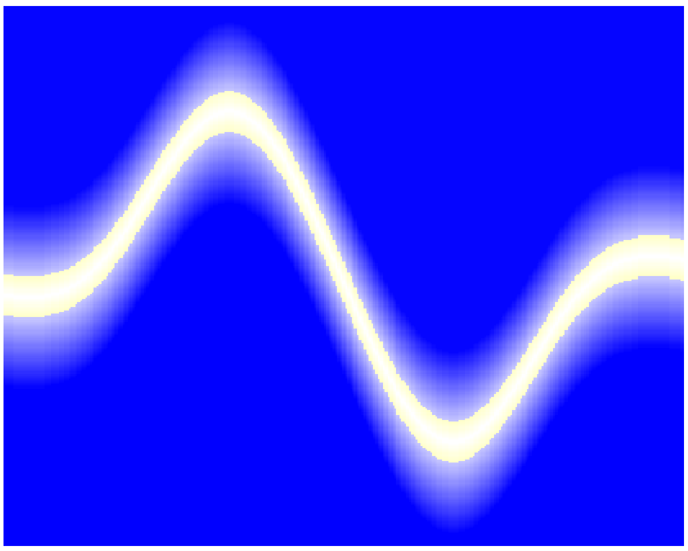
Online Support Vector Regression

Francesco Parrella



A Thesis presented for the degree of
Information Science

Department of Information Science
University of Genoa
Italy

June 2007

Online Support Vector Machines for Regression

Francesco Parrella

Submitted for the degree of Information Science
May 2007

Abstract

Many approaches for obtaining systems with intelligent behavior are based on components that learn automatically from previous experience. The development of these learning techniques is the objective of the area of research known as machine learning. During the last decade, researchers have produced numerous and outstanding advances in this area, boosted by the successful application of machine learning techniques. This thesis presents one of this techniques, an online version of the algorithm for training the support vector machine for regression and also how it has been extended in order to be more flexible for the hyper parameter estimation. Furthermore the algorithm has been compared with a batch implementation and tested in a real application at the Department of Information, Systematics and Telematics in Genoa.

Declaration

The work in this thesis is based on research carried out at the Department of Information, Systematics and Telematics, Italy. No part of this thesis has been submitted elsewhere for any other degree or qualification and it all my own work unless referenced to the contrary in the text.

Acknowledgements

I would like to thank all people who helped me in a way or in another to have the skills necessary to complete this thesis. In alphabetic order, they are: Alessandra, Andrea, Anna, Anna Maria, Antonio, Barbara, Daniela, Debora, Emanuele, Enrico, Francesco, Giacomo, Giorgia, Giorgio, Ilaria, Katia, Lara, Livio, Lorena, Marianna, Marco, Marina, Matteo, Mil, Nicola, Paolo, Pellegrino, Roald, Salvatore, Stefania, Simona, Simone, Valentina, Vera.

Contents

Chapter 1

Online Support Vector Machines for Regression

The field of machine learning is expanding in the last years, and many new technologies are growing using these principles. Among the various existing algorithms, one of the most recognized is the so-called support vector machine for classification (SVM) and regression (SVR), which permit the creation of systems which, after training from a series of examples, can successfully predict the output at an unseen location performing an operation known as induction.

One of the drawbacks of this algorithm is that it works in batch mode only. In this form, learning is carried out starting from all the examples and adding a new example means reprogramming the whole system from the beginning.

The algorithm, which I have studied and implemented, is an online version of support vector machine for regression, in other words it is possible to add and delete elements in the machine without restarting the training from the beginning.

My implementation has also been compared to libsvm, one of the batch implementations of SVM, which is one of the most frequently used libraries in research [8].

The work elaborated on this thesis can be broken down into five parts:

- *The study of the algorithm*, beginning from the papers of some researchers, from the mathematical point of view as well as its implementation, correcting some inexact formulas.

- *The implementation* of the algorithm is in prototypical form in Matlab as well as an efficient version in C++

- *Test* of the performance and the results of the online version in comparison to that of batch version.

- *Practical case* testing the algorithm in practice for the prediction of the applied voltages needed to compensate the gravity load of the arm of a robot (with three degrees of liberty).

- *Improving the algorithm* with the creation of a stabilizing technique which correct the errors approximation due to the finite precision floating point operations of the computers. This also permits to extend the functionality of the algorithm.

This thesis describes the most part of the actual work. Other information as well as the software developed and the documentation are available at http://onlinesvr.altervista.org.

The thesis is organized as follow:

- *Chapter 2*: introduction to the notion of learning machines, the biological motivations and the possible applications. It will also be presented the principal types of machine learning.

- *Chapter 3*: here are explained, very informally, how the SVM algorithm works, which are its characteristics and its potential.

- *Chapter 4*: this chapter represents the heart of the theoretical part of the algorithm. Demonstrations and the mathematical principles of SVR will be shown. The stabilization problem and resolution will also be addressed.

- *Chapters 5*: this is the algorithmic part of the thesis, in which the pseudo-code of the algorithm is shown and furthermore some implementation techniques to be the algorithm more efficient will be explained.
- *Chapter 6*: here complexity analysis is presented. Be it from the time point of view as well as the one of space. We will also indicate some techniques implemented to reduce its complexity.
- *Chapter 7*: here we will compare the libsvm batch implementation with onlinesvr. We will also show the efficiency of stabilization (a technique which extends the functionally of SVR) and the description of the experiment carried out at Lira-Lab on the mechanical arm.
- *Chapter 8*: conclusions and future work.
- *Appendix A*: contains a brief description to all the the software developed. Recommended to all those who wish to try out this algorithm.
- *Appendix B*: are recommendations formulated from many researchers on how to find the correct parameters values.

Chapters 2-3 are only an introduction to the topic, and are recommended for those not fully acquainted to kernel methods and SVM in particular. Chapters 4 is directed towards those whom wish to understand deeply the mathematical aspect behind the algorithm. The technical part is recommended only for the brave reader! Chapters 5-6 contain the pseudo-code of the algorithm, the choice of implementation as well as its analysis; this part is recommended only who wish to study the algorithm. Chapter 7, the one for the test, has been written for whomever has the desire to use the algorithm for practical cases, as a matter of fact it contains the speed comparison with libsvm. And last but not last is the conclusion (chapter 8) that contains the summary of the work done.

Chapter 2

Learning

In this chapter we explain the learning basis, from the biological point of view to the mathematical one and we will give a brief definition on learning types.

2.1 What is learning

Nowadays, the meaning of "learning" has many definitions. The purpose of this first chapter is try to explain the definition of learning used in this thesis and in machine learning. First of all, we will start with the biological definition:

Figure 2.1: What is learning?

Learning is the process of acquiring knowledge, skills, attitudes, or values, through study, experience, or teaching, which causes a change of behavior that is persistent, measurable, and specified or allows an individual to formulate a new mental construction or revise a prior mental one (conceptual knowledge such as attitudes or *values). It is a process that depends on experience and leads to long-term changes in behavior potential. Behavior potential describes the possible behavior that an individual may have (not actual behavior) in a given situation in order to achieve a goal. But potential is not enough; if individual learning is not periodically reinforced, it becomes shallower and shallower, and eventually will be lost.* [12]

This definition is the one that we use in everyday life and one that we have known since we were children.

2.2 The learning process

The most basic learning process is imitation, one's personal repetition of an observed process, such as a smile. Thus an imitation will take one's time (attention to the details), space (a location for learning), skills (or practice), and other resources (for example, a protected area). Through copying, most infants learn how to hunt (i.e., direct one's attention), feed and perform most basic tasks necessary for survival. Bloom's Taxonomy divides the learning process into a six-level hierarchy, where knowledge is the lowest order of cognition and evaluation the highest:

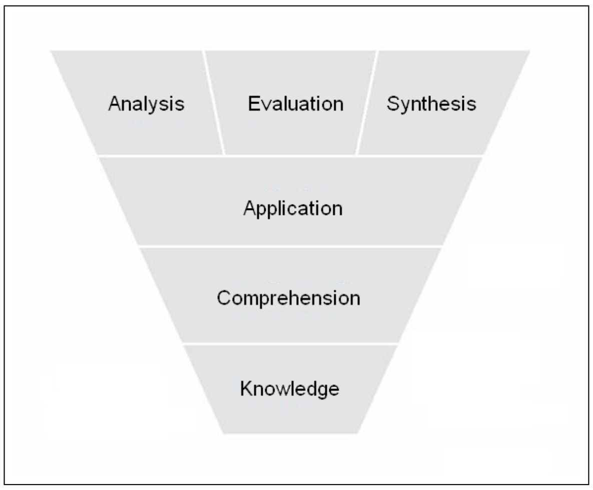


Figure 2.2: Bloom's Taxonomy

- **Knowledge** is the memory of previously-learnt materials such as facts, terms, basic concepts and answers.

- **Comprehension** is the understanding of facts and ideas by organization, comparison, translation, interpretation, and description.
- **Application** is the use of new knowledge to solve problems.
- **Analysis** is the examination and division of information into parts by identifying motives or causes. A person can analyze by making inferences and finding evidence to support generalizations.
- **Synthesis** is the compilation of information in a new way by combining elements into patterns or proposing alternative solutions.
- **Evaluation** is the presentation and defense of opinions by making judgments about information, validity of ideas or quality of work based on a set of criteria.

2.3 Machine learning

Starting from ”biological learning”, many studies have tried to reproduce and simulate the human learning techniques for machines, developing a new area of science called *machine learning*. But why do we want to try to teach computers how to learn? In actual fact, a pc can do the same job which that in a normal person would take years. Moreover, pc’s seem able to do all we need. Unfortunately, many problems cannot be solved using common programming techniques. One example would be characters recognition. People can easily distinguish alphabet characters, to make words and give them a meaning. Instead a computer see characters such as a sequence of 0 and 1, without semantic understanding.

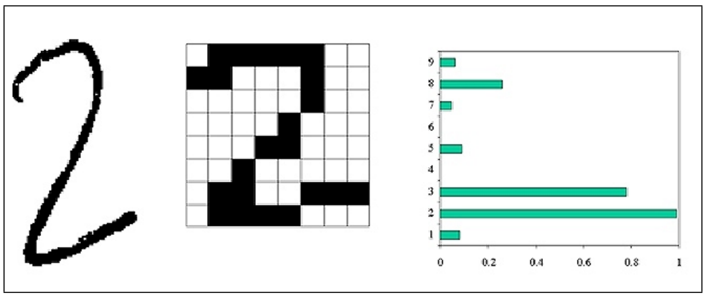


Figure 2.3: How machines see characters

With machine learning, after a training phase, it is possible to associate each character to a meaning, and therefore distinguish one from another.
Other problems that require machine learning are virus detection, the DNA sequence analysis, voice recognition, and many others.

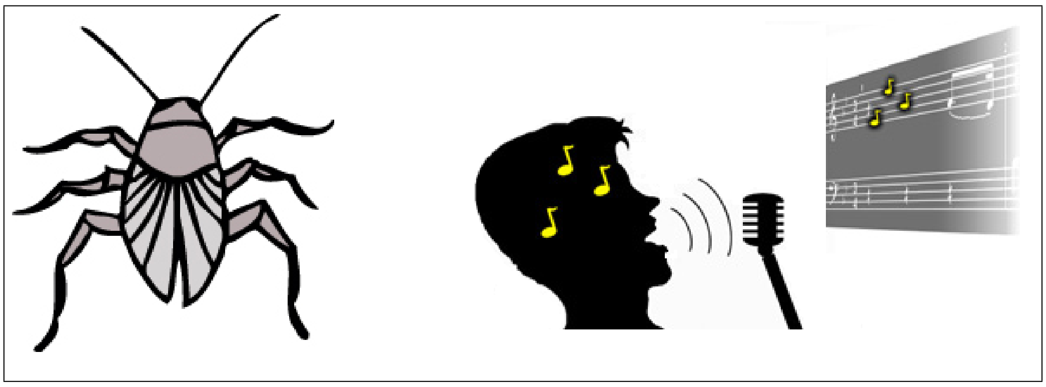


Figure 2.4: Machine learning applications: virus detection and speech recognition

Actually, all the computational learning techniques developed, reach only the first three levels of Bloom's Taxonomy, but research is growing fast...

