Extension of TSVM to Multi-Class and Hierarchical Text Classification Problems With General Losses

S.Sathiya Keerthi(1) *S.Sundararajan*(2) *Shirish Shevade*(3)
(1) Cloud and Information Services Lab, Microsoft, Mountain View, CA 94043
(2) Microsoft Research India, Bangalore, India
(3) Computer Science and Automation, Indian Institute of Science, Bangalore, India
keerthi@microsoft.com, ssrajan@microsoft.com, shirish@csa.iisc.ernet.in

Abstract

Transductive SVM (TSVM) is a well known semi-supervised large margin learning method for binary text classification. In this paper we extend this method to multi-class and hierarchical classification problems. We point out that the determination of labels of unlabeled examples with fixed classifier weights is a linear programming problem. We devise an efficient technique for solving it. The method is applicable to general loss functions. We demonstrate the value of the new method using large margin loss on a number of multi-class and hierarchical classification datasets.

1 Introduction

Consider the following supervised learning problem corresponding to a general structured output prediction problem:

$$\min_{\mathbf{w},\xi^s} \; F^s(\mathbf{w}) = \; \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{l}\sum_{i=1}^{l}\xi_i^s \tag{1}$$

where $\xi_i^s = \xi(\mathbf{w}, \mathbf{x}_i^s, y_i^s)$ is the loss term and $\{(\mathbf{x}_i^s, y_i^s)\}_{i=1}^{l}$ is the set of labeled examples. For example, in large margin and maxent models respectively we have

$$\xi(\mathbf{w}, \mathbf{x}_i, y_i) = \max_y L(y, y_i) - \mathbf{w}^T \Delta\mathbf{f}(y, y_i; \mathbf{x}_i) \;\; and \;\; \xi(\mathbf{w}, \mathbf{x}_i, y_i) = -\mathbf{w}^T \mathbf{f}(y_i; \mathbf{x}_i) + \log Z \tag{2}$$

where $\Delta\mathbf{f}(y, y_i; \mathbf{x}_i) = \mathbf{f}(y_i; \mathbf{x}_i) - \mathbf{f}(y; \mathbf{x}_i)$ and $Z = \sum_y \exp(\mathbf{w}^T \mathbf{f}(y; \mathbf{x}_i))$. Text classification problems involve a rich and large feature space (e.g., bag-of-words features) and so linear classifiers work very well (Joachims, 1999). We particularly focus on multi-class and hierarchical classification problems (and hence our use of scalar notation for y). In multi-class problems y runs over the classes and, $\mathbf{w}$ and $\mathbf{f}(y; \mathbf{x}_i)$ have one component for each class, with the component corresponding to y turned on. More generally, in hierarchical classification problems, y runs over the set of leaf nodes of the hierarchy and, $\mathbf{w}$ and $\mathbf{f}(y; \mathbf{x}_i)$ consist of one component for each node of the hierarchy, with the node components in the path to leaf node y turned on. $\lambda > 0$ is a regularization parameter. A good default value for λ can be chosen depending on the loss function used.[1] The superscript s denotes 'supervised'; we will use superscript u to denote elements corresponding to unlabeled examples.

In semi-supervised learning we use a set of unlabeled examples, $\{\mathbf{x}_i^u\}_{i=1}^{n}$ and include the determination of the labels of these examples as part of the training process:

$$\min_{\mathbf{w},\mathbf{y}^u} \; F^s(\mathbf{w}) + \frac{C^u}{n}\sum_{i=1}^{n}\xi_i^u \;\; \text{s.t.} \;\; \sum_{i=1}^{n}\delta(y, y_i^u) = n(y) \;\; \forall y \tag{3}$$

where $\mathbf{y}^u = \{y_i^u\}$, $\xi_i^u = \xi(\mathbf{w}, \mathbf{x}_i^u, y_i^u)$ and δ is the Kronecker delta function. C^u is a regularization parameter for the unlabeled part. A good default value is $C^u = 1$; we use this value in all our experiments. (3) consists of constraints on the label counts that come from domain knowledge. (In practice, one specifies $\phi(y)$, the fraction of examples in class y; then the values in $\{\phi(y)n\}$ are rounded to integers $\{n(y)\}$ in a suitable way so that $\sum_y n(y) = n$.[2]) Such constraints are crucial for the effective solution of the semi-supervised learning problem; without them the semi-supervised solution tends to move towards assigning the majority class label to most unlabeled examples. In more general structured prediction problems (3) may include other domain constraints (Chang et al., 2007). In this paper we will use just the label constraints in (3).

