Journal of Machine Learning Research 13 (2012) 1891-1926 Submitted 12/10; Revised 12/11; Published 6/12

Confidence-Weighted Linear Classification for Text Categorization

Koby Crammer KOBY@EE.TECHNION.AC.IL
Department of Electrical Engineering
The Technion
Haifa 32000, Israel

Mark Dredze MDREDZE@CS.JHU.EDU
Human Language Technology Center of Excellence
Johns Hopkins University
Baltimore, MD 21211, USA

Fernando Pereira PEREIRA@GOOGLE.COM
Google, Inc.
1600 Amphiteatre Pkwy
Mountain View, CA 94043, USA

Editor: Michael Collins

Abstract

Confidence-weighted online learning is a generalization of margin-based learning of linear classifiers in which the margin constraint is replaced by a probabilistic constraint based on a distribution over classifier weights that is updated online as examples are observed. The distribution captures a notion of confidence on classifier weights, and in some cases it can also be interpreted as replacing a single learning rate by adaptive per-weight rates. Confidence-weighted learning was motivated by the statistical properties of natural-language classification tasks, where most of the informative features are relatively rare. We investigate several versions of confidence-weighted learning that use a Gaussian distribution over weight vectors, updated at each observed example to achieve high probability of correct classification for the example. Empirical evaluation on a range of text-categorization tasks show that our algorithms improve over other state-of-the-art online and batch methods, learn faster in the online setting, and lead to better classifier combination for a type of distributed training commonly used in cloud computing.

Keywords: online learning, confidence prediction, text categorization

1. Introduction

While online learning is among the oldest approaches to machine learning, starting with the perceptron algorithm (Rosenblatt, 1958), it is still one of the most popular and and successful for many practical tasks. In online learning, algorithms operate in rounds, whereby the algorithm is shown a single example for which it must first make a prediction and then update its hypothesis once it has seen the correct label. While predictions traditionally take the form of either positive or negative labels (binary classification), algorithms have been extended to a variety of multi-class, regression, ranking and structured prediction problems. By operating one example at a time, online methods are fast, simple, make few assumptions about the data, and perform fairly well across many domains and tasks. For those reasons, online methods are often favored for large data problems, and they are also a natural fit for systems that learn from interaction with a user or another system. In addition to their nice empirical properties, online algorithms have been analyzed in the mistake bound model (Littlestone, 1989), which supports both theoretical and empirical comparisons of performance. Cesa-Bianchi and Lugosi (2006) provide an in-depth analysis of online learning algorithms.

Much of the machine learning in natural-language processing (NLP) is based on linear classifiers over very high dimension sparse representations of the input trained on large training sets. These properties make online learning a natural choice. Extensions of online learning to structured problems (Collins, 2002; McDonald et al., 2004) achieved some of the best results in structured tasks such as part-of-speech tagging (Collins, 2002; Shen et al., 2007), text segmentation (McDonald et al., 2005a), noun-phrase chunking (Collins, 2002), parsing (McDonald et al., 2005b; Carreras et al., 2008), and machine translation (Chiang et al., 2008). Popular online methods for those tasks include the perceptron (Rosenblatt, 1958), passive-aggressive (Crammer et al., 2006a) and exponentiated gradient (Globerson et al., 2007).

Online learning algorithms are typically used as blackboxes in NLP, without consideration of the peculiarities of natural language. Feature representations of text for tasks from spam filtering to parsing need to capture the variety of words, word combinations, and word attributes in the text, yielding very high-dimensional feature vectors, even though most of the features are absent in most texts. Nevertheless, those many rare features are very informative about the examples that contain them; indeed, features that occur frequently are typically less informative, hence the common use of stop-lists of frequent words such as function words, and of tf-idf term weighting.[1] In Figure 1, we show the most predictive features for a simple NLP classification task and their frequency in data. Notice that while some predictive features are very common, most are relatively rare, indicating that modeling even infrequent features may be useful for learning. Therefore, it is worth investigating whether learning algorithms for linear classifiers could be improved to take advantage of these particularities of natural language data.

The foregoing motivation led us to propose *confidence-weighted* (CW) learning, a class of online learning methods that maintain a probabilistic measure of confidence in each weight. Less confident weights are updated more aggressively than more confident ones. Weight confidence is formalized with a Gaussian distribution over weight vectors, which is updated for each new training example so that the probability of correct classification for that example under the updated distribution meets a specified confidence. The result is an algorithm with superior classification accuracy over state-of-the-art online and batch baselines, faster learning, and new classifier combination methods for parallel training.

While our motivation for CW learning is from observations about NLP problems, the approach makes no assumptions about the input space and can be applied to other machine learning problems (Ma et al., 2009).

This paper brings together two types of confidence-weighted algorithms originally introduced by Dredze et al. (2008) and Crammer et al. (2008). In addition to a unified presentation, we include alternative formulations of the diagonal covariance algorithms along with empirical results. We also include further empirical evidence of the strength of these methods and an analysis of algorithmic behavior on NLP problems.

1. We note that data sparsity is different from model sparsity. Sparsifying regularizers, such as those that constrain the L_1 norm of weight vectors (Andrew and Gao, 2007; Gao et al., 2007). are often proposed to remove redundant features in very high-dimensional data, but they are complementary to the methods we present here to learn better in the presence of many rare but relevant features.

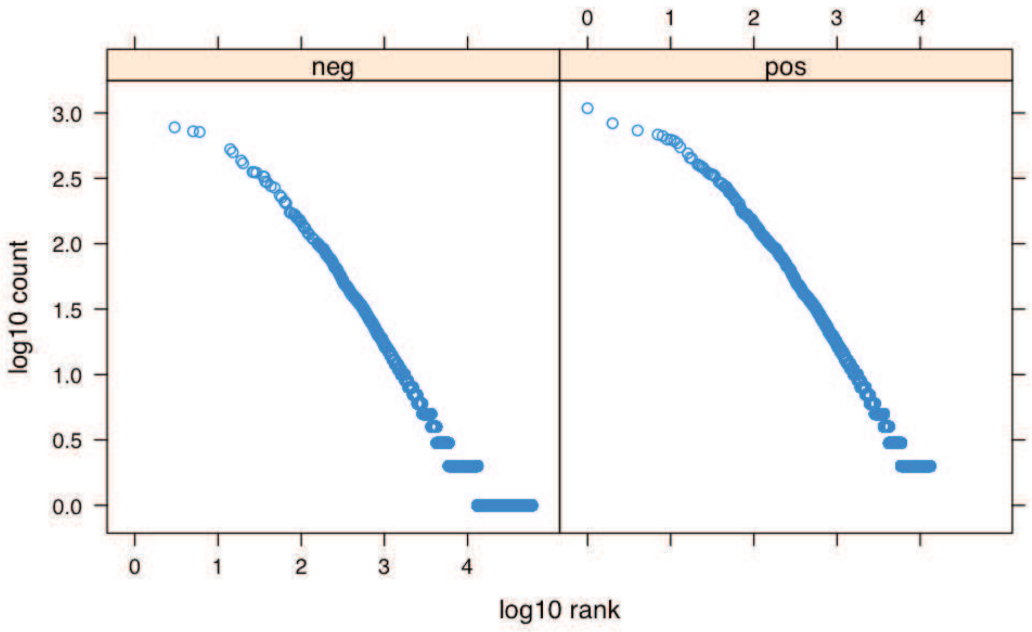

Figure 1: The top quartile of negative (left) and positive (right) features as ranked by mutual information with the label for sentiment data (described in Section 7). The x-axis is their (log) rank by mutual information and the y-axis is their total (log) count in the data. While some very frequent features are useful for predicting the label (high on the curve) there are a large number of low frequency features (low on the curve) that are still useful for learning. A sparse model would likely remove these low frequency features despite their predictive value.

We begin with a discussion of the motivating particularities of natural language data. We then introduce the confidence-weighted framework. From this framework we derive two types of algorithm following different formulations of the main constraint, each with a full covariance and several diagonalized versions. A series of experiments shows CW learning's empirical benefits and an analysis reveals how algorithmic properties manifest themselves empirically. We conclude with a discussion of related work.

2. Characteristics of NLP Data

Extensive experience with building classifiers for a wide range of language processing tasks shows that correct classification requires many specific features, including the presence at specified positions of particular words, affixes, or word combinations (such as bigrams) in the example to be classified. An individual example has a very small fraction of those features, but collectively, examples to be classified may involve a very large number of features ($10^6 - 10^9$), most of which only occur in a few examples. The vector representation of the typical example is a very sparse high dimensional vector where only a small fraction of elements is nonzero, and feature frequencies have a heavy-tailed distribution (Figure 1).

Online algorithms do well with large numbers of features and examples, but they are not designed specifically for very sparse examples with a heavy-tailed feature frequency distribution. This can have a detrimental effect on learning. Typical linear classifier training algorithms update the weights of binary features only when they occur. The result is many updates for frequent features and few updates for rare features. Similarly, features that occur early in the data stream take more responsibility for correct prediction than those observed later. The result is a model that could have good weight estimates for common features but inaccurate weights for the great majority of features, which occur relatively rarely.

An illustrative case arises in sentiment classification. In this task, a product review is represented as n-grams and the goal is to label the review as being positive or negative about the product. Consider a positive review that simply read *"I liked this author."* An online update would increase the weight of both "liked" and "author." Since both are common words, over several examples the algorithm would converge to the correct values, a positive weight for "liked" and zero weight for "author." Now consider a slightly modified negative example: *"I liked this author, but found the book dull."* Since "dull" is a rare feature, the algorithm has a poor estimate of its weight. An update would decrease the weight of both "liked" and "dull." The algorithm does not know that "dull" is rare and the changed behavior is likely caused by the poorly estimated rare feature ("dull") instead of the well estimated common feature ("liked.") An algorithm that maintains no information about the relative frequency or of second order information about features would attribute equal negative weight to both "liked" and "dull", which slows convergence.

This example demonstrates how a lack of memory for previous examples—a property that allows online learning—can hurt learning. A simple solution is to augment an online algorithm with additional information, a memory of past examples. Specifically, the algorithm can maintain a *confidence value for each feature weight*. For example, assuming binary features, the algorithm could keep a count of the number of times each feature has been observed or how many times each weight has been updated. The larger the count, the more confidence we have in the weight of that feature. These estimates are then used to influence weight updates. Instead of equally updating every feature weight for the on-features of an example, the update favors changing low-confidence weights more aggressively than high-confidence ones. At each update, the confidence in the weights of observed features is increased, which will focus the update on the low confidence weights. In the example above, the update would decrease the weight of "dull" but make only a small change to "liked" since the algorithm already has a good estimate of this weight.

In the next section, we use this motivation from language data to present a new family of learning algorithms that associate a confidence value with each weight. For now, we wish to dispel two potential misinterpretations of the preceding very informal argument. First, while our approach is motivated by learning with sparse binary features with a heavy-tailed frequency distribution, the algorithms do not depend on those assumptions. Second, our notion of weight confidence is based on a probabilistic interpretation of passive-aggressive online learning, which differs from the more familiar Bayesian learning for linear classifiers. Nevertheless, analogously to Bayesian learning, it can be used to provide a useful notion of prediction confidence through a margin distribution (Dredze and Crammer, 2008a,b; Dredze et al., 2010).

A summary of the notation used throughout this paper appears in Table 1.

x_i	Example on round i
$\hat{y}_i$	Prediction on round i
y_i	Label on round i
w_i	Weight vector on round i
μ_i	The mean of the distribution on round i
Σ_i	The covariance matrix of the distribution on round i
m_i	Margin on round i
v_i	Margin variance on round i
η	Confidence level
ϕ	The free parameter for CW, defined as $\phi = \Phi^{-1}(\eta)$

Table 1: A reference table for notation used throughout the paper.

3. Online Learning of Linear Classifiers

Online algorithms operate in rounds, where each round corresponds to a single example. On round i the algorithm receives an example $x_i \in \mathbb{R}^d$ to which it applies its current prediction rule to produce a prediction $\hat{y}_i \in \{-1,+1\}$ (for binary classification). It then receives the true label $y_i \in \{-1,+1\}$ and suffers a loss $\ell(y_i,\hat{y}_i)$, which in this work will be the zero-one loss: $\ell(y_i,\hat{y}_i) = 1$ if $y_i \neq \hat{y}_i$ and $\ell(y_i,\hat{y}_i) = 0$ otherwise. The algorithm then updates its prediction rule and proceeds to the next round. For online evaluations, error is reported as the total loss ℓ on the training data and in batch evaluations, error is reported on held out data.

As is common in linear classification, our prediction rules are linear threshold functions

$$f_w(x) : f_w(x) = \text{sign}(x \cdot w) .$$

Two functions f_w and f_{cw} are the same for non-negative c. Thus, we can identify f_w with w, which we will do in what follows.

The signed *margin* of an example (x,y) with respect to a specific classifier w is defined to be $y(w \cdot x)$. The sign of the margin is positive iff the classifier w correctly predicts the true label y. The absolute value of the margin $|y(w \cdot x)| = |w \cdot x|$ can be thought of as the *confidence*[2] in the prediction, with larger positive values corresponding to more confident correct predictions. We denote the margin at round i by $m_i = y_i(w_i \cdot x_i)$.

