Journal of Machine Learning Research 8 (2007) 227-248 Submitted 11/05; Revised 10/06; Published 2/07

Noise Tolerant Variants of the Perceptron Algorithm

Roni Khardon RONI@CS.TUFTS.EDU
Gabriel Wachman GWACHM01@CS.TUFTS.EDU
Department of Computer Science, Tufts University
Medford, MA 02155, USA

Editor: Michael J. Collins

Abstract

A large number of variants of the Perceptron algorithm have been proposed and partially evaluated in recent work. One type of algorithm aims for noise tolerance by replacing the last hypothesis of the perceptron with another hypothesis or a vote among hypotheses. Another type simply adds a margin term to the perceptron in order to increase robustness and accuracy, as done in support vector machines. A third type borrows further from support vector machines and constrains the update function of the perceptron in ways that mimic soft-margin techniques. The performance of these algorithms, and the potential for combining different techniques, has not been studied in depth. This paper provides such an experimental study and reveals some interesting facts about the algorithms. In particular the perceptron with margin is an effective method for tolerating noise and stabilizing the algorithm. This is surprising since the margin in itself is not designed or used for noise tolerance, and there are no known guarantees for such performance. In most cases, similar performance is obtained by the voted-perceptron which has the advantage that it does not require parameter selection. Techniques using soft margin ideas are run-time intensive and do not give additional performance benefits. The results also highlight the difficulty with automatic parameter selection which is required with some of these variants.

Keywords: perceptron algorithm, on-line learning, noise tolerance, kernel methods

1. Introduction

The success of support vector machines (SVM) (Boser et al., 1992; Cristianini and Shawe-Taylor, 2000) has led to increasing interest in the perceptron algorithm. Like SVM, the perceptron algorithm has a linear threshold hypothesis and can be used with kernels, but unlike SVM, it is simple and efficient. Interestingly, despite a large number of theoretical developments, there is no result that explains why SVM performs better than perceptron, and similar convergence bounds exist for both (Graepel et al., 2000; Cesa-Bianchi et al., 2004). In practice, SVM is often observed to perform slightly better with significant cost in run time. Several on-line algorithms have been proposed which iteratively construct large margin hypotheses in the feature space, and therefore combine the advantages of large margin hypotheses with the efficiency of the perceptron algorithm. Other variants adapt the on-line algorithms to work in a batch setting choosing a more robust hypothesis to be used instead of the last hypothesis from the on-line session. There is no clear study in the literature, however, that compares the performance of these variants or the possibility of combining them to obtain further performance improvements. We believe that this is important since these algorithms have already been used in applications with large data sets (e.g., Collins, 2002; Li et al., 2002) and a better understanding of what works and when can have a direct implication for future

use. This paper provides such an experimental study where we focus on noisy data and more generally the "unrealizable case" where the data is simply not linearly separable. We chose some of the basic variants and experimented with them to explore their performance both with hindsight knowledge and in a statistically robust setting.

More concretely, we study two families of variants. The first explicitly uses the idea of hard and soft margin from SVM. The basic perceptron algorithm is mistake driven, that is, it only updates the hypothesis when it makes a mistake on the current example. The perceptron algorithm with margin (Krauth and Mézard, 1987; Li et al., 2002) forces the hypothesis to have some margin by making updates even when it does not make a mistake but where the margin is too small. Adding to this idea, one can mimic soft-margin versions of support vector machines within the perceptron algorithm that allow it to tolerate noisy data (e.g., Li et al., 2002; Kowalczyk et al., 2001). The algorithms that arise from this idea constrain the update function of the perceptron and limit the effect of any single example on the final hypothesis. A number of other variants in this family exist in the literature. Each of these performs margin based updates and has other small differences motivated by various considerations. We discuss these further in the concluding section of the paper.

The second family of variants tackles the use of on-line learning algorithms in a batch setting, where one trains the algorithm on a data set and tests its performance on a separate test set. In this case, since updates do not always improve the error rate of the hypothesis (e.g., in the noisy setting), the final hypothesis from the on-line session may not be the best to use. In particular, the longest survivor variant (Kearns et al., 1987; Gallant, 1990) picks the "best" hypothesis on the sequential training set. The voted perceptron variant (Freund and Schapire, 1999) takes a vote among hypotheses produced during training. Both of these have theoretical guarantees in the PAC learning model (Valiant, 1984). Again, other variants exist in the literature which modify the notion of "best" or the voting scheme among hypotheses and these are discussed in the concluding section of the paper.

It is clear that each member of the first family can be combined with each member of the second. In this paper we report on experiments with a large number of such variants that arise when combining some of margin, soft margin, and on-line to batch conversions. In addition to real world data, we used artificial data to check the performance in idealized situations across a spectrum of data types.

The experiments lead to the following conclusions: First, the perceptron with margin is the most successful variant. This is surprising since among the algorithms experimented with it is only one not designed for noise tolerance. Second, the soft-margin variants on their own are weaker than the perceptron with margin, and combining soft-margin with the regular margin variant does not provide additional improvements.

The third conclusion is that in most cases the voted perceptron performs similarly to the perceptron with margin. The voted perceptron has the advantage that it does not require parameter selection (for the margin) that can be costly in terms of run time. Combining the two to get the voted perceptron with margin has the potential for further improvements but this occasionally degrades performance. Finally, both the voted perceptron and the margin variant reduce the deviation in accuracy in addition to improving the accuracy. This is an important property that adds to the stability and robustness of the algorithms.

The rest of the paper is organized as follows. The next section reviews all the algorithms and our basic settings for them. Section 3 describes the experimental evaluation. We performed two kinds of experiments. In "parameter search" we report the best results obtained with any parameter setting.

Input set of examples and their labels $\mathcal{Z} = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathbb{R}^n \times \{-1, 1\})^m$, η, θ_{Init}, C

- Initialize $w \leftarrow 0^n$ and $\theta \leftarrow \theta_{Init}$
- for every training epoch:
- for every $x_j \in \mathcal{X}$:
  - $\hat{y} \leftarrow sign(\langle w, x_j \rangle - \theta)$
  - if $(\hat{y} \neq y_j)$
    - $w \leftarrow w + \eta y_j x_j$
    - $\theta \leftarrow \theta + \eta y_j C$

Figure 1: The Basic Perceptron Algorithm

This helps set the scope and evaluate the potential of different algorithms to improve performance and provides insight about their performance, but it does not give statistically reliable results. In "parameter optimization" the algorithms automatically select the parameters and the performance can be interpreted statistically. The concluding section further discusses the results and puts related work in their context leading to directions for future work.

2. Algorithms

This section provides details of all the algorithms used in the experiments and our basic settings for them.

2.1 The Perceptron Algorithm

The perceptron algorithm (Rosenblatt, 1958) takes as input a set of training examples in $\mathbb{R}^n$ with labels in $\{-1, 1\}$. Using a weight vector, $w \in \mathbb{R}^n$, initialized to 0^n, and a threshold, θ, it predicts the label of each training example x to be $y = sign(\langle w, x \rangle - \theta)$. The algorithm adjusts w and θ on each misclassified example by an additive factor. The algorithm is summarized in Figure 1 following the form by Cristianini and Shawe-Taylor (2000). Unlike Cristianini and Shawe-Taylor (2000), we allow for a non-zero value of θ_{Init} and we allow C to be initialized to any value.

The "learning rate" η controls the extent to which w can change on a single update. Note that if θ is initialized to 0 then η has no effect since $sign(\langle w, x_i \rangle - \theta) = y_i$ iff $sign(\eta(\langle w, x_i \rangle - \theta)) = y_i$. However, the initial choice of θ is important as it can be significant in early iterations of the algorithm. Note also that there is only one "degree of freedom" between η and θ_{Init} since scaling both by the same factor does not alter the result. So one could fix $\eta = 1$ and vary only θ_{Init}. But we keep the general form for convenience.

When the data are linearly separable via some hyperplane (w, θ), the margin is defined as $\gamma = \min_{1 \leq i \leq m}(y_i(\langle w, x_i \rangle - \theta))$. When $\|w\| = 1$, γ is the minimum Euclidean distance of any point in the data set to (w, θ). If the data are linearly separable, and θ is initialized to 0, the perceptron algorithm

is guaranteed to converge in $\leq (\frac{R}{\gamma})^2$ iterations (Novikoff, 1962; Cristianini and Shawe-Taylor, 2000), where $R = \max_{1 \leq i \leq m} \|x_i\|$.

