Parallel Transductive Linear for Text Categorization

Miguel Fernando Cabrera
Universidad Nacional de Colombia - Sede Medellín
Facultad de Minas
Medellín, Colombia
mfcabrer@unal.edu.co

Jairo José Espinosa
Universidad Nacional de Colombia - Sede Medellín
Facultad de Minas
Medellín, Colombia
jespinov@unal.edu.co

ABSTRACT

Transductive Support Vector Machines (TSVM) have shown improvements on classification tasks such as Text Categorization (TC), where the features present co-occurrence and the training samples are few in comparison with the data available. Although the nature of the TSVM algorithm as described by Joachim makes it difficult to be as fast as a regular SVM, the parallelization of this algorithm is interesting when the gains in the classification performance are crucial. Solving TSVM algorithm requires the solution of multiple quadratic programming problems that generally scales to $O(n^3)$. We describe the implementation of a TSVM Solver for a 2-class problem using a parallel architecture which aims to improve the necessary computation times while preserving the classification performance. This strategy produced an implementation that is 2x to 7x faster than a regular TSVM for a TC problem of 2,000 vectors and more than 30,000 features.

Categories and Subject Descriptors

H.4 [**Intelligent Systems**]: Miscellaneous; D.2.8 [**Artificial Intelligence**]: Machine Learning—*text classification, parallel algorithms, information retrieval*

Keywords

SVM, Text Classification, Machine Learning, Algorithm, Parallel Algorithm, Information Retrieval, Artificial Intelligence

1. INTRODUCTION

Text classification is a key aspect of text filtering, document management and retrieval tasks. Besides basic document classification many problems can be seen as instances of a Text Categorization (TC) problem. Spam detection, Web search improvement and automated metadata generation are just a few examples of this [16]. Some of those tasks can be achieved by human beings, but a manual classification is at best expensive and practically impossible for large amounts of documents found today in modern information systems.

Support Vector Machines [17] are powerful tools for classifying large data sets, and due to the nature of the classical text representation models, it has been applied successfully in automated document classification tasks [8, 10]. An special type of SVM, based on Transductive inference (TSVM), has demonstrated to be more effective for document classification than the common inductive inference based SVM [10].

One characteristic of the SVM is that, in its formal definition, the computation and memory storage requirements increase rapidly with the number of training vectors. This is due the fact that the SVM classification problem is a Quadratic Programming problem (QP) that finds the support vectors in all training data set. Solvers of this kind of problem generally scales to $O(n^3)$ making it a very computationally expensive problem.

This paper describes an implementation of a parallel SVM with the cascade model described in [6] using a Transductive learning algorithm.

2. MODELS OF TEXT AND TEXT REPRESENTATION

Usually in a digital library database a set of words that defines the topics or the main characteristics of a document is manually defined. The main objective of this is to help the search of the physical document (e.g books, papers,etc.). This list of terms or keywords enables the user to query documents related to one or more concepts. One of the main problems of this is that only few keywords are associated with a defined document, and generally this assignation is made by hand, making it expensive and error prone.

During the last few decades many representation forms have been developed ranging from simple word appearance binary representation to a concept [4], probabilistic and even Neural Networks based ones [11] [15].

The most common approach to automatically extract the index terms from a corpus is called vector space model. In this model each document d is represented as a vector$(\theta_1, ..., \theta_M)$ where θ_j is a function of the frequency in d of the j^{th} word of a chosen dictionary M.

Based in this definition various forms of $\theta(j)$have been defined, however the most popular ones are the variations of [*tf-idf*] In which θ_j is equal to the formula:

$$\theta_j = tf_j(d)\,.\,log(\frac{N}{df_j}), \quad \forall j \in \{1, \ldots, M\} \tag{1}$$

The $tf_j(d)$ (term frequency) is the number of times that the j^{th} word appears in the document d, df_j is the number of documents the term appears in all of the corpus and N is the total number of documents in the corpus.

3. SUPPORT VECTOR MACHINES

Support Vector Machines (SVMs) [17] are powerful classification and regression tools that have been widely applied in the solutions of many problems, generally yielding comparable or even better performance than other algorithms.