A variety of linear classifier training algorithms, including the perceptron and linear support vector machines, restrict w to be a linear combination of the input examples. Online algorithms of that kind typically have updates of the form

$$w_{i+1} = w_i + \alpha_i y_i x_i \ , \tag{1}$$

for some non-negative coefficients α_i.

In this paper we focus on passive-aggressive (PA) updates (Crammer et al., 2006a) for linear classifiers. After predicting with w_i on the ith round and receiving the true label y_i, the algorithm

2. Note that we use the term "confidence" here as is commonly used in the literature to refer to the size of the margin. This should not be confused with the idea of weight confidence used in this work. In fact, while margin size is often taken as prediction confidence, such as in active learning (Tong and Koller, 2001), this interpretation is open to debate.

updates the prediction function such that the example (x_i, y_i) will be classified correctly with a fixed margin (which can always be scaled to 1):

$$w_{i+1} = \min_{w} \quad \frac{1}{2} \|w_i - w\|^2 \quad \text{s.t.} \quad y_i (w \cdot x_i) \geq 1 \ . \tag{2}$$

The general form of this problem is to enforce some learning constraints, in this case a prediction margin on the example, while minimizing the divergence to the current weights, which are assumed to be good since they encapsulate all previously observed examples. Solving this problem leads to an update of the form given by (1) with coefficient α_i defined on each round as:

$$\alpha_i = \frac{\max\{1 - y_i (w_i \cdot x_i), 0\}}{\|x_i\|^2} \ , \tag{3}$$

Like the perceptron, this is a mistake driven update, whereby $\alpha_i > 0$ iff the learning condition was not met, ie. the example was not classified with a margin of at least 1. Note that the numerator of (3) is the hinge loss, which is zero only if the example is classified with a margin of 1. In practice, slack variables are introduced for non-separable data, restricting (3) as $\max\{\alpha_i, C\}$, for some free parameter C.

Crammer et al. (2006a) provide a theoretical analysis of algorithms of this form, which have been shown to work well in a variety of applications (McDonald et al., 2004, 2005a; Chiang et al., 2008).

4. Distributions over Classifiers

Following the motivation of Section 2, we need a notion of confidence for the weight vector w maintained by an online learner for linear classifiers. Before any examples are seen, all of the weights in w are equally uncertain. As examples are observed, the confidence in the weights of features that are often active should increase faster than the confidence in the weights of rarely seen features.

Our concrete implementation of this idea is to represent the state of the learner with a probability density over w, specifically a Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ with mean $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$. The values μ_p and $\Sigma_{p,p}$ represent knowledge of and confidence in the weight of feature p. The smaller $\Sigma_{p,p}$, the more confidence we have in the mean weight value μ_p. Each covariance term $\Sigma_{p,p'}$ captures our knowledge of the interaction between features p and p'. The Gaussian distribution naturally matches our intuition for confidence, as the covariance of the distribution is inversely proportional to our confidence: the smaller the determinant of the covariance, the less we expect the true weight value to deviate from the current estimate. This Gaussian representation is illustrated in Figure 2, which shows a Gaussian distribution over two-dimensional weight vectors. The black line represents an example $x = (0.5, 1)$, $y = +1$, which divides the space between classifiers that correctly classify this point (blue crosses below) and those that classify it incorrectly (green dots above).

In the CW model, the traditional signed margin $y(w \cdot x)$ becomes a univariate Gaussian random variable M, where the mean of the distribution is the signed margin,

$$M \sim \mathcal{N}\left(y(\mu \cdot x), x^\top \Sigma x\right) \ . \tag{4}$$

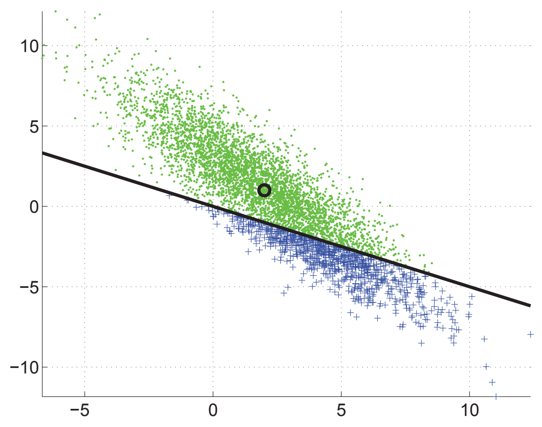

Figure 2: Gaussian distribution over two-dimensional weight vectors. Points above the black line (green dots) incorrectly classify the example $((0.5,1),+1)$ and points below the line (blue crosses) classify it correctly. The density around a point is proportional to its relative weight. The black circle marks the mean of the Gaussian.

There are several ways to make predictions in this framework. A Gibbs predictor samples from the distribution a single weight vector w, which is equivalent to drawing a margin value using (4), and takes its sign as the prediction. Other alternatives use averaging rather than sampling. For example, we can use the average weight vector $\mathrm{E}[w] = \mu$, as is done in Bayes point machines (Herbrich et al., 2001), which use a single weight vector to approximate a distribution. Alternatively, we can use the average margin $\mathrm{E}[M]$. These two approaches are equivalent by linearity of expectation, $\mathrm{E}[w \cdot x] = \mu \cdot x$. Another approach estimates $\mathrm{E}[\mathrm{sign}(M)]$ from many draws of w for fixed μ,Σ, and x. Since the sign function attains only two values (-1 or $+1$) this is equivalent to computing the probability of a *correct* prediction (not a large margin prediction), given by

$$\Pr[M \geq 0] = \Pr_{w\sim\mathcal{N}(\mu,\Sigma)}[y(w \cdot x) \geq 0] .$$

When possible we omit the explicit dependence on the distribution parameters and simply write $\Pr[y(w \cdot x) \geq 0]$. If the probability is larger than half, then the (weighted) majority votes for $y = +1$, otherwise, for $y = -1$. Note that from the discussion below this prediction rule is equivalent to the previous two. Conceptually, it is useful to think of prediction as drawing a weight vector w from the distribution, ie. $w \sim \mathcal{N}(\mu,\Sigma)$, and predicting the label according to the sign of $w \cdot x$. However, as we said above, the average of many such draws is equivalent to the simple prediction rule $\mathrm{sign}(\mu \cdot x)$, which we will use in what follows.

5. Learning Confidence-Weighted Classifiers

In the previous section we formalized our confidence-weighted learning framework in terms of Gaussian distributions over weight vectors. In this section we discuss how to learn such distributions.

CW is an online learning algorithm, so on round i the algorithm receives example x_i for which it issues a prediction $\hat{y}_i$.[3] The algorithm predicts $\hat{y}_i$ as $\mathrm{sign}\left(\mu_i \cdot x_i\right)$, which is equivalent to averaging the predictions of many sampled weight vectors from the distribution. On being presented with the label y_i, the algorithm adjusts the distribution to enforce a learning condition. Following the intuition underlying the PA algorithms of Crammer et al. (2006a), we require that an update achieves both a large margin on the example and minimizes the change in weights. In this case, a large prediction margin is formalized as ensuring that the probability of a correct prediction for training example i is no smaller than the confidence level $\eta \in [0,1]$:

$$\Pr\left[y_i\left(w \cdot x_i\right) \geq 0\right] \geq \eta \ .$$

Minimization of weight changed is enforced by finding a new distribution closest in the KL divergence[4] sense to the current distribution $\mathcal{N}\left(\mu_i, \Sigma_i\right)$. Thus, on round i, the algorithm updates the distribution by solving the following optimization problem:

$$\left(\mu_{i+1}, \Sigma_{i+1}\right) = \min \mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}\left(\mu, \Sigma\right) \parallel \mathcal{N}\left(\mu_i, \Sigma_i\right)\right) \tag{5}$$

$$\text{s.t. } \Pr\left[y_i\left(w \cdot x_i\right) \geq 0\right] \geq \eta \ . \tag{6}$$

This update can be understood as a probabilistic counterpart of the PA objective (2).

We now develop both the objective and the constraint of this optimization problem following Boyd and Vandenberghe (2004, page 158). We start with the objective (5) and write the KL divergence between two Gaussians as

$$\mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}\left(\mu_0, \Sigma_0\right) \parallel \mathcal{N}\left(\mu_1, \Sigma_1\right)\right) = \frac{1}{2}\left(\log\left(\frac{\det \Sigma_1}{\det \Sigma_0}\right) + \mathrm{Tr}\left(\Sigma_1^{-1}\Sigma_0\right) + \left(\mu_1 - \mu_0\right)^\top \Sigma_1^{-1}\left(\mu_1 - \mu_0\right) - d\right) \ .$$

We now proceed with the constraint in (6). As noted above, under the distribution $\mathcal{N}\left(\mu, \Sigma\right)$, the margin for $\left(x_i, y_i\right)$ has a Gaussian distribution with mean

$$m_i = y_i\left(\mu_i \cdot x_i\right) \ , \tag{7}$$

and variance

$$\sigma_i^2 = v_i = x_i^\top \Sigma_i x_i \ . \tag{8}$$

Thus the probability of a *wrong* classification is

$$\Pr\left[M \leq 0\right] = \Pr\left[\frac{M-m}{\sigma} \leq \frac{-m}{\sigma}\right] \ .$$

Since $\left(M-m\right)/\sigma$ is a normally distributed random variable, the above probability equals $\Phi\left(-m/\sigma\right)$, where

$$\Phi(u) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{u} e^{-v^2} dv \ ,$$

3. For a related batch formulation of CW learning, see recent work of Crammer et al. (2009b).

4. $\mathrm{D}_{\mathrm{KL}}\left(p(x) \parallel q(x)\right) = \int p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$.

is the cumulative Gaussian distribution. Thus we can rewrite (6) as

$$\frac{-m}{\sigma} \leq \Phi^{-1}(1-\eta) = -\Phi^{-1}(\eta) \ .$$

Substituting m and σ by their definitions and rearranging terms we obtain

$$y_i(\mu \cdot x_i) \geq \phi\sqrt{x_i^\top \Sigma x_i} \ ,$$

where $\phi = \Phi^{-1}(\eta)$. To conclude the update rule solves the following optimization problem:

$$(\mu_{i+1}, \Sigma_{i+1}) = \arg\min_{\mu,\Sigma} \frac{1}{2}\log\left(\frac{\det\Sigma_i}{\det\Sigma}\right) + \frac{1}{2}\mathrm{Tr}\left(\Sigma_i^{-1}\Sigma\right) + \frac{1}{2}(\mu_i - \mu)^\top \Sigma_i^{-1}(\mu_i - \mu)$$
$$\text{s.t. } y_i(\mu \cdot x_i) \geq \phi\sqrt{x_i^\top \Sigma x_i} \ . \tag{9}$$

Conceptually, this is a large-margin constraint, where the value of the margin requirement depends on the example x_i via a quadratic form.

Unfortunately, this constraint is not convex in Σ since the term $\sqrt{x_i^\top \Sigma x_i}$ is concave in Σ. We propose two alternatives to obtain a convex constraint: linearization (Section 5.1) and change of variables (Section 5.2). Additionally, we propose few alternatives to solve the learning optimization problem restricted to *diagonal* matrices in Section 6.

5.1 Linearization of the Constraint

In out first approach to obtain a convex problem we simply linearize the constraint of (9) by omitting the square root to obtain the revised optimization problem.

$$(\mu_{i+1}, \Sigma_{i+1}) = \arg\min \frac{1}{2}\log\left(\frac{\det\Sigma_i}{\det\Sigma}\right) + \frac{1}{2}\mathrm{Tr}\left(\Sigma_i^{-1}\Sigma\right) + \frac{1}{2}(\mu_i - \mu)^\top \Sigma_i^{-1}(\mu_i - \mu)$$
$$\text{s.t. } y_i(\mu \cdot x_i) \geq \phi\left(x_i^\top \Sigma x_i\right) \ . \tag{10}$$

We call this formulation `var`, since we have replaced the standard deviation in the constraint with the variance. This formulation was introduced by Dredze et al. (2008). The following lemma summarizes the solution of this formulation,

Lemma 1 *The optimal solution of this form is,*

$$\mu_{i+1} = \mu_i + \alpha y_i \Sigma_i x_i$$
$$\Sigma_{i+1}^{-1} = \Sigma_i^{-1} + 2\alpha\phi x_i x_i^\top \ ,$$

where the value of the parameter α (a Lagrange multiplier) is given by

$$\alpha_i = \max\left\{0, \frac{-(1+2\phi m_i) + \sqrt{(1+2\phi m_i)^2 - 8\phi(m_i - \phi v_i)}}{4\phi v_i}\right\} \ .$$

where $m_i = y_i(\mu_i \cdot x_i)$ (see (7)) and $v_i = x_i^\top \Sigma_i x_i$ (see (8)).

The derivation appears in Section 5.1.1 below. The resulting algorithm is shown in Figure 1, where the update uses (11) and (13) to update the distribution with coefficients β_i ((15)) and α_i ($\max\{(18), 0\}$.)