In the case of non-separable data, the extent to which a data point fails to have margin γ via the hyperplane w can be quantified by a slack variable $\xi_i = \max(0, \gamma - y_i(\langle w, x_i \rangle + \theta))$. Observe that when $\xi_i = 0$, the example x_i has margin at least γ via the hyperplane defined by (w, θ). The perceptron is guaranteed to make no more than $(\frac{2(R+D)}{\gamma})^2$ mistakes on m examples, for any $w, \gamma > 0$ where $D = \sqrt{\sum_{i=1}^{m} \xi_i^2}$ (Freund and Schapire, 1999; Shalev-Shwartz and Singer, 2005). Thus one can expect some robustness to noise.

It is well known that the perceptron can be re-written in "dual form" whereby it can be used with kernel functions (Cristianini and Shawe-Taylor, 2000). The dual form of the perceptron is obtained by observing that the weight vector can be written as $w = \sum_{i=1}^{m} \eta \alpha_i y_i x_i$ where α_i is the number of mistakes that have been made on example i. Subsequently, a new example x_j is classified according to $sign(SUM - \theta)$ where $SUM = \sum_i \eta \alpha_i y_i \langle x_i, x_j \rangle$. Replacing the inner product with a kernel function yields the kernel perceptron. All the perceptron variants we discuss below have dual form representations. However, the focus of this paper is on noise tolerance and this is independent of the use of primal or dual forms and the use of kernels. We therefore present all the algorithms in primal form.

2.2 The λ-trick

The λ-trick (Kowalczyk et al., 2001; Li et al., 2002) attempts to minimize the effect of noisy examples during training similar to the L_2 soft margin technique used with support vector machines (Cristianini and Shawe-Taylor, 2000). We classify example x_j according to $sign(SUM - \theta)$ where

$$SUM = \langle w, x_j \rangle + I_j y_j \lambda \|x_j\|^2 \tag{1}$$

and where I_j is an indicator variable such that

$$I_j = \begin{cases} 1 & \text{if } x_j \text{ was previously classified incorrectly} \\ 0 & \text{otherwise} \end{cases}.$$

Thus during training, if a mistake has been made on x_j then in future iterations we increase SUM artificially by a factor of $\lambda \|x_j\|^2$ when classifying x_j but not when classifying other examples. A high value of λ can make the term $y_j \lambda \|x_j\|^2$ dominate the sum and make sure that x_j does not lead to an update, so it is effectively ignored. In this way λ can prevent large number of updates on noisy (as well as other) examples limiting their effect.

We note that this technique is typically presented using an additive λ instead of $\lambda \|x_j\|^2$. While there is no fundamental difference, adding $\lambda \|x_j\|^2$ is more convenient in the experiments since it allows us to use the same values of λ for all data sets (since the added term is scaled relative to the data).

2.3 The α-bound

This variant is motivated by the L_1 soft margin technique used with support vector machines (Cristianini and Shawe-Taylor, 2000). The α-*bound* places a bound α on the number of mistakes the algorithm can make on a particular example, such that when the algorithm makes a mistake on some x_i, it does not update w if more than α mistakes have already been made on x_i. As in the case

of the λ-trick, the idea behind this procedure is to limit the influence of any particular noisy example on the hypothesis. Intuitively, a good setting for α is some fraction of the number of training iterations. To see that, assume that the algorithm has made α mistakes on a particular noisy example x_k. In all subsequent training iterations, x_k never forces an update, whereas the algorithm may continue to increase the influence of other non-noisy examples. If the ratio of non-noisy examples to noisy examples is high enough the algorithm should be able to bound the effect of noisy examples in the early training iterations while leaving sufficient un-bounded non-noisy examples to form a good hypothesis in subsequent iterations. While this is a natural variant, we are not aware of any experimental results using it. Note that in order for the α-bound to work effectively, the algorithm must perform a high enough number of training iterations.

2.4 Perceptron Using Margins

The classical perceptron attempts to separate the data but has no guarantees on the separation margin obtained. The Perceptron Algorithm using Margins (PAM) (Krauth and Mézard, 1987) attempts to establish such a margin, τ, during the training process. Following work on support vector machines (Boser et al., 1992; Cristianini and Shawe-Taylor, 2000) one may expect that providing the perceptron with higher margin will add to the stability and accuracy of the hypothesis produced.

To establish the margin, instead of only updating on examples for which the classifier makes a mistake, PAM updates on x_j if

$$y_j(SUM - \theta) < \tau$$

where SUM is as in Equation (1). Notice that this includes the case of a mistake where $y_j(SUM - \theta) < 0$ and the case of correct classification with low margin when $0 \leq y_j(SUM - \theta) < \tau$. In this way, the algorithm "establishes" some minimal separation of the data. Notice that the weight vector *w* is not normalized during the learning process and its norm can increase with updates. Therefore, the constraint imposed by τ becomes less restrictive as learning progresses, and the normalized margin is not τ. When the data are linearly separable and $\tau < \gamma_{opt}$, PAM finds a separating hyperplane with a margin that is guaranteed to be at least $\gamma_{opt} \frac{\tau}{\sqrt{8(\eta R^2 + \tau)}}$, where γ_{opt} is the maximum possible margin (Krauth and Mézard, 1987; Li et al., 2002).[1]

As above, it is convenient to make the margin parameter data set independent so we can use the same values across data sets. To facilitate this we replace the above and update if

$$y_j(SUM - \theta) < \tau\theta_{Init}$$

so that τ can be thought of as measured in units of θ_{Init}.

A variant of PAM exists in which a different value of τ is chosen for positive examples than for negative examples (Li et al., 2002). This seems to be important when the class distribution is skewed. We do not study that variant in this paper.

2.5 Longest Survivor and Voted Perceptron

The classical perceptron returns the last weight vector *w*, that is, the one obtained after all training has been completed, but this may not always be useful especially if the data is noisy. This is a general issue that has been studied in the context of using on-line algorithms that expect one example at a

1. Other variants (e.g., Gentile, 2001; Tsampouka and Shawe-Taylor, 2005; Shalev-Shwartz and Singer, 2005) do normalize the weight vector and thus have better margin guarantees.

time in a batch setting where a set of examples is given for training and one hypothesis is used at the end to classify all future instances. Several variants exist that handle this issue. In particular Kearns et al. (1987) show that *longest survivor* hypothesis, that is, the one who made the largest number of correct predictions during training in the on-line setting, is a good choice in the sense that one can provide guarantees on its performance in the PAC learning model (Valiant, 1984). Several variations of this idea were independently proposed under the name of the *pocket algorithm* and empirical evidence for their usefulness was provided (Gallant, 1990).

The voted perceptron (Freund and Schapire, 1999) assigns each vector a "vote" based on the number of sequential correct classifications by that weight vector. Whenever an example is misclassified, the voted perceptron records the number of correct classifications made since the previous misclassification, assigns this number to the current weight vector's vote, saves the current weight vector, and then updates as normal. After training, all the saved vectors are used to classify future examples and their classifications are combined using their votes. At first sight the voted-perceptron seems to require expensive calculation for prediction. But as pointed out by Freund and Schapire (1999), the output of the weight vector resulting from the first k mistakes can be calculated from the output of the weight vector resulting from the first $k-1$ mistakes in constant time. So when using the dual perceptron the prediction phase of the voted perceptron is not substantially more expensive than the prediction phase of the classical perceptron, although it is more expensive in the primal form.

When the data are linearly separable and given enough iterations, both these variants will converge to a hypothesis that is very close to the simple perceptron algorithm. The last hypothesis will predict correctly on all examples and indeed its vote will be the largest vote among all hypotheses used. When the data are not linearly separable the quality of hypothesis may fluctuate during training as noisy examples are encountered.

2.6 Algorithms Summary

Figure 2 summarizes the various algorithms and prediction strategies in primal form. The algorithms have corresponding dual forms and kernelized versions that we address in the experimental sections.