2.4 The machine learning process

Learning algorithms developed up until now, can be divided into these categories:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

2.4.1 Supervised learning

Supervised learning is a machine learning technique for creating a function from training data. The training data consist of pairs of input objects (typically vectors), and desired outputs. The output of the function can be a continuous value (as in regression), or can predict a class label of the input object (for classification). The task of the supervised learner is to predict the value of the function for any valid input point after having seen a finite number of training examples.

The main algorithms of supervised learning are: neural networks, nearest neighbor algorithm, decision tree learning, and support vector machines, the algorithm analyzed in this thesis.

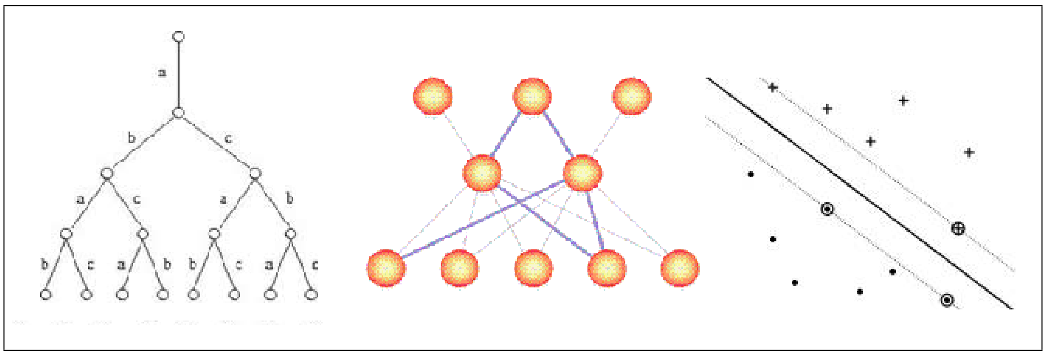


Figure 2.5: Supervised learning algorithms: a decision tree, a neural network and a support vector machine

2.4.2 Unsupervised learning

Unsupervised learning is literally learning without supervisors; only input data without output data (answer) are given. The goal of unsupervised learning depends on situations and the unsupervised learning problem is sometimes not mathematically well-defined. For example, data clustering aimed at grouping similar data. In data clustering, how to measure the affinity between data samples needs to be predetermined. However, there is no objective criterion that can quantitatively evaluate the validity of the affinity measure; often it is subjectively determined.

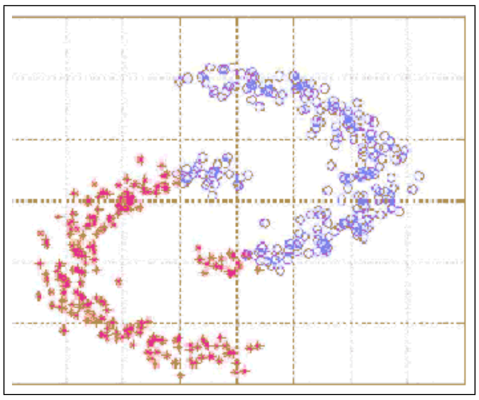


Figure 2.6: Unsupervised algorithm example: clustering

2.4.3 Semi-Supervised learning

Semi-supervised learning is a type of machine learning technique which makes use of both unlabeled and labeled data for training - typically a small amount of labeled data with a large amount of unlabeled data. Many researchers of machine learning have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce considerable improvement in learning accuracy. The acquisition of labeled data for a learning problem often requires a skilled human agent to manually classify training examples. The cost associated with the labeling process thus may render a fully labeled training set infeasible, whereas acquisition of unlabeled data is relatively inexpensive. In such situations, semi-supervised learning can be of great practical value.

One example of a semi-supervised learning technique is co-training, in which two or possibly more learners are each trained on a set of examples, but with each learner using a different, and ideally independent, set of features for each example.

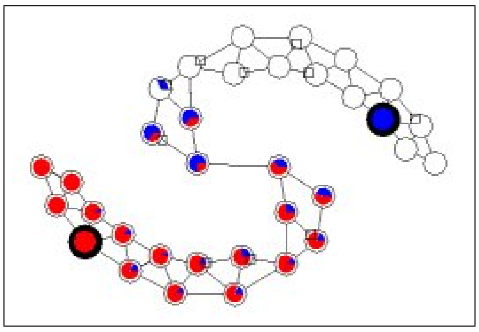


Figure 2.7: Semi-supervised learning example

2.4.4 Reinforcement learning

Reinforcement learning refers to a class of problems in machine learning which postulate an agent exploring an environment in which the agent perceives its current state and takes actions. The environment, in return, provides a reward (which can be positive or negative). Reinforcement learning algorithms attempt to find a policy for maximizing cumulative reward for the agent over the course of the problem. The principal algorithms are temporal difference learning and actor critic reinforcement learning.

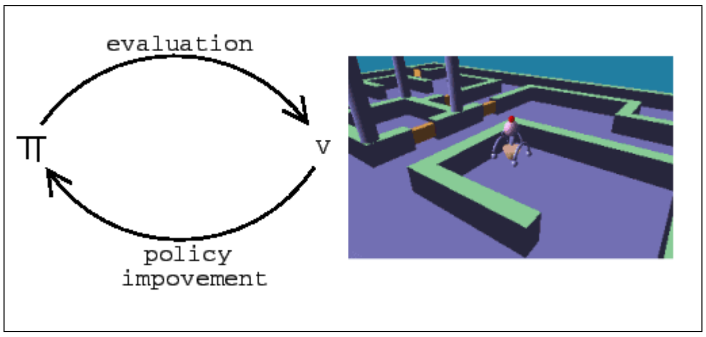


Figure 2.8: Reinforced learning example

Chapter 3

Support Vector Machines

3.1 Main Concepts

Support Vector Machines (SVM), mentioned in previous chapter, form part of Supervised Learning, a branch of statistical learning which part from a series of examples that create a "decision-maker" system which tries to predict new values. This technique can be subdivided into two distinct parts:

- **Learning**: consists of training SVM with examples at its disposition
- **Prediction**: where new samples are inserted in which the result is not known. Results produced are the more probable respect to those examples used in the learning phase

Each example can be written as a pair (input, output) where input is the data set and output makes up how they should be catalogued.

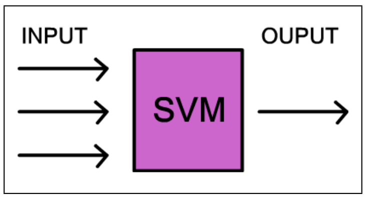


Figure 3.1: SVM high-level view

Mathematically, examples can be regarded as a pair (x, y) where x is a vector of real numbers and output can be a boolean value (like as yes/no, 1/-1, true/false), or a real number. In the first case, we speak of a problem of classification, in the latter of a problem of regression. Moreover, the results of learning can be compared to a real function in which the pairs (x, y) are points in the space where the SVM generates the function which interpolates, to the best of its capability these series of points.

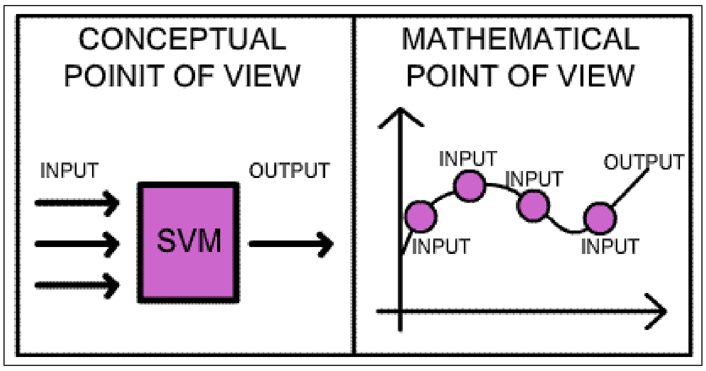


Figure 3.2: From a conceptual to a mathematical view

For conventional purposes, the group of examples that make up the SVM is called Training Set, where as the group which contains the examples used in the prediction is called Test Set. This name derives from the fact that generally, error is calculated based on the total results in the test and this in turn measures the quality of the system.

3.2 A concrete example

To better understand the function of the SVM, and some of the common errors that must be kept in mind, we present here an example of a real case which can be utilized for understanding purposes. Try to think of yourself as a stockbroker, who wants to construct an instrument which would allow him to judge the best stokes, a way to predict which to buy and which to avoid altogether. The first thing we must do is understand what the input and output mean to us. In our case input corresponds to the information we have on the stocks, like the rate of growth in the last year, the quantity of sales, the quotes on that specific stock and the market for the same. To better understand our case, consider that information is extracted from the rate of growth and the quantity sold. Our output. instead, will have a boolean value (yes/no) which indicates whether the stock is good or a bad investment. In order to make our SVM work we need to look to all the information. In the following case, suppose we know the value of these figurative companies: Zoogle, Ciat, Carmalat and Karmacotto.

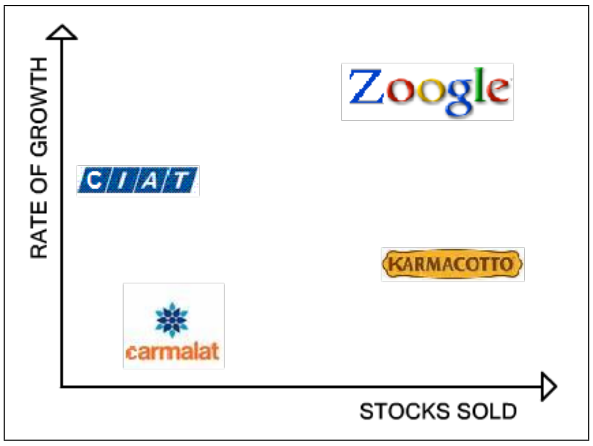


Figure 3.3: Information about stokes.

We are informed that up until now Zoogle and Ciat have been trading well whereas Carmalat and Karmacotto have not had much success. We therefore have all the information we need to make the SVM work. The results obtained are as follows:

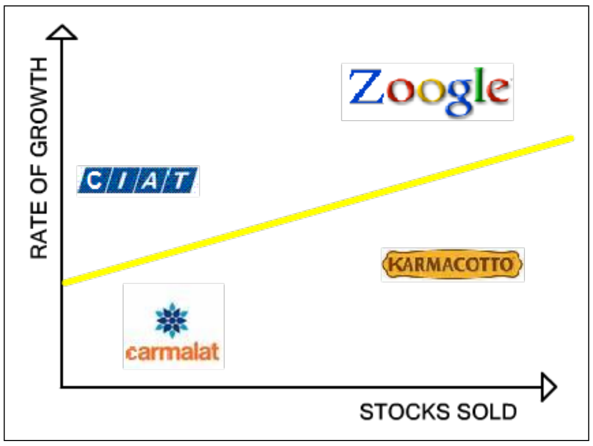


Figure 3.4: The yellow line shows the result of the training

The line corresponds to the results of the learning phase, or rather, how SVM has used up all the data until now. All the stocks that have positive results are plotted above the line, those shown below, have negative results and are considered not a good investment. For example, if we want to find the value of the stock "eBiscott" we just have a look at its position in the chart to get a good idea:

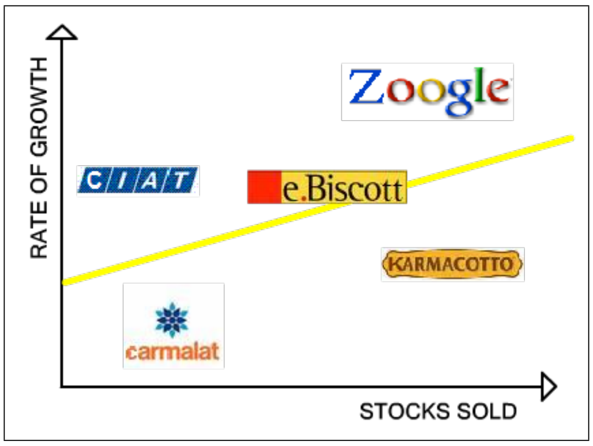


Figure 3.5: Evaluation of a new stock after the training

Because of the fact that it is positioned above the yellow line, the SVM considers it a good stock. This procedure can be applied to all the stocks in which we'd like to evaluate the performance.

