Scaling large margin classifiers for spoken language understanding

Patrick Haffner

*AT&T Labs - Research*
200 Laurel Ave S, Middletown, NJ 07748

Abstract

Large margin classifiers, such as SVMs and AdaBoost, have achieved state-of-the-art performance for semantic classification problems that occur in spoken language understanding or textual data mining applications. However, these computationally expensive learning algorithms cannot always handle the very large number of examples, features, and classes that are present in the available training corpora. This paper provides an original and unified presentation of these algorithms within the framework of regularized and large margin linear classifiers, reviews some available optimization techniques, and offers practical solutions to scaling issues. Systematic experiments compare the algorithms according to a number of criteria: performance, robustness, computational and memory requirements, and ease of parallelization. Furthermore, they confirm that the 1-vs-other multiclass scheme is a simple, generic and easy to implement baseline that has excellent scaling properties. Finally, this paper identifies the limitations of the classifiers and the multiclass schemes that are implemented.

*Key words:* Spoken Language Understanding, Machine Learning, SVM, AdaBoost, Maximum Entropy

1 Introduction

Since the successful demonstration of the “How May I Help You” (AT&T HMIHY(SM)) spoken language understanding system (Gorin et al., 1997), significant efforts have been invested in the automation of customer care centers. The natural language understanding stage of the process implies classifying the output of the Automatic Speech Recognition (ASR) system into semantic

*Email address: haffner@research.att.com* (Patrick Haffner).
entities that will be interpreted by the dialog manager. For instance, in a deployed customer-care application, users interact with a spoken-dialog system via the telephone, speaking naturally, to ask about their bills, require a refill on a medication, ask for a change in service, or other topics depending on the application. Their responses to the open-ended prompts of the system are not constrained by the system, they may be any natural language sequence. The first objective of the spoken-dialog classification is to assign one or several categories or call-types, e.g., Pay Bill, or Account Balance, to the users’ speech utterances. After having recognized the intent of the caller, the utterance often contains necessary information to complete the request, such as dates, names, amounts or locations. This stage, known as slot-filling or named entity recognition, can also rely on classification, at least at the named entity detection stage (Levit et al., 2004).

Available pattern recognition methods to perform classification include generative approaches, where the classifier learns the class densities, and discriminative approaches, which focus on learning the class boundaries (Ng and Jordan, 2002). Generative approaches include Naive Bayes methods, and more generally any Maximum Likelihood (ML) modeling. The first classifiers (Gorin et al., 1997) relied on generative approaches, as they allow a stronger coupling with the statistical language modeling techniques used in the last stage of the ASR. However, experimental comparisons show that generative approaches are outperformed by discriminative approaches (Tang et al., 2003; Haffner et al., 2003) and need to be modified in a discriminative way to yield good results (Chelba and Acero, 2003).

One reason may be that generative approaches critically rely on the choice of the features which are used to characterize the input data and on the assumptions about the production of these features by the generative model. At the same time, research on the optimal features for a spoken language understanding system is still a work in progress, with a current state-of-the art that seems to leave a lot of room for improvement. At a matter of fact, it is very hard to outperform the \( n \)-gram features used in currently deployed systems, where most of the temporal information is lost. Discriminative approaches have recently been very popular and spoken language understanding systems suitable for deployment (Tang et al., 2003; Haffner et al., 2003; Tur et al., 2004) mostly consist of a discriminative semantic classifier. The latter can be seen as a “black box” that takes as input a very large feature vector consisting of every possible \( n \)-gram (sequence of \( n \) consecutive words) and produces as output a detection score for each semantic entity (call-type in the language of customer care center). The number of such call types can be quite large, sometimes measured in hundreds.

Some of the most successful discriminative classification approaches are regularized linear classifiers, often called “large margin”, such as Support Vector
Machines (SVMs) (Vapnik, 1998) or AdaBoost (Freund and Schapire, 1996). The loss function that is optimized includes an explicit or implicit (in the case of AdaBoost) regularization term that enforces a large margin of separation between the classes in the training data. This property increases the probability of better generalization on test data. Another advantage of these classifiers is that the training process is convex optimization, thus guaranteed to converge in a predictable time. The fact that no hand-crafting of rules, features or models is needed, the robustness of learning convergence, and the quality of generalization make make these regularized linear classifiers ideal candidates for tasks where minimum human intervention is desirable. SVMs (Joachims, 1998b) and AdaBoost (Schapire and Singer, 2000) are regarded as reference methods for topic classification and spoken language understanding, and are currently deployed in the AT&T VoiceTone® spoken dialog system.

Since the original work on “How May I Help You”, spoken language understanding systems have become increasingly sophisticated and the use of more and more information causes major scaling issues for classifiers such as SVMs and AdaBoost. The following scaling problems can be encountered:

**Large number of classes.** The building up of user-experience expertise enables much more ambitious systems with sophisticated dialog modeling and fine-grain distinction between call-types. Combined with labeling automation (Tur et al., 2003) and quality assurance tools (Begeja et al., 2005), this expertise makes it possible for a specific IVR (Interactive Voice Response) service to define many different call types that can be reliably learned (typically more than 100 call types (DiFabbriazio et al., 2004)).

**Large number of examples.** While there are logistical limits to the amount of data one can collect for a given IVR task, the re-use of data between different tasks and the definition of common call-types (DiFabbriazio et al., 2004) make it possible to gather hundreds of thousands of training examples.

**Large number of alternative input representations.** Many ASR systems are now able to produce more alternatives than the best recognized sentence. They can output lattices that represent countless possible recognized sentences. This scaling issue can be addressed in a generic and elegant manner by defining kernels over weighted graphs(Cortes et al., 2004). Compared to results obtained with the best sentence output by the ASR, the computational overhead is very manageable. The experiments provided by Cortes et al. use the exact same datasets as the ones used in this study, therefore, we will not present new results on this scaling issue. Word confusion networks are approximate compact representations of lattices, and allow an even smaller computational overhead (Tur et al., 2002).

**Large number of features.** Using more data also implies a larger vocabulary and many more features. Because the average number of non zero features per example is low, we have a sparse representation that, as shown below, scales very well. However, the existence of examples with more fea-
tures than this average, as they represent very long sentences or a succession of sentences, is a major challenge. It should be noted that the experiments reported in this paper do not use dialog-level contextual, syntactic and semantic information, which could add a large number of new features.

Can learning technology adapt to these scaling issues? The run time of a deployed system should be on the order of milliseconds. A manageable learning time should be in the order of hours, with memory requirements not exceeding a few GigaBytes. To remain within these constraints, one is often led to choose sub-optimal solutions either by restricting the data or by using methods with lower classification performance but better scaling properties.

This paper offers efficient scaling solutions for many of these issues. It reviews existing solutions and offers new ones, supported by extensive experiments on large scale data. Section 2 generalizes large margin classifiers as regularized linear classifiers. Section 3 compares multiclass extensions to these algorithms with a few preliminary experiments. Building on the observations made on multiclass algorithms, Section 4 specifies an efficient learning software implementation and proposes parallelization strategies. Section 5 presents the datasets used for the full-scale experiments toward a very large number of examples, which are described in Section 6. A major challenge awaiting large margin classifiers for spoken language understanding is identified in Section 7, namely to improve their ability to handle variable-length sequences.

2 Large Margin and Regularized Linear Classifiers

This section offers a brief overview of regularized linear classifiers, their optimization process, which we call learning, and its efficient implementation. We focus on three classes of algorithms: two are very well known, SVMs (Vapnik, 1998) and AdaBoost (Schapire and Singer, 1999), and one was just published in 2004, Regularized Maximum Entropy (Dudik et al., 2004). This Sequential L1-regularized Maxent algorithm will be referred as SL1-Max in the rest of the paper.

These algorithms were chosen because of the scalability of their learning process and the fact that their runtime implementation can be very efficient. They give us three ways to train the same linear classifier using very different frameworks. Rather than detailing these frameworks, which are extensively described in the references given in the previous paragraph, we focus on a common interpretation that will make comparison of their implementations and scaling properties much easier.

Here are some of the traditional interpretations of these algorithms:
**SVMs or Support Vector Machines** look for the separating hyperplane with the maximum separation margin between two classes, as shown in Figure 1.