The SVM algorithm is based on the concept of the VC Dimension[19] and on the principle of structural risk minimization developed by Vapnik [18, 19]. Roughly, the VC dimension is a metric of how complex a classifier machine is, complex classifier has more capacity to fit the training data, thus, overfitting is more likely to occur, therefore, a classifier with minimum VC Dimension is classifier.

The basic idea of empirical risk minimization principle is to find an hypothesis s from an hypothesis space S for which the lowest probability of error is guaranteed for a given set of training examples. For linear classification problems this is equivalent to find the

4. SVM AND TEXT CATEGORIZATION

The usage of SVM for TC was first introduced by Joachims [8, 10] and similar setups have been used in recent literature [5]. The TC task using SVM can bee seen geometrically as finding an hyperplane (decision surface) that separates two groups of points. Each point is a vector representation of a document based on the terms that the document contains. This can can be done using any of the models used in information retrieval, such as binary, term frequency (tf), or the popular term frecuency - inverse document frequency ($tf - idf$) among others [16, 14]. As argued by Joachims [8], SVMs offer two important advantages for TC:

- Doing term selection is generally not needed, because SVM are robust to overfitting problems and can scale up to high dimensions.
- No human and machine effort in parameter tuning on a validation set is needed, as there is a theoretical default choice of parameter settings.

These two characteristics makes the SVMs attractive for tackling this kind of problems.

5. TRANSDUCTIVE LEARNING FOR SVM AND TEXT CATEGORIZATION

The goal of the TSVM or Transductive learner is to select a function $hL = L(S_{train}, S_{test})$ from the hypothesis space H using S_{train} and S_{test} such that the expected number of erroneous predictions on the test and the training samples are minimized.

The setting of transductive SVM for TC was introduced by Joachims [10] based on the Work of Vapnik [17]. It consists on a set of training examples: $\{(\overrightarrow{x_1}, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, and each of them consists of a document vector $\overrightarrow{x}$ and a binary label $y \in \{-1, 1\}$. The main difference with the inductive setting is that the learning is also given a set of i.i.d samples of j test examples $\overrightarrow{x*}_{n+1}, ..., \overrightarrow{x*}_j$ from the same distribution. The transductive learning function L aims to find the function h_L that minimize

$$R(L) = \int \frac{1}{k} \sum_{i=1}^{k} \Theta(h_L(\overrightarrow{x_i^*}), y_i^*) dP(\overrightarrow{x_i^*}, y_1)...dP(\overrightarrow{x_k^*}, y_k)$$

$\Theta(a, b)$ is 0 when $a = b$, 1 otherwise. Vapnik [17] gives the bounds on the training error:

$$R_{train} = \frac{1}{n} \sum_{i=1}^{n} \Theta(h(\overrightarrow{x_i}), y_i) \tag{3}$$

and the test error:

$$R_{test} = \frac{1}{k} \sum_{i=1}^{k} \Theta(h(\overrightarrow{x_i^*}), y_i^{true}) \tag{4}$$

This leads to the following optimization problem (here with slack variables for handling non-separable cases)

$$\begin{aligned}
&Minimize\,over\,(y_1^*, ..., y_n^*, \overrightarrow{\omega}, b, \xi_1, ..., \xi_n, \xi_1^*, ..., \xi_k^*) : \\
&\qquad \frac{1}{2}\|\overrightarrow{\omega}\|^2 + C\sum_{i=0}^{n} \xi_i \\
&subject\,to \qquad \forall_{i=1}^{n} : y_i[\overrightarrow{\omega} \cdot \overrightarrow{x_i} + b] \geq 1 - \xi_i \\
&\qquad \forall_{j=1}^{k} : y_j^*[\overrightarrow{\omega} \cdot \overrightarrow{x_j^*} + b] \geq 1 - \xi_j^* \\
&\qquad \forall_{i=1}^{n} : \xi_i > 0 \\
&\qquad \forall_{j=1}^{k} : \xi_i^* > 0
\end{aligned}$$

The algorithm shown on figure 1 solves the minimization problem defined above. For a complete description and an in-depth study of the Joachims' approach see the seminal paper [10] and the work of Collobert [3].

