Confidence-Weighted Linear Classification

Mark Dredze MDREDZE@CIS.UPENN.EDU
Koby Crammer CRAMMER@CIS.UPENN.EDU
Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104 USA

Fernando Pereira[1] PEREIRA@GOOGLE.COM
Google, Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043 USA

Abstract

We introduce confidence-weighted linear classifiers, which add parameter confidence information to linear classifiers. Online learners in this setting update both classifier parameters and the estimate of their confidence. The particular online algorithms we study here maintain a Gaussian distribution over parameter vectors and update the mean and covariance of the distribution with each instance. Empirical evaluation on a range of NLP tasks show that our algorithm improves over other state of the art online and batch methods, learns faster in the online setting, and lends itself to better classifier combination after parallel training.

1. Introduction

Online learning algorithms operate on a single instance at a time, allowing for updates that are fast, simple, make few assumptions about the data, and perform well in wide range of practical settings. Online learning algorithms have become especially popular in natural language processing for tasks including classification, tagging, and parsing. In this paper, we revisit the design of linear classifier learning informed by the particularities of natural language tasks. Specifically, feature representations for natural language processing have very high dimension (millions of features derived from words and word combinations are common), and most features are observed on only a small fraction of instances. Nevertheless, those many rare features are important in classifying the instances in which they occur. Therefore, it is worth investigating whether online learning algorithms for linear classifiers could be improved to take advantage of these particularities of natural language data.

We introduce *confidence-weighted* (CW) learning, a new class of online learning methods that maintain a probabilistic measure of confidence in each parameter. Less confident parameters are updated more aggressively than more confident ones. Parameter confidence is formalized with a Gaussian distribution over parameter vectors, which is updated for each new training instance so that the probability of correct classification for that instance under the updated distribution meets a specified confidence. We show superior classification accuracy over state-of-the-art online and batch baselines, faster learning, and new classifier combination methods after parallel training.

We begin with a discussion of the motivating particularities of natural language data. We then derive our algorithm and discuss variants. A series of experiments shows CW learning's empirical benefits. We conclude with a discussion of related work.

2. Online Algorithms and NLP

In natural language classification tasks, many different features, most of which are binary and are infrequently on, can be weakly indicative of a particular class. Therefore, we have both data sparseness, which demands large training sets, and very high dimensional parameter vectors. For certain types of problems, such as structured prediction in tagging or parsing, the size and processing complexity of individual instances make it difficult to keep more than a small number of instances in main memory. These particularities make online algorithms, which process a single instance at a time, a good match for natural-language tasks. Processing large amounts of data is simple for online methods, which require observing each instance once — though in practice several iterations may be

[1] Work done at the University of Pennsylvania.

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008. Copyright 2008 by the author(s)/owner(s).

necessary — and update parameter vectors using a single instance at a time. In addition, the simple nature of most online updates make them very fast.

However, while online algorithms do well with large numbers of features and instances, they are not designed for the heavy tailed feature distributions characteristic of natural language tasks. This type of feature distribution can have a detrimental effect on learning. With typical linear classifier training algorithms, such as the perceptron or passive-aggressive (PA) algorithms (Rosenblatt, 1958; Crammer et al., 2006), the parameters of binary features are only updated when the features occur. Therefore, frequent features typically receive more updates. 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 parameter estimates for common features and inaccurate values for rare features. However, no distinction is made between these feature types in most online algorithms.

Consider an illustrative example from the problem of 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 feature ("dull") instead of the common well estimated feature ("liked.") This update incorrectly modified "liked" and does not attribute enough negative weight to "dull," thereby decreasing the rate of convergence.

This example demonstrates how a lack of memory for previous instances — 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 parameter 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, for general real-valued features, it could keep the cumulative second moment per feature. The larger the count or second moement, the more confidence in a feature's weight. These estimates are then used to influence parameter updates. Instead of equally updating every feature weight for the features present in an instance, the update favors changing more low-confidence weights than high-confidence ones. At each update, the confidence in all observed features is increased by focusing the update on low confidence features. 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 parameter.

