Paralell Training of Linear Transductive SVMs for Automated Text Categorization

Miguel Fernando Cabrera Granados
mfcabrer@unal.edu.co

Advisor
Jairo José Espinosa

Presented in Partial Fulfillment
of the Requirements
for the Degree of
Ingeniero de Sistemas e Informática

Escuela de Sistemas - Facultad de Minas
Universidad Nacional de Colombia - Sede Medellín

July 8, 2011

Resumen

Maquinas de Soporte Vectoriales Transductivos (TSVM) han mostrado mejoras en tareas de clasificación donde las características presentan co-ocurrencia y los ejemplos de entrenamiento son pocos en comparación con los datos disponible como es común en problemas de categorización de textos (TC). Aunque la naturaleza del algoritmo de TSVM descrito por Joachims hace difícil que sea tan veloz como una SVM regular, la paralelización de este algoritmo es interesante cuando las ganancias en la efectividad de clasificación es crucial. Resolver el algoritmo de TSVM requiere la solución de múltiples problemas de programación cuadrática que generalmente escalan a $O(n^3)$. Aquí describimos una implementación de un TSVM para problemas de clasificación binaria usando una arquitectura paralela que busca mejorar los tiempos de computo necesarios mientras preserva la efectividad de la clasificación. Para lograr este objetivo adaptamos la arquitectura en cascada desarrollada por Graf et al. para SVM regulares. La unión de estas estrategias produjo una implementación que es de 2 a 7 veces mas rápida que una TSVM regular para un problema de TC de 2.000 vectores y mas de 30.000 características.

Palabras Clave: Maquinas de Soporte Vectorial,Aprendizaje de Máquina, IA, Clasificación, Recuperación de Información, Paralelización.

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 is difficult to be as fast as a regular SVM, the parallelization of this algorithm is interesting when the gains in 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 2-class problem using a parallel architecture which aims to improve the necessary computation times while preserving the classification performance. For this purpose we adapt the parallel cascade architecture for regular SVM described by Graf et al.. This strategy produced an implementation that is 2x - 7x faster than a regular TSVM for a TC problem of 2,000 vectors and more than 30,000 features.

Keywords: SVM, Classification, Machine Learning, Algorithm, Parallel, Information Retrieval.

Contents

1 Overview **4**
- 1.1 Introduction . . . 4
- 1.2 Machine Learning . . . 5
- 1.3 Information Retrieval . . . 6
  - 1.3.1 Models of Text and Text Representation . . . 7
  - 1.3.2 Feature Selection and Dimensionality Reduction . . . 8
- 1.4 Text Categorization . . . 8
  - 1.4.1 Performance Measures . . . 9
  - 1.4.2 Applications of Automated Text Categorization . . . 10
  - 1.4.3 Machine Learning Text Categorization Techniques . . . 10
- 1.5 Parallel Machine Learning . . . 11

2 Text Categorization With SVM **13**
- 2.1 Support Vector Machines . . . 13
  - 2.1.1 Finding the Optimal Hyperplane . . . 13
- 2.2 SVM and Text Categorization . . . 17
- 2.3 Transductive Learning for SVM . . . 17

3 Parallel Methods for Training Transductive SVM **19**
- 3.1 Introduction . . . 19
- 3.2 The Cascade SVM . . . 20
  - 3.2.1 Cascade Transductive SVM . . . 20
  - 3.2.2 Merging Data Sets for the Transductive Case . . . 21

4 Experiments and Results **23**
- 4.1 Experiments . . . 23
  - 4.1.1 Toy Experiment . . . 24
  - 4.1.2 Easy Task . . . 25
  - 4.1.3 Hard Task . . . 25
- 4.2 Conclusion And Future Direction . . . 28

Chapter 1

Overview

1.1 Introduction

Text classification is a key aspect of text filtering, document management and retrieval tasks. Besides basic document classification for some kind of digital library, many problems can be seen as instances of the Text Categorization (TC) problem . Spam detection, Web search improvement and automated metadata generation are just a few examples of this [34]. 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.

A TC technique uses example documents that have been previously categorized in classes by an authority in order to learn a model that, with an associated error value, can automatically predict the class that the authority would have given to future documents.

Support Vector Machines [38] are powerful tools for classifying large data sets, and due the nature of the classical text representation models, it has been applied successfully in automated document classification tasks [17, 19]. 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 [19].

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.

One approach to cope with this limitation is to divide the problem into chunks [18, 30] and train those sub-problems. Even with these optimizations, the problem still has large computational requirements, and the required time to train grows sufficiently enough for making it not useful for real-time training when using large data sets. The implementation of Transductive inference for a SVM requires to solve the same problem many times over generally large data sets, until finding the optimal classifier. This makes the scaling problem for Transductive SVM even more difficult, therefore, methods for optimizing the training process need to be developed.

Taking advantage actual trends in processor technologies, where the multi-core processor is becoming the norm [28, 11] parallel implementation of such algorithm along with other optimization will help make the application of this technique practical in real world situations. Also, as an extra motivation, empirical results have showed that some parallel settings for SVM have achieved better generalization

than their classic counterparts for some problems [4].

This work describes an implementation of a parallel SVM using the cascade model described in [12] using a Transductive learning algorithm. The first part, Overview, introduces the basic concepts a Machine Learning, Information Retrieval and Text Categorization. Later we describe the SVM algorithm and justify the reason of using SVM for text classification over other techniques. In chapter 3 we exhibit our experimental set-up and the show the analysis of the results.

1.2 Machine Learning

Machine Learning (ML) is concerned with the question of how to construct computer programs that automatically improve with experience [29]. The experience encountered by the computer program are examples that it takes as input. In order to to develop such programs ML borrows concepts from many areas such Statistics, Information Theory, Biology and Control Theory, furthermore, given the amount of statistical based ML algorithms developed in the last years, many scientists see ML as a crossroad between Statistics and Computer Science.

The learning problem is defined formally in [29] as follows:

Learning: A computer program is said to *learn* from experience E with respect to some class of Task T and performance measurer P, if its performance at tasks in T, as measured by P improves with E.

In other words, the program learns from an experience commonly represented by a group of data belonging to a universe on which the result of Task T is known. Using this information the computer program tries to generalize for over the available data, thus learning a representation of the universe.

