothesis by, for example, the length of its mistake-free stretch (Littlestone, 1995; Cesa-Bianchi et al., 1994). The second layer of WinSpell, i.e., the weighted-majority layer, partly serves this function, though it does so in an on-line fashion.

A second motivation for the weighted-majority layer comes from the desire to have an algorithm that tunes its own parameters. For the task of context-sensitive spelling correction, self-tuning is used to automatically accommodate differences among confusion sets—in particular, differences in the degree to which the words in the confusion set have overlapping usages. For {*weather*, *whether*}, for example, the words occur in essentially disjoint contexts; thus, if a training example gives one word, but the classifier predicts the other, it is almost surely wrong. On the other hand, for {*among*, *between*}, there are numerous contexts in which both words are acceptable; thus disagreement with the training example does not necessarily mean the classifier is wrong. Following a mistake, therefore, we want to demote the weights by more in the former case than in the latter. Updating weights with various demotion parameters in parallel allows the algorithm to select by itself the best setting of parameters for each confusion set. In addition, using a weighted-majority layer strictly increases the expressivity of the architecture. It is plausible that in some cases, a linear separator would be unable to achieve good prediction, while the two-layer architecture would be able to do so.

4.3. Discussion

Multiplicative learning algorithms have been studied extensively in the learning theory community in recent years (Littlestone, 1988; Kivinen & Warmuth, 1995). Winnow has been shown to learn efficiently any linear threshold function (Littlestone, 1988), with a mistake bound that depends on the margin between positive and negative examples. These are functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$ for which there exist real weights $w_1, \ldots, w_n$ and a real threshold θ such that $f(x_1, \ldots, x_n) = 1$ iff $\sum_{i=1}^{n} w_i x_i \geq \theta$. In particular, these functions include Boolean disjunctions and conjunctions on $k \leq n$ variables and r-of-k threshold functions ($1 \leq r \leq k \leq n$).

The key feature of Winnow is that its mistake bound grows linearly with the number of *relevant* attributes and only logarithmically with the total number of attributes n. Using the sparse architecture, in which we do not keep all the variables from the beginning, but rather add variables as necessary, the number of mistakes made on disjunctions and conjunctions is logarithmic in the size of the largest example seen and linear in the number of relevant attributes; it is independent of the total number of attributes in the domain (Blum, 1992).

Winnow was analyzed in the presence of various kinds of noise, and in cases where no linear threshold function can make perfect classifications (Littlestone, 1991). It was proved, under some assumptions on the type of noise, that Winnow still learns correctly, while retaining its abovementioned dependence on the number of total and relevant attributes. (See Kivinen and Warmuth (1995) for a thorough analysis of multiplicative update algorithms versus additive update algorithms, and exact bounds that depend on the sparsity of the target function and the number of active features in the examples.)

The algorithm makes no independence or other assumptions about the attributes, in contrast to Bayesian predictors which are commonly used in statistical NLP. This condition was recently investigated experimentally (on simulated data) (Littlestone, 1995). It was shown that redundant attributes dramatically affect a Bayesian predictor, while superfluous independent attributes have a less dramatic effect, and only when the number of attributes is very large (on the order of 10,000). Winnow is a mistake-driven algorithm; that is, it updates its hypothesis only when a mistake is made. Intuitively, this makes the algorithm more sensitive to the relationships among attributes—relationships that may go unnoticed by an algorithm that is based on counts accumulated separately for each attribute. This is crucial in the analysis of the algorithm and has been shown to be crucial empirically as well (Littlestone, 1995).

One of the advantages of the multiplicative update algorithms is their logarithmic dependence on the number of domain features. This property allows one to learn higher-than-linear discrimination functions by increasing the dimensionality of the feature space. Instead of learning a discriminator in the original feature space, one can generate new features, as conjunctions of original features, and learn a linear separator in the new space, where it is more likely to exist. Given the modest dependency of Winnow on the dimensionality, it can be worthwhile to increase the dimensionality so as to simplify the functional form of the resulting discriminator. The work reported here can be regarded as following this path, in that we define collocations as *patterns* of words and part-of-speech tags, rather than restricting them to tests of singleton elements. This increases the dimensionality and adds redundancy among features, but at the same time simplifies the functional form of the discriminator, to the point that the classes are almost linearly separable in the new space. A similar philosophy, albeit very different technically, is followed by the work on Support Vector Machines (Cortes & Vapnik, 1995).

Theoretical analysis has shown Winnow to be able to adapt quickly to a changing target concept (Herbster & Warmuth, 1995). We investigate this issue experimentally in Section 5.5. A further feature of WinSpell is that it can prune poorly-performing attributes, whose weight falls too low relative to the highest weight of an attribute used by the classifier. By pruning in this way, we can greatly reduce the number of features that need to be retained in the representation. It is important to observe, though, that there is a tension between compacting the representation by aggressively discarding features, and maintaining the ability to adapt to a new test environment. In this paper we focus on adaptation, and do not study discarding techniques. This tradeoff is currently under investigation.

5. Experimental results

To understand the performance of WinSpell on the task of context-sensitive spelling correction, we start by comparing it with BaySpell using the pruned set of features from the feature extractor, which is what BaySpell normally uses. This evaluates WinSpell purely as a method of combining evidence from multiple features. An important claimed strength of the Winnow-based approach, however, is the ability to handle large numbers of features. We tested this by (essentially) disabling pruning, resulting in tasks with over 10,000 features, and seeing how WinSpell and BaySpell scale up.

