Noise Tolerant Variants of the Perceptron Algorithm[⋆]

Roni Khardon and Gabriel M. Wachman

Department of Computer Science, Tufts University
Medford, MA 02155, USA
{roni,gwachm01}@cs.tufts.edu

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 substantial performance benefits. The results also highlight the difficulty with automatic parameter selection which is required with some of these variants.

1 Introduction

The success of support vector machines (SVM) [1, 24] has led to increasing interest in the perceptron algorithm. Like SVM, the perceptron algorithm has a linear threshold hypothesis, 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 (see e.g. [13]). 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

[⋆] Partly supported by NSF grant IIS-0099446.

large margin hypotheses with the efficiency of the perceptron algorithm (see e.g. [9, 12, 18]).

The focus in this paper is on variants that are designed to handle noisy data or otherwise compensate for the fact that, since the perceptron is a sequential algorithm, its final hypothesis may not be the best one to use. In particular, the longest survivor variant [15, 10] and the voted perceptron variant [8] do not use the last hypothesis from training the perceptron but instead pick a "best" hypothesis or take a vote among hypotheses produced during training. This change in hypothesis can be used to guarantee that the hypothesis is good in a statistical sense and provide guarantees on the performance of the algorithm in the PAC learning model [25]. A second family of variants utilize the idea of a margin as in SVM. The perceptron algorithm with margin [17, 19] forces the hypothesis to have an explicitly given margin on the training data if that is possible. Adding to this idea, one can mimic soft-margin versions of support vector machines [19, 16] within the perceptron algorithm. Technically, one constrains the update function of the perceptron algorithm so as to perform a trade-off similar to the resulting optimization in SVM. Most of these variants have already been introduced in the literature and studied in isolation from other variants. However, no study has explored the possibility of combining these variants or comparing their performance. We believe that this is important since these algorithms have already been used and demonstrated in applications with large datasets (e.g. [4]) and a better understanding of what works and when can have a direct implication for future use. This paper provides such an experimental study and indeed the results reveal interesting facts.

In particular the experiments show that 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. The voted perceptron comes second, and it has the advantage that no parameter selection is required for it. Combining the two has the potential for further improvements but this occasionally degrades performance. The results also suggest that the soft-margin variants do not provide additional improvements. 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 experiments also highlight the problems involved with parameter selection. Both the margin variant and the soft-margin extensions require selection of parameters and one must design an automatic method for doing that. Any such method would be time intensive since many versions of the algorithms need to be run and compared. For small datasets this can be done, but then the variance in performance may be too high for reliable parameter selection. For large datasets the computational overhead may be a limiting factor even if using a hold-out set for parameter selection. Identifying a good trade-off and methodology is an important experimental challenge.

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 have performed two kinds of experiments. In "parameter search" we report the best results obtained with any parameter setting. This helps set the scope and evaluate the potential of different algorithms to improve performance and provides insight about their performance, but of course it does not give statistically reliable results. In "parameter optimization" the algorithms automatically select the parameters and the performance can be interpreted statistically. Finally, in Section 4 we discuss the results and future work.

2 Algorithms

2.1 The Perceptron Algorithm

Basics — The perceptron algorithm [22] takes as input a set of training examples in $\mathbb{R}^n$ labeled $\{-1, 1\}$. Using a weight vector, $\mathbf{w}$, initialized to $\mathbf{0}$, and a threshold, θ, it predicts the label of each training example $\mathbf{x}$ to be $y = sign(\langle \mathbf{w}, \mathbf{x} \rangle + \theta)$. The algorithm adjusts $\mathbf{w}$ and θ on each misclassified example by an additive factor.

More formally, we can abstract the domain $\mathcal{X}$ by using a mapping Φ to represent the examples. Given a set of m training examples and their labels, $\mathcal{Z} = \{(x_1, y_1), \ldots, (x_m, y_m)\} \in (\mathcal{X} \times \{-1, 1\})$, a number of training iterations T, and a function $\Phi : \mathcal{X} \to \mathcal{F} \subseteq \mathbb{R}^n$, for each training iteration, the perceptron classifies each $x_i \in \mathcal{X}$ according to $sign(\langle \mathbf{w}, \Phi(x_i) \rangle - \theta)$ and updates $\mathbf{w}$ and θ if $sign(\langle \mathbf{w}, \Phi(x_i) \rangle - \theta) \neq y_i$. At the end of the T training iterations, the algorithm returns $(\mathbf{w}, \theta)$. The final weight vector and corresponding threshold returned by the algorithm are also known as the *hypothesis*. The algorithm is summarized in Figure 1.

Parameters — The parameter θ is known as the threshold or bias. In the classical version of the perceptron, θ is initialized to 0 and updated by an additive factor of each misclassified example. One way to conceptualize θ is to add a $(n+1)$ dimension to each training example with a fixed value of $\sqrt{C}$ as as opposed to maintaining a separate quantity and updating it. The parameter C controls the update rate of θ relative to other weights.