The way in which an actual computer program realizes the task of learning from the experience can different, there are several modes in which the learning task can be accomplished [14]:

- *Supervised Learning*: In which the program generates an internal representation that maps inputs, to desired outputs. Classification problems can be seen as an instance of this type of learning, where the desired output is a binary value stating whether the input belongs to a specific category.
- *Unsupervised Learning:* Combines both labeled and unlabeled examples to generate the representation.
- *Reinforcement Learning*: In which the algorithm learns a policy of how to act given an observation of the world.

Besides these three main ways of learning, there are others that can be seen as variations of the ones mentioned above.

- *Semi-supervised Learning*: Combines both labeled and unlabeled examples to generate the representation.
- *Transductive Learning*: Similar to Semi-supervised Learning, but does not explicitly construct a function: instead, tries to predict new outputs based on training inputs, training outputs and test input which are available while training.

In this thesis we are going to use a restricted definition in order to illustrate the concepts described above. In these pages machine learning will refer to a generalized regression characterizing a set of label events $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ with a function $\Phi : X \rightarrow Y$ from event to label . Many researcher has used this setting with success [2] . The reader will notice the similarity between this definition and the definition of text categorization described in section 1.4 on page 8.

The question of how good the function Φ can generalize has spawned an entire subfield of machine learning called computational learning theory. Many of the techniques classified in this field have come from development of frameworks in this particular subfield. These frameworks generally don't state how good is going to perform an algorithm, instead create a probabilistic bound to the performance.

Some of techniques generally classified under ML are Artificial Neural Networks, Genetics Algorithm, k-Nearest Neighbor, Bayesian Networks and and Support Vector Machines (SVM) and has been successfully applied to.

1.3 Information Retrieval

Information Retrieval (IR) is branch of computer science whose original objective was to obtain information from generally large amounts of data. Today modern IR deals with content-based document management task, including document filtering, translation, summarization, question answering among others.

IR Systems makes large amounts of text accessible to people that need the information. A user the approaches to the system with a vague idea of what is looking for, so the goal of the system is to provide the information required. The user provides generally provides some information about what kind of documents is looking for, in the form of keywords.

And information retrieval system comprises several modules, Figure 1.1 shows the basic scheme of work of a basic IR system.

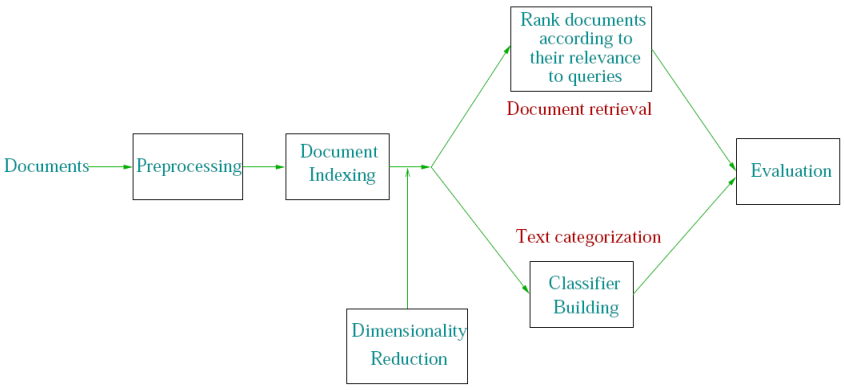


Figure 1.1: Parts of an Information Retrieval System

In today's information society where digital text is becoming more and more available, boosted by the usage of the Internet and more and more fast networks and larger capacities of storage, the need of better information retrieval techniques becomes everyday more strong.

1.3.1 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 at hand, making it expensive and error prone. Generally this querying these systems was limited to the numbers of keywords assigned and some metadata like title, author and dates of publication, being the user unable to query the whole document, making the retrieval task significantly harder.

Nowadays in digital libraries and similar systems where the all the text is available in digital form, automatic keyword extraction makes sense because these keywords are tightly associated with the content. This keywords are called indexes and the task of associating each document with a set of keywords is called *indexing*.

Indexing is based is predefined group of keywords: the dictionary or vocabulary. As mentioned above, in a digitalized text database (also called a corpus) the dictionary is usually extracted from the set of words present in the corpus, and the expensive manual indexing can be avoided by representing the documents by the dictionary terms they are made of. Given that all the full text is available for querying it, now problem is to find the best way of representing the text in order to extract the more important words, that is, the keywords that define it.

During the last few decades many representation forms have been developed ranging from simple word appearance binary representation to a concept [8], probabilistic [22] and even Neural Networks [23] based ones.

The most common approach to automatically extract the index terms from a corpus is called vector space model [33]. In this model each document d is represents 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:

Binary In which θ_j is equal to 1 if the j^{th} word appears in the document or 0 otherwise.

Frequency In which θ_j is equal to the number of times j^{th} word appears in the document.

tf-idf In which θ_j is equal to the *tf-idf* formula:

$$\theta_j = tf_j(d) \cdot log(\frac{N}{df_j}), \quad \forall j \in \{1, \ldots, M\} \tag{1.1}$$

The equation 1.1 was defined proposed by Salton and Buckley [32] and it is the more complex of the three. 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 the corpus and N is the total number of documents in the corpus. The main reason behind this formulation is that in the context of document retrieval it is considered that words appearing too frequently across the corpus may not be discriminant, therefore this formula gives more importance to terms appearing more frequently in the documents while penalizing the ones that appears in to many documents. The reader may have noticed that this representation of text does not take into account the actual order of the words inside the document, despite this, vector space model have performed well and it still used with some variations in the majority of IR systems.

the vector space model representation described above may seem simplistic in comparison with the more complex representation mentioned first, nevertheless, it seems that for some tasks like text

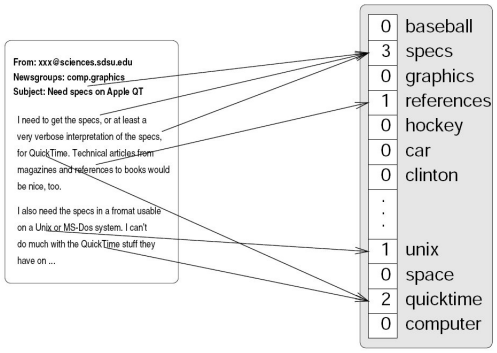


Figure 1.2: Representing text as a feature vector [17]

categorization, the classical tf-idf works just well and more complex representations of the text do not significantly affect the performance of the methods.