2.7 Multi-Class Data

Some of the data sets we experiment with have more than two labels. For these we used a standard solution as follows. For each label l, we train a classifier to separate examples labeled l (the positive examples) from other examples (all of which become negative examples). This can be done on-line where we train all the classifiers "in parallel". During testing we calculate the weight given by each classifier (before the sign function is applied) and choose the one maximizing the weight.

3. Experimental Evaluation

We ran two sets of experiments with the algorithms described above. In one set of experiments we searched through a pre-determined set of values of τ, α, and λ by running each of the classical, longest survivor, and voted perceptron using 10-fold cross-validation and parameterized by each value of τ, α, and λ as well as each combination of $\tau \times \alpha$ and $\tau \times \lambda$. This first set of experiments is called *parameter search*. The purpose of the parameter search experiments was to give us a comprehensive view of the effects of the various parameters. This can show whether a method has any

TRAINING: Input set of examples and their labels
$\mathcal{Z} = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathbb{R}^n \times \{-1, 1\})^m$, θ_{Init}, η, C, α-bound, λ, τ

- Initialize $\alpha \leftarrow 0^m, k \leftarrow 0, w_0 \leftarrow 0^n$, $tally \leftarrow 0$, $best\_tally \leftarrow 0$, $\theta_0 \leftarrow \theta_{Init}, \theta_{l.s.} \leftarrow \theta_{Init}$
- for every training iteration
- for every $z_j \in \mathcal{Z}$:
  - $SUM \leftarrow \langle w_k, x_j \rangle + I_{(\alpha_j \neq 0)}\ y_j \lambda \|x_j\|^2$
  - Predict:
    - if $SUM < \theta_k - \tau$ then $\hat{y} = -1$
    - else if $SUM > \theta_k + \tau$ then $\hat{y} = 1$
    - else $\hat{y} = 0$ // This forces an update
  - if $(\hat{y} \neq y_j)$ AND $(\alpha_j < \alpha\text{-bound})$
    - $w_{k+1} \leftarrow w_k + \eta y_j x_j$
    - $\alpha_j \leftarrow \alpha_j + 1$
    - Update ''mistakes'' data structure
    - $\theta_{k+1} \leftarrow \theta_k + \eta y_j C$
    - $vote_{k+1} \leftarrow 0$
    - $k \leftarrow k + 1$
    - $tally \leftarrow 0$
  - else if $(\hat{y} = y_j)$
    - $vote_k \leftarrow vote_k + 1$
    - $tally \leftarrow tally + 1$
    - if $(tally > best\_tally)$
      - $best\_tally \leftarrow tally$
      - $w_{l.s.} \leftarrow w_k$
      - $\theta_{l.s.} \leftarrow \theta_k$

PREDICTION: To predict on a new example x_{m+1}:

- CLASSICAL:
  - $SUM \leftarrow \langle w_k, x_{m+1} \rangle$
  - $\hat{y} \leftarrow sign(SUM - \theta_k)$
- LONGEST SURVIVOR:
  - $SUM \leftarrow \langle w_{l.s.}, x_{m+1} \rangle$
  - $\hat{y} \leftarrow sign(SUM - \theta_{l.s.})$
- VOTED:
  - for $i \leftarrow 1$ to k
    - $SUM \leftarrow \langle w_i, x_{m+1} \rangle$
    - $\hat{y} \leftarrow sign(SUM - \theta_i)$
    - VOTES $\leftarrow$ VOTES $+ vote_i\ \hat{y}$
  - $\hat{y}_{final} \leftarrow sign(\text{VOTES})$

Figure 2: Illustration of All Algorithm Variants

chance of improving performance since the experiments give us hindsight knowledge. The experiments can also show general patterns and trends in the parameter landscape again giving insight into the performance of the methods. Notice that parameter search cannot be used to indicate good values of parameters as this would be hand-tuning the algorithm based on the test data. However, it can guide in developing methods for automatic selection of parameters.

In the second set of experiments, we used the same set of values as in the first experiment, but using a method for automatic selection of parameters. In particular we used a "double cross validation" where in each fold of the cross validation (1) one uses parameter search on the training data only using another level of cross validation, (2) picks values of parameters based on this search, (3) trains on the complete training set for the fold using these values, and (4) evaluates on the test set. We refer to this second set of experiments as *parameter optimization*. This method is expensive to run as it requires running all combinations of parameter values in each internal fold of the outer cross validation. So if both validations use 10 folds then we run the algorithm 100 times per parameter setting. This sets a strong limitation on the number of parameter variations that can be tried. Nonetheless this is a rigorous method of parameter selection.

Both sets of experiments were run on randomly-generated artificial data and real-world data from the University of California at Irvine (UCI) repository and other sources. The purpose of the artificial data was to simulate an ideal environment that would accentuate the strengths and weaknesses of each algorithm variant. The other data sets explore the extent to which this behavior is exhibited in real world problems.

For further comparison, we also ran SVM on the data sets using the L_1-norm soft margin and L_2-norm soft margin. We used SVM Light (Joachims, 1999) and only ran *parameter search*.

3.1 Data Set Selection and Generation

We have experimented with 10 real world data sets of varying size and 150 artificial data sets composed of 600 examples each.

We first motivate the nature of artificial data used by discussing the following four idealized scenarios. The simplest, type 1, is linearly separable data with very small margin. Type 2 is linearly separable data with a larger, user-defined margin. For type 3 we first generate data as in type 1, and then add random class noise by randomly reversing the labels of a given fraction of examples. For type 4, we generate data as in type 2, and then add random class noise. One might expect that the basic perceptron algorithm will do fine on type 1 data, the perceptron with margin will do particularly well on type 2 data, that noise tolerant variants without margin will do well on type 3, and that some combination of noise tolerant variant with margin will be best for type 4 data. However, our experiments show that the picture is more complex; preliminary experiments confirmed that the expected behavior is observed, except that the perceptron with margin performed well on data of type 3 as well, that is, when the "natural margin" was small and the data was not separable due to noise. In the following, we report on experiments with artificial data where the margin and noise levels are varied so we effectively span all four different types.

Concretely the data was generated as follows. We first specify parameters for number of features, noise rate and the required margin size. Given these, each example x_i is generated independently and each attribute in an example is generated independently using an integer value in the range $[-10,10]$. We generated the weight vector, w, by flipping a coin for each example and adding it to the weight vector on a head. We chose θ as the average of $\frac{1}{2}|\langle w,x_i\rangle|$ over all x_i. We

Name	# features*	# examples	Baseline
Adult	105	32561	75.9
Breast-cancer-wisconsin	9	699	65.5
Bupa	6	345	58
USPS	256	9298	N/A
Wdbc	30	569	62.7
Crx	46	690	55.5
Ionosphere	34	351	64.4
Wpbc	33	198	76.3
sonar.all-data	60	208	53.4
MNIST	784	70,000	N/A
*after preprocessing			

Table 1: UCI and Other Natural Data Set Characteristics

measured the actual margin of the examples with respect to the weight vector and then discarded any examples x_i such that $|\langle w, x_i\rangle - \theta| < M * \theta$ where M is the margin parameter specified. For the noisy settings, for each example we randomly switched the label with probability equal to the desired noise rate. In the tables of results presented below, f stands for number of features, M stands for the margin size, and N stands for the noise rate. We generated data sets with parameters $(f, M, N) \in \{50, 200, 500\} \times \{0.05, 0.1, 0.25, 0.5, 0.75\} \times \{0, 0.05, 0.1, 0.15, 0.25\}$, and for each parameter setting we generated two data sets, for a total 150 data sets.

For real world data we first selected two-class data sets from the UCI Machine Learning Repository (Newman et al., 1998) that have been used in recent comparative studies or in recent papers on linear classifiers (Cohen, 1995; Dietterich et al., 1996; Dietterich, 2000; Garg and Roth, 2003). Since we require numerical attributes, any nominal attribute in these data sets was translated to a set of binary attributes each being an indicator function for one of the values. Since all these data sets have a relatively small number of examples we added three larger data sets to strengthen statistically our conclusions: "Adult" from UCI (Newman et al., 1998), and "MNIST[2]" and "USPS[3]," the 10-class character recognition data sets. Due to their size, however, for the "USPS," "MNIST," and "Adult" data sets, there is no outer cross-validation in *parameter optimization*. For ease of comparison to other published results, we use the $7291/2007$ training/test split for "USPS", and a $60000/10000$ split for "MNIST".