Inspired by the effectiveness of the TSVM model of Joachims (1999), there have been a number of works on the solution of (3) for binary classification with large margin losses. These methods fall into one of two types: (a) combinatorial optimization; and (b) continuous optimization.

[1]In the experiments of this paper, for multi-class and hierarchical classification with large margin loss, we use $\lambda = 10$.

[2]We will assume that quite precise values are given for $\{n(y)\}$. The effect of noise in these values on the semi-supervised solution needs a separate study.

See (Chapelle et al., 2008, 2006) for a detailed coverage of various specific methods falling into these two types. In combinatorial optimization the label set $\mathbf{y}^u$ is determined together with $\mathbf{w}$. It is usual to use a sequence of alternating optimization steps (fix $\mathbf{y}^u$ and solve for $\mathbf{w}$, and then fix $\mathbf{w}$ and solve for $\mathbf{y}^u$) to obtain the solution. An important advantage of doing this is that each of the sub-optimization problems can be solved using simple and/or standard solvers. In continuous optimization $\mathbf{y}^u$ is eliminated and the resulting (non-convex) optimization problem is solved for $\mathbf{w}$ by minimizing

$$F^s(\mathbf{w}) + \frac{C^u}{n}\sum_{i=1}^{n}\rho(\mathbf{w},\mathbf{x}_i^u) \tag{4}$$

where $\rho(\mathbf{w},\mathbf{x}_i^u) = \min_{y^u}\xi(\mathbf{w},\mathbf{x}_i^u,y_i^u)$. The loss function ξ as well as ρ are usually smoothed so that the objective function is differentiable and gradient-based optimization techniques can be employed. Further, the constraints in (3) involving $\mathbf{y}^u$ are replaced by smooth constraints on $\mathbf{w}$ expressing balance of the mean outputs of each label over the labeled and unlabeled sets.

Zien et al. (2007) extended the continuous optimization approach to (4) for multi-class and structured output problems. But their experiments only showed limited improvement over supervised learning. The combinatorial optimization approach, on the other hand, has not been carefully explored beyond binary classification. Methods based on semi-definite programming (Xu et al., 2006; De Bie and Cristianini, 2004) are impractical, even for medium size problems. One-versus-rest and one-versus-one ideas have been tried, but it is unclear if they work well: Zien et al. (2007) and Zubiaga et al. (2009) report failure while Bruzzone et al. (2006) use a heuristic implementation and report success in one application domain. Unlike these methods which have binary TSVM as the basis, we take up an implementation of the approach for the direct multi-class and hierarchical classification formulation in (3). The special structure in constraints allows the $\mathbf{y}^u$ determination step to reduce to a degenerate transportation linear programming problem. So the well-known transportation simplex method can be used to obtain $\mathbf{y}^u$. We show that even this method is not efficient enough. As an alternative we suggest an effective and much more efficient heuristic label switching algorithm. For binary classification problems this algorithm is an improved version of the multiple switching algorithm developed by Sindhwani and Keerthi (2006) for TSVM. Experiments on a number of multi-class and hierarchical classification datasets show that, like the TSVM method of binary classification, our method yields a strong lift in performance over supervised learning, especially when the number of labeled examples is not sufficiently large. Although we demonstrate our method using hinge loss, the applicability of our approach to general loss functions (e.g., maxent loss) is a key advantage. The reader is referred to the longer version of this paper (Keerthi et al., 2012) for details on specialization to maxent losses ((Gärtner et al., 2005), (Graca et al., 2007), (Ganchev et al., 2009) and (Mann and McCallum, 2010)) and more experimental results.