1.3.2 Feature Selection and Dimensionality Reduction

Due to the fact that the representation of the text can have impact in any retrieval task, any strategy to enhance the representation of text should be studied. With the years, many techniques have been developed in the field of Information Retrieval in order to get a better performance on the representation of the text made. IR research shows us that the word stems work well as representation units [17]. The word stemming process (sometimes called term conflation) consists in removing the case and flection information from a word [31]. For example words as "engine", "engineer", "engineering" becomes the same term "engin". Generally this techniques are applied to the document before the transforming it into a term vector representation, thus making the resulting document vector notably smaller.

Another technique used to avoid unnecessary large feature vectors is removed the so-called stop-words, or the words that for their common use is known that are not significant for the text. Words like "and", "or", "of", etc are removed from the original text.

There are other approaches for selecting the most important features other than the described above, these approaches try to select the most important features (the "best" words) from the dictionary M, creating a subset $M0$, that used for document representation provides better results. However, Joachims [17] shows that the "worst" features still contain important information, therefore suggest that all features should be selected, when it is possible.

1.4 Text Categorization

The goal of text categorization (TC) is to automatically assign, for each documents, categories selected among a predefined set. In opposition to the machine learning typical multi-class classification task which aims at attributing one class among the possible ones to each example, documents in a text categorization task may belong to several categories. Sebastiani [34] formally defines TC as task of assigning a Boolean to each pair $\langle d_j, c_j \rangle \in \mathcal{D}\ x\ \mathcal{C}$ where $\mathcal{D}$ is a domain of documents and $\mathcal{C} = \{c_1, \ldots, c_{|c|}\}$ is a set of predefined categories. a true value assigned to the pair $\langle d_j, c_j \rangle$ indicates the

decision of filling d_j under category c_j, and a false value the decision of not filling it under that category. More formally the task is to approximate a unknown target function $\breve{\Phi} : \mathcal{D}\ x\ \mathcal{C} \rightarrow \{T, F\}$ using $\Phi : \mathcal{D}\ x\ \mathcal{C} \rightarrow \{T, F\}$ such that $\Phi$and $\breve{\Phi}$ "coincide as much as possible" [34].

A text categorization task can be performed in many ways, for instance one might want to classify a document in just one category (single label case or not overlapping categories) or from 0 to $|\mathcal{C}|$ categories (multilabel case or overlapping categories). A special case of single label TC is the binary TC in which each d_j is assigned to category c_i or its complement $\widetilde{c}_i$.

It is possible to perceive the multilabel case as different problem but one can express the multi-labeled TC task for a set of categories $\mathcal{C} = \{c_1, \ldots, c_k\}$ as a group of k binary case problems, this requires that categories be stochastic independent or each other although some recent development in structured classification [36] might be useful to tackle multilabel interdependent TC task. In this work we are going to deal with binary categorization tasks because as stated by Sebastiani [34] many important TC applications are in fact binary categorization problems and moreover, as described above we can solve multilabel TC as many binary TC task assuming independent categories, which is a valid assumption for most of the problems.

TC heavily relies on the models that have been developed for IR. the reason of this is that TC is a content based document management task, and such it shares many characteristics with other IR tasks such as text search, specifically IR-style indexing using one of the models mentioned in 1.3.1 and the evaluation of effectiveness of the classifier using the measures described below.

1.4.1 Performance Measures

One question that arises after reading the definition of TC is how to measure the how well the classifier function$\breve{\Phi}$ (the approximation or hypothesis) matches the unknown target function Φ (effectiveness). This is generally measured in terms of the classic IR metrics of precision P and recall R but adapted to the specific case of TC. P is defined as the probability that if a given a random document d_x is classified under the category c_i the classification is correct $(P(\breve{\Phi}(d_x, c_i) = T \mid P(\Phi(d_x, c_i) = T))$. Analogously R is defined as the probability that if a random document d_x is ought to be classified under c_i, this decision is taken $(P(\Phi(d_x, c_i) = T \mid P(\breve{\Phi}(d_x, c_i) = T))$.

In binary TC, with we can express the concepts of precision and recall mathematically using the numbers of true positive (TP), false positive (FP), true negative (TN) and false negative (TN) documents classified. The true of false value means that the classification was performed correctly or not, and the positive or negative define whether it belonged to the evaluated category or not. Using these definitions we can write precision P and recall R as:

$$P = \frac{TP}{TP + FP} \quad , \quad R = \frac{TP}{TP + FN}$$

For multilabel classification this only would be local measures for a single category c, in that case the local measures are averaged using different methods such as *microaveraging* and *macroaveraging* [34].

The values of precision and recall for TC vary in an inverse relation, for that reason many articles about TC uses break-even point of P and R [17, 9, 26] which generally implies to modify in some degree the hypothesis. Another approach is to use the $F1$ measure [37, 27, 25] which combines both precision and recall and is defined as:

$$F1 = \frac{2RP}{P + R} \tag{1.2}$$

This implies that $F1$, like P and R is bounded by 1 and the best results under this measure are the higher values.

1.4.2 Applications of Automated Text Categorization

One can think that the application of TC are somewhat limited to automated document classification but many problems can be seen as special instances of TC, Sebastini [34] list some of them:

Document Organization

Maybe the first application that comes to the mind when thinking in TC. Indexing with a controlled vocabulary is an instance of this problem that can be solved with TC techniques. Also any kind of document organization and filing such personal information or enterprise data organization can be viewed as instances of this problem. For example one can think in newspaper ads and the categories in which they should be classified, this is generally done at hand but can be addressed with many of the existing TC techniques. Other applications includes automatic patent classification, automatic news classification under topic sections (e.g Politics, Lifestyles, Sports, etc.).

Text Filtering

Text filtering is the activity of classifying a stream of incoming documents dispatched in an asynchronous way by an information producer to an information consumer [34], for example a newsfeed where the producer is the news agency and the consumer is the producer is the newspaper or a RSS feed where the producer is the website and the consumer the individual users that syndicate the feed.

Automated Metadata Extraction

In digital libraries, one is usually interested in tagging documents by metadata that describes them under a variety of aspects (e.g., creation date, document type or format, availability, etc.). Some of this metadata is thematic, that is, its role is to describe the semantics of the document by means of bibliographic codes, key words or key phrases. The generation of this metadata may thus be viewed as a problem of document indexing with controlled dictionary, and thus tackled by means of TC techniques.

