Exploiting Feature Covariance in High-Dimensional Online Learning

Justin Ma
UC San Diego
jtma@cs.ucsd.edu

Alex Kulesza
University of Pennsylvania
kulesza@cis.upenn.edu

Mark Dredze
Johns Hopkins University
mdredze@cs.jhu.edu

Koby Crammer
The Technion
koby@ee.technion.ac.il

Lawrence K. Saul
UC San Diego
saul@cs.ucsd.edu

Fernando Pereira
Google
pereira@google.com

Abstract

Some online algorithms for linear classification model the uncertainty in their weights over the course of learning. Modeling the full covariance structure of the weights can provide a significant advantage for classification. However, for high-dimensional, large-scale data, even though there may be many second-order feature interactions, it is computationally infeasible to maintain this covariance structure. To extend second-order methods to high-dimensional data, we develop low-rank approximations of the covariance structure. We evaluate our approach on both synthetic and real-world data sets using the confidence-weighted (Dredze et al., 2008; Crammer et al., 2009a) online learning framework. We show improvements over diagonal covariance matrices for both low and high-dimensional data.

1 Introduction

Online linear classification is well-suited for learning from large, high-dimensional, rapidly growing data sets because it makes a single pass over the training data and only needs to store the current example and the current classification hypothesis. Among online linear classifier learners, those that maintain second-order information have shown special promise because they offer faster convergence on a single pass over the data. In confidence-weighted (CW) learning (Dredze et al., 2008; Crammer et al., 2009a) and Bayesian logistic regression (Jaakkola & Jordan, 2000; MacKay, 1992; Spiegelhalter & Lauritzen, 1990), second-order information represents uncertainty about the linear classifier's feature weight estimates and can be modeled as a Gaussian distribution over the classifier's weight vector. The mean of the weight vector is used for classification, and the covariance matrix is used to modulate the learning rate over different features.

Unfortunately, storing and updating the full covariance matrix requires time and space quadratic in the number of features, which becomes prohibitively expensive when that number grows much beyond 10^4. Efficient diagonal approximations, which scale linearly with the number of features, are often used in practice (Dredze et al., 2008; Crammer et al., 2009a; Crammer et al., 2009b). However, these approximations sacrifice information about cross-feature correlations that can lead to faster convergence. Thus diagonal approximations trade accuracy for speed.

We investigate the nature of this tradeoff, using synthetic experiments to show when it is advantageous to use a full covariance rather than a diagonal covariance matrix. We consider variations in the data's dimensionality, the amount of correlation among the features, and the amount of noise in the data set. We then propose a novel method for online low-rank approximation of the inverse covariance matrices. Our approach forms a practical middle ground, improving performance over diagonal methods without incurring the high computational costs of modeling full covariance. We base our methods on the CW framework, although we believe the approach is also applicable to other covariance-tracking online algorithms such as Bayesian logistic regression, the second-order perceptron (Cesa-Bianchi et al., 2005) and quasi-Newton gra-

Appearing in Proceedings of the 13^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9.

dient descent (Bottou, 1998). We show empirical benefits on a variety of real world data sets.

2 Confidence-Weighted Online Learning

Online learning algorithms operate in rounds. During round t, the algorithm receives an instance $\boldsymbol{x}_t \in \mathbb{R}^d$ and applies its current rule to make a prediction $\hat{y}_t$. It then receives the true label y_t and suffers a loss $\ell(y_t, \hat{y}_t)$. Using this information, the algorithm updates its prediction rule and proceeds to the next round. The goal is to minimize cumulative loss.

In this work we consider binary classification problems where $\hat{y}_t, y_t \in \{-1, +1\}$ and $\ell(y_t, \hat{y}_t) = \mathbb{I}(y_t \neq \hat{y}_t)$ is the zero-one loss function. In this case, the cumulative loss is simply the number of incorrect predictions (mistakes). To predict $\hat{y}_t$ we use a linear model parameterized by a weight vector $\boldsymbol{w} \in \mathbb{R}^d$, $\hat{y}_t = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x}_t)$.

The design of the update rule has a significant impact on performance. A simple approach is to increment the weight vector by $y_t \boldsymbol{x}_t$ whenever the loss is nonzero; this moves the score $\boldsymbol{w} \cdot \boldsymbol{x}_t$ in the right direction and yields the perceptron algorithm. A better approach in many cases is the passive-aggressive rule, which scales the perceptron update to ensure that $\boldsymbol{x}_t$ is correctly classified with margin (Crammer et al., 2006).

More recently, Dredze, Crammer and Pereira proposed a new framework called confidence weighted (CW) learning that allows the update rule to consider confidence information about the model parameters (Dredze et al., 2008; Crammer et al., 2009a). Rather than maintaining a single weight vector from round to round, a CW learner maintains a Gaussian distribution over weight vectors, parameterized by a mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and a covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$, that represents the learner's confidence about its parameter values. By accounting for the shape of this distribution, CW algorithms can make more effective updates to the weights, for example by refining them preferentially along directions that are currently low-confidence (high-variance).