The data sets used and their characteristics after the nominal-to-binary feature transformation are summarized in Table 1. The column labeled "Baseline" indicates the percentage of the majority class.

3.2 Exploratory Experiments and General Setup

The algorithms described above have several parameters that can affect their performance dramatically. In this paper we are particularly interested in studying the effect of parameters related to noise tolerance, and therefore we fix the value of θ_{Init}, η, C, and the number of training iterations. Prior to

2. http://yann.lecun.com/exdb/mnist/.
3. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html.

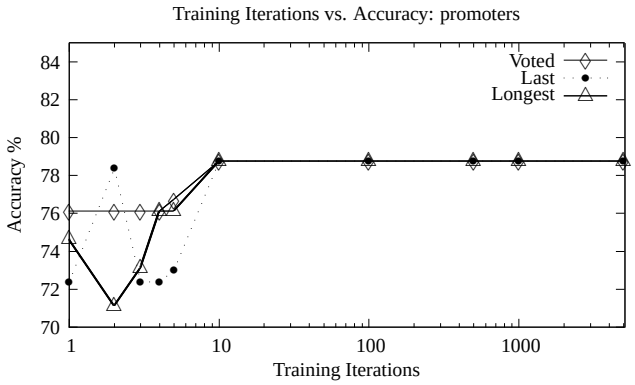

Figure 3: Example Learning Curve for UCI Data Set

fixing these values, we ran preliminary experiments in which we varied these parameters. In addition we ran experiments where we normalized the example vectors. The results showed that while the performance of the algorithms overall was different in these settings, the relationship between the performance of individual algorithms seems to be stable across these variations.

We set $\theta_{Init} = avg(\langle x_i, x_i \rangle)$, initializing the threshold at the same scale of inner products. Combined with a choice of $\eta = 0.1$, this makes sure that a few iterations should be able to guarantee that an example is classified correctly given no other changes to the hypothesis. We also set $C = avg(\langle x_i, x_i \rangle) = \theta_{Init}$. This is similar to the version analyzed by Cristianini and Shawe-Taylor (2000) except that we use the average instead of maximum norm. Using this setting we essentially maintain θ in units of θ_{Init}.

As explained above, the number of iterations must be sufficiently high to allow the α parameter to be effective, as well as to allow the weight vector to achieve some measure of stability. Typically a small number of iterations is sufficient, and preliminary experiments illustrated by the curve in Figure 3 showed that 100 iterations are more than enough to allow all the algorithms to converge to a stable error rate. Therefore, except where noted below, we report results for 100 iterations.

For the large data sets, we reduced the number of training iterations and increased η accordingly in order to run the experiments in a reasonable amount of time; for "Adult" we set $\eta = 0.4$ with 5 training iterations and for "MNIST" we set $\eta = 1$ with 1 training iteration. For "USPS," we used $\eta = 0.1$ and 100 iterations for *parameter search*, and $\eta = 0.1$ and 10 iterations for *parameter optimization*.

The parameters of interest in our experiments are τ, α, λ that control the margin and soft margin variants. Notice that we presented these so that their values can be fixed in a way that is data set independent. We used values for $\tau \in \{0, 0.125, 0.25, 0.5, 1, 2, 4\}$, $\alpha \in \{\infty, 80, 60, 40, 20, 10, 5\}$, and $\lambda \in \{0, 0.125, 0.25, 0.5, 1, 2, 4\}$. Notice that the values as listed from left to right vary from no effect to a strong effect for each parameter. We ran *parameter search* and *parameter optimization* over all of these values, as well as all combinations $\tau \times \lambda$ and $\tau \times \alpha$. Since *parameter optimization* over combined values is particularly expensive we have also experimented with a variant that first searches for a good τ value and then searches for a value of α (denoted $\tau \rightarrow \alpha$) and a variant that does likewise with τ and λ (denoted $\tau \rightarrow \lambda$).

We did not perform any experiments involving α on "MNIST" or "Adult" as their size required too few iterations to justify any reasonable α-bound.

	V > C	V > LS	LS > C	LS > V	C > LS	C > V	V = LS = C
Noise = 0	0	0	0	0	0	0	30
Noise = 0.05	12	10	11	3	0	2	16
Noise = 0.1	15	15	14	3	3	3	12
Noise = 0.15	16	14	13	5	6	3	10
Noise = 0.25	16	12	13	8	7	4	10

Table 2: Noise Percentage vs. Dominance: V = Voted, C = Classical, LS = Longest Survivor

Finally we performed a comparison of the classical perceptron, the longest survivor and the voted perceptron algorithms without the additional variants. Table 2 shows a comparison of the accuracies obtained by the algorithms over the artificial data. We ignore actual values but only report whether one algorithm gives higher accuracy or whether they tie (the variance in actual results is quite large). One can see that with higher noise rate the voted perceptron and longest survivor improve the performance over the base algorithm. Over all 150 artificial data sets, the voted perceptron strictly improves performance over the classical perceptron in 59 cases, and ties or improves in 138 cases. Using the distribution over our artificial data sets one can calculate a simple weak statistical test that supports the hypothesis that using the voted algorithm does not hurt accuracy, and can often improve it.

Except where noted, all the results reported below give average accuracy in 10-fold cross-validation experiments. To avoid any ordering effects of the data, the training set is randomly permuted before running the experiments.

3.3 Parameter Search

The *parameter search* experiments reveal several interesting aspects. We observe that in general the variants are indeed helpful on the artificial data since the performance increases substantially from the basic version. The numerical results are shown in Tables 3 and 4 and discussed below. Before showing these we discuss the effects of single parameters. Figure 4 plots the effect of single parameters for several data sets. For the artificial data set shown, the accuracy is reasonably well-behaved with respect to the parameters and good performance is obtained in a non negligible region; this is typical of the results from the artificial data. Data obtained for the real world data sets show somewhat different characteristics. In some data sets little improvement is obtained with any variant or parameter setting. In others, improvement was obtained for some parameter values but the regions were not as large implying that that automatic parameter selection may not be easy. Nonetheless it appears that when improvement is possible, τ on it own was quite effective when used with the classical perceptron. Notice that τ is consistently effective across all data sets; λ and α do not help on all data sets and λ sometimes leads to a drop in performance. As one might expect, our preliminary experiments also showed that very large values of τ harm performance significantly. These are not shown in the graphs as we have limited the range of τ in the experiments.

Tables 3 and 4 summarize the results of parameter search experiments on the real world data and some of the artificial data, respectively. For each data set and algorithm the tables give the best accuracy that can be achieved with parameters in the range tested. This is useful as it can indicate whether an algorithm has a potential for improvement, in that for some parameter setting it gives good performance. In order to clarify the contribution of different parameters, each column with