1.4.3 Machine Learning Text Categorization Techniques

The construction of Text Classifiers using an inductive approach have been tackled using different strategies, we are going to mention some of the most popular approaches included the technique that we will use through the rest of the text. There are two ways of classify a item in a category, the "hard" (automated) way which is basically a binary approach and the "soft" o ranking (semi automated).

A ranking classifier is generally accomplished defining a function $CSV_i : D \rightarrow [0, 1]$ that, given a document $d_j$returns a categorization status value for the c_i category (how much the classier believes that the document belong to that category) whereas the hard approach only accepts or reject a document under a defined category.

Probabilistic Classifiers

The probabilistic based text classifiers are very popular, they see the $CSV_i(d_j)$ in terms of $P(c_i|d_j)$. In other words, the probability that a document represented by vector $d_j = (w_{1j}, w_{2j}, \ldots, w_{|M|j})$ (that

can be binary or weighted as mentioned in section 1.3.1) belongs c_i and computes this probability by application of the Bayes theorem.

Decision Trees Classifiers

Probabilistic classifier are quantitative, that is, numeric in nature. This kind of methods are often criticized since, effective as they may be, they are no easily interpretable by humans. Symbolic (non-numeric) techniques no suffer of this problem. In this category decision trees learning is one of the most important example. A decision tree (DT) text classifier [29, 34] is a tree in which internal nodes are labeled by terms, branches departing from them are labeled by tests on the weight that the term has in the test document, and leaf are labeled by categories. This kind of classifier categorizes a test document d_j by recursively testing for the weights that the terms labeling internal nodes have in vector $\overrightarrow{d_j}$, until a leaf node is reached. the label of this node is then assigned to d_j.

Support Vector Machines

Support Vector Machine (SVM) method has been introduced in TC by Joachims [17, 19] and has been user by others in their experiments with good results [9, 26, 25]. Basically SVM tries to find among all the surfaces $\sigma_{1,}\sigma_{2,}\ldots$ in a $|T|$-dimensional space that separate the positive from the negative training examples (decision surface). This is done by finding the decision surface with the widest margin of separation. How is this accomplished and how can be applied to TC will be discussed in depth in chapter 2.

1.5 Parallel Machine Learning

Only recently the topic of parallel implementation of machine learning algorithms have come to importance, although it is rarely addressed at major machine learning conferences, many factors points out that the importance of this factor will increase. In his popular weblog[1] John Langford[2] describes the main tendencies of computation that impulse the necessity of parallel implementation of machine learning algorithms:

1. Larger data sets available due to the availability of more sensors on the Internet, making processor power a need in other to handle these large amounts of information.

2. Serial processor speedups are stalled as the result of no having the ability to drive chips at ever higher clock rates without excessively increasing the cost. The new tendency for improving processor speed is to have more cores per processor [11], that is, creating multicore processors, for example, IBM cell processor[3] has 9 cores.

In addition, since 2005 there is low-cost parallel hardware aimed to desktop users [28] making even more important to take advantage of this architecture and learn to use it the best way.

With all these precedents the following question arises: how take advantage of this trend and use all the parallel computing power of new processor for solving machine learning problems? We are in the beginning of the multicore era and there is no easy answer to this question, mostly because there

[1] Machine Learning (Theory) - http://hunch.net/
[2] http://hunch.net/~jl/
[3] http://www.research.ibm.com/cell/

is not even an unified general approach to take advantage of this architecture, although some basic strategies can be described.

The basic approach is to run the same algorithm with different parameters on different processors. Cluster software like Open Mosix, Condor might be used in this situation. As noticed by Langford, this particular approach does not speed up the run of a particular machine learning algorithm. Other approaches includes the decomposition of the algorithm into statistical queries [21] and parallelize the queries over the the samples, for example using map-reduce [3]. The last and most difficult way is to develop fine-grained parallelism, which is the way brain actually works.

We have discussed the evident speed gain using parallel methods for running but the question whether the a parallel algorithm fine-grained parallelize implementation can achieve better performance is still unanswered. Another question that arises is if there are methods that not necessarily incur in the complexity of dissecting a specific algorithm and then paralleling and still harness parallel architectures. Graf et Al. [12] presents the Cascade SVM, an approach for parallelize the SVM without modifying the SVM problem at all but using a pyramid-like array of SVM subproblem that are joined in each level until a the solution is found. This approach is chosen for the experiments of this work and will be discussed in depth in chapter 3

Chapter 2

Text Categorization With SVM

2.1 Support Vector Machines

Support Vector Machines (SVMs) [38] are powerful classification and regression tools that have been widely applied in the solutions of many problem, generally yielding comparable or even better performance that other algorithms.

The SVM algorithm is based on the concept of VC Dimension[40] and on the principle of structural risk minimization developed by Vapnik [39, 40]. 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, is preferred a classifier with minimum VC Dimension.

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 discrimination function that maximizes the distances within the classes, assuring the lowest probability of error[13]. The aim of the Support Vector classification is to devise a computationally efficient way of learning the optimal separating hyperplanes in high dimensional feature space [7],

2.1.1 Finding the Optimal Hyperplane

Let us consider a training sample $\{(x_i, y_i)\}_{i=1}^{N}$, where x is a point in $\mathbb{R}^n$ that belongs to the group of training examples or input patterns for ith example and d_i is the corresponded, desired response. In this case we are gonna simplifier the setting by stating that d_i can only take two values $\{+1, -1\}$ for the positive and negative classification case respectively. This is case of a binary classification problem, for a multiclass classification problem, we can transform it into different binary classification ones. We also assume that the class represented by $y_i = \{+1, -1\}$ are linearly separable. The equation of a decision hyperplane that separates the data is:

$$w^T x + b = 0 \tag{2.1}$$

Where x is an input vector, w is the weight vector, normal to the separation hyperplane, and b is the offset or bias:

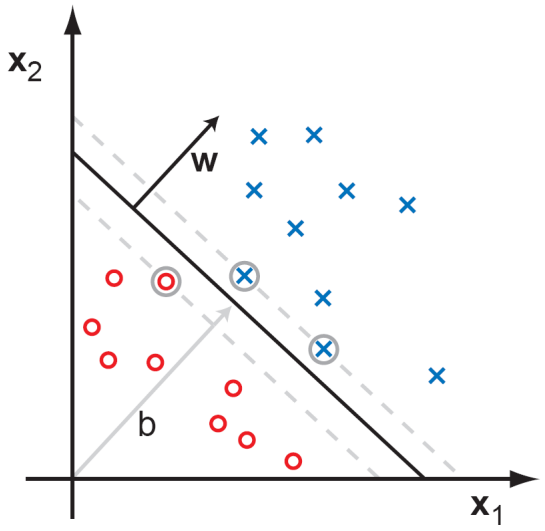


Figure 2.1: The red circles and blue x's are the training data, the red circles for the $y_i = -1$ class and the blue x's for the $y_i = +1$ class. The data inside the grey circles are the support vectors that maximize the distance between the data and the hyperplane defined by w, a normal vector to the hyperplane and the offset b

$$\begin{aligned} w^T x + b \geq 0 \qquad & for \quad y_i = +1 \\ w^T x + b < 0 \qquad & for \quad y_i = -1 \end{aligned}$$

For a given vector w and offset b the distance between the the hyperplane defined in and a point x is called margin of separation represented by the equation2.1.