2 Semi-Supervised Learning Algorithm

The semi-supervised learning algorithm for multi-class and hierarchical classification problems follows the spirit of the TSVM algorithm (Joachims, 1999). Algorithm 1 gives the steps. It consists of an initialization part (steps 1-9) that sets starting values for $\mathbf{w}$ and $\mathbf{y}^u$, followed by an iterative part (steps 10-15) where $\mathbf{w}$ and $\mathbf{y}^u$ are refined by semi-supervised learning. Using exactly the same arguments as those in (Joachims, 1999; Sindhwani and Keerthi, 2006) it can be proved that Algorithm 1 is convergent.

Initialization of $\mathbf{w}$ is done by solving the supervised learning problem. This $\mathbf{w}$ can be used to predict $\mathbf{y}^u$. However such a $\mathbf{y}^u$ usually violates the constraints in (3). To choose a $\mathbf{y}^u$ that satisfies (3), we do a greedy modification of the predicted $\mathbf{y}^u$. Steps 3-9 of Algorithm 1 give the details.

The iterative part of the algorithm consists of an outer loop and an inner loop. In the outer loop (steps 10-15) the regularization parameter C^u is varied from a small value to the final value of 1 in annealing steps. This is done to avoid drastic switchings of the labels in $\mathbf{y}^u$, which helps the algorithm reach a better minimum of (3) and hence achieve better performance. For example, on ten runs of the multi-class dataset, *20NG* (see Table 1) with 100 labeled examples and 10,000 unlabeled examples, the average macro F values on test data achieved by supervised learning, Algorithm 1 without annealing and Algorithm 1 with annealing are, respectively, 0.4577, 0.5377 and 0.6253. Similar performance differences are seen on other datasets too.

The inner loop (steps 11-14) does alternating optimization of $\mathbf{w}$ and $\mathbf{y}^u$ for a given C^u. In steps 12 and 13 we use the most recent $\mathbf{w}$ and $\mathbf{y}^u$ as the starting points for the respective sub-optimization problems. Because of this, the overall algorithm remains very efficient in spite of the many annealing steps involving C^u. Typically, the overall cost of the algorithm is only about 3-5 times that of solving a supervised learning problem involving $(n+l)$ examples. For step 12 one can employ any standard algorithm suited to the chosen loss function. In the rest of the section we will focus on step 13.

Algorithm 1 *Semi-Supervised Learning Algorithm*

```
1:  Solve the supervised learning problem, (1) and get $\mathbf{w}$.
2:  Set initial labels for unlabeled examples, $\mathbf{y}^u$ using steps 3-9 below.
3:  Set $Y = \{y\}$, the set of all classes, $A_y = \emptyset \ \forall y$, and $I = \{1, \ldots, n\}$.
4:  repeat
5:    $S_i = \max_{y \in Y} \mathbf{w}^T \mathbf{f}(y; \mathbf{x}_i^u)$ and $y_i = \arg\max_{y \in Y} \mathbf{w}^T \mathbf{f}(y; \mathbf{x}_i^u) \ \forall i \in I$.
6:    Sort $I$ by decreasing order of $S_i$.
7:    By order allocate $i$ to $A_{y_i}$ while not exceeding sizes specified by $n(y_i)$.
8:    Remove all allocated $i$ from $I$ and remove all saturated $y$ (i.e., $|A_y| = n(y)$) from $Y$.
9:  until $Y = \emptyset$
10: for $C^u = \{10^{-4}, 3 \times 10^{-4}, 10^{-3}, 3 \times 10^{-3}, \ldots, 1\}$ (in that order) do
11:   repeat
12:     Solve (3) for $\mathbf{w}$ with $\mathbf{y}^u$ fixed (i.e., without constraints).
13:     Solve (3) for $\mathbf{y}^u$ with $\mathbf{w}$ fixed.
14:   until step 13 does not alter $\mathbf{y}^u$
15: end for
```