3.3 A more detailed analysis

The observer may ask himself "why is the line exactly there where it is,why isn't it a little higher or lower?". In theory, all these are possible solutions:

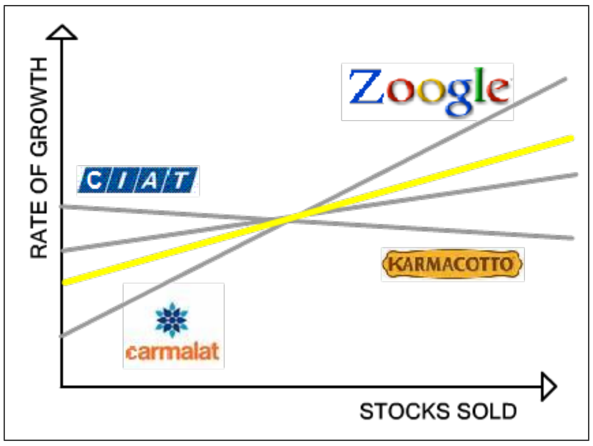


Figure 3.6: Other possible SVM solutions

Questions like these find their answer in the way the SVM calculates its results. Exact function is the one that minimizes the errors. In order to make this possible, the solution is the most distant among the examples used. The distance between solution and examples is called *margin*. In our case, it can be represented as:

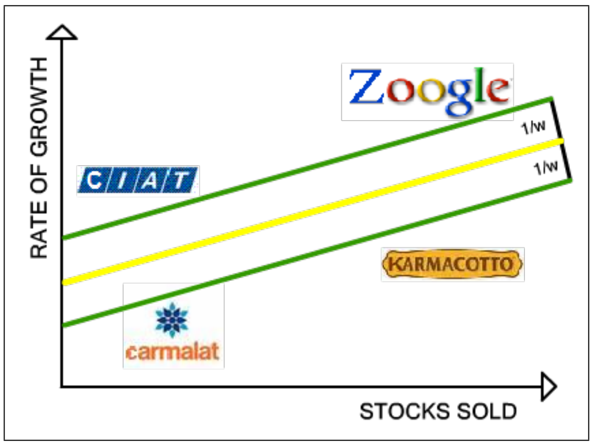


Figure 3.7: Margin evaluation

Suppose that the possible outputs are 1 and -1, they are shown on those two green lines as follows:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x} + b &= 1 \\ \mathbf{w} \cdot \mathbf{x} + b &= -1 \end{aligned} \tag{3.1}$$

The value **x** is the input, 2/w is the distance between the two categories (the margin) and we seek to maximize it is equivalent to minimize its norm:

$$\begin{aligned} &\min_{\mathbf{w}} \ \frac{1}{2}\mathbf{w}^T \cdot \mathbf{w} \\ &s.t. \ \ y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) \geq 1 \ \ i = 1...l \end{aligned} \tag{3.2}$$

This type of problems is called the Quadratic Optimization Problem. The solution to this important mathematical problem brings us to the conclusion that the result of the learning step: the yellow line can calculated by the linear combination from the elements of the Training Set, and the prediction of f(**x**) in the new sample **x** is shown like this:

$$f(\mathbf{x}) = sign(\sum_{i=1}^{l} y_i \alpha_i (\mathbf{x} \cdot \mathbf{x}_i) + b) \tag{3.3}$$

Where l is the number of elements in the Training Set, α and b make up the parameters that the SVM uses in order to construct an optimal solution (shown by the yellow line).

3.4 Nonlinear separable data

In the former example, the data were linearly separable or, in other words, it exists a line that divides correctly the samples into two sets. In reality this does not always happen. For example, let us suppose that added to our list of 4 stocks is the Coyota stock:

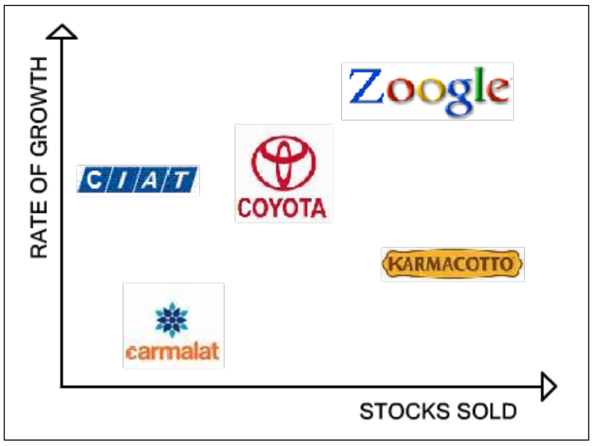


Figure 3.8: Coyota is not a good stock, but is placed near some good stocks. Now is no more possible to divide the stocks using a single line

If Coyota isn't a good stock we would not be able to find a line to divide it. This is a typical problem which is not linearly separable. In order to use the SVM in these cases we need to manipulate the errors so that they become tolerable to the SVM. If ξ is the maximum error permitted in an element, we can increase the Quadratic Programming Problem by adding to it the possibility of errors:

$$\begin{aligned} &\min_{\mathbf{w}} \ \frac{1}{2}\mathbf{w}^T \cdot \mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ &s.t. \ \ y_i(\mathbf{w}^t \cdot \mathbf{x}_i + b) \geq 1 - \xi, \quad \xi_i \geq 0 \ \ i = 1...l \end{aligned} \tag{3.4}$$

ξ and C are parameters which define the limit of maximal tolerance. Finding the right value is very complicated and there is a vast literature on how to choose the best ones. Generally, the most used technique is to find them by trial and error. In the appendix however, are some recommendations on how to select them.

3.5 More generic Support Vector Machines

Up to now, we have only paralleled the SVM to a line which divides the input space in two parts. This would solve all the problems that have a solution, represented by a simple line or, generally, by a hyperplane that is dimensionally equal to the number of components which is made up each example in the Training Set. Many other problems require more elaborate and complex solutions. For example, in the case of the Coyota stock, we can visualize the probable solution like this:

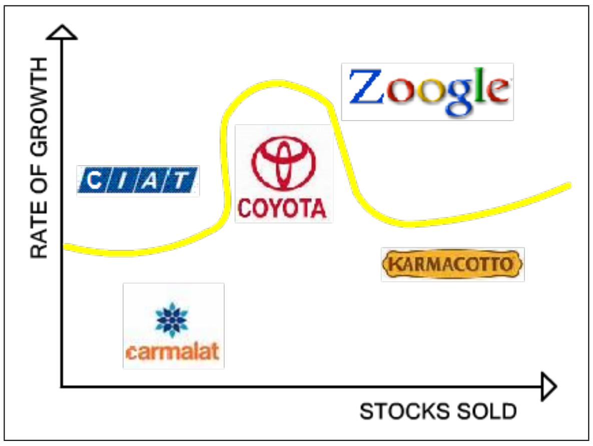


Figure 3.9: A solution at the Coyota problem

But how can one obtain different functions from a line? How can we introduce new functions without change the SVM completely? To find out the solution to this problem we must analyze the formula of the solution of the SVM problem:

$$f(\mathbf{x}) = sign(\sum_{i=1}^{l} y_i \alpha_i (\mathbf{x} \cdot \mathbf{x}_i) + b) \tag{3.5}$$

Each sample is multiplied by the value of the examples trained. If, on the other hand, instead of a simple multiplication other more complicated operations come into play the end result changes. If we substitute $(\mathbf{x} \cdot \mathbf{x}_i)$ in the expression with another function, called *Kernel Function*, that allows growth of the SVM solution, which starts invading other space, and this ”other space” is called the *Features Space*. This allows us to change the information from one linear space to another one. This permits us to better classify the examples.

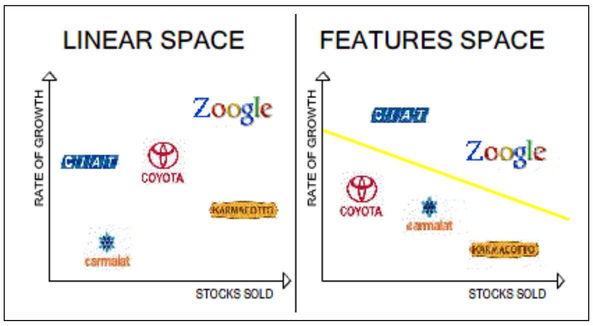


Figure 3.10: From linear to features space

When the results of the SVM expand into an other space is not linear anymore, it becomes much more complex, a not linear function.

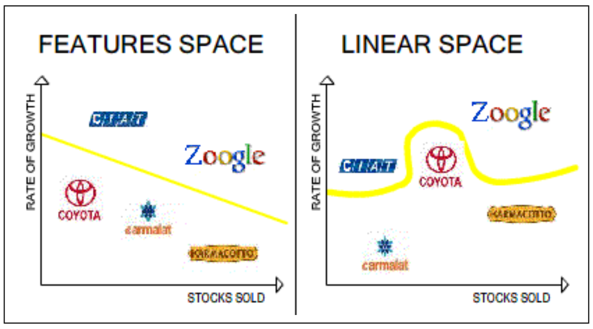


Figure 3.11: From features space back to linear space

This characteristic of the SVM increases its usability but, at the same time, causes new problems. Now, other parameters have been added to the equation: the value of C, the type of kernel to use and optionally, the kernel parameters. For example, one of the most used kernels is the *gaussian kernel*, which permits us to find solutions to problems such us this one:

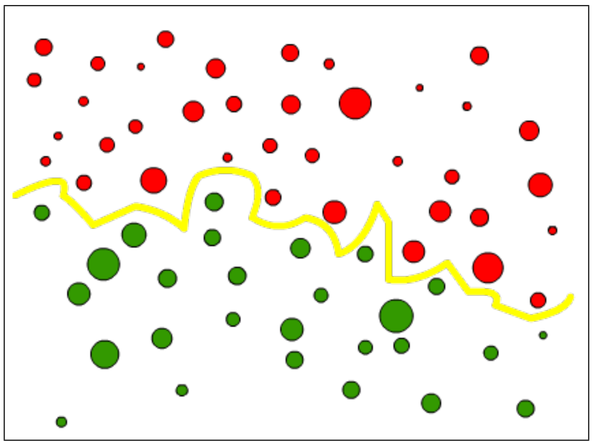


Figure 3.12: A difficult problem solved using a gaussian kernel

This kernel, however, requires as its parameter σ, which, if wrongly chosen could also make wrong classifications, like provided by this example:

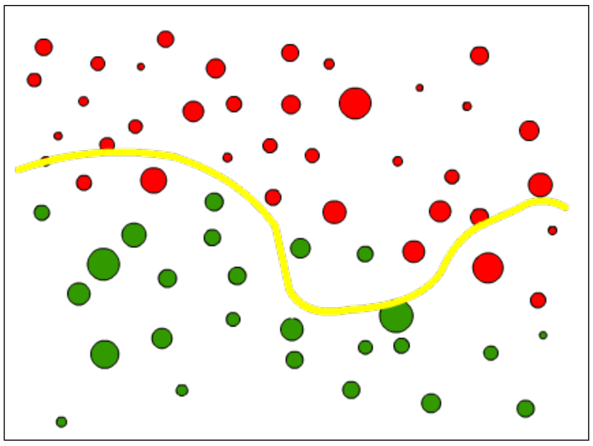


Figure 3.13: An example of a bad parameters setting

It is also possible to invent personalized kernel functions, specific for the type of problems we wish to solve. The characteristics which this function must have are as follow [5]:

- must accept as its parameters $(\mathbf{x}, \mathbf{x_i})$ (as well as other eventual parameters of the kernel)
- must be defined semi positive
- $k_{xy} = k_{yx}$

In any case many kernel functions are often utilized, one has only to read a few of the many articles written on this topic. Anyway, whatever the problem you wish to solve, one needs to realize the abundance of parameters which can be utilized. Each one must be set correctly or the products will result in non satisfactory and insufficient.