Then we can write the discriminant function:

$$g(x) = w^T x + b \tag{2.2}$$

An the classification function as:

$$y = sign\{w^T x + b\} \tag{2.3}$$

Equation 2.2 gives a measure of the distance from x to the optimal hyperplane, and 2.3 assign a category for a specific point.

As described above, the complexity of the classifier should be minimized in order to achieve a good generalization, for linear classifier this equivalent lowering that complexity is to maximize the margin of separation between the points and the hyperplane. An approach to find this separator is to maximize the margin between two parallel supporting planes, if we define the margin as follows:

$$\begin{aligned} w^T x + b = \ & 1 \\ w^T x + b = & -1 \end{aligned}$$

Then the margin of separation is equal then to $\gamma = \frac{2}{w}$. Now our problem becomes maximize $\frac{2}{w}$ which is equivalent to minimize $||w||_2/2$ in the following quadratic programming problem:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}||w||^2 \\ s.t \quad & y_i(w^T x_i + b) \geq 1 \end{aligned} \tag{2.4}$$

Condition $d_i(w^T x_i + b) \geq 1$ specify that all the data points have to be classified correctly. In order to solve this problem we introduce the Lagrangian formulation:

$$\begin{aligned} \max_{\alpha} \min_{w,b} \; J(w,b,\alpha) \quad &= \quad \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i [y_i(w^T x_i + b) - 1] \\ s.t \quad & \alpha_i \geq 0 \end{aligned} \tag{2.5}$$

Now the conditions of optimality for $J(w,b,\alpha)$ are given by:

$$\frac{\delta J(w,b,\alpha)}{\delta w} = 0 \longrightarrow w = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{2.6}$$

$$\frac{\delta J(w,b,\alpha)}{\delta b} = 0 \longrightarrow \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.7}$$

Replacing 2.6 and 2.7 into 2.5 and defining the cost function as $J(w,b,\alpha) = Q(\alpha)$ yields:

$$\begin{aligned} min\, Q(\alpha) \quad &= \quad \sum_{i=1}^{N} \alpha - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ s.t \quad & \alpha_i \geq 0 \; and \; \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{2.8}$$

Which is a quadratic programming problem (QP). Notice that the dual problem is cast entirely in terms of the training data. The fact that this is a QP problem is one of the strength of this methodology and also a weakness. It's a strength because as opposed to others techniques such as Neural Networks, the optimization problem is convex and a global optimum is always reached in a finite amount of time [17]. On the other hand, a QP is a computationally expensive problem, this property has led to the development of both, novel ways of solving the problem described in equation 2.8 and alternative formulations for solving the same optimization problem [35, 20], with varying degrees of performance. The x_i's whose α_i are not defines are the support vectors, that is they are on the hyperplanes that defines the maximum margin separator.

When all points cannot be shattered by one Hyperplane an error bound is added in the form of a non negative slack variable, the primal problem described in 2.4 becomes:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i \\ & y_i(w^T x_i + b) + \xi_i \geq 1 \\ s.t \quad & \\ & \xi_i \geq 0 \quad i = 1, ..., m \end{aligned} \tag{2.9}$$

Where C is a user-specified positive parameter and depends on the data, empirical ways of finding the adequate C exist. After following a similar procedure to the one that lead us to 2.8 we now have:

$$\begin{aligned} min\, Q(\alpha) \quad = \quad & \sum_{i=1}^{N}\alpha - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j x_i^T x_j \\ s.t \quad & \\ & \alpha \geq 0 \\ & \sum_{i=1}^{N}\alpha_i y_i = 0 \\ & 0 < \alpha_i \leq C \quad i = 1, ..., m \end{aligned} \tag{2.10}$$

The C parameter defines the error bound, and in plain words in states how "hard" the classifier should try to correctly classify a given point before removing it from the training set. Once calculated the α_i the optimal weight vector w_o and offset b_o can be calculated as follows:

$$\begin{aligned} w_o = \quad & \sum_{i=1}^{N}\alpha_i y_i x_i \\ & \\ b_o = \quad & 1 - w_o^T x^{(s)} \end{aligned} \tag{2.11}$$

Where $x^{(s)}$ is any support vector. We can also write the discriminant function as follows:

$$y = sign\{\sum_{i=1}^{N} d_i\alpha_i x_i^T x + b_0\} \tag{2.12}$$

Many times a linear separator is not enough to classify the data properly, geometrically speaking, a more complex surface is needed in order to separate the data correctly. In order to classify the data a non-liner mapping into a high-dimensional feature space where the data is linearly separable, the Cover's theorem on separability [6] supports this procedure . in order to accomplish this, SVM uses the so called "Kernel Trick" which consist in the usage of a Kernel function that maps the input data into a higher dimensional space. The reader may consult the appropriate literature [13, 14, 7] for more information on this method. The dual for formulation using this approach is:

$$
\begin{aligned}
min\, Q(\alpha) \quad &= \quad \sum_{i=1}^{N} \alpha - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j K(x_i, x_j) \\
s.t & \\
& \alpha \geq 0 \\
& \sum_{i=1}^{N} \alpha_i d_i = 0 \\
& 0 < \alpha_i \leq C \quad i = 1, ..., m
\end{aligned} \tag{2.13}
$$

And the discriminant function is:

$$y = sign\{\sum_{i=1}^{N} d_i \alpha_i K(x_i, x) + b_0\} \tag{2.14}$$

There are many available Kernel functions, and they depend on the characteristics of the input data. In this work we are going to use a linear SVM as defined in equation 2.10, which also can be viewed as 2.13 equation using the kernel function: $K(x_i,x_j) = x_i^T x_j$.