At test time, one imagines drawing a weight vector from the learned distribution and then using it to make a prediction. However, for binary classification it turns out that predictions made using the mean weight vector $\boldsymbol{\mu}$ are Bayes optimal with respect to sampling $\boldsymbol{w}$ (because the Gaussian is symmetric) as well as simpler to produce. Thus in practice the confidence information $\boldsymbol{\Sigma}$ serves primarily as a regularizer for training.

The specific update used by CW classifiers is a passive-aggressive rule modified to account for confidence information. Following round t, a weight vector drawn from the updated distribution is required to correctly classify $\boldsymbol{x}_t$ with probability at least $\eta \in (0.5, 1]$. Subject to this constraint, the algorithm makes the lowest possible KL divergence change to the hypothesis weight distribution:

$$(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) = \min_{\boldsymbol{\mu}, \boldsymbol{\Sigma}} \ \mathrm{D}_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \,\|\, \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)\right) \tag{1}$$

$$\text{s.t.} \ \ \Pr_{\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})}\left[y_t(\boldsymbol{w} \cdot \boldsymbol{x}_t) \geq 0\right] \geq \eta. \tag{2}$$

This optimization can be solved in closed form, yielding the following update equations, known as the CW-Stdev update (Crammer et al., 2009a):

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \alpha_t y_t \boldsymbol{\Sigma}_t \boldsymbol{x}_t, \tag{3}$$

$$\boldsymbol{\Sigma}_{t+1} = \boldsymbol{\Sigma}_t - \beta_t \boldsymbol{\Sigma}_t \boldsymbol{x}_t \boldsymbol{x}_t^\top \boldsymbol{\Sigma}_t. \tag{4}$$

The constants α_t and β_t are nonnegative learning rates computed as given in Eq. 22 of Crammer et al. (2009a). We note that the covariance update can alternatively be written in the inverse:

$$\boldsymbol{\Sigma}_{t+1}^{-1} = \boldsymbol{\Sigma}_t^{-1} + \frac{\alpha_t \phi}{\sqrt{u_t}} \boldsymbol{x}_t \boldsymbol{x}_t^\top, \tag{5}$$

as defined in Eq. 10–11 of Crammer et al. (2009a). This representation of the update is particularly useful for the factored inverse covariance approximation that we discuss in Section 3.

2.1 High-Dimensional Applications

Some of the most successful applications of CW learning involve high-dimensional data, e.g., from natural language processing tasks (Crammer et al., 2009a; Dredze et al., 2008; Ma et al., 2009; Dredze & Crammer, 2008). Clearly, it is impractical to maintain a full d^2 covariance matrix in those cases. Instead, $\boldsymbol{\Sigma}$ is approximated with a diagonal matrix to produce an algorithm that scales linearly with the size of the feature vocabulary. An empirically successful approximation method is to begin with a diagonal matrix and project back onto the set of diagonal matrices after each update. This projection can be done using the ℓ_2 norm, which simply drops the off-diagonal terms in Eq. (4), or using KL-divergence, which corresponds to dropping the off-diagonal terms in Eq. (5). Both of these approaches work well in practice, and for simplicity we proceed here using ℓ_2 projection.

2.2 Benefits of Full $\boldsymbol{\Sigma}$

In diagonal CW learning, the element $\Sigma_{p,p}$ of the covariance matrix encodes the learner's confidence in the

mean weight μ_p for feature p. Given the update rule in Eq. (4), it is easy to see that $\Sigma_{p,p}$ shrinks whenever feature p is observed in the data; this corresponds to increased confidence in μ_p and smaller subsequent updates to that value. Thus, the diagonal of $\mathbf{\Sigma}$ serves to decay the effective learning rate on a per-feature basis, hopefully leading to faster convergence.

However, off-diagonal elements of $\mathbf{\Sigma}$, discarded by the diagonal approximation, can also provide useful guidance during training. In the following we attempt to characterize some of the ways in which full $\mathbf{\Sigma}$ improves training regularization compared to diagonal $\mathbf{\Sigma}$.

Consider a pair of binary features (x_p, x_q) that co-occur frequently. We maintain a separate weight for each of these features, and given enough data a learner can estimate their values independently. However, knowing that the features are correlated, we might hope to do better by replacing them with a new pair of features: $(x_p + x_q, x_p - x_q)$. While this change has no effect on the expressive power of the model (or its literal dimension), it does change its geometry: we have replaced two similar features with one more common feature and another that is usually zero.

In the context of diagonal CW learning, we are now better equipped to learn from these data. Our confidence about the weight for $x_p + x_q$ will grow more quickly than for $x_p - x_q$, because the observed values $x_p + x_q$ are far from zero more often. This enables us to quickly reduce the *effective* dimensionality of the learning problem, since we need not consider large changes to weights that are highly confident. We no longer zig (upon seeing feature p alone) and zag (upon seeing feature q alone); instead we make consistent changes to the newly combined weights. This is analogous to the shift from gradient to conjugate gradient methods in the optimization literature (Nocedal & Wright, 1999, Figs. 5.1 and 5.2). While gradient descent will zigzag when the Hessian of the objective is non-diagonal, conjugate methods effectively diagonalize the Hessian and converge quickly and directly.