3. Online Learning of Linear Classifiers

Online algorithms operate in rounds. On round i the algorithm receives an instance $\boldsymbol{x}_i \in \mathbb{R}^d$ 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.

Just as in many well known algorithms, such as the perceptron and support vector machines, in this work our prediction rules are linear classifiers

$$f_{\boldsymbol{w}}(\boldsymbol{x}) : f_{\boldsymbol{w}}(\boldsymbol{x}) = \text{sign}(\boldsymbol{x} \cdot \boldsymbol{w}) \ . \tag{1}$$

If we fix the norm of $\boldsymbol{w}$, we can identify $f_{\boldsymbol{w}}$ with $\boldsymbol{w}$, and we will use $\boldsymbol{w}$ in the rest of this work.

The *margin* of an example $(\boldsymbol{x}, y)$ with respect to a specific classifier $\boldsymbol{w}$ is given by $y(\boldsymbol{w} \cdot \boldsymbol{x})$. The sign of the margin is positive iff the classifier $\boldsymbol{w}$ predicts correctly the true label y. The absolute value of the margin $|y(\boldsymbol{w} \cdot \boldsymbol{x})| = |\boldsymbol{w} \cdot \boldsymbol{x}|$ is often thought of as the *confidence* in the prediction, with larger positive values corresponding to more confident correct predictions. We denote the margin at round i by $m_i = y_i(\boldsymbol{w}_i \cdot \boldsymbol{x}_i)$.

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

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \alpha_i y_i \boldsymbol{x}_i \ , \tag{2}$$

for some non-negative coefficients α_i.

In this paper we focus on PA updates (Crammer et al., 2006) for linear classifiers. After predicting with $\boldsymbol{w}_i$ on the ith round and receiving the true label y_i, the algorithm updates the prediction function such that the example $(\boldsymbol{x}_i, y_i)$ will be classified correctly with a fixed margin (which can always be scaled to 1):

$$\begin{aligned} \boldsymbol{w}_{i+1} = \min_{\boldsymbol{w}} \quad & \frac{1}{2}\left\|\boldsymbol{w}_i - \boldsymbol{w}\right\|^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i) \geq 1 \ . \end{aligned} \tag{3}$$

The dual of (3) gives us the round coefficients for (2):

$$\alpha_i = \max\left\{\frac{1 - y_i\left(\boldsymbol{w}_i\cdot\boldsymbol{x}_i\right)}{\left\|\boldsymbol{x}_i\right\|^2}, 0\right\}$$

Crammer et al. (2006) provide a theoretical analysis of algorithms of this form, and they have been shown to work well in a variety of applications.

4. Distributions over Classifiers

We model parameter confidence for a linear classifier with a diagonal Gaussian distribution with mean $\boldsymbol{\mu} \in \mathbb{R}^d$ and standard deviation $\boldsymbol{\sigma} \in \mathbb{R}^d$. The values μ_j and σ_j represent our knowledge of and confidence in the parameter for feature j. The smaller σ_j, the more confidence we have in the mean parameter value μ_j. For simplicity of presentation, we use a covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$ for the distribution, with diagonal $\boldsymbol{\sigma}$ and zero for off-diagonal elements. Note that while our motivation assumed sparse binary features, the algorithm does not depend on that assumption.

Conceptually, to classify an input instance $\boldsymbol{x}$, we draw a parameter vector $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and predict the label according to the sign of $\boldsymbol{w}\cdot\boldsymbol{x}$. This multivariate Gaussian distribution over parameter vectors induces a univariate Gaussian distribution over the margin viewed as a random variable:

$$M \sim \mathcal{N}\left(y_i(\boldsymbol{\mu}\cdot\boldsymbol{x}_i)\ ,\ \boldsymbol{x}_i^\top\Sigma\boldsymbol{x}_i\right) \ .$$