2.2 SVM and Text Categorization

The usage of SVM was first introduced by Joachims [17, 19] and similar setups have been used in posterior literature[9]. 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 which can be done using any of the models described back in section 1.3.1. As argued by Joachims [17], 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. In real life scenarios, TC problems involves high dimensional data and large amount of computing processing. As mentioned before, the fact that SVM need a lot of calculation power for training makes it an important motivation for using a parallel algorithm when training SVM.

2.3 Transductive Learning for SVM

So far with the formulations described in the last section we are using what is called inductive learning, in other words we start from a set of examples, and based on them we try to find a classificator function that classifies correctly the data. This is called Inference learning, that is, going from particular examples to a general hypothesis. The Transductive setting on the other hand, tries to use the known distribution of the test data in order produce a classifier.

This setting has a clear application when we don't care about the particular function; we only need to classify a given set of examples (i.e test set) with as few errors as possible. The Transductive inference uses the training examples previously labeled and the unclassified examples to generate an optimal partition and labeling of the unclassified examples, using the prior knowledge provided by the distribution of the unclassified examples.

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 is minimized. But, why not use the learning rule obtained by means of the training examples to classify the test examples? The problem arises when a small training set is use. The classifier will have a "poor" generalization due to the lack of knowledge about the distribution of points in the space X [10].

The classifier function estimated with very few points will lead to disappointing results. Unlike of the inductive setting, the Transductive setting uses the location of the test examples when defining the structure. Such structure corresponds to a structure of possible hypothesis solution. Using prior knowledge about the nature of testing data provides extra information to built an appropriate structure. The structure is build based on the margin on both the training and the test data.

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$((C^*_- < C^*) \parallel (C^*_+ < C^*))\{$ // 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 2.2: Algorithm for training Transductive Support Vector Machines [19]

[TODO Description of the algorithm] For a complete description and a in depth study of the Joachims' approach see the nominal paper [19] and work of Collobert [5].

Chapter 3

Parallel Methods for Training Transductive SVM

3.1 Introduction

Solving the Transductive SVM using the algorithm described by Joachims implies 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 in the fact that there is only a finite number 2^U ways of labeling of U unlabeled points and it is unlikely that all of them are examined [5]. 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 [18] thus still intractable for large data sets.

In order to optimize this kind problem one can try to modify the formulation, for instance example solving it in the primal rather than in the dual with an alternate technique although the question of how widely applicable are these methods are remains to bee seen. or use any of the general approaches described in section 1.5 that makes use of the computation power given by the new multicore processors.

For the specific case of SVM and TSVM one could try to modify the problem in a way that makes it easily parallelizable [42] Generally this is accomplished through the use of decomposition technique of some kind. Another approach is to use a mixture of SVM [4] in order to to train each of them with a subset and them using the mixture of expert approach [15] which is easily parallelizable. All these approach modify in some manner the original formulation of the problem.

A recent approach called the Cascade SVM consist in an array of SVM solvers arranged in pyramid form. This structure allows an easy parallelization and does not modify the original formulation at all [12, 16]. This is the approach selected for the parallelization of TSVM because there is no need to modify the original quadratic programming problem and is straightforward to implement. Some small modifications are needed in order to be fully adapted to the Transductive algorithm described by Joachims.

3.2 The Cascade SVM

The Cascade SVM was described by Graf et al [12] and further explored by Zhang et al [16] and is based in the strategy of early elimination of non-support vectors from the training set [18]. The Cascade SVM is a distributed architecture where smaller optimization problems are solved independently making them easily parallelizable. The results of this smaller problems are then assembled in a way that is ensured that it eventually converges to globally optimal solution.

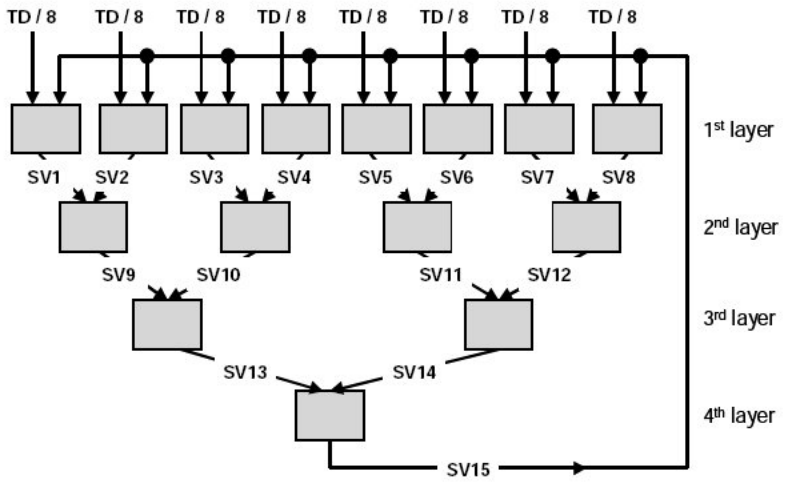
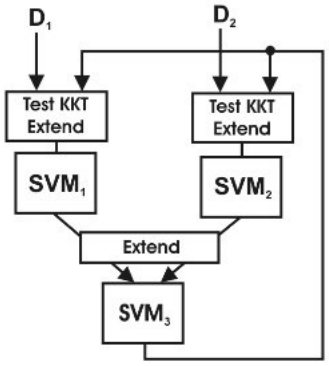


Figure 3.1: Schematic of a Cascade architecture [12]

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 as shown in figure 3.1. The data is split into subsets and everyone of them is evaluated for to find support vectors in the first layer, then the results are combined two-by-two and entered as training set for the next layer. Often a single pass through the cascade produces satisfactory results and if it is not the case, that is, if solution is not the global maximum, the result of the last layer is fed back into the first layer, where each of the SVM of the first layer receives all the support vectors and test them with fraction of his inputs to check if any of them have to be incorporated into the optimization. If this is not the case for all SVM of the input layer, the cascade have converged otherwise it proceeds with another pass through the network. The formal proof of the convergence can be found in [12].

3.2.1 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. There are at least two approach for this, the first approach is to make each of theSVM in the array Transductive, using the algorithm described in Figure 2.3. Even if this might seem obvious ultimatly the loop in each of the SVM will be run in just one processor so the only possible speed up would be in terms of the number of training and testing data available. Also splitting and mergin the subsets might need additional modifications, despite this fact, it might be interesting to vizualise the behaviour of this



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

Figure 3.2: 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 3.1 and are entered into SVM_3. [12]

approach and for that reason is left as a future work.