For our toy example, we have described a transformation of the features explicitly; however, similar effects can be obtained adaptively and implicitly through the use of full $\mathbf{\Sigma}$. While diagonal methods deal with each feature independently, full methods can tie them together, simplifying the problem of locating a good weight vector by regularizing the updates into an effectively lower-dimensional space. Especially when there are many features and relatively few examples, this can be a significant advantage. We demonstrate this effect with a simple synthetic experiment.

We begin each trial by sampling a true weight vector $\boldsymbol{w}^* \in \mathbb{R}^{10}$ from a ten-dimensional normal distribution. On round t, we flip ten coins to produce "true" binary features $\boldsymbol{z}_t \in \{-1, +1\}^{10}$ and compute the label $y_t = \text{sign}(\boldsymbol{w}^* \cdot \boldsymbol{z}_t)$. We then construct the observed features by creating k duplicates of $\boldsymbol{z}_t$ and randomly flipping 5% of the resulting binary values, producing $\boldsymbol{x}_t \in \{-1, +1\}^{10k}$. These data share many properties with the simple example discussed above.

We applied both full and diagonal CW methods to data sets of varying size for $k \in \{1, \ldots, 10\}$ and recorded the average difference in online accuracy over 100 trials. The results are shown in Fig. 1(a). When there are many features and few examples, full CW learning significantly outperforms the diagonal method (error bars not shown). To demonstrate the regularizing effects of full CW we plot $\text{Tr}(\mathbf{\Sigma}_t)/\lambda_1(\mathbf{\Sigma}_t)$ at each learning round t for the 50 feature case, averaged over 20 trials (Fig. 1(b)). Here, $\lambda_1(\mathbf{\Sigma}_t)$ is the largest eigenvalue of $\mathbf{\Sigma}_t$. We refer to this measurement as the "effective dimension" because it characterizes the eigenvalue distribution of $\mathbf{\Sigma}$ as being either spherical (high-dimension) or squashed (low-dimension). When the effective dimension is low, the learner has fewer degrees of freedom to update its parameters. From the figure it is clear that full CW tightens its regularization more quickly. Note that the two methods have roughly equal $\text{Tr}(\mathbf{\Sigma})$ at each round.

From Fig. 1(a) we also see that, given enough data, diagonal CW learning significantly outperforms the full version. This is because the same ability to adapt to data co-dependencies that helps full CW learning during the early rounds leads it to adapt to noise as it approaches the optimal weight vector when the data are not separable. For example, during rounds 400-500 of the 10 feature experiment (where both methods are essentially converged), the diagonal algorithm adjusts the angle of $\boldsymbol{\mu}_t$ by an average of 0.81° per round, while the full algorithm adjusts it by an average of 2.42° per round. This increased "thrashing" leads to reduced long-term performance of full CW learning.

These observations raise the question of possible intermediates between full and diagonal learning that balance fast convergence (full) and efficient, robust learning in high dimension (diagonal). We explore such a middle ground: a method that can approximate the inter-feature correlations of full CW learning but also scales well to high-dimensional data.

3 Factoring the Covariance Matrix

We observe that a matrix can be stored compactly if it is well approximated by a matrix of low rank. In our approach, we model the inverse covariance matrix for CW as the sum of a diagonal matrix plus a low rank positive semidefinite matrix, giving the following

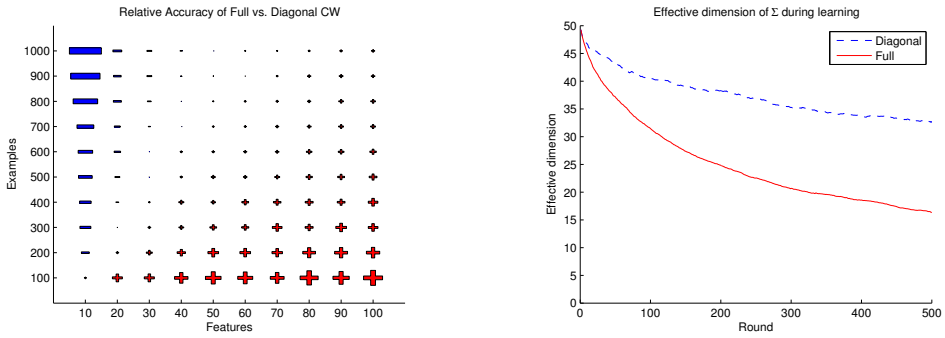


Figure 1: (a) Average accuracy gap between full and diagonal CW, averaged over 100 trials. Pluses indicate full CW outperforming diagonal CW, minuses indicate the reverse, and the scale of each symbol depends linearly on the magnitude of the gap. The largest minus reflects a gap of -7.5%, and the largest plus reflects a gap of 4.2%. (b) Effective dimension of $\boldsymbol{\Sigma}$ at each round, 50 features, averaged over 20 trials.

factored approximation where $\boldsymbol{D}$ is a $d \times d$ diagonal matrix and $\boldsymbol{R}$ is a $d \times m$ rectangular matrix:

$$\boldsymbol{\Sigma}^{-1} \approx \boldsymbol{D} + \boldsymbol{R}\boldsymbol{R}^\top. \tag{6}$$

Intuitively, this approximation is well-suited to CW because each update to the inverse covariance matrix is the addition of a vector outer product.