The first experiment showed how WinSpell and BaySpell perform relative to each other, but not to an outside reference. To calibrate their performance, we compared the two algorithms with other methods reported in the literature, as well as a baseline method.

The success of WinSpell in the previous experiments brought up the question of *why* it was able to outperform BaySpell and the other methods. We investigated this in an ablation study, in which we stripped WinSpell down to a simple, non-learning algorithm, and gave it an initial set of weights that allowed it to emulate BaySpell's behavior exactly. From there, we restored the missing aspects of WinSpell one at a time, observing how much each contributed to improving its performance above the Bayesian level.

The preceding experiments drew the training and test sets from the same population, following the traditional PAC-learning assumption. This assumption may be unrealistic for the task at hand, however, where a system may encounter a target document quite unlike those seen during training. To check whether this was in fact a problem, we tested the across-corpus performance of the methods. We found it was indeed significantly worse than within-corpus performance. To address this problem, we tried a strategy of combining learning on the training set with unsupervised learning on the (noisy) test set. We tested how well WinSpell and BaySpell were able to perform on an unfamiliar test set using this strategy.

The sections below describe the experimental methodology, and present the experiments, interleaved with discussion.

5.1. *Methodology*

In the experiments that follow, the training and test sets were drawn from two corpora: the one-million-word Brown corpus (Kučera & Francis, 1967) and a 3/4-million-word corpus of articles from The Wall Street Journal (WSJ) (Marcus, Santorini, & Marcinkiewicz, 1993). Note that no particular annotations are needed on these corpora for the task of context-sensitive spelling correction; we simply assume that the texts contain no context-sensitive spelling errors, and thus the observed spellings may be taken as a gold standard.

The algorithms were run on 21 confusion sets, which were taken largely from the list of "Words Commonly Confused" in the back of the Random House dictionary (Flexner, 1983). The confusion sets were selected on the basis of being frequently-occurring in Brown and WSJ, and include mainly homophone confusions (e.g., {*peace*, *piece*}). Several confusion sets not in Random House were added, representing grammatical errors (e.g., {*among*, *between*}) and typographic errors (e.g., {*maybe*, *may be*}).

Results are reported as a percentage of correct classifications on each confusion set, as well as an overall score, which gives the percentage correct for all confusion sets pooled together. When comparing scores, we tested for significance using a McNemar test (Dietterich, 1998) when possible; when data on individual trials was not available (the system comparison), or the comparison was across different test sets (the within/across study), we instead used a test for the difference of two proportions (Fleiss, 1981). All tests are reported for the 0.05 significance level.

5.2. *Pruned versus unpruned*

The first step of the evaluation was to test WinSpell under the same conditions that BaySpell normally runs under—i.e., using the pruned set of features from the feature extractor. We used a random 80-20 split (by sentence) of Brown for the training and test sets. The results of running each algorithm on the 21 confusion sets appear in the 'Pruned' columns of Table 1. Although for a few confusion sets, one algorithm or the other does better, overall WinSpell performs comparably to BaySpell.

The preceding comparison shows that WinSpell is a credible method for this task, but it does not test the claimed strength of Winnow—the ability to deal with large numbers of features. To test this, we modified the feature extractor to do only minimal pruning of features: features were pruned only if they occurred exactly once in the training set (such features are both extremely unlikely to afford good generalizations, and extremely numerous). The hope is that by considering the full set of features, we will pick up many "minor cases"—what

Table 1. Pruned versus unpruned performance of BaySpell and WinSpell. In the pruned condition, the algorithms use the pruned set of features from the feature extractor; in the unpruned condition, they use the full set (excluding features occurring just once in the training set). The algorithms were trained on 80% of Brown and tested on the other 20%. The first two columns give the number of features in the two conditions. Bar graphs show the differences between adjacent columns, with shading indicating significant differences (using a McNemar test at the 0.05 level).

Confusion set	Pruned features	Unpruned features	Pruned BaySpell	Pruned WinSpell	Unpruned BaySpell	Unpruned WinSpell
accept, except	78	849	88.0	87.8	92.0	96.0
affect, effect	36	842	98.0	100.0	98.0	100.0
among, between	145	2706	75.3	75.8	78.0	86.0
amount, number	68	1618	74.8	73.2	80.5	86.2
begin, being	84	2219	95.2	89.7	95.2	97.9
cite, sight, site	24	585	76.5	64.7	73.5	85.3
country, county	40	1213	88.7	90.0	91.9	95.2
fewer, less	6	1613	96.0	94.4	92.0	93.3
I, me	1161	11625	97.8	98.2	98.3	98.5
its, it's	180	4679	94.5	96.4	95.9	97.3
lead, led	33	833	89.8	87.5	85.7	91.8
maybe, may be	86	1639	90.6	84.4	95.8	97.9
passed, past	141	1279	90.5	90.5	90.5	95.9
peace, piece	67	992	74.0	72.0	92.0	88.0
principal, principle	38	669	85.3	84.8	85.3	91.2
quiet, quite	41	1200	95.5	95.4	89.4	93.9
raise, rise	24	621	79.5	74.3	87.2	89.7
than, then	857	6813	93.6	96.9	93.4	95.7
their, there, they're	946	9449	94.8	96.6	94.5	98.5
weather, whether	61	1226	93.4	98.4	98.4	100.0
your, you're	103	2738	90.4	93.6	90.9	97.3
Overall			93.0	93.7	93.8	96.4

Holte, Acker, and Porter (1989) have called "small disjuncts"—that are normally filtered out by the pruning process. The results are shown in the 'Unpruned' columns of Table 1. While both algorithms do better in the unpruned condition, WinSpell improves for almost every confusion set, sometimes markedly, with the result that it outperforms BaySpell in the unpruned condition for every confusion set except one. The results below will all focus on the behavior of the algorithms in the unpruned case.

