A Support Vector Method for Multivariate Performance Measures

Thorsten Joachims TJ@CS.CORNELL.EDU
Cornell University, Dept. of Computer Science, 4153 Upson Hall, Ithaca, NY 14853 USA

Abstract

This paper presents a Support Vector Method for optimizing multivariate non-linear performance measures like the F_1-score. Taking a multivariate prediction approach, we give an algorithm with which such multivariate SVMs can be trained in polynomial time for large classes of potentially non-linear performance measures, in particular ROCArea and all measures that can be computed from the contingency table. The conventional classification SVM arises as a special case of our method.

1. Introduction

Depending on the application, measuring the success of a learning algorithm requires application specific performance measures. In text classification, for example, F_1-Score and Precision/Recall Breakeven Point (PRBEP) are used to evaluate classifier performance while error rate is not suitable due to a large imbalance between positive and negative examples. However, most learning methods optimize error rate, not the application specific performance measure, which is likely to produce suboptimal results. How can we learn rules that optimize measures other than error rate?

Current approaches that address this problem fall into three categories. Approaches of the first type aim to produce accurate estimates of the probabilities of class membership of each example (e.g. (Platt, 2000; Langford & Zadrozny, 2005)). While based on these probabilities many performance measures can be (approximately) optimized (Lewis, 1995), estimating the probabilities accurately is a difficult problem and arguably harder than the original problem of optimizing the particular performance measure. A second class of approaches circumvents this problem by optimizing many different variants of convenient and tractable performance measures, aiming to find one that performs well for the application specific performance measure after post-processing the resulting model (e.g. (Lewis, 2001; Yang, 2001; Abe et al., 2004; Caruana & Niculescu-Mizil, 2004)). However, in particular for non-linear performance measures like F_1-score or PRBEP, the relationship to tractable measures is at best approximate and requires extensive search via cross-validation. The final category of approaches aims to directly optimize the application specific performance measure. Such methods exist for some linear measures. In particular, most learning algorithms can be extended to incorporate unbalanced misclassification costs via linear loss functions (e.g. (Morik et al., 1999; Lin et al., 2002) in the context of SVMs). Also, methods for optimizing ROCArea have been proposed in the area of decision trees (Ferri et al., 2002), neural networks (Yan et al., 2003; Herschtal & Raskutti, 2004), boosting (Cortes & Mohri, 2003; Freund et al., 1998), and SVMs (Herbrich et al., 2000; Rakotomamonjy, 2004). However, for non-linear performance measures like F_1-score, the few previous attempts towards their direct optimization noted their computational difficulty (Musicant et al., 2003).

In this paper, we present a Support Vector Method that can directly optimize a large class of performance measures like F_1-score, Precision/Recall Breakeven Point (PRBEP), Precision at k ($\text{Prec}_{@k}$), and ROCArea. One difficulty common to most application specific performance measures is their non-linear and multivariate nature. This results in decision theoretic risks that no longer decompose into expectations over individual examples. To accommodate this problem, we propose an approach that is fundamentally different from most conventional learning algorithms: instead of learning a univariate rule that predicts the label of a single example, we formulate the learning problem as a multivariate prediction of all examples in the dataset. Based on the sparse approximation algorithm for structural SVMs (Tsochantaridis et al., 2004), we propose a method with which the training problem can be solved in polynomial time. We show that the

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.

method applies to any performance measure that can be computed from the contingency table, as well as to the optimization of ROCArea. The new method can be thought of as a direct generalization of classification SVMs, and we show that the conventional classification SVM arises as a special case when using error rate as the performance measure. We present experiments that compare our algorithm to a conventional classification SVMs with linear cost model and observe good performance without difficult to control heuristics.

2. Multivariate Performance Measures

In this section we first review the typical assumptions (often implicitly) made by most existing learning algorithms (Vapnik, 1998). This gives insight into why they are not suitable for directly optimizing non-linear performance measures like the F_1-Score.

Most learning algorithms assume that the training data $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n))$ as well as the test data S' is independently identically distributed (i.i.d.) according to a learning task $\Pr(X, Y)$. The goal is to find a rule $h \in \mathcal{H}$ from the hypothesis space $\mathcal{H}$ that optimizes the expected prediction performance on new samples S' of size n'.

$$R^{\Delta}(h) = \int \Delta\left((h(\mathbf{x}'_1), ..., h(\mathbf{x}'_{n'})), (y'_1, ..., y'_{n'})\right) d\Pr(S')$$

If the loss function Δ over samples decomposes linearly into a sum of a loss function δ over individual examples

$$\Delta((h(\mathbf{x}'_1), ..., h(\mathbf{x}'_{n'})), (y'_1, ..., y'_{n'})) = \frac{1}{n'}\sum_{i=1}^{n'} \delta(h(\mathbf{x}'_i), y'_i) \quad (1)$$

and since the examples are i.i.d., this expression can be simplified to

$$R^{\Delta}(h) = R^{\delta}(h) = \int \delta\left(h(\mathbf{x}'), y'\right) d\Pr(\mathbf{x}', y')$$

Discriminative learning algorithms approximate this expected risk $R^{\delta}(h)$ using the empirical risk on the training data S.

$$\hat{R}^{\delta}_S(h) = \frac{1}{n}\sum_{i=1}^{n} \delta\left(h(\mathbf{x}_i), y_i\right) \quad (2)$$

$\hat{R}^{\delta}_S(h)$ is an estimate of $R^{\delta}(h)$ for each $h \in \mathcal{H}$. Selecting a rule with low empirical risk $\hat{R}^{\delta}_S(h)$ (e.g. training error) in this decomposed form is the strategy followed by virtually all discriminative learning algorithms.