Algorithm 1 Binary CW Online Algorithm. The two versions of the Confidence-Weighted algorithm: (1) linearization and (2) change of variables. The numbers in parentheses refer to equations in the text, where more detail can be found.

```
Input: $\eta \in [0.5, 1]$
Initialize:
    $\mu_1 = 0 \; , \; \Sigma_1 = I,$
    $\phi = \Phi^{-1}(\eta) \; , \; \phi' = 1 + \phi^2/2 \; , \; \phi'' = 1 + \phi^2 \; .$
for $i = 1, 2 \ldots$ do
    Receive a training example $x_i \in \mathbb{R}^d$
    Compute Gaussian margin distribution $m_i \sim \mathcal{N}\left(\mu_i \cdot x_i, x_i^\top \Sigma_i x_i\right)$
    Receive true label $y_i$
    Suffer loss $\ell_i = 1$ iff $y_i \mathrm{E}[\mathrm{sign}(m_i)] \le 0$
    Compute Update:
    • Define:  $m_i = y_i(\mu_i \cdot x_i)$ (7)   $v_i = x_i^\top \Sigma_i x_i$ (8)
    • Linearization:
        $\alpha_i = \max\left\{0, \frac{-(1+2\phi m_i) + \sqrt{(1+2\phi m_i)^2 - 8\phi(m_i - \phi v_i)}}{4\phi v_i}\right\}$   (18)
        $\beta_i = \frac{2\alpha_i\phi}{1+2\alpha\phi v_i}$   (15)
    • Change of Variables:
        $v_i^+ = \left(\frac{-\alpha v_i\phi + \sqrt{\alpha^2 v_i^2\phi^2 + 4v_i}}{2}\right)^2$   (28)
        $\alpha_i = \max\left\{0, \frac{-m_i\phi' + \sqrt{m_i^2\frac{\phi^4}{4} + v_i\phi^2\phi''}}{v_i\phi''}\right\}$   (31)
        $\beta_i = \frac{\alpha_i\phi}{\sqrt{v_i^+} + v_i\alpha_i\phi}$   (27)
    Update
        $\mu_{i+1} = \mu_i + \alpha_i y_i \Sigma_i x_i$   (11,20)
        $\Sigma_{i+1} = \Sigma_i - \beta_i \Sigma_i x_i x_i^\top \Sigma_i$   (14,25)
end for
Output: Final $\mu$ and $\Sigma$
```

5.1.1 DERIVATION OF LEMMA 1

The optimization objective is convex in μ and Σ simultaneously and the constraint became linear, so any convex optimization solver van be used to solve this problem. The Lagrangian for this optimization is

$$\begin{aligned}\mathcal{L} &= \frac{1}{2}\log\left(\frac{\det\Sigma_i}{\det\Sigma}\right)+\frac{1}{2}\mathrm{Tr}\left(\Sigma_i^{-1}\Sigma\right)+\frac{1}{2}\left(\mu_i-\mu\right)^\top\Sigma_i^{-1}\left(\mu_i-\mu\right)\\ &\quad+\alpha\left(-y_i\left(\mu\cdot x_i\right)+\phi\left(x_i^\top\Sigma x_i\right)\right)\ .\end{aligned}$$

Taking partial derivatives, we know that at the optimum, we must have

$$\frac{\partial}{\partial\mu}\mathcal{L}=\Sigma_i^{-1}\left(\mu-\mu_i\right)-\alpha y_i x_i=0\ .$$

Assuming Σ_i is non-singular and rearranging terms we get

$$\mu_{i+1}=\mu_i+\alpha y_i\Sigma_i x_i\ . \tag{11}$$

At the optimum, we must also have

$$\frac{\partial}{\partial\Sigma}\mathcal{L}=-\frac{1}{2}\Sigma^{-1}+\frac{1}{2}\Sigma_i^{-1}+\phi\alpha x_i x_i^\top=0\ , \tag{12}$$

and solving for Σ^{-1} we obtain

$$\Sigma_{i+1}^{-1}=\Sigma_i^{-1}+2\alpha\phi x_i x_i^\top\ . \tag{13}$$

Before proceeding, we observe that (13) computes Σ_{i+1}^{-1} as the sum of a rank-one positive semi-definite (PSD) matrix and Σ_i^{-1} . Thus, if Σ_i^{-1} is PSD, so are Σ_{i+1}^{-1} and Σ_{i+1} thus Σ_i is indeed non-singular, as assumed above. The update guarantees that the eigenvalues of the inverse-covariance matrix always increase.

Finally, we compute the inverse of (13) using the Woodbury identity (Petersen and Pedersen, 2008, Equation 135) and get

$$\begin{aligned}\Sigma_{i+1} &= \left(\Sigma_i^{-1}+2\alpha\phi x_i x_i^\top\right)^{-1}\\ &= \Sigma_i-\Sigma_i x_i\left(\frac{1}{2\alpha\phi}+x_i^\top\Sigma_i x_i\right)^{-1}x_i^\top\Sigma_i\\ &= \Sigma_i-\Sigma_i x_i\frac{2\alpha\phi}{1+2\alpha\phi v_i}x_i^\top\Sigma_i\\ &= \Sigma_i-\beta_i\Sigma_i x_i x_i^\top\Sigma_i\ ,\end{aligned} \tag{14}$$

where

$$\begin{aligned}v_i &= x_i^\top\Sigma_i x_i\\ \beta_i &= \frac{2\alpha\phi}{1+2\alpha\phi v_i}\ .\end{aligned} \tag{15}$$

The KKT conditions for the optimization imply that the either $\alpha = 0$, and no update is needed, or the constraint in (10) is an equality after the update. Substituting (11) and (14) into the equality version of (10), we obtain

$$y_i \left(x_i \cdot \left(\mu_i + \alpha y_i \Sigma_i x_i \right) \right) = \phi \left(x_i^\top \left(\Sigma_i - \Sigma_i x_i \beta_i x_i^\top \Sigma_i \right) x_i \right) . \tag{16}$$

Rearranging terms we get

$$y_i \left(x_i \cdot \mu_i \right) + \alpha x_i^\top \Sigma_i x_i = \phi x_i^\top \Sigma_i x_i - \phi v_i^2 \beta_i . \tag{17}$$

Substituting (7), (8), and (15) into (17) we obtain

$$m_i + \alpha v_i = \phi v_i - \phi v_i^2 \frac{2\alpha\phi}{1 + 2\alpha\phi v_i} .$$

We multiply both sides by $1 + 2\alpha\phi v_i$ and get

$$\left(m_i + \alpha v_i \right) \left(1 + 2\alpha\phi v_i \right) = \phi v_i \left(1 + 2\alpha\phi v_i \right) - 2\alpha\phi^2 v_i^2 .$$

Rearranging the terms we obtain,

$$\begin{aligned} 0 &= m_i + \alpha v_i + 2\alpha\phi v_i m_i + 2\alpha^2 \phi v_i^2 - \phi v_i \\ &= \alpha^2 \left(2\phi v_i^2 \right) + \alpha v_i \left(1 + 2\phi m_i \right) + \left(m_i - \phi v_i \right) . \end{aligned}$$

The above equality is a quadratic equation in α. Its smaller root is always negative and thus is not a valid Lagrange multiplier. Let γ_i be its larger root:

$$\gamma_i = \frac{-\left(1 + 2\phi m_i \right) + \sqrt{\left(1 + 2\phi m_i \right)^2 - 8\phi \left(m_i - \phi v_i \right)}}{4\phi v_i} . \tag{18}$$

The constraint (10) is satisfied before the update if $m_i - \phi v_i \geq 0$. If $1 + 2\phi m_i \leq 0$, then $m_i \leq \phi v_i$ and from (18) we have that $\gamma_i > 0$. If, instead, $1 + 2\phi m_i \geq 0$, then, again by (18), we have

$$\begin{aligned} & \gamma_i > 0 \\ & \Leftrightarrow \sqrt{\left(1 + 2\phi m_i \right)^2 - 8\phi \left(m_i - \phi v_i \right)} > \left(1 + 2\phi m_i \right) \\ & \Leftrightarrow m_i < \phi v_i . \end{aligned}$$

From the KKT conditions, either $\alpha_i = 0$ or (10) is satisfied as an equality. In the later case, (16) holds, and thus $\alpha_i = \gamma_i > 0$, which concludes the derivation of the lemma.

5.2 Change of Variables

While linearization yielded a closed form convex solution to our optimization, it required approximating the constraint. We now proceed with the second alternative of obtaining a convex optimization problem by a change of variables, which allows us to achieve an exact convex update.

Since Σ is positive-semidefinite (PSD) it can be written as the square of another PSD matrix[5] Υ:

$$\Sigma = \Upsilon^2 \ , \ \Upsilon = \sqrt{\Sigma} \ .$$

Substituting in (9) gives the revised optimization problem

$$\begin{aligned}(\mu_{i+1}, \Upsilon_{i+1}) = \arg\min\ & \frac{1}{2}\log\left(\frac{\det \Upsilon_i^2}{\det \Upsilon^2}\right) + \frac{1}{2}\mathrm{Tr}\left(\Upsilon_i^{-2}\Upsilon^2\right) + \frac{1}{2}(\mu_i - \mu)^\top \Upsilon_i^{-2}(\mu_i - \mu) \\ \text{s.t.}\ & y_i(\mu \cdot x_i) \geq \phi \|\Upsilon x_i\| \\ & \Upsilon \text{ is PSD} \ . \end{aligned} \tag{19}$$

Note that, the objective is convex since $-\log\det\Upsilon^2 = -2\log\det\Upsilon$ which is well defined since Υ is PSD. The constraint is a second-order cone inequality and therefore convex.

We call this formulation `stdev`, since we have maintained the standard deviation in the constraint. This formulation was introduced by Crammer et al. (2008).

Standard optimization techniques can solve the convex program (19), but these methods can be slow. Instead, as before we derive a closed-form solution which we summarize in the following lemma:

Lemma 2 *The optimal solution of this form is,*

$$\begin{aligned}\mu_{i+1} &= \mu_i + \alpha y_i \Sigma_i x_i \\ \Sigma_{i+1} &= \Sigma_i - \beta \Sigma_i x_i x_i^\top \Sigma_i \ ,\end{aligned}$$

where

$$\beta = \frac{\alpha\phi}{\sqrt{v_i^+} + v_i \alpha\phi} \quad , \quad v_i^+ = x_i^\top \Sigma_{i+1} x_i \ .$$

and the value of the parameter α (a Lagrange multiplier) is given by

$$\alpha = \max\left\{0, \frac{1}{v_i}\frac{-m_i\phi' + \sqrt{m_i^2\frac{\phi^4}{4} + v_i\phi^2\phi''}}{\phi''}\right\} \ .$$

where $m_i = y_i(\mu_i \cdot x_i)$ (see (7)), $v_i = x_i^\top \Sigma_i x_i$ (see (8)), and for simplicity we define $\phi' = 1 + \phi^2/2$, $\phi'' = 1 + \phi^2$.

The resulting algorithm is shown in Figure 1.

5.2.1 DERIVATION OF LEMMA 2

The Lagrangian for (19) is

$$\mathcal{L} = \frac{1}{2}\log\left(\frac{\det \Upsilon_i^2}{\det \Upsilon^2}\right) + \frac{1}{2}\mathrm{Tr}\left(\Upsilon_i^{-2}\Upsilon^2\right) + \frac{1}{2}(\mu_i - \mu)^\top \Upsilon_i^{-2}(\mu_i - \mu) + \alpha\left(-y_i(\mu \cdot x_i) + \phi\|\Upsilon x_i\|\right) \ .$$

5. We use a decomposition in terms of PSD matrices because it yields a convex optimization problem. In general, a PSD matrix Σ can be written as $\Sigma = AA^\top$, which is not convex because it is rotation-invariant. Alternatively, any symmetric S matrix can be used $\Sigma = S^2$, but this is not convex either, since it is invariant to reflections.

At the optimum, it must be that

$$\frac{\partial}{\partial \mu}\mathcal{L} = \Upsilon_i^{-2}(\mu - \mu_i) - \alpha y_i x_i = 0 \ .$$

Therefore, if Υ_i is non-singular, the update for the mean is

$$\mu_{i+1} = \mu_i + \alpha y_i \Upsilon_i^2 x_i \ . \tag{20}$$

At the optimum, we must also have

$$\frac{\partial}{\partial \Upsilon}\mathcal{L} = -\Upsilon^{-1} + \frac{1}{2}\Upsilon_i^{-2}\Upsilon + \frac{1}{2}\Upsilon\Upsilon_i^{-2} + \alpha\phi\frac{x_i x_i^\top \Upsilon}{2\sqrt{x_i^\top \Upsilon^2 x_i}} + \alpha\phi\frac{\Upsilon x_i x_i^\top}{2\sqrt{x_i^\top \Upsilon^2 x_i}} = 0 \ . \tag{21}$$

Defining the matrix

$$C = \Upsilon_i^{-2} + \alpha\phi\frac{x_i x_i^\top}{\sqrt{x_i^\top \Upsilon^2 x_i}} \ , \tag{22}$$

we get

$$\frac{\partial}{\partial \Upsilon}\mathcal{L} = -\Upsilon^{-1} + \frac{1}{2}\Upsilon C + \frac{1}{2}C\Upsilon = 0$$

at the optimum. From this, it follows easily that at the optimum

$$\Upsilon = C^{-\frac{1}{2}} \ .$$

Substituting (22) into this equation, we obtain the update

$$\Upsilon_{i+1}^{-2} = \Upsilon_i^{-2} + \alpha\phi\frac{x_i x_i^\top}{\sqrt{x_i^\top \Upsilon_{i+1}^2 x_i}} \ .$$

Conveniently, the final form of the updates can be expressed in terms of the covariance matrix:[6]

$$\mu_{i+1} = \mu_i + \alpha y_i \Sigma_i x_i \tag{23}$$

$$\Sigma_{i+1}^{-1} = \Sigma_i^{-1} + \alpha\phi\frac{x_i x_i^\top}{\sqrt{x_i^\top \Sigma_{i+1} x_i}} \ . \tag{24}$$

As before we observe that if Σ_i^{-1} is PSD, so are Σ_{i+1}^{-1} and Σ_{i+1} with monotonically decreasing eigenvalues. Thus Σ_i is indeed non-singular, as assumed above.

