Solving MultiClass Support Vector Machines with LaRank

Antoine Bordes[†*] BORDES@POLEIA.LIP6.FR
Léon Bottou[†] LEON@BOTTOU.ORG
Patrick Gallinari[*] GALLINARI@POLEIA.LIP6.FR
Jason Weston[†] JASEWESTON@GMAIL.COM
(†) NEC Laboratories America, Inc., 4 Independence Way, Princeton, NJ08540, USA.
(∗) LIP6, Université de Paris 6, 104 Avenue du Pdt Kennedy, 75016 Paris, France

Abstract

Optimization algorithms for large margin multiclass recognizers are often too costly to handle ambitious problems with structured outputs and exponential numbers of classes. Optimization algorithms that rely on the full gradient are not effective because, unlike the solution, the gradient is not sparse and is very large. The LaRank algorithm sidesteps this difficulty by relying on a randomized exploration inspired by the perceptron algorithm. We show that this approach is competitive with gradient based optimizers on simple multiclass problems. Furthermore, a single LaRank pass over the training examples delivers test error rates that are nearly as good as those of the final solution.

1. Introduction

Much has been written about the recognition of multiple classes using large margin kernel machines such as support vector machines (SVMs). The most widely used approaches combine multiple binary classifiers separately trained using either the *one-versus-all* or *one-versus-one* scheme (e.g. Hsu & Lin, 2002). Alternative proposals (Weston & Watkins, 1998; Crammer & Singer, 2001) reformulate the large margin problem to directly address the multiclass problem. These algorithms are more expensive because they must simultaneously handle all the support vectors associated with different inter-class boundaries. Rigorous experiments (Hsu & Lin, 2002; Rifkin & Klautau, 2004) suggest that this higher cost does not translate into higher generalization performance.

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

The picture changes when one considers learning systems that predict *structured outputs* (e.g. Bakır et al., 2007). Instead of predicting a class label y for each pattern x, structured output systems produce complex discrete outputs such as a sequences, trees, or graphs. Since these potential outputs can be enumerated (in theory), these systems can be viewed as multiclass problems with a number of classes growing exponentially with the characteristic size of the output. Dealing with so many classes in a large margin classifier would be infeasible without smart factorizations that leverage the specific structure of the outputs (Taskar et al., 2005; Tsochantaridis et al., 2005). This is best achieved using a direct multiclass formulation because the factorization of the output space implies that all the classes are handled simultaneously. It is therefore important to reduce the computational cost of multiclass SVMs with a potentially large number of classes.

MCSVM Crammer and Singer (2001) propose a multiclass formulation that we call *partial ranking*. The dual cost is a function of a $n \times k$ matrix of Lagrange coefficients where n is the number of examples and k the number of classes. Each iteration of the MCSVM algorithm maximizes the restriction of the dual cost to a single row of the coefficient matrix. Successive rows are selected using the gradient of the cost function. Unlike the coefficients matrix, the gradient is not sparse. This approach is not feasible when the number of classes k grows exponentially, because the gradient becomes too large.

SVMstruct Tsochantaridis et al. (2005) essentially use the same partial ranking formulation for the SVMstruct system. The clever *cutting plane* algorithm ensures convergence but only requires to store and compute a small part of the gradient. This crucial difference makes SVMstruct suitable for structured output problems with a large number of classes.

Kernel Perceptrons Online algorithms inspired by the perceptron (Collins, 2002; Crammer & Singer, 2003) can be interpreted as the successive solution of optimization subproblems restricted to coefficients associated with the current training example. There is no need to represent the gradient. The random ordering of the training examples drives the successive optimizations. Perceptrons provide surprisingly strong theoretical guarantees (Graepel et al., 2000). They run very quickly but provide inferior generalization performances in practice.

LaRank This paper proposes LaRank, a stochastic learning algorithm that combines partial gradient information with the randomization arising from the sequence of training examples.

- LaRank uses gradients as sparingly as SVMstruct and yet runs considerably faster. In fact, LaRank reaches an equivalent accuracy faster than algorithms that use the full gradient information.
- LaRank generalizes better than perceptron-based algorithms. In fact, LaRank provides the performance of SVMstruct or MCSVM because it solves the same optimization problem.
- LaRank achieves nearly optimal test error rates after a single pass over the randomly reordered training set. Therefore, LaRank offers the practicality of an online algorithm.

This paper first reviews and discusses the multiclass formulation of Crammer and Singer. Then it presents the LaRank algorithm, discusses its convergence, and reports experimental results on well known multiclass problems.