However, many performance measures (e.g. F_1, PRBEP) do not decompose linearly like in Eq. (1). They are a non-linear combination of the individual classifications. An example is the F_1 score $\Delta_{F_1}((h(\mathbf{x}_1), ..., h(\mathbf{x}_n)), (y_1, ..., y_n)) = \frac{2\,Prec\,Rec}{Prec+Rec}$, where $Prec$ and Rec are the precision and the recall of h on the sample $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)$. There is no known example-based loss function δ which can be used to decompose Δ. Therefore, learning algorithms restricted to optimizing an empirical risk of the kind in Eq. (2) are of questionable validity. What we need instead are learning algorithms that directly optimize an empirical risk that is based on the sample loss Δ.

$$\hat{R}^{\Delta}_S(h) = \Delta\left((h(\mathbf{x}_1), ..., h(\mathbf{x}_n)), (y_1, ..., y_n)\right)$$

Clearly, at least if the size n of the training set and the size n' of the test set are equal, $\hat{R}^{\Delta}_S(h)$ is again an estimate of $R^{\Delta}(h)$ for each $h \in \mathcal{H}$. Note that $\hat{R}^{\Delta}_S(h)$ does not necessarily have higher variance than a decomposed empirical risk $\hat{R}^{\delta}_S(h)$ just because it does not "average" over multiple examples. The key factor is the variance of Δ with respect to samples S drawn from $\Pr(X, Y)$. This variance can be low.

To design learning algorithms that do discriminative training with respect to $\hat{R}^{\Delta}_S(h)$, we need algorithms that find an $h \in \mathcal{H}$ that minimizes $\hat{R}^{\Delta}_S(h)$ over the training sample S. Since Δ is some non-linear function of S, this can be a challenging computational problem. We will now present a general approach to this problem based on Support Vector Machines.

3. SVM Approach to Optimizing Non-Linear Performance Measures

Support Vector Machines (SVMs) were developed by Vapnik et al. (Boser et al., 1992; Cortes & Vapnik, 1995; Vapnik, 1998) as a method for learning linear and, through the use of Kernels, non-linear rules. For the case of binary classification with unbiased hyperplanes[1], SVMs learn a classifier

$$h(\mathbf{x}) = sign\left[\mathbf{w}^T\mathbf{x}\right]$$

by solving the following optimization problem.

Optimization Problem 1. (UNBIASED SVM$_{org}$)

$$\min_{\mathbf{w},\xi\geq 0} \ \frac{1}{2}\,\mathbf{w}\cdot\mathbf{w} + C\sum_{i=1}^{n}\xi_i \quad (3)$$

$$s.t.: \quad \forall_{i=1}^{n} : y_i[\mathbf{w}\cdot\mathbf{x}_i] \geq 1 - \xi_i \quad (4)$$

The ξ_i are called slack variables. If a training example lies on the "wrong" side of the hyperplane, the corresponding ξ_i is greater than 1. Therefore $\sum_{i=1}^{n}\xi_i$ is an upper bound on the number of training errors. This means that the SVM finds a hyperplane classifier that

[1]Unbiased hyperplanes assume a threshold of 0 in the classification rule. This is not a substantial restriction, since a bias can be introduced by adding an artificial feature to each example.

optimizes an approximation of the training error regularized by the L_2 norm of the weight vector. The factor C in (3) controls the amount of regularization. To differentiate between different types of SVMs, we will denote this version as SVM_{org}.

In the following, we will use the same principles used in SVM_{org} to derive a class of SVM algorithms that optimize a broad range of non-linear performance measures. The key idea is to treat the learning problem as a multivariate prediction problem. Instead of defining our hypotheses h as a function from a single feature vector $\mathbf{x}$ to a single label $y \in \{-1, +1\}$,

$$h : \mathcal{X} \longrightarrow \mathcal{Y}$$

we will consider hypotheses $\bar{h}$ that map a tuple $\bar{\mathbf{x}} \in \bar{\mathcal{X}}$ of n feature vectors $\bar{\mathbf{x}} = (\mathbf{x}_1, ..., \mathbf{x}_n)$ to a tuple $\bar{y} \in \bar{\mathcal{Y}}$ of n labels $\bar{y} = (y_1, ..., y_n)$

$$\bar{h} : \bar{\mathcal{X}} \longrightarrow \bar{\mathcal{Y}},$$

where $\bar{\mathcal{X}} = \mathcal{X} \times ... \times X$ and $\bar{\mathcal{Y}} \subseteq \{-1, +1\}^n$ is the set of all admissible label vectors[2]. To implement this multivariate mapping, we will use linear discriminant functions of the following form.

$$\bar{h}_{\mathbf{w}}(\bar{\mathbf{x}}) = \underset{\bar{y}' \in \bar{\mathcal{Y}}}{\operatorname{argmax}} \left\{ \mathbf{w}^T \Psi(\bar{\mathbf{x}}, \bar{y}') \right\} \tag{5}$$

Intuitively, the prediction rule $\bar{h}_{\mathbf{w}}(\bar{\mathbf{x}})$ returns the tuple of labels $\bar{y}' = (y'_1, ..., y'_n)$ which scores highest according to a linear function. $\mathbf{w}$ is a parameter vector and Ψ is a function that returns a feature vector describing the match between $(\mathbf{x}_1, ..., \mathbf{x}_n)$ and $(y'_1, ..., y'_n)$. Whether this argmax can be computed efficiently hinges on the structure of Ψ. For the purposes of this paper, we can restrict Ψ to be of the following simple form:

$$\Psi(\bar{\mathbf{x}}, \bar{y}') = \sum_{i=1}^{n} y'_i \mathbf{x}_i$$

For this $\Psi(\bar{\mathbf{x}}, \bar{y})$ and $\bar{\mathcal{Y}} = \{-1, +1\}^n$, the argmax is achieved when y'_i is assigned to $h(\mathbf{x}_i)$. So, in terms of the resulting classification rule, this is equivalent to SVM_{org}. But did we gain anything from the reformulation of the prediction rule?