6. Furthermore, writing the Lagrangian of (10) and solving it would yield the same solution as Equations (23,24). Thus the optimal solution of both (10) and (19) are the same.

It remains to determine the value of the Lagrange multiplier α. As before we compute the inverse of (24) using the Woodbury identity (Petersen and Pedersen, 2008) to get,

$$\begin{aligned}
\Sigma_{i+1} &= \left(\Sigma_i^{-1} + \alpha\phi \frac{x_i x_i^\top}{\sqrt{x_i^\top \Sigma_{i+1} x_i}}\right)^{-1} \\
&= \Sigma_i - \Sigma_i x_i \left(\frac{\sqrt{x_i^\top \Sigma_{i+1} x_i}}{\alpha\phi} + x_i^\top \Sigma_i x_i\right)^{-1} x_i^\top \Sigma_i \\
&= \Sigma_i - \Sigma_i x_i \left(\frac{\alpha\phi}{\sqrt{x_i^\top \Sigma_{i+1} x_i} + x_i^\top \Sigma_i x_i \alpha\phi}\right) x_i^\top \Sigma_i \\
&= \Sigma_i - \beta_i \Sigma_i x_i x_i^\top \Sigma_i \; . \qquad (25)
\end{aligned}$$

where we define

$$v_i^+ = x_i^\top \Sigma_{i+1} x_i \; , \qquad (26)$$

and

$$\beta_i = \frac{\alpha_i \phi}{\sqrt{v_i^+} + v_i \alpha_i \phi} \; . \qquad (27)$$

Multiplying (25) by $x_i^\top$ (left) and x_i (right) we get

$$v_i^+ = v_i - v_i \left(\frac{\alpha\phi}{\sqrt{v_i^+} + v_i \alpha\phi}\right) v_i \; ,$$

which is equivalent to

$$\begin{aligned}
v_i^+ \sqrt{v_i^+} + v_i^+ v_i \alpha\phi &= v_i \sqrt{v_i^+} + v_i^2 \alpha\phi - v_i^2 \alpha\phi \\
&= v_i \sqrt{v_i^+} \; .
\end{aligned}$$

Dividing both sides by $\sqrt{v_i^+}$, we obtain

$$v_i^+ + \sqrt{v_i^+} v_i \alpha\phi - v_i = 0 \; ,$$

which can be solved for v_i^+ to obtain

$$\sqrt{v_i^+} = \frac{-\alpha v_i \phi + \sqrt{\alpha^2 v_i^2 \phi^2 + 4 v_i}}{2} \; . \qquad (28)$$

The KKT conditions for the optimization imply that either $\alpha = 0$ and no update is needed, or the constraint (19) is an equality after the update.

Using the equality version of (19) and Equations (23,25,26,28) we obtain

$$m_i + \alpha v_i = \phi \frac{-\alpha v_i \phi + \sqrt{\alpha^2 v_i^2 \phi^2 + 4 v_i}}{2} \;, \tag{29}$$

which can be rearranged into the following quadratic equation in α:

$$\alpha^2 v_i^2 \left(1+\phi^2\right) + 2\alpha m_i v_i \left(1 + \frac{\phi^2}{2}\right) + \left(m_i^2 - v_i \phi^2\right) = 0 \;.$$

The smaller root of this equation is always negative and thus not a valid Lagrange multiplier. We use the following abbreviations for writing the larger root γ_i,

$$\phi' = 1 + \phi^2/2 \quad ; \quad \phi'' = 1 + \phi^2 \;.$$

The larger root is then

$$\gamma_i = \frac{-m_i v_i \phi' + \sqrt{m_i^2 v_i^2 \phi'^2 - v_i^2 \phi'' \left(m_i^2 - v_i \phi^2\right)}}{v_i^2 \phi''} \;. \tag{30}$$

The constraint (19) is satisfied before the update if $m_i - \phi\sqrt{v_i} \geq 0$. If $m_i \leq 0$, then $m_i \leq \phi\sqrt{v_i}$ and from (30) we have that $\gamma_i > 0$. If instead $m_i \geq 0$, then, again by (30), we have

$$\begin{aligned} &\gamma_i > 0 \\ &\Leftrightarrow m_i v_i \phi' < \sqrt{m_i^2 v_i^2 \phi'^2 - v_i^2 \phi' \left(m_i^2 - v_i \phi^2\right)} \\ &\Leftrightarrow m_i < \phi v_i \;. \end{aligned}$$

From the KKT conditions, either $\alpha_i = 0$ or (10) is satisfied as an equality, so (29) holds and $\alpha_i = \gamma_i > 0$.

The solution of (30) satisfies the KKT conditions, that is either $\alpha_i \geq 0$ or the constraint of (10) is satisfied before the update with the weights μ_i and Σ_i. We obtain the final form of α_i by simplifying (30) together with last comment and get,

$$\alpha_i = \max\left\{0, \frac{1}{v_i} \frac{-m_i \phi' + \sqrt{m_i^2 \frac{\phi^4}{4} + v_i \phi^2 \phi''}}{\phi''}\right\} \;. \tag{31}$$

6. Diagonal Covariance Matrices

So far we have said nothing about the covariance matrix Σ, which grows quadratically in the number of features. Since our intended applications are NLP tasks, computing the full matrix Σ is computationally infeasible. Additionally, even though we initialize the matrix to be diagonal (Figure 1), after applying the updates rule of either (14) (linearization/`var`) or (25) (change of variables/`stdev`), we may obtain a full covariance matrix, as we subtract from Σ_i a rank-one matrix proportional to the outer product of Σx. Therefore, successful applications to NLP problems require a restriction on the size of the matrix Σ.

In this section, we reduce the size of Σ by restriction to a diagonal-covariance matrix.[7] We discuss two main approaches, each of which can be applied to either the linearization or the change of variables formulations. In Section 6.1 we show two ways to use the full-covariance updates discussed above and add a diagonalization step. In Section 6.2 we take an alternative approach and re-develop the update step assuming an explicit diagonal representation of the covariance matrix.

6.1 Approximate Diagonal Update

Both updates above (linearization or change-of-variables) share the same form when updating the covariance matrix ((14) or (25))

$$\Sigma_{i+1} = \Sigma_i - \beta_i \Sigma_i x_i x_i^\top \Sigma_i \; . \tag{32}$$

Our diagonalization step will define the final matrix to be a diagonal matrix with its non-zero elements equals to the diagonal elements of (32). Formally we get,

$$\begin{aligned} \Sigma_{i+1} &= \operatorname{diag}\left(\Sigma_i - \beta_i \Sigma_i x_i x_i^\top \Sigma_i\right) \\ &= \operatorname{diag}\left(\Sigma_i\right) - \operatorname{diag}\left(\beta_i \Sigma_i x_i x_i^\top \Sigma_i\right) \\ &= \Sigma_i - \beta_i \operatorname{diag}\left(\Sigma_i x_i x_i^\top \Sigma_i\right) \; , \end{aligned}$$

where the last equality follows since we assume that Σ_i is diagonal and

$$\operatorname{diag}(A) = \begin{cases} A_{p,p'} & p = p' \\ 0 & p \neq p' \end{cases} \; .$$

A naïve implementation of the diagonal operator takes $\Theta(d^2)$ time and space. An efficient implementation first defines $z_i = \Sigma_i x_i$ and then sets,

$$\left(\Sigma_{i+1}\right)_{p,p} = \left(\Sigma_i\right)_{p,p} - \beta_i \left[\left(z_i\right)_p\right]^2 \text{ for } p = 1, \ldots, d \; .$$

We refer to this diagonalization scheme as L_2 since it is equivalent to a projection of the full matrix onto the set of diagonal matrices using the Euclidean norm.

We note in passing that since the diagonalization operator and the inverse operator are *not* commutative, we can first diagonalize the inverse of the covariance matrix and then invert the result. Concretely we start from the update of the inverse-covariance,

$$\Sigma_{i+1}^{-1} = \Sigma_i^{-1} + \eta_i x_i x_i^\top \; ,$$

where

$$\eta_i = 2\alpha_i \phi \; ,$$

for the linearization approach ((13)) and

$$\eta_i = \frac{\alpha_i \phi}{\sqrt{x_i^\top \Sigma_{i+1} x_i}} \; ,$$

7. There are other possible choices for reducing the matrix size, such as enforcing a sparse block diagonal matrix. We select diagonalization since it is the most straightforward reduction and yields a first order model. See a recent paper by Ma et al. (2010) for low rank options.

for the change of variables approach ((24)). We first diagonalize the inverse-covariance and get

$$\begin{aligned}\Sigma_{i+1}^{-1} &= \operatorname{diag}\left(\Sigma_i^{-1} + \eta_i x_i x_i^\top\right)\\ &= \operatorname{diag}\left(\Sigma_i^{-1}\right) + \operatorname{diag}\left(\eta_i x_i x_i^\top\right)\\ &= \Sigma_i^{-1} + \eta_i \operatorname{diag}\left(x_i x_i^\top\right) .\end{aligned}$$

As before we implement the update efficiently by writing

$$\left(\Sigma_{i+1}^{-1}\right)_{p,p} = \left(\Sigma_i^{-1}\right)_{p,p} + \eta_i \left[\left(x_i\right)_p\right]^2 \text{ for } p = 1, \ldots, d ,$$

or in terms of the covariance matrix

$$\left(\Sigma_{i+1}\right)_{p,p} = \frac{1}{\frac{1}{\left(\Sigma_i\right)_{p,p}} + \eta_i \left[\left(x_i\right)_p\right]^2} \text{ for } p = 1, \ldots, d .$$

We refer to this diagonalization scheme as KL since it is equivalent to a projection of the full matrix onto the set of diagonal matrices using the Kullback-Leibler (KL) divergence.

6.2 Exact Diagonal Update

An alternative to the approximate formulation is to explicitly maintain a diagonal and develop a corresponding update. We now assume that the matrix Σ is diagonal. We denote by $\Sigma_{i,(p)}$ the rth diagonal element of the matrix Σ_i, and by $x_{i,(p)}$ the rth element of x_i. We start with the first alternative above where we used linearization. We follow the derivation of Section 5.1 until (11). Proceeding with derivation of (12), but only for the diagonal elements indexed by p we get,

$$\frac{\partial}{\partial \Sigma_{(p)}} \mathcal{L} = -\frac{1}{2\Sigma_{(p)}} + \frac{1}{2\Sigma_{i,(p)}} + \phi\alpha x_{i,(p)}^2 = 0 \text{ for } p = 1, \ldots, d ,$$

Solving for $\Sigma_{(p)}$ we get

$$\Sigma_{i+1,(p)} = \frac{\Sigma_{i,(p)}}{1 + 2\alpha\Sigma_{i,(p)}\phi x_{i,(p)}^2} .$$

Following the logic presented after (15) we get that at the optimum we have

$$y_i \left(x_i \cdot \left(\mu_i + \alpha y_i \Sigma_i x_i\right)\right) = \phi \sum_p x_{i,(p)}^2 \frac{\Sigma_{i,(p)}}{1 + 2\alpha\Sigma_{i,(p)}\phi x_{i,(p)}^2} .$$

Substituting (7) and (8) and rearranging the terms we get the constraint

$$f(\alpha) = 0 ,$$

where we defined

$$f(\alpha) = m_i + \alpha v_i - \sum_r \frac{\Sigma_{i,(p)}\phi x_{i,(p)}^2}{1 + 2\alpha\Sigma_{i,(p)}\phi x_{i,(p)}^2} . \tag{33}$$

We will analyze (33) after developing its equivalent for the second alternative above where we perform a change of variables. As above we denote by $\Upsilon_{i,(p)}$ the rth diagonal element of the matrix Υ_i. We follow the derivation of Section 5.2 until (20). Proceeding with derivation of (21), but only for the diagonal elements indexed by r we get

$$\frac{\partial}{\partial \Upsilon_{(p)}} \mathcal{L} = -\Upsilon_{(p)}^{-1} + \Upsilon_{i,(p)}^{-2} \Upsilon_{(p)} + \alpha\phi \frac{x_{i,(p)}^2 \Upsilon_{(p)}}{\sqrt{x_i^\top \Upsilon^2 x_i}} = 0 \ .$$