5.3. System comparison

The previous section shows how WinSpell and BaySpell perform relative to each other; to evaluate them with respect to an external standard, we compared them to other methods reported in the literature. Two recent methods use some of the same test sets as we do, and thus can readily be compared: RuleS, a transformation-based learner (Mangu & Brill, 1997); and a method based on latent semantic analysis (LSA) (Jones & Martin, 1997). We also compare to Baseline, the canonical straw man for this task, which simply identifies

Table 2. System comparison. All algorithms were trained on 80% of Brown and tested on the other 20%; all except LSA used the same 80-20 breakdown. The version of RuleS is the one that uses the same feature set as we do. BaySpell and WinSpell were run in the unpruned condition. The first column gives the number of test cases. Bar graphs show the differences between adjacent columns, with shading indicating significant differences (using a test for the difference of two proportions at the 0.05 level). Ragged-ended bars indicate a difference of more than 15 percentage points. The three 'overall' lines pool the results over different sets of confusion sets.

Confusion set	Test cases	Baseline	LSA	RuleS	BaySpell	WinSpell
accept, except	50	70.0	82.3	88.0	92.0	96.0
affect, effect	49	91.8	94.3	97.9	98.0	100.0
among, between	186	71.5	80.8	73.1	78.0	86.0
amount, number	123	71.5	56.6	78.0	80.5	86.2
begin, being	146	93.2	93.2	95.3	95.2	97.9
cite, sight, site	34	64.7	78.1		73.5	85.3
country, county	62	91.9	81.3	95.2	91.9	95.2
fewer, less	75	90.7			92.0	93.3
I, me	1225	83.0			98.3	98.5
its, it's	366	91.3	92.8		95.9	97.3
lead, led	49	46.9	73.0	89.8	85.7	91.8
maybe, may be	96	87.5			95.8	97.9
passed, past	74	68.9	80.3	83.7	90.5	95.9
peace, piece	50	44.0	83.9	90.0	92.0	88.0
principal, principle	34	58.8	91.2	88.2	85.3	91.2
quiet, quite	66	83.3	90.8	92.4	89.4	93.9
raise, rise	39	64.1	80.6	84.6	87.2	89.7
than, then	514	63.4	90.5	92.6	93.4	95.7
their, there, they're	850	56.8	73.9		94.5	98.5
weather, whether	61	86.9	85.1	93.4	98.4	100.0
your, you're	187	89.3	91.4		90.9	97.3
Overall (14 sets)	1503	71.1	84.5	88.5	89.9	93.5
Overall (18 sets)	2940	70.6	82.8		91.8	95.6
Overall	4336	74.8			93.8	96.4

the most common member of the confusion set during training, and guesses it every time during testing.

The results appear in Table 2. The scores for LSA, taken from Jones and Martin (1997), are based on a different 80-20 breakdown of Brown than that used by the other systems. The scores for RuleS are for the version of that system that uses the same feature set as we do. The comparison shows WinSpell to have significantly higher performance than the other systems. Interestingly, however, Mangu and Brill were able to improve RuleS's overall score from 88.5 to 93.3 (almost up to the level of WinSpell) by making various clever enhancements to the feature set, including using a tagger to assign a word its possible tags in context, rather than merely using the word's complete tag set. This suggests that WinSpell might get a similar boost by adopting this enhanced set of features.

A note on the LSA system: LSA has been reported to do its best for confusion sets in which the words all have the same part of speech. Since this does not hold for all of our confusion sets, LSA's overall score was adversely affected.

5.4. Ablation study

The previous sections demonstrate the superiority of WinSpell over BaySpell for the task at hand, but they do not explain *why* the Winnow-based algorithm does better. At their core, WinSpell and BaySpell are both linear separators (Roth, 1998); is it that Winnow, with its multiplicative update rule, is able to learn a better linear separator than the one given by Bayesian probability theory? Or is it that the non-Winnow enhancements of WinSpell, particularly weighted-majority voting, provide most of the leverage? To address these questions, we ran an ablation study to isolate the contributions of different aspects of WinSpell.

The study was based on the observation that the core computations of Winnow and Bayesian classifiers are essentially isomorphic: Winnow makes its decisions based on a weighted sum of the observed features. Bayesian classifiers make their decisions based not on a sum, but on a product of likelihoods (and a prior probability)—but taking the logarithm of this functional form yields a linear function. With this understanding, we can start with the full BaySpell system; strip it down to its Bayesian essence; map this (by taking the log) to a simplified, non-learning version of WinSpell that performs the identical computation; and then add back the removed aspects of WinSpell, one at a time, to understand how much each contributes to eliminating the performance difference between (the equivalent of) the Bayesian essence and the full WinSpell system.

The experiment proceeds in a series of steps that morph BaySpell into WinSpell:

BaySpell: The full BaySpell method, which includes dependency resolution and interpolative smoothing.

Simplified BaySpell: Like BaySpell, but without dependency resolution. This means that all matching features, even highly interdependent ones, are used in the Bayesian calculation. We do not strip BaySpell all the way down to Naive Bayes, which would use MLE likelihoods, because the performance would then be so poor as to be unrepresentative of BaySpell—and this would undermine the experiment, which seeks to investigate how WinSpell improves on BaySpell (not on a pale imitation thereof).