Thinking about the prediction problem in term of a multivariate rule $\bar{h}$ instead of a univariate rule h allows us to formulate the SVM optimization problem in a way that enables inclusion of a sample-based loss function Δ instead of the example-based loss function in SVM_{org}. Following (Tsochantaridis et al., 2004), we formulate the following alternative optimization problem for non-negative Δ.

[2]Note that $\bar{\mathcal{Y}}$ can be a strict subset for some measures, e.g. for $\text{Prec}_{@k}$ it is restricted to label vectors with k positive predictions.

Optimization Problem 2. (MULTIVAR. $\text{SVM}^{\Delta}_{multi}$)

$$\min_{\mathbf{w}, \xi \geq 0} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\,\xi$$

$$s.t. \quad \forall \bar{y}' \in \bar{\mathcal{Y}} \backslash \bar{y}: \; \mathbf{w}^T[\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}')] \geq \Delta(\bar{y}', \bar{y}) - \xi$$

Like for the SVM_{org}, this optimization problem is a convex quadratic program. In contrast to the SVM_{org}, however, there is one constraint for each possible $\bar{y} \in \bar{\mathcal{Y}}$. Due to the exponential size of $\bar{\mathcal{Y}}$, this may seem like an intractably large problem. However, by adapting the sparse approximation algorithm of (Tsochantaridis et al., 2004) implemented in SVM^{struct}[3], we will show that this problem can be solved in polynomial time for many types of multivariate loss functions Δ. Unlike in the SVM_{org} optimization problem there is only one slack variable ξ in this training problem. Similar to $\sum \xi_i$ in SVM_{org}, the value of this slack variable is an upper bound on the training loss.

Theorem 1. *At the solution $\mathbf{w}^*, \xi^*$ of the SVM^{Δ}_{multi} optimization problem on the training data $\bar{\mathbf{x}}$ with labels $\bar{y}$, the value of ξ^* is an upper bound on the training loss $\Delta(\bar{h}_{\mathbf{w}^*}(\bar{\mathbf{x}}), \bar{y})$.*

Proof. Let $\bar{y}' = \bar{h}_{\mathbf{w}^*}(\bar{\mathbf{x}})$ be the prediction of the learned multivariate hypothesis on the training data itself. Following from the definition of $\bar{h}$, this is the labeling $\bar{y}'$ that minimizes $\mathbf{w}^{*T}[\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}')]$, and this quantity will be less than zero unless $\bar{y}' = \bar{y}$. Therefore $\xi \geq \Delta(\bar{y}', \bar{y}) - \mathbf{w}^T[\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}')] \geq \Delta(\bar{y}', \bar{y})$. □

This shows that the multivariate $\text{SVM}^{\Delta}_{multi}$ is similar to the original SVM_{org} in the sense that it optimizes a convex upper bound on the training loss regularized by the norm of the weight vector. We will later show that, in fact, both formulations are identical if Δ is the number of training errors.

It is straightforward to extend the multivariate $\text{SVM}^{\Delta}_{multi}$ to non-linear classification rules via the dual representation of $\bar{h}$. Similar to the univariate SVM_{org}, the Wolfe dual of Optimization Problem 2 can be expressed in terms of inner products between feature vectors, allowing the use of kernels. We omit this extension for brevity.

4. Efficient Algorithm

How can the optimization problem of the multivariate $\text{SVM}^{\Delta}_{multi}$ be solved despite the huge number of constraints? This problem is a special case of the multivariate prediction formulations in (Tsochantaridis et al., 2004) as well as in (Taskar et al., 2003). The

[3]http://svmlight.joachims.org

Algorithm 1 Algorithm for solving quadratic program of multivariate $\text{SVM}^{\Delta}_{multi}$.

Input: $\bar{\mathbf{x}} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ $\bar{y} = (y_1, \ldots, y_n)$, C, ϵ, $\bar{\mathcal{Y}}$
$\mathcal{C} \leftarrow \emptyset$
repeat
  $\bar{y}' \leftarrow \text{argmax}_{\bar{y}' \in \bar{\mathcal{Y}}} \left\{ \Delta(\bar{y}', \bar{y}) + \mathbf{w}^T \Psi(\bar{\mathbf{x}}, \bar{y}') \right\}$
  $\xi \leftarrow \max\{0, \Delta(\bar{y}', \bar{y}) - \mathbf{w}^T [\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}')]\}$
  if $\Delta(\bar{y}', \bar{y}) - \mathbf{w}^T [\Psi(\bar{\mathbf{x}}, \bar{y}) - \Psi(\bar{\mathbf{x}}, \bar{y}')] > \xi + \epsilon$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\bar{y}'\}$
    $\mathbf{w} \leftarrow$ optimize $\text{SVM}^{\Delta}_{multi}$ objective over $\mathcal{C}$
  end if
until $\mathcal{C}$ has not changed during iteration
return($\mathbf{w}$)

algorithm proposed in (Taskar et al., 2003) for solving these types of large quadratic programs is not applicable to non-linear loss functions Δ, since it assumes that the loss decomposes linearly. The sparse approximation algorithm of (Tsochantaridis et al., 2004) does not have this restriction, and we will show in the following how it can be used to solve Optimization Problem 2 in polynomial time for a large class of loss functions Δ.