The approximation in Eq. (6) is inspired by the statistical method of factor analysis (Gorsuch, 1983). However, in standard factor analysis this approximation is used to model the covariance matrix, not the inverse covariance matrix; we will return to this point later. For CW learning, the approximation in Eq. (6) has important advantages over purely low-rank approximations (e.g., singular value decomposition) that do not include a diagonal component. First, for CW learning, we require a proper Gaussian density over the weight vector; the diagonal component in Eq. (6) is needed to ensure that the density is normalizable. Second, the diagonal component models the per-component errors of the remaining low rank approximation, as opposed to assuming that all weights are equally uncertain. The latter assumption, though simplifying, is entirely contrary to the spirit of CW learning. Finally, as we show next, there are iterative updates for learning approximations of the form in Eq. (6) that scale well with the dimensionality of the problem, whereas singular value decomposition may not be feasible for matrices of extremely large size. Indeed, one leading algorithm for PCA in high-dimensional spaces is an iterative estimation procedure (Roweis, 1998) that can be viewed as a special case of the algorithm for maximum likelihood factor analysis. Although we focus on CW, our approach can be applied to other second-order online algorithms that need to store and maintain a positive semidefinite matrix.

3.1 Approximation Algorithm

Our algorithm attempts to minimize a measure of discrepancy between a target matrix $\boldsymbol{P}$ (assumed to be positive semidefinite) and its approximation $\boldsymbol{D}+\boldsymbol{R}\boldsymbol{R}^\top$. To measure discrepancy, we use the KL divergence between a pair of multivariate Gaussian distributions with the same mean (assumed without loss of generality to lie at the origin) but different covariance matrices $\boldsymbol{P}$ and $\boldsymbol{D}+\boldsymbol{R}\boldsymbol{R}^\top$[1]:

$$\min_{\boldsymbol{D},\boldsymbol{R}} \; \mathrm{D}_{\mathrm{KL}}\big(\, \mathcal{N}(\boldsymbol{0}, \boldsymbol{P}) \;||\; \mathcal{N}(\boldsymbol{0}, \boldsymbol{D}+\boldsymbol{R}\boldsymbol{R}^\top) \,\big) \tag{7}$$

Unlike the updates for CW learning in section 2, the optimization in Eq. (7) cannot be solved in closed form. However, we can search for a local minimum of the KL divergence by adapting the iterative updates for maximum likelihood factor analysis. As shorthand, we define the following matrices:

$$\boldsymbol{\Phi} = (\mathbf{I} + \boldsymbol{R}^\top \boldsymbol{D}^{-1} \boldsymbol{R})^{-1}, \tag{8}$$

$$\boldsymbol{\Upsilon} = \boldsymbol{\Phi}\boldsymbol{R}^\top \boldsymbol{D}^{-1}. \tag{9}$$

Note that the matrices $\boldsymbol{\Phi}$ and $\boldsymbol{\Upsilon}$ depend on the current approximation parameters, namely the rectangular matrix $\boldsymbol{R}$ and the diagonal matrix $\boldsymbol{D}$. In terms of these matrices, the updates to minimize Eq. (7) are

$$\boldsymbol{R} \leftarrow \boldsymbol{P}\boldsymbol{\Upsilon}^\top (\boldsymbol{\Phi} + \boldsymbol{\Upsilon}\boldsymbol{P}\boldsymbol{\Upsilon}^\top)^{-1}, \tag{10}$$

$$\boldsymbol{D} \leftarrow \mathrm{diag}(\boldsymbol{P} - \boldsymbol{R}\boldsymbol{\Upsilon}\boldsymbol{P}). \tag{11}$$

To minimize Eq. (7), we alternate between recomputing the matrices in Eqs. (8–9) and updating the model parameters in Eqs. (10–11). Applied in this way, the updates converge monotonically to a local minimum of the KL divergence in Eq. (7). Note that the "target" matrix $\boldsymbol{P}$ remains fixed throughout this procedure.

[1]This distance between matrices is aks as matrix Itakura-Saito or log-det divergence

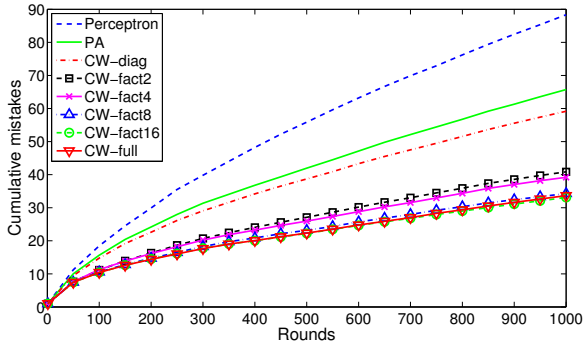


Figure 2: Synthetic data cumulative error for CW-fact(m), CW-full and CW-diag.