3.6 SVR - Support Vector Machines for Regression

Up to now, support vector machines have been concerned with classification, or about the problem of dividing the elements into different types. Sometimes, however it is a lot more useful to have values that are less rigid with more possible results. After the realization of SVM's which are used to classify, other machines where devised to resolve the problem of regression, where the solution generated is a real number. The construction of an instrument which could see the success of a stock, in a way to better understand when it is the most convenient time to sell or to buy more, so to maximize your gain. Let's image that after having utilized the SVM for viewing which stocks are good and which ones bad, we decide that the Zoogle stock is the one that seems to be the most probably trustworthy. Now, we must decide when is the right moment to buy or sell the stock. Every good broker knows that the stock market has four main phases: growth, expansion, slowdown and decline which change unexpectedly in a continuous cycle. To be able to come out with maximal gain, ideally, one should buy stock when its in its declining phase (at its lowest value) and sell it in turn in its expansion phase (at its highest value).

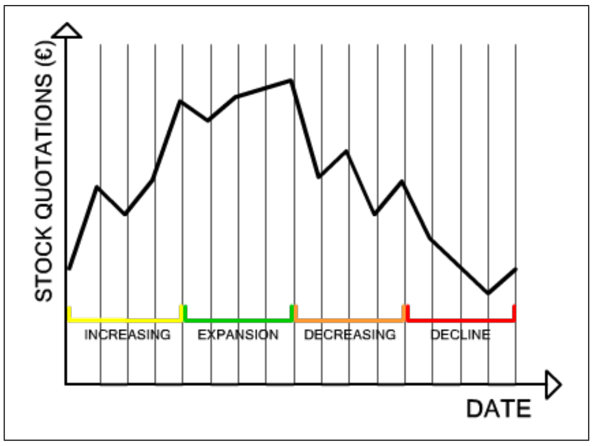


Figure 3.14: Stock market phases

It is obviously not possible to know when its the best moment to buy or sell stocks, however being that each stock behaves in a cyclical manner, it is therefore a little predictable because they behave in a oscillating manner. The examples used in the training (dates, quotations of the stock on those date) can be extracted from many websites that have as a function to analyze the financial market. Here's an example:

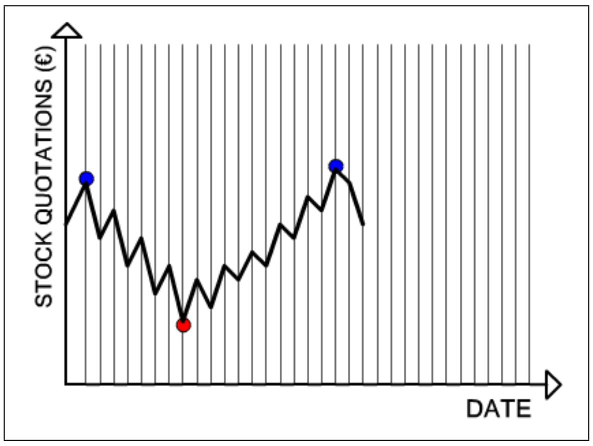


Figure 3.15: Stock quotations example

The blue points represents the best moments to buy (in the declining phase) while the red points represents the best moment to sell (in the expansion phase) once the SVM is working it is possible to predict how the stock will act in the following days. Once you have the prediction its easy to tell when the best moments to buy and sell are. Obviously the solution obtained isn't a mirror of reality because the definitive value of a stock is given from the brokers choices. However, it is a useful tool to use as support in the stock brokers choices.

3.7 How the SVR Works

The SVR uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance ϵ is set in approximation to the SVM which would have already requested from the problem. For example, in the case of our stocks, and because they are all expressed in euro's, there will be a leeway of 0.01 in order to be able consider the eurocent. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

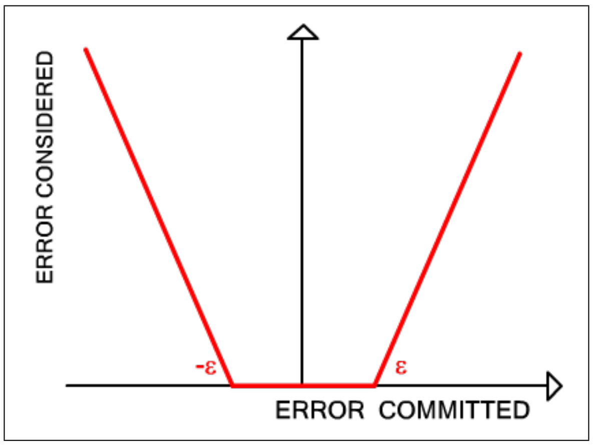


Figure 3.16: The error function

The figure shows how the error of SVR is calculated. Up until the threshold ϵ, the error is considered 0, after the error it becomes calculated as ”error-epsilon”. The solution to the problem is another, from one line it becomes a ”tube”, tolerant to errors.

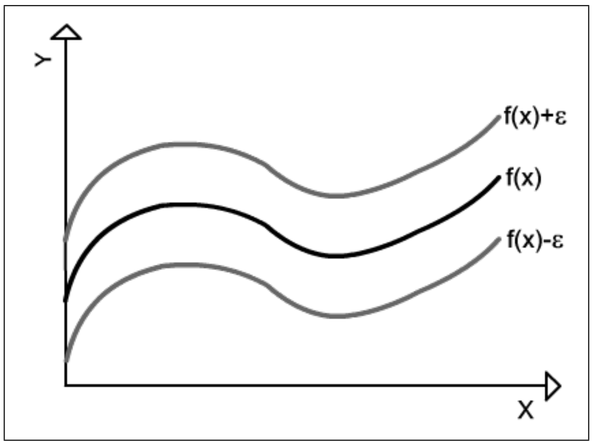


Figure 3.17: The SVR tube

Coming back to the mathematical point of view, the quadratic optimization problem becomes:

$$\min_{\mathbf{w}} \; \frac{1}{2}\,\mathbf{w}^T \cdot \mathbf{w}$$

$$s.t. \begin{cases} y_i - (\mathbf{w}^T \cdot \phi(\mathbf{x}) + b) \leq \epsilon \\ \\ (\mathbf{w}^T \cdot \phi(\mathbf{x}) + b) - y_i \leq \epsilon \end{cases} \tag{3.6}$$

Where $\phi(\mathbf{x})$ is the kernel function seen previously, $\mathbf{w}$ is the margin and the couple $(\mathbf{x_i}, \mathbf{y_i})$ the Training Set. Other than this, as in classification, we can add a bound in order to set the tolerance on errors number that can be committed:

$$\min_{\mathbf{w},\, b} \; \frac{1}{2}\,\mathbf{w}^T \cdot \mathbf{w} + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

$$s.t. \begin{cases} y_i - (\mathbf{w}^T \cdot \phi(\mathbf{x}) + b) \leq \epsilon + \xi_i \\ \\ (\mathbf{w}^T \cdot \phi(\mathbf{x}) + b) - y_i \leq \epsilon + \xi_i^* \\ \\ \xi_i, \xi_i^* \geq 0, \;\; i = 1..l \end{cases} \tag{3.7}$$

The principle is similar respect to SVM and, once trained, the SVR will generate predictions using the following formula:

$$f(\mathbf{x}) \equiv \sum_{i=1}^{l} \theta_i \phi(\mathbf{x}, \mathbf{x}_i) + b \tag{3.8}$$

3.8 The practical case

Let us go back to the practical case. In this case, our motive is to find the probable door for a useful stock in the future. Let's suppose that our Training Set is made up of pairs (day, market quotes) which correspond to this plot:

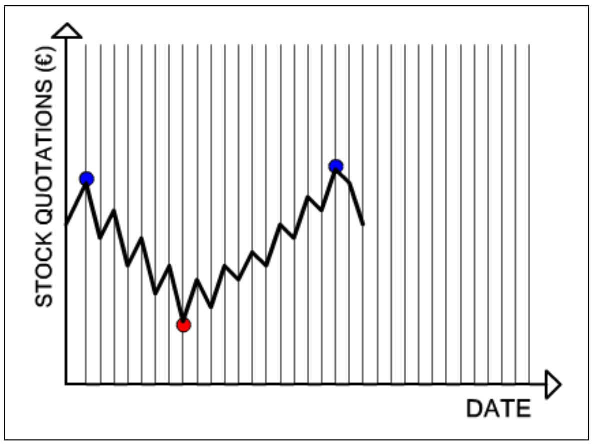


Figure 3.18: The stock quotations

To make the SVR work, let's make the tolerance margin at 0.01 over the final result. We could also choose the gaussian kernel, which would permit a certain flexibility in the solution. The C parameter, on the other hand, is chosen experimentally over information that give us the best solution. The result of this would be:

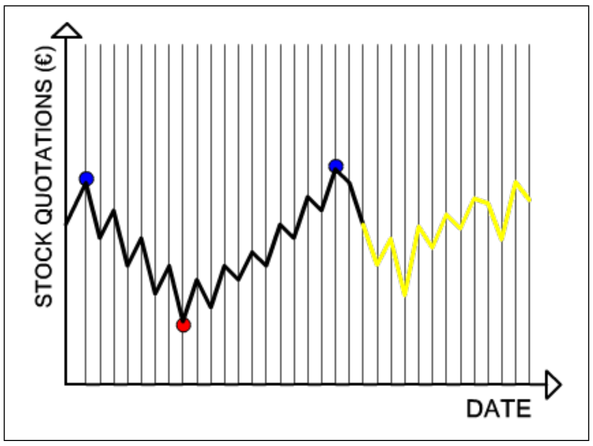


Figure 3.19: The SVR prediction

The yellow line corresponds to prevision of the SVR in the following days. In the experiment, our motive about all, is gain: buying when stock is at its lowest and selling when is at its highest give us the best profit:

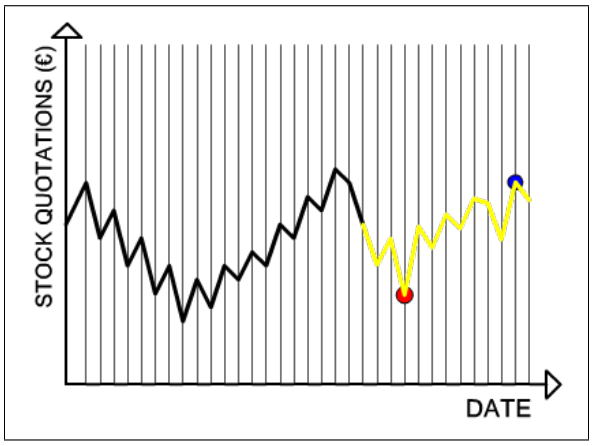


Figure 3.20: When buy and sell the stocks

The two points made evident the days in which it is advised to buy and sell from the information we have. Naturally, as in the previous example, this is only an estimation, it doesn't guarantee us that the results are 100% correct. For example, combining the SVM and the SVR, we can create a "little broker" who has the knowledge to decide whether a particular stock will be a good or bad investment, and based upon certain predictions, when to sell and buy the stock.