2. Multiclass Support Vector Machines

This section describes the partial ranking formulation of multiclass SVMs (Crammer & Singer, 2001). The presentation first follows (Tsochantaridis et al., 2005) then introduces a new parametrization of the dual program.

2.1. Partial Ranking

We want to learn a function f that maps patterns $x \in \mathcal{X}$ to discrete class labels $y \in \mathcal{Y}$. We introduce a discriminant function $S(x, y) \in \mathbb{R}$ that measures the correctness of the association between pattern x and class label y. The optimal class label is then

$$f(x) = \arg\max_{y \in \mathcal{Y}} S(x, y) \ . \tag{1}$$

We assume that the discriminant function has the form

$$S(x, y) = \langle w, \Phi(x, y) \rangle$$

where $\Phi(x, y)$ maps the pair (x, y) into a suitable feature space endowed with the dot product $\langle \cdot, \cdot \rangle$. As usual with kernel machines, the feature mapping function Φ is implicitly defined by the specification of a joint kernel function

$$K(x, y, \bar{x}, \bar{y}) = \langle \Phi(x, y), \Phi(\bar{x}, \bar{y}) \rangle \ . \tag{2}$$

Consider training patterns $x_1 \dots x_n \in \mathcal{X}$ and their class labels $y_1 \dots y_n \in \mathcal{Y}$. For each pattern x_i, we want to make sure that the score $S(x_i, y_i)$ of the correct association is greater than the scores $S(x_i, y)$, $y \neq y_i$, of the incorrect associations. This amounts to enforcing a partial order relationship on the elements of $\mathcal{X} \times \mathcal{Y}$. This *partial ranking* can be expressed by constraints

$$\forall i = 1 \dots n \quad \forall y \neq y_i \quad \langle w, \delta\Phi_i(y_i, y) \rangle \geq 1$$

where $\delta\Phi_i(y, \bar{y})$ stands for $\Phi(x_i, y) - \Phi(x_i, \bar{y})$.

Following the standard SVM derivation, we introduce slack variables ξ_i to account for the potential violation of the constraints and optimize a combination of the norm of w and of the size of the slack variables.

$$\min_{w} \ \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^{n} \xi_i \tag{3}$$

$$\text{subject to} \begin{cases} \forall i \quad \xi_i \geq 0 \\ \forall i \quad \forall y \neq y_i \quad \langle w, \delta\Phi_i(y_i, y) \rangle \geq 1 - \xi_i \end{cases}$$

2.2. Dual Programs

The usual derivation leads to solving the following equivalent dual problem (Crammer & Singer, 2001; Tsochantaridis et al., 2005):

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \sum_{i, y \neq y_i} \alpha_i^y - \frac{1}{2} \sum_{\substack{i, y \neq y_i \\ j, \bar{y} \neq y_j}} \alpha_i^y \alpha_j^{\bar{y}} \langle \delta\Phi_i(y_i, y), \delta\Phi_j(y_j, \bar{y}) \rangle \\ &\text{subject to} \begin{cases} \forall i \quad \forall y \neq y_i \quad \alpha_i^y \geq 0 \\ \forall i \quad \displaystyle\sum_{y \neq y_i} \alpha_i^y \leq C \end{cases} \end{aligned} \tag{4}$$

This problem has $n(k-1)$ variables α_i^y, $y \neq y_i$ corresponding to the constraints of (3). Once we have the solution, the discriminant function is

$$S(x, y) = \sum_{i, \bar{y} \neq y_i} \alpha_i^{\bar{y}} \langle \delta\Phi_i(y_i, \bar{y}), \Phi(x, y) \rangle$$

This dual problem can be considerably simplified by reparametrizing it with nk variables β_i^y defined as

$$\beta_i^y = \begin{cases} -\alpha_i^y & \text{if } y \neq y_i \\ \displaystyle\sum_{\bar{y} \neq y_i} \alpha_i^{\bar{y}} & \text{otherwise} \end{cases} \tag{5}$$

Note that only the $\beta_i^{y_i}$ can be positive. Substituting in (4), and taking into account the relation $\sum_y \beta_i^y = 0$, leads to a much simpler expression for the dual problem (the $\delta\Phi_i(\dots)$ have disappeared.)

$$\max_{\boldsymbol\beta} \sum_i \beta_i^{y_i} - \frac{1}{2}\sum_{i,j,y,\bar y} \beta_i^y \beta_j^{\bar y} \langle \Phi(x_i,y), \Phi(x_j,\bar y)\rangle$$
$$\text{subject to} \begin{cases} \forall i \ \ \forall y \neq y_i \ \ \beta_i^y \leq 0 \\ \forall i \ \ \beta_i^{y_i} \leq C \\ \forall i \ \ \sum_y \beta_i^y = 0 \end{cases} \tag{6}$$