3.2 Integration with CW Learning

The algorithm in Section 3.1 integrates naturally with CW learning to provide a compact approximation for full covariance matrices. We refer to this approach as *CW-fact*. To avoid performing the relatively expensive minimization procedure described in Eqs. (8–11) following every update, we augment our approximation with a buffer:

$$\boldsymbol{\Sigma}^{-1} = \boldsymbol{D} + \boldsymbol{R}\boldsymbol{R}^\top + \mathbf{B}\mathbf{B}^\top, \tag{12}$$

where $\mathbf{B}$ is a $d \times m$ matrix holding up to m of the most recent exact updates. We update the buffer $\mathbf{B}$ after each example, but fit the factored model using Eqs. (8–11) only when the buffer becomes full.

We initialize $\boldsymbol{D}$ to the identity matrix, and $\boldsymbol{R}$ and $\boldsymbol{B}$ to zero. The first m updates to $\boldsymbol{\Sigma}$ dictated by the CW algorithm are stored directly in $\boldsymbol{R}$. Using the inverse rule in Eq. (5), the tth column of $\boldsymbol{R}$ is given by $(\frac{\alpha_t \phi}{\sqrt{u_t}})^{\frac{1}{2}} \boldsymbol{x}_t$. The next m updates fill $\boldsymbol{B}$ in the same way. When the buffer is full, we run the algorithm in Section 3.1 to compress the contents of both $\boldsymbol{R}$ and $\boldsymbol{B}$ into $\boldsymbol{D}$ and $\boldsymbol{R}$ (where we set the target matrix to $\boldsymbol{P} = \boldsymbol{D} + \boldsymbol{R}\boldsymbol{R}^\top + \mathbf{B}\mathbf{B}^\top$). This leaves the buffer empty, and the cycle of filling the buffer and compressing repeats for the remainder of the examples.

3.3 Comparison with full and diagonal covariance representations

We begin our empirical results by demonstrating that CW-fact occupies a middle ground between CW-full and CW-diag on the synthetic data described in Section 2. This time we generate 1000 examples with 1000 features ($k = 100$).

Figure 2 shows the cumulative mistake counts averaged over 100 runs. We include perceptron and passive-aggressive results for reference. As we increase m, the accuracy of CW-fact approaches CW-full. (CW-fact's performance did not improve beyond $m = 16$, which makes sense given the ten-dimensional underlying distribution.) Table 1 shows the average runtime and memory overhead for the different variations of CW in this experiment. The memory usage for CW-fact is an order of magnitude improvement over CW-full, and the runtime is 5× faster. Thus, CW-fact provides an adjustable compromise between the high-accuracy of CW-full and the low-overhead of CW-diag.

Table 1: Runtime and memory benchmarks for the synthetic experiment in Figure 2.

	Time (s)	Time w/o buffer (s)	Mem (KB)
CW-diag	0.09	—	7.81
CW-fact2	1.61	2.61	87.21
CW-fact4	1.35	4.11	181.27
CW-fact8	1.21	7.16	370.15
CW-fact16	1.50	16.45	750.90
CW-full	7.00	—	7812.50

3.4 Benefits of buffering and $\boldsymbol{\Sigma}^{-1}$

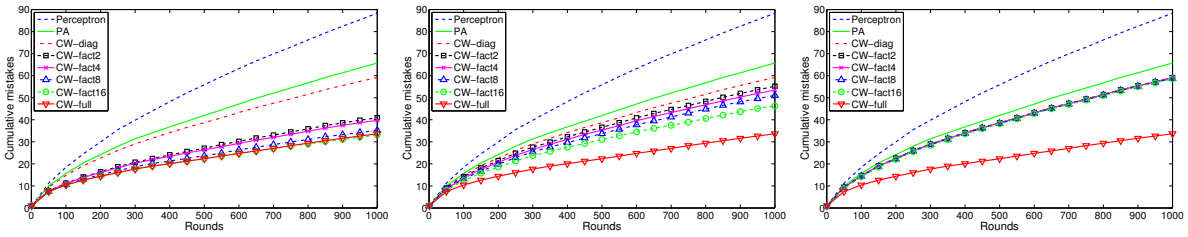
Before we move on, it is worth briefly addressing the effect of the buffer matrix on the performance of CW-fact. Figure 3(a) shows the results of the same synthetic experiment when we eliminate the buffer. The difference in accuracy from the experiment in Figure 2 is negligible. However, Table 1 shows that the computational cost is quite high.