Simplified WinSpell: This is a minimalist WinSpell, set up to emulate the computation of Simplified BaySpell. It has a 1-layer architecture (i.e., no Weighted Majority layer); it uses a full network (not sparse); it is initialized with Bayesian weights (to be explained momentarily); and it does no learning (i.e., it does not update the Bayesian weights). The Bayesian weights are simply the log of Simplified BaySpell's likelihoods, plus a constant, to make them all non-negative (as required by Winnow). Occasionally, a likelihood will be 0.0, in which case we smooth the log(likelihood) from $-\infty$ to a large negative constant (we used -500). In addition, we add a pseudo-feature to Winnow's representation, which is active for every example, and corresponds to the prior. The weight for this feature is the log of the prior.

1-layer WinSpell: Like Simplified WinSpell, but adds learning. This lets us see whether Winnow's multiplicative update rule is able to improve on the Bayesian feature weights. We ran learning for 5 cycles of the training set.

2-layer WinSpell: Like 1-layer WinSpell, but adds the weighted-majority voting layer to the architecture.

Table 3. Ablation study. Training was on 80% of Brown and testing on the other 20%. The algorithms were run in the unpruned condition. Bar graphs show the differences between adjacent columns, with shading indicating significant differences (using a McNemar test at the 0.05 level).

Confusion set	BaySpell	Simplified BaySpell	1-layer WinSpell	2-layer WinSpell	(Bayesian) WinSpell
accept, except	92.0	92.0	94.0	90.0	96.0
affect, effect	98.0	95.9	98.0	98.0	100.0
among, between	78.0	79.6	77.4	90.9	89.2
amount, number	80.5	78.0	84.6	88.6	85.4
begin, being	95.2	88.4	96.6	98.6	99.3
cite, sight, site	73.5	73.5	79.4	76.5	88.2
country, county	91.9	80.6	91.9	93.5	96.8
fewer, less	92.0	94.7	93.3	96.0	97.3
I, me	98.3	97.9	98.6	99.1	99.5
its, it's	95.9	94.5	95.9	98.4	97.8
lead, led	85.7	91.8	87.8	87.8	93.9
maybe, may be	95.8	96.9	95.8	99.0	99.0
passed, past	90.5	93.2	91.9	87.8	93.2
peace, piece	92.0	84.0	88.0	84.0	88.0
principal, principle	85.3	85.3	82.4	85.3	91.2
quiet, quite	89.4	97.0	92.4	90.9	93.9
raise, rise	87.2	79.5	82.1	82.1	89.7
than, then	93.4	95.7	95.3	97.1	96.7
their, there, they're	94.5	92.7	97.3	98.1	98.2
weather, whether	98.4	96.7	98.4	100.0	100.0
your, you're	90.9	89.3	96.8	97.9	98.9
Overall	93.8	93.1	95.1	96.6	97.2

(Bayesian) WinSpell: Replaces the full network of 2-layer WinSpell with a sparse network. This yields the complete WinSpell algorithm, although its performance is affected by the fact that it started with Bayesian, not uniform weights.

The ablation study used the same 80-20 breakdown of Brown as in the previous section, and the unpruned feature set. The results appear in Table 3. Simplified WinSpell has been omitted from the table, as its results are identical to those of Simplified BaySpell.

The primary finding is that all three measured aspects of WinSpell contribute positively to its improvement over BaySpell; the ranking, from strongest to weakest benefit, is (1) the update rule, (2) the weighted-majority layer, and (3) sparse networks. The large benefit afforded by the update rule indicates that Winnow is able to improve considerably on the Bayesian weights. The likely reason that the Bayesian weights are not already optimal is that the Bayesian assumptions—conditional feature independence and adequate data for estimating likelihoods—do not hold fully in practice. The Winnow update rule can surmount these difficulties by tuning the likelihoods via feedback to fit whatever situation holds in the (imperfect) world. The feedback is obtained from the same training set that is used to set the Bayesian likelihoods. Incidentally, it is interesting to note that the use of a sparse network improves accuracy fairly consistently across confusion sets. The reason it improves accuracy is that, by omitting links for features that never co-occurred with a given target word during training, it effectively sets the weight of such features to 0.0, which is apparently better for accuracy than setting the weight to the log of the Bayesian likelihood (which, in this case, is some *smoothed* version of the 0.0 MLE probability).

A second observation concerns the performance of WinSpell when starting with the Bayesian weights: its overall score was 97.2%, as compared with 96.4% for WinSpell when starting with uniform weights (see Table 2). This suggests that the performance of Winnow can be improved by moving to a hybrid approach in which Bayes is used to initialize the network weights. This hybrid approach is also an improvement over Bayes: in the present experiment, the pure Bayesian approach scored 93.1%, whereas when updates were performed on the Bayesian weights, the score increased to 95.1%.