The discriminant function then becomes

$$S(x,y) = \sum_{i,\bar y} \beta_i^{\bar y} \langle \Phi(x_i,\bar y), \Phi(x,y)\rangle$$

2.3. Kernel Factorization

In practice, smart factorizations of the joint kernel (2) are crucial to reduce the memory required to store or cache the kernel values. This paper focuses on simple multiclass problems whose kernel function (2) takes the form

$$\langle \Phi(x,y), \Phi(\bar x,\bar y)\rangle = k(x,\bar x)\,\delta(y,\bar y)$$

where $k(x,\bar x)$ is a kernel defined on the patterns, and where $\delta(y,\bar y)$ is 1 if $y=\bar y$ and 0 otherwise.

The dual problem (6) then becomes

$$\max_{\boldsymbol\beta} \sum_i \beta_i^{y_i} - \frac{1}{2}\sum_{i,j}\sum_y \beta_i^y \beta_j^y \, k(x_i,x_j)$$
$$\text{subject to} \begin{cases} \forall i \ \ \forall y \ \ \beta_i^y \leq C\delta(y,y_i) \\ \forall i \ \ \sum_y \beta_i^y = 0 \end{cases} \tag{7}$$

and the discriminant function becomes

$$S(x,y) = \sum_i \beta_i^y \, k(x_i,x)$$

When there are only two classes, this reduces to the standard SVM solution (without equality constraint.)

Structured output learning systems (Tsochantaridis et al., 2005) call for much more sophisticated factorizations of the joint kernel. For the sake of simplicity, we describe the LaRank algorithm in the context of the multiclass problem (7) which is the focus of this paper. Dealing with the general problem (6), or handling the variable margins suggested by Tsochantaridis et al. (2005), only requires minor changes.

3. Optimization Algorithm

During the execution of the optimization algorithm, we call *support vectors* all pairs (x_i,y) whose associated coefficient β_i^y is non zero; we call *support patterns* all patterns x_i that appear in a support vector.

The LaRank algorithm stores the following data:

- The set $\mathcal{S}$ of the current support vectors.
- The coefficients β_i^y associated with the support vectors $(x_i,y)\in\mathcal{S}$. This describes the solution since all the other $\boldsymbol\beta$ coefficients are zero.
- The derivatives $g_i(y)$ of the dual objective function with respect to the coefficients β_i^y associated with the support vectors $(x_i,y)\in\mathcal{S}$.

$$\begin{aligned} g_i(y) &= \delta(y,y_i) - \textstyle\sum_j \beta_j^y \, k(x_i,x_j) \\ &= \delta(y,y_i) - S(x_i,y) \end{aligned} \tag{8}$$

  Note that we do not store or even compute the remaining coefficients of the gradient. In general, these missing derivatives are not zero because the gradient is not sparse.

A naive implementation could simply precompute all the kernel values $k(x_i,x_j)$. This would be a waste of processing time because the location of the optimum depends only on the fraction of the kernel matrix that involves support patterns. Our code computes kernel values on demand and caches them in sets of the form

$$E(y,j) = \{\ k(x_i,x_j) \ \text{ such that } (x_i,y)\in\mathcal{S}\ \}.$$

Although this cache stores several copies of the same kernel values, caching individual kernel values has a higher overhead.

3.1. Elementary Step

Problem (7) lends itself to a simple iterative algorithm whose elementary steps are inspired by the well known sequential minimal optimization (SMO) algorithm (Platt, 1999).

Algorithm 1 SmoStep(i, y_+, y_-):

1: Retrieve or compute $g_i(y_+)$.
2: Retrieve or compute $g_i(y_-)$.
3: Let $\lambda^u = \frac{g_i(y_+)-g_i(y_-)}{2\,k(x_i,x_i)}$
4: Let $\lambda = \max\left\{0,\ \min(\ \lambda^u,\ C\,\delta(y_+,y_i) - \beta_i^{y_+}\ )\right\}$
5: Update $\beta_i^{y_+} \leftarrow \beta_i^{y_+} + \lambda$ and $\beta_i^{y_-} \leftarrow \beta_i^{y_-} - \lambda$
6: Update $\mathcal{S}$ according to whether $\beta_i^{y_+}$ and $\beta_i^{y_-}$ are zero.
7: Update gradients:
   $\forall j$ s.t. $(x_j,y_+)\in\mathcal{S}$, $g_j(y_+) \leftarrow g_j(y_+) + \lambda\, k(x_i,x_j)$
   $\forall j$ s.t. $(x_j,y_-)\in\mathcal{S}$, $g_j(y_-) \leftarrow g_j(y_-) - \lambda\, k(x_i,x_j)$