2.1 Linear programming formulation

Let us now consider optimizing $\mathbf{y}^u$ with fixed $\mathbf{w}$. Let us represent each y_i^u in a 1-of-m representation by defining boolean variables z_{iy} and requiring that, for each i, exactly one z_{iy} takes the value 1. This can be done by using the constraint $\sum_y z_{iy} = 1$ for all i. The label constraints become $\sum_i z_{iy} = n(y)$ for all y. Let $c_{iy} = \xi(\mathbf{w}, \mathbf{x}_i^u, y)$. With these definitions the optimization problem of step 13 becomes (irrespective of the type of loss function used) the integer linear programming problem,

$$\min \sum_{i,y} c_{iy} z_{iy} \ \text{ s.t. } \sum_y z_{iy} = 1 \ \forall i, \ \sum_i z_{iy} = n(y) \ \forall y, \ z_{iy} \in \{0,1\} \ \forall i, y \tag{5}$$

This is a special case of the well known Transportation problem (Hadley, 1963) in which the constraint matrix satisfies unimodularity conditions; hence, the solution of the integer linear programming problem (5) is same as the solution of the linear programming (LP) problem (i.e., with the integer constraints left out), i.e., the integer constraints hold automatically at LP optimality. Previous works (Joachims, 1999; Sindhwani and Keerthi, 2006) do not make this neat connection to linear programming. The constraints $\sum_y z_{iy} = 1 \ \ \forall i$ allow exactly n non-zero elements in $\{z_{iy}\}_{iy}$; thus there is degeneracy of order m, i.e., there are $(n+m)$ constraints but only n non-zero solution elements.

2.2 Transportation simplex method

The transportation simplex method (a.k.a., stepping stone method) (Hadley, 1963) is a standard and generally efficient way of solving LPs such as (5). However, it is not efficient enough for typical large scale learning situations in which n, the number of unlabeled examples is large and m, the number of classes, is small. Let us see why. Each iteration of this method starts with a basis set of $n+m-1$ basis elements. Then it computes reduced costs for all remaining elements. This step requires $O(nm)$ effort. If all reduced costs are non-negative then it implies that the current solution is optimal. If this condition does not hold, elements which have negative reduced costs are potential elements for entering the basis.[3] One non-basis element with a negative reduced cost (say, the element with the most negative reduced cost) is chosen. The algorithm now moves the solution to a new basis in which an element of the previous basis is replaced by the newly entering element. This operation corresponds to moving a chosen set of examples between classes in a loop so that the label constraints are not violated. The number of such iterations is observed to be $O(nm)$ and so, the algorithm requires $O(n^2m^2)$ time. Since n can be large in semi-supervised learning, the transportation simplex algorithm is not sufficiently efficient. The main cause of inefficiency is that the step (one basis element changed) is too small for the amount of work put in (computing all reduced costs)!

2.3 Switching algorithm

We now propose an efficient heuristic *switching algorithm* for solving (5) that is suited to the case where n is large but m is small. The main idea is to use only pairwise switching of labels between classes in order to improve the objective function. (Note that switching makes sure that the label constraints are not violated.) This algorithm is sub-optimal for $m \geq 3$, but still quite powerful because of two reasons: (a) the solution obtained by the algorithm is usually close to the true optimal solution; and (b) reaching optimality precisely is not crucial for the alternating optimization approach (steps 12 and 13 of Algorithm 1) to be effective.

Let us now give the details of the switching algorithm. Suppose, in the current solution, example i is in class y. Let us say we move this example to class $\bar{y}$. The change in objective function due to the move is given by $\delta c(i,y,\bar{y}) = c_{i\bar{y}} - c_{iy}$. Suppose we have another example $\bar{i}$ which is currently in class $\bar{y}$ and we switch i and $\bar{i}$, i.e., move i to class $\bar{y}$ and move $\bar{i}$ to class y. The resulting change in objective function is given by $\rho(i,y,\bar{i},\bar{y}) = \delta c(i,y,\bar{y}) + \delta c(\bar{i},\bar{y},y)$. The more negative $\rho(i,y,\bar{i},\bar{y})$ is, the better will be the objective function reduction due to the switching of i and $\bar{i}$. The algorithm looks greedily for finding as many good switches as possible

[3]Presence of negative reduced costs may not mean that the current solution is non-optimal. This is due to degeneracy. It is usually the case that, even when an optimal solution is reached, the transportation algorithm requires several end steps to move the basis elements around to reach an end state where positive reduced costs are seen.