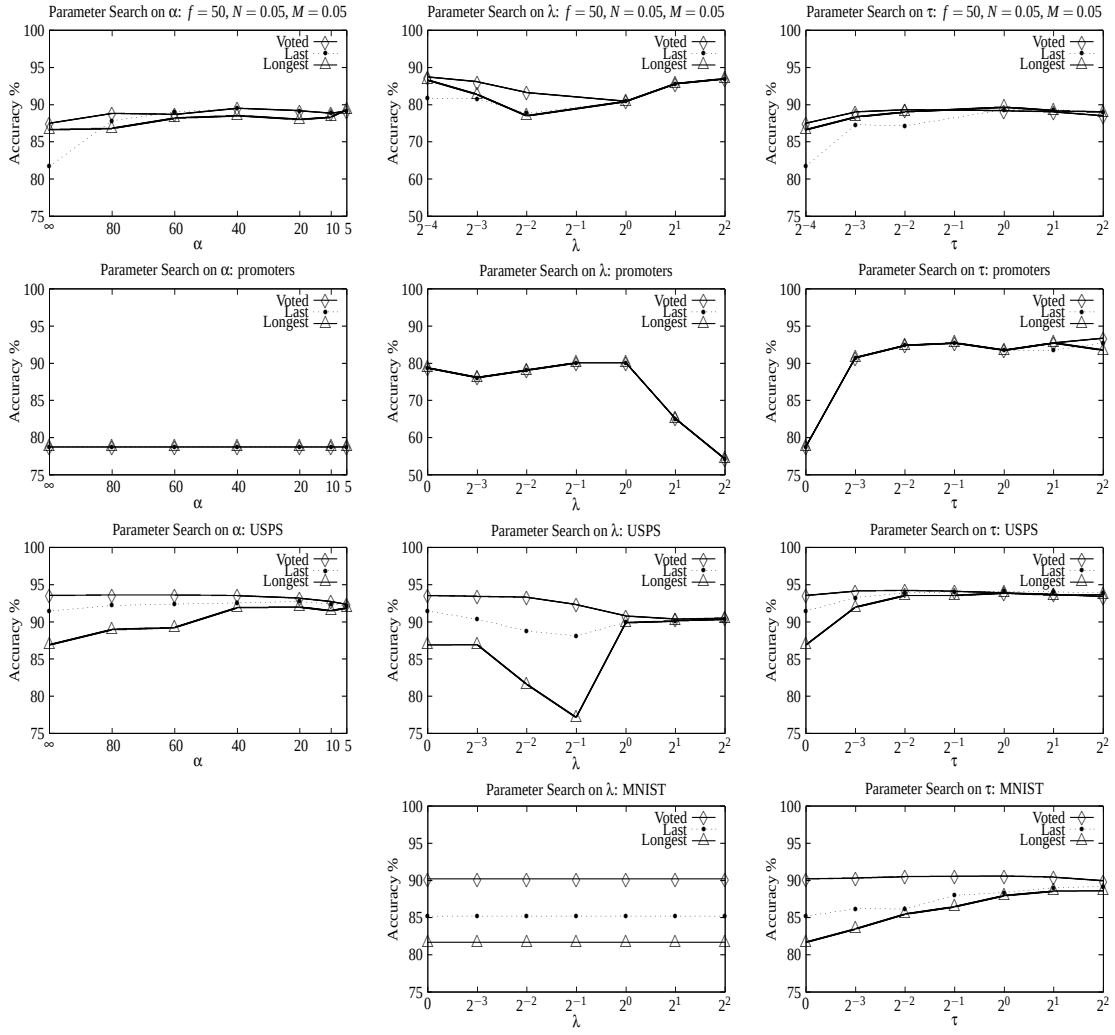

Figure 4: Parameter Search on Artificial and Real-world Data

parameters among τ, α, λ includes all values for the parameter except the non-active value. For example, any accuracy obtained in the τ column is necessarily obtained with a value $\tau \neq 0$. The column labeled "Nothing" shows results when all parameters are inactive.

Several things can be observed in the tables. Consider first the margin parameters. We can see that τ is useful even in data sets with noise; this is obvious both in the noisy artificial data sets and in the real-world data sets, all of which are inseparable in the native feature space. We can also see that α and λ do improve performance in a number of cases. However, they are less effective in general than τ, and do not provide additional improvement when combined with τ.

Looking next at the on-line to batch conversions we see that the differences between the basic algorithm, the longest survivor and the voted perceptron are noticeable without margin based variants. For the artificial data sets this only holds for one group of data sets ($f = 50$), the one with

		Baseline	Nothing	τ	λ	α	$\tau \times \lambda$	$\tau \times \alpha$
breast-cancer-wisconsin	Last	65.5	90.6	96.8	97.2	96.9	97.3	97.2
	Longest		96.9	97.0	97.2	97.0	97.3	97.2
	Voted		96.9	96.8	97.2	96.9	97.3	97.2
bupa	Last	58	57.5	71.8	64.1	69.2	71.5	67.8
	Longest		64.1	58.2	64.8	68.4	60.9	61.2
	Voted		68.9	65.9	67.7	70.7	67.4	66.0
wdbc	Last	62.7	92.4	93.2	92.8	93.3	93.9	92.6
	Longest		92.4	93.2	92.6	92.4	92.8	92.8
	Voted		92.3	92.0	93.2	92.4	93.2	92.4
crx	Last	55.5	62.5	68.6	65.5	66.7	69.0	66.7
	Longest		55.1	63.6	65.5	63.5	65.5	65.2
	Voted		64.9	65.4	65.5	66.7	65.5	66.4
promoters	Last	50	78.8	92.8	82.1	78.8	94.4	93.8
	Longest		78.8	92.8	82.1	78.8	94.4	94.4
	Voted		78.8	93.4	82.1	78.8	94.4	93.8
ionosphere	Last	64.4	86.6	87.5	86.9	87.7	88.6	87.2
	Longest		87.2	87.5	87.8	88.0	88.3	87.7
	Voted		88.0	87.7	87.5	88.6	88.9	87.5
wpbc	Last	76.3	77.3	76.4	80.6	76.9	79.0	76.4
	Longest		77.2	76.4	77.4	78.5	76.9	76.4
	Voted		78.8	76.4	80.6	77.4	78.5	76.4
sonar.all-data	Last	53.4	71.9	74.6	72.5	77.1	75.8	79.1
	Longest		75.3	77.1	73.3	77.2	77.7	78.7
	Voted		75.1	77.2	73.8	77.1	78.7	77.7
USPS	Last	N/A	91.5	94.1	90.4	92.8	94.3	94.1
	Longest	N/A	86.9	93.9	90.4	92	93.9	93.9
	Voted	N/A	93.5	94.2	93.4	93.6	94.3	94.3
MNIST	Last	N/A	85.2	89.2	85.2		89.2	
	Longest	N/A	81.7	88.6	81.7		88.6	
	Voted	N/A	90.2	90.6	90.2		90.6	

Table 3: Parameter Search on UCI and Other Data Sets

highest ratio of number of examples to number of features ($12:1$).[4] The longest survivor seems less stable and degrades performance in some cases.

Finally compare the τ variant with voted perceptron and their combination. For the artificial data sets using τ alone (with last hypothesis) gives higher accuracy than using the voted perceptron alone. For the real-world data sets this trend is less pronounced. Concerning their combination we see that while τ always helps the last hypothesis, it only occasionally helps voted, and sometimes hurts it.

Table 5 gives detailed results for parameter search over τ on the adult data set, where we also parameterize the results by the number of examples in the training set. We see that voted perceptron and the τ variant give similar performance on this data set as well. We also see that that in contrast

4. The results for data with $f = \{200, 500\}$ are omitted from the table. In these cases these was no difference between the basic, longest and voted versions except when combining with other variants.

Noise Pctg. (N)	Nothing	τ	λ	α	$\tau \times \lambda$	$\tau \times \alpha$
Last, $N = 0$	95.4	97	95.6	96.1	96.8	97
Longest	95.4	96.6	95.6	95.9	97.3	96.8
Voted	95.4	96.6	95.4	96.1	96.8	97
Last, $N = 0.05$	81.7	89.4	87	89.5	89.7	89.9
Longest	86.6	89.7	87	89.3	89.9	89.9
Voted	87.5	89.4	87	89.5	89.7	90.2
Last, $N = 0.1$	77.8	84.6	81.9	85.1	84.9	85.8
Longest	77.2	84.9	81.9	85.1	84.6	85.5
Voted	81.6	84.9	81.9	85.1	84.9	85.6
Last, $N = 0.15$	71.2	81.6	79.9	80.6	81.6	81.7
Longest	72.9	80.7	79.9	79.2	80.7	81.7
Voted	75.6	81.6	79.9	80.6	82.1	81.6
Last, $N = 0.25$	59.9	70.7	68.2	70.7	71.1	71.1
Longest	65	70.7	68.2	70.1	70.4	70.6
Voted	68	70.4	69.4	70.7	70.6	71.1

Table 4: Parameter Search on Artificial Data Sets with $f = 50$ and $M = 0.05$

with the performance on the artificial data mentioned above, voted perceptron performs well even with small training set size (and small ratio of examples to features).

The table adds another important observation about stability of the algorithms. Note that since we report results for concrete values of τ we can measure the standard deviation in accuracy observed. One can see that both the τ variant and the voted perceptron significantly reduce the variance in results. The longest survivor does so most of the time but not always. The fact that the variants lead to more stable results is also consistently true across the artificial and UCI data sets discussed above and is an important feature of these algorithms.