The parameter η is known as the "learning rate," as it controls the extent to which $\mathbf{w}$ can change on a single update. If η is too large, the hypothesis may seem unstable during the learning process since each update will make substantial changes. If η is too small, the required running time to find the separating hyperplane may be longer than the number of training iterations chosen. Note, however, that initializing θ to 0 means that η has no effect since $sign(\langle \mathbf{w}, \Phi(x_i) \rangle - \theta) = y_i$ iff $sign(\eta(\langle \mathbf{w}, \Phi(x_i) \rangle - \theta)) = y_i$. A value of $\theta = 0$ is often used in practice, but θ does affect results obtained and this can be significant in early iterations of the algorithm.

The Dual Form — Note that after k examples have been classified, $\mathbf{w} = \sum_{i=1}^{m} \eta \alpha_i y_i \Phi(x_i)$ where α_i is the number of mistakes that have been made on

```
Input set of examples and their labels $\mathcal{Z} = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \{-1, 1\})$, $\Phi : \mathcal{X} \to \mathcal{F} \subseteq \mathbb{R}^n$, $\eta$

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

Fig. 1. The basic perceptron algorithm

example i. Subsequently, a new example x_j is classified according to $sign(SUM - \theta)$ where

$$SUM = \sum_i \eta \alpha_i y_i \langle \Phi(x_i), \Phi(x_j) \rangle. \qquad (1)$$

The expression of $\mathbf{w}$ as the sum of the examples with coefficients is known as the "dual form" of the perceptron [2]. When using the dual form, the update procedure consists of simply incrementing the α_i for the x_i on which the mistake was made. The algorithm is summarized in Figure 2.

If the perceptron makes mistakes on k different examples, classifying a new example in the primal form takes $O(n)$ multiplications and additions, however classifying the new example in the dual form takes $O(nk)$ multiplications and additions. As $k = O(n)$, this is a significant performance difference. As we discuss below the dual version can be faster when using kernels to work implicitly in large feature spaces.

Margin, Separability, and Mistake Bounds When the data are linearly separable via some hyperplane $(\mathbf{w}, \theta)$, the margin is defined as $\gamma = \min_{1 \leq i \leq m}(y_i(\langle \mathbf{w}, x_i \rangle - \theta))$. When $(\mathbf{w}, \theta)$ is normalized, γ is the minimum Euclidean distance of any point in the dataset to $(\mathbf{w}, \theta)$. 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 [21], where $R = \max_{1 \leq i \leq m} \|x_i\|$.

In the case of non-separable data, the extent to which the data is non-separable can be quantified. The quantity $\xi_i = \max(0, \gamma - y_i(\langle \mathbf{w}, \mathbf{w} \rangle + \theta))$, known as a *slack variable*, is a measure of the degree to which x_i fails to have a margin γ via $\mathbf{w}$ [2]. Observe that when $\xi_i = 0$, it means that the example x_i has margin at least γ via the hyperplane defined by $(\mathbf{w}, \theta)$. The perceptron is guaranteed to make no more than $(\frac{2(R+D)}{\gamma})^2$ mistakes on T examples, where $D = \sqrt{\sum_{i=1}^{T} \xi_i^2}$ for any $\mathbf{w}, \gamma > 0$ [8, 23][1]

[1] [8] has a proof of this bound for one training iteration only. [23] has a proof for multiple iterations on a different perceptron variant, however the proof can be adapted to show the same bound for the classical perceptron.

```
Input as in primal form.