Rearranging the terms we get

$$\frac{1}{\Upsilon_{(p)}^2} = \frac{1}{\Upsilon_{i,(p)}^2} + \alpha\phi \frac{x_{i,(p)}^2}{\sqrt{x_i^\top \Upsilon^2 x_i}} \ .$$

Thus,

$$\Upsilon_{(p)}^2 = \frac{\Upsilon_{i,(p)}^2 \sqrt{x_i^\top \Upsilon^2 x_i}}{\sqrt{x_i^\top \Upsilon^2 x_i} + \alpha\phi x_{i,(p)}^2 \Upsilon_{i,(p)}^2} \ .$$

Multiplying both sides by $x_{i,(p)}^2$ and summing over r we get

$$x_i^\top \Upsilon^2 x_i = \sum_r x_{i,(p)}^2 \Upsilon_{(p)}^2 = \sqrt{x_i^\top \Upsilon^2 x_i} \sum_r \frac{x_{i,(p)}^2 \Upsilon_{i,(p)}^2}{\sqrt{x_i^\top \Upsilon^2 x_i} + \alpha\phi x_{i,(p)}^2 \Upsilon_{i,(p)}^2} \ .$$

Finally, we obtain

$$\sqrt{x_i^\top \Upsilon^2 x_i} = \sum_r \frac{x_{i,(p)}^2 \Upsilon_{i,(p)}^2}{\sqrt{x_i^\top \Upsilon^2 x_i} + \alpha\phi x_{i,(p)}^2 \Upsilon_{i,(p)}^2} \ .$$

As before we employ the KKT conditions which state that when $\alpha > 0$ we have

$$m_i + \alpha v_i = \phi \sqrt{x_i^\top \Upsilon^2 x_i} \ .$$

Substituting in the last equality we get

$$\sqrt{x_i^\top \Upsilon^2 x_i} = \sum_r \frac{\phi x_{i,(p)}^2 \Upsilon_{i,(p)}^2}{m_i + \alpha v_i + \alpha\phi^2 x_{i,(p)}^2 \Upsilon_{i,(p)}^2} \ .$$

We use again the KKT conditions and get that the optimal value α_{i+1} is the solution of $g(\alpha) = 0$ for

$$g(\alpha) = m_i + \alpha v_i - \sum_r \frac{\phi^2 x_{i,(p)}^2 \Upsilon_{i,(p)}^2}{m_i + \alpha v_i + \alpha\phi^2 x_{i,(p)}^2 \Upsilon_{i,(p)}^2} \ . \tag{34}$$

The function $g(\alpha)$ defined in (34) and the function $f(\alpha)$ defined in (33) are both of the form

$$h(\alpha) = m_i + \alpha v_i - \sum_r \frac{a_r}{b + c_r \alpha} \ ,$$

where $v_i, a_r, c_r \geq 0$. The only difference is that $b = 1 > 0$ in (33) and $b = m_i$ in (34). Nevertheless, the optimal value of α satisfies $h(\alpha_i) = 0$. The following lemma summarizes few properties of both functions:

Lemma 3 *Assume that $v_i > 0$ and let $L_i = \max\{0, -m_i/v_i\}$. Both* (33) *and* (34) *have the following properties:*

1. *Their value at L_i is non-positive, that is $f(L_i) \leq 0$, $g(L_i) \leq 0$.*

2. *They are strictly-increasing for $\alpha \geq L_i$*

3. *For each function there exists a value U_i such that their value at U_i is positive, $f(U_i) > 0$, $g(U_i) > 0$*

Proof For the first property we consider two cases $m_i \geq 0$ and $m_i < 0$. We start with the first case and thus $L_i = 0$. Thus, $f(0) = m_i - \sum_r \Sigma_{i,(p)} \phi x_{i,(p)}^2 = m_i - \phi v_i < 0$, where the last ineqluaity follows since we assume that the constraint of (10) does not hold. Also, $g(0) = m_i - \frac{1}{m_i} \sum_r \phi^2 x_{i,(p)}^2 \Upsilon_{i,(p)}^2 = m_i - \phi^2 \frac{v_i}{m_i} < 0$ since we assumed that the constraint of (19) does not hold. When $m_i < 0$ we have $L_i = -m_i/v_i > 0$. In this case (33) becomes,

$$f(L_i) = -\sum_r \frac{\Sigma_{i,(p)} \phi x_{i,(p)}^2}{1 + 2L_i \Sigma_{i,(p)} \phi x_{i,(p)}^2} \leq 0 ,$$

since $\Sigma_{i,(p)} \phi x_{i,(p)}^2 \geq 0$. Similarly,

$$g(L_i) = -\sum_r \frac{\phi^2 x_{i,(p)}^2 \Upsilon_{i,(p)}^2}{L_i \phi^2 x_{i,(p)}^2 \Upsilon_{i,(p)}^2} = -\frac{d}{L_i} < 0 .$$

The second property of strictly-increasing follows immediately since $v_i > 0$ and since for both functions the denominator of each term in the sum over p is an increasing function in α which is non-negative in the range $\alpha \geq L_i$. Finally, the last property follows directly from the second property. ■

The lemma states that for each of f and g there is exactly one α_i (possibly different for each function) such that $f(\alpha_i) = 0$ and $g(\alpha_i) = 0$, but it does not provide an expression for computing α_i explicitly such as in Lemma 1. However, it further tells us that for each function the value of α_i is in the interval $[L_i, U_i]$. A value not far from α_i up to an accuracy of ε can be found using binary search in time proportional to $[(U_i - L_i) \log(1/\varepsilon)]$.

We conclude this section by computing a possible value U_i for each function and start with (33). Note that $a_i = \max\{0, -2m_i/v_i\}$ satisfies $m_i + (a_i/2)v_i \geq 0$. Thus, $b_i = \max_r \left\{ \left(2d\Sigma_{i,(p)} \phi x_{i,(p)}^2\right) / v_i \right\}$ satisfies $b_i v_i / (2d) - \Sigma_{i,(p)} \phi x_{i,(p)}^2 / \left(1 + 2\alpha \Sigma_{i,(p)} \phi x_{i,(p)}^2\right) \geq 0$ for $p = 1 \ldots d$. Therefore setting $U_i = \max\{a_i, b_i\}$ satisfies $f(U_i) \geq 0$ as desired. Finally, note that $U_i \geq L_i$ since $a_i \geq L_i$ by construction. For (34) we use the same definition of a_i but define $b_i = \max_r \left\{ \left(2d\Upsilon_{i,(p)}^2 \phi^2 x_{i,(p)}^2\right) / v_i \right\}$ and $U_i = \max\{a_i, b_i\}$. By a similar argument we have $g(U_i) > 0$ and $L_i \leq U_i$.

To summarize, as opposed to the full covariance case, in the exact diagonal case we do not compute the value of α_i explicitly, but use a binary-search algorithm to efficiently find a good approximation for the optional solution.

7. Evaluation

In this section we evaluate diagonalized versions of the CW algorithm on a range of binary classification problems for NLP tasks. We compare our methods against each other and against competitive online and batch learning algorithms.

We selected a range of 5 tasks and created 17 binary classification problems. We begin with a description of each task.

7.1 20 Newsgroups

The 20 Newsgroups corpus contains approximately 20,000 newsgroup messages, partitioned across 20 different newsgroups.[8] The data set is a popular choice for binary and multi-class text classification as well as unsupervised clustering. Following common practice, we created binary problems from the data set by creating binary decision problems of choosing between two similar groups. Our groups are:

- comp: `comp.sys.ibm.pc.hardware` vs. `comp.sys.mac.hardware`
- sci: `sci.electronics` vs. `sci.med`
- talk `talk.politics.guns` vs. `talk.politics.mideast`

Each message was represented as a binary bag-of-words. For each problem we selected 1800 examples balanced between the two labels.

7.2 Reuters

The Reuters Corpus Volume 1 (RCV1-v2/LYRL2004) contains over 800,000 manually categorized newswire stories (Lewis et al., 2004). Each article contains one or more labels describing its general topic, industry and region. We created the following binary decision tasks from the labeled documents:

- Insurance: Life (I82002) vs. Non-Life (I82003)
- Business Services: Banking (I81000) vs. Financial (I83000)
- Retail Distribution: Specialist Stores (I65400) vs. Mixed Retail (I65600).

These distinctions involve neighboring categories so they are fairly hard to make. Details on document preparation and feature extraction are given by Lewis et al. (2004). For each problem we selected 2000 examples using a bag-of-words representation with binary features. Each problem contains a balanced mixture of examples from each label.

7.3 Sentiment

We used a larger version of the sentiment multi-domain data set of Blitzer et al. (2007) used in Dredze et al. (2010).[9] This data consists of product reviews from 7 Amazon domains (apparel, book, dvd, electronics, kitchen, music, video). The goal in each domain is to classify a product

8. Corpus can be found at `http://people.csail.mit.edu/jrennie/20Newsgroups/`.
9. Data set can be found at `http://www.cs.jhu.edu/~mdredze/datasets/sentiment/`.

review as either positive or negative. Feature extraction creates unigram and bigram features using counts following Blitzer et al. (2007). For the apparel domain we used all 1940 examples and for all other domains we used 2000 examples. Each problem contains a balanced mixture of example labels.

7.4 Spam

We include a spam classification problem as a sample problem from the space of email classification tasks. We chose spam since it is a widely studied problem with several publicly available data sets. We selected the 2006 ECML/PKDD Discovery Challenge spam data set (Bickel, 2006) and use the provided representations (bag-of-words). The goal is to classify an email (bag-of-words) as either spam or ham (not-spam). This corpus contains two data sets: task A, which has three users, and task B, which has 15 users. We use the three users from task A since it has more training examples. For each user we select 2000 examples.

7.5 Pascal

The PASCAL large scale learning challenge workshop provided several large scale binary data sets.[10] We selected the NLP task, which is a Webspam filtering problem. Each example is the text from a web page. The task is to classify a webpage as either spam or ham. We used the default format provided by the workshop and selected 2000 examples.

7.6 USPS

The USPS data set contains examples of all 10 digits as part of a digit recognition task (OCR) (Hull, 1994). We created binary tasks by pairing each digit with another in order: 0/9, 1/2, 3/4, 5/6, 7/8. We used the standard value of each pixel in the image, as well as the product of all the pixel pairs in the image (bi-grams.)

Each data set was randomly divided for 10-fold cross validation experiments. Classifier parameters (ϕ) and the number of training iterations (up to 10) were tuned for each classification task on a single randomized run over the data. Results are reported for each problem as the average accuracy over the 10 folds. Statistical significance is computed using McNemar's test.

7.7 Results

We start by comparing the performance of the diagonalized CW algorithms: `var` (linearization) against `stdev` (change of variables), approximate against exact diagonalization, and for approximate updates, KL against L_2. All six algorithm combinations were run on the data sets described above. The average test error on all data sets is shown in Table 2. For each method, we summarize its overall performance by computing its mean rank among all the other algorithm: if an algorithm has a mean rank of 1 then on average across all data sets it achieved the lowest error on average, whereas a rank of 6 indicates that it ranked 6th in error on average across all tests.