AdaBoost incrementally refines a weighted combination of weak classifiers. In the process, the importance of examples that are still erroneous is “adaptively boosted”.

**Maximum Entropy** looks for a distribution over training samples with maximum entropy that satisfies a set of user-defined constraints. Regularization implies that we allow some slack in the satisfaction of these constraints.

These interpretations offer three different procedural approaches to implementing regularized linear classifiers: geometric, based on boosting, and probabilistic. However, if we define our input space as a feature set where each feature corresponds to (i) a dimension in the SVM separation space, (ii) a weak classifier used by AdaBoost, and (iii) a constraint imposed to SL1-Max, we obtain in the three cases a linear classifier $f_w(x) = w^T x$ where the trainable weight vector $w$ lives in a space with one dimension per possible input feature. Such a linear classifier transforms a vector of input features $x$ into a single binary output $y \in \{-1, 1\}$, which is obtained by taking the sign of a single numerical function $y = \text{sign}(f_w(x))$. This scheme can handle two classes only; more classes will require some form of output code which will be considered later.

Given a set of $M$ training input vectors $X = \{x_1, \ldots, x_M\}$ and their corresponding target labels $Y = \{y_1, \ldots, y_M\}$, the loss function we want to minimize is a combination of 2 terms

$$\mathcal{L} = \sum_{i=1}^{M} \mathcal{C}(y_i w^T x_i) + \sum_{k=1}^{N} \beta_k |w_k|^p \tag{1}$$

- The error term $\mathcal{C}(y_i w^T x_i)$ accounts for the error on the training sample $x_i$ with respect to the targets $y_i$. For good generalization, it is not enough to properly classify training examples and to verify, for each training pair

---

1 The addition of a bias will be discussed in Section 2.1.3
(x, y), the inequality \( yw^T x > 0 \). This error term is a direct function of the *sample margin* \( yw^T x \) and makes it as large as possible, so that a small perturbation added to \( x \) will not impact the classification result.

- The regularization term \( R = \sum_{k=1}^{N} \beta_k |w_k|^p \) measures the complexity of the model and encourages simple models that cannot over-train. The regularization parameters \( \beta_k \) are part of the meta-parameters, or design parameters. When \( \beta_k = 1 \), then \( R \) corresponds to the \( l_p \)-norm of the weight vector \( \left( \sum_{k=1}^{N} |w_k|^p \right)^{\frac{1}{p}} \).

Our definition of the error and regularization terms rely on two concepts that are critical for good generalization on test data: large margin and simplicity. Can we quantify these concepts and turn them into guarantees of reasonable generalization? One of the main breakthroughs in statistical learning theory was the introduction of worst case bounds for the generalization error (Vapnik, 1998; Cristianini and Shawe-Taylor, 2000). SVMs were the first classification algorithm for which such bounds were proposed (Vapnik, 1998). The bound on the generalization error is minimized when the *separation margin* \( \rho > 0 \) such that \( \frac{\|w\|^2}{\|w\|} \geq \rho \) is maximized. The norm \( \|w\| \) is a normalizing factor that ensures that the margin is not simply caused by a large weight vector. If it is the Euclidean norm, one can verify that \( \rho \) corresponds to the separation margin between the two classes shown in Figure 1. More generally, \( \|w\| \) is a weighted \( l_p \)-norm which scales as the regularization term \( \|w\| = \left( \sum_{k=1}^{N} \beta_k |w_k|^p \right)^{\frac{1}{p}} \). The example of SVMs shows how to obtain generalization bounds when (i) the error maximizes the sample margins \( yw^T x \) and (ii) the regularizer minimizes the weight norm \( \|w\| \). Deriving similar generalization bounds for other types of classifier is the subject of ongoing research.

Giving the loss function is only half of the problem. We also need to define the learning process that looks for the weight \( w \) minimizing the loss. A common characteristic of the best optimization algorithms is that they are easy to formulate into simple *sequential* algorithms, which only look at a small number of features or examples at a time.

From a practical viewpoint, large margin classifiers can be divided into two broad categories, whose choice depends on the goals we want to achieve:

- Feature-based optimization methods add features to the weight vector. It is desirable that only a few features are necessary to express the solution; this sparseness property encourages the use of the \( l_1 \)-norm in the regularization term. The first of these methods was AdaBoost (Freund and Schapire, 1996). It inspired many other algorithms, including arcing (Breiman, 1999), LogitBoost (Friedman et al., 1998) and SL1-Max (Dudik et al., 2004). SL1-Max and AdaBoost are described in Section 2.1. Feature-based methods have been popular in natural language applications, as expert users often
like to track which features are selected by the learning process.

- Example-based optimization methods enrich, at each iteration, the weight vector with information coming from a few examples. This is a typical optimization technique used for SVMs. Provided that the we use the $l_2$-norm in the regularization term, this optimization allows kernel extensions, which is the main reason for using it. After the presentation of SVMs in Section 2.2, Section 2.3 presents further analysis of kernel methods. Kernel-based methods have a richer representation power, but are less tractable by an expert user.

The rest of this section is more technical. It first shows how the large-margin algorithms we consider fit in the regularization framework with different error and regularization terms. We then discuss what type of optimization solution is allowed by the form of the loss function. This allows us to predict how the training time of each algorithm will scale with the size of the task.

### 2.1 AdaBoost and Maximum Entropy

#### 2.1.1 AdaBoost

Boosting algorithms are voting algorithms, where the final classification is a weighted combination of simpler or weak classifiers.

AdaBoost (adaptive boosting) was introduced as a sequential algorithm (Freund and Schapire, 1996). In our framework, where a weak classifier corresponds to the simple detection of a feature, AdaBoost selects at each iteration a feature $k$ and finds, analytically or using a line search algorithm, the optimal weight $w_k$. In the process, the importance of examples that are still erroneous is “adaptively boosted”, as their weight in a distribution over the training examples that is initially uniform is increased. It was only later that this greedy sequential optimization process was shown to converge and to be more efficient than other methods minimizing the same loss function (Collins et al., 2000). The original AdaBoost sequential algorithm minimizes the exponential loss; therefore, the error term for example $i$ is

$$C_{\text{boost}}(y_i w^T x_i) = \exp(-y_i w^T x_i)$$

(2)

AdaBoost has been shown to be a large margin algorithm, both in theory, with generalization error bounds based on $l_1$-norm margin (Schapire et al., 1997), and in practice, as, over iterations, learning concentrates on examples with the smallest margin. However, no explicit regularization term appears in the loss function. Extensions to AdaBoost where one can control the width of the margin and the regularization are complex; they require hypothesizing
the width of the margin in advance (Ratsch and Warmuth, 2002).

2.1.2 SL1-Max

SL1-Max is a new regularization scheme and learning algorithm for Maximum Entropy (Maxent) modeling. This regularized linear algorithm (it is more than a classifier) relies on feature-based optimization. Its framework offers a superior way of understanding and controlling generalization, as well as a very straightforward optimization algorithm.

The initial constrained optimization problem consists in finding the distribution over training samples with maximum entropy that satisfies a set of feature-level constraints. Using convex duality, we obtain as a loss function the regularized Maximum Likelihood. The optimization problem is applied to a Gibbs distribution, which is exponential in a linear combination of the features:

\[ P(x) = \frac{\exp(w^T x)}{Z} \]  

with \( Z = \sum_{i=1}^{M} \exp(w^T x_i) \). A description of the derivation of this function from the constrained optimization problem is beyond the scope of this paper and can be found elsewhere (Dudik et al., 2004; Haffner et al., 2005a).