The mean of the margin is the margin of the averaged parameter vector and its variance is proportional to the length of the projection of $\boldsymbol{x}_i$ on Σ_i. Since a prediction is correct iff the margin is non-negative, the probability of a correct prediction is

$$\Pr_{\boldsymbol{w}\sim\mathcal{N}(\boldsymbol{\mu},\Sigma)}\left[M \geq 0\right] = \Pr_{\boldsymbol{w}\sim\mathcal{N}(\boldsymbol{\mu},\Sigma)}\left[y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i\right) \geq 0\right] \ .$$

When possible, we omit the explicit dependency on the distribution parameters and write $\Pr\left[y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i\right) \geq 0\right]$.

4.1. Update

On round i, the algorithm adjusts the distribution to ensure that the probability of a correct prediction for training instance i is no smaller than the confidence hyperparameter $\eta \in [0, 1]$:

$$\Pr\left[y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i\right) \geq 0\right] \geq \eta \ . \tag{4}$$

Following the intuition underlying the PA algorithms (Crammer et al., 2006), our algorithm chooses the distribution closest in the KL divergence sense to the current distribution $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$. Thus, on round i, the algorithm sets the parameters of the distribution by solving the following optimization problem:

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

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

We now develop both the objective and the constraint of this optimization problem following Boyd and Vandenberghe, (2004, page 158). We start with the constraint (6). As noted above, under the distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, the margin for $(\boldsymbol{x}_i, y_i)$ has a Gaussian distribution with mean $\mu_M = y_i\left(\boldsymbol{\mu}\cdot\boldsymbol{x}_i\right)$ and variance $\sigma_M^2 = \boldsymbol{x}_i^\top\Sigma\boldsymbol{x}_i$. Thus the probability of a *wrong* classification is

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

Since $\left(M - \mu_M\right)/\sigma_M$ is a normally distributed random variable, the above probability equals $\Phi\left(-\mu_M/\sigma_M\right)$, where Φ is the cumulative function of the normal distribution. Thus we can rewrite (6) as

$$\frac{-\mu_M}{\sigma_M} \leq \Phi^{-1}\left(1-\eta\right) = -\Phi^{-1}\left(\eta\right) \ .$$

Substituting μ_M and σ_M by their definitions and rearranging terms we obtain:

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

where $\phi = \Phi^{-1}\left(\eta\right)$.

Unfortunately, this constraint is not convex in Σ, so we linearize it by omitting the square root:

$$y_i(\boldsymbol{\mu}\cdot\boldsymbol{x}_i) \geq \phi\left(\boldsymbol{x}_i^\top\Sigma\boldsymbol{x}_i\right) \ . \tag{7}$$

Conceptually, this is a large-margin constraint, where the value of the margin requirement depends on the example $\boldsymbol{x}_i$ via a quadratic form.

We now study the objective (5). The KL divergence between two Gaussians is given by

$$\begin{aligned} &\mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0) \parallel \mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)\right) = \\ &\qquad \frac{1}{2}\left(\log\left(\frac{\det\Sigma_1}{\det\Sigma_0}\right) + \mathrm{Tr}\left(\Sigma_1^{-1}\Sigma_0\right)\right. \\ &\qquad\qquad \left. + \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0\right)^\top\Sigma_1^{-1}\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0\right) - d\right) \ . \end{aligned} \tag{8}$$

Using the foregoing equations and omitting irrelevant constants, we obtain the following revised optimization problem:

$$\begin{aligned}(\boldsymbol{\mu}_{i+1}, \Sigma_{i+1}) = \min\ & \frac{1}{2}\log\left(\frac{\det \Sigma_i}{\det \Sigma}\right) + \frac{1}{2}\mathrm{Tr}\left(\Sigma_i^{-1}\Sigma\right) \\ & + \frac{1}{2}\left(\boldsymbol{\mu}_i - \boldsymbol{\mu}\right)^\top \Sigma_i^{-1}\left(\boldsymbol{\mu}_i - \boldsymbol{\mu}\right) \\ \text{s.t.}\ & y_i(\boldsymbol{\mu}\cdot\boldsymbol{x}_i) \geq \phi\left(\boldsymbol{x}_i^\top \Sigma \boldsymbol{x}_i\right)\ . \end{aligned} \tag{9}$$