Algorithm 1 is the sparse approximation algorithm adapted to the multivariate $\text{SVM}^{\Delta}_{multi}$. The algorithm iteratively constructs a sufficient subset of the set of constraints in Optimization Problem 2. The algorithm starts with an empty set of constraints $\mathcal{C}$ and adds the currently most violated constraint in each iteration, i.e. the constraint corresponding to the label that maximizes $H(\bar{y}) = \left\{ \Delta(\bar{y}', \bar{y}) + \mathbf{w}^T \Psi(\bar{\mathbf{x}}, \bar{y}') \right\}$. The next approximation to the solution of Optimization Problem 2 is then computed on the new set of constraints. The algorithm stops when no constraint of Optimization Problem 2 is violated by more than ϵ. It is easy to see that the solution $\mathbf{w}^*$ returned by Algorithm 1 fulfills all constraints up to precision ϵ, and that the norm of $\mathbf{w}^*$ is no bigger than the norm of the exact solution of Optimization Problem 2. Furthermore, Tsochantaridis et al. (2004) show that the algorithm terminates after a polynomial number of iterations. We restate the theorem adapted to the $\text{SVM}^{\Delta}_{multi}$ optimization problem.

Theorem 2. *For any $\epsilon > 0$ and a training sample $\bar{\mathbf{x}} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and $\bar{y} = (y_1, \ldots, y_n)$ with $R = \max_i ||\mathbf{x}_i||$ and $L = \max_{\bar{y}' \in \bar{\mathcal{Y}}} \Delta(\bar{y}', \bar{y})$, Algorithm 1 terminates after incrementally adding at most*

$$\max \left\{ \frac{2L}{\epsilon}, \frac{8Cn^2R^2L}{\epsilon^2} \right\} \tag{6}$$

constraints to the working set $\mathcal{C}$.

The bound is rather lose. In our experiments we observe that the algorithm often converges after a few hundred iterations even for large problems. If the search for the most violated constraint

$$\underset{\bar{y}' \in \bar{\mathcal{Y}}}{\text{argmax}} \left\{ \Delta(\bar{y}', \bar{y}) + \mathbf{w}^T \Psi(\bar{\mathbf{x}}, \bar{y}') \right\} \tag{7}$$

can be performed in polynomial time, the overall algorithm has polynomial time complexity. We will show in the following that solving the argmax efficiently is indeed possible for a large class of multivariate loss functions Δ. We will first consider multivariate loss functions that can be computed from the contingency table, and then consider the case of ROC Area.

4.1. Loss Functions Based on Contingency Table

An exhaustive search over all $\bar{y}' \in \bar{\mathcal{Y}}$ is not feasible. However, the computation of the argmax in Eq. (7) can be stratified over all different contingency tables,

	y=1	y=-1
h($\mathbf{x}$)=1	a	b
h($\mathbf{x}$)=-1	c	d

so that each subproblem can be computed efficiently. Algorithm 2 is based on the observation that there are only order $O(n^2)$ different contingency tables for a binary classification problem with n examples. Therefore, any loss function $\Delta(a, b, c, d)$ that can be computed from the contingency table can take at most $O(n^2)$ different values.

Lemma 1. *Algorithm 2 computes the solution of*

$$\underset{\bar{y}' \in \bar{\mathcal{Y}}}{\text{argmax}} \left\{ \Delta(a, b, c, d) + \mathbf{w}^T \Psi(\bar{\mathbf{x}}, \bar{y}') \right\} \tag{8}$$

in polynomial time for any loss function $\Delta(a, b, c, d)$ that can be computed from the contingency table in polynomial time.

Proof. By iterating over all possible contingency tables, the algorithm iterates over all possible values l of $\Delta(a, b, c, d)$. For each contingency table (a, b, c, d) it computes the argmax over all $\bar{\mathcal{Y}}_{abcd}$, which is the set of $\bar{y}$ that correspond to this contingency table.

$$\begin{aligned} \bar{y}_{abcd} &= \underset{\bar{y}' \in \bar{\mathcal{Y}}_{abcd}}{\text{argmax}} \left\{ \mathbf{w}^T \Psi(\bar{\mathbf{x}}, \bar{y}') \right\} & (9) \\ &= \underset{\bar{y}' \in \bar{\mathcal{Y}}_{abcd}}{\text{argmax}} \left\{ \sum_{i=1}^{n} y'_i (\mathbf{w}^T \mathbf{x}_i) \right\} & (10) \end{aligned}$$

Since the objective function is linear in $\bar{y}'$, the solution can be computed by maximizing $\bar{y}'$ element wise. The maximum value for a particular contingency table is achieved when the a positive examples with the largest value of $(\mathbf{w}^T \mathbf{x}_i)$ are classified as positive, and the d negative examples with the lowest value of $(\mathbf{w}^T \mathbf{x}_i)$ are

Algorithm 2 Algorithm for computing argmax with loss functions that can be computed from the contingency table.