Finally, recall that the *average algorithm* (Servedio, 1999), which updates exactly once on each example, is known to tolerate classification noise for data distributed uniformly on the sphere and where the threshold is zero. For comparison, we ran this algorithm on all the data reflected in the tables above and it was not competitive. Cursory experiments to investigate the differences show that, when the data has a zero threshold separator and when perceptron is run for just one iteration, then (as reported in Servedio, 1999) the average algorithm performs better than basic perceptron. However, the margin based perceptron performs better and this becomes more pronounced with non-zero threshold and multiple iterations. Thus in some sense the perceptron variants are doing something non-trivial that is beyond the scope of the simple average algorithm.

3.4 Parameter Optimization

We have run the parameter optimization on the real-world data sets and the artificial data sets. Selected results are given in Tables 6 and 7. For these experiments we report average accuracy across the outer cross-validation as well as a 95% T-confidence interval around these, as suggested

# examples:	10		100		1000		5000		10000		20000	
	Accuracy	Std. Dev.										
$\tau = 0$												
Last	75.6	3.5	70.2	9.2	77.8	3.1	78.3	4.1	80.3	2.8	76.5	11.1
Longest	75.7	3.5	76.5	3.6	79.9	3.5	81.8	1.8	82.9	0.7	82.3	1.3
Voted	75.8	3.4	79.4	1.8	82.5	0.5	84	0.7	84.2	0.6	84.5	0.5
$\tau = 0.125$												
Last	72.7	4.6	73.3	8.9	79.7	3.7	80.8	2.7	82.7	1.4	80.4	3.9
Longest	73.1	4.6	76.6	3.9	81	2.1	81.1	2.4	81.8	1.5	82.1	1.7
Voted	74.1	4.4	79.6	1.4	82.7	0.5	83.9	0.7	84.2	0.6	84.3	0.4
$\tau = 0.25$												
Last	74.5	1.8	75.1	7.4	79.3	5.1	80.5	2.8	81.5	2.2	80.6	3.7
Longest	73.9	3.9	77.2	2.1	80.3	2.6	81.8	1.8	81.7	1.7	82.3	2.4
Voted	74.1	4.7	79.5	1.5	82.7	0.5	83.8	0.7	84.1	0.7	84.3	0.5
$\tau = 0.5$												
Last	71.8	6.3	75.5	1.9	80.4	3.4	82.6	1	82.7	1	82.8	1.5
Longest	74	5.7	77.2	7.5	80.4	2.3	82.1	1.6	82.5	1.6	80.8	2.9
Voted	73.6	5.6	78.3	0.8	82.6	0.6	83.7	0.7	84	0.7	84.2	0.5
$\tau = 1$												
Last	72.9	6.6	78.4	2.5	81.9	1.2	83	1.1	82.7	1.5	83.5	0.8
Longest	75.9	0.3	75.9	0.9	78.3	2.8	82.5	1.1	83	1	81.9	2.7
Voted	74.8	3.4	76.4	0.9	82.4	0.6	83.5	0.7	83.7	0.7	84.1	0.5
$\tau = 2$												
Last	75.5	1.2	75.8	2.6	82.4	0.4	83.2	0.9	82.8	1.2	83.6	0.8
Longest	70.7	15.6	75.9	0.9	77.8	2.7	81.4	2.9	82.5	2	81.9	2.8
Voted	70.7	15.6	76.4	1.8	81.8	1	83.2	0.7	83.5	0.6	83.8	0.6
$\tau = 4$												
Last	75.9	0.3	75.7	1.4	81.8	1	83.1	2.1	83.3	0.6	83.6	0.6
Longest	70.7	15.6	75.9	0.9	75.9	0.5	81.8	0.7	79.4	3.4	80.8	3.1
Voted	70.7	15.6	75.9	0.9	78.4	2.1	83	0.7	83.2	0.6	83.5	0.6

Table 5: Performance on "Adult" Data Set as a Function of Margin and Training Set Size

in Mitchell (1997).[5] No confidence interval is given for "USPS," "MNIST," or "Adult," as the outer cross-validation loop is not performed.

In contrast with the tables for parameter search, columns of parameter variants in Tables 6 and 7 do include the inactive option. Hence the search in the τ column includes the value of $\tau = 0$ as well. This makes sense in the context of parameter optimization since the algorithm can choose between different active values and the inactive value. Notice that the standard deviation in accuracies is high on these data sets. This highlights the difficulty of parameter selection and algorithm comparison and suggests that results from single split into training and test sets that appear in the literature may not be reliable.

5. In more detail, we use the maximum likelihood estimate of the standard deviation σ, $s = \sqrt{\frac{1}{k}\sum_i (y_i - \bar{y})^2}$ where k is the number of folds (here $k = 10$), y_i is the accuracy estimate in each fold and $\bar{y}$ is the average accuracy. We then use the T confidence interval $y \in \hat{y} \pm t_{(k-1),0.975}\sqrt{\frac{1}{k(k-1)}\sum (y_i - \bar{y})^2}$. Notice that since $t_{9,0.975} = 2.262$ and $k = 10$ the confidence interval is 0.754 of the standard deviation.