Algorithm 2 *Switching Algorithm to solve* (5)

```
1: repeat
2:   for each class pair $(y, \bar{y})$ do
3:     Compute $\delta c(i, y, \bar{y})$ for all $i$ in class $y$ and sort the elements in increasing order of $\delta c$ values.
4:     Compute $\delta c(\bar{i}, \bar{y}, y)$ for all $\bar{i}$ in class $\bar{y}$ and sort the elements in increasing order of $\delta c$ values.
5:     Align these two lists (so that the best pair is at the top) to form a switch list of 5-tuples, $\{(i, y, \bar{i}, \bar{y}, \rho(i, y, \bar{i}, \bar{y})\}$.
6:     Remove any 5-tuple with $\rho(i, y, \bar{i}, \bar{y}) \geq 0$.
7:   end for
8:   Merge all the switch lists into one and sort the 5-tuples by increasing order of $\rho$ values.
9:   while switch list is non-empty do
10:    Pick the top 5-tuple from the switch list; let's say it is $(i, y, \bar{i}, \bar{y}, \rho(i, y, \bar{i}, \bar{y}))$. Move $i$ to class $\bar{y}$ and move $\bar{i}$ to class $y$.
11:    From the remaining switch list remove all 5-tuples involving either $i$ or $\bar{i}$.
12:  end while
13: until the merged switch list from step 8 is empty
```

at a time. Algorithm 2 gives the details. Steps 2-12 consist of one major greedy iteration and has cost $O(nm^2)$. Steps 2-7 consist of the background work needed to do the greedy switching of several pairs of examples in steps 9-12. Step 11 is included because, when i and $\bar{i}$ are switched, data related to any 5-tuple in the remaining switch list that involves either i or $\bar{i}$ is messed up. Removing such elements from the remaining switched list allows the algorithm to continue finding more pairs to apply switching without a need for repeating steps 2-7. It is this multiple switching idea that gives the needed efficiency lift over the transportation simplex algorithm.

The algorithm is convergent due to the following reasons: the algorithm only performs switchings which reduce the objective function; thus, once a pair of examples is switched, that pair will not be switched again; and, the number of possible switchings is finite. A typical run of Algorithm 2 requires about 3 loops through steps 2-12. Since this algorithm only allows pairwise switching of examples, it cannot assure that the class assignments resulting from it will be optimal for (5) if $m \geq 3$. However, in practice the objective function achieved by the algorithm is very close to the true optimal value; also, as pointed out earlier, reaching true optimality turns out to be not crucial for good performance of the semi-supervised algorithm.

We compared the speed performance of transportation simplex and switching algorithms on real-world datasets such as *Ohscal* and found that the switching algorithm is faster by two orders of magnitude. Note that if m is large then steps 2-7 of Algorithm 2 can become expensive. We have applied the switching algorithm to datasets that have $m \leq 105$, but haven't observed any inefficiency. If m happens to be much larger then steps 2-7 can be modified to work with a suitably chosen subset of class pairs instead of all possible pairs.

3 Experiments with large margin loss

In this section we give results of experiments on our method as applied to multi-class and hierarchical classification problems using the large margin loss function, (2). We used the loss, $L(y, y_i) = \delta(y, y_i)$. Eight multi-class datasets and two hierarchical classification datasets were

Table 1: Properties of datasets. N : number of examples, d : number of features, m : number of classes, Type: M=Multi-Class; H=Hierarchical, with D=Depth and I=# Internal Nodes

	20NG	*la1*	*webkb*	*ohscal*	*reut8*	*sector*	*mnist*	*usps*	*20NG*	*rcv-mcat*
N	19928	3204	8277	11162	8201	9619	70000	9298	19928	154706
d	62061	31472	3000	11465	10783	55197	779	256	62061	11429
m	20	6	7	10	8	105	10	10	20	7
Type	M	M	M	M	M	M	M	M	H	H
D/I									3/8	2/10