Input: $\bar{\mathbf{x}} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, $\bar{y} = (y_1, \ldots, y_n)$, and $\bar{\mathcal{Y}}$
$(i_1^p, \ldots, i_{\#pos}^p) \leftarrow$ sort $\{i : y_i = 1\}$ by $\mathbf{w}^T\mathbf{x}_i$
$(i_1^n, \ldots, i_{\#neg}^n) \leftarrow$ sort $\{i : y_i = -1\}$ by $\mathbf{w}^T\mathbf{x}_i$
for $a \in [0, \ldots, \#pos]$ **do**
    $c \leftarrow \#pos - a$
    set $y'_{i_1^p}, \ldots, y'_{i_a^p}$ to 1 AND set $y'_{i_{a+1}^p}, \ldots, y'_{i_{\#pos}^p}$ to -1
    for $d \in [0, \ldots, \#neg]$ **do**
        $b \leftarrow \#neg - d$
        set $y'_{i_1^n}, \ldots, y'_{i_b^n}$ to 1 AND set $y'_{i_{b+1}^n}, \ldots, y'_{i_{\#neg}^n}$ to -1
        $v \leftarrow \Delta(a, b, c, d) + \mathbf{w}^T \sum_{i=1}^n y'_i \mathbf{x}_i$
        if v is the largest so far **then**
            $\bar{y}^* \leftarrow (y'_1, ..., y'_n)$
        end if
    end for
end for
return($\bar{y}^*$)

classified as negative. The overall argmax can be computed by maximizing over the stratified maxima plus their constant loss. □

By slightly rewriting the algorithm, it can be implemented to run in time $O(n^2)$. Exploiting that many loss functions are upper bounded, pruning can further improve the runtime of the algorithm. We will now give some examples of how this algorithm applies to the loss functions we will later use in experiments.

F_β-Score: The F_β-Score is a measure typically used to evaluate binary classifiers in natural language applications like text classification. It is particularly preferable over error rate for highly unbalanced classes. The F_β-Score is a weighted harmonic average of Precision and Recall. It can be computed from the contingency table as

$$\mathrm{F}_\beta(h) = \frac{(1+\beta^2)\, a}{(1+\beta^2)\, a + b + \beta^2 c}. \tag{11}$$

The most common choice for β is 1. For the corresponding loss $\Delta_{F_1}(\bar{y}', \bar{y}) = 100(1 - \mathrm{F}_\beta)$, Algorithm 2 directly applies.

Precision/Recall at k In Web search engines, most users scan only the first few links that are presented. Therefore, a common way to evaluate such systems is to measure precision only on these (e.g. ten) positive predictions. Similarly, in an archival retrieval system not precision, but recall might be the most indicative measure. For example, what fraction of the total number of relevant documents did a user find after scanning the top 100 documents. Following this intuition, $\mathrm{Prec}_{@k}$ and $\mathrm{Rec}_{@k}$ measure the precision and recall of a classifier that predicts exactly k documents to be positive.

$$\mathrm{Prec}_{@k}(h) = \frac{a}{a+b} \qquad \mathrm{Rec}_{@k}(h) = \frac{a}{b+d} \tag{12}$$

For these measures, the space of possible prediction vectors $\bar{\mathcal{Y}}$ is restricted to those that predict exactly k examples to be positive. For this $\bar{\mathcal{Y}}$, the multivariate discriminant rule $\bar{h}_{\mathbf{w}}(\bar{\mathbf{x}})$ in Eq. (5) can be computed by assigning label 1 to the k examples with highest $\mathbf{w}^T\mathbf{x}_i$. Similarly, a restriction to this $\bar{\mathcal{Y}}$ can easily be incorporated into Algorithm 2 by excluding all $\bar{y}' \neq \bar{y}$ from the search for which $a + b \neq k$.

Precision/Recall Break-Even Point The Precision/Recall Break-Even Point (PRBEP) is a performance measure that is often used to evaluate text classifiers. It requires that the classifier makes a prediction $\bar{y}$ so that precision and recall are equal, and the value of the PRBEP is defined to be equal to both. As is obvious from the definition of precision and recall, this equality is achieved for contingency tables with $a + b = a + c$ and we restrict $\bar{\mathcal{Y}}$ appropriately. Again, we define the corresponding loss as $\Delta_{PRBEP}(\bar{y}', \bar{y}) = 100(1 - PRBEP)$ and it is straightforward to compute $\bar{h}_{\mathbf{w}}(\bar{\mathbf{x}})$ and modify Algorithm 2 for this $\bar{\mathcal{Y}}$.

4.2. ROC Area

ROCArea is a performance measure that cannot be computed from the contingency table, but requires predicting a ranking. However, both SVM_{org} and $\mathrm{SVM}^{\Delta}_{multi}$ naturally predict a ranking by ordering all examples according to $\mathbf{w}^T\mathbf{x}_i$. From such a ranking, ROCArea can be computed from the number of swapped pairs

$$SwappedPairs = |\{(i,j) : (y_i > y_j) \text{ and } (\mathbf{w}^T\mathbf{x}_i < \mathbf{w}^T\mathbf{x}_j)\}|,$$

i.e. the number of pairs of examples that are ranked in the wrong order.

$$ROCArea = 1 - \frac{SwappedPairs}{\#pos \cdot \#neg} \tag{13}$$

We can adapt the $\mathrm{SVM}^{\Delta}_{multi}$ to optimizing ROCArea by (implicitly) considering a classification problem of all $\#pos \cdot \#neg$ pairs (i, j) of a positive example $(\mathbf{x}_i, 1)$ and a negative example $(\mathbf{x}_j, -1)$, forming a new classification problem $\bar{\bar{\mathcal{X}}}$ and $\bar{\bar{\mathcal{Y}}} = \{-1, 1\}^{\#pos \cdot \#neg}$ as follows. Each pos/neg pair (i, j) receives the target

Algorithm 3 Algorithm for computing argmax with ROCArea-loss.