	Nothing	τ	λ	α	$\tau \times \alpha$	$\tau \times \lambda$	$\tau \rightarrow \alpha$	$\tau \rightarrow \lambda$	SVM L1	SVM L2
Breast-cancer-wisconsin										
Last	90.6 +/- 2.3	95.2 +/- 2.1	95.2 +/- 3.1	94.7 +/- 3.3	95.3 +/- 2.6	96.3 +/- 2.1	95.1 +/- 2.3	96.9 +/- 1.3		
Longest	96.9 +/- 1.2	96.8 +/- 1.7	97.2 +/- 1.4	96.3 +/- 1.9	96.8 +/- 1.6	97 +/- 1.6	96.8 +/- 1.7	96.9 +/- 1.8	96.8 +/- 2.2	96.7 +/- 1.7
Voted	96.9 +/- 1.3	96.8 +/- 1.4	97 +/- 1.4	96.6 +/- 1.6	97 +/- 1.6	97.2 +/- 1.4	96.8 +/- 1.4	96.9 +/- 1.5		
Bupa										
Last	57.5 +/- 7.8	69.8 +/- 6.5	61.5 +/- 6.3	68.9 +/- 4.1	69.9 +/- 5.7	69.8 +/- 5.8	69.8 +/- 6.5	70.9 +/- 5.9		
Longest	64.1 +/- 7.1	64.1 +/- 7.1	64.1 +/- 6.6	65.3 +/- 4.3	65.3 +/- 4.3	64.1 +/- 6.6	65.3 +/- 4.3	64.1 +/- 6.6	66.5 +/- 8.6	63.2 +/- 5.2
Voted	68.9 +/- 5.8	68.9 +/- 5.8	67.5 +/- 6.4	68.9 +/- 6.3	68.9 +/- 6.3	67.5 +/- 6.2	68.9 +/- 6.3	67.2 +/- 6.0		
Wdbc										
Last	92.4 +/- 2.9	93 +/- 2.7	93.2 +/- 2.5	91.6 +/- 2.7	92.8 +/- 2.8	93.2 +/- 2.8	93 +/- 2.7	93.5 +/- 2.2		
Longest	92.4 +/- 2.0	91.7 +/- 2.5	92.1 +/- 2.5	92.3 +/- 2.3	93.2 +/- 1.7	92.5 +/- 2.1	92.6 +/- 2.1	91.4 +/- 2.9	92.8 +/- 4.4	94.7 +/- 2.1
Voted	92.3 +/- 2.3	92.3 +/- 2.6	92.8 +/- 2.7	91.7 +/- 1.9	91.6 +/- 2.2	92.4 +/- 2.4	91.6 +/- 2.6	92.8 +/- 2.3		
Crx										
Last	62.5 +/- 5.1	68.7 +/- 2.9	64.5 +/- 4.1	65.8 +/- 4.1	68.1 +/- 3.5	68.7 +/- 2.2	68.7 +/- 2.9	67.8 +/- 2.8		
Longest	55.1 +/- 7.3	60.4 +/- 6.5	62.6 +/- 4.8	62.6 +/- 4.4	64.5 +/- 4.9	60.9 +/- 5.7	62.9 +/- 4.6	59.4 +/- 6.7	76.6 +/- 13.7	65.9 +/- 22.8
Voted	64.9 +/- 4.0	64.2 +/- 2.7	65.4 +/- 3.2	66.2 +/- 3.8	66.4 +/- 3.7	64.9 +/- 2.6	65.7 +/- 3.3	64.3 +/- 3.1		
Ionosphere										
Last	86.6 +/- 3.8	87.2 +/- 3.4	86.3 +/- 3.5	85.7 +/- 4.8	86.9 +/- 3.4	86.9 +/- 2.6	86.6 +/- 3.0	86.6 +/- 2.6		
Longest	87.2 +/- 3.8	86.9 +/- 2.6	87.8 +/- 3.6	87.5 +/- 3.5	86.9 +/- 4.0	87.2 +/- 3.2	86.9 +/- 3.4	87.7 +/- 2.9	87.1 +/- 7.1	86.9 +/- 4.2
Voted	88 +/- 3.7	86.3 +/- 4.3	86.6 +/- 3.5	87.7 +/- 3.2	87.5 +/- 3.2	87.7 +/- 3.1	86 +/- 4.9	86 +/- 3.9		
Wpbc										
Last	77.3 +/- 4.7	76.7 +/- 4.4	76.4 +/- 4.4	75.1 +/- 6.1	75.1 +/- 6.1	77 +/- 5.2	75.7 +/- 6.2	78.3 +/- 5.1		
Longest	77.2 +/- 3.8	76.7 +/- 5.3	75.8 +/- 3.6	75.8 +/- 6.0	76.9 +/- 4.4	74.8 +/- 4.3	75.8 +/- 4.4	74.6 +/- 5.4	78.6 +/- 7.8	78.6 +/- 8.4
Voted	78.8 +/- 4.7	78.5 +/- 4.8	79 +/- 5.0	76.4 +/- 4.8	76.9 +/- 4.8	79 +/- 5.0	76.9 +/- 4.8	78.5 +/- 5.1		
Sonar										
Last	71.9 +/- 6.3	73.1 +/- 5.4	72.4 +/- 6.0	75.5 +/- 6.4	72.1 +/- 7.4	71.1 +/- 6.5	74.1 +/- 6.4	73.6 +/- 5.1		
Longest	75.3 +/- 5.1	77.6 +/- 6.2	73.3 +/- 6.0	74.3 +/- 6.9	73.5 +/- 5.9	74.1 +/- 7.5	74 +/- 5.5	78.1 +/- 6.6	57.2 +/- 11.9	55 +/- 11.8
Voted	75.1 +/- 6.0	75.6 +/- 6.9	74.3 +/- 5.6	77.5 +/- 7.0	78.1 +/- 7.0	77.1 +/- 7.2	76.1 +/- 7.9	75.6 +/- 6.9		
f=50,N=0.05,M=0.05										
Last	81.7 +/- 6.5	87.7 +/- 6.1	84.6 +/- 6.6	87.2 +/- 5.2	88 +/- 5.2	89.4 +/- 4	89 +/- 4.6	89.5 +/- 4.2		
Longest	86.6 +/- 4.3	88.2 +/- 4.2	85.6 +/- 4.7	87.3 +/- 6.1	89.4 +/- 4.5	88.2 +/- 3.9	89 +/- 5.4	88.8 +/- 3.6		
Voted	87.5 +/- 3.3	88.8 +/- 4.8	86.3 +/- 3.9	88.3 +/- 4.2	88.2 +/- 4.9	88.8 +/- 4.8	88.7 +/- 4.9	88.8 +/- 4.8		

Table 6: Parameter Optimization Results for UCI and Artificial Data Sets

	Nothing	τ	λ	$\tau \times \lambda$	$\tau \rightarrow \lambda$
USPS					
Last	87.4	90.7	87.4	90.3	90.2
Longest	87.5	89.9	87.4	90.3	89.9
Voted	89.7	90.9	89.6	90.9	90.9
Adult					
Last	81.9	83.9	83	83.8	83.8
Longest	83.4	83.4	83.4	83.6	83.4
Voted	84.4	84.2	84.4	84.3	84.3
MNIST					
Last	85.6	87.1	85.5	87.1	87.1
Longest	79.8	86.8	79.8	86.8	86.8
Voted	88	88.4	88	88.4	88.4
USPS,degree 4 poly kernel					
Last	92.9	93.6			
Longest	91.6	92.9			
Voted	92.7	93.2			
MNIST,degree 4 poly kernel					
Last	95.2	95.4			
Longest	93.2	94.9			
Voted	94.8	95			

Table 7: Parameter Optimization Results for Large Data Sets

Table 6 shows improvement over the basic algorithm in all data sets where parameter search suggested a potential for improvement, and no decrease in performance in the other cases, so the parameter selection indeed picks good values. Both τ and the voted perceptron provide consistent improvement over the classical perceptron; the longest survivor provides improvement over the classical perceptron on its own, but a smaller one than voted or τ in most cases. Except for improvements over the classical perceptron, none of the differences between algorithms is significant according to the T-intervals calculated. As observed above in *parameter search*, the variants with α and λ offer improvements in some cases, but when they do, τ and voted almost always offer a better improvement. Ignoring the intervals we also see that neither the τ variant nor the voted perceptron dominates the other. Combining the two is sometimes better but may decrease performance in the high variance cases.

For further comparison we also ran experiments with kernel based versions of the algorithms. Typical results in the literature use higher degree polynomial kernel on the "MNIST" and "USPS" data sets. Table 7 includes results using τ with a degree 4 polynomial kernel for these data sets. We can see that for "MNIST" the variants make little difference in performance but that for "USPS" we get small improvements and the usual pattern relating the variants is observed.

Finally, Table 6 also gives results for SVM. We have used SVMlight (Joachims, 1999) and ran with several values for the constants controlling the soft margin. For L_1 optimization the values used for the "-c" switch in SVMLight are $\{10^{-5}, 10^{-4}, 10^{-3}, \ldots, 10^{4}\}$. For the L_2 optimization, we

used the following values for λ: $\{10^{-4}, 10^{-3}, \ldots, 10^{3}\}$. The results for SVM are given for the best parameters in a range of parameters tried. Thus these are parameter search values and essentially give upper bounds on the performance of SVM on these data sets. As can be seen the perceptron variants give similar accuracies and smaller variance and they are therefore an excellent alternative for SVM.

4. Discussion and Conclusions

The paper provides an experimental evaluation of several noise tolerant variants of the perceptron algorithm. The results are surprising since they suggest that the perceptron with margin is the most successful variant although it is the only one not designed for noise tolerance. The voted perceptron has similar performance in most cases, and it has the advantage that no parameter selection is required for it. The difference between voted and perceptron with margin are most noticeable in the artificial data sets, and the two are indistinguishable in their performance on the UCI data. The experiments also show that the soft-margin variants are generally weaker than voted or margin based algorithm and they do not provide additional improvement in performance when combined with these. Both the voted perceptron and the margin variant reduced the deviation in accuracy in addition to improving the accuracy. This is an important property that adds to the stability of the algorithms. Combining voted and perceptron with margin has the potential for further improvements but can harm performance in high variance cases. In terms of run time, the voted perceptron does not require parameter selection and can therefore be faster to train. On the other hand its test time is slower especially if one runs the primal version of the algorithm.

Overall, the results suggest that a good tradeoff is obtained by fixing a small value of τ; this gives significant improvements in performance without the training time penalty for parameter optimization or the penalty in test time for voting. In addition, since we have ruled out the use of the soft margin α variant we no longer need to run the algorithms for a large number of iterations; on the other hand, although on-line to batch conversion guarantees only hold for one iteration, experimental evidence suggests that a small number of iterations is typically helpful and sufficient.

Our results also highlight the problems involved with parameter selection. The method of double cross-validation is time intensive and our experiments for the large data sets were performed using the primal form of the algorithms since the dual form is too slow. In practice, with a large data set one can use a hold-out set for parameter selection so that run time is more manageable. In any case, such results must be accompanied by estimates of the deviation to provide a meaningful interpretation.