 – Initialize $\alpha \leftarrow 0^m$, $k \leftarrow 0$, $\mathbf{w_0} \leftarrow \mathbf{0}$ and $\theta \leftarrow \theta_{Init}$
 – for every training epoch:
 – for every $x_j \in \mathcal{X}$:
     • $SUM \leftarrow \sum_{i|\alpha_i \neq 0} \eta \alpha_i y_i (\langle \Phi(x_i), \Phi(x_j) \rangle)$
     • $\hat{y} \leftarrow sign(SUM - \theta)$
     • if $(\hat{y} \neq y_j)$
         ∗ $\alpha_j \leftarrow \alpha_j + 1$
         ∗ $\theta \leftarrow \theta + \eta C y_j$
```

Fig. 2. The perceptron in dual form

2.2 The λ-trick

The λ-*trick* [16, 19] attempts to minimize the effect of noisy examples during training similar to the L_2 soft margin technique used with support vector machines [2]. We classify example x_j according to $sign(SUM - \theta)$ where

$$SUM = \sum_{i|\alpha_i \neq 0} \eta(\alpha_i + \delta_{ij}\lambda) y_i \langle \Phi(x_i), \Phi(x_j) \rangle \tag{2}$$

and where

$$\delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ otherwise} \end{cases} .$$

Thus during training, if a mistake has been made on x_j then in future iterations we increase α_j artificially by λ when classifying x_j but not when classifying other examples. A high enough value of λ can make the term $(\alpha_j + \lambda) y_j \langle x_j, x_j \rangle$ dominate the sum in Equation (2). Consider a noisy example x_k such that Equation (1) repeatedly mis-classifies x_k. Using the λ-trick, the algorithm will correctly classify the noisy example as soon as $(\alpha_k + \lambda)$ is large enough to dominate the rest of the sum in Equation (2) and hence will not continue to mis-classify x_k, effectively ignoring the noisy example for the remaining training iterations. The disadvantage of making λ too large is that all examples will be classified correctly immediately thus eliminating the training procedure altogether.

2.3 The α-bound

This variant is motivated by the L_1 soft margin technique used with support vector machines [2]. The α-*bound* places a bound α on α_i in Equation (1), such that when the algorithm makes a mistake on some x_i, it does not increment α_i if $\alpha_i \geq \alpha$. 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, α_k remains the same, whereas the algorithm may continue to increase the α-coefficients of other non-noisy examples, hence increasing their influence in the final hypothesis. 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.

2.4 Perceptron Using Margins

The Perceptron Algorithm using Margins (PAM) [17] attempts to establish a constant sized margin, τ, during the training process. 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 [19]. In contrast, the classical perceptron has no lower bound on the margin of its final hypothesis. Following work on support vector machines [1, 2] 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

$$y_j(SUM - \theta) < \tau \tag{3}$$

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" the margin parameterized by τ.

A variant of PAM exists in which a different value of τ is chosen for positive examples than for negative examples [19]. This seems to be important when the class distribution is skewed. We do not study that variant in this paper.

2.5 The Kernel Perceptron

It is often the case that data in $\mathbb{R}^n$ are not separable by any hyperplane in $\mathbb{R}^n$ but the data are separable by some non-linear function in $\mathbb{R}^n$. The perceptron algorithm as described above is incapable of expressing non-linear functions of the input. By modifying the perceptron to take advantage of *kernel functions*, we can express non-linear separators.

A *kernel* is a function $\mathcal{K}$ such that $\mathcal{K}(x, y) = \langle \Phi(x), \Phi(y)\rangle$ for all $x, y \in \mathcal{X}$, where Φ is a mapping from $\mathcal{X}$ to some inner product space $\mathcal{F}$. In other words, the function $\mathcal{K}$ allows us to compute the inner product of x and y in the feature space $\mathcal{F}$ *without explicitly representing* $\Phi(x)$ *or* $\Phi(y)$. As an example, consider the kernel $\mathcal{K}(x, y) = (\langle x, y\rangle + T)^d$, where T is some constant. Algebraic manipulation shows that this is equivalent to calculating $\langle \Phi(x), \Phi(y)\rangle$ where our feature space is all monomials of degree $\leq d$, each weighted by some binomial coefficient. Hence this kernel function allows us to express non-linear relationships between the data while only doing a linear amount of work in the dimension of the original feature space. Now the dual form of the algorithm can be modified to use the kernel function to calculate the inner product implicitly.

2.6 Longest Survivor and Voted Perceptron

The classical perceptron returns the last weight vector **w**, i.e. the one obtained after all training has been completed, but this may not always be useful especially if the data is noisy. For example if our data is arranged such that a number of noisy examples reside at the end of the set then the last hypothesis is most affected by the noisy examples and may not be good. This is a general issue that has been studied in the context of using on-line algorithms that expect one example at a 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 to handle this issue exist. In particular [15] show that *longest survivor* hypothesis, i.e. 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 [25]. Several variations of this idea were independently proposed under the name of the *pocket algorithm* and empirical evidence for their usefulness was provided [10].

The voted perceptron [8] 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. This algorithm was analyzed in [8] where bounds on the error rate were provided for the noisy case as well. At first sight the voted-perceptron seems to require expensive calculation for prediction. But as pointed out in [8], 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 (e.g. if kernels are used) the prediction phase of the voted perceptron is not substantially more expensive than the prediction phase of the classical perceptron.

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. The voted perceptron is the only variant with known theoretical guarantees on performance in this case.

2.7 Algorithms Summary

We summarize the various algorithms and prediction strategies in Figure 3 that also gives a template for our implementation.

```
TRAINING: Input set of examples and their labels
$\mathcal{Z} = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \{-1, 1\})$, $\Phi : \mathcal{X} \to \mathcal{F} \subseteq \mathbb{R}^n$

 – Initialize $\alpha \leftarrow 0^m$,$k \leftarrow 0$,$\mathbf{w_0} \leftarrow \mathbf{0}$, $tally \leftarrow 0$, $best\_tally \leftarrow 0$, $\alpha_{\mathrm{l.s.}} \leftarrow 0^m$,
   $\theta \leftarrow \theta_{Init}$
 – for every training epoch
 – for every $x_j \in \mathcal{X}$:
     • $SUM \leftarrow \sum_{i|\alpha_i \neq 0} \eta(\alpha_i + \delta_{ij}\lambda) y_i \mathcal{K}(x_i, x_j)$
     • Predict:
         ∗ if $SUM < \theta - \tau$, $\hat{y} = -1$
         ∗ else if $SUM > \theta + \tau$,$\hat{y} = 1$
         ∗ else $\hat{y} = 0$
     • if $(\hat{y} \neq y_j)$ AND $(\alpha_i < \alpha)$
         ∗ $\alpha_j \leftarrow \alpha_j + 1$
         ∗ Update ‘‘mistakes’’ data structure
         ∗ $\theta \leftarrow \theta + \eta C y_i$
         ∗ $vote_{k+1} \leftarrow 0$
         ∗ $k \leftarrow k + 1$
         ∗ $tally \leftarrow 0$
     • else if $(\hat{y} = y_j)$
         ∗ $vote_k \leftarrow vote_k + 1$
         ∗ $tally \leftarrow tally + 1$
         ∗ if $(tally > best\_tally)$
             · $best\_tally \leftarrow tally$
             · $\alpha_{\mathrm{l.s.}} \leftarrow \alpha$