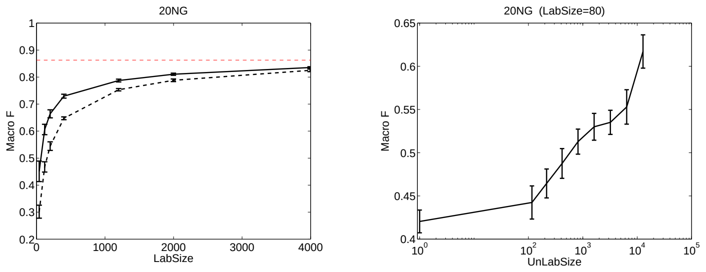

Figure 1: Hierarchical classification dataset - Variation of performance (Macro F): **Left** - as a function of the number of labeled examples (LabSize). Dashed black line corresponds to supervised learning; Continuous black line corresponds to the semi-supervised method; Dashed horizontal red line corresponds to the supervised classifier built using L and U with their labels known. **Right** - as a function of the number of unlabeled examples (UnLabSize), with the number of labeled examples fixed at 80.

used. Due to lack of space, performance results are given only for some datasets. The reader is referred to the longer version of this paper (Keerthi et al., 2012) for details on other data sets. Properties of these datasets (Lang, 1995; Forman, 2003; McCallum and Nigam, 1998; Lewis et al., 2006; LeCun, 2011; Tibshirani, 2011) are given in Table 1. Most of these datasets are standard text classification benchmarks. We include two image datasets, *mnist* and *usps* to point out that our methods are useful in other application domains too. *rcv-mcat* is a subset of rcv1 (Lewis et al., 2006) corresponding to the sub-tree belonging to the high level category MCAT with seven leaf nodes consisting of the categories, EQUITY, BOND, FOREX, COMMODITY, SOFT, METAL and ENERGY. In one run of each dataset, 50% of the examples were randomly chosen to form the unlabeled set, U; 20% of the examples were put aside in a set L to form labeled data; the remaining data formed the test set. Ten such runs were done to compute the mean and standard deviation of (test) performance. Performance was measured in terms of Macro F (mean of the F values associated with various classes).

In the first experiment, we fixed the number of labeled examples (to 80) and varied the number of unlabeled examples from small to big values. The variation of performance as a function of the number of unlabeled examples, for the multi-class dataset, *20NG*, is given in Figure 1 (Right). Performance steadily improves as more unlabeled data is added. Next we fixed the unlabeled data to U and varied the labeled data size from small values up to $|L|$. This is an important study for semi-supervised learning methods since their main value is when labeled

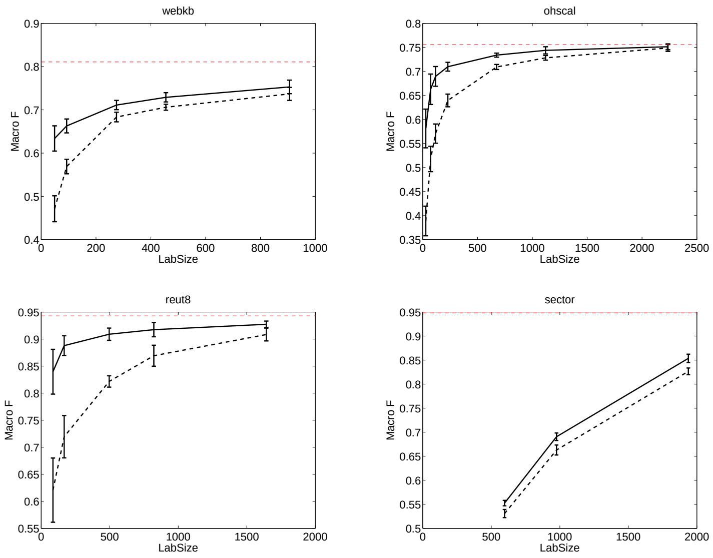

Figure 2: Multi-class datasets: Variation of performance (Macro F) as a function of the number of labeled examples (LabSize). Dashed black line corresponds to supervised learning; Continuous black line corresponds to the semi-supervised method; Dashed horizontal red line corresponds to the supervised classifier built using L and U with their labels known.

data is sparse (lower side of the learning curve). The variation of performance as a function of the number of labeled examples is shown in Figure 1 (Left). The same holds in other datasets too. The results for four multi-class datasets are given in Figure 2. Clearly, semi-supervised learning is very useful and yields good improvement over supervised learning especially when labeled data is sparse. The degree of improvement is sharp in some datasets (e.g., *reut8*) and mild in some datasets (e.g., *sector*). While the semi-supervised method is successful in linear classifier settings such as in text classification and natural language processing, we want to caution, like (Chapelle et al., 2008), that it may not work well on datasets originating from nonlinear manifold structure.