6. PARALLEL METHODS FOR TRAINING TRANSDUCTIVE SVM

Solving the Transductive SVM using the algorithm described by Joachims implies to solve many times an SVM inductive optimization problem. The algorithm improves the objective function by switching the labels interactively of two unlabeled data points x_i and x_j with $\xi_i + \xi_j > 2$. It uses two nested loops to optimize a TSVM which solves a quadratic programming every time it changes a label from an unlabeled data. The convergence of the nested loop relies on the fact that there is only a finite number 2^U ways of labeling U unlabeled points and it is unlikely that all of them are examined [3]. However because the heuristic only swaps the label of two unlabeled examples at each nested loop iteration it might need (and in fact it does) many iterations to reach a minimum which makes it intractable for big data sets in practice.

TSVM in the worst case have complexity of $O(L+U)^3$ with U labeled points, although it generally scales to square in most practical cases [9] thus still intractable for large data sets.

A recent approach called the Cascade SVM, consists of an array of SVM solvers arranged in pyramid form. This structure allows an easy parallelization and does not modify the original formulation at all [6, 7]. This is the approach selected for the parallelization of TSVM because there is no need to modify the original quadratic programming problem and it is straightforward to implement.

7. THE CASCADE SVM

The Cascade SVM was described by Graf et al [6] and further explored by Zhang et al [7] and it is based on the strategy of early elimination of non-support vectors from the training set [9]. The Cascade SVM is a distributed architecture where smaller optimization problems are solved independently making them easily parallelizable. The results of these smaller problems are then assembled in a way that ensures that it eventually converges to globally optimal solution.

In order to find a minimum using this approach the problem is initialized with a number of independent smaller optimization whose results are combined in later stages in a hierarchical fashion

The data is split into subsets and each one of them is evaluated to find support vectors in the first layer, then the results are combined two-by-two and entered as a training set for the next layer.

7.1 Cascade Transductive SVM

The original formulation for this cascade SVM is defined for the classical SVM problem so it is necessary to adapt the architecture described above in order to handle properly the TSVM formulation. The approach we chose was to parallelize the more computational expensive part of the TSVM algorithm: the *solve_svm_qp* function which solves the dual problem associated with formulation described in section 5. Now for each iteration of the loop the a SVM inductive problem is solved in a parallel manner. We aim for two goals with this, in the first place, we need to implement an algorithm that is at least is as good as the serial (non parallel) version and also we need to ensure the velocity of the process through the use of the parallel architecture.

8. MERGING DATA SETS FOR THE TRANSDUCTIVE CASE

In this section, we describe the procedure to merge two subsets using as example the description in the figure shown in figure 2. For Transductive SVM there is not any major change besides the fact that creating the problem for SVM we have to take into account two boundaries for the values of α_i instead of one: C for $\{a_1 ... \alpha_L\}$and $C*$ for $\{a_{L+1} ... \alpha_{L+U}\}$. Also, the process of elimination of early non-support vector machine is made on both groups, the labeled and the unlabeled data.

The starting point and gradient for the merged set are defined below:

$$W = \frac{1}{2} \begin{bmatrix} \vec{\alpha_1} \\ \vec{\alpha_2} \end{bmatrix}^T \begin{bmatrix} Q_1 & Q_{12} \\ Q_{21} & Q_2 \end{bmatrix} \begin{bmatrix} \vec{\alpha_1} \\ \vec{\alpha_2} \end{bmatrix} + \begin{bmatrix} \vec{e_1} \\ \vec{e_2} \end{bmatrix}^T \begin{bmatrix} \vec{\alpha_1} \\ \vec{\alpha_2} \end{bmatrix} \tag{5}$$

$$\vec{G_3} = \begin{bmatrix} \vec{\alpha_1} \\ \vec{\alpha_2} \end{bmatrix}^T \begin{bmatrix} Q_1 & Q_{12} \\ Q_{21} & Q_2 \end{bmatrix} + \begin{bmatrix} \vec{e_1} \\ \vec{e_2} \end{bmatrix}$$

With $Q_{12} = Q_1 Q_2$ and $Q_{21} = Q_2 Q_1$.