Starting with the KL methods for `var` and `stdev`, the `stdev` method does slightly better, a result shown in Crammer et al. (2008). Comparing the two methods for diagonalization (KL vs. L_2), while L_2 does slightly better for `var` (the best overall), the KL method appears to be more stable overall,

10. Data sets can be found at `http://largescale.first.fraunhofer.de/workshop/`.

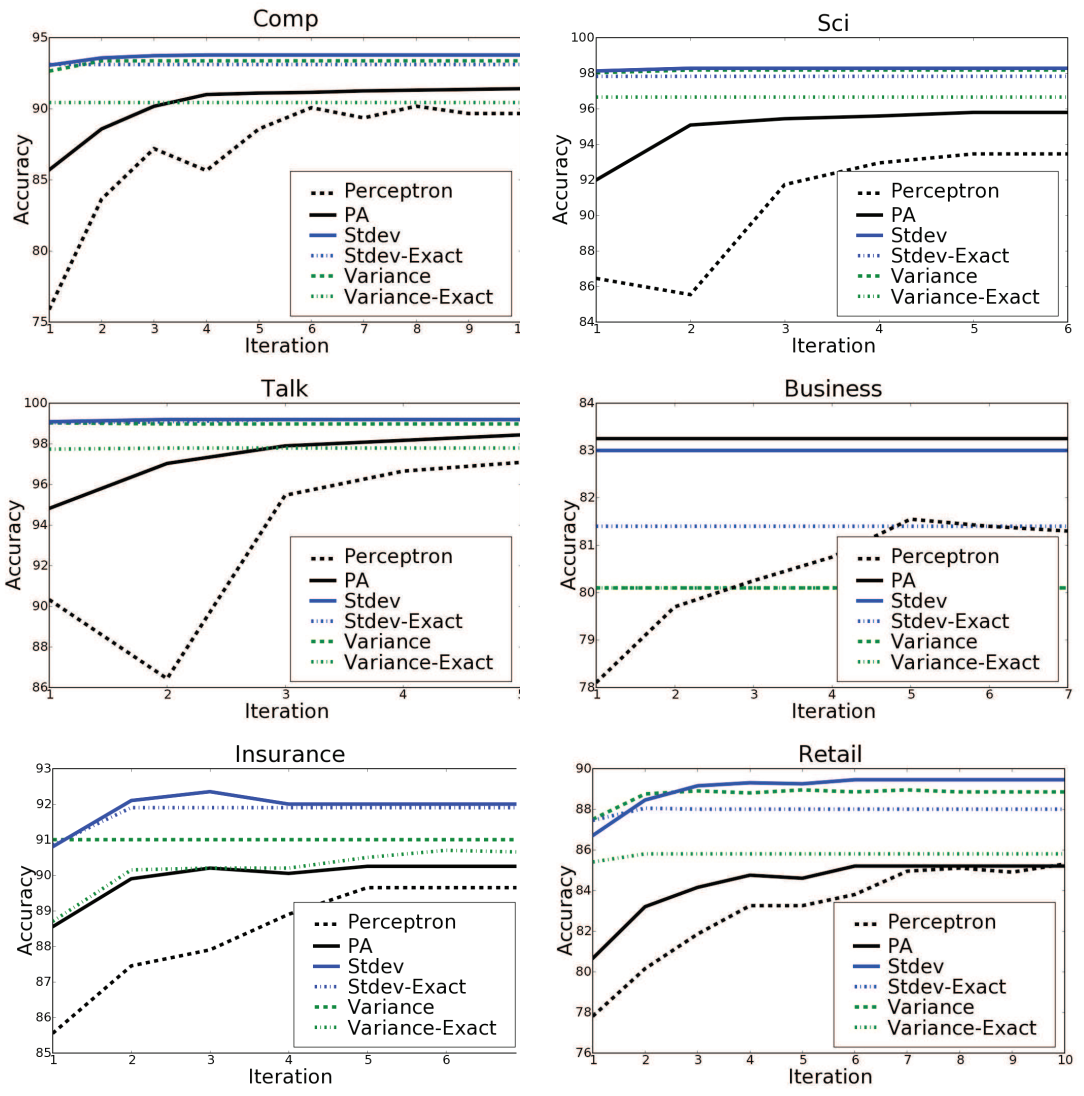

Figure 3: Accuracy on test data after each iteration on six data sets.

achieving the best or closest to best results for the `var` and `stdev` methods. In comparison, the exact methods do worse than the approximations. To understand these results, we examine some learning curves from several of the online experiments in Figure 7.6 and in Figure 7.6, which show accuracy on test data after each iteration. In many of these plots, the exact method does very well after the first iteration, surpassing the performance of the approximate methods. However, after the first iteration the exact method stops improving while the approximate methods continue to improve. By finding a solution that exactly achieves the constraint, exact produces a more aggressive algorithm that learns faster but overfits (Figure 5). In contrast, the approximate solutions do not fully enforce the constraint on each update but this slower learning reduces overfitting and improves generalization error over several iterations.

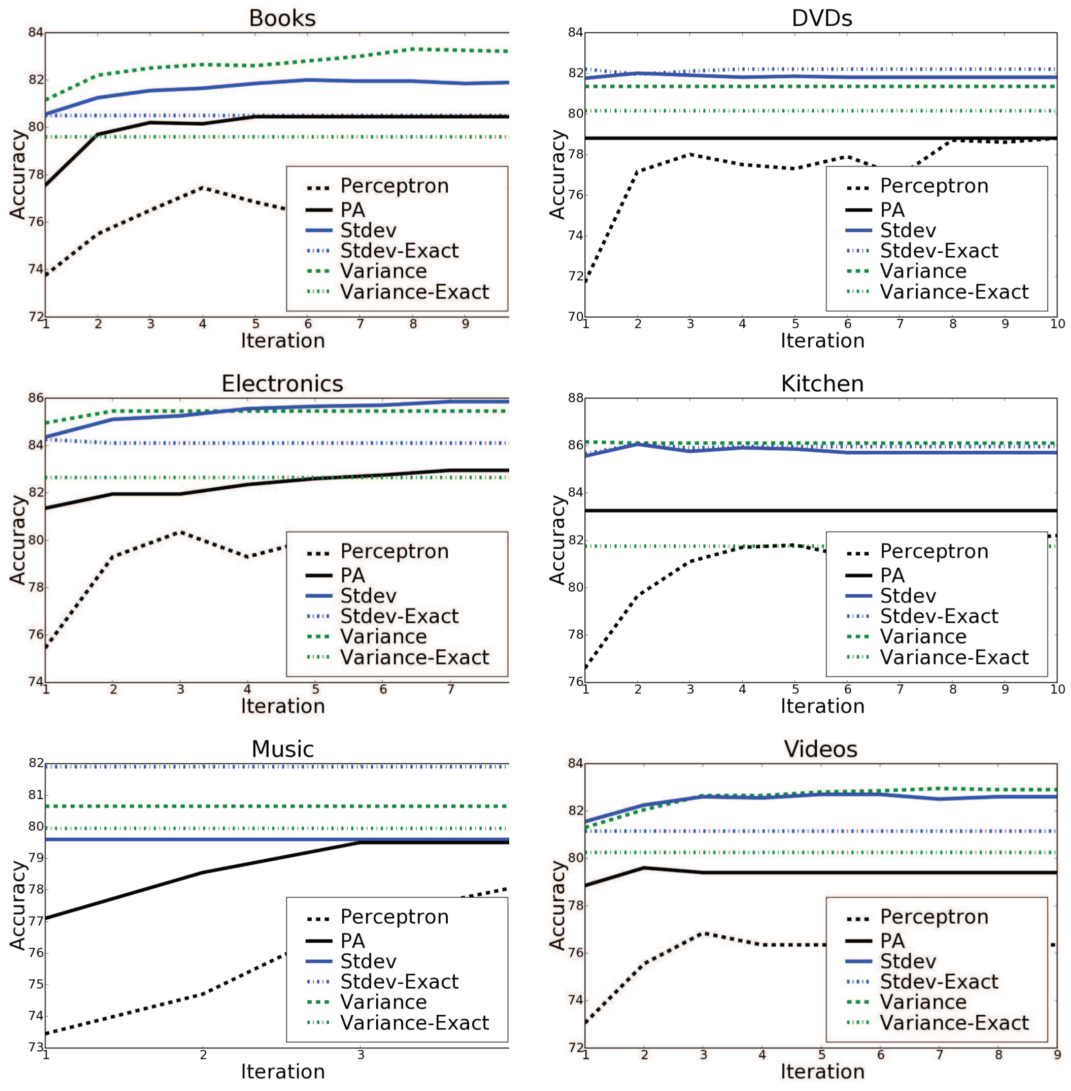

Figure 4: Accuracy on test data after each iteration on the six Amazon data sets.

We next compare the results from our approximation diagonalization CW methods to other popular online learning algorithms (Table 3). We evaluated the perceptron (Rosenblatt, 1958), passive-aggressive (Crammer et al., 2006a), stochastic gradient descent (Zhang, 2004; Blitzer et al., 2007) and a diagonalized second order perceptron (Cesa-Bianchi et al., 2005), all of which perform well for NLP problems. In every experiment, a CW method improved over all of the online learning baselines.

As discussed above, online algorithms are attractive even for batch learning because of their simplicity and ability to operate on extremely large data sets. In the batch setting, these algorithms are run several times over the training data, which yields slower performance than single pass learning (Carvalho and Cohen, 2006). While we have shown that CW improves on accuracy, it also learns faster than other baselines, requiring fewer iterations over the training data. Such behavior can be

		`var`			`stdev`		
	Task	*KL*	L_2	*Exact*	*KL*	L_2	*Exact*
Sentiment	Apparel	12.53	**12.47**	14.79	13.66	13.51	14.28
	Books	16.90	17.30	19.60	16.25	22.35	**15.25**
	DVD	17.45	17.05	19.05	17.60	18.30	**16.95**
	Electronics	14.95	15.40	16.65	**14.75**	20.95	15.50
	Kitchen	13.75	**13.65**	15.30	15.40	15.40	14.25
	Music	**17.15**	17.55	19.90	17.75	17.85	19.35
	Video	21.75	**18.55**	25.85	22.50	19.00	23.60
ECML	Spam A	2.65	1.45	3.10	**0.75**	4.15	0.80
	Spam B	1.35	1.20	2.65	**1.00**	1.10	1.05
	Spam C	1.50	1.40	3.40	1.50	3.55	**1.35**
Reuters	Retail	10.55	18.80	18.75	**10.25**	11.05	11.05
	Business	16.35	**15.35**	17.10	16.45	16.80	17.20
	Insurance	**8.20**	9.15	10.20	8.55	9.55	10.10
20 News	Comp	6.69	**5.61**	8.59	6.79	16.64	6.90
	Sci	**2.44**	2.74	3.20	3.04	13.35	3.10
	Talk	0.86	0.43	2.43	**0.27**	8.38	1.14
Pascal	Webspam	3.55	3.10	3.85	**2.95**	3.10	5.35
	Mean Rank	2.53	*2.35*	5.29	*2.47*	4.53	3.59

Table 2: Average Error of all variants of confidence-weighted algorithms presented in this paper over 17 binary text classification tasks. The best score for each data set is set in bold. The mean rank is the average rank of each algorithm across data sets, ranging from 1 (best) to 6.

seen in Figure 7.6 and Figure 7.6, which shows test error after each training iteration for CW and PA. While CW clearly improves over PA, it converges very quickly, reaching near best performance on the first iteration. In contrast, PA benefits from multiple iterations over the data; its performance changes significantly from the first to fifth iteration. The plot also illustrates exact's behavior, which initially beats PA but does not improve. In fact, on eleven of the twelve data sets, `var`-Exact beats PA on the first iteration.

7.8 Batch Learning

While online algorithms are widely used, batch algorithms are still preferred for many tasks. Batch algorithms can make global learning decisions by examining the entire data set, an ability beyond online algorithms. In general, when batch algorithms can be applied they perform better. We compare CW to three standard batch algorithms: naïve Bayes (default configuration in MALLET McCallum, 2002), maximum entropy classification (default configuration in MALLET McCallum, 2002) and support vector machines (LibSVM Chang and Lin, 2001). Classifier parameters (Gaussian prior for maxent and C for SVM) were tuned as for the online methods.

Results for batch learning are shown in table Table 4. As expected, the batch methods tend to do better than the online methods (perceptron, PA, and SGD). However, in 13 out of 17 tasks the CW

		`var`		`stdev`					
	Task	*KL*	L_2	*KL*	L_2	*Per*	*PA*	*SOP*	*SGD*
Sentiment	Apparel	12.53	**12.47**	13.66	13.51	17.84	13.35	17.42	13.76
	Books	16.90	17.30	$*$**16.25**	22.35	23.10	18.45	19.75	18.60
	DVD	17.45	$*$**17.05**	$*$17.60	18.30	20.45	20.95	20.55	18.70
	Electronics	14.95	15.40	$\star$**14.75**	$\star$20.95	18.65	17.45	20.20	16.00
	Kitchen	$\star$13.75	$*$**13.65**	15.40	15.40	16.65	15.20	21.20	16.00
	Music	**17.15**	17.55	17.75	17.85	22.35	19.40	21.80	18.20
	Video	21.75	**18.55**	22.50	19.00	21.50	18.90	21.90	19.25
ECML	Spam A	2.65	1.45	$\star$**0.75**	4.15	3.20	1.40	4.20	2.30
	Spam B	1.35	$\star$1.20	$*$**1.00**	1.10	3.00	2.40	1.95	2.80
	Spam C	1.50	$\star$**1.40**	$\star$1.50	3.55	3.65	2.10	2.90	2.15
Reuters	Retail	†10.55	18.80	†**10.25**	†11.05	19.60	17.50	19.45	14.30
	Business	16.35	**15.35**	16.45	16.80	19.00	16.15	21.80	15.45
	Insurance	**8.20**	9.15	8.55	9.55	11.35	12.35	10.15	9.35
20 News	Comp	6.69	†**5.61**	6.79	†16.64	10.30	8.65	10.45	7.88
	Sci	†**2.44**	†2.74	$*$3.04	†13.35	6.70	8.06	4.67	4.06
	Talk	0.86	$*$0.43	†**0.27**	†8.38	3.24	1.57	2.59	1.19
Pascal	Webspam	3.55	3.10	$\star$**2.95**	3.10	7.60	3.90	5.05	3.50
USPS	0 vs 9	0.56	0.56	0.56	0.56	**0.37**	0.93	0.75	0.56
	1 vs 2	1.73	0.87	0.87	4.33	1.73	**0.65**	2.38	42.86
	3 vs 4	1.37	1.09	1.09	1.09	**0.82**	1.09	2.73	45.36
	5 vs 6	**0.91**	0.91	1.52	1.52	4.24	0.91	3.03	48.48
	7 vs 8	2.24	2.24	1.92	2.24	2.56	**1.60**	4.15	53.04

Table 3: Average Error of approximate-diagonal confidence-weighted algorithms and four other online algorithms: The perceptron algorithm (Per), the passive-aggressive (PA) algorithm, the second order perceptron (SOP) and stochastic gradient decent evaluated using 17 binary text classification tasks. The best score for each data set is set in bold. Statistical significance measured by McNemar's test indicates when a CW algorithm is statistically significant ($\star$ $p = 0.05$, $*$ $p = 0.01$, † $p = 0.001$) from each of the four baselines (perceptron, PA, SOP, SGD).

algorithm beats all of the batch methods. The much faster and simpler online algorithm performs better than the slower more complex batch methods.