The other approach is 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 2.3. Now for each iteration of the loop the a SVM inductive problem is solved in a parallel manner. We have two goals to achieve 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 a speed up of the process through the use of the parallel architecture.

3.2.2 Merging Data Sets for the Transductive Case

In this section it is described the procedure to merge two subsets using as example the description in the figure 3.2.2. For Transductive SVM there is not practically any change besides that when recreating creating the problem for SVM_3 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 that is made on both groups, the labeled and the unlabeled data.

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

$$W = \frac{1}{2}\begin{bmatrix} \vec{\alpha_1} \\ \vec{\alpha_2} \end{bmatrix}^T \begin{bmatrix} Q_1 & Q_{12} \\ Q_{21} & Q_2 \end{bmatrix} \begin{bmatrix} \vec{\alpha_1} \\ \vec{\alpha_2} \end{bmatrix} + \begin{bmatrix} \vec{e_1} \\ \vec{e_2} \end{bmatrix}^T \begin{bmatrix} \vec{\alpha_1} \\ \vec{\alpha_2} \end{bmatrix} \tag{3.1}$$

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

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

One thing that should be noted is the error values handling. As the reader can see in 2.3 the decision whether to swap the values of the labels for each uncategorized data depends on the error of the classification of that particular value but as in each layer we eliminate them in the process, we have

to keep a record of the classification error of the eliminated unlabeled vector because we are going to use in the decision of the second loop.

Chapter 4

Experiments and Results

In this section we present the experimental results for 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 [24] data set found in the UCI ML Repository[1]. The data set was preprocessed using English stop-words and the Porter stemmer algorithm [31] in a similar fashion as described in section 1.3.2 and the feature vectors were created using the *tf-idf* formula. The code was written in Ruby using an already existing implementation of the stemmer algorithm.

4.1 Experiments

The 20 Newsgroup data sets is set is a collection of approximately 20,000 newsgroup documents, partitioned evenly across 20 different newsgroups. The collection is organized in the following form:

• comp.graphics • comp.os.ms-windows.misc • comp.sys.ibm.pc.hardware • comp.sys.mac.hardware • comp.windows.x	• rec.autos • rec.motorcycles • rec.sport.baseball • rec.sport.hockey	• sci.crypt • sci.electronics • sci.med sci.space
• misc.forsale	• talk.politics.misc • talk.politics.guns • talk.politics.mideast	• talk.religion.misc • alt.atheism • soc.religion.christian

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

.

As we can see 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 an "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 nearly separable problem. The hard task we chose related news groups were is expected that the most important terms similar for both collection. We chose *comp.sys.ibm.pc.hardware* 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 ulabeled set (80%). The unlabeled set for the TSVM was used as the testing set for the classic SVM. We also made some test with fewer training data but with the same amount of test data in order to visualize better the goods of transductive inference when only few training data is available.

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 [19, 9], 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. This parameter is of great importance for the computation time of the overall algorithm because the annealing used by Joachim's algorithm, for that reason we wanted to improve the computation time by choosing the right C* parameter so we tested the implementation effectiveness with the original Joachim's heuristic $C = C*$.

The Cascade architecture used in the experiments were formed 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.

4.1.1 Toy Experiment

With the objective of prove the methods and their performance, a test with few training and test samples was carried out first, the number of features is also less than the full problem formulation because only a fraction of these were preprocessed for this initial test. For this toy experiments the hard task was performed with 200 testing messages, 100 from *comp.sys.ibm.pc.hardware* labeled positive and 100 from *comp.sys.mac.hardware* labeled negative. The training set consisted also of 200 messages with the same characteristics. In these experiments (Table 4.1) the number of training samples varies for each test so we can appreciate the difference in the performance of the methods. T stand for number of training samples and S for number of testing samples. We can appreciate that transduction algorithm perform much better than inductive when few training samples, a very attractive characteristic for many real problems. Even when the number of training samples is equal to the number of testing samples the transductive outperforms the inductive algorithm for the F measure.

T: 30 S: 200	P	R	$F1$
SVM	0.44	0.8148	0.5714
TSVM	0.7	0.7	0.7
TSVM Parallel	0.7	0.7	0.7

T: 100 S: 200	P	R	$F1$
SVM	0.94	0.7581	0.8393
TSVM	0.84	0.84	0.84
TSVM Parallel	0.84	0.84	0.84

T: 200 S:200	P	R	$F1$
SVM	0.94	0.7666	0.8624
TSVM	0.87	0.87	0.87
TSVM Parallel	0.87	0.87	0.87

Table 4.1: Small Scale Experiment: *comp.sys.ibm.pc.hardware* and *comp.sys.mac.hardware* with 200 messages from each newsgroup.

4.1.2 Easy Task

As mentioned above the easy task is an experiment were is expected a linear separable problem 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 4.3 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 more there is more unlabeled data the differences in classification performance begin to be more noticeable, even with this "easy" problem. Regarding the speed, 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. One thing that can be noticed in the tables is that the parallel version of the algorithm no necessarily is slower when facing much more data, this is due to the process of early elimination of non-support vector and it may happen that with much more data more non-support vectors are eliminated in proportion with the amount of unlabeled data, making the whole process faster. This particular property can be really appealing when working with larger datasets, of course it will depend entirely on the dataset.