9. EXPERIMENTS AND RESULTS

In this section we present the experimental results for the text classification task. The base algorithm and auxiliary routines were programmed in Matlab 7.4[1] using the Matlab *quadprog* function which is a convex QP solver. In order to fully parallelize the Cascade Architecture we made use of the Distributed Computer Tool Box which enabled us to run the algorithm in all the cores available. For the experiments we used a Desktop computer with Processor Intel® Pentium® Dual Core CPU 3.40GHz with 1.5 GB of RAM.

For the experiments we use a subset of the 20 Newsgroups [12] data set found in the UCI ML Repository[1]. We used the classic tf-idf weight to create the document vectors [15, 14] and then it was preprocessed using English stop-words and the Porter stemmer algorithm [13]. The code was written in Ruby using an already existing implementation of the stemmer algorithm.

The 20 Newsgroup data sets is a collection of approximately 20,000 newsgroup documents, partitioned evenly across 20 different newsgroups. The collection is organized in categories ranging from computer related topics
(e.g comp.sys.ibm.pc.hardware, comp.windows.x, etc.) to politics (talk.politics.misc, talk.politics.guns,etc).

The data set is divided by newsgroup topics and can be grouped into similar categories. For the experiments we define two tasks, an "easy" one and a "hard" one. For the easy task we took 1000 messages from *alt.atheism* and 1000 messages from *comp.graphics.* We call this an easy task because the terms that appear in the documents from each collection are different, so we can expect a full separable problem. For the hard task we chose related news groups where it is expected that the most important terms are similar for both collection. We chose c*omp.sys.ibm.pc.hardware*

[1]http://www.mathworks.com/

	#L	#U	P	R	$F1$	Time
SVM	200	N/A	0.9384	1	0.9682	0.0135
TSVM	200	400	0.9823	0.9823	0.9823	44.4103
TSVM-P	200	400	0.9823	0.9823	0.9823	**15.3406**
	#L	#U	P	R	$F1$	Time
SVM	100	N/A	0.9062	1	0.9508	0.0135
TSVM	100	400	0.9823	0.9823	0.9823	32.1135
TSVM-P	100	400	0.9823	0.9823	0.9823	**11.0953**
	#L	#U	P	R	$F1$	Time
SVM	100	N/A	0.9975	0.8957	0.9438	0.0135
TSVM	100	800	0.9798	0.9798	0.9798	214.2297
TSVM-P	100	800	0.9798	0.9798	0.9798	**68.4745**
	#L	#U	P	R	$F1$	Time
SVM	100	0	0.8966	0.9975	0.9444	0.0135
TSVM	100	1600	0.9800	0.9800	0.9800	278.3236
TSVM-P	100	1600	0.9800	0.9800	0.9800	**41.5395**

Table 1: The "Easy" Task: *comp.graphics* and *alt.atheism*. Using C=10

	#L	#U	P	R	$F1$	Time
SVM	100	N/A	0.7400	0.8916	0.8087	0.0025
TSVM	100	200	0.8600	0.8600	0.8600	1.5370
TSVM-P	100	200	0.8700	0.8700	0.8700	**1.4863**
	#L	#U	P	R	$F1$	Time
SVM	100	N/A	0.8830	0.7550	0.8140	0.0025
TSVM	100	400	0.8550	0.8550	0.8550	6.0628
TSVM-P	100	400	0.8550	0.8550	0.8550	**4.3323**
	#L	#U	P	R	$F1$	Time
SVM	100	N/A	0.7350	0.8724	0.7978	0.0025
TSVM	100	800	0.8200	0.8200	0.8200	31.0015
TSVM-P	100	800	0.8200	0.8200	0.8200	**19.5472**
	#L	#U	P	R	$F1$	Time
SVM	100	0	0.7362	0.8624	0.7943	0.0135
TSVM	100	1600	0.8187	0.8187	0.8187	221.5013
TSVM-P	100	1600	0.8187	0.8187	0.8187	**86.1924**

Table 2: The "Hard" Task: of *comp.sys.ibm.pc.hardware* and *comp.sys.mac.hardware*. Using C=9

and *comp.sys.mac.hardware* for evaluating the SVM effectiveness and computation times with also 1000 messages from each of them. The data was split in a training set (20%) and testing or unlabeled set (80%).