Input: $\bar{\mathbf{x}} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, $\bar{y} = (y_1, \ldots, y_n)$
for $i \in \{i : y_i = 1\}$ **do** $s_i \leftarrow -0.25 + \mathbf{w}^T\mathbf{x}_i$
for $i \in \{i : y_i = -1\}$ **do** $s_i \leftarrow 0.25 + \mathbf{w}^T\mathbf{x}_i$
$(r_1, \ldots, r_n) \leftarrow$ sort $\{1, \ldots, n\}$ by s_i
$s_p = \#pos$, $s_n = 0$
for $i \in \{1, \ldots, n\}$ **do**
  if $y_{r_i} > 0$ **then**
    $c_{r_i} \leftarrow (\#neg - 2\, s_n)$
    $s_p \leftarrow s_p - 1$
  else
    $c_{r_i} \leftarrow (-\#pos + 2\, s_p)$
    $s_n \leftarrow s_n + 1$
  end if
end for
return$(c_1, \ldots, c_n)$

label $y_{ij} = 1$ and is described by the feature vector $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$. In this representation, the discriminant rule $\bar{\bar{h}}_{\mathbf{w}}(\bar{\bar{\mathbf{x}}}) = \text{argmax}_{\bar{\bar{y}}' \in \bar{\bar{\mathcal{Y}}}} \left\{\mathbf{w}^T\Psi(\bar{\bar{\mathbf{x}}}, \bar{\bar{y}}')\right\}$ corresponds to labeling a pair (i,j) as $sign(\mathbf{w}^T\mathbf{x}_i - \mathbf{w}^T\mathbf{x}_j)$, i.e. according to the ordering w.r.t $\mathbf{w}^T\mathbf{x}_i$ as desired. Note that the error between the prediction $\bar{\bar{y}}'$ and the true pairwise labels $\bar{\bar{y}} = (1, ..., 1)^T$ is proportional to $1 - ROCArea$ of the original data $\bar{\mathbf{x}}$ and $\bar{y}$. We call this quantity the ROCArea-loss.

$$\Delta_{ROCArea}(\bar{\bar{y}}', \bar{\bar{y}}) = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{2}(1 - y'_{ij}) = SwappedPairs$$

Actually representing all $\#pos \cdot \#neg$ pairs would be rather inefficient, but can be avoided using the following representation which is linear in the number of examples n.

$$\Psi(\bar{\bar{\mathbf{x}}}, \bar{\bar{y}}) = \sum_{i=1}^{n} c_i\mathbf{x}_i \text{ with } c_i = \begin{cases} \sum_{j=1}^{\#neg} y_{ij}, & if(y_i = 1) \\ -\sum_{j=1}^{\#pos} y_{ji}, & if(y_i = -1) \end{cases}$$

Note that $\Delta_{ROCArea}(\bar{\bar{y}}', \bar{\bar{y}})$ can now be computed as $\Delta_{ROCArea}(\bar{\bar{y}}', \bar{\bar{y}}) = \frac{1}{2}\sum_{i=1}^{n} y_i(c_i - c'_i)$. Algorithm 3 computes the argmax in this representation.

Lemma 2. *For $\bar{\mathbf{x}}$ and $\bar{y}$ of size n, Algorithm 3 computes the solution $c_1, \ldots, c_n$ corresponding to*

$$\underset{\bar{\bar{y}}' \in \bar{\bar{\mathcal{Y}}}}{\text{argmax}} \left\{\Delta_{ROCArea}(\bar{\bar{y}}', \bar{\bar{y}}) + \mathbf{w}^T\Psi(\bar{\bar{\mathbf{x}}}, \bar{\bar{y}}')\right\} \qquad (14)$$

in time $O(n \log n)$.

Proof. The argmax can be written as follows in the pairwise representation.

$$\bar{\bar{y}}^* = \underset{\bar{\bar{y}}' \in \bar{\bar{\mathcal{Y}}}}{\text{argmax}} \sum_{i=1}^{\#pos}\sum_{j=1}^{\#neg}\frac{1}{2}(1 - y'_{ij}) + y'_{ij}\mathbf{w}^T(\mathbf{x}_i - \mathbf{x}_j)$$

Since the loss function decomposes linearly over the pairwise representation, we can maximize each y_{ij} individually.

$$\begin{aligned} y^*_{ij} &= \underset{y'_{ij} \in \{-1,+1\}}{\text{argmax}} \frac{1}{2}(1 - y'_{ij}) + y'_{ij}\mathbf{w}^T(\mathbf{x}_i - \mathbf{x}_j) \\ &= \underset{y'_{ij} \in \{-1,+1\}}{\text{argmax}} y_{ij}[(\mathbf{w}^T\mathbf{x}_i - \frac{1}{4}) - (\mathbf{w}^T\mathbf{x}_j + \frac{1}{4})] \end{aligned}$$

This means that a pair (i,j) should be labeled as $y_{ij} = 1$, if the score $\mathbf{w}^T\mathbf{x}_i$ of the positive example decremented by $\frac{1}{4}$ is larger than the score $\mathbf{w}^T\mathbf{x}_j$ of the negative example incremented by $\frac{1}{4}$. This is precisely how the algorithm assigns the labels and collects them in the compressed representation. The runtime of the algorithm is dominated by a single sort operation. $\square$

5. SVM$^{\Delta}_{multi}$ Generalizes SVM$_{org}$

The following theorem shows that our multivariate SVM$^{\Delta}_{multi}$ is a direct generalization of the conventional classification SVM. When using error rate as the loss function, the conventional SVM arises as a special case of SVM$^{\Delta}_{multi}$.