Each iteration starts with the selection of one pattern x_i and two classes y_+ and y_-. The elementary step modifies the coefficients $\beta_i^{y_+}$ and $\beta_i^{y_-}$ by opposite amounts,

$$\begin{aligned} \beta_i^{y_+} &\longleftarrow \beta_i^{y_+} + \lambda \\ \beta_i^{y_-} &\longleftarrow \beta_i^{y_-} - \lambda \end{aligned} \tag{9}$$

where $\lambda \geq 0$ maximizes the dual objective function (7) subject to the constraints. This optimal value is easily computed by first calculating the unconstrained optimum

$$\lambda^u = \frac{g_i(y_+) - g_i(y_-)}{2\,k(x_i, x_i)} \tag{10}$$

and then enforcing the constraints

$$\lambda = \max\left\{\, 0,\ \min(\, \lambda^u,\ C\,\delta(y_+, y_i) - \beta_i^{y_+}\,)\right\} \tag{11}$$

Finally the stored derivatives $g_j(y)$ are updated to reflect the coefficient update. This is summarized in algorithm 1.

3.2. Step Selection Strategies

Popular SVM solvers based on SMO select successive steps by choosing the pair of coefficients that defines the feasible search direction with the highest gradient. We cannot use this strategy because we have chosen to store only a small fraction of the gradient.

Stochastic algorithms inspired by the perceptron perform quite well by successively updating coefficients determined by randomly picking training patterns. For instance, in a multiclass context, Taskar (2004, section 6.1) iterates over the randomly ordered patterns: for each pattern x_i, he computes the scores $S(x_i, y)$ for all classes and runs SmoStep on the two most violating classes, that is, the classes that define the feasible search direction with the highest gradient.

In the context of binary classification, Bordes and Bottou (2005) observe that such perceptron-inspired updates lead to a slow optimization of the dual because the coefficients corresponding to the few support vectors are not updated often enough. They suggest to alternatively update the coefficient corresponding to a fresh random example and the coefficient corresponding to an example randomly chosen among the current support vectors. The related LaSVM algorithm (Bordes et al., 2005) alternates steps exploiting a fresh random training example and steps exploiting current support vectors selected using the gradient.

We now extend this idea to the multiclass formulation. Since the multiclass problem has both *support vectors* and *support patterns*, we define three ways to select a triple (i, y_+, y_-) for the elementary SmoStep.

Algorithm 2 ProcessNew(x_i):

```
1: if x_i is a support pattern then exit.
2: y+ ← y_i.
3: y- ← arg min_{y∈Y} g_i(y)
4: Perform SmoStep(i, y+, y-)
```

- ProcessNew (algorithm 2) operates on a pattern x_i that is *not a support pattern*. It chooses the classes y_+ and y_- that define the feasible direction with the highest gradient. Since all the β_i^y are zero, y_+ is always y_i. Choosing of y_- consists of finding $\arg\max_y S(x_i, y)$ since equation (8) holds.

Algorithm 3 ProcessOld:

```
1: Randomly pick a support pattern x_i.
2: y+ ← arg max_{y∈Y} g_i(y) subject to β_i^y < C δ(y, y_i)
3: y- ← arg min_{y∈Y} g_i(y)
4: Perform SmoStep(i, y+, y-)
```

Algorithm 4 Optimize:

```
1: Randomly pick a support pattern x_i.
2: Let Y_i = { y ∈ Y such that (x_i, y) ∈ S }
3: y+ ← arg max_{y∈Y_i} g_i(y) subject to β_i^y < C δ(y, y_i)
4: y- ← arg min_{y∈Y_i} g_i(y)
5: Perform SmoStep(i, y+, y-)
```

- ProcessOld (algorithm 3) randomly picks a *support pattern* x_i. It chooses the classes y_+ and y_- that define the feasible direction with the highest gradient. The determination of y_+ mostly involves labels y such that $\beta_i^y < 0$, for which the corresponding derivatives $g_i(y)$ are known. The determination of y_- again consists of computing $\arg\max_y S(x_i, y)$.
- Optimize (algorithm 4) resembles ProcessOld but picks the classes y_+ and y_- among those that correspond to existing *support vectors* (x_i, y_+) and (x_i, y_-). Using the gradient is fast because the relevant derivatives are already known and their number is moderate.