There are several ways to represent a classification problem as likelihood maximization. Maxent applications to classification problems (Nigam et al., 1999) usually model the conditional \( P(y|x) \) distribution, but can also model the joint \( P(y, x) \) distribution. In theory, Maxent requires one weight vector per class. For instance, in our binary classification problem, conditional Maxent probabilities for the positive and negative examples are:

\[ P(y = +1|x) = \frac{\exp(y(w_+)^T x)}{Z(x)} \]

\[ P(y = -1|x) = \frac{\exp(y(w_-)^T x)}{Z(x)} \]

Noting that \( Z(x) = \exp(y(w_+)^T x) + \exp(y(w_-)^T x) \), conditional Maxent is equivalent to the logistic classifier, where a sigmoid function is applied to a linear classifier with weight \( w = (w_+) - (w_-) \). The error term is

\[ C_{logit}(y_i w^T x_i) = \log(1 + e^{-2y_i w^T x_i}) \]  

In the joint distribution case, the formulation of the error term as a function of the sample margin is not as straightforward. However, Section 3.2 will show that the joint and the conditional Maxent models yield very similar performance.

Differing from the usual Gaussian regularization (Nigam et al., 1999) that
results into a $l_2$-norm, the regularizer is the $l_1$-norm of the weight vector

$$R = \sum_{k=1}^{N} \beta_k |w_k|. \quad (5)$$

Compared to the $l_2$-norm, the $l_1$-norm encourages sparse solutions where a large number of weights are zero. The choice of $\beta$ is critical for good performance. Based on the results of statistical analysis, Dudik et al. hypothesize that $\beta_k$ should scale as $\sigma[x_k]/\sqrt{m}$, where $\sigma[x_k]$ is the standard deviation of $x_k$ under the true distribution. They reduce $\beta_k$ to a single regularization parameter $\beta$ with $\beta_k = \beta \sigma[x_k]/\sqrt{N}$. Experiments in Section 6.3 will discuss optimal values for $\beta$.

There are only very few theoretical studies of maxent generalization bounds (Dudik et al., 2004), and, to our knowledge, no studies of conditional Maxent as a large margin classifier. It should be noted that conditions for the existence of a large margin bound are present: the error term in Eq. (4) encourages large sample margins, and its regularization term is the $l_1$-norm of the weight.

2.1.3 The critical bias term

AdaBoost has been interpreted as a form of Maxent where the exponential distributions are not normalized as in Eq. (3) (Lebanon and Lafferty, 2002). However, it is possible to define a logistic version of AdaBoost (Collins et al., 2000) that minimizes the same error term as in Eq. (4). Our experiments show that while logistic AdaBoost is very similar to exponential AdaBoost, it is still very different from conditional Maxent. To understand this difference, imagine a feature with a constant value. In the case of Maxent, the weight corresponding to this feature is learned as a bias and the presence of this feature is critical for good performance. However, AdaBoost only learns features that produce some form of classification and cannot learn a constant feature. This absence of learnable bias may be the reason why AdaBoost with basic feature detectors as weak classifiers performs poorly in Section 6.1. However, a modified version of AdaBoost where weak classifiers can assign confidences to each prediction (Schapire and Singer, 1999) brings AdaBoost to a level of performance very similar to unregularized Maxent. This confidence-rated version of AdaBoost is equivalent to adding a learnable bias to each weak classifier and will be referred to as AdaBoost-C in the rest of the paper.

2.2 Support Vector Machines

SVMs look for the separating hyperplane with the maximum separation margin between two classes, as shown in Figure 1. The margin is the minimum
distance between the projections of the points of each class on the direction of the weight vector. This figure represents hard margin SVMs, where classification errors on the training set are not permitted. In the rest of the paper, they are generalized as soft margin SVMs, which allow some errors, i.e. vectors that are inside or on the wrong side of the margin.

The most common error term is the hinge loss function

\[ C_{SVM}(y_i w^T x_i) = \max \left(0, 1 - y_i w^T x_i \right). \]  

(6)

It implies that a linear loss occurs as soon as sample \((x_i, y_i)\) is not classified beyond the margin, i.e., \(y_i w^T x_i \geq 1\). A quadratic loss, which corresponds to the mean square error, is also possible. The regularizer is the \(l_2\)-norm of the weight vector and is usually parameterized with a single constant \(C\): \( R = \frac{1}{C} \sum_{k=1}^{N} w_k^2 \).

The connection with the maximization of the margin is straightforward, with an optimization process that:

- Minimizes regularizer \( R \): maximize the width of the margin
- Minimizes error term \( C \): for a given margin, minimize the number of outliers that are on the wrong side of the margin.

This is a constrained optimization problem that can be solved with convex duality (Vapnik, 1998). The weight vector can be expressed as a combination of training vectors

\[ w = \sum_{i=1}^{M} \alpha_i x_i \]  

(7)

We call support vectors examples for which \( \alpha_i \neq 0 \): they touch the margin or are on the wrong side of the margin. The \( \alpha_i \), called Lagrange multipliers, represent another, or dual, way to parameterize the weight vector. It can be used to compute the classifier output directly:

\[ w^T x = \sum_{i=1}^{M} \alpha_i w^T x_i \]  

(8)

More generally, the entire optimization and learning process has a dual representation when only computations of inner products between input vectors are needed (without ever needing an explicit representation of the weight vector \( w \)).

SVM optimization techniques (Schölkopf and Smola, 2002) are usually referred as Quadratic Programming (QP) and work in the sample space over the \( \alpha_i \) variables. The weight vector (if we do not use a kernel) can be computed as \( w = \sum_{i=1}^{M} \alpha_i x_i \). Sequential strategies can be derived from Zoutendijk's
method (Joachims, 1998a), where one looks for the steepest direction of descent in the loss gradient with only $q$ non-zero elements. Sequential Minimum Optimization (SMO) (Platt, 1998)$^2$ is an implementation with $q = 2$. It optimizes only two $\alpha_i$ at a time, and the updating step has a simple analytical solution. Because of its speed, simplicity and robustness, this algorithm is the reference optimization technique for SVMs and is used in our experiments. It can be combined with many speedup techniques such as shrinking (Joachims, 1998a) or cascading (Graf et al., 2005), which will be described in Section 4.2.

2.3 The kernel trick

This technique is applicable to algorithms such as SVMs where all computations over input vectors can be represented as inner products. In SL1-Max, the fact that the regularizer term uses the $l_1$-norm makes the use of the kernel trick impossible: this $l_1$-norm cannot be expressed as an inner product.

Suppose we apply a non-linear mapping from the original input vector $\mathbf{x}$ to a higher dimensional vector $\Phi(\mathbf{x})$. This high-dimensional vector can for instance consist of all pairwise correlations between elements of $\mathbf{x}$. The computation of the inner product between two high dimensional $\Phi(\mathbf{x})$ vectors may be very costly. One solution to this problem is to use instead a kernel

$$K(x_0, x_1) = \langle \Phi(x_0), \Phi(x_1) \rangle$$ (9)

Kernels may be more efficient to compute because they do not require an explicit computation of $\Phi$ and the inner product. There is some flexibility in the choice of the kernel function $K$, provided that there exists a mapping $\Phi$ such that Eq. (9) holds. Functions $K$ that verify this condition must be positive definite symmetric (Schölkopf and Smola, 2002).

SVMs are not the only approach to implement kernel-based regularized classifiers. A simple extrapolation on SVMs is regularized logistic regression, which also establishes the link with SL1-Max. As a matter of fact, this extension shows that logistic regression converges asymptotically toward the hinge loss, with very similar performance on text classification (Zhang et al., 2003) and other tasks (Keerthi et al., 2002). The drawback of the logistic regression loss is that its kernel implementation results in solutions that are much less sparse (in terms of support vectors) than the ones using the hinge loss (Keerthi et al., 2002).

$^2$ In its most general form, SMO does not specify how the 2 working examples are selected. Choosing the two with steepest gradient is the most common strategy, which combines efficiency and simplicity.
Bayesian inference (Schölkopf and Smola, 2002) explains the cost term as a likelihood to observe the data $\log P(X, Y|w)$ and the regularization term as the log probability from the prior distribution of the weight vector $\log P(w)$ (it accounts for the prior knowledge we have about the weight distribution). A Gaussian prior leads to a $l_2$-norm regularization term, while a Laplacian prior gives a $l_1$-norm regularization term. Successful kernelizable Bayesian approaches are Gaussian processes (Gaussian prior) and Relevance Vector Machines (Tipping, 2000) (conditional Gaussian prior). Unfortunately, Bayesian approaches do not lead to the type of efficient optimization algorithms described in the next section. Bayesian inference could be a good candidate for unifying large margin classifiers. While SVMs, AdaBoost or SL1-Max do not seem to exactly fit in this Bayesian inference framework, they can be seen as approximations that provide much more efficient optimization algorithms.