The speed advantage of online methods in the batch setting can be seen in Table 5, which shows the average training time in seconds for a single experiment (fold) for a representative selection of CW algorithms and some of the baselines. The online times include the multiple iterations selected for each online learning experiment. The differences between the online and batch algorithms are striking. While CW performs better than the batch methods, it is also much faster, while being equivalent in speed to the other online methods. For webspam data, which contains many features, an SVM takes over 1.5 minutes to train while the CW algorithms take between 1-2 seconds.

We also evaluated the effects of commonly used techniques for online and batch learning, including averaging and TFIDF features; they did not improve results so details are omitted. Although

		`var`		`stdev`				
	Task	*KL*	L_2	*KL*	L_2	*NB*	*Maxent*	*SVM*
Sentiment	Apparel	12.53	**12.47**	13.66	13.51	12.63	13.56	13.92
	Books	16.90	17.30	$\star$**16.25**	†22.35	18.40	18.05	18.25
	DVD	17.45	**17.05**	17.60	18.30	21.00	18.00	19.60
	Electronics	14.95	15.40	$\star$**14.75**	†20.95	17.05	15.85	16.25
	Kitchen	$\star$13.75	**13.65**	15.40	15.40	15.00	15.25	15.50
	Music	**17.15**	17.55	17.75	17.85	18.65	17.90	18.25
	Video	21.75	18.55	22.50	19.00	22.95	**18.40**	18.80
ECML	Spam A	$\star$2.65	1.45	$\star$**0.75**	4.15	3.70	1.30	1.75
	Spam B	1.35	1.20	$\star$**1.00**	1.10	4.20	1.55	1.90
	Spam C	1.50	1.40	1.50	†3.55	1.40	**1.35**	1.40
Reuters	Retail	$*$10.55	$\star$18.80	†**10.25**	$*$11.05	16.55	12.55	12.90
	Business	16.35	**15.35**	16.45	16.80	20.00	15.85	15.60
	Insurance	**8.20**	9.15	8.55	9.55	11.80	9.10	9.75
20 News	Comp	6.69	5.61	6.79	†16.64	**5.56**	7.82	7.67
	Sci	$\star$2.44	$\star$2.74	3.04	†13.35	**1.42**	3.40	3.86
	Talk	0.86	$\star$0.43	+**0.27**	†8.38	0.97	1.03	1.24
Pascal	Webspam	3.55	$\star$3.10	+**2.95**	$\star$3.10	19.10	6.05	3.85
USPS	0 vs 9	**0.56**	0.56	0.56	0.56	1.12	33.02	0.56
	1 vs 2	1.73	0.87	0.87	4.33	1.52	42.86	**0.65**
	3 vs 4	1.37	1.09	1.09	1.09	1.91	45.36	**0.55**
	5 vs 6	0.91	0.91	1.52	1.52	3.03	48.48	**0.61**
	7 vs 8	2.24	2.24	1.92	2.24	2.56	53.04	**0.96**

Table 4: Average Error of approximate-diagonal confidence-weighted algorithms and three batch algorithms: Naïve Bayes (NB), Maximum entropy classifier (Maxent) and support vector machine (SVM) evaluated using 17 binary text classification tasks. The best score for each data set is set in bold. Statistical significance measured by McNemar's test indicates when a CW algorithm is statistically significant ($\star\ p = 0.05$, $*\ p = 0.01$, $\dagger\ p = 0.001$) from each of the three baselines (NB, Maxent, SVM).

the above data sets are balanced with respect to labels, we also evaluated the methods on variant data sets with unbalanced label distributions, and still saw similar benefits from the CW methods.

7.9 Large Data Sets

Online algorithms are especially attractive in tasks where training data exceeds available main memory or in streaming settings where training examples cannot be saved. In both of these settings, a single sequential pass over the data is highly preferred to multiple passes common in batch training cases. So far, we have shown that CW algorithms are more aggressive than other online algorithms, an advantage when the algorithm is limited to a single pass. The results is both higher performance and fewer training iterations. The question we now answer is whether this advantage is maintained in large data settings.

Task	*variance KL*	*variance Exact*	*Perceptron*	*PA*	*Maxent*	*SVM*
Apparel	0.2	0.2	0.1	0.04	1	11
Books	0.1	0.8	0.1	0.1	6	25
DVD	0.1	0.9	0.1	0.04	6	22
Electronics	0.1	0.4	0.1	0.03	2	17
Kitchen	0.1	0.6	0.1	0.1	2	14
Music	0.1	0.5	0.1	0.1	4	19
Video	0.3	0.7	0.1	0.2	6	24
Spam A	0.03	0.2	0.1	0.8	3	3
Spam B	0.04	0.2	0.1	0.1	3	3
Spam C	0.1	0.2	0.04	0.04	1	2
Retail	0.1	0.1	0.03	0.03	0.6	4
Business	0.1	0.3	0.1	0.02	0.3	5
Insurance	0.1	0.2	0.1	0.02	0.8	4
Comp	0.04	0.2	0.1	0.2	2	11
Sci	0.1	0.3	0.1	0.04	2	7
Talk	0.1	0.3	0.1	0.1	4	7
Webspam	1	2	3	1	12	103

Table 5: Training times in seconds for a single training run (averaged over 10 trials.)

We selected two large data sets for evaluation. The combined product reviews for all the domains by Blitzer et al. (2007) yield one million sentiment examples. While most reviews were from the book domain, the reviews are taken from a wide range of Amazon product types and are mostly positive. From the Reuters corpus, we created a one vs. all classification task for the *Corporate* topic label, yielding 804,411 examples of which 381,325 are labeled corporate. For the two data sets, we created four random splits each with 10,000 test examples and the remaining examples saved for training. Parameters were optimized by training on 5,000 randomly chosen examples. We evaluated the CW `var`-KL algorithm and the passive-aggressive algorithm using a single pass over this data.

The results are shown as horizontal lines in Figure 6. For the Sentiment data, CW maintains over a 1% lead when compared to PA. On the Reuters data, the results are reversed with PA having the advantage. The difference between these behaviors may be related to the different feature representations used by each data set. The Reuters data contains $288,062$ unique features, for a feature to document ratio of 0.36. In contrast, the sentiment data contains $13,460,254$ unique features, a feature to document ratio of 13.33. This means that Reuters features will occur several times during training while many sentiment features only once. This may give CW an advantage on Sentiment. It is also possible that CW over-fits the Reuters data, something that will be observed in the next set of experiments below.

7.10 Distributed Training

While faster learning over a data stream is important, not all large data sets can be processed by a single processor. Therefore, we looked at the case where many processors are available, each with easy access to a fraction of the training data, but where communication between processors

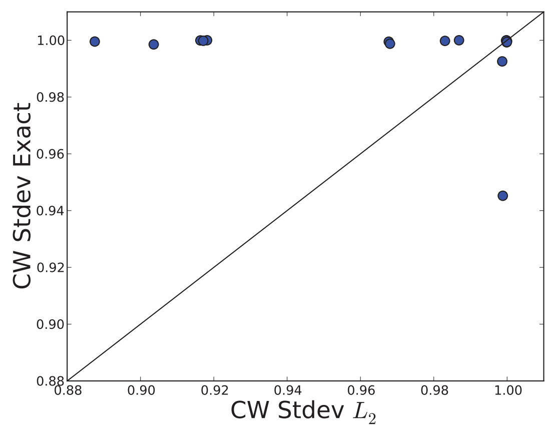

Figure 5: The trained model's accuracy on the training data for fifteen of the data sets for the exact diagonal and L_2 diagonal approximation Stdev methods. Points above the line indicate that the exact algorithm obtained a higher training accuracy than the L_2 diagonal method. Observe that the exact method almost always obtains a higher training accuracy, and is nearly 100% in every case. Coupled with the results on test data, which are worse for the exact methods, these results indicate that the exact method overfits the training data. These results are typical when comparing the exact algorithms against the diagonal approximations.

is limited. In this setting, we would like an algorithm where individual processors train models on their easily accessible data, and then they combine their models. While this often does not perform as well as a single model trained on all of the data, it is a cost-effective way of learning from very large training sets.

One simple approach is to combine many trained models by averaging their weights (McDonald et al., 2010). However, averaging models trained in parallel assumes that each model has an equally accurate estimate of the model weights. This is obviously not the case where different processors saw different portions of the data, made different updates, or saw features that other processors did not. Rather than taking an average over all models, CW provides a confidence value for each weight, allowing for a more intelligent combination of weights from multiple models.

Since each model is a Gaussian distribution over weights, combining multiple trained CW classifiers is equivalent to combining multiple Gaussian distributions. Specifically, we compute the combined model by finding the Gaussian that minimizes the total divergence to the set C of Gaussian distributions (individually trained classifiers) for some divergence operator D:

$$\min_{\mu,\Sigma} \sum_{c \in C} D((\mu,\Sigma)||(\mu_c,\Sigma_c)),$$

If D is the Euclidean distance, then this is just the average of the individual models. However, we can instead rely on the variance estimates of each Gaussian by choosing the KL divergence for D.

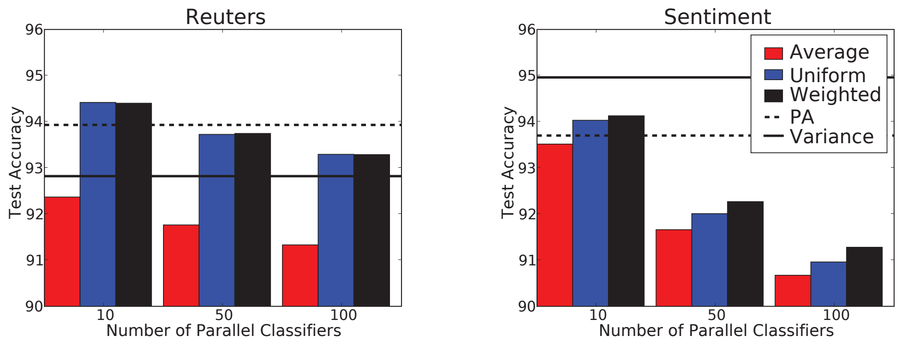

Figure 6: Results for Reuters (800k) and Sentiment (1000k) averaged over 4 runs. Horizontal lines show the test accuracy of a model trained on the entire training set. Vertical bars show the performance of n (10, 50, 100) classifiers trained on disjoint sections of the data as the average performance, uniform combination, or weighted combination.

This minimization leads to the following weighted combination of individual model means:

$$\mu = \left(\sum_{c\in C} \Sigma_c^{-1}\right)^{-1} \sum_{c\in C} \Sigma_c^{-1}\mu_c \qquad \Sigma^{-1} = \sum_{c\in C} \Sigma_c^{-1}.$$

We evaluate classifier combination by training n (10, 50, 100) models by dividing the example stream into n disjoint parts and report the average performance of each of the n classifiers (average), the combined classifier from taking the average of the n sets of weights (L_2) and the combination using the KL divergence on the test data across 4 randomized runs.

Average accuracy on the test sets are reported in Figure 6. As stated above, the PA single model achieves higher accuracy for Reuters, possibly because of the low feature to document ratio. However, combining 10 CW classifiers achieves the best performance. For sentiment, combining 10 classifiers beats PA but is not as good as a single CW model. In every case, combining the classifiers improves over each model individually. On sentiment, the KL combination improves over the L_2 combination and in Reuters the models are equivalent. For comparison, we show the accuracy on the test data for a single run on the CW Variance KL model on sentiment data Figure 7. When trained on all of the data and distributed across 10 machines, the classifier loses 1% of its performance which, using Figure 7 as a guide, corresponds to using 22% of the training data.

Finally, we computed the actual run time of both PA and CW on the large data sets to compare the speed of each model. While CW is more complex, requiring more computation per example, the actual speed is comparable to PA; in all tests the run time of the two algorithms was indistinguishable.

8. Related Work

The idea of using weight-specific variable learning rates has a long history in neural-network learning (Sutton, 1992), although we do not know of a previous model that specifically models confidence in a way that takes into account the frequency of features.