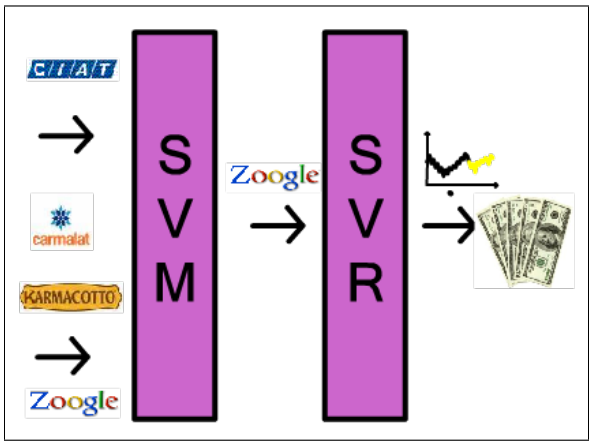


Figure 3.21: A virtual stock broker build using SVM and SVR

Chapter 4

KKT Conditions, Incremental Learning and Samples Moving

This is the mathematical and most complex part of the thesis. Here you can find how you can build the online version of support vector machines algorithm starting from its definition.

4.1 Problem Description

The main purpose of support vector machines is to find a function $f(\mathbf{x})$ that has deviated away from all the training data. At the same time, this function should be as flat as possible, to prevent overfitting. We can write this problem as a convex optimization problem [5] [1]:

$$\min_{\mathbf{w}} \ \frac{1}{2}\,\mathbf{w}^T \cdot \mathbf{w}$$

$$s.t. \begin{cases} y_i - (\mathbf{w}^T \cdot \phi(\mathbf{x}_i) + b) \leq \epsilon \\ \\ (\mathbf{w}^T \cdot \phi(\mathbf{x}_i) + b) - y_i \leq \epsilon \end{cases} \tag{4.1}$$

The model presented is correct only if we assume that the problem is feasible. If we want to allow some errors we should introduce some slack-variables that enlarge the tolerance of the machine:

$$\min_{\mathbf{w},\,b} \ \frac{1}{2}\,\mathbf{w}^T \cdot \mathbf{w} + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

$$s.t. \begin{cases} y_i - (\mathbf{w}^T \cdot \phi(\mathbf{x}_i) + b) \leq \epsilon + \xi_i \\ (\mathbf{w}^T \cdot \phi(\mathbf{x}_i) + b) - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0, \ \ i = 1..l \end{cases} \tag{4.2}$$

The constant C determines the trade-off between the flatness of the function and the amount of larger deviations of tolerance. Because this is a minimization problem, we can set all the constraints ≥ 0:

$$\min_{\mathbf{w},\,b} \ P = \ \frac{1}{2}\,\mathbf{w}^T \cdot \mathbf{w} + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

$$s.t. \begin{cases} -y_i + (\mathbf{w}^T \cdot \phi(\mathbf{x}_i) + b) + \epsilon + \xi_i \geq 0 \\ y_i - (\mathbf{w}^T \cdot \phi(\mathbf{x}_i) + b) + \epsilon + \xi_i^* \geq 0 \\ \xi_i, \xi_i^* \geq 0, \ \ i = 1..l \end{cases} \tag{4.3}$$

4.2 Optimizing the problem

From the minimization problem, we can calculate a Lagrange function which includes all the constraints, introducing some Lagrange multipliers:

$$\begin{aligned} L_P = \frac{1}{2}\ \mathbf{w}^T \cdot \mathbf{w} + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)\ + \\ -\sum_{i=1}^{l}(\eta_i\xi_i + \eta_i^*\xi_i^*)\ + \\ -\sum_{i=1}^{l}\alpha_i(\epsilon + \xi_i + y_i - \mathbf{w}^T \cdot \phi(\mathbf{x}_i) - b)\ + \\ -\sum_{i=1}^{l}\alpha_i^*(\epsilon + \xi_i^* - y_i + \mathbf{w}^T \cdot \phi(\mathbf{x}_i) + b) \end{aligned} \tag{4.4}$$

$$s.t. \quad \alpha_i, \alpha_i^*, \eta_i, \eta_i^* \geq 0,\ \ i = 1..l$$

(The constraints are inserted with minus signs because inequalities are in the form "≥")

The partial derivatives of the Lagrangian considering the variables $(\mathbf{w}, \mathbf{b}, \xi, \xi^*)$ have to vanish for optimality:

$$\begin{aligned}
&\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 \\
&\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l}\phi(\mathbf{x}_i)(\alpha_i - \alpha_i^*) = 0 \quad \Longrightarrow \quad \mathbf{w} = \sum_{i=1}^{l}\phi(\mathbf{x}_i)(\alpha_i - \alpha_i^*) \\
&\frac{\partial L_P}{\partial \xi_i} = C - \eta_i - \alpha_i = 0 \quad \Longrightarrow \quad \eta_i = C - \alpha_i, \quad \alpha_i \in [0, C] \\
&\frac{\partial L_P}{\partial \xi_i^*} = C - \eta_i^* - \alpha_i^* = 0 \quad \Longrightarrow \quad \eta_i^* = C - \alpha_i^*, \quad \alpha_i^* \in [0, C] \\
&\frac{\partial L_P}{\partial \eta_i} = \sum_{i=1}^{l}\xi_i = 0 \\
&\frac{\partial L_P}{\partial \eta_i^*} = \sum_{i=1}^{l}\xi_i^* = 0
\end{aligned} \tag{4.5}$$

Substituting the new definitions of $\mathbf{w}, \eta, \eta^*$ in the Lagrangian:

$$\begin{aligned}
&-\frac{1}{2}\ \sum_{i=1}^{l}\sum_{j=1}^{l}\phi(\mathbf{x}_i)(\alpha_i-\alpha_i^*)\cdot\phi(\mathbf{x}_j)(\alpha_j-\alpha_j^*)\ +\\
&-\sum_{i=1}^{l}((C-\alpha_i)\xi_i+(C-\alpha_i^*)\xi_i^*)\ +\\
&-\sum_{i=1}^{l}\alpha_i(-y_i+(\sum_{j=1}^{l}(\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)(\alpha_j-\alpha_j^*))+b)+\epsilon+\xi_i)\ +\\
&-\sum_{i=1}^{l}\alpha_i^*(y_i-(\sum_{j=1}^{l}(\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)(\alpha_j-\alpha_j^*))+b)+\epsilon+\xi_i^*)
\end{aligned} \tag{4.6}$$

That can also be written:

$$\begin{aligned}
&-\frac{1}{2}\ \sum_{i=1}^{l}\sum_{j=1}^{l}\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\ +\\
&-\sum_{i=1}^{l}(C-\alpha_i)\xi_i\quad-\sum_{i=1}^{l}(C-\alpha_i^*)\xi_i^*\ +\\
&-\sum_{i=1}^{l}\sum_{j=1}^{l}\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)\,\alpha_i(\alpha_j-\alpha_j^*)\ +\\
&+\sum_{i=1}^{l}\sum_{j=1}^{l}\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)\,\alpha_i^*(\alpha_j-\alpha_j^*)\ +\\
&+\sum_{i=1}^{l}\alpha_i y_i\quad-\sum_{i=1}^{l}\alpha_i^* y_i\ +\\
&-\sum_{i=1}^{l}\alpha_i\epsilon\quad-\sum_{i=1}^{l}\alpha_i^*\epsilon\ +\\
&-\sum_{i=1}^{l}\alpha_i\xi_i\quad-\sum_{i=1}^{l}\alpha_i^*\xi_i^*
\end{aligned} \tag{4.7}$$

Some monomials can be merged together forming:

$$\begin{aligned}
&-\frac{1}{2}\ \sum_{i=1}^{l}\sum_{j=1}^{l}\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\ +\\
&+\sum_{i=1}^{l}y_i\,(\alpha_i-\alpha_i^*)\ +\\
&-\sum_{i=1}^{l}\epsilon\,(\alpha_i+\alpha_i^*)\ +\\
&-\sum_{i=1}^{l}(C\xi_i-\alpha_i\xi_i+\alpha_i\xi_i)\ +\\
&-\sum_{i=1}^{l}(C\xi_i^*-\alpha_i^*\xi_i^*+\alpha_i^*\xi_i^*)
\end{aligned} \tag{4.8}$$

And deleting some unnecessary parts:

$$\begin{aligned}
&-\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\ + \\
&+\sum_{i=1}^{l}y_i\,(\alpha_i-\alpha_i^*)\ + \\
&-\sum_{i=1}^{l}\epsilon\,(\alpha_i+\alpha_i^*)\ + \\
&-C\sum_{i=1}^{l}(\xi_i+\xi_i^*)
\end{aligned} \tag{4.9}$$

Finally also the C parameter can be deleted, using the derivative constraints

$$\sum_{i=1}^{l}\xi_i=0\ ,\quad \sum_{i=1}^{l}\xi_i^*=0 \tag{4.10}$$

we obtain:

$$\begin{aligned}
&-\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\ + \\
&+\sum_{i=1}^{l}y_i\,(\alpha_i-\alpha_i^*)\ + \\
&-\sum_{i=1}^{l}\epsilon\,(\alpha_i+\alpha_i^*)
\end{aligned} \tag{4.11}$$

To increase the reading facility, kernel function can be defined as:

$$K(\mathbf{x}_i,\mathbf{x}_j)=\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j)=Q_{ij} \tag{4.12}$$

The matrix Q contains the values of kernel function and it is called *kernel matrix*. It is useful to save kernel function values in this way for computational reasons.

In the next paragraph will be shown how, combining the results found until now, we can arrive to the definition of KKT conditions.