2.4 Optimization Complexity

As shown above, the learning process looks for the weight $w$ that minimizes the loss. This optimization can be applied over the feature space (where $w$ lives) directly, or, in the case of SVMs, over the training sample space (where the Lagrange multipliers $\alpha$ live).

A common characteristic of the best optimization algorithms that have been studied for these loss functions is that they are coordinate-wise, modifying one or only a few parameters (either feature weight or support vector Lagrange multiplier) at a time. They are also easy to formulate into simple sequential algorithms. The success of these sequential algorithms may also be due to the type of application we deal with (text classification), where the vectors are extremely sparse. In the case of AdaBoost, Collins et al. show that, with sparse vectors, algorithms which optimize features sequentially converge significantly faster than algorithms which optimize all the features in parallel.

The sequential optimization algorithms proposed for AdaBoost (Collins et al., 2000) and SL1-Max (Dudik et al., 2004) include two steps:

1. Look for the feature that maximizes the variation in the loss (worst case complexity $O(MN)$).
2. Update the estimated distribution over the training samples (worst case complexity $O(M)$).

These worst case complexity figures are pessimistic, especially with sparse data. Moreover, the number of iterations depends on a stopping criterion that is heuristically defined.

SVM QP optimization complexity has been more extensively studied, and
found to lie between $O(M^2)$ and $O(M^3)$ (see (Bordes et al., 2005) for a discussion). Bordes et al. derive the $O(M^2)$ lower bound from the simple verification that, at the end of the optimization process, the set of Lagrange multipliers $\alpha$ is a solution of the SVM QP problem. This verification requires the computation of $MS$ kernels, where $S$ is the number of support vectors. If the best possible error rate is non-zero, $S$ is asymptotically proportional to $M$ (Steinwart, 2004).

Complexities lower than $O(M^2)$ can be achieved, but with strings attached. In the case of linear SVMs, primal (or feature based optimization) would be possible. For instance, Lagrangian SVMs (Mangasarian and Musicant, 2001) transfer most of the computation to the inversion of a $N \times N$ matrix in the $N$-dimensional feature space. The very recent publication of the Core SVM (Tsang et al., 2005) and LASVM (Bordes et al., 2005) implementations show that techniques based on probabilistic, on-line or active sampling can produce dramatic speedups. While both methods are clearly described as approximations, their accuracy matches the so-called “exact” SVM methods. As we will see in Section 4.2, parallelization can also be considered as a technique to break the $O(M^2)$ barrier. It can be applied to the entire architecture (Collobert et al., 2002) or the learning process alone (Graf et al., 2005). However, our experimental results in Section 6.1 suggest that SVM after parallelization still are $O(M^2)$.

The data used in this paper is extremely sparse, which means that only a small proportion of the features are non-zero. This results in speedup algorithms at every level of processing. The most efficient of such algorithms is partial pricing applied to Adaboost and SL1-Max (Phillips and Haffner, 2005). The impact of these algorithms is complex and highly task dependent. It is also hard to set up a meaningful experiment that would measure how training speed scales with the number of features. As a consequence, we were not able to quantify the dependency between training time and the number of features.

2.5 Stopping Criteria

Deciding when to stop an optimization process is always an important decision. It is usually desirable to stop as early as possible to save time, to prevent the model from growing too large, and to avoid overfitting. After extensive research on how to stop training on Neural Network architectures, the common practice is to measure the error on a development dataset and stop when this error reaches a minimum. This procedure has some drawbacks: it reduces the amount of data available for training, it is time consuming because of the looking ahead required, and it can be inaccurate if the development set is too small.
One of the reasons for the success of SVMs is that they made the use of such a procedure unnecessary, as they hardly ever overfit and offer easy to implement stopping criteria, purely based on measures on the training data. Most SVM training packages (Joachims, 1998a; Collobert and Bengio, 2001), which work in the dual space, stop the optimization process when the constraints over the dual variables are satisfied within a given gap. Geometrically, this duality gap would represent the distance between a support vector and the margin it is supposed to support (the width of the margin being normalized to 2). SVM practitioners have found that, for most problems, values between 0.01 and 0.001 guarantee a convergence to the minimum error rate.

With AdaBoost, when to stop has always been an issue. As some overfitting is observed with AdaBoost on noisy data, stopping early is important. The use of a development dataset is the preferred technique. For instance, the AdaBoost software Boostexter (Schapire and Singer, 2000) encourages the use of such a development set to track the “test” error at each iteration.

3 Multiclass learning

The search for the most efficient multiclass classification architecture is an open problem; there is a lack of experimental evidence clearly pointing toward one approach. This section briefly reviews the current literature. It suggests that the 1-vs-other architecture chosen in this paper is reasonable. However, in the future, improved algorithms or data labeling processes may make the choice of another architecture more attractive.

Large margin classifiers were initially demonstrated on binary classification problems, where the definition of the margin is unambiguous and its impact on generalization is reasonably well understood. The simplest multiclass classification scheme is to train $C$ binary classifiers, each trained to distinguish the examples belonging to one class from examples not belonging to this class. The situation where each example can only belong to one class leads to training one class versus all other classes. That is why this scheme is usually referred to in the literature as 1-vs-other or 1-vs-all. Many other combinations of binary classifiers are possible, in particular 1-vs-1 (also called all-vs-all) where each classifier is trained to separate a pair of classes. More general combination schemes include error correcting codes (ECOC) (Crammer and Singer, 2002) and hierarchies of classifiers.

The problem with methods based on recombination of binary classifiers is that each classifier corresponds to a different optimization problem and that they do not guarantee a global optimality (where a single optimization problem minimizes the classification error that enforces the output of the target class
to be the largest by some margin). Single machine schemes have been proposed for both SVMs (Weston and Watkins, 1998; Crammer and Singer, 2002) and AdaBoost (AdaBoost-MR in (Schapire and Singer, 1999)). Note that in the case of SVMs, a global optimization procedure can be computationally extremely expensive.

A recent overview of most multiclass methods for SVMs (Rifkin and Klautau, 2004) suggests that neither ECOC nor single machine schemes yield significantly higher accuracies than the 1-vs-other scheme. Therefore, Rifkin and Klautau suggests that when there is no natural class hierarchy (or any other form of prior knowledge about the class dependency), the practitioner can use the simpler 1-vs-other scheme. As practitioners, we agree with this recommendation. The few comparative experiments we carried out in the case of SL1-Max (see Section 3.2) confirmed that the best results were obtained with a 1-vs-other architecture. Our multiclass AdaBoost, which relies on independently trained classifiers, is also called AdaBoost.MI (Abney et al., 1999)\(^3\). Section 3.1 shows that using the 1-vs-other scheme, the SVM training time is independent of the number of classes, which is a highly satisfactory scaling property. We currently use a 1-vs-other scheme to implement our multiclass architecture. However, we expect many of the results reported in this paper to also apply to other combinations of independent binary classifiers (which implement other forms of ECOC than 1-vs-other). Another practical advantage of using independent classifiers is that it leads to efficient implementations. The memory required to train all classifiers can be allocated sequentially instead of at once. On parallel architectures, both computing and memory can be distributed over processors.

In the near future, we may find that other combinations of binary classifiers are more efficient than 1-vs-other:

- 1-vs-other may be the simplest scheme, but experiments suggest that in some situations, the 1-vs-1 scheme is faster (see end of Section 3.1). Two implementation issues prevented its systematic use on the very large corpora studied in this paper. First the explosion of the number of classifiers causes a computational overhead, which we have not optimized yet. Second, serious robustness issues arise from the lack of data for some classifiers.
- The problems described in the experimental section (Section 5) have a natural class hierarchy. Using such a hierarchy could also result in significant gains in speed. For instance, a binary hierarchy of classifiers has the same number of classifiers as 1-vs-other, yet much fewer examples in the leaf classifiers. Ideally, this hierarchy should be defined at the same time as the

\(^3\) The single-machine multiclass version of AdaBoost.MI corresponds to AdaBoost.MH. We never observed significant difference in performance between the two implementations of AdaBoost
Fig. 2. log $I_e$ as a function of log $M_c$ after training on half of the VtoneN data (see Section 5), i.e. 289351 examples (all the data would have taken too much time)

labeling process. Automatic, clustering-based procedures to build the hierarchy are also possible. They have been studied only recently (Vural and Dy, 2004) and have not been shown to outperform 1-vs-other.