As performance measure we chose $F1$ as measure of effectiveness, we also show the values of precision and recall for the experiments. The kernel choice for the SVM was the linear kernel which has been reported to be effective for text categorization tasks [10, 5], thanks in part to the high dimensional nature of the input vectors. The C parameter was estimated for the two sets using 5-fold cross validation with the training set. The other parameter defined by Joachim's algorithm is *num+*, the number of test inputs to be labeled initially as positive, in our implementation this number is also half of the total test set, thus ensuring the precision and recall break-even. The choice of the parameter $C*$ is done using many heuristics, however we use the one proposed by Joachims which consists in making it equal to C.

The Cascade architecture used in the experiments was formed with 3 layers of SVM with 4 SVMs in the first, 2 in the second and finally 1 in the last layer.

Besides the categorization performance, we are interested in the computation time, we expect that with the appropriate formulation of the parallel architecture, reduce the impact of the computation time of the TSVM Algorithm.

9.1 Easy Task

As mentioned above, the easy task is an experiment where a linear separable problem is expected formed by messages from newsgroups with very disjoint topics. The number of features for each vector was 30854, a high dimensional problem the with the parameter $C = 10$ from the 5-fold cross validation. Table 1 shows the results of the experiments.

We can notice how the classical SVM algorithm is always faster that the Transductive implementation and for this specific problem, which is expected to be almost linear separable, with few unlabeled data the classification performance is very similar, while the difference in speed of the algorithm is really huge. However when the training data decreases and there is more unlabeled data the differences in classification performance begin to be more noticeable, even with this "easy" problem. Regarding the speed, it is obvious by the results that that the standard SVC algorithm is by far faster than its transductive counterparts, therefore we are going to discuss the differences between the transductive and its parallel version. For this problem, the classification performance of the transductive algorithms were the same, thus one of the objectives was accomplished, to maintain the same classification performance while performing the tasks in parallel. Of course, the gains of the parallel approach and the early eliminations of non-support vectors are devised in the time used for the calculations on the Transductive SVM problem.

9.2 Hard Task

This task is created with 2000 messages from the categories of *comp.sys.ibm.pc.hardware* and *comp.sys.mac.hardware* which have many equal terms thus making it a possible not linearly separable problem. The number of features extracted for this problem is 26997. The parameter choice for $C =$ 9 and was taken from the same 5-fold cross-validation with the available training data.

As expected, the TSVM performed better than the classic linear SVM, however due to the nature of the data, the classification performance difference between the two the TSVM and the SVM was not really substantial. Another thing to notice is that this example is somewhat counter-intuitive, because you could expect that for more training data available the difference between the classification performance between the classical and the transductive SVM would increase. However the opposite happened, when more training data was available, the difference between their performance decreased. Another result of the experiment, that surprised us, but we thought it could happen, was the for the 100-200 setup the transductive parallel SVM had a little bit more classification performance than the non-parallel version. Even with this particularities in general terms the expected occurred, the classification performance of the transductive setting was higher than the inductive one, and the parallel version of the TSVM run almost 3 times faster than the non transductive one.

10. CONCLUSIONS AND FUTURE DIRECTIONS

In this work we have implemented a parallel version of TSVM as described by Joachims [10], an approach that because its annealing heuristics requires many iterations solving similar QP problems. Our approach to speed up the process was to parallelize the SVM solver function using a Cascade architecture. We have shown that this approach speed up the process 2 to 7 times using just one dual core processor. As this architecture scales very well is expected that, with more processors availabl,e the the speed gains will be even more noticeable. In addition, more layers in the basic architecture could help in the improvement of the computation time.