As mentioned in the introduction a number of variants that perform margin based updates exist in the literature (Friess et al., 1998; Gentile, 2001; Li and Long, 2002; Crammer et al., 2005; Kivinen et al., 2004; Shalev-Shwartz and Singer, 2005; Tsampouka and Shawe-Taylor, 2005). The algorithms vary in some other details. Aggressive ROMMA (Li and Long, 2002) explicitly maximizes the margin on the new example, relative to an approximation of the constraints from previous examples. NORMA (Kivinen et al., 2004) performs gradient descent on the soft margin risk resulting in an algorithm that rescales the old weight vector before the additive update. The Passive-Aggressive Algorithm (Crammer et al., 2005) adapts η on each example to guarantee it is immediately separable with margin (although update size is limited per noisy data). The Ballseptron (Shalev-Shwartz and Singer, 2005) establishes a normalized margin and replaces margin updates with updates on hypothetical examples on which a mistake would be made. ALMA (Gentile, 2001) renormalizes

the weight vector so as to establish a normalized margin. ALMA is distinguished by tuning its parameters automatically during the on-line session, as well as having "p-norm variants" that can lead to other tradeoffs improving performance, for example, when the target is sparse. The algorithms of Tsampouka and Shawe-Taylor (2005) establish normalized margin by different normalization schemes. Following Freund and Schapire (1999) most of these algorithms come with mistake bound guarantees (or relative loss bounds) for the unrealizable case, that is, relative to the best hypothesis in a comparison class. Curiously, identical or similar bounds hold for the basic perceptron algorithm so that these results do not establish an advantage of the variants over the basic perceptron.

Another interesting algorithm, the Second Order Perceptron (Cesa-Bianchi et al., 2005), does not perform margin updates but uses spectral properties of the data in the updates in a way that can reduce mistakes in some cases.

Additional variants also exist for the on-line to batch conversions. Littlestone (1989) picks the best hypothesis using cross validation on a separate validation set. The Pocket algorithm with ratchet (Gallant, 1990) evaluates the hypotheses on the entire training set and picks the best, and the scheme of Cesa-Bianchi et al. (2004) evaluates each hypothesis on the remainder of the training set (after it made a mistake and is replaced) and adjusts for the different validation set sizes. The results in Cesa-Bianchi et al. (2004) show that loss bounds for the on line setting can be translated to error bounds in the batch setting even in the unrealizable case. Finally, experiments with aggressive ROMMA (Li and Long, 2002; Li, 2000) have shown that adding a voted perceptron scheme can harm performance, just as we observed for the margin perceptron. To avoid this, Li (2000) develops a scheme that appears to work well where the voting is done on a tail of the sequence of hypotheses which is chosen adaptively (see also Dekel and Singer, 2005, for more recent work).

Consider the noise tolerance guarantees for the unrealizable case. Ideally, one would want to find a hypothesis whose error rate is only a small additive factor away from the error rate of the best hypothesis in the class of hypotheses under consideration. This is captured by the agnostic PAC learning model (Kearns et al., 1994). It may be worth pointing out here that, although we have relative loss bounds for several variants and these can be translated to some error bounds in the batch setting, the bounds are not sufficiently strong to provide significant guarantees in the agnostic PAC model. So this remains a major open problem to be solved.

Seeing our experimental results in light of the discussion above there are several interesting questions. The first is whether the more sophisticated versions of margin perceptron add significant performance improvements. In particular it would be useful to know what normalization scheme is useful in what contexts so they can be clearly applied. We have raised parameter tuning as an important issue in terms of run time and the self tuning capacity of ALMA and related algorithms seems promising as an effective solution. Given the failure of the simple longest survivor it would be useful to evaluate the more robust versions of Gallant (1990) and Cesa-Bianchi et al. (2004). Notice, however that these methods have a cost in training time. Alternatively, one could further investigate the tail variants of the voting scheme (Li, 2000) or the "averaging" version of voting (Freund and Schapire, 1999; Gentile, 2001), explaining when they work with different variants and why. Finally, as mentioned above, to our knowledge there is no theoretical explanation to the fact that perceptron with margin performs better than the basic perceptron in the presence of noise. Resolving this is an important problem.

Acknowledgments

This work was partially supported by NSF grant IIS-0099446 and by Research Semester Fellowship Award from Tufts University. Many of the experiments were run on the Research Linux Cluster provided by Tufts Computing and Communications Services. We are grateful to Phil Long for pointing out the results in Li (2000) and the reviewers for pointing out Littlestone (1989) and Cesa-Bianchi et al. (2004) and for comments that helped improve the presentation of the paper.

References

B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, 1992.

N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the generalization ability of on-line learning algorithms. *Information Theory, IEEE Transactions on*, 50(9):2050–2057, 2004.

N. Cesa-Bianchi, A. Conconi, and C. Gentile. A second-order Perceptron algorithm. *SIAM Journal on Computing*, 34(3):640–668, 2005.

W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

M. Collins. Ranking algorithms for named entity extraction: Boosting and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 489–496, 2002.

K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. Online passive aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2005.

N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines*. Cambridge University Press, 2000.

O. Dekel and Y. Singer. Data driven online to batch conversions. In *Advances in Neural Information Processing Systems 18 (Proceedings of NIPS 2005)*, 2005.

T. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40:139–158, 2000.

T. Dietterich, M. Kearns, and Y. Mansour. Applying the weak learning framework to understand and improve c4.5. In *Proceedings of the International Conference on Machine Learning*, pages 96–104, 1996.

Y. Freund and R. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37:277–296, 1999.

T. Friess, N. Cristianini, and C. Campbell. The kernel adatron algorithm: A fast and simple learning procedure for support vector machines. In *Proceedings of the International Conference on Machine Learning*, pages 188–196, 1998.

S. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2): 179–191, 1990.

A. Garg and D. Roth. Margin distribution and learning algorithms. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 210–217, 2003.

C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001.

T. Graepel, R. Herbrich, and R. Williamson. From margin to sparsity. In *Advances in Neural Information Processing Systems 13 (Proceedings of NIPS 2000)*, pages 210–216, 2000.

T. Joachims. Making large-scale svm learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

M. Kearns, M. Li, L. Pitt, and L. G. Valiant. Recent results on boolean concept learning. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 337–352, 1987.

M. Kearns, R. Schapire, and L. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17 (2/3):115–141, 1994.

J. Kivinen, A. J. Smola, and R. C. Williamson. Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8):2165–2176, 2004.

A. Kowalczyk, A. Smola, and R. Williamson. Kernel machines and boolean functions. In *Advances in Neural Information Processing Systems 14 (Proceedings of NIPS 2001)*, pages 439–446, 2001.

W. Krauth and M. Mézard. Learning algorithms with optimal stability in neural networks. *Journal of Physics A*, 20(11):745–752, 1987.

Y. Li. Selective voting for perception-like online learning. In *Proceedings of the International Conference on Machine Learning*, pages 559–566, 2000.

Y. Li and P. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46:361–387, 2002.

Y. Li, H. Zaragoza, R. Herbrich, J. Shawe-Taylor, and J. Kandola. The perceptrom algorithm with uneven margins. In *International Conference on Machine Learning*, pages 379–386, 2002.

N. Littlestone. From on-line to batch learning. In *Proceedings of the Conference on Computational Learning Theory*, pages 269–284, 1989.

T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

A. B. Novikoff. On convergence proofs on perceptrons. *Symposium on the Mathematical Theory of Automata*, 12:615–622, 1962.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958.

R.A. Servedio. On PAC learning using Winnow, Perceptron, and a Perceptron-like algorithm. In *Proceedings of the Twelfth Annual Conference on Computational learning theory*, pages 296–307, 1999.

S. Shalev-Shwartz and Y. Singer. A new perspective on an old perceptron algorithm. In *Proceedings of the Sixteenth Annual Conference on Computational Learning Theory*, pages 264–278, 2005.

P. Tsampouka and J. Shawe-Taylor. Analysis of generic perceptron-like large margin classifiers. In *Proceedings of the European Conference on Machine Learning*, pages 750–758, 2005.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.