4.3 Another optimization problem

After many math passages, minimizing α, α^* values, there is another dual optimization problem to solve:

$$\begin{aligned}\min_{\alpha,\,\alpha^*} \; D = \; & \frac{1}{2} \sum_{i=1}^{l}\sum_{j=1}^{l} Q_{ij}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \; + \\ & - \sum_{i=1}^{l} y_i\,(\alpha_i - \alpha_i^*) \; + \\ & + \sum_{i=1}^{l} \epsilon\,(\alpha_i + \alpha_i^*)\end{aligned} \tag{4.13}$$

$$s.t. \begin{cases} \alpha_i, \alpha_i^* \in [0, C], \;\; i = 1..l \\ \\ \displaystyle\sum_{i=1}^{l} (\alpha_i - \alpha_i^*) = 0 \end{cases}$$

As before, we can compute another Lagrange function that includes all the constraints, introducing other Lagrange multipliers:

$$\begin{aligned} L_D = \; & \frac{1}{2} \sum_{i=1}^{l}\sum_{j=1}^{l} Q_{ij}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \; + \\ & - \sum_{i=1}^{l} y_i\,(\alpha_i - \alpha_i^*) \; + \\ & + \sum_{i=1}^{l} \epsilon\,(\alpha_i + \alpha_i^*) \; + \\ & - \sum_{i=1}^{l} (\delta_i\alpha_i + \delta_i^*\alpha_i^*) \; + \\ & - \sum_{i=1}^{l} [u_i(\alpha_i - C) + u_i^*(\alpha_i^* - C)] \; + \\ & + \zeta \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \end{aligned} \tag{4.14}$$

$$s.t. \quad \delta_i, \delta_i^*, u_i, u_i^*, \zeta \geq 0, \;\; i = 1..l$$

Computing partial derivatives of the Lagrangian:

$$\frac{\partial L_D}{\partial \alpha_i} = \frac{1}{2}\sum_{j=1}^{l} Q_{ij}(\alpha_j - \alpha_j^*) + \epsilon - y_i - \delta_i + u_i + \zeta = 0$$

$$\frac{\partial L_D}{\partial \alpha_i^*} = -\frac{1}{2}\sum_{j=1}^{l} Q_{ij}(\alpha_j - \alpha_j^*) + \epsilon + y_i - \delta_i + u_i - \zeta = 0 \tag{4.15}$$

$$\frac{\partial L_P}{\partial \zeta} = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0$$

In order to increase readability, we can replace ζ with b:

$$\zeta = b \tag{4.16}$$

Like exposed above, this is an optimization problem with a convex domain. The sufficient conditions for a point to be an optimum are (Kuhn-Tucker Theorem):

$$\begin{aligned}
&\alpha_i \in [0, C], \quad \alpha_i \,(\sum_{j=1}^{l} Q_{ij}(\alpha_j - \alpha_j^*) + b - y_i + \epsilon - \delta_i + u_i) = 0 \\
&\alpha_i^* \in [0, C], \quad \alpha_i^* \,(-\sum_{j=1}^{l} Q_{ij}(\alpha_j - \alpha_j^*) - b + y_i + \epsilon - \delta_i^* + u_i^*) = 0 \\
&\delta_i \geq 0, \qquad \delta_i \alpha_i = 0 \\
&\delta_i^* \geq 0, \qquad \delta_i^* \alpha_i^* = 0 \\
&u_i \geq 0, \qquad u_i(\alpha_i - C) = 0 \\
&u_i^* \geq 0, \qquad u_i^*(\alpha_i^* - C) = 0
\end{aligned} \tag{4.17}$$

Function estimated can be written as:

$$f(\mathbf{x}_i) = \sum_{j=1}^{l} Q_{ij}(\alpha_j - \alpha_j^*) + b \tag{4.18}$$

And margin function is defined as:

$$h(\mathbf{x}_i) \equiv f(\mathbf{x}_i) + b - y_i \tag{4.19}$$

And replacing it in the system of equalities:

$$\begin{aligned}
&\alpha_i \in [0, C], && \alpha_i \left(h(\mathbf{x}_i) + \epsilon - \delta_i + u_i\right) = 0 \\
&\alpha_i^* \in [0, C], && \alpha_i^* \left(-h(\mathbf{x}_i) + \epsilon - \delta_i + u_i\right) = 0 \\
&\delta_i \geq 0, && \delta_i \alpha_i = 0 \\
&\delta_i^* \geq 0, && \delta_i^* \alpha_i^* = 0 \\
&u_i \geq 0, && u_i(\alpha_i - C) = 0 \\
&u_i^* \geq 0, && u_i^*(\alpha_i^* - C) = 0
\end{aligned} \tag{4.20}$$

All we need to find is the relation of $h(\mathbf{x})$ and ϵ at the changing of α:

$$\begin{aligned}
&\alpha_i = 0 \\
&\quad\Longrightarrow\quad \alpha_i^* > 0, \quad \delta_i \geq 0, \quad u_i = 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) + \epsilon - \delta_i = 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) = -\epsilon + \delta_i \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) \in [-\epsilon, 0] \\
\\
&\alpha_i^* = 0 \\
&\quad\Longrightarrow\quad \alpha_i > 0, \quad \delta_i^* \geq 0, \quad u_i^* = 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad -h(\mathbf{x}_i) + \epsilon - \delta_i^* = 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) = +\epsilon - \delta_i^* \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) \in [0, +\epsilon] \\
\\
&0 < \alpha_i < C \\
&\quad\Longrightarrow\quad \alpha_i^* = 0, \quad \delta_i = 0, \quad u_i = 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) + \epsilon = 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) = -\epsilon \\
\\
&0 < \alpha_i^* < C \\
&\quad\Longrightarrow\quad \alpha_i = 0, \quad \delta_i^* = 0, \quad u_i^* = 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) - \epsilon = 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) = +\epsilon \\
\\
&\alpha_i = C \\
&\quad\Longrightarrow\quad \alpha_i^* = 0, \quad \delta_i = 0, \quad u_i \geq 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) + \epsilon + u_i = 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) = -\epsilon - u_i \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) \leq -\epsilon \\
\\
&\alpha_i^* = C \\
&\quad\Longrightarrow\quad \alpha_i = 0, \quad \delta_i^* = 0, \quad u_i^* \geq 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad -h(\mathbf{x}_i) + \epsilon + u_i^* = 0 \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) = +\epsilon + u_i^* \quad\Longrightarrow \\
&\quad\Longrightarrow\quad h(\mathbf{x}_i) \geq +\epsilon
\end{aligned} \tag{4.21}$$

4.4 KKT conditions definition

For simplicity, θ can be defined as the difference between α and α^*:

$$\theta_i \equiv \alpha_i - \alpha_i^* \tag{4.22}$$

Inequalities can be rewritten as a system of conditions, called Karush-Kuhn-Tucker (KKT) Conditions [6]:

$$\begin{cases} h(\mathbf{x}_i) \geq +\epsilon & \theta_i = -C \\ h(\mathbf{x}_i) = +\epsilon & \theta_i \in [-C, 0] \\ h(\mathbf{x}_i) \in [-\epsilon, +\epsilon] & \theta_i = 0 \\ h(\mathbf{x}_i) = -\epsilon & \theta_i \in [0, C] \\ h(\mathbf{x}_i) \leq -\epsilon & \theta_i = +C \end{cases} \tag{4.23}$$

The Online Support Vector Regression algorithm main purpose is to verify these conditions after that a new sample is added.

4.5 Support, Error and Remaining Sets

Now, using these conditions the samples can be divided into three sets:

$$\begin{array}{ll} \textsc{Support Set} & S = \{i|\ (\theta_i \in [0, +C] \ \wedge\ h(\mathbf{x}_i) = -\epsilon) \ \vee \\ & \qquad\qquad (\theta i \in [-C, 0] \ \wedge\ h(\mathbf{x}_i) = +\epsilon)\} \\ & \\ \textsc{Error Set} & E = \{i|\ (\theta_i = -C \ \wedge\ h(\mathbf{x}_i) \geq +\epsilon) \ \vee \\ & \qquad\qquad (\theta_i = +C \ \wedge\ h(\mathbf{x}_i) \leq -\epsilon)\} \\ & \\ \textsc{Remaining Set} & R = \{i|\ \theta_i = 0 \ \wedge\ |h(\mathbf{x}_i)| \leq \epsilon\} \end{array} \tag{4.24}$$

The goal is to find a way to add a new sample to one of the three sets maintaining KKT conditions consistent.

4.6 Adding a new sample

The margin function can be defined as:

$$h(\mathbf{x}_i) = f(\mathbf{x}_i) + b - y_i = \sum_{j=1}^{l} Q_{ij}\theta_j + b - y_i \tag{4.25}$$

Adding a new sample c, the margin function changes to:

$$h(\mathbf{x}_i) = \sum_{j=1}^{l} Q_{ij}\theta'_j + Q_{ic}\theta'_c + b' - y_i \tag{4.26}$$

The variation of the margin can be easily computed:

$$\begin{aligned} &\Delta\theta_i = \theta'_i - \theta_i \\ &\Delta b = b' - b \\ &\Delta h(\mathbf{x}_i) = \sum_{j=1}^{l} Q_{ij}\Delta\theta_j + Q_{ic}\Delta\theta_c + \Delta b \end{aligned} \tag{4.27}$$

The sum of the weights is zero, and this property can be used:

$$\begin{aligned} &\sum_{i=1}^{l} \theta_i = 0 \quad \text{and} \quad \theta_c + \sum_{i=1}^{l} \theta'_i = 0 \quad \Longrightarrow \\ &\Longrightarrow \quad \sum_{i=1}^{l} \Delta\theta_i + \Delta\theta_c = 0 \quad \Longrightarrow \quad \sum_{i=1}^{l} \Delta\theta_i = -\Delta\theta_c \end{aligned} \tag{4.28}$$

For the properties of the three sets, only support set samples can change θ_j, so only they contribute to the new solution:

$$\begin{aligned} &\sum_{j \in S} Q_{ij}\Delta\theta_j + \Delta b = -Q_{ic}\Delta\theta_c \quad \text{where} \quad i \in \textsc{SupportSet} \\ &\sum_{j \in S} \Delta\theta_j = -\theta_c \end{aligned} \tag{4.29}$$

Defining the support set as a list of elements:

$$S = \{s_1, s_2, \cdots, s_{ls}\} \tag{4.30}$$

The equations above can be rewritten in an equivalent matrix form:

$$\begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q_{s_1 s_1} & \cdots & Q_{s_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q_{s_{l_s} s_1} & \cdots & Q_{s_{l_s} s_{l_s}} \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta\theta_{s_1} \\ \vdots \\ \Delta\theta_{s_{l_s}} \end{bmatrix} = - \begin{bmatrix} 1 \\ Q_{s_1 c} \\ \vdots \\ Q_{s_{l_s} c} \end{bmatrix} \Delta\theta_c \tag{4.31}$$

It's interesting to discover the values of the variations of θ and b:

$$\begin{bmatrix} \Delta b \\ \Delta\theta_{s_1} \\ \vdots \\ \Delta\theta_{s_{l_s}} \end{bmatrix} = - \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q_{s_1 s_1} & \cdots & Q_{s_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q_{s_{l_s} s_1} & \cdots & Q_{s_{l_s} s_{l_s}} \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ Q_{s_1 c} \\ \vdots \\ Q_{s_{l_s} c} \end{bmatrix} \Delta\theta_c \tag{4.32}$$

For simplicity, matrix R and the vector β can be defined as:

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q_{s_1 s_1} & \cdots & Q_{s_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q_{s_{l_s} s_1} & \cdots & Q_{s_{l_s} s_{l_s}} \end{bmatrix}^{-1} \tag{4.33}$$

$$\beta = \begin{bmatrix} \beta_b \\ \beta_{s_1} \\ \vdots \\ \beta_{s_{l_s}} \end{bmatrix} = -\mathbf{R} \begin{bmatrix} 1 \\ Q_{s_1 c} \\ \vdots \\ Q_{s_{l_s} c} \end{bmatrix} \tag{4.34}$$

The solution can be rewritten as follow. Updating the θ_i values, the support set samples will be consistent. The h values for support samples don't need updates, because they are always $|\epsilon|$.

$$\begin{bmatrix} \Delta b \\ \Delta\theta_{s_1} \\ \vdots \\ \Delta\theta_{s_{l_s}} \end{bmatrix} = \beta\Delta\theta_c = \begin{bmatrix} \beta_b \\ \beta_{s_1} \\ \vdots \\ \beta_{s_{l_s}} \end{bmatrix} \Delta\theta_c \tag{4.35}$$

For error and remaining samples, the situation is opposite. They do not change the value, but they change h. Not Support Set is defined as:

$$N = E \cup R = \{n_1, n_2, \cdots, n_{l_n}\} \tag{4.36}$$

And rewriting the variation of the h formula in a matrix notation:

$$\begin{bmatrix} \Delta h(\mathbf{x}_{n_1}) \\ \vdots \\ \Delta h(\mathbf{x}_{n_{l_n}}) \end{bmatrix} = \begin{bmatrix} \Delta Q_{n_1 c} \\ \vdots \\ \Delta Q_{n_{l_n} c} \end{bmatrix} \Delta\theta_c + \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q_{n_1 s_1} & \cdots & Q_{n_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q_{n_{l_n} s_1} & \cdots & Q_{n_{l_n} s_{l_s}} \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta\theta_{s_1} \\ \vdots \\ \Delta\theta_{s_{l_s}} \end{bmatrix} \tag{4.37}$$