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

 – CLASSICAL:
     • $SUM \leftarrow \sum_{i=1}^{m} \eta y_i \alpha_i \mathcal{K}(x_i, x_{m+1})$
     • $\hat{y} \leftarrow sign(SUM - \theta)$
 – LONGEST SURVIVOR:
     • $SUM \leftarrow \sum_{i=1}^{m} \eta y_i \alpha_{l.s._i} \mathcal{K}(x_i, x_{m+1})$
     • $\hat{y} \leftarrow sign(SUM - \theta)$
 – VOTED:
     • $SUM \leftarrow 0$
     • for $k \leftarrow 1$ to num_mistakes
         ∗ $SUM_k \leftarrow SUM + \eta y_{mistake_i} \mathcal{K}(x_{mistake_i}, x_{m+1})$
         ∗ $\hat{y} \leftarrow sign(SUM - \theta)$
         ∗ VOTES $\leftarrow$ VOTES $+ vote_i \hat{y}$
     • $\hat{y}_{final} \leftarrow sign(VOTES)$
```

Fig. 3. Illustration of All Algorithm Variants

2.8 Choice and Setting of Parameters

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

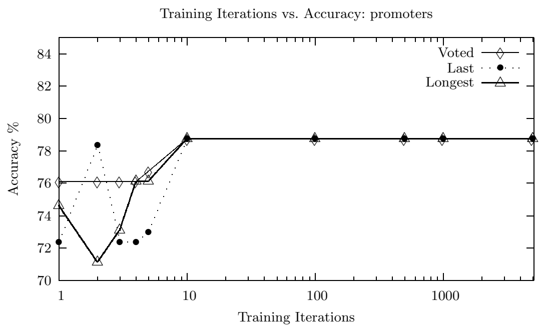

Fig. 4. Example Learning Curve for UCI Dataset

of θ_{Init},η, and the number of training iterations. In addition, for all parameters we would like to have fixed values or ranges that are comparable across all datasets, and so they should be independent of dataset characteristics. In the following we explain how these are formulated and chosen.

First we present the fixed values or formulas for η, θ_{Init}, C and the number of iterations. Prior to fixing these values, we ran experiments in which we varied θ_{init}, η, the number of training iterations, and 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. As explained above, the number of iterations must be sufficiently high to allow the α parameter (bounding α_i) to be effective, as well as to allow the weight vector to achieve some measure of stability. Except where noted below, we report results for 100 iterations and $\eta = 0.1$; our preliminary experiments revealed that while number of iterations does affect performance, the relationship between the performance of the algorithms remains similar. Additionally, 100 iterations appeared sufficient to achieve a stable hypothesis as illustrated on an example dataset in Figure 4. For the larger datasets, "USPS," "Adult" and "MNIST," which we will discuss further in section 3.1, 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.*

We set $\theta_{Init} = avg(\mathcal{K}(x_i, x_i))$, 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 fix the value of C to be in the same range as inner products in the original space, that is $C = avg(\langle x_i, x_i \rangle)$. This seems reasonable as C has a chance to affect the result but does not overwhelm the value of the original inner products.

Exploratory experiments with both natural and artificial data using the polynomial kernel with different degrees suggested that modifying the degree of the polynomial kernel will not provide further insight into the behavior of the algorithms in question; our experiments resulted in either degree-independent behaviors or overfitting. We therefore use a degree 1 polynomial, or linear, kernel, which is simply the inner product in the original feature space with an additive factor of C. Note that this means that we can run the experiments using the primal representation in order to reduce run times.

For the remaining parameters, τ, α, λ, we want to choose values that are dataset independent. The values of α and λ are naturally in this category since they are comparable to the α_i and therefore they scale with the number of iterations. For τ we rewrite the formulation above as follows. Instead of updating on x_j if

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

we update if

$$y_j(SUM - \theta) < \tau\theta_{Init} = \tau \cdot avg(\mathcal{K}(x_i, x_i))$$

so that τ can be thought of as measured in units of the average inner product of data in the domain. Our experiments explore natural ranges for these parameters whose size and granularity are constrained by the computational requirements of trying the different variations. Concrete settings are given in the experimental section.

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 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" wherein 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. Due to their size, for the "USPS," "MNIST," and "Adult" datasets, there is no outer cross-validation and we perform steps 1-4 as above only once. For ease of comparison to other published results, in steps 1-4 we use the 7291/2007 training/test split for "USPS" as in [19], and a 60000/10000 split for "MNIST" as in `http://yann.lecun.com/exdb/mnist/`.