The ProcessNew operation is closely related to the perceptron algorithm. It can be interpreted as a stochastic gradient update for the minimization of the generalized margin loss (LeCun et al., 2007, §2.2.3), with a step size adjusted according to the curvature of the dual (Hildreth, 1957). Crammer and Singer (2003) use a very similar approach for the MIRA algorithm.

3.3. Adaptive Schedule

Previous works (Bordes & Bottou, 2005; Bordes et al., 2005) simply alternate two step selection strategies according to a fixed schedule. They also report results suggesting that the optimal schedule is in fact data-dependent.

We would like to select at each iteration an operation that causes a large increase of the dual in a small amount of time. For each operation type, LaRank maintains a running estimate of the average ratio of the dual increase over the duration. Running times are measured; dual increases are derived from the value of λ computed during the elementary step.

Each iteration of the LaRank algorithm (algorithm 5)

Algorithm 5 LARANK:

$\mathcal{S} \leftarrow \emptyset$.
$r_{\text{OPTIMIZE}}, r_{\text{PROCESSOLD}}, r_{\text{PROCESSNEW}} \longleftarrow 1$.
loop
  Randomly reorder the training examples.
  $k \longleftarrow 1$.
  while $k \leq n$ **do**
    Pick operation s with odds proportional to r_s.
    if $s =$ OPTIMIZE **then**
      Perform OPTIMIZE.
    else if $s =$ PROCESSOLD **then**
      Perform PROCESSOLD.
    else
      Perform PROCESSNEW(x_k).
      $k \leftarrow k + 1$.
    end if
    $r_s \leftarrow \mu \, \frac{\text{dual increase}}{\text{duration}} + (1-\mu)\, r_s$.
  end while
end loop

randomly selects which operation to perform with a probability proportional to these estimates. Our implementation uses $\mu = 0.05$. In order to facilitate timing, we treat sequences of ten OPTIMIZE as a single atomic operation.

3.4. Correctness and Complexity

Let $\nu^2 = \max_i\{k(x_i, x_i)\}$ and let κ, τ, η be small positive tolerances. We assume that the algorithm implementation enforces the following properties:

- SMOSTEP exits when $g_i(y_+) - g_i(y_-) \leq \tau$.
- OPTIMIZE and PROCESSOLD chooses y_+ among the y that satisfy $\beta_i^y \leq C\,\delta(y, y_i) - \kappa$.
- LARANK makes sure that every operation has probability greater than η to be selected at each iteration (see algorithm 5).

We refer to this as the (κ, τ, η)-algorithm.

Theorem *With probability 1, the (κ, τ, η)-algorithm reaches a $\kappa\tau$-approximate solution of problem (7), with no more than $\max\{\frac{2\nu^2 nC}{\tau^2}, \frac{2nC}{\kappa\tau}\}$ successful SMOSTEPs.*

Proof Sketch The convergence is a consequence from theorem 18 from (Bordes et al., 2005). To apply this theorem, we must prove that the directions defined by (9) form a *witness family* for the polytope defined by the constraints of problem (7). This is the case because this polytope is a product of n polytopes for which we can apply proposition 7 from (Bordes et al., 2005). The number of iterations is then bounded using a technique similar to that of (Tsochantaridis et al., 2005). The complete proof will be given in an extended version of this paper. □

The bound on the number of iterations is also a bound on the number of support vectors. It is linear in the number of examples and does not depend on the possibly large number of classes.

3.5. Stopping

Algorithm 5 does not specify a criterion for stopping its outer loop. Excellent results are obtained by performing just one or two outer loop iterations (epochs). We use the name LaRank×1 to indicate that we perform a single epoch, that is to say, a single pass over the randomly ordered training examples.

Other stopping criteria include exploiting the duality gap (Schölkopf & Smola, 2002, §10.1.1) and monitoring the performance measured on a validation set. We use the name LaRankGap to indicate that we iterate algorithm 5 until the difference between the primal cost (3) and the dual cost (7) becomes smaller than C. However, computing the duality gap can become quite expensive.

4. Experiments

This section report experiments carried out on various multiclass pattern recognition problems. Although our approach is partly motivated by structured output problems, this work focuses on well understood multiclass tasks in order best characterize the algorithm behavior.

4.1. Experimental Setup

Experiments were carried out on four datasets briefly described in table 1. The LETTER and USPS datasets are available from the UCI repository.[1] The MNIST dataset[2] is a well known handwritten digit recognition benchmark. The INEX dataset contains scientific articles from 18 journals and proceedings of the IEEE. We use a flat TF/IDF feature space (see Denoyer & Gallinari, 2006 for further details).