The optimization objective is convex in $\boldsymbol{\mu}$ and Σ simultaneously and the constraint is linear, so any convex optimization solver could be used to solve this problem. We call the corresponding update *Variance-Exact*. However, for efficiency we prefer a closed-form approximate update that we call *Variance*. In this approximation, we allow the solution for Σ_{i+1} in (9) to produce (implicitly) a full matrix, and then project it to a diagonal matrix, where the non-zero off-diagonal entries are dropped. The Lagrangian for this optimization is

$$\begin{aligned}\mathcal{L} \ =\ & \frac{1}{2}\log\left(\frac{\det \Sigma_i}{\det \Sigma}\right) + \frac{1}{2}\mathrm{Tr}\left(\Sigma_i^{-1}\Sigma\right) \\ & + \frac{1}{2}\left(\boldsymbol{\mu}_i - \boldsymbol{\mu}\right)^\top \Sigma_i^{-1}\left(\boldsymbol{\mu}_i - \boldsymbol{\mu}\right) \\ & + \alpha\left(-y_i\left(\boldsymbol{\mu}\cdot\boldsymbol{x}_i\right) + \phi\left(\boldsymbol{x}_i^\top \Sigma \boldsymbol{x}_i\right)\right)\ . \end{aligned} \tag{10}$$

At the optimum, we must have

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

Assuming Σ_i is non-singular we get,

$$\boldsymbol{\mu}_{i+1} = \boldsymbol{\mu}_i + \alpha y_i \Sigma_i \boldsymbol{x}_i\ . \tag{11}$$

At the optimum, we must also have

$$\frac{\partial}{\partial \Sigma}\mathcal{L} = -\frac{1}{2}\Sigma^{-1} + \frac{1}{2}\Sigma_i^{-1} + \phi\alpha \boldsymbol{x}_i \boldsymbol{x}_i^\top = 0\ .$$

Solving for Σ^{-1} we obtain

$$\Sigma_{i+1}^{-1} = \Sigma_i^{-1} + 2\alpha\phi \boldsymbol{x}_i \boldsymbol{x}_i^\top\ . \tag{12}$$

Finally, we compute the inverse of (12) using the Woodbury identity (Petersen & Pedersen, 2007, Eq. 135) and get,

$$\begin{aligned}\Sigma_{i+1} &= \left(\Sigma_i^{-1} + 2\alpha\phi \boldsymbol{x}_i \boldsymbol{x}_i^\top\right)^{-1} \\ &= \Sigma_i - \Sigma_i \boldsymbol{x}_i \left(\frac{1}{2\alpha\phi} + \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i\right)^{-1} \boldsymbol{x}_i^\top \Sigma_i \\ &= \Sigma_i - \Sigma_i \boldsymbol{x}_i \frac{2\alpha\phi}{1 + 2\alpha\phi \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i} \boldsymbol{x}_i^\top \Sigma_i\ . \end{aligned} \tag{13}$$

The KKT conditions for the optimization imply that the either $\alpha = 0$, and no update is needed, or the constraint (7) is an equality after the udpate. Substituting (11) and (13) into the equality version of (7), we obtain:

$$\begin{aligned}& y_i\left(\boldsymbol{x}_i\cdot\left(\boldsymbol{\mu}_i + \alpha y_i \Sigma_i \boldsymbol{x}_i\right)\right) = \\ & \phi\left(\boldsymbol{x}_i^\top\left(\Sigma_i - \Sigma_i \boldsymbol{x}_i \frac{2\alpha\phi}{1 + 2\alpha\phi \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i} \boldsymbol{x}_i^\top \Sigma_i\right) \boldsymbol{x}_i\right)\ . \end{aligned} \tag{14}$$