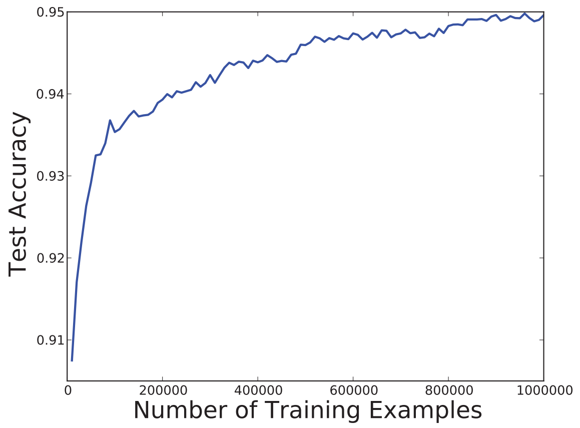

Figure 7: Results from CW Variance KL run on the large scale Sentiment data (1000k) averaged over 4 runs. Accuracy on test data is measured every 10k training examples to demonstrate the improvement with increases in training data.

Online additive algorithms have a long history, from the perceptron (Rosenblatt, 1958) to more recent methods (Kivinen and Warmuth, 1997; Crammer et al., 2006b). Our update has a more general form, in which the input vector x_i is linearly transformed using the covariance matrix, both rotating the input and assigning weight specific learning rates.

The second order perceptron (SOP) (Cesa-Bianchi et al., 2005) demonstrated that second-order techniques can improve first-order online methods. Both SOP and CW maintain second-order information. SOP is mistake driven while CW is passive-aggressive. SOP uses the current example in the correlation matrix for prediction while CW updates after prediction. A variant of `stdev` similar to SOP follows from our derivation if we fix the Lagrange multiplier in (20) to a predefined value $\alpha_i = \alpha$, omit the square root, and use a gradient-descent optimization step. Fundamentally, CW algorithms have a probabilistic motivation, while the SOP is geometric: replace the ball around an example with a refined ellipsoid. Shivaswamy and Jebara (2007) used a similar motivation in batch learning.

Ensemble learning shares the idea of combining multiple classifiers. Gaussian process classification (GPC) maintains a Gaussian distribution over weight vectors (primal) or over regressor values (dual). Our algorithm uses a different update criterion than the standard GPC Bayesian updates (Rasmussen and Williams, 2006, Chapter 3), avoiding the challenge of approximating posteriors. Bayes point machines (Herbrich et al., 2001) maintain a collection of weight vectors consistent with the training data, and use the single linear classifier which best represents the collection. Conceptually, the collection is a non-parametric distribution over the weight vectors. Its online version (Harrington et al., 2003) maintains a set of weight vectors that are updated simultaneously. The relevance vector machine (Tipping, 2001) incorporates probability into the dual formulation of SVMs. As in our work, the dual parameters are random variables distributed according to a diagonal Gaussian with example specific variance. The weighted-majority (Littlestone and Warmuth, 1994) algorithm and later improvements (Cesa-Bianchi et al., 1997) combine the output of multiple arbitrary classifiers, maintaining a multinomial distribution over the experts. We assume linear classifiers as experts and maintain a Gaussian distribution over their weight vectors.

With the growth of available data there is an increasing need for algorithms that process training data very efficiently. A similar approach to ours is to train classifiers incrementally (Bordes and Bottou, 2005). The extreme case is to use each example once, without repetitions, as in the multiplicative update method of Carvalho and Cohen (2006).

In Bayesian modeling, we note few approaches that use parameterized distributions over weight vectors. Borrowing concepts from support vector machines, Jaakkola et al. (1999) developed maximum entropy discrimination, which models the generation of examples with one generative model for each class. The model consisted of distributions over the weights and over margin thresholds. They used Bayesian prediction and set the weights using the maximum-entropy principle. In a more recent approach, Minka et al. (2009) proposed using additional virtual vectors to allow more expressive power beyond Gaussian prior and posterior.

Passing the output of a linear model through a logistic function has a long-history in the statistical literature, and is extensively covered in many textbooks (e.g., Hastie et al., 2001). Platt (1998) used similar ideas to convert the output of a support vector machine into probabilistic quantities.

Since the conference versions of this work were published, a few algorithms reminiscent of CW were proposed. Duchi et al. (2010) and McMahan and Streeter (2010) proposed to replace the standard Euclidean distance in stochastic gradient decent with general Mahalanobis distance defined by the second order information, captured by the instantaneous second order moment. Crammer et al. (2009a) proposed to replace the hard constraint enforced by the CW algorithm with a relaxed version, formulated using an additional term in the objective function. They call their algorithm AROW for adaptive regularization of weight vectors. Orabona and Crammer (2010) proposed later a framework for online learning, which contains an algorithm close to AROW as a special case, as well as other new algorithms. From a different perspective, Crammer and Lee (2010) proposed a microscopic view for learning, that tracks individual weight-vectors as opposed only to their macroscopic quantities, such as mean and covariance. Their algorithm has similar update form as CW ((11) and (13)), yet with different rates.

Finally, Shivaswamy and Jebara (2010b,a) proposed to use second order information, or the variance in the batch setting where an iid distribution over the examples is assumed. Their algorithm both maximizes the (average) margin and at the same time minimizes its variance. Note, that they do not maintain a distribution over weight vectors, and the probability space is induced using the distribution over training examples.

9. Conclusion

We have presented confidence-weighted linear classifiers, a new learning method designed for NLP problems based on the notion of weight confidence. The algorithm maintains a distribution over weight vectors; online updates both improve the weight estimates and reduce the distribution's variance. Our method improves over both online and batch methods and learns faster on over a dozen NLP data sets. Additionally, our new algorithms allow more intelligent classifier combination techniques, yielding improved performance in distributed learning.

Acknowledgments

Most of the work was performed while the authors were in the Department of Computer and Information Science at the University of Pennsylvania. This research was supported in part by the Israeli Science Foundation grant ISF-1567/10. Koby Crammer is a Horev Fellow, supported by the Taub Foundations.

References

Galen Andrew and Jianfeng Gao. Scalable training of l1-regularized log-linear models. In *ICML '07: Proceedings Of The 24th International Conference On Machine Learning*, pages 33–40, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: http://doi.acm.org/10.1145/1273496.1273501.

Steffen Bickel. ECML-PKDD discovery challenge overview. In *The ECML-PKDD Discovery Challenge Workshop*, 2006.

John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, Bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Association For Computational Linguistics (ACL)*, 2007.

Antoine Bordes and Léon Bottou. The Huller: a simple and efficient online SVM. In *European Conference On Machine Learning(ECML), LNAI 3720*, 2005.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Xavier Carreras, Michael Collins, and Terry Koo. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Conference On Natural Language Learning (CONLL)*, 2008.

Vitor R. Carvalho and William W. Cohen. Single-pass online learning: Performance, voting schemes and online feature selection. In *KDD-2006*, 2006.

Nicoló Cesa-Bianchi and Gabor Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

Nicoló Cesa-Bianchi, Yoav Freund, David Haussler, David P. Helmbold, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. *Journal of the ACM*, 44(3):427–485, May 1997.

Nicoló Cesa-Bianchi, Alex Conconi, and Claudio Gentile. A second-order perceptron algorithm. *Siam Journal Of Commutation*, 34(3):640–668, 2005.

Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

David Chiang, Yuval Marton, and Philip Resnik. Online large-margin training of syntactic and structural translation features. In *Empirical Methods In Natural Language Processing (EMNLP)*, 2008.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Empirical Methods In Natural Language Processing (EMNLP)*, 2002.

Koby Crammer and Daniel D. Lee. Learning via Gaussian herding. In *Advances In Neural Information Processing Systems 24*, 2010.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal Of Machine Learning Research*, 7:551–585, 2006a.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal Of Machine Learning Research*, 7:551–585, 2006b.

Koby Crammer, Mark Dredze, and Fernando Pereira. Exact convex confidence-weighted learning. In *Neural Information Processing Systems (NIPS)*, 2008.

Koby Crammer, Alex Kulesza, and Mark Dredze. Adaptive regularization of weight vectors. In *Advances In Neural Information Processing Systems 23*, pages 414–422, 2009a.

Koby Crammer, Mehryar Mohri, and Fernando Pereira. Gaussian margin machines. In *Proceedings Of The Twelfth Intentional Conference On Artificial Intelligence And Statistics (AISTATS)*, 2009b.

Mark Dredze and Koby Crammer. Active learning with confidence. In *Association For Computational Linguistics (ACL)*, 2008a.

Mark Dredze and Koby Crammer. Online methods for multi-domain learning and adaptation. In *Empirical Methods In Natural Language Processing (EMNLP)*, 2008b.

Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *International Conference On Machine Learning (ICML)*, 2008.

Mark Dredze, Alex Kulesza, and Koby Crammer. Multi-domain learning by confidence-weighted parameter combination. *Machine Learning*, 79(1-2):123–149, 2010.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *Proceedings Of The Twenty Third Annual Conference On Learning Theory*, 2010.

Jianfeng Gao, Galen Andrew, Mark Johnson, and Kristina Toutanova. A comparative study of parameter estimation methods for statistical natural language processing. In *Proceedings Of The 45th Annual Meeting Of The Association Of Computational Linguistics*, pages 824–831, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P07-1104`.

Amir Globerson, Terry Y. Koo, Xavier Carreras, and Michael Collins. Exponentiated gradient algorithms for log-linear structured prediction. In *Proceedings Of The 24th International Conference On Machine Learning*, pages 305–312. ACM New York, NY, USA, 2007.

Edward Harrington, Ralf Herbrich, Jyrki Kivinen, John Platt, and Robert C. Williamson. Online Bayes point machines. In *7th Pacific-Asia Conference On Knowledge Discovery And Data Mining (PAKDD)*, 2003.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

Ralf Herbrich, Thore Graepel, and Colin Campbell. Bayes point machines. *Journal Of Machine Learning Research*, 1:245–279, 2001.

Jonathan J. Hull. A database for handwritten text recognition research. *Pattern Analysis And Machine Intelligence, IEEE Transactions On*, 16(5):550–554, 1994.

Tommi Jaakkola, Marina Meila, and Tony Jebara. Maximum entropy discrimination, 1999.

Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information And Computation*, 132(1):1–64, January 1997.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal Of Machine Learning Research (JMLR)*, 5:361–397, 2004.

Nick Littlestone. *Mistake bounds and logarithmic linear-threshold learning algorithms*. PhD thesis, U. C. Santa Cruz, March 1989.

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

Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious URLs: An application of large-scale online learning. In *Proc. Of The International Conference On Machine Learning (ICML)*, 2009.

Justin Ma, Alex Kulesza, Koby Crammer, Mark Dredze, Lawrence Saul, and Fernando Pereira. Exploiting feature covariance in high-dimensional online learning. In *AIStats*, 2010.

Andrew McCallum. MALLET: A machine learning for language toolkit. `http://mallet.cs.umass.edu`, 2002.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Large margin online learning algorithms for scalable structured classification. In *NIPS Workshop On Structured Outputs*, 2004.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Flexible text segmentation with structured multilabel classification. In *Empirical Methods In Natural Language Processing (EMNLP)*, 2005a.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Association For Computational Linguistics (ACL)*, 2005b.

Ryan McDonald, Keith Hall, and Gideon Mann. Distributed training strategies for the structured perceptron. In *North American Chapter Of The Association For Computational Linguistics (NAACL)*, 2010.

H. Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. In *Proceedings Of The Twenty Third Annual Conference On Learning Theory*, 2010.

Thomas P. Minka, Rongjing Xiang, and Yuan Alan Qi. Virtual vector machine for Bayesian online classification. In *Proceedings Of The Twenty Fifth Conference On Uncertainty In Artificial Intelligence*, 2009.

Francesco Orabona and Koby Crammer. New adaptive algorithms for online classification. In *Advances In Neural Information Processing Systems 24*, 2010.

Kaare B. Petersen and Michael S. Pedersen. The matrix cookbook, oct 2008. URL `http://www2.imm.dtu.dk/pubdb/p.php?3274`. Version 20081110.

John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In P.J. Bartlett, B. Schölkopf, D. Schuurmans, and A. J Smola, editors, *Advances In Large Margin Classifiers*. MIT Press, 1998.

Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958. (Reprinted in *Neurocomputing* (MIT Press, 1988).).

Libin Shen, Giorgio Satta, and Aravind K. Joshi. Guided learning for bidirectional sequence classification. In *Association For Computational Linguistics (ACL)*, 2007.

Pannaga Shivaswamy and Tony Jebara. Ellipsoidal kernel machines. In *Artificial Intelligence And Statistics (AISTATS)*, 2007.

Pannaga Shivaswamy and Tony Jebara. Empirical Bernstein boosting. In Y.W. Teh and M. Titterington, editors, *Proceedings Of The Thirteenth International Conference On Artificial Intelligence And Statistics (AISTATS) 2010*, volume Volume 9 of JMLR: W&CP, pages 733–740, May 13-15 2010a.

Pannagadatta K. Shivaswamy and Tony Jebara. Maximum relative margin and data-dependent regularization. *Journal Of Machine Learning Research*, 11:747–788, 2010b.

Richard S. Sutton. Adapting bias by gradient descent: an incremental version of delta-bar-delta. In *Proceedings Of The Tenth National Conference On Artificial Intelligence*, pages 171–176. MIT Press, 1992.

Michael E. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal Of Machine Learning Research*, 1:211–244, 2001.

Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal Of Machine Learning Research (JMLR)*, 2001.

Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *International Conference On Machine Learning (ICML)*, 2004.