In a more distant future, single machine schemes may become more efficient and practical:

- Parallelization strategies which efficiently partition the training of a single machine are being developed (Graf et al., 2005).
- In the case of Maxent, Section 3.2 suggests that, in theory, a single machine scheme could work better. The fact that our experimental results favor the 1-vs-other scheme may be due to some imperfection in our modeling hypotheses.
- Most studies focus on the classification error. However, some form of ranking error may be more appropriate for many natural language problems. In this case, single machine schemes are recommended (Aiolli and Sperduti, 2005).
- Section 7 suggests that large margin structured classification approaches (Bartlett et al., 2005), which generalize single machine schemes, could be used to represent time dependencies.

3.1 A multiclass SVM scaling property

Suppose the $M$ training examples are split into $C$ classes, each class $c$ containing $M_c$ examples. Assuming a 1-vs-other decomposition, learning class $c$ is a $M_c$ versus $M - M_c$ samples learning problem. Our observation is that the number of SMO iterations typically scales with the number of support vectors, which itself scales with the number of examples in the least represented class
(\(M_c\) here). Figure 2 shows this prediction to be accurate: the number of SMO iteration \(I_c\) required to learn class \(c\) scales as \(kM_c^{0.9}\), with \(k\) independent of \(c\). If we had \(I_c = kM_c\), the number of SMO iterations, and thus the learning time would be more or less independent of the number of classes. In effect, if \(I_c\) is proportional to \(M_c\), as we have \(\sum_c M_c = M\), the total number of iterations \(\sum_c I_c = kM\) is independent of \(C\).

However, there are several caveats:

- The \(I_c = kM_c\) rule is approximate, and large classes seem to require a lower number of iterations than this prediction, which is why we have this 0.9 exponent.
- A very large number of classes will incur an overhead for just initializing and saving the SVM learning process, not reflected in the number of iterations.
- For the same number of classes, different class frequency distributions may lead to different training times, depending on whether the kernel cache is enough to contain all the kernel computations required by the largest classes or not. This means that a large class may use a lower number of iteration than predicted, but each iteration could involve more kernel computations.

Each SMO iteration adds or modifies two support vectors, thus requiring the computation of \(2M\) kernel products. In the ideal case where all kernel products would have been all cached in advance, most of the computing time for each iteration would be spent searching for the work gradient violators and would require \(2M\) comparisons. In all cases, the time per iteration is \(\tau M\), where \(\tau\) is a time constant. As a consequence, the total training time is \(k\tau M^2 = O(M^2)\). This will be confirmed experimentally in Section 6 \(^4\). Note that while SMO is quadratic in the number of examples, it is far more efficient than Quadratic Programming solvers that tend to scale in \(O(M^3)\).

It is interesting to check the impact of the \(I_c = kM_c\) rule in the case of the 1-vs-1 architecture. We assume classes are balanced, that is \(M_c = \frac{M}{C}\). The number of iterations for each of the \(\frac{1}{2}C(C-1)\) classifiers is \(k\frac{M}{C}\), which gives a total of \(\frac{1}{2}M(C-1)\) iterations. The time per iteration equals \(\tau \frac{2M}{C}\): thus, the total training time is approximately \(k\tau M^2\), which is the same as in the 1-vs-other case. Because it does not require to cache as many kernel products, a faster training speed may be observed when using the 1-vs-1 scheme on computers with limited cache memory.

\(^4\) This is still the case when using shrinking, where well classified examples are removed from the pool of active examples and no longer considered as SMO candidates.
Table 1

TCER (The Top Class Error Rate, our standard measure for classification performance, is defined in Section 5) and memory usage for various SL1-Max configurations on the Vtone2 data. The memory requirements are expressed as a function of the number of classes \( C \) and examples \( M \). The dependency on the number of features is more complex and depends on the task.

### 3.2 Maxent: a probabilistic understanding of multiclass modeling

The probabilistic understanding of Maxent gives us a different perspective on the choice between a single machine or a combination of binary classifiers. This section offers a brief summary of several multiclass extensions to SL1-Max (Haffner et al., 2005a). Both in the case of the joint and the conditional models, two probabilistic assumptions lead to two different types of multiclass modeling:

1. If one assumes that classes occur independently of each other, i.e. the occurrence of two classes is statistically independent, then 1-vs-other models can be trained independently for each class. From a statistical viewpoint, this hypothesis is highly unrealistic, as it implies that no classes can be mutually exclusive. If we use independent models to produce an output code, this suggests that the elements of the output code should be chosen so as to be statistically independent.

2. It is more realistic to hypothesize\(^5\) that the classes are exclusive. Each example can only belong to one class, and the recognition of this class corresponds to a single discrete distribution that gives the conditional probability of the class. The corresponding conditional model, a straightforward generalization of the model described in Section 2.1.2, defines a single machine and is presented elsewhere (Haffner et al., 2005a). It is also possible to use of a joint model instead of a conditional model.

Table 1 compares the classification performance for the smallest of the datasets described in Section 5 (Vtone2), and finds that, despite the probabilistic intuition, the 1-vs-other scheme performs better than the single machine scheme. Furthermore, the 1-vs-other scheme provides considerable savings in memory and computational requirements. This result is confirmed elsewhere (Haffner et al., 2005a) on a variety of datasets (including standard benchmark datasets

\(^5\) This is usually true for our language classification problems, with some exceptions.
such as WebKB and Reuters-21758). It would be interesting to further these comparisons using other Maxent optimization algorithms, for instance based on Iterative Scaling. Because of memory requirements, single machine implementations of SL1-Max do not scale to the largest dataset presented in Section 5. In the rest of the paper, we will assume a 1-vs-other scheme and a conditional distribution.

4 Description of the software implementation and parallelization

Our multiclass software library that implements a variety of large margin classifiers is named Llama (Learning Large Margins). It relies on the assumption that we have a combination of independent binary classifiers, which can be heterogeneous. For the same problem, one can use SVMs to discriminate one class and SL1-Max to discriminate another class.

Llama implements different learning algorithms in learners modules. Our experiments use the following Llama learners:

**SVMs** At each iteration, the two examples with the largest gradient are selected (Joachims, 1998a), which is equivalent to the SMO algorithm (Platt, 1998). The number of examples considered is reduced with a multilevel shrinking algorithm. The kernel cache can be shared between binary classifiers, provided that they use the same kernel. A great emphasis was put into improvements that make these algorithms more robust to very noisy data, in particular with respect to the computation of the threshold and the unshrinking algorithm. Llama implementation of SVMs was found to converge to the same solution as SVMlight (Joachims, 1998a), while being faster because of the cache that is shared between classifiers.

**SL1-Max** This implementation starts from the sequential algorithm described by Dudik et al. and implements a multiclass Maxent extension with both joint and conditional distributions (Haffner et al., 2005a). Speed is further improved with a partial pricing algorithm that looks for the next feature in a restricted working set (Phillips and Haffner, 2005).

**AdaBoost** Adaboost is an exact implementations of the algorithm described by Collins et al.. The same partial pricing algorithm as SL1-Max was added to speedup the search for the next feature.

Llama is based on the assumption that the maximum amount of code must be reusable between learners, which typically only includes a few hundred lines of code. All the code to recombine classifiers, either 1-vs-other or 1-vs-1, is learner-independent. The code that loads the examples and computes sparse inner products is also shared between all learners.
<table>
<thead>
<tr>
<th>Call type</th>
<th>Full training set</th>
<th>Reduced set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pos</td>
<td>neg</td>
</tr>
<tr>
<td>Not(Information)</td>
<td>75547</td>
<td>503155</td>
</tr>
<tr>
<td>Yes</td>
<td>58377</td>
<td>520325</td>
</tr>
<tr>
<td>Request(Call_Transfer)</td>
<td>54431</td>
<td>524271</td>
</tr>
<tr>
<td>Ask(Info)</td>
<td>16294</td>
<td>562408</td>
</tr>
<tr>
<td>Other</td>
<td>11543</td>
<td>567159</td>
</tr>
</tbody>
</table>

Table 2
Impact of splitting on the final training data size. The Full training set corresponds to \( M = 578702 \) examples of the VtoneN dataset described in Section 5. For each class, they are split onto positive (pos) and negative (neg) training examples. After merging the S sets of support vectors, each class has a reduced set with \( M'_c \) support vectors, which we split into positive and negative examples.