Rearranging terms we get,

$$\begin{aligned}& y_i\left(\boldsymbol{x}_i\cdot\boldsymbol{\mu}_i\right) + \alpha \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i = \\ & \quad \phi \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i - \phi\left(\boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i\right)^2 \frac{2\alpha\phi}{1 + 2\alpha\phi \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i}\ . \end{aligned} \tag{15}$$

For simplicity, let $M_i = y_i\left(\boldsymbol{x}_i\cdot\boldsymbol{\mu}_i\right)$ be the mean margin and $V_i = \boldsymbol{x}_i^\top \Sigma_i \boldsymbol{x}_i$ be the margin variance before the update. Substituting these into (15) we get,

$$M_i + \alpha V_i = \phi V_i - \phi V_i^2 \frac{2\alpha\phi}{1 + 2\alpha\phi V_i}\ .$$

It is straightforward to see that this 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:

$$\gamma_i = \frac{-(1+2\phi M_i) + \sqrt{(1+2\phi M_i)^2 - 8\phi\left(M_i - \phi V_i\right)}}{4\phi V_i}\ . \tag{16}$$

The constraint (7) 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 (16) we have that $\gamma_i > 0$. If, instead, $1 + 2\phi M_i \geq 0$, then, again by (16), we have

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

From the KKT conditions, either $\alpha_i = 0$ or (9) is satisfied as an equality. In the later case, (14) holds, and thus $\alpha_i = \gamma_i > 0$. To summarize, we have proved the following:

Lemma 1 *The optimal value of the Lagrange multiplier is given by* $\alpha_i = \max\left\{\gamma_i, 0\right\}$.

The above derivation yields a full covariance matrix. As noted above, we restrict ourself to diagonal matrices and thus we project the solution into the set of diagonal matrices to get our approximation. In practice, it is equivalent to compute α_i as above but update with the following rule instead of (12).

$$\Sigma_{i+1}^{-1} = \Sigma_i^{-1} + 2\alpha\phi\,\mathrm{diag}\left(\boldsymbol{x}_i\right)\ , \tag{17}$$

Algorithm 1 Variance Algorithm (Approximate)

```
**Input:** confidence parameter $\phi = \Phi^{-1}(\eta)$
           initial variance parameter $a > 0$
**Initialize:** $\boldsymbol{\mu}_1 = \mathbf{0}$ , $\Sigma_1 = aI$
**for** $i = 1, 2 \ldots$ **do**
  Receive $\boldsymbol{x}_i \in \mathbb{R}^d$ , $y_i \in \{+1, -1\}$
  Set the following variables:
    $\alpha_i$ as in Lemma 1
    $\boldsymbol{\mu}_{i+1} = \boldsymbol{\mu}_i + \alpha_i y_i \Sigma_i \boldsymbol{x}_i$ (11)
    $\Sigma_{i+1}^{-1} = \Sigma_i^{-1} + 2\alpha_i \phi \, \mathrm{diag}(\boldsymbol{x}_i)$ (17)
**end for**
```

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

Table 1. 20 Newsgroups binary decision tasks.

where $\mathrm{diag}(\boldsymbol{x}_i)$ is a diagonal matrix with the square of the elements of $\boldsymbol{x}_i$ on the diagonal.

The pesudocode of the algorithm appears in Alg. 1. From the initalization of Σ_1 and the update rule of (12), we conclude that the eigenvalues of Σ_i are shirinking, but never set to zero expliclty, and thus the covariance matrices Σ_i are not singular.

5. Evaluation

We evaluated our Variance and Variance-Exact algorithms on three popular NLP datasets. Each dataset contains several binary classification tasks from which we selected a total of 12 problems, each contains a balanced mixture of instance labels.

20 Newsgroups The 20 Newsgroups corpus contains approximately 20,000 newsgroup messages, partitioned across 20 different newsgroups.[2] The dataset 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 dataset by creating binary decision problems of choosing between two similar groups, as shown in Table 1. Each message was represented as a binary bag-of-words. For each problem we selected 1800 instances.