We also tested the performance of buffering alone; that is, simply throwing out the oldest update whenever a new one arrives. The results are in Figure 3(b). They show that while buffering alone offers some benefit, it is less effective than compressing information into $\boldsymbol{R}$ to provide a long-term summary of updates to $\boldsymbol{\Sigma}^{-1}$, as done for CW-fact.

Finally, the algorithm in Section 3.1 appears to be a good fit for approximating $\boldsymbol{\Sigma}^{-1}$, since the update Eq. (5) is additive, matching the form of the approximation. However, we could also consider approximating $\boldsymbol{\Sigma}$ in a similar way, subtracting instead of adding the buffer term in Eq. 12. Figure 3(c) shows that performance is dramatically reduced in this case, perhaps because the additive approximation is a poor fit for the subtractive $\boldsymbol{\Sigma}$ update Eq. (4).

4 Large-Scale Learning

We evaluate CW learning with our factored approximation (CW-fact) on several high-dimensional real-world data sets where the number of features exceeds the number of examples and many features are correlated. We use perceptron, passive-aggressive (PA)



(a) $\boldsymbol{\Sigma}^{-1}$, no buffer (b) $\boldsymbol{\Sigma}^{-1}$, buffering only (c) $\boldsymbol{\Sigma}$, no buffer

Figure 3: Evaluating alternative design decisions for CW-fact with respect to buffering and whether to approximate $\boldsymbol{\Sigma}$ vs. $\boldsymbol{\Sigma}^{-1}$. Results for perceptron, PA, CW-diag and CW-full are repeated for reference.

learning (Crammer et al., 2006), and diagonal CW as baselines. For perceptron and PA, we make predictions at each round using the average of all previous weight vectors, which tends to outperform the single most recent weight vector.

4.1 Detecting malicious URLs

We evaluate CW-fact on a 20-day subset of a live URL data set (Ma et al., 2009) with about 1 million binary features and 64 real-valued features scaled to the interval $[0, 1]$. The goal is to determine whether each website is malicious; the ratio of positive to negative examples is $1 : 2$. There are 20,000 examples collected per day. We subsample the data, preserving the temporal ordering, to produce error bars: in each run an example has a 50% chance of being included, resulting in 10,000 examples per day. Results are computed on 10 samples of 200,000 examples each.

Fig. 4 shows absolute and relative mistake counts over time. CW-fact provides a consistent, 5% relative improvement over CW-diag, which is itself superior to PA and perceptron. This corroborates our findings in Sec. 3, which show improvements in accuracy when there are more features than examples and many features are correlated.

4.2 Web spam

We next consider the web spam data from the PASCAL Large Scale Learning Challenge (Sonnenburg et al., 2008). We divide the data set into 35 epochs containing 10,000 examples each. Over 10 trials, we include an example in the evaluation data with probability 0.5 (on average, 5,000 examples per epoch). This results in 680,000 features and an average of 175,000 examples per run. The default representation contains trigram counts, which were normalized so that each example had unit length.

Results (Fig. 5) are similar to those on the URL data: CW-fact outperforms CW-diag significantly (18% relative). Again, this data set has more features than examples and various sets of features are correlated, for example trigrams with shared bigrams or unigrams.

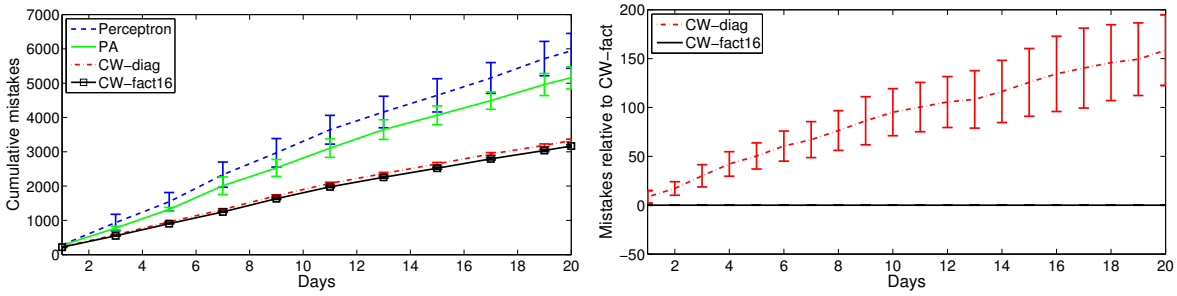
4.3 Stock market data

Given computational resource constraints, the choice of learning method effectively determines the size of the feature set that can be used in practice. CW-diag allows very large sets, while CW-full imposes relatively harsh limits. We show here, using a stock market prediction task, that CW-fact offers an advantageous middle ground: performance losses from approximating the covariance matrix can be compensated by improvements from the use of a more informative feature set, giving the highest overall performance with limited resources.

The task is to predict whether the price of a target stock went up or down each day, based on the open, high, low, and close prices of a set of predictor stocks for that day and the preceding 49 days (200 features per predictor). The feature set is thus highly correlated. Our data spans a 15-year period from 1994 to 2009, for a total of 4006 instances (2120 up days and 1886 down days). We use DELL as a target, although other tested targets showed similar results. The number of predictor stocks is variable, offering a tradeoff between speed and accuracy.