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 datasets explore the extent to which this behavior is exhibited in real world problems.

For further comparison, we also ran SVM on the datasets using the L_1-norm soft margin and L_2-norm soft margin. We used SVM Light [14] and only ran *parameter search.*

3.1 Dataset Selection and Generation

For the artificial data we generated idealized scenarios. We first identify 4 types. The simplest, type 1, is linearly separable data with very small margin. Type 2 is linearly separable data with a larger margin. For types 3 and 4 we first generate data as in types 1 and 2 and then add random class noise by randomly reversing the labels of a given fraction of examples. 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 required for type 4 data. However, our experiments show that the picture is more complex; preliminary experiments with artificial data from types 1-3 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. We report on experiments with artificial data where the margin and noise levels are varied to effectively span the 4 different types.

Concretely the data was generated as follows. Given requested parameters for number of features, noise rate and the required margin size (as percentage of average of the square of the L_2-norm of the examples), we randomly generated the weight vector and examples. We measured the margin of the examples with respect to the weight vector and then discarded any examples that fell within the margin. 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

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 Dataset Characteristics

stands for the noise rate. We generated datasets 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 datasets, for a total 150 datasets.

For real world data we first selected two-class datasets from the UCI Machine Learning Repository [7] that have been used in recent comparative studies or in recent papers on linear classifiers [3, 6, 5, 11]. Since we require numerical attributes, any nominal attribute in these datasets was translated to a set of binary attributes each being an indicator function for one of the values. Since all these datasets have a relatively small number of examples we added three larger datasets to strengthen statistically our conclusions: "Adult" from UCI [7], and "MNIST[2]," and "USPS[3]," the 10-class character recognition datasets.

For the multiclass data, we trained one classifier for each class simultaneously, then for each example on the test set we chose the label of the classifier generating the maximum output. The datasets used and their characteristics after the nominal-to-binary feature transformation are summarized in Table 1.

3.2 Exploratory Experiments and General Setup

All the results reported give average accuracy in 10-fold cross-validation experiments, except where noted (in *parameter optimization* the average is over the outer 10-fold cross-validation). To avoid any ordering effects of the data, each dataset is randomly permuted before running the experiments. For the larger datasets, "MNIST," "Adult," and "USPS," in the *parameter optimization* experiments, the training/test split is maintained by permuting the training and test sets independently.

Finally we performed a comparison of the classical perceptron, the longest survivor and the voted perceptron algorithms without the additional variants.

[2] `http://yann.lecun.com/exdb/mnist/`

[3] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html\#usps`

	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

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 datasets, 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 datasets 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.

3.3 Parameter Values