Theorem 3. *Using error as the loss function, in particular $\Delta_{Err}(\bar{y}', \bar{y}) = 2\,(b + c)$, SVM^{Err}_{multi} with regularization parameter C_{multi} computes the same hyperplane $\mathbf{w}$ as SVM_{org} with $C_{org} = 2\, C_{multi}$.*

Proof. We will show that both optimization problems have the same objective value and an equivalent set of constraints. In particular, for every $\mathbf{w}$ the smallest feasible ξ and $\sum_i \xi_i$ are related as $\xi = 2\sum_i \xi_i$.

For a given $\mathbf{w}$, the ξ_i in SVM$_{org}$ can be optimized individually, and the optimum is achieved for $\xi_i = max\{0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i)\}$. For the SVM$^{Err}_{multi}$, the optimal ξ for a given $\mathbf{w}$ is

$$\xi = \max_{\bar{y}' \in \bar{\mathcal{Y}}} \left\{\Delta_{Err}(\bar{y}', \bar{y}) + \sum_{i=1}^{n} y'_i\mathbf{w}^T\mathbf{x}_i - \sum_{i=1}^{n} y_i\mathbf{w}^T\mathbf{x}_i\right\}$$

Since the function is linear in the y'_i, each y'_i can be optimized independently. Denote with $\delta_{Err}(y'_i, y_i)$ the univariate loss function that returns 2 if both arguments differ, and 0 otherwise.

$$\begin{aligned} \xi &= \sum_{i=1}^{n} \max_{y'_i \in \{-1,+1\}} \left\{\delta_{Err}(y'_i, y_i) + y'_i\mathbf{w}^T\mathbf{x}_i - y_i\mathbf{w}^T\mathbf{x}_i\right\} \\ &= \sum_{i=1}^{n} \max\left\{0, 2 - 2y_i\mathbf{w}^T\mathbf{x}_i\right\} = 2\sum_{i=1}^{n}\xi_i \end{aligned}$$

Therefore, if $C_{org} = 2\, C_{multi}$, the objective functions of both optimization problems are equal for any $\mathbf{w}$, and consequently so are their optima w.r.t. $\mathbf{w}$. $\square$

Table 1. Comparing an SVM optimized for the performance measure to one that is trained with linear cost model.

Dataset	Method	F_1	PRBEP	Rec$_{@2p}$	ROCArea
Reuters (90 classes)	$\text{SVM}^{\Delta}_{multi}$	62.0	68.2	78.3	99.1
Examples: 9603/3299	SVM_{org}	56.1	65.7	77.2	98.6
Features: 27658	improvement	+5.9 (51/20)**	+2.5 (16/8)**	+1.1 (14/8)	+0.5 (43/33)*
ArXiv (14 classes)	$\text{SVM}^{\Delta}_{multi}$	56.8	58.4	73.3	92.8
Examples: 1168/32487	SVM_{org}	49.6	57.9	74.4	92.7
Features: 13525	improvement	+7.2 (9/5)*	+0.5 (9/4)	-1.1 (1/13)**	+0.1 (8/6)
Optdigits (10 classes)	$\text{SVM}^{\Delta}_{multi}$	92.5	92.7	98.4	99.4
Examples: 3823/1797	SVM_{org}	91.5	91.5	98.7	99.4
Features: 64	improvement	+1.0 (8/2)*	+1.2 (5/1)*	-0.3 (1/5)	0.0 (6/4)
Covertype (7 classes)	$\text{SVM}^{\Delta}_{multi}$	73.8	72.1	93.1	94.6
Examples: 1000/2000	SVM_{org}	73.9	71.0	94.7	94.1
Features: 54	improvement	-0.1 (3/4)	+1.1 (5/2)	-1.6 (2/5)	+0.5 (4/3)

6. Experiments

To evaluate the proposed SVM approach to optimizing non-linear performance measures, we conducted experiments on four different test collection. We compare F_1-score, PRBEP, Rec$_{@k}$ for k twice the number of positive examples (Rec$_{@2p}$), and ROCArea achieved by the respective $\text{SVM}^{\Delta}_{multi}$ with the performance of a classification SVM that includes a cost model. The cost model is implemented by allowing different regularization constants for positive and negative examples (Morik et al., 1999). Using the parameter j of SVM^{light}, the C parameter of positive examples is multiplied by j to increase their influence. This setup is a strong baseline to compare against. For example, David Lewis won the TREC-2001 Batch Filtering Evaluation (Lewis, 2001) using SVM^{light} with such cost models. Furthermore, Musicant et al. (2003) make a theoretical argument that such cost models approximately optimize F_1-score.

We compare performance on four test collections, namely the ModApte Reuters-21578 text classification benchmark[4], a dataset of abstracts from the Physics E-Print ArXiv, and the OPTDIGITS and COVERTYPE benchmarks [5]. Train/test split and the number of features are given in Table 1.

Initial experiments indicated that biased hyperplane (i.e. adjustable threshold) outperform unbiased hyperplanes. We therefore add a constant feature with value 1 to each training example for the $\text{SVM}^{\Delta}_{multi}$ and use biased hyperplanes for the regular SVM as implemented in SVM^{light}. To select the regularization parameter C for the $\text{SVM}^{\Delta}_{multi}$, and C and j for the classification SVM, we used holdout testing with a random $\frac{2}{3}$ / $\frac{1}{3}$ split of the training set for each class in a collection. We search within $C \in [2^{-6}, ..., 2^{6}]$ and $j \in [2^{0}, ..., 2^{7}]$, but extended the search space if the most frequently selected parameter setting over all classes in the collection was on a boundary.

Our implementation of $\text{SVM}^{\Delta}_{multi}$ is available at `http://svmlight.joachims.org`.