A final observation from this experiment is that, while it was intended primarily as an ablation study of WinSpell, it also serves as a mini-ablation study of BaySpell. The difference between the BaySpell and Simplified BaySpell columns measures the contribution of dependency resolution. It turns out to be almost negligible, which, at first glance, seems surprising, considering the level of redundancy in the (unpruned) set of features being used. For instance, if the features include the collocation "*a* __ *treaty*", they will also include collocations such as "DET __ *treaty*", "*a* __ $\text{NOUN}_{\text{sing}}$", and so on. Nevertheless, there are two reasons that dependency resolution is of little benefit. First, the features are generated *systematically* by the feature extractor, and thus tend to overcount evidence equally for all words. Second, Naive Bayes is known to be less sensitive to the conditional independence assumption when we only ask it to predict the most probable class (as we do here), as opposed to asking it to predict the exact probabilities for all classes (Duda & Hart, 1973; Domingos & Pazzani, 1997). The contribution of interpolative smoothing—the other enhancement of BaySpell over Naive Bayes—was not addressed in Table 3. However, we investigated this briefly by comparing the performance of BaySpell with interpolative smoothing to its performance with MLE likelihoods (the naive method), as well as a number of alternative smoothing methods. Table 4 gives the overall scores. While the overall score for BaySpell with

Table 4. Overall score for BaySpell using different smoothing methods. The last method, interpolative smoothing, is the one presented here. Training was on 80% of Brown and testing on the other 20%. When using MLE likelihoods, we broke ties by choosing the word with the largest prior (ties arose when all words had probability 0.0). For Katz smoothing, we used absolute discounting (Ney, Essen, & Kneser, 1994), as Good-Turing discounting resulted in invalid discounts for our task. For Kneser-Ney smoothing, we used absolute discounting and the backoff distribution based on the "marginal constraint". For interpolation with a fixed λ, Katz, and Kneser-Ney, we set the necessary parameters separately for each word W_i using deleted estimation.

Smoothing method	Reference	Overall
MLE likelihoods		85.8
Interpolation with a fixed λ	Ney, Essen, & Kneser (1994)	89.8
Laplace-*m*	Kohavi, Becker, & Sommerfield (1997)	90.9
No-matches-0.01	Kohavi, Becker, & Sommerfield (1997)	91.0
Katz smoothing	Katz (1987)	91.6
Kneser-Ney smoothing	Kneser & Ney (1995)	93.4
Interpolative smoothing	Section 3	93.8

interpolative smoothing was 93.8%, it dropped to 85.8% with MLE likelihoods, and was also lower when alternative smoothing methods were tried. This shows that while dependency resolution does not improve BaySpell much over Naive Bayes, interpolative smoothing does have a sizable benefit.

5.5. *Across-corpus performance*

The preceding experiments assumed that the training set will be representative of the test set. For context-sensitive spelling correction, however, this assumption may be overly strong; this is because word usage patterns vary widely from one author to another, or even one document to another. For instance, an algorithm may have been trained on one corpus to discriminate between *desert* and *dessert*, but when tested on an article about the Persian Gulf War, will be unable to detect the misspelling of *desert* in *Operation Dessert Storm*. To check whether this is in fact a problem, we tested the across-corpus performance of the algorithms: we again trained on 80% of Brown, but tested on a randomly-chosen 40% of the sentences of WSJ. The results appear in Table 5. Both algorithms were found to degrade significantly. At first glance, the magnitude of the degradation seems small—from 93.8 to

Table 5. Within- versus across-corpus performance of BaySpell and WinSpell. Training was on 80% of Brown in both cases. Testing for the within-corpus case was on 20% of Brown; for the across-corpus case, it was on 40% of WSJ. The algorithms were run in the unpruned condition. Bar graphs show the differences between adjacent columns, with shading indicating significant differences (using a test for the difference of two proportions at the 0.05 level). Ragged-ended bars indicate a difference of more than 15 percentage points.

Confusion set	Test cases Within	Test cases Across	BaySpell Within	BaySpell Across	WinSpell Within	WinSpell Across
accept, except	50	30	92.0	80.0	96.0	93.3
affect, effect	49	66	98.0	87.9	100.0	95.5
among, between	186	256	78.0	79.3	86.0	87.1
amount, number	123	167	80.5	69.5	86.2	73.7
begin, being	146	174	95.2	89.1	97.9	98.9
cite, sight, site	34	18	73.5	50.0	85.3	55.6
country, county	62	71	91.9	94.4	95.2	95.8
fewer, less	75	148	92.0	94.6	97.3	97.3
I, me	1225	328	98.3	97.9	97.9	92.5
its, it's	366	1277	95.9	95.5	93.3	95.9
lead, led	49	69	85.7	79.7	98.5	98.5
maybe, may be	96	67	95.8	92.5	91.8	89.9
passed, past	74	148	90.5	95.9	95.9	98.0
peace, piece	50	19	92.0	78.9	88.0	73.7
principal, principle	34	30	85.3	70.0	91.2	86.7
quiet, quite	66	20	89.4	65.0	93.9	75.0
raise, rise	39	118	87.2	72.0	89.7	82.2
than, then	514	637	93.4	96.5	95.7	98.4
their, there, they're	850	748	94.5	91.7	98.5	98.1
weather, whether	61	95	98.4	94.7	100.0	96.8
your, you're	187	74	90.9	85.1	97.3	95.9
Overall	4336	4560	93.8	91.2	96.4	95.2

91.2% for the overall score of BaySpell, and 96.4 to 95.2% for WinSpell. However, when viewed as an increase in the error rate, it is actually fairly serious: for BaySpell, the error rate goes from 6.2 to 8.8% (a 42% increase), and for WinSpell, from 3.6 to 4.8% (a 33% increase). In this section, we present a strategy for dealing with the problem of unfamiliar test sets, and we evaluate its effectiveness when used by WinSpell and BaySpell.