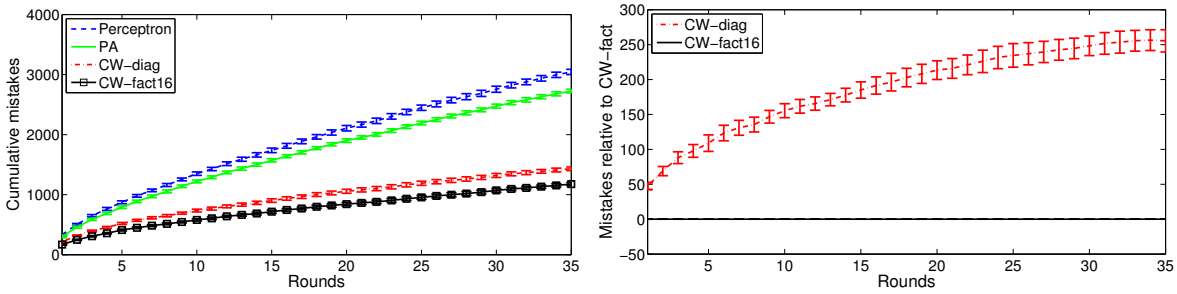
Table 2 shows the number of mistakes made by each method for a range of predictor set sizes, subject to a memory constraint of 2GB and a time constraint of two hours. Each result is the mean over 10 random draws of the predictor stocks from our collection of 378, excluding DELL.

CW-diag handles all feature set sizes, but is unable to extract useful signal; note that perceptron and PA returned similarly poor results on these data (not shown). CW-full extracts the most performance from each feature set, but incurs large computational costs, requiring 54 minutes and 200MB of memory for the 25-predictor test. Completing the 100-predictor test would have required over 3GB of memory and more



(a) Mistake count (b) Relative mistake count

Figure 4: URL Data: (a) Mistakes made by each algorithm over 20 days. (b) The relative number of mistakes between CW-diag and CW-fact. Error bars are at one standard deviation.



(a) Mistake count (b) Relative mistake count

Figure 5: Web Spam Data: (a) Mistakes made by each algorithm (10k examples per epoch.) (b) The relative number of mistakes between CW-diag and CW-fact. Error bars are at one standard deviation.

Table 2: Number of mistakes on stock market data. Values are omitted for any test that required more than 2GB of memory (†) or two hours of runtime (∗). All differences of at least 44 mistakes are statistically significant at $p = 0.01$ based on a paired t-test.

Predictors	CW-diag	CW-fact8	CW-full
1	2141.0	2083.1	2028.3
10	2142.0	2036.8	1891.4
25	2142.0	1949.3	1806.3
100	2142.0	1779.0	†∗
250	2142.0	1749.9	†∗
378	2142.0	∗	†∗

than 10 hours. CW-fact, on the other hand, requires only 38MB of memory and 63 minutes to run with 250 predictors, and achieves the overall best result.

4.4 Document Classification

Finally, we evaluate CW-fact on the `Reuters`, `20 Newsgroups` and `Sentiment` document classification data sets used earlier to evaluate CW learning (Dredze et al., 2008). For each data set, we performed 10 runs of the experiment where we randomized the order of the examples. The results in Table 3 show that CW-fact consistently improves over CW-diag.

5 Conclusion and Related Work

We examined the effect of covariance matrix representation on confidence-weighted learning, but we believe our results have the potential to generalize to other second-order online learning algorithms. Depending on properties of the data, full covariance or an approximate covariance matrix obtained by factored representations may improve on the more efficient diagonal covariance version of CW learning.

A desire for compact representations of second-order information arises in contexts outside of our own work. For instance, in the limited memory BFGS method (L-BFGS) for quasi-Newton algorithms, the Hessian is computed based on the last m updates (Liu & Nocedal, 1989). This is similar in spirit to buffering the last m updates as described in Section 3.2. Other techniques such as Kronecker factorization and incomplete Cholesky factorization have been explored in the context of approximating kernel matrices for support-vector machine training (Wu et al., 2006).

In synthetic experiments and large-scale applications, we showed that full and factored representations performed better than diagonal when there were many correlated features and the effective dimensionality of the data was small. Conversely, we also showed that

Table 3: Document Classification: Average error rates (%) and standard deviations for perceptron, PA, CW-diag and CW-fact8 on `Reuters`, `20 Newsgroups` and `Sentiment` binary classification tasks.