Table 1 also lists our choices for the parameter C and for the kernels $k(x, \bar{x})$. These choices were made on the basis of past experience. We use the same parameters for all algorithms because we mostly compare algorithms that optimize the same criterion. The kernel cache size was 500MB for all experiments.

4.2. Comparing Optimizers

Table 2 (top half) compares three optimization algorithms for the same dual cost (7).

- MCSVM (Crammer & Singer, 2001) uses the full gradient and therefore cannot be easily extended to handle structured output problems. We have used the MCSVM implementation distributed by the authors.

[1] `http://www.ics.uci.edu/~mlearn/databases`.
[2] `http://yann.lecun.com/exdb/mnist`.

Table 1. Datasets used for the experiments.

	Train Ex.	Test Ex.	Classes	Features	C	$k(x, \bar{x})$
LETTER	16000	4000	26	16	10	$e^{-0.025\|x-\bar{x}\|^2}$
USPS	7291	2007	10	256	10	$e^{-0.025\|x-\bar{x}\|^2}$
MNIST	60000	10000	10	780	1000	$e^{-0.005\|x-\bar{x}\|^2}$
INEX	6053	6054	18	167295	100	$x \cdot \bar{x}$

Table 2. Compared test error rates and training times.

		LETTER	USPS	MNIST	INEX
MCSVM (stores the full gradient)	Test error (%)	*2.42*	*4.24*	1.44	*26.26*
	Dual	*5548*	*537*	3718	235204
	Training time (sec.)	*1200*	*60*	*25000*	*520*
	Kernels ($\times 10^6$)	241	51.2	6908	32.9
SVMstruct (stores partial gradient)	Test error (%)	*2.40*	*4.38*	*1.40*	*26.25*
	Dual	5495	528	*3730*	*235631*
	Training time (sec.)	23000	6300	265000	14500
	Kernels ($\times 10^6$)	2083	1063.3	158076	n/a†
LaRankGap (stores partial gradient)	Test error (%)	*2.40*	*4.38*	*1.44*	*26.25*
	Dual	5462	518	3718	235183
	Training time (sec.)	2900	175	82000	1050
	Kernels ($\times 10^6$)	*156*	*13.7*	*4769*	*19.3*
LaRank×1 (online)	Test error (%)	2.80	**4.25**	**1.41**	27.20
	Dual	5226	503	3608	214224
	Training time (sec.)	**940**	85	30000	**300**
	Kernels ($\times 10^6$)	**55**	**9.4**	**399**	**17.2**

† Not applicable because SVMstruct bypasses the cache when using linear kernels.