As explained above the parameters are naturally scaled so that we can use the same ranges for all datasets. Since the double cross validation is expensive we limited the searches to the values: $\tau \in \{0, 0.125, 0.25, 0.5, 1, 2, 4\}$, $\alpha \in \{\infty, 80, 60, 40, 20, 10, 5\}$, and $\lambda \in \{0, 0.125, 0.25, 0.5, 1, 2, 4\}$ as well as all combinations $\tau \times \alpha$ and $\tau \times \lambda$ in these ranges. Notice that the values as listed from left to right vary from no effect to a strong effect for each parameter. Since search 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 or λ while fixing the chosen τ value.

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.

3.4 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 Table 4 and discussed below. Before showing these we discuss the effects of single parameters. Figure 5 illustrates that the accuracy is reasonably well-behaved with respect to the parameters; good performance is obtained in a non negligible region, but local maxima exist; this is typical of the results from

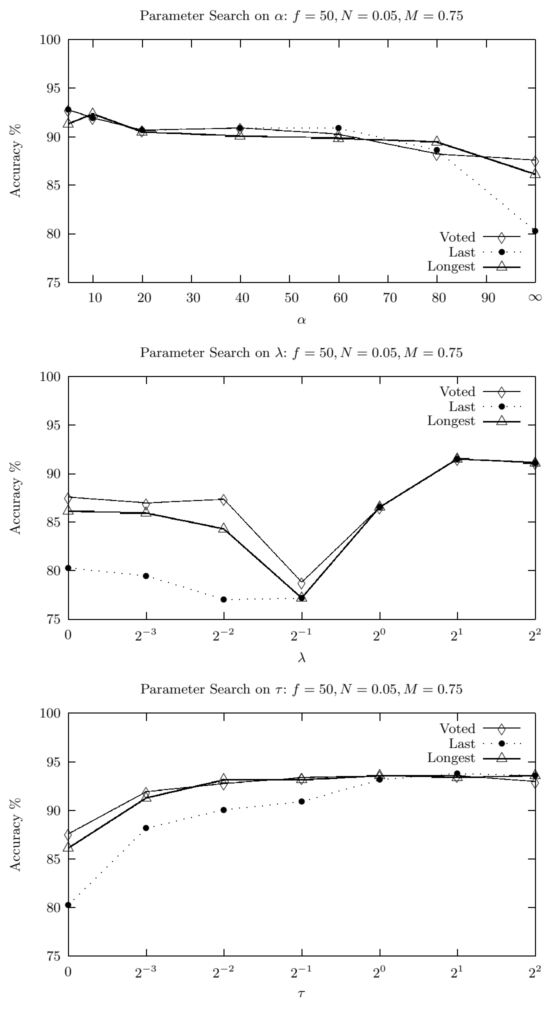

Fig. 5. Parameter Search on Artificial Data

the artificial data. Data obtained for the non-artificial datasets showed somewhat different characteristics. In some datasets little improvement was obtained with any variant or parameter setting. In others, improvement was obtained for some parameter values but the regions were not as large. This suggests that automatic parameter selection may be tricky for these datasets. Nonetheless it appears that when improvement is possible, τ on it own was quite effective; notice in Figures 6,7, and 8 that τ is consistently effective, but λ and α are not. As expected, our 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.

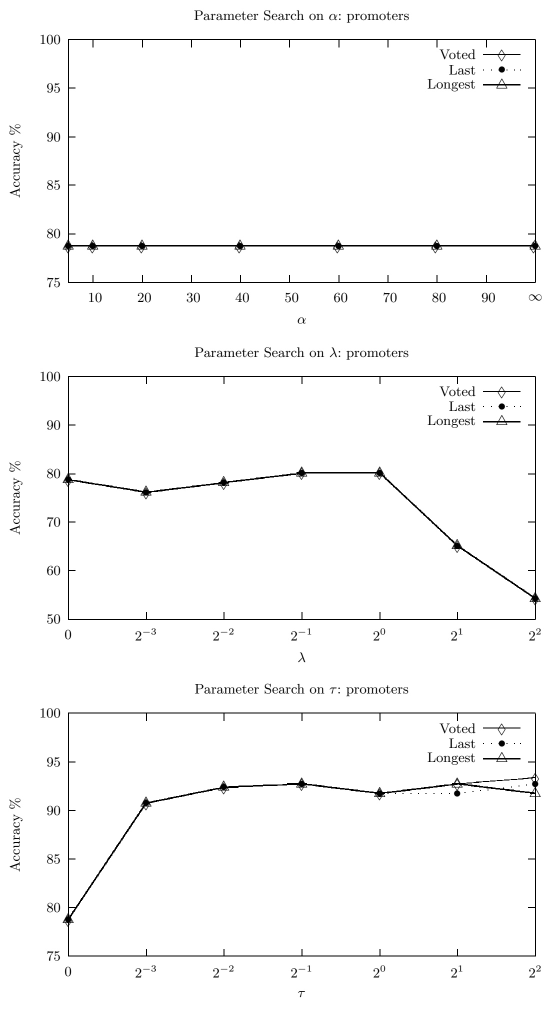

Fig. 6. Parameter Search on promoters Dataset from UCI

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 dataset 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 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 value for $\tau = 0$ is included in other columns.

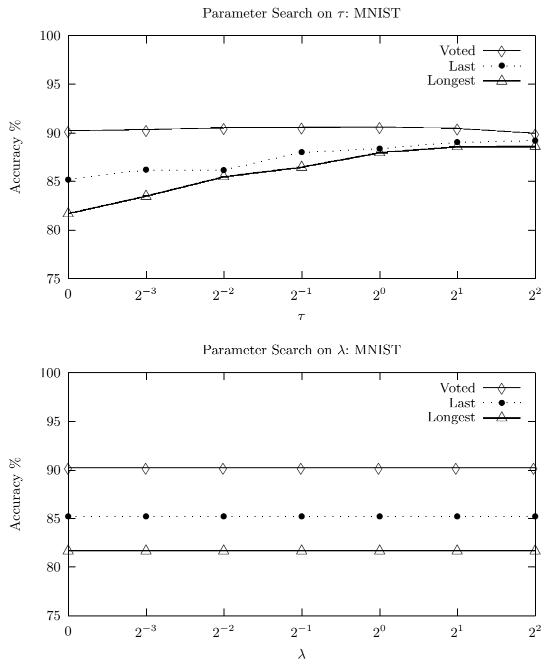

Fig. 7. Parameter Search on MNIST Dataset

Several things can be observed in the tables. First as mentioned above τ is useful even in datasets with noise; this is obvious both in the artificial datasets with noise and in the UCI datasets, all of which are inseparable in the native feature space. Second, differences between the basic algorithm, the longest survivor and the voted perceptron are noticeable without additional variants. For the artificial datasets this only holds for one group of datasets ($f = 50$), the one with highest ratio of number of examples to number of features ($12 : 1$). [4] The longest survivor seems less stable and has lower performance in some cases. Third, for the artificial datasets using τ alone (with last hypothesis) gives higher accuracy than using the voted perceptron alone. For the UCI, "MNIST," and "USPS" datasets this trend is less pronounced; while τ always helps the last hypothesis, it only occasionally helps voted, and sometimes hurts it. In all datasets, while α and λ do improve performance in a number of cases, they are less effective in general than τ, and do not provide additional improvement when combined with τ.

The results for the artificial data seem to suggest that the voted perceptron is only likely to help in noisy data and when the ratio of numbers of examples to

[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.

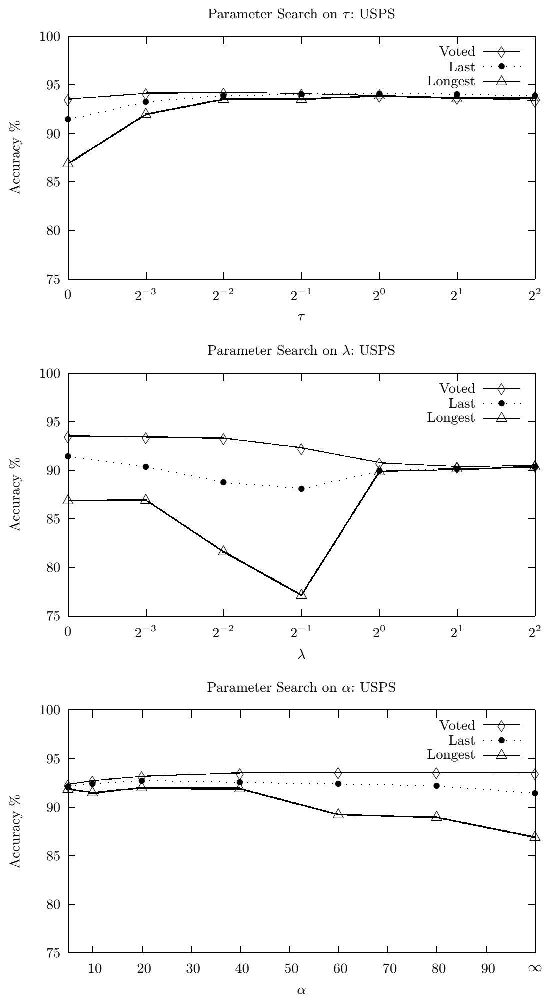

Fig. 8. Parameter Search on USPS Dataset

features is sufficiently high. On the other hand the τ variant seems to be useful even with low ratios of examples to features. To explore this hypothesis, we ran a set of experiments using the "Adult" dataset. We measure performance when training with 10, 100, 1000, 5000, 10000 and 20000 examples. The results for several values of τ are shown in Table 5. The table contradicts the observation from the artificial data. While τ continues to help everywhere, we see that with a small number of examples (e.g. 100) the voted perceptron performs as well or better than the τ variant.

The table also adds a fifth observation about stability of the algorithms. Note that since we report results for concrete values of τ we can measure the standard

		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 Datasets