4.1.3 Hard Task

This is task is formed from 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

	#Labeled	#Unlabeled	P	R	$F1$	Time (min)
SVM	200	N/A	0.9245	1.0	0.9608	0.0137
TSVM	200	200	0.9847	0.9847	0.9847	10.2205
TSVM Parallel	200	200	0.9847	0.9847	0.9847	7.2050

	#Train	#Unlabeled	P	R	$F1$	Time (min)
SVM	200	N/A	0.9384	1	0.9682	0.0135
TSVM	200	400	0.9823	0.9823	0.9823	44.4103
TSVM Parallel	200	400	0.9823	0.9823	0.9823	15.3406

	#Train	#Unlabeled	P	R	$F1$	Time (min)
SVM	100	N/A	0.9062	1	0.9508	0.0135
TSVM	100	400	0.9823	0.9823	0.9823	32.1135
TSVM Parallel	100	400	0.9823	0.9823	0.9823	11.0953

	#Train	#Unlabeled	P	R	$F1$	Time (min)
SVM	100	N/A	0.9975	0.8957	0.9438	0.0135
TSVM	100	800	0.9798	0.9798	0.9798	214.2297
TSVM Parallel	100	800	0.9798	0.9798	0.9798	68.4745

	#Train	#Unlabeled	P	R	$F1$	Time (min)
SVM	100	0	0.8966	0.9975	0.9444	0.0135
TSVM	100	1600	0.9800	0.9800	0.9800	278.3236
TSVM Parallel	100	1600	0.9800	0.9800	0.9800	41.53950

Table 4.3: The "Easy" Task: *comp.graphics* and *alt.atheism*. Using C=10

	#Train	#Unlabeled	P	R	$F1$	Time (min)
SVM	100	N/A	0.7400	0.8916	0.8087	0.0025
TSVM	100	200	0.8600	0.8600	0.8600	1.5370
TSVM Parallel	100	200	0.8700	0.8700	0.8700	1.4863

	#Train	#Unlabeled	P	R	$F1$	Time (min)
SVM	100	N/A	0.8830	0.7550	0.8140	0.0025
TSVM	100	400	0.8550	0.8550	0.8550	6.0628
TSVM Parallel	100	400	0.8550	0.8550	0.8550	4.3323

	#Train	#Unlabeled	P	R	$F1$	Time (min)
SVM	100	N/A	0.7350	0.8724	0.7978	0.0025
TSVM	100	800	0.8200	0.8200	0.8200	31.0015
TSVM Parallel	100	800	0.8200	0.8200	0.8200	19.5472

	#Train	#Unlabeled	P	R	$F1$	Time (min)
SVM	100	0	0.7362	0.8624	0.7943	0.0135
TSVM	100	1600	0.8187	0.8187	0.8187	221.5013
TSVM Parallel	100	1600	0.8187	0.8187	0.8187	86.1924

Table 4.5: The "Hard" Task: of *comp.sys.ibm.pc.hardware* and *comp.sys.mac.hardware*. Using C=9

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 knew 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.

4.2 Conclusion And Future Direction

In this work we have implemented a parallel version of TSVM as described by Joachims' [19], an approach that because its annealing heuristics requires many iterations solving a similar problems. Our approach, in order 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. This architecture scales very well, therefore is expected that, with more processors available the the speed gains will be even more noticeable. In addition, more layers in basic architecture could help in the improvement of the computation time.

From the data tables one could argue that the classification performance gains for using TSVM does 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 in 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 C* for the transductive setting does not generally match in the sense that the best C parameter for the inductive problem does not produce the best classification performance in the transductive problem for the given set of data. However, we have chosen the parameter that works best for the inductive setting.

However it is truth that more optimization strategies have to be developed in order to make this algorithm really usable in 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 the scope of this work and 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 the labels of two of the entries) there is not need 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 the most of the iterations we would only need to calculate 3 problems.

Another strategy would be try new improved heuristics as described by Collobert et al [4] 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 Joachim's TSVM is 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. [41]could be applied to each one

of the SVM solvers in order to have multilevel parallelization, further improving the computation performance.

Bibliography

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

[2] Berger, A. *Statistical machine learning for information retrieval*. PhD thesis, 2001. Chair-John Lafferty.

[3] Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G. R., Ng, A. Y., and Olukotun, K. Map-reduce for machine learning on multicore. In *NIPS* (2006), B. Schölkopf, J. Platt, and T. Hoffman, Eds., MIT Press, pp. 281–288.

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

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

[6] Cover, T. M. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *Electronic Computers, IEEE Transactions on EC-14*, 3 (1965), 326–334.

[7] Cristianini, N., and Shawe-Taylor, J. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, March 2000.

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

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

[10] Espinosa, J. Kernel methods for improving text search engines, msc thesis, 2003.

[11] Geer, D. Industry trends: Chip makers turn to multicore processors. *Computer 38*, 5 (2005), 11–13.

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

[13] Haykin, S. *Neural Networks: A Comprehensive Foundation (2nd Edition)*. Prentice Hall, July 1998.

[14] Herbrich, R. *Learning Kernel Classifiers: Theory and Algorithms (Adaptive Computation and Machine Learning)*. The MIT Press, December 2001.

[15] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixture of local experts. *Neural Computation 3* (1991), 79–87.

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

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

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

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

[20] Joachims, T. Training linear SVMs in linear time. In *ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)* (2006), pp. 217 – 226.

[21] Kearns, M. Efficient noise-tolerant learning from statistical queries. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing* (New York, NY, USA, 1993), ACM, pp. 392–401.

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

[23] Keller, M., and Bengio, S. A neural network for text representation. In *ICANN (2)* (2005), pp. 667–672.

[24] Lang, K. 20 newsgroups data set.

[25] Lewis, D. D. Evaluating text categorization. In *HLT '91: Proceedings of the workshop on Speech and Natural Language* (Morristown, NJ, USA, 1991), Association for Computational Linguistics, pp. 312–318.

[26] Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res. 5* (2004), 361–397.

[27] Manevitz, L. M., and Yousef, M. One-class svms for document classification. *J. Mach. Learn. Res. 2* (2002), 139–154.

[28] Marowka, A. Parallel computing on any desktop. *Commun. ACM 50*, 9 (2007), 74–78.

[29] Mitchell, T. M. *Machine Learning*. McGraw-Hill Science/Engineering/Math, March 1997.

[30] Osuna, E., Freund, R., and Girosi, F. Training support vector machines: an application to face detection. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)* (Washington, DC, USA, 1997), IEEE Computer Society, p. 130.

[31] Porter, M. F. An algorithm for suffix stripping. 313–316.

[32] Salton, G., and Buckley, C. Term weighting approaches in automatic text retrieval. Tech. rep., Ithaca, NY, USA, 1987.

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

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

[35] Suykens, J., Gestel, T. V., Brabanter, J. D., Moor, B. D., and Vandewalle, J. *Least Squares Support Vector Machines.* World Scientific Publishing, 2002.

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

[37] Van Rijsbergen, C. J. *Information Retrieval, 2nd edition.* Dept. of Computer Science, University of Glasgow, 1979.

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

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

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

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

[42] Zanni, L., Serafini, T., and Zanghirati, G. Parallel software for training large scale support vector machines on multiprocessor systems. *J. Mach. Learn. Res. 7* (2006), 1467–1492.