The strategy is based on the observation that the test document, though imperfect, still provides a valuable source of information about its own word usages. Returning to the Desert Storm example, suppose the system is asked to correct an article containing 17 instances of the phrase *Operation Desert Storm*, and 1 instance of the (erroneous) phrase *Operation Dessert Storm*. If we treat the *test* corpus as a training document, we will then start by running the feature extractor, which will generate (among others) the collocation:

(3) *Operation* __ *Storm*.

The algorithm, whether BaySpell or WinSpell, should then be able to learn, during its training on the test (qua training) corpus, that feature (3) typically co-occurs with *desert*, and is thus evidence in favor of that word. The algorithm can then use this feature to fix the one erroneous spelling of the phrase in the test set.

It is important to recognize that the system is not "cheating" by looking at the test set; it would be cheating if it were given an answer key along with the test set. What the system is really doing is enforcing consistency across the test set. It can detect sporadic errors, but not systematic ones (such as writing *Operation Dessert Storm* every time). However, it should be possible to pick up at least some systematic errors by also doing regular supervised learning on a training set.

This leads to a strategy, which we call *sup/unsup*, of combining supervised learning on the training set with unsupervised learning on the (noisy) test set. The learning on the training set is *supervised* because a benevolent teacher ensures that all spellings are correct (we establish this simply by assumption). The learning on the test set is *unsupervised* because no teacher tells the system whether the spellings it observes are right or wrong.

We ran both WinSpell and BaySpell with sup/unsup to see the effect on their across-corpus performance. We first needed a test corpus containing errors; we generated one by corrupting a correct corpus. We varied the amount of corruption from 0 to 20%, where p% corruption means we altered a randomly-chosen p% of the occurrences of the confusion set to be a different word in the confusion set.

The sup/unsup strategy calls for training on both a training corpus and a corrupted test corpus, and testing on the uncorrupted test corpus. For purposes of this experiment, however, we split the test corpus into two parts to avoid any confusion about training and testing on the same data. We trained on 80% of Brown plus a corrupted version of 60% of WSJ; and we tested on the uncorrupted version of the other 40% of WSJ.

The results for the 5% level of corruption are shown in Table 6; this level of corruption corresponds to typical typing error rates.[7] The table compares across-corpus performance of each algorithm with and without the additional boost of unsupervised learning on part of the test corpus. Both BaySpell and WinSpell benefit from the unsupervised learning by about the same amount; the difference is that WinSpell suffered considerably less than BaySpell when moving from the within- to the across-corpus condition. As a result, WinSpell, unlike

Table 6. Across-corpus performance of BaySpell and WinSpell using the sup/unsup strategy. Performance is compared with doing supervised learning only. Training in the sup/unsup case is on 80% of Brown plus 60% of WSJ (5% corrupted); in the supervised case, it is on 80% of Brown only. Testing in all cases is on 40% of WSJ. The algorithms were run in the unpruned condition. Bar graphs show the differences between adjacent columns, with shading indicating significant differences (using a McNemar test at the 0.05 level). Ragged-ended bars indicate a difference of more than 15 percentage points.

		BaySpell		WinSpell	
Confusion set	Test cases	Sup only	Sup/unsup	Sup only	Sup/unsup
accept, except	30	80.0	86.7	93.3	86.7
affect, effect	66	87.9	90.9	95.5	93.9
among, between	256	79.3	81.2	87.1	90.6
amount, number	167	69.5	78.4	73.7	87.4
begin, being	174	89.1	94.3	98.9	99.4
cite, sight, site	18	50.0	66.7	55.6	72.2
country, county	71	94.4	95.8	95.8	97.2
fewer, less	148	94.6	93.2	95.9	98.0
I, me	328	97.9	98.5	98.5	99.1
its, it's	1277	95.5	95.6	97.3	97.8
lead, led	69	79.7	75.4	89.9	88.4
maybe, may be	67	92.5	91.0	92.5	97.0
passed, past	148	95.9	96.6	98.0	98.0
peace, piece	19	78.9	84.2	73.7	89.5
principal, principle	30	70.0	76.7	86.7	90.0
quiet, quite	20	65.0	75.0	75.0	90.0
raise, rise	118	72.0	87.3	82.2	89.8
than, then	637	96.5	96.2	98.4	98.3
their, there, they're	748	91.7	90.8	98.1	98.5
weather, whether	95	94.7	95.8	96.8	96.8
your, you're	74	85.1	87.8	95.9	97.3
Overall	4560	91.2	92.4	95.2	96.6

BaySpell, is actually able to recover to its within-corpus performance level, when using the sup/unsup strategy in the across-corpus condition.

It should be borne in mind that the results in Table 6 depend on two factors. The first is the size of the test set: the larger the test set, the more information it can provide during unsupervised learning. The second factor is the percentage corruption of the test set. Figure 2 shows performance as a function of percentage corruption for a representative confusion set, {*amount, number*}. As one would expect, the improvement from unsupervised learning tends to decrease as the percentage corruption increases. For BaySpell's performance on {*amount, number*}, 20% corruption is almost enough to negate the benefit of unsupervised learning.