From the data tables one could argue that the classification performance gains from using TSVM is not worth the computation time spent even with this parallel approach. However this last statement is in someway subjective, as the trade-off between speed and classification performance depends on the nature of the problem, it would be possible that we would prefer more classification performance at cost of computation time. Also, if more and more processors/cores available (as it is the actual trend) the speed gains could be much more noticeable.

Regarding the classification performance of the TSVM algorithm we noticed something particular, the best C for the inductive setting and the $C*$ for the transductive setting does not generally match in the sense that the best C parameter does not produce the best classification performance in the transductive case for a given set of data. However, we have chosen the parameter that works best for the inductive setting.

It is true that more optimization strategies have to be developed in order to make this algorithm really usable in a real world situation. Now, with a parallel and scalable version of the transductive algorithm a series of strategies could be explored in order to further optimize the computation time, however the implementation of such strategies is outside of the scope of this work and they are mentioned here as a possible future work.

Maybe the most obvious approach is that given that the data is split in equal parts (subsets), and per each iteration only one of the parts is modified (i.e. the labels of two of the entries) there is no need to recalculate the other parts of the n and they can be reused. For example in our particular case, we calculated 7 distinct SVM problems (as our architecture o 3 layers t in 4-2-1 e respectively), using caching we wold only need to calculate all of them once at the r and then only when a pair of of vector fell of different SVM problem. For most of the iterations we would only need to calculate 3 problems.

Another strategy would be to try new improved heuristics as described by Collobert et al [2] and modifying in Joachims' TSVM by reducing iteration in the cycles or interchanging more labels per iteration. This last is important because the annealing heuristic used by Joachims' TSVM is a source of the elevated computation time.

A good thing about this approach is that any improvement to the SVM formulation can be directly applied to the SVM, for example the technique of Zanghirati et al. [20]could be applied to each one of the SVM solvers in order to have multilevel parallelization, further improving the computation performance.

11. REFERENCES

[1] D. N. A. Asuncion. UCI machine learning repository, 2007. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[2] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of svms for very large scale problems. In *Advances in Neural Information Processing Systems*. MIT Press, 2002.

[3] R. Collobert, F. Sinz, J. Weston, and L. Bottou. Large scale transductive svms. *J. Mach. Learn. Res.*, 7:1687–1712, 2006.

[4] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[5] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *CIKM '98: Proceedings of the seventh international conference on Information and knowledge management*, pages 148–155, New York, NY, USA, 1998. ACM Press.

[6] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade SVM. In *NIPS*, 2004.

[7] J. Y. Jian-Pei Zhang, Zhong-Wei Li. A parallel svm training algorithm on large-scale classification problems. In *Machine Learning and Cybernetics, 2005*, pages 1637 – 1641, 2005.

[8] T. Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *ECML '98: Proceedings of the 10th European Conference on Machine Learning*, pages 137–142, London, UK, 1998. Springer-Verlag.

[9] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

[10] T. Joachims. Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning (ICML)*, pages 200–209, Bled, Slowenien, 1999.

[11] M. Keller and S. Bengio. Theme topic mixture model: A graphical model for document representation.

[12] K. Lang. 20 newsgroups data set.

[13] M. F. Porter. An algorithm for suffix stripping. pages 313–316, 1997.

[14] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. Technical report, Ithaca, NY, USA, 1987.

[15] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

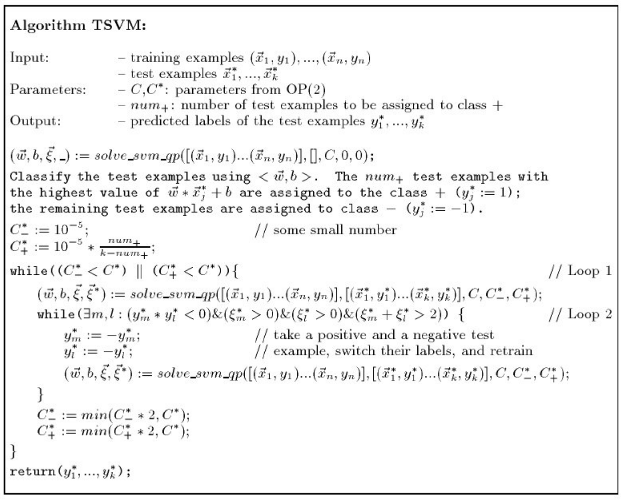
[16] F. Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.