Replacing the variations of θ and b with the results obtained in the equation (4.13):

$$\begin{bmatrix} \Delta h(\mathbf{x}_{n_1}) \\ \vdots \\ \Delta h(\mathbf{x}_{n_{l_n}}) \end{bmatrix} = \begin{bmatrix} \Delta Q_{n_1 c} \\ \vdots \\ \Delta Q_{n_{l_n} c} \end{bmatrix} \Delta\theta_c + \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q_{n_1 s_1} & \cdots & Q_{n_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q_{n_{l_n} s_1} & \cdots & Q_{n_{l_n} s_{l_s}} \end{bmatrix} \beta\Delta\theta_c \tag{4.38}$$

Now, γ can be defined as:

$$\gamma = \begin{bmatrix} \Delta Q_{n_1 c} \\ \vdots \\ \Delta Q_{n_{l_n} c} \end{bmatrix} + \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q_{n_1 s_1} & \cdots & Q_{n_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q_{n_{l_n} s_1} & \cdots & Q_{n_{l_n} s_{l_s}} \end{bmatrix} \beta \tag{4.39}$$

And rewriting the formula with new notations:

$$\begin{bmatrix} \Delta h(\mathbf{x}_{n_1}) \\ \vdots \\ \Delta h(\mathbf{x}_{n_{l_n}}) \end{bmatrix} = \gamma\Delta\theta_c \tag{4.40}$$

In this way, $\Delta\theta_i$ and Δb values can be updated by computing β, and $\Delta(\mathbf{x}_i)$ values can be updated by computing γ.

4.7 Empty SupportSet

In the special case that SupportSet is empty, values should be updated differently. $\Delta\theta_c$ is no more distributed in the SupportSet, now empty, but it is fully added to b:

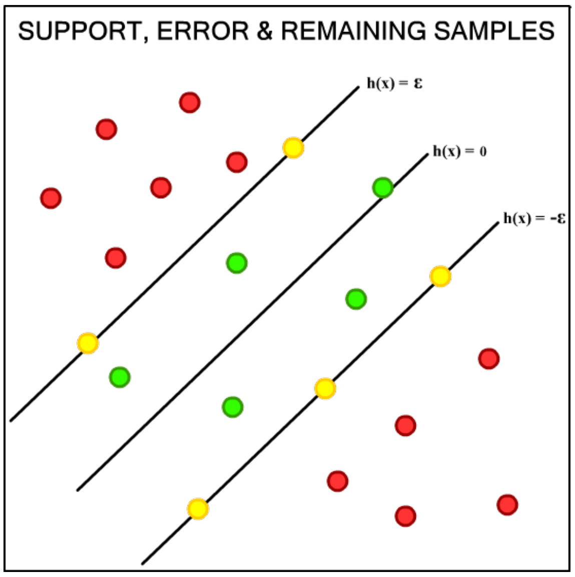
$$\begin{aligned} &\Delta b = \Delta\theta_c \\ &\Delta h(\mathbf{x}_i) = \Delta b, \quad i = 1..l \end{aligned} \tag{4.41}$$

4.8 Samples Moving

Up until now, we defined a way to update all three sets values of $\Delta\theta_c$ maintaining the KKT conditions consistent.

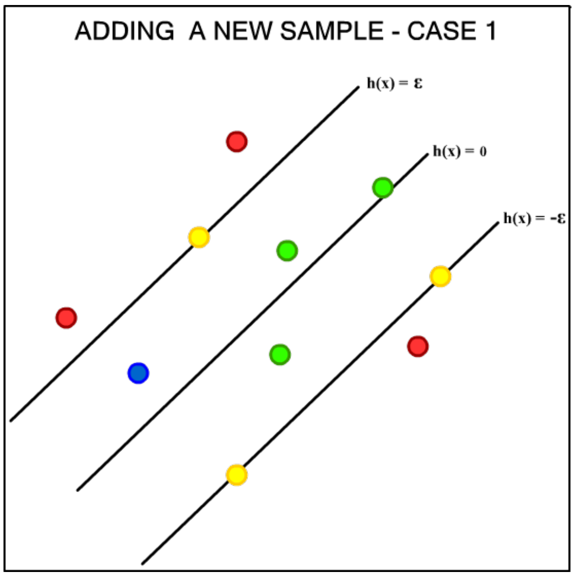
Now, the problem is to identify the right value of $\Delta\theta_c$ in all possible situations that can happen.

A simple representation of support set, error set and remaining set in relation to $h(\mathbf{x})$ as follows:



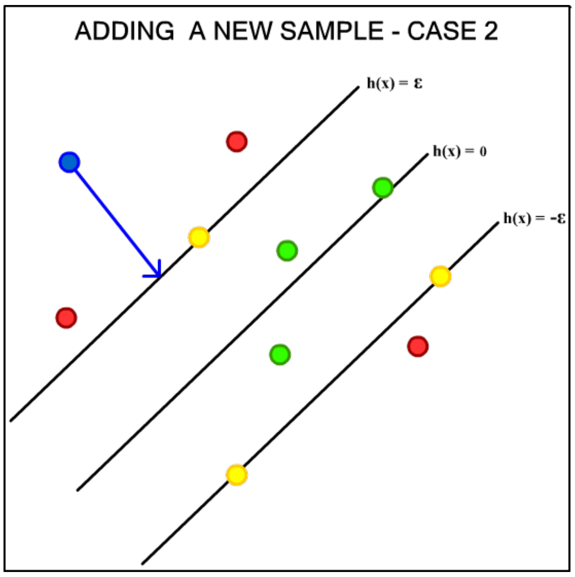
In this image, each circle corresponds to a sample. The red circles are error samples, the yellow-ones are support samples and the green-ones are remaining samples.

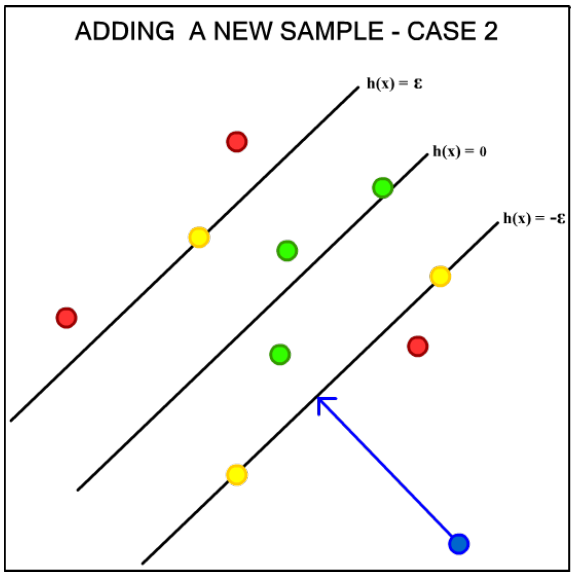
When you try to add a new sample, there can be two different possibilities (blue circle is the new sample):



The first possibility is that the new sample error is less than ϵ. This is the easiest case: the sample can be added to the remaining samples set without changes of other samples.

Otherwise, when the error is bigger than ϵ the situation is more complicated:

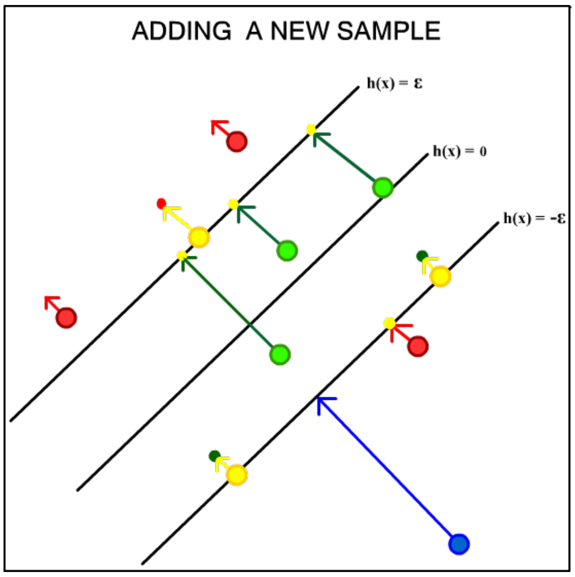




In this situation θ_c or $h(\mathbf{x}_c)$ value of the new sample changes until it reaches the support or the error set.

Notice that the arrow represents the sample movements of θ_i or $h(\mathbf{x}_i)$ components.

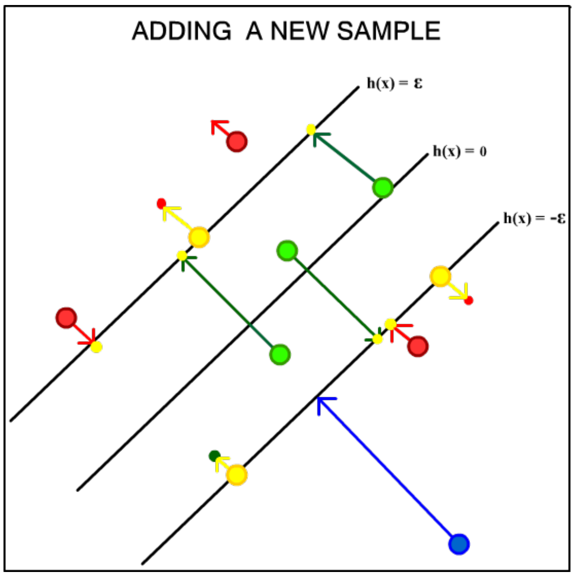
The problem is that these variations influence other samples:



At the end of the updating procedure, if variation is too big, some samples could change their θ_i and $h(\mathbf{x}_i)$ values and exit from their set.

So before move new sample to support or error set, we should move all samples that change their set with less variation.

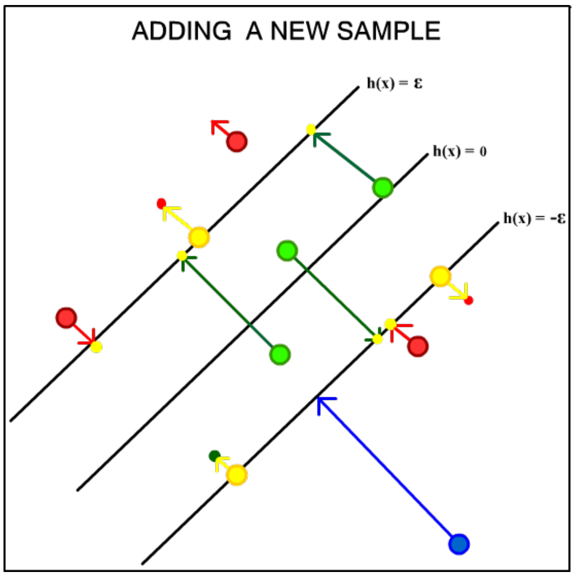
Another important element that should be considered is that not all the samples are moving in the same direction as the new sample:



The direction of a sample is determined by the sign of β_i for support samples and by the sign of γ_i for error and remaining samples.

Following, are the possible moves from the set.

4.9 From Support to Error Set



When a sample goes from support to error set, his θ_i from $0 < |\theta_i| < C$ becomes $= C$ and $|h(\mathbf{x}_i)|$ from $= \epsilon$ becomes $> \epsilon$.