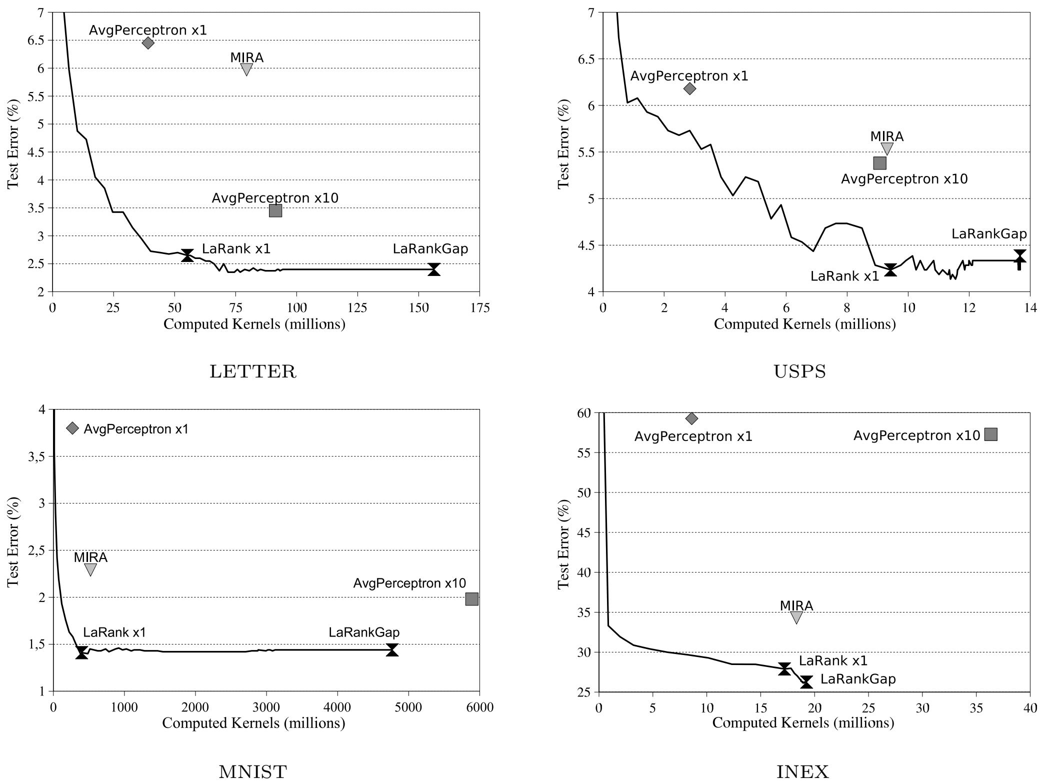

Figure 1. Evolution of the test error as a function of the number of kernel calculations

- SVMstruct (Tsochantaridis et al., 2005) targets structured output problems and therefore uses only a small fraction of the gradient. We have used the implementation distributed by the authors. The authors warn that this implementation has not been thoroughly optimized.
- LaRankGap iterates algorithm 5 until the duality gap becomes smaller than parameter C. This algorithm only stores a small fraction of the gradient, comparable to that used by SVMstruct. We have implemented LaRank using an *interpreted scripting language* with a specialized C function for algorithm 1 (SmoStep).

Both SVMstruct and LaRankGap use small subsets of the gradient coefficients. Although these subsets have similar size, LaRankGap avoids the training time penalty experienced by SVMstruct.

Both SVMstruct and LaRank make heavy use of kernel values involving two support patterns. In contrast, MCSVM updates the complete gradient vector after each step and therefore uses the kernel matrix rows corresponding to support patterns. On our relatively small problems, this stronger memory requirement is more than compensated by the lower overhead of MCSVM's simpler cache structure.

4.3. Comparing Online Learning Algorithms

Table 2 (bottom half) also reports the results obtained with a single LaRank epoch (LaRank×1). This single pass over the training examples is sufficient to nearly reach the optimal performance. This result is understandable because (*i*) online perceptrons offer strong theoretical guarantees after a single pass over the training examples, and (*ii*) LaRank drives the optimization process by replicating the randomization that happens in the perceptron.

For each dataset, figure 1 shows the evolution of the test error with respect to the number of kernel calculations. The point marked LaRank×1 corresponds to running a single LaRank epoch. The point marked LaRankGap corresponds to using the duality gap stopping criterion as explained in section 4.2. Figure 1 also reports results obtained with two popular online algorithms:

- The points marked AvgPerceptron×1 and AvgPerceptron×10 respectively correspond to performing one and ten epochs of the average perceptron algorithm (Freund & Schapire, 1998; Collins, 2002). Multiple epochs of the averaged perceptron are very effective when the necessary kernel values fit in the cache (first row). Training time increases considerably when this is not the case (second row.)

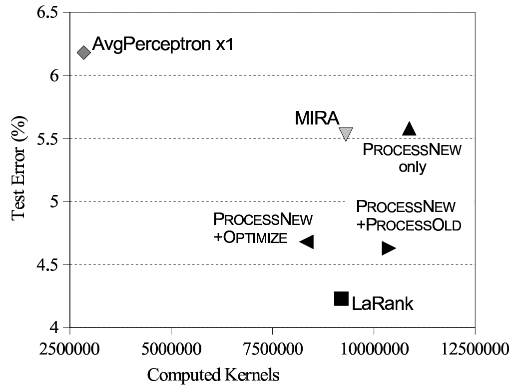

Figure 2. Impact of the LaRank operations (USPS dataset).

- The point marked MIRA corresponds to the online multiclass algorithm proposed by Crammer and Singer (2003). We have used the implementation provided by the authors as part of the MCSVM package. This algorithm computes more kernel values than AvgPerceptron×1 because its solution contains more support patterns. Its performance seems sensitive to the choice of kernel: Crammer and Singer (2003) report substantially better results using the same code but different kernels.

These results indicate that performing single LaRank epoch is an attractive online learning algorithm. Although LaRank×1 usually runs slower than AvgPerceptron×1 or MIRA, it provides better and more predictable generalization performance.

4.4. Comparing Optimization Strategies

Figure 2 shows the error rates and kernel calculations achieved when one restricts the set of operations chosen by algorithm 5. These results were obtained after a single pass on the USPS dataset.

As expected, using only the ProcessNew operation performs like MIRA. The average perceptron requires significantly less kernel calculations because its solution is much more sparse. However, it looses this initial sparsity when one performs several epochs (see figure 1.)