4 Conclusion

In this paper we extended the TSVM approach of semi-supervised binary classification to multi-class and hierarchical classification problems with general loss functions, and demonstrated the effectiveness of the extended approach. As a natural next step we are exploring the approach for structured output prediction. The $\mathbf{y}^u$ determination process is harder in this case since reduction to linear programming is not automatic. But good solutions are still possible. In many applications of structured output prediction, labeled data consists of examples with partial labels. This can be handled in our approach by including all unknown labels as a part of $\mathbf{y}^u$.

References

Bruzzone, L., Chi, M., and Marconcini, M. (2006). A novel transductive SVM for semisupervised classification of remote-sensing images. volume 44, pages 3363–3373.

Chang, M. W., Ratinov, L., and Roth, D. (2007). Guiding semi-supervision with constraint-driven learning. In *ACL*.

Chapelle, O., Chi, M., and Zien, A. (2006). A continuation method for semi-supervised SVMs. In *ICML*.

Chapelle, O., Sindhwani, V., and Keerthi, S. S. (2008). Optimization techniques for semi-supervised support vector machines. In *JMLR*, volume 9, pages 203–233.

De Bie, T. and Cristianini, N. (2004). Convex methods for transduction. In *NIPS*.

Forman, G. (2003). An extensive empirical study of feature selection metrics for text classification. In *JMLR*, volume 3, pages 1289–1305.

Ganchev, K., Graca, J., Gillenwater, J., and Taskar, B. (2009). Posterior regularization for structured latent variable models. Technical report, Dept. of Computer & Information Science, University of Pennsylvania.

Gärtner, T., Le, Q. V., Burton, S., Smola, A. J., and Vishwanathan, S. V. N. (2005). Large-scale multiclass transduction. In *NIPS*.

Graca, J., Ganchev, K., and Taskar, B. (2007). Expectation maximization and posterior constraints. In *NIPS*.

Hadley, G. (1963). *Linear Programming*. Addison-Wesley, 2nd edition.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *ICML*.

Keerthi, S. S., Sundararajan, S., and Shevade, S. (2012). Extension of TSVM to multi-class and hierarchical text classification problems with general losses. `http://arxiv.org/abs/1211.0210`.

Lang, K. (1995). Newsweeder: Learning to filter netnews. In *ICML*.

LeCun, Y. (2011). The MNIST database of handwritten digits.

Lewis, D., Yang, Y., Rose, T., and Li, F. (2006). Rcv1: A new benchmark collection for text categorization research. In *JMLR*, volume 5, pages 361–397.

Mann, G. S. and McCallum, A. (2010). Generalized expectation criteria for semi-supervised learning with weakly labeled data. In *JMLR*, volume 11, pages 955–984.

McCallum, A. and Nigam, K. (1998). A comparison of event models for naive Bayes text classification. In *AAAI Workshop on Learning for Text Categorization*.

Sindhwani, V. and Keerthi, S. (2006). Large-scale semi-supervised linear SVMs. In *SIGIR*.

Tibshirani, R. (2011). USPS handwritten digits dataset. `http://www-stat-class.stanford.edu/~tibs/ElemStatLearn/datasets/zip.info`.

Xu, L., Wilkinson, D., Southey, F., and Schuurmans, D. (2006). Discriminative unsupervised learning of structured predictors. In *ICML*.

Zien, A., Brefeld, U., and Scheffer, T. (2007). Transductive support vector machines for structured variables. In *ICML*.

Zubiaga, A., Fresno, V., and Martinez, R. (2009). Is unlabeled data suitable for multiclass SVM-based web page classification? In *NAACL HLT Workshop on Semi-supervised Learning for Natural Language Processing*.