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 datasets discussed above and is an important feature of these algorithms.

3.5 Parameter Optimization

We have run the parameter optimization on "USPS," "MNIST," the UCI datasets, and the artificial datasets. 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 in

Noise Pctg. (N)	Nothing	τ	λ	α	$\tau \times \lambda$	$\tau \times \alpha$
$f = 50, M = 0.75$						
Last, $N = 0$	97.5	100	97.32	97.5	100	100
Longest	97.5	100	97.32	97.5	100	100
Voted	97.5	100	97.32	97.5	100	100
Last, $N = 0.05$	80.29	93.79	91.51	92.78	93.79	94.21
Longest	86.13	93.58	91.51	91.32	93.58	94
Voted	87.58	93.58	91.51	92.78	93.58	94
Last, $N = 0.1$	80.13	88.39	86.92	86.13	88.39	89.01
Longest	79.88	88.18	86.92	86.53	88.39	88.82
Voted	82.58	88.8	86.92	86.13	88.8	89.01
Last, $N = 0.15$	70.28	82.59	80.1	80.52	82.59	83.64
Longest	76.78	81.14	80.1	80.31	83.83	83.23
Voted	78.24	83.63	80.1	80.52	83.84	83.64
Last, $N = 0.25$	63.08	68.9	67.43	66.18	68.9	69.1
Longest	58.73	68.67	67.43	64.53	70.33	69.92
Voted	65.35	69.3	67.43	66.18	69.71	69.71

Table 4. Parameter Search on Artificial Datasets

[20]. No confidence interval is given for "USPS," "MNIST," or "Adult," as the outer cross-validation loop is not performed.

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.

In Tables 6 and 7 columns of parameter variants do include the inactive option. Hence, in contrast with the tables for parameter search, 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 datasets. 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.

Tables 6 and 7 show improvement over the basic algorithm in all datasets 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 ob-

# examples:	10		100		1000		5000		10000		20000	
	Accuracy	Std. Dev.										
$\tau = 0$												
Last	77.1	1.8	75.9	3.2	76.6	10.1	79.4	3.9	78.8	4.7	81.1	2
Longest	77.3	1.7	78.2	1.4	80.9	1.8	82.3	0.7	81.9	1.4	82	2.3
Voted	77.1	1.7	78.7	1.3	82.7	0.3	83.9	0.2	84.3	0.2	84.5	0.2
$\tau = 0.01$												
Last	76.6	2.1	71.3	10.6	78.5	3.8	79.3	2.8	79.9	3.2	80.8	1.8
Longest	77.1	1.6	78.2	1.8	81	2.5	82.1	0.8	82.6	0.4	82.5	1
Voted	77	1.6	78.9	1.3	82.8	0.2	83.9	0.2	84.3	0.2	84.6	0.2
$\tau = 0.1$												
Last	76	2.8	75.3	6.4	78.9	6.5	80.7	2.4	81.8	2	81.7	1.9
Longest	76.4	2.1	76.9	3.9	80.7	1.9	82.2	1.4	82.6	0.8	82.4	1.7
Voted	76.5	2.1	79.2	1.3	82.7	0.33	83.9	0.1	84.3	0.2	84.5	0.2
$\tau = 1$												
Last	75.3	1.7	77	3.4	81.4	2	83	0.9	83.3	0.8	83.9	0.4
Longest	75.9	0	76.4	1.5	78.8	2.9	81.6	2.5	81.1	2.8	81.4	2.3
Voted	76	0.33	77.4	1.7	82.6	0.3	83.6	0.2	83.9	0.2	84.2	0.2
$\tau = 2$												
Last	76.1	0.5	77.4	1.7	82.3	0.6	83.1	0.5	83.4	0.9	84.1	0.2
Longest	75.92	0	75.9	0.01	77.5	2.5	81.5	2.3	82.3	2.2	81.2	2.8
Voted	70.7	15.6	75.9	0.04	82.3	0.4	83.4	0.2	83.7	0.2	84	0.2
$\tau = 4$												
Last	76.1	0.5	75.9	0.03	82.2	0.5	83.1	0.4	83.5	0.5	83.9	0.2
Longest	75.92	0	75.9	0.01	75.9	0.04	80.5	3.1	80.3	3.4	81.9	3.1
Voted	70.7	15.6	75.9	0.01	80.2	1.23	83.2	0.2	83.5	0.2	83.7	0.2

Table 5. Performance on "Adult" dataset as a function of margin and training set size

served 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. Typical results in the literature use higher degree polynomial kernel on the "MNIST" and "USPS" datasets. Table 7 includes results using τ with a degree 4 polynomial kernel for these datasets. 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.

Table 6 also gives results for SVM. We have used SVMlight [14] and ran with several values for the constants controlling the L_1 soft margin. For L_1 optimization the values used for the "-c" switch in SVMLight are $\{10^{-5}$, 10^{-4}, 10^{-3},...,$10^4\}$. 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 essentially upper bounds on the performance of SVM on these datasets. As can be seen the perceptron variants give similar accuracies and smaller variance and they are therefore excellent alternative for SVM.

These experiments distinguish the voted perceptron, the τ variant and their combination as the best potential algorithm in our suite.

Table 6. Parameter Optimization Results for UCI and Artificial Datasets

	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=200,N=0.05,M=0.75										
Last	80.1 +/- 3.6	84.1 +/- 4.0	80.1 +/- 3.6	83.4 +/- 4.4	83.9 +/- 4.1	86.8 +/- 4.4	82 +/- 4.7	84.9 +/- 4.6		