6. Conclusion

While theoretical analyses of the Winnow family of algorithms have predicted an excellent ability to deal with large numbers of features and to adapt to new trends not seen during

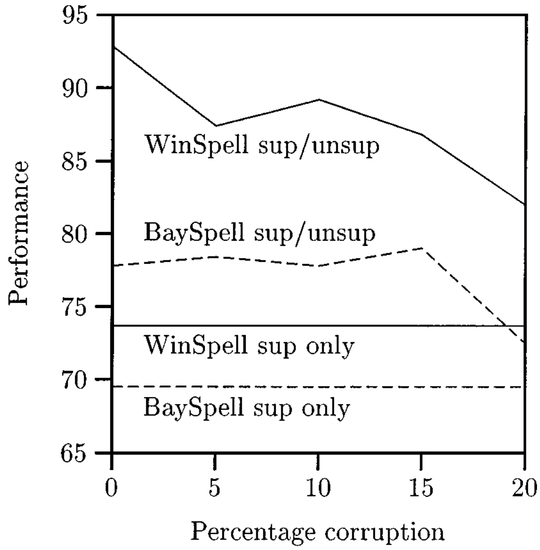

Figure 2. Across-corpus performance of BaySpell (dotted lines) and WinSpell (solid lines) with the sup/unsup strategy and with supervised learning only. The curves show performance as a function of the percentage corruption of the test set. Training in the sup/unsup case is on 80% of Brown, plus 60% of WSJ (corrupted); for the supervised-only case, it is on 80% of Brown only. Testing in both cases is on 40% of WSJ. The algorithms were run for the confusion set {*amount*, *number*} in the unpruned condition.

training, these properties have remained largely undemonstrated. In the work reported here, we have presented an architecture based on Winnow and Weighted Majority, and applied it to a real-world task, context sensitive spelling correction, that has a potentially huge number of features (over 10,000 in some of our experiments). We showed that our algorithm, WinSpell, performs significantly better than other methods tested on this task with a comparable feature set. When comparing WinSpell to BaySpell, a Bayesian statistics-based algorithm representing the state of the art for this task, we found that WinSpell's mistake-driven update rule, its use of weighted-majority voting, and its sparse architecture all contributed significantly to its superior performance.

WinSpell was found to exhibit two striking advantages over the Bayesian approach. First, WinSpell was substantially more accurate than BaySpell when running with full (unpruned) feature sets, outscoring BaySpell on 20 out of 21 confusion sets, and achieving an overall score of over 96%. Second, WinSpell was better than BaySpell at adapting to an unfamiliar test corpus, when using a strategy we presented that combines supervised learning on the training set with unsupervised learning on the test set.

This work represents an application of techniques developed within the theoretical learning community in recent years, and touches upon some of the important issues still under active research. First, it demonstrates the ability of a Winnow-based algorithm to successfully utilize the strategy of expanding the space of features in order to simplify the functional form of the discriminator; this was done in generating collocations as *patterns* of words and part-of-speech tags. The use of this strategy in Winnow shares much the same philosophy—if none of the technical underpinnings—as Support Vector Machines (Cortes & Vapnik, 1995). Second, the two-layer architecture used here is related to various voting and boosting techniques studied in recent years in the learning community (Freund & Schapire, 1995; Breiman, 1994; Littlestone & Warmuth, 1994). The goal is to learn to combine simple learners in a way that improves the overall performance of the system. The focus in the work reported here is on doing this learning in an on-line fashion.

There are many issues still to investigate in order to develop a complete under-standing of the use of multiplicative update algorithms in real-world tasks. One of the important issues this work raises is the need to understand and improve the ability of algorithms to adapt to unfamiliar test sets. This is clearly a crucial issue for algorithms to be used in real systems. A related issue is that of the size and comprehensibility of the output representation. Mangu and Brill (1997), using a similar set of features to the one used here, demonstrate that massive feature pruning can lead to highly compact classifiers, with surprisingly little loss of accuracy. There is a clear tension, however, between achieving a compact representation and retaining the ability to adapt to unfamiliar test sets. Further analysis of this tradeoff is under investigation.

The Winnow-based approach presented in this paper is being developed as part of a research program in which we are trying to understand how networks of simple and slow neuron-like elements can encode a large body of knowledge and perform a wide range of interesting inferences almost instantaneously. We investigate this question in the context of learning knowledge representations that support language understanding tasks. In light of the encouraging results presented here for context-sensitive spelling correction, as well as other recent results (Dagan, Karov, & Roth, 1997; Reddy & Tadepalli, 1997; Roth & Zelenko, 1998), we are now extending the approach to other tasks.

Acknowledgments

We would like to thank Neal Lesh, Grace Ngai, Stan Chen, the reviewers, and the editors for helpful comments on the paper. The second author's research was supported by the Feldman Foundation and a grant from the Israeli Ministry of Science and the Arts; it was done partly while at Harvard University supported by NSF grant CCR-92-00884 and DARPA contract AFOSR-F4962-92-J-0466.

Notes

1. We have tested successfully with up to 40,000 features, but the results reported here use up to 11,000.
2. Each word in the sentence is tagged with its *set* of possible part-of-speech tags, obtained from a dictionary. For a tag to match a word, the tag must be a member of the word's tag set.
3. The maximum-likelihood estimate of $P(f \mid W_i)$ is the number of occurrences of f in the presence of W_i divided by the number of occurrences of W_i.
4. For the purpose of the experimental studies presented here, we do not update the knowledge representation while testing. This is done to provide a fair comparison with the Bayesian method which is a batch approach.
5. This does not interfere with the subsequent updating of the weights—conceptually, we treat a "non-connection" as a link with weight 0.0, which will remain 0.0 after a multiplicative update.

6. The exact form of the decreasing function is unimportant; we interpolate quadratically between 1.0 and 0.67 as a decreasing function of the number of examples.
7. Mays, Damerau, and Mercer (1991), for example, consider error rates from 0.01 to 10% for the same task.