The availability of clusters of computers makes large-grain parallelization implementations very attractive. The following discussion presents two effective parallelization strategies that do not require any communication between computing units. It also describes the critical stopping heuristics.

4.1 Parallelization: AdaBoost and SL1-Max

Multiclass implementations with independent 1-vs-other classifiers allow for an easy parallelization:

1. Sort the classes by their frequency distribution.
2. Perform a Round Robin partition of this list into S groups of classes. The frequency distribution of each group should be similar, leading to similar training times.
3. Learning can then be distributed over S processors.
4. Gain by parallelizing into S subsets: up to S times faster.

This parallelization technique is used for AdaBoost and SL1-Max, and we could observe that with the \( \delta \)-loss stopping criterion, the training times for each of the class subsets were very similar.

4.2 Parallelization: SVM

The previous solution is possible, but as shown in Section 3.1, learning all M examples together is \( O(M^2) \).
Recently, a cascading approach where the training data is split into a hierarchical fashion was proposed (Graf et al., 2005) to reduce the number of examples each sub-learning task had to handle. To be able to re-use our code in the simplest way, we only implemented a simplified version of this cascading approach, which we simply refer to as splitting.

1. Split the training set into $S$ subsets of equal size, with $\frac{N}{S}$ examples.
2. Each $S$ processors learns all the classes on one of the subsets.
3. Gain by parallelizing into S subsets: up to $S^2$ faster.
4. For each class $c$, merge the $S$ sets of support vectors to form a reduced training set with $M_c'$ examples.
5. Retrain the complete SVM with the reduced training set using the parallelization scheme described in Section 4.1. This step is the most time consuming, especially when the data is noisy and when a majority of training vectors become support vectors.

Compared to the simpler parallelization proposed in Section 4.1, this algorithm would improve learning speed for classes for which $M_c'$ is much smaller than $M$. Table 2 shows how the number of positive and negative examples are reduced for five selected call types when learning with the VtongN dataset and performing a splitting with $S = 10$. The reduction in the number of positive examples has the most critical impact on the training: it is striking to see how the splitting algorithm works for well defined call types such as “Yes” and fails for call types that mostly consist of noise such as “Other”.

Graf et al. establish that cascade SVMs converge toward the optimum after verifying that all the training vectors are properly classified and re-injecting improperly classifier vectors back into the set of support vectors to train. This is especially important when the data is noisy. We have not implemented this very time-consuming process yet, which causes our splitting algorithm to be sub-optimal. Experiments in Section 6 show that this approximate splitting still offers an excellent speed/performance compromise.

4.3 A multiclass stopping criterion

Deciding when to stop is often one of the most delicate part of the implementation of a learning algorithm.

As discussed above, a general stopping criterion for binary classification learning is to look for the minimum error with a development dataset. This technique generalizes well to single-machine multiclass schemes. Take for instance the Boostexter software (Schapire and Singer, 2000), which supports single-machine multiclass procedures such as AdaBoost.MH and AdaBoost.MR and involves a single optimization process to stop.
When training C independent classifiers, one has to deal with C independent stopping decisions. A minimum in the classification error of a binary classifier is not very helpful: further learning could increase the range of variation of the output so that the multiclass error, which is based on comparing outputs of different classifiers, is actually increased.

An examination of an optimal multiclass implementation of AdaBoost such as AdaBoost.MH justifies a stopping criterion based on the variation of the loss function, or $\delta$-loss (this was initially suggested as a stopping criterion for SL1-Max (Phillips et al., 2004)). In AdaBoost.MH, each iteration selects the combination of classifier and feature that would provide the largest reduction, or $\delta$-loss, in the loss. This approach, which is optimal, implies that, for all classifiers, the $\delta$-loss caused by the last feature update are comparable. A classifier with a much larger potential $\delta$-loss would be sequentially updated until it is brought down to the level of the other classifiers.

When the classifiers are trained independently, their $\delta$-loss cannot be compared on the fly. However, if we wish to stop the optimization of each one of them at a “comparable” level of advancement, the $\delta$-loss stopping criterion comes as the logical solution: stopping occurs when the averaged $\delta$-loss over the last few iterations gets below a given value.

This $\delta$-loss criterion works for both AdaBoost and SL1-Max, but was never applied to AdaBoost before. It is clearly superior to the use a fixed number of iterations, as one needs to specify enough iterations to learn the most difficult class, which results in too many learning iterations for the other classes. Experiments suggest that the $\delta$-loss criterion is considerably more efficient than the use of a development set, which has the following drawbacks:

• Training data must be set apart in this development set.
• Because of the presence of plateaus, one can only be sure that a minimum has been reached in the binary error specific to this classifier after the error has stopped going down for a large number of iterations (how to specify this number is problematic by itself).
• The independent minimization of binary errors does not mean that the multiclass error is minimized. We found examples where further training would have been desirable.

5 Descriptions of the tasks and datasets

The first task we considered is that of a deployed customer-care application (“How may I help you?” or HMIHY). The set of categories is finite and is limited to 64 classes. The calls are classified based on the user’s response to
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of classes (C)</th>
<th>Training size (M)</th>
<th>Testing size</th>
<th>Number of n-grams (N)</th>
<th>Active Features</th>
<th>ASR word accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMIHY</td>
<td>64</td>
<td>35551</td>
<td>5000</td>
<td>24177</td>
<td>20.4</td>
<td>72.5%</td>
</tr>
<tr>
<td>Vtone1</td>
<td>97</td>
<td>29561</td>
<td>5537</td>
<td>22007</td>
<td>22.9</td>
<td>70.5%</td>
</tr>
<tr>
<td>Vtone2</td>
<td>82</td>
<td>9093</td>
<td>5172</td>
<td>8689</td>
<td>22.6</td>
<td>68.8%</td>
</tr>
<tr>
<td>VtoneN</td>
<td>519</td>
<td>578702</td>
<td>64300</td>
<td>95474</td>
<td>18.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 3

Key characteristics of the four datasets used in the experiments. The fifth column displays the total number of unigrams, bigrams, and trigrams found in the one-best output of the ASR for the utterances of the training set, that is the number of features used by Boostexter or SVMs used with the one-best outputs. The sixth column gives the average number of active (or non-zero) features per training vector. The ASR word accuracy is indicative of the quality of the sentence input to the semantic classification system.

the first greeting prompt: “Hello, this is AT&T. How may I help you?”

Table 3 indicates the size of the HMIHY dataset we used for training and testing. The word accuracy of the system based on the best hypothesis of the speech recognizer is 72.5%

Table 3 reports similar information for two other datasets, Vtone1 and Vtone2. These are more recently deployed spoken-dialog systems in different areas, e.g., Vtone1 is a task where users interact with a system related to health-care with a larger set of categories (97). The size of the Vtone1 datasets we used and the word accuracy of the recognizer (70.5%) make this task otherwise similar to HMIHY. The datasets provided for Vtone2 are significantly smaller with a higher word error rate. The word error rate is indicative of the difficulty of classification task since a higher error rate implies a more noisy input.

By collecting together many datasets and merging call-types when possible (for instance, all the “Request(customer care)” call types were merged), we built a much larger dataset named VtoneN. The labeling in VtoneN may be noisy, as call types with the same name are not exactly similar from one dataset to the other. For instance, the percentage of VtoneN utterances that are labeled as “Request(Customer_Care)” is around 9.5%. This is to be compared with the best binary classification accuracy we could obtain on this call type (6%) and indicates a high level of noise.