[17] V. N. Vapnik. *Statistical Learning Theory.* John_Wiley, Sept. 1998.

[18] V. N. Vapnik. *The Nature of Statistical Learning Theory (Stat's for Eng. and Inf. Sci.).* Springer Verlag, 1999.

[19] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[20] G. Zanghirati and L. Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel Comput.*, 29(4):535–551, 2003.

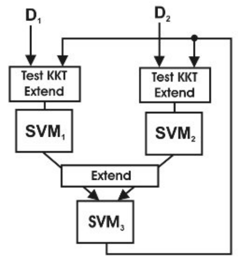


Algorithm TSVM:

```
Input:        – training examples $(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)$
              – test examples $\vec{x}^*_1, ..., \vec{x}^*_k$
Parameters:   – $C$,$C^*$: parameters from OP(2)
              – $num_+$: number of test examples to be assigned to class +
Output:       – predicted labels of the test examples $y^*_1, ..., y^*_k$

$(\vec{w}, b, \vec{\xi}, \_) := solve\_svm\_qp([(\vec{x}_1, y_1)...(\vec{x}_n, y_n)], [], C, 0, 0)$;
Classify the test examples using $< \vec{w}, b >$.  The $num_+$ test examples with
the highest value of $\vec{w} * \vec{x}^*_j + b$ are assigned to the class + $(y^*_j := 1)$;
the remaining test examples are assigned to class − $(y^*_j := -1)$.
$C^*_- := 10^{-5}$;                          // some small number
$C^*_+ := 10^{-5} * \frac{num_+}{k-num_+}$;
while(($C^*_- < C^*$) || ($C^*_+ < C^*$)){                                  // Loop 1
    $(\vec{w}, b, \vec{\xi}, \vec{\xi}^*) := solve\_svm\_qp([(\vec{x}_1, y_1)...(\vec{x}_n, y_n)], [(\vec{x}^*_1, y^*_1)...(\vec{x}^*_k, y^*_k)], C, C^*_-, C^*_+)$;
    while($\exists m, l : (y^*_m * y^*_l < 0) \& (\xi^*_m > 0) \& (\xi^*_l > 0) \& (\xi^*_m + \xi^*_l > 2)$) {   // Loop 2
        $y^*_m := -y^*_m$;                   // take a positive and a negative test
        $y^*_l := -y^*_l$;                   // example, switch their labels, and retrain
        $(\vec{w}, b, \vec{\xi}, \vec{\xi}^*) := solve\_svm\_qp([(\vec{x}_1, y_1)...(\vec{x}_n, y_n)], [(\vec{x}^*_1, y^*_1)...(\vec{x}^*_k, y^*_k)], C, C^*_-, C^*_+)$;
    }
    $C^*_- := min(C^*_- * 2, C^*)$;
    $C^*_+ := min(C^*_+ * 2, C^*)$;
}
return($y^*_1, ..., y^*_k$);
```

Figure 1: Algorithm for training Transductive Support Vector Machines [10]



$$W_i = -\frac{1}{2}\vec{\alpha}_i^T Q_i \vec{\alpha}_i + \vec{e}_i^{\,T} \vec{\alpha}_i;$$
$$\vec{G}_i = -\vec{\alpha}_i^T Q_i + \vec{e}_i;$$

Figure 2: A Cascade with two input sets D_1,D_2. W_i, G_i and Q_i are objective function, gradient, and kernel matrix, respectively, of SVM$_i$ (in vector notation); Gradients of SVM$_1$ and SVM$_2$ are merged as indicated in 5 and are entered into SVM$_3$. [6]