Longest	80.1 +/- 3.6	83.5 +/- 3.3	80.1 +/- 3.6	81.6 +/- 3.8	84.7 +/- 4.1	87.1 +/- 3.5	83.3 +/- 3.4	84.1 +/- 3.8		
Voted	80.1 +/- 3.6	82 +/- 3.5	80.1 +/- 3.6	83.4 +/- 4.4	85.6 +/- 4.2	87.3 +/- 4.3	82.2 +/- 3.5	84.3 +/- 4.4		

	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 Datasets

4 Summary 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 comes second, 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 datasets, and the two are indistinguishable in their performance on the UCI data. The experiments also show that the soft-margin variants do not provide additional improvement in performance.

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. 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 penalty in run time for optimization.

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 datasets were performed using the primal form of the algorithms since the dual form is too slow. In practice, with a large dataset one can afford to 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.

Our work raises several interesting questions for further work. Finding a theoretical explanation for the success of the margin variant on noisy data is an important problem. The proofs in [8] and [23] provide mistake bounds for the noisy case. But they do not distinguish between the classical perceptron and perceptron using margin and therefore they do not resolve this question. Notice that we cannot run a maximum margin algorithm to completion in such a case without using a soft margin heuristic whereas the perceptron algorithm performs quite well in this case. One potential explanation is that the fact that we are performing a bounded number of iterations bounds the effect of every example, similar to the soft-margin α variant. However, the fact that the α variant did not provide any additional consistent improvement does not support this explanation. Concerning the longest survivor, Gallant [10] has experimented with the "ratchet" variation that re-evaluates each hypothesis on the entire dataset and picks the best one according to this measure instead of the original streak of good predictions. It would be interesting to see if this would stabilize the algorithm better.

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.

References

[1] 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.

[2] N. Christianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000.

[3] W. Cohen. Fast effective rule induction. In *Proceedings of the International Conference on Machine Learning*, 1995.

[4] 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.

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

[6] 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*, 1996.

[7] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.

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

[9] T. Friess, N. Cristianini, and C. Campbell. The kernel adatron algorithm: a fast and simple learning procedure for support vector machine. In *Proceedings of the International Conference on Machine Learning*, 1998.

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

[11] A. Garg and D. Roth. Margin distribution and learning algorithms. In *Proc. of the International Conference on Machine Learning (ICML)*, 2003.

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

[13] T. Graepel, R. Herbrich, and R. Williamson. From margin to sparsity. In *Conference on Neural Information Processing Systems*, 2000.

[14] 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.

[15] 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, University of California, Irvine, June 1987.

[16] A. Kowalczyk, A. Smola, and R. Williamson. Kernel machines and boolean functions. In *Conference on Neural Information Processing Systems*, 2001.

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

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

[19] 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.

[20] Tom. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

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

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

[23] 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*, 2005.

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

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