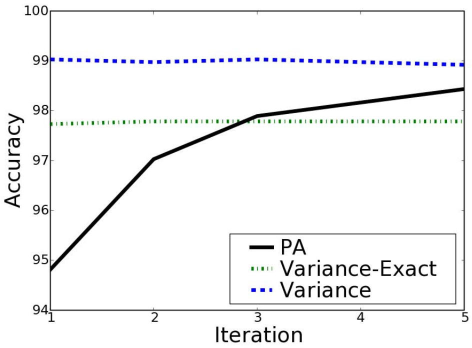

Figure 1. Accuracy on test data after each iteration on the "talk" dataset.

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), and 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 instances using a bag of words representation with binary features.

Sentiment We obtained a larger version of the sentiment multi-domain dataset of Blitzer et al. (2007) containing product reviews from 6 Amazon domains (book, dvd, electronics, kitchen, music, video). The goal in each domain is to classify a product review as either positive or negative. Feature extraction follows Blitzer et al. (2007). For each problem we selected 2000 instances using uni/bi-grams with counts.

Each dataset was randomly divided for 10-fold cross validation experiments. Classifier parameters (ϕ for CW and C for PA) 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.

5.1. Results

We start by examining the performance of the Variance and Variance-Exact versions of our method, discussed in the preceding section, against a PA algorithm. All three algorithms were run on the datasets described above and each training phase consisted of five passes over the training data, which seemed to be

[2] `http://people.csail.mit.edu/jrennie/20Newsgroups/`

	Task	*PA*	*Variance*	*Variance-Exact*	*SVM*	*Maxent*	*SGD*
20 Newsgroups	comp	8.90	**†6.33**	9.63	∗7.67	∗7.62	7.36
	sci	4.22	**†1.78**	3.3	†3.51	†3.55	†4.77
	talk	1.57	1.09	2.21	**0.91**	**0.91**	1.36
Reuters	Business	17.80	17.65	17.70	⋆15.64	**⋆15.10**	⋆15.85
	Insurance	9.76	**∗8.45**	9.49	9.19	8.59	9.05
	Retail	15.41	**†11.05**	14.14	∗12.80	∗12.30	†14.31
Sentiment	books	19.55	**∗17.40**	20.45	†20.45	†19.91	∗19.41
	dvds	19.71	**19.11**	19.91	20.09	19.26	20.20
	electronics	17.40	**†14.10**	17.44	†16.80	†16.21	†16.81
	kitchen	15.64	**∗14.24**	16.35	15.20	14.94	∗15.60
	music	20.05	**∗18.10**	19.66	19.35	19.45	18.81
	videos	19.86	**⋆17.20**	19.85	†20.70	†19.45	⋆19.65

Table 2. Error on test data using batch training. Statistical significance (McNemar) is measured against PA or the batch method against Variance. (∗ p=.05, ⋆ p=.01, † p=.001)

enough to yield convergence. The average error on the test set for the three algorithms on all twelve datasets is shown in table 2.

Variance-Exact achieved about the same performance as PA, with each method achieving a lower error on half of the datasets. In contrast, Variance (approximate) significantly improves over PA, achieving lower error on all twelve datasets, with statistically significant results on nine of them.

As discussed above, online algorithms are attractive even for batch learning because of their simplicity and ability to operate on extremely large datasets. In the batch setting, these algorithms are run several times over the training data, which yields slower performance than single pass learning (Carvalho & Cohen, 2006). Our algorithm improves on both accuracy and learning speed by requiring fewer iterations over the training data. Such behavior can be seen on the "talk" dataset in Figure 1, which shows accuracy on test data after each iteration of the PA baseline and the two variance algorithms. While Variance 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. Across the twelve tasks, Variance yields a 3.7% error reduction while PA gives a 12.4% reduction between the first and fifth iteration, indicating that multiple iterations help PA more. The plot also illustrates Variance-Exact's behavior, which initially beats PA but does not improve. In fact, on eleven of the twelve datasets, Variance-Exact beats PA on the first iteration. The exact update results in aggressive behavior causing the algorithm to converge very quickly, even more so than Variance. It appears that the approximate update in Variance reduces overtraining and yields the best accuracy.