Each utterance may be labeled with several classes. The evaluation is based on the top-class error rate (TCER) criterion: an example is considered an error if the highest scoring class given by the classifier is not part of the target labels.
Fig. 3. Class Frequency distribution for the top 50 classes of VtoneN.

In our experiments, we used the $n$-gram detectors with $n = 3$ for Vtone2, HMIHY and Vtonel and $n = 2$ for VtoneN\textsuperscript{6}.

Thus, the feature set we used was that of all $n$-grams with $n \leq 2$ or 3. Table 5 indicates the total number of distinct features of this type found in the datasets. For all datasets, the $n$-gram detector resulted in one binary feature per $n$-gram in the dictionary. The resulting input vector has $N$ dimensions (the total number of $n$-grams) and its feature values which are either 1 ($n$-gram is present in the sentence) or 0 ($n$-gram is absent from the sentence). In the case of SVMs, a further normalization to the unit sphere was performed. No further processing was necessary for AdaBoost and SL1-Max. Figure 3 shows the class frequency distribution for the VtoneN task. This figure shows that the top 3 classes account for 33% of the data and the top 50 classes account for 89% of the data.

\textsuperscript{6} We found on Vtone2, HMIHY and Vtonel that there was no difference in performance between $n = 2$ and $n = 3$. We decided to use $n = 2$ for experiments on VtoneN as the total number of features has a critical impact on memory requirements. We also found that using $n$-gram counts instead of simple detectors requires removing common stop words such as the, and even when doing so, do not impact performance in a noticeable way. As stop word removal requires human monitoring for each new task, we decided not to use this procedure. Moreover, AdaBoost and SL1-Max training procedures become significantly more complex with non-binary features.
6 Experiments with a large number of examples

This section addresses the most critical scaling issue, namely the number of training examples. We start by varying the number of training examples and observe the evolution in performance, both in term of error rate and training time, of each algorithm. The way the results are reported in this first series of experiments does not give a clear view of the trade-off between the error rate and the training time. A second set of experiments shows the error rate as a function of the training time. The choice of the regularization parameter is also covered in this section, as it strongly interacts with the training set size.

6.1 Evolution of error rate and training time with training data size

In these experiments, we scale up the size of the training subsets (while keeping the same number of classes) from 1/32 of the VtoneN training data to all the data. This enables us to determine scaling "laws".
In Figures 4 and 5, five algorithms (AdaBoost, AdaBoost-C, SL1-Max, SVM with linear kernels, and SVM with second degree polynomials) are compared. Moreover, the impact of the stopping criterion (for AdaBoost, AdaBoost-C and SL1-Max) and of the parallelization strategy (“SVM split” refers to the splitting algorithm) is studied.

Each algorithm corresponds to one line style, and variations on the algorithm (such as the $\delta$-loss stopping criterion or the use of splitting) correspond to different point styles. The six points of performance measure correspond to six reductions of the full 578702 training samples by increasing powers of 2. All the training times reported here are obtained by adding the times used by each computing unit, which are Pentium 4 Xeon processors clocked at 2.4MHz.

Figure 4 shows how each algorithm training time $T$ scales with the set size $M$. The reference curve “Slope 1” indicates perfect $T = O(M)$ scaling behavior, while “Slope 2” represents $T = O(M^2)$ scaling behavior.

How close SVMs are to the $T = O(M^2)$ is remarkable, and was predicted in Section 3.1. The distance between curves of same line style indicate the impact of splitting on the training speed. Speedup strategies such as splitting the training set considerably improve training speed (our time figures do not report the additional gain one could expect from parallelization). They shift the curve but the same scaling behavior is observed.

While AdaBoost and SL1-Max show a $O(M)$ behavior, the phenomenon is harder to interpret. The stopping criterion is much more subjective and has a considerable influence on the learning time. Our observations are that both the time per iteration and the number of iterations are in $O(\sqrt{M})$, but this does not verify very well. The distance between curves of same line style indicate the impact of the stopping criterion on the training speed.

A comparison between algorithms also requires knowing the test error, which is given in Figure 5. It suggests that SVMs and SL1-Max are the best choices, SVM for smaller sets, and SL1-Max as we scale up. The SVM splitting speedup trick, given that it is an approximation and that the data is noisy, results into a loss of performance, but is still very competitive in terms of learning time/TCER trade-off. The next section will give detailed plots of learning time/TCER curves.

6.2 Error rate as a function of training time

In this section, we compare the learning time/TCER curves for all the algorithms considered on our largest (Vtone\textit{N} in Figure 6) corpora. These learning time/TCER trade-off curves are the most critical to help us in our choice of
algorithm. We explore various points in this trade-off by varying the stopping criterion.

In Figure 6, the SVM results were obtained using the splitting parallelization algorithm, which is approximate. Figure 5 suggests that in the case of linear SVMs, the use of this splitting algorithms adds between 1% and 2% to the TCER. This explains why linear SVMs, which normally are very close to SL1-Max, perform significantly worse here.

Having made this caveat, these curves confirm the observations of the previous section, and further the comparative analysis of different implementations of SL1-Max and AdaBoost. The results in Figure 6 also verify that, in the case of SL1-Max, to use a joint distribution instead of a conditional distribution causes a small loss in performance.
Fig. 6. Learning time/TCER trade-off on the VtoneN dataset. Stopping points for SL1-Max range from $\delta - loss = 10^{-5}$ to $10^{-9}$. Stopping points for SL1-Max range from $\delta - loss = 10^{-5}$ to $10^{-7}$. Only a single stopping point was considered for SVMs, corresponding to a duality gap of 0.01.

Fig. 7. Evolution of the TCER as a function of the stopping $\delta$-loss and the regularization parameter. The left plot corresponds to Vtone2 and the right plot to VtoneN.

6.3 **Choosing the regularization parameter**

One of the advantages of SVMs or SL1-Max over AdaBoost is the choice of the regularization constants $C$ or $\beta$. In the case of SVMs, Joachims (1998b) shows that, for text classification applications, $C = 1$ is usually close to the optimal when the examples are normalized to 1. We verified that $C = 1$ is usually a good choice for our problems.

In the case of SL1-Max, systematic experiments reported in Figure 7 suggest that $\beta = 0.5$ is a good compromise:

- It is clearly the best performer on a small dataset like Vtone2.
• It is close to the best on the larger VtoneN. Better performance is only obtained with smaller $\beta$ values, which results in a significantly slower learning. If one chooses to trade learning speed for performance, it is better to decrease $\delta$-loss than $\beta$.

$\beta = 0$ corresponds to unregularized maximum entropy. While it shows very poor performance on Vtone2, results on VtoneN are nearly as good as $\beta = 0.5$. This suggests that regularization is less and less critical as we deal with larger datasets.

6.4 Experimental Summary

Adaboost This noisy dataset presented a major challenge for Adaboost, which is not regularized. Figure 4 shows that AdaBoost-C, an implementation of AdaBoost with confidence rated predictions, is considerably faster than Adaboost with basic feature detectors as weak classifiers. However, AdaBoost-C also offers more parameters to learn (namely the bias) and, in the absence of regularization, tends to overfit on the very noisy VtoneN dataset. While Figure 5 shows much higher error rates for AdaBoost-C, the problem becomes less pronounced when the amount of data grows larger and the need for regularization becomes less critical. Figure 6 shows that AdaBoost-C still overfits with 578,702 examples. Note that on Internet traffic classification problems, where there is little noise in the data, we found that Adaboost did outperform SL1-Max (Haffner et al., 2005b).

SL1-Max SL1-Max may turn out to be the computationally efficient regularized adaptation of AdaBoost that has eluded researchers for a long time. As a matter of fact, a unification with Maxent is one of the techniques that has been proposed to regularize AdaBoost (Lebanon and Laflerty, 2002). SL1-Max also appears as a serious challenger to the linear SVMs which have been highly successfully for text classification applications. On a cluster of 32 2.4MHz Pentium 4 Xeon processors, SL1-Max with stopping $\delta$-loss=$10^{-7}$ takes 10 minutes to learn 578,702 examples and 519 classes. Because of the linear scaling factor, one can easily contemplate millions of examples. The largest set of data on spoken language understanding we could find contained only a little bit more than 600,000 examples. However, we are also starting applications of the same ideas to tagging or Internet packet classification, where millions of examples and thousands of features are available. The main limiting factor is not the time, but the memory to store the examples. Another advantage of SL1-Max over AdaBoost is more flexibility, in the choice of the regularization parameter, but also in the choice of the model.