Enabling ProcessOld and Optimize significantly reduces the test error. The best test error is achieved when all operations are enabled. The number of kernel calculations is also reduced because ProcessOld and Optimize often eliminate support patterns.

4.5. Comparing ArgMax Calculations

The previous experiments measure the computational cost using training time and number of kernel calculations. Certain structured output problems use costly

Table 3. Numbers of arg max (in thousands).

	LETTER	USPS	MNIST	INEX
AvgPerceptron×1	16	7	60	6
AvgPerceptron×10	160	73	600	60
LaRank×1	190	25	200	28
LaRankGap	550	86	2020	73
SVMstruct	141	56	559	78

algorithms to find the class with the best score (1). The cost of this arg max calculation is partly related to the required number of new kernel values.

The average perceptron (and MIRA) performs one such arg max calculation for each example it processes. In contrast, LaRank performs one arg max calculation when processing a new example with ProcessNew, and also when running ProcessOld.

Table 3 compares the number of arg max calculations for various algorithms and datasets.[3] The SVMstruct optimizer performs very well with this metric. The AvgPerceptron and LaRank are very competitive on a single epoch and become more costly when performing many epochs. One epoch is sufficient to reach good performance with LaRank. This is not the case for the AvgPerceptron.

5. Conclusion

We have presented a large margin multiclass algorithm that uses gradients as sparingly as SVMstruct without experiencing the same training time penalty. LaRank can be considered an online algorithm because it nearly reaches its optimal performance in a single pass over the training examples. Under these conditions, LaRank achieves test error rates that are competitive with those of the full optimization, and significantly better than those achieved by perceptrons.

Acknowledgments

Nicolas Usunier helped proving the theorem bound. Part of this work was funded by NSF grant CCR-0325463. Antoine Bordes was also supported by the DGA and by the Network of Excellence IST–2002–506778 PASCAL.

References

Bakır, G., Hofmann, T., Schölkopf, B., Smola, A. J., Taskar, B., & Vishwanathan, S. V. N. (Eds.). (2007). *Predicting structured outputs*. MIT Press. in press.

Bordes, A., & Bottou, L. (2005). The Huller: a simple and efficient online SVM. *Machine Learning: ECML 2005* (pp. 505–512). Springer Verlag. LNAI 3720.

Bordes, A., Ertekin, S., Weston, J., & Bottou, L. (2005). Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, *6*, 1579–1619.

Collins, M. (2002). Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing* (pp. 1–8). Morristown, NJ: Association for Computational Linguistics.

Crammer, K., & Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, *2*, 265–292.

Crammer, K., & Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, *3*, 951–991.

Denoyer, L., & Gallinari, P. (2006). The XML document mining challenge. *Advances in XML Information Retrieval and Evaluation, 5th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2006*. Schloß Dagsthul, Germany.

Freund, Y., & Schapire, R. E. (1998). Large margin classification using the perceptron algorithm. *Machine Learning: Proceedings of the Fifteenth International Conference*. San Francisco, CA: Morgan Kaufmann.

Graepel, T., Herbrich, R., & Williamson, R. C. (2000). From margin to sparsity. In *Advances in neural information processing systems*, vol. 13, 210–216. MIT Press.

Hildreth, C. (1957). A quadratic programming procedure. *Naval Research Logistics Quarterly*, *4*, 79–85. Erratum, ibid. p361.

Hsu, C.-W., & Lin, C.-J. (2002). A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, *13*, 415–425.

LeCun, Y., Chopra, S., Hadsell, R., HuangFu, J., & Ranzato, M. (2007). A tutorial on energy-based learning. In (Bakır et al., 2007), 192–241. in press.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods – Support Vector Learning* (pp. 185–208). MIT Press.

Rifkin, R. M., & Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, *5*, 101–141.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels*. MIT Press.

Taskar, B. (2004). *Learning structured prediction models: A large margin approach*. Doctoral dissertation, Stanford University.

Taskar, B., Chatalbashev, V., Koller, D., & Guestrin, C. (2005). Learning structured prediction models: a large margin approach. *International Conference on Machine Learning (ICML)* (pp. 896–903).

Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, *6*, 1453–1484.

Weston, J., & Watkins, C. (1998). *Multi-class support vector machines* (Technical Report CSD-TR-98-04). Department of Computer Science, Royal Holloway, University of London, Egham, UK.

[3]The LETTER results in table 3 are outliers because the LETTER kernel runs as fast as the kernel cache. Since LaRank depends on timings, it often runs ProcessOld when a simple Optimize would have be sufficient.