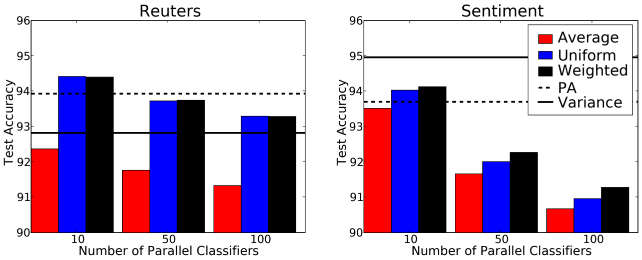

Figure 2. Results for Reuters (800k) and Sentiment (1000k) averaged over 4 runs. Horizontal lines show the test accuracy of a model trained on the entire training set. Vertical bars show the performance of n (10, 50, 100) classifiers trained on disjoint sections of the data as the average performance, uniform combination, or weighted combination. All improvements are statistically significant except between uniform and weighted for Reuters.

5.2. 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 dataset, an ability beyond online algorithms. In general, when batch algorithms can be applied they perform better. We compare our new online algorithm (Variance) against two standard batch algorithms: maxent classification (default configuration of the maxent learner in McCallum (2002)) and support vector machines (LibSVM (Chang & Lin, 2001)). We also include stochastic gradient descent (SGD) (Blitzer et al., 2007), which performs well for NLP tasks. Classifier parameters (Gaussian prior for maxent, C for SVM and the learning rate for SGD) were tuned as for the online methods.

Results for batch learning are shown in Table 2. As expected, the batch methods tend to do better than PA, with SVM doing better 9 times and maxent 11 times. However, in most cases Variance improves over the batch method, doing better than SVM and maxent 10 out of 12 times (7 statistically significant.) These results show that in these tasks, the much faster and simpler online algorithm performs better than the slower more complex batch methods.

We also evaluated the effects of commonly used techniques for online and batch learning, including averaging and TFIDF features, none of which improved accuracy. Although the above datasets are balanced with respect to labels and predictive features, we also evaluated the methods on variant datasets with unbalanced label or feature distributions, and still saw similar benefits from the Variance method.

5.3. Large Datasets

Online algorithms are especially attractive in tasks where training data exceeds available main memory. However, even a single sequential pass over the data can be impractical for extremely large training sets, so we investigate training different models on different portions of the data in parallel and combining the learned classifiers into a single classifier. 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.

Averaging models trained in parallel assumes that each model has an equally accurate estimate of the model parameters. However, our model provides a confidence value for each parameter, allowing for a more intelligent combination of parameters from multiple models. Specifically, we compute the combined model Gaussian that minimizes the total divergence to the set C of individually trained classifiers for some divergence operator D:

$$\min_{\boldsymbol{\mu},\Sigma} \sum_{c\in C} D((\boldsymbol{\mu},\Sigma)||(\boldsymbol{\mu}_c,\Sigma_c)), \tag{18}$$

If D is the Euclidean distance, this is just the average of the individual models. If D is the KL divergence, the minimization leads to the following weighted combination of individual model means:
$\boldsymbol{\mu} = \left(\sum_{c\in C}\Sigma_c^{-1}\right)^{-1}\sum_{c\in C}\Sigma_c^{-1}\boldsymbol{\mu}_c$, $\Sigma^{-1} = \sum_{c\in C}\Sigma_c^{-1}$.

We evaluate the single model performance of the PA baseline and our method. For our method, we evaluate classifier combination by training n (10, 50, 100) models by dividing the instance 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 parameters (uniform) and the combination using the KL distance (weighted) on the test data across 4 randomized runs.