References

Blum, A. (1992). Learning boolean functions in an infinite attribute space. *Machine Learning*, *9*(4), 373–386.

Breiman, L. (1994). Bagging predictors (Technical Report 421). University of California, Berkeley.

Cesa-Bianchi, N., Freund, Y., Helmbold, D.P., & Warmuth, M. (1994). On-line prediction and conversion strategies. *Computational Learning Theory: Eurocolt'93* (pp. 205–216). Oxford University Press.

Chen, S.F., & Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, CA.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297.

Dagan, I., Karov, Y., & Roth, D. (1997). Mistake-driven learning in text categorization. *EMNLP-97, The Second Conference on Empirical Methods in Natural Language Processing* (pp. 55–63).

Dietterich, T.G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, *10*(7), 1895–1924.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, *29*, 103–130.

Duda, R.O., & Hart, P.E. (1973). *Pattern classification and scene analysis*. Wiley.

Fleiss, J.L. (1981). *Statistical methods for rates and proportions*. John Wiley and Sons.

Flexner, S.B. (Ed.). (1983). *Random House unabridged dictionary* (2nd ed.). New York: Random House.

Freund, Y., & Schapire, R.E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *Computational Learning Theory: Eurocolt'95* (pp. 23–37). Springer-Verlag.

Gale, W.A., Church, K.W., & Yarowsky, D. (1993). A method for disambiguating word senses in a large corpus. *Computers and the Humanities*, *26*, 415–439.

Golding, A.R. (1995). A Bayesian hybrid method for context-sensitive spelling correction. *Proceedings of the 3rd Workshop on Very Large Corpora*, Boston, MA.

Golding, A.R., & Schabes, Y. (1996). Combining trigram-based and feature-based methods for context-sensitive spelling correction. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, CA.

Herbster, M., & Warmuth, M. (1995). Tracking the best expert. *Proceedings of the 12th International Conference on Machine Learning* (pp. 286–294). Morgan Kaufmann.

Holte, R.C., Acker, L.E., & Porter, B.W. (1989). Concept learning and the problem of small disjuncts. *Proceedings of the International Joint Conference on Artificial Intelligence*, Detroit.

Jones, M.P., & Martin, J.H. (1997). Contextual spelling correction using latent semantic analysis. *Proceedings of the 5th Conference on Applied Natural Language Processing*, Washington, DC.

Katz, S.M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, *ASSP-35*(3), 400–401.

Kivinen, J., & Warmuth, M.K. (1995). Exponentiated gradient versus gradient descent for linear predictors. *ACM Symp. on the Theory of Computing*.

Kneser, R., & Ney, H. (1995). Improved backing-off for m-gram language modeling. *Proceedings of the International Conf. on Acoustics, Speech, and Signal Processing* (Vol. 1, pp. 181–184).

Kohavi, R., Becker, B., & Sommerfield, D. (1997). Improving simple Bayes. *Proceedings of the European Conference on Machine Learning*.

Kučera, H., & Francis, W.N. (1967). *Computational analysis of present-day American English*. Providence, RI: Brown University Press.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, *24*(4), 377–439.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2*, 285–318.

Littlestone, N. (1991). Redundant noisy attributes, attribute errors, and linear threshold learning using Winnow. *Proceedings of the 4th Annual Workshop on Computational Learning Theory* (pp. 147–156). Morgan Kaufmann.

Littlestone, N. (1995). Comparing several linear-threshold learning algorithms on tasks involving superfluous attributes. *Proceedings of the 12th International Conference on Machine Learning* (pp. 353–361). Morgan Kaufmann.

Littlestone, N., & Warmuth, M.K. (1994). The weighted majority algorithm. *Information and Computation*, *108*(2), 212–261.

Mangu, L., & Brill, E. (1997). Automatic rule acquisition for spelling correction. *Proceedings of the 14th International Conference on Machine Learning*. Morgan Kaufmann.

Marcus, M.P., Santorini, B., & Marcinkiewicz, M. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, *19*(2), 313–330.

Mays, E., Damerau, F.J., & Mercer, R.L. (1991). Context based spelling correction. *Information Processing and Management*, *27*(5), 517–522.

Ney, H., Essen, U., & Kneser, R. (1994). On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, *8*, 1–38.

Ng, H.T., & Lee, H.B. (1996). Integrating multiple knowledge sources to disambiguate word sense: An examplar-based approach. *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, CA.

Powers, D. (1997). Learning and application of differential grammars. *Proceedings of the ACL Special Interest Group in Natural Language Learning*, Madrid.

Reddy, L., & Tadepalli, P. (1997). Active learning with committees for text categorization. *Proceedings of the National Conference on Artificial Intelligence* (pp. 602–608).

Roth, D. (1998). Learning to resolve natural language ambiguities: A unified approach. *Proceedings of the National Conference on Artificial Intelligence* (pp. 806–813).

Roth, D., & Zelenko, D. (1998). Part of speech tagging using a network of linear separators. *COLING-ACL 98, The 17th International Conference on Computational Linguistics* (pp. 1136–1142).

Valiant, L.G. (1994). *Circuits of the mind*. Oxford University Press.

Valiant, L.G. (1995). Rationality. *Workshop on Computational Learning Theory* (pp. 3–14).

Yarowsky, D. (1994). Decision lists for lexical ambiguity resolution: Application to accent restoration in Spanish and French. *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, Las Cruces, NM.