Task	Examples	Features	Perceptron	PA	CW-diag	CW-fact8
Reuters						
business	2000	12167	21.9 ± 0.7	19.0 ± 0.5	18.7 ± 0.6	**18.1** ± 0.5
insurance	2000	9094	17.8 ± 0.5	15.5 ± 0.8	13.7 ± 0.6	**12.7** ± 0.5
retail	2000	8768	26.4 ± 1.0	24.3 ± 0.6	20.9 ± 0.7	**18.4** ± 0.5
20 Newsgroups						
comp	1943	29409	27.0 ± 5.6	19.6 ± 0.9	13.6 ± 0.5	**12.3** ± 0.6
sci	1971	38699	21.9 ± 4.3	13.6 ± 0.7	8.1 ± 0.4	**7.1** ± 0.4
talk	1850	44397	23.2 ± 5.4	8.6 ± 0.8	4.7 ± 0.3	**3.7** ± 0.4
Sentiment						
apparel	1940	63088	22.7 ± 0.6	18.9 ± 0.5	17.5 ± 0.5	**16.8** ± 0.4
books	18391	1042928	18.6 ± 0.3	16.8 ± 0.1	14.7 ± 0.1	**14.3** ± 0.1
dvd	13152	850348	20.2 ± 0.4	18.0 ± 0.2	15.8 ± 0.2	**15.4** ± 0.2
electronics	5774	249863	20.7 ± 0.6	18.0 ± 0.5	16.1 ± 0.3	**15.5** ± 0.3
kitchen	5212	193467	20.4 ± 0.4	17.2 ± 0.2	15.3 ± 0.2	**14.8** ± 0.2
music	12927	666927	20.5 ± 0.3	18.7 ± 0.2	16.3 ± 0.1	**15.7** ± 0.2
video	4349	346659	25.0 ± 0.4	22.1 ± 0.3	20.1 ± 0.4	**19.4** ± 0.4

full methods performed worse when the data was noisy or had fewer correlations between features.

Acknowledgments: We thank the reviewers for valuable feedback. Koby Crammer is a Horev Fellow, supported by the Taub Foundations. This work was also supported by National Science Foundation grants NSF-0238323, NSF-0433668 and NSF-0829469 and by generous research, operational and in-kind support from Cisco, Google, Microsoft, Yahoo and the UCSD Center for Networked Systems.

References

Bottou, L. (1998). Online Learning and Stochastic Approximations. In *Online Learning and Neural Networks*, 9–42. Cambridge, UK: Cambridge University Press.

Cesa-Bianchi, N., Conconi, A., & Gentile, C. (2005). A Second-Order Perceptron Algorithm. *SIAM Journal on Computing*, *34*, 640–668.

Crammer, K., Dekel, O., Shalev-Shwartz, S., & Singer, Y. (2006). Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, *7*, 551–585.

Crammer, K., Dredze, M., & Pereira, F. (2009a). Exact Convex Confidence-Weighted Learning. *Advances in Neural Information Processing Systems 21* (pp. 345–352).

Crammer, K., Kulesza, A., & Dredze, M. (2009b). Adaptive Regularization of Weight Vectors. *Advances in Neural Information Processing Systems 22* (pp. 414–422).

Dredze, M., & Crammer, K. (2008). Online Methods for Multi-Domain Learning and Adaptation. *Empirical Methods in Natural Language Processing (EMNLP)*.

Dredze, M., Crammer, K., & Pereira, F. (2008). Confidence-Weighted Linear Classification. *Proceedings of the International Conference on Marchine Learning (ICML)* (pp. 264–271). Helsinki, Finland: Omnipress.

Gorsuch, R. L. (1983). *Factor Analysis*. New York, NY: Lawrence Erlbaum Associates. 2nd edition.

Jaakkola, T. S., & Jordan, M. I. (2000). Bayesian Parameter Estimation via Variational Methods. *Statistics and Computing*, *10*, 25–37.

Liu, D. C., & Nocedal, J. (1989). On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming*, *45*, 503–528.

Ma, J., Saul, L. K., Savage, S., & Voelker, G. M. (2009). Identifying Suspicious URLs: An Application of Large-Scale Online Learning. *Proc. of the International Conference on Machine Learning (ICML)* (pp. 681–688). Montreal, Quebec.

MacKay, D. J. C. (1992). The Evidence Framework Applied to Classification Networks. *Neural Computation*, *4*, 720–736.

Nocedal, J., & Wright, S. (1999). *Numerical Optimization*. Springer.

Roweis, S. (1998). EM Algorithms for PCA and SPCA. In M. I. Jordan, M. J. Kearns and S. A. Solla (Eds.), *Advances in Neural Information Processing Systems 10*, 626–632. Cambridge, MA: MIT Press.

Sonnenburg, S., Franc, V., Yom-Tov, E., & Sebag, M. (2008). PASCAL Large Scale Learning Challenge. http://largescale.first.fraunhofer.de/workshop/.

Spiegelhalter, D. J., & Lauritzen, S. L. (1990). Sequential Updating of Conditional Probabilities on Directed Graphical Structures. *Networks*, *20*, 579–605.

Wu, G., Chang, E., Chen, Y.-K., & Hughes, C. (2006). Incremental Approximate Matrix Factorization for Speeding up Support Vector Machines. *Proceedings of the ACM SIGKDD, International Conference on Knowledge Discovery and Data Mining (KDD)*. Philadelphia, PA.