Table 1 shows the macro-average of the performance over all classes in a collection. Each "improvement" line shows the amount by which the $\text{SVM}^{\Delta}_{multi}$ outperforms (or underperforms) the regular SVM. Both the difference in performance, as well as the number of classes on which the $\text{SVM}^{\Delta}_{multi}$ won/lost are shown. Stars indicate the level of significance according to a two-tailed Wilcoxon test applied to pairs of results over classes. One star indicates a significance level of 0.9, two stars a level of 0.95. Overall, 11 macroaverages in Table 1 show an improvement (6 significant), and only 4 cases (1 significant) show a decline in performance. Comparing the results between datasets, the improvements are largest for the two text classification tasks, while especially for COVERTYPE there is no significant difference between the two methods. With respect to different performance measures, the largest gains are observed for F_1-score on the text classification tasks. PRBEP and ROCArea also show consistent, but smaller gains. On Rec$_{@2p}$, the regular SVM appears to perform better on average.

Figure 1 further analyzes how the performance differs between the individual binary tasks in the Reuters collection. The 90 tasks were binned into 5 sets by their ratio of positive to negative examples. Figure 1 plots the average performance improvement in each bin from the most popular classes on the left to the least popular classes on the right. For most measures, especially F_1-score, improvements are larger for the less popular categories.

[4] `http://www.daviddlewis.com/`

[5] `http://www.ics.uci.edu/~mlearn/MLRepository.html`

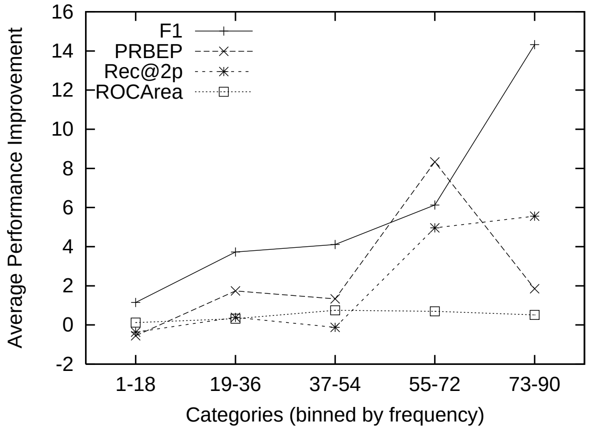

Figure 1. Improvement in prediction performance on Reuters of $\text{SVM}^{\Delta}_{multi}$ over SVM_{org} depending on the balance between positive and negative examples. Results are averaged by binning the 90 categories according to their number of examples.

7. Conclusions

This paper generalized SVMs to optimizing large classes of multivariate non-linear performance measures often encountered in practical applications. We presented a training algorithm and showed that is it computationally tractable. The new approach leads to improved performance particularly for text classification problems with highly unbalanced classes. Furthermore, it provides a principled approach to optimizing such measures and avoids difficult to control heuristics.

This work was funded in part under NSF awards IIS-0412894 and IIS-0412930.

References

Abe, N., Zadrozny, B., & Langford, J. (2004). An iterative method for multi-class cost-sensitive learning. *Proc. KDD*.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A traininig algorithm for optimal margin classifiers. *Proc. COLT*.

Caruana, R., & Niculescu-Mizil, A. (2004). Data mining in metric space: an empirical analysis of supervised learning performance criteria. *Proc. KDD*.

Cortes, C., & Mohri, M. (2003). Auc optimization vs. error rate minimization. *Proc. NIPS*.

Cortes, C., & Vapnik, V. N. (1995). Support–vector networks. *Machine Learning*, *20*, 273–297.

Ferri, C., Flach, P., & Hernandez-Orallo, J. (2002). Learning decision trees using the area under the roc curve. *Proc. ICML*.

Freund, Y., Iyer, R., Schapire, R., & Singer, Y. (1998). An efficient boosting algorithm for combining preferences. *Proc. ICML*.

Herbrich, R., Graepel, T., & Obermayer, K. (2000). Large margin rank boundaries for ordinal regression. In et A. S. al. (Ed.), *Advances in large margin classifiers*. MIT Press.

Herschtal, A., & Raskutti, B. (2004). Optimising area under the roc curve using gradient descent. *Proc. ICML*.

Langford, J., & Zadrozny, B. (2005). Estimating class membership probabilities using classifier learners. *Proc. AISTATS*.

Lewis, D. (1995). Evaluating and optimizing autonomous text classification systems. *Proc. SIGIR*.

Lewis, D. (2001). Applying support vector machines to the trec-2001 batch filtering and routing tasks. *Proc. TREC*.

Lin, Y., Lee, Y., & Wahba, G. (2002). Support vector machines for classification in nonstandard situations. *Machine Learning*, *46*, 191 – 202.

Morik, K., Brockhausen, P., & Joachims, T. (1999). Combining statistical learning with a knowledge-based approach. *Proc. ICML*.

Musicant, D., Kumar, V., & Ozgur, A. (2003). Optimizing f-measure with support vector machines. *Proc. FLAIRS*.

Platt, J. (2000). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In et A. S. al. (Ed.), *Advances in large margin classifiers*. MIT Press.

Rakotomamonjy, A. (2004). *Svms and area under roc curve* (Technical Report). PSI-INSA de Rouen.

Taskar, B., Guestrin, C., & Koller, D. (2003). Maximum-margin markov networks. *Proc. NIPS*.

Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. *Proc. ICML*.

Vapnik, V. (1998). *Statistical learning theory*. Wiley.

Yan, L., Dodier, R., Mozer, M., & Wolniewicz, R. (2003). Optimizing classifier performance via approximation to the wilcoxon-mann-witney statistic. *Proc. ICML*.

Yang, Y. (2001). A study of thresholding strategies for text categorization. *Proc. SIGIR*.