For support samples, the updating formula is:

$$\Delta\theta_i = \beta_i \Delta\theta_c \tag{4.1}$$

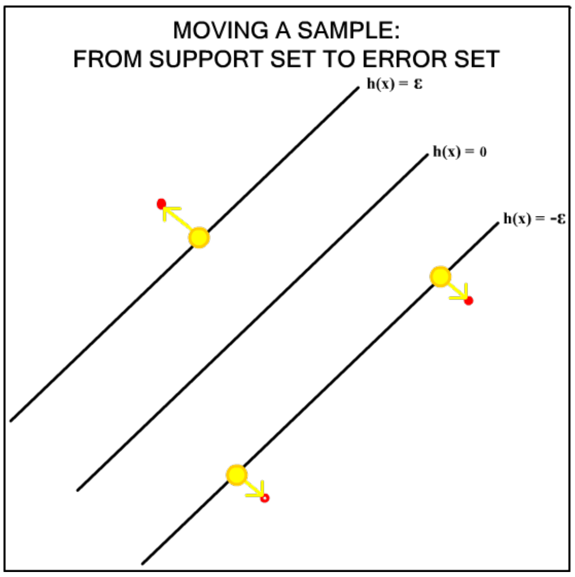
The relative variation required is:

$$\Delta\theta_c = \Delta\theta_i/\beta_i \tag{4.2}$$

In this case:

$$\Delta\theta_c = (C - \theta_i)/\beta_i \;\; \mathit{or} \;\; (-C - \theta_i)/\beta_i \tag{4.3}$$

4.10 From Support to Remaining Set



When a sample goes from support to remaining set, his θ_i from $0 < |\theta_i| < C$ becomes $= 0$ and $|h(\mathbf{x}_i)|$ from $= \epsilon$ becomes $< \epsilon$.

For support samples, the updating formula is:

$$\Delta\theta_i = \beta_i\Delta\theta_c \tag{4.4}$$

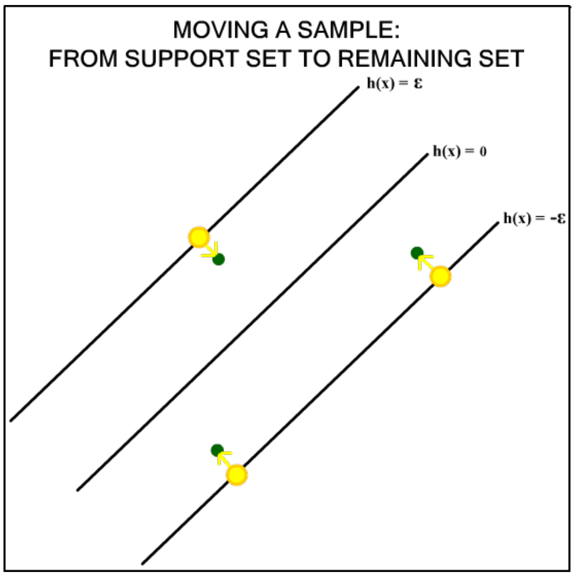
The relative variation required is:

$$\Delta\theta_c = \Delta\theta_i/\beta_i \tag{4.5}$$

In this case:

$$\Delta\theta_c = -\theta_i/\beta_i \tag{4.6}$$

4.11 From Error to Support Set



When a sample goes from error to support set, his $|h(\mathbf{x}_i)|$ from $> \epsilon$ becomes $= \epsilon$. For error samples, the updating formula is:

$$\Delta h(\mathbf{x}_i) = \gamma_i \Delta\theta_c \tag{4.7}$$

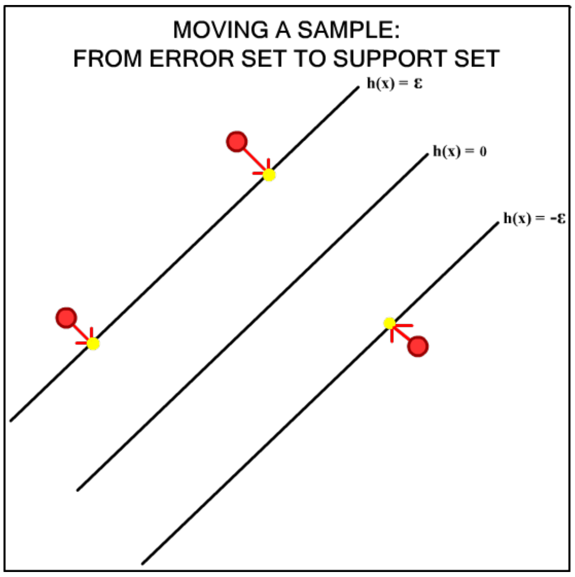
The relative variation required is:

$$\Delta\theta_c = \Delta h(\mathbf{x}_i)/\gamma_i \tag{4.8}$$

In this case:

$$\Delta\theta_c = (-\epsilon - h(\mathbf{x}_i))/\gamma_i \quad or \quad (\epsilon - h(\mathbf{x}_i))/\gamma_i \tag{4.9}$$

4.12 From Remaining to Support Set



When a sample goes from remaining to support set, his $|h(\mathbf{x}_i)|$ from $< \epsilon$ becomes $= \epsilon$.

For error samples, the updating formula is:

$$\Delta h(\mathbf{x}_i) = \gamma_i \Delta\theta_c \tag{4.10}$$

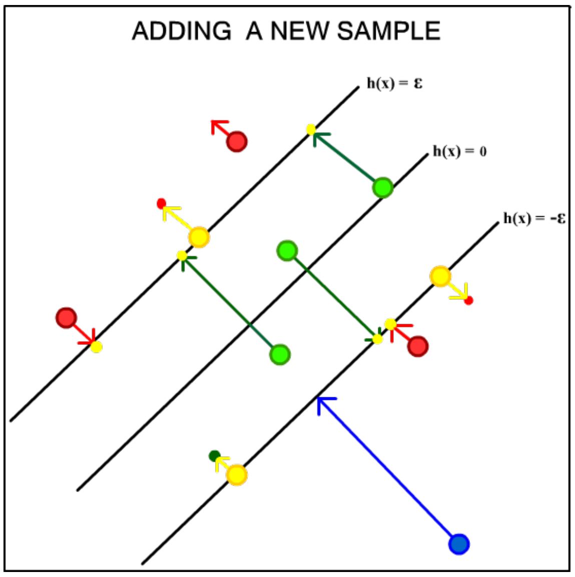
The relative variation required is:

$$\Delta\theta_c = \Delta h(\mathbf{x}_i)/\gamma_i \tag{4.11}$$

In this case:

$$\Delta\theta_c = (-\epsilon - h(\mathbf{x}_i))/\gamma_i \;\; or \;\; (\epsilon - h(\mathbf{x}_i))/\gamma_i \tag{4.12}$$

4.13 How the algorithm works



Before a new sample is added, all the samples that requires a less variation are moved. In every iteration, $h(\mathbf{x}_c)$ and θ_c value of the new sample change.

When the value of $|h(\mathbf{x}_c)|$ is $= \epsilon$, the sample is added to the support set.

Before that this happens, $|\theta_c|$ value becomes $= C$ and the sample is added to the error set.

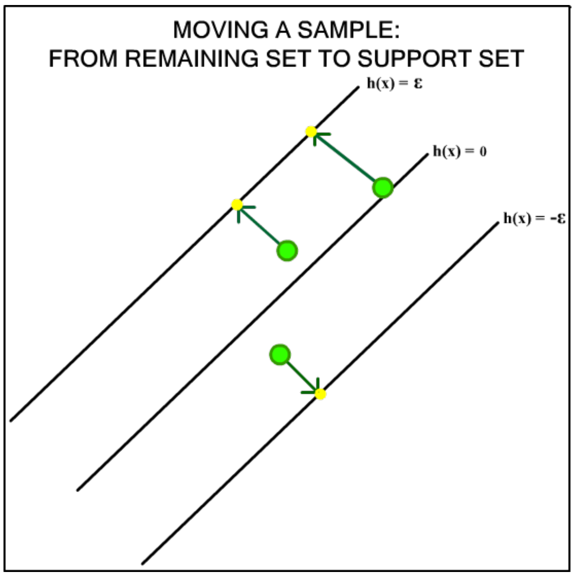
From these considerations, we can be build an incremental algorithm that, starting from an already definite situation, can add an element to the training set. In the next chapter we will expose the complete algorithm.

This chapter shows that before we add a new sample, there can be other moves of other samples from one set to another in order to maintain KKT conditions verified.

4.14 Stabilization

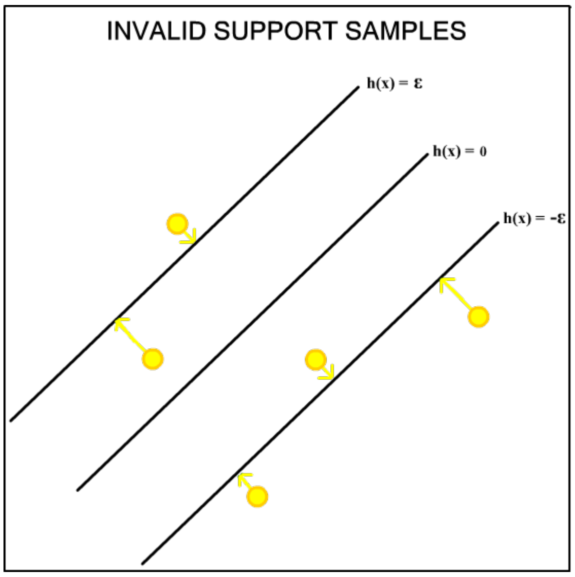
Unfortunately, this algorithm produces a big number of instability errors due to an enormous number of floating-point operations.
So it often happens that a sample exit from its set and does no more verify KKT conditions.
To avoid this problem, there has been ideated a way to correct these samples and move back to its original sets.
The basic idea is to use the same principles applied in normal algorithm and extend them with new possible moves permitted only when a sample exit from its set.
In the next paragraphs are listed all the possible variations.

4.14.1 Invalid Support Samples



A support sample becomes invalid when its $|\theta_i|$ becomes $> C$. In this case the sample, is neither a support sample, neither an error sample.
Note that if KKT conditions aren't violated, this part of the algorithm isn't used.

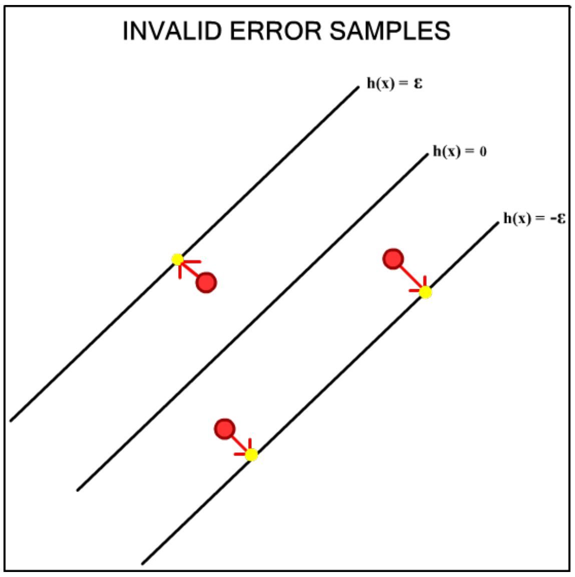
4.14.2 Invalid Error Samples



An error sample becomes invalid when its $|h(\mathbf{x}_i)|$ becomes $< \epsilon$.

In this case the sample, is neither an error sample, neither a remaining sample, because $|\theta_i| = C$.

4.14.3 Invalid Remaining Samples



A remaining sample becomes invalid when its $|h(\mathbf{x}_i)|$ becomes $> \epsilon$.

In this case the sample, is neither a remaining sample, neither an error sample, because $\theta_i = 0$.

Chapter 5

The Algorithm

This chapter finally shows the Online Support Regression Algorithm.
This is the heart