We evaluated classifier combination on two datasets. The combined product reviews for all the domains in Blitzer et al. (2007) yield one million sentiment instances. 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 instances of which 381,325 are labeled corporate. For the two datasets, we created four random splits each with one million training instances and 10,000 test instances. Parameters were optimized by training on 5K random instances and testing on 10K.

The two datasets use very different feature representations. 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 tend to occur several times during training while many sentiment features occur only once.

Average accuracy on the test sets are reported in Figure 2. For Reuters data, the PA single model achieves higher accuracy than Variance, possibly because of the low feature to document ratio. However, combining 10 Variance classifiers achieves the best performance. For sentiment, combining 10 classifiers beats PA but is not as good as a single Variance model. In every case, combining the classifiers using either uniform or weighted improves over each model individually. On sentiment weighted combination improves over uniform combination and in Reuters the models are equivalent.

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

6. Related Work

The idea of using parameter-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. The second-order perceptron (SOP) (Cesa-Bianchi et al., 2005) is perhaps the closest to our CW algorithm. Both are online algorithms that maintain a weight vector and some statistics about previous examples. While the SOP models certainty with feature counts, CW learning models uncertainty with a Gaussian distribution. CW algorithms have a probabilistic motivation, while the SOP is based on the geometric idea of replacing a ball around the input examples with a refined ellipsoid. Shivaswamy and Jebara (2007) used this intuition in the context of batch learning.

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 the standard Bayesian updates used in GPC (Rasmussen & Williams, 2006, Ch. 3), avoiding the challenging issues in approximating posteriors in GPC. 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 finite number of weight-vectors updated simultaneously.

Finally, 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 & Bottou, 2005). The extreme case is to use each example once, without repetitions, as in the multiplicative update method of Carvalho and Cohen (2006).

Conclusion: We have presented confidence-weighted linear classifiers, a new learning method designed for NLP problems based on the notion of parameter confidence. The algorithm maintains a distribution over parameter vectors; online updates both improve the parameter estimates and reduce the distribution's variance. Our method improves over both online and batch methods and learns faster on a dozen NLP datasets. Additionally, our new algorithms allow more intelligent classifier combination techniques, yielding improved performance after parallel learning. We plan to explore theoretical properties and other aspects of CW classifiers, such as multi-class and structured prediction tasks, and other data types.

Acknowledgements: This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185.

References

Blitzer, J., Dredze, M., & Pereira, F. (2007). Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. *Association of Computational Linguistics (ACL).*

Bordes, A., & Bottou, L. (2005). The huller: a simple and efficient online svm. *European Conference on Machine Learning(ECML), LNAI 3720.*

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization.* Cambridge University Press.

Carvalho, V. R., & Cohen, W. W. (2006). Single-pass online learning: Performance, voting schemes and online feature selection. *KDD-2006.*

Cesa-Bianchi, N., Conconi, A., & Gentile, C. (2005). A second-order perceptron algorithm. *SIAM Journal on Computing*, *34*, 640 – 668.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines.* Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., & Singer, Y. (2006). Online passive-aggressive algorithms. *JMLR*, *7*, 551–585.

Harrington, E., Herbrich, R., Kivinen, J., Platt, J., & Williamson, R. (2003). Online bayes point machines. *7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD).*

Herbrich, R., Graepel, T., & C.Campbell (2001). Bayes point machinesonline passive-aggressive algorithms. *JMLR*, *1*, 245–279.

Lewis, D. D., Yand, Y., Rose, T., & Li., F. (2004). Rcv1: A new benchmark collection for text categorization research. *JMLR*, *5*, 361–397.

McCallum, A. K. (2002). Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu.

Petersen, K. B., & Pedersen, M. S. (2007). *The matrix cookbook.*

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning.* The MIT Press.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psych. Rev.*, *68*, 386–407.

Shivaswamy, P., & Jebara, T. (2007). Ellipsoidal kernel machines. *Artificial Intelligence and Statistics.*

Sutton, R. S. (1992). Adapting bias by gradient descent: an incremental version of delta-bar-delta. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 171–176). MIT Press.