SVMs SVMs with second degree polynomial kernels yield the best performance. However, their testing time is unacceptably slow, as they require to compute the kernel with all the 360,000 support vectors (this is what we
obtained with 578,702 training examples and 519 classes).

Linear SVMs yield the same performance as SL1-Max. A speed comparison may not be entirely fair, as the results reported for SVMs in this paper leave significant room for improvement:

- While our Llama-based SVM implementation was found to be significantly faster than SVMlight, it is not as well optimized as more recent SVM packages such as LIBSVM (Chang and Lin, 2001).
- Considerable research effort was invested in making SL1-Max and AdaBoost very efficient on sparse data (Phillips and Haffner, 2005). We are just starting research SVM speedups that would be specific to sparse data (beyond simple kernel speedups). Preliminary experiments suggest we could speed up training by up to 10 times on the datasets considered here.
- New approaches such as Core SVM (Tsang et al., 2005) and LASVM (Bordes et al., 2005) could provide significant speedups on large datasets.
- **Primal** (or feature based optimization) would be possible for linear SVMs. We have not tried techniques such as Lagrangian SVMs (Mangasarian and Musicant, 2001), as the inversion of a $N \times N$ matrix where $N \approx 100,000$ would be problematic.

7 Impact of the number of features

By gathering observations from the previous sections, it is easy to show that the total number of features (typically the number of word $n$-grams) is not a major issue in semantic classification problems.

Table 3 shows that the input vectors are very sparse: the number of active (non zero) feature per example is low. If we call $|\mathbf{x}|$ this number of active features in vector $\mathbf{x}$:

- Storing $\mathbf{x}$ only requires $|\mathbf{x}|$ integer numbers to store the indexes of the non-zero features. In a binary vector, non-zero features implicitly have a 1 value. If we are dealing with a real-valued vector, an additional $|\mathbf{x}|$ floating point numbers are required.
- A properly implemented inner product between $\mathbf{x}_0$ and $\mathbf{x}_1$ only requires $\min(|\mathbf{x}_0|, |\mathbf{x}_1|)$ multiply-adds.

Thus kernel-based algorithms such as SVMs do not have any computational or memory requirement which scales in $O(N)$ (where $N$ is the total number of features). For feature based algorithms, one can avoid this $O(N)$ scaling behavior by using working-set methods, where only a subset of features is selected.

30
<table>
<thead>
<tr>
<th>Length</th>
<th>Vtone2</th>
<th>Vtone1</th>
<th>HMIHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sh</td>
<td>1678</td>
<td>1546</td>
<td>2199</td>
</tr>
<tr>
<td>Md</td>
<td>2545</td>
<td>3315</td>
<td>1911</td>
</tr>
<tr>
<td>Lg</td>
<td>949</td>
<td>676</td>
<td>890</td>
</tr>
</tbody>
</table>

Table 4
Number of test examples as function of the number of words in the recognized utterance for the Vtone2, Vtone1, and HMIHY datasets.

A much more complex issue is the variability in the number of features, which comes from the fact that we classify sentences of different lengths. Table 4 shows the distribution of test sentences among three categories.

**Sh** Short utterances ($|x| \leq 3$) have less than three words and typically correspond to discourse call types such as “yes” and “hello”.

**Md** Medium utterances ($|x| > 3$ and $|x| \leq 15$) have less than 15 words. They contain a single clause or sentence that represents the user request.

**Lg** Long utterances ($|x| > 15$) have more than 15 words. They usually contain several grammatical sentences or clauses. In general, only one of them carries the call-type and the others can be considered as noise.

Table 5 analyzes the influence of utterance length over the test error rate. All classification methods considered here fail to recognize long utterances. The HMIHY dataset was labeled with extreme care, which leads to a very low error rate for short utterances; nonetheless, the error rate on long utterances is close to 50%.

One solution under investigation is to rely on preprocessing and segment the long utterances into clause units (Gupta and Bangalore, 2002). Global learning approaches combining both segmentation of the relevant clause and classification are also being investigated. A particular effort has been invested in *alignment kernels*, where time alignment is part of the kernel computation process. However, it was recently established (Cortes et al., 2004) that such kernels, which belong to the same family as edit-distance kernels, are not positive definite, and therefore poor candidates for SVM optimization. Large margin structured classification approaches (Bartlett et al., 2005), where the output $y$ is a graphical model with a meaningful structure, could also represent a large class of potential solutions for this problem.

8 Conclusion

This paper provides systematic comparisons between SVMs, Adaboost and regularized Maximum Entropy for the classification of a large and noisy corpus
<table>
<thead>
<tr>
<th>Classifier</th>
<th>Vtone2 TCER</th>
<th>Vtone1 TCER</th>
<th>HMY TCER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sh</td>
<td>Md</td>
<td>Lg</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>16.6</td>
<td>29.0</td>
<td>59.1</td>
</tr>
<tr>
<td>SVM/Lin</td>
<td>16.9</td>
<td>27.5</td>
<td>53.6</td>
</tr>
<tr>
<td>SVM/Poly2</td>
<td>16.4</td>
<td>27.1</td>
<td>51.8</td>
</tr>
</tbody>
</table>

Table 5
Top Class Error Rate (TCER) reported for various utterance lengths

of conversational data. For all these classifiers, this study argues that breaking a multiclass problem into binary classification subtasks that can be optimized independently is a simple and efficient approach. For instance, in the case of SVMs, the number of classes can be expanded with hardly any penalty in terms of learning time. It also identifies Maximum Entropy modeling as an alternative source of inspiration for novel multiclass schemes.

The main factor that determines the learning time is the number of examples. These experiments suggest that current computer clusters can handle millions of examples. For a task of this size, the paper recommends two algorithms: SVMs, a kernel-based method, and Regularized Maximum Entropy, a feature-based method. Kernel-based methods have a greater representation power, while feature-based methods offer sparser models that are easier to interpret and provide faster runtime performance. An efficient regularized version of AdaBoost would be required for the noisy data employed in this paper and may be obtained in the future from a unification with Maxent algorithms.

The main limitation of these methods reside in their poor ability to learn variable length sequences, where the relevant classification information is contained in a sub-sequence and where some form of alignment would be necessary. This could be the focus of future research.

9 Acknowledgments

The author wishes to thank Gokhan Tur, Steven Phillips, Rob Schapire, Srini-vas Bangalore, Mazin Rahim and James McKinney for their critical help at various stages of the work presented here.

References

pirical Methods in Natural Language Processing and Very Large Corpora.
In: Advances in Neural Information Processing Systems 17.
Gradient Algorithms for Large-margin Structured Classification. In: Ad-
Begega, L., Drucker, H., Gibbon, D., Haffner, P., Liu, Z., Renger, B.,
document: Fast kernel classifiers with online and active learning.
Breiman, L., 1999. Prediction Games and Arcing Algorithms. Neural Compu-
tation 11 (7), 1493–1517.
Chelba, C., Acero, A., September 2003. Discriminative Training of N-gram
Classifiers for Speech and Text Routing. In: Proceedings of INTER-
SPEECH’03 - Eurospeech. Geneva, Switzerland.
158–169.
Collobert, R., Bengio, S., 2001. SVMTorch: Support vector machines for large-
large scale problems. Neural Computation 14 (5).
Crammer, K., Singer, Y., 2002. On the algorithmic implementation of multi-
class kernel-based vector machines. Journal of Machine Learning Research
2, 265–292.
Machines. Cambridge University Press, Cambridge UK.
Dialog Systems with Data Reuse. In: Proceedings of the 5th SIGdial Work-
shop on Discourse and Dialogue.
Regularized Maximum Entropy Density Estimation. In: Proceedings of
COLT’04. Springer Verlag, Banff, Canada.
Freund, Y., Schapire, R. E., 1996. Experiments with a New Boosting Algo-
statistical view of boosting.
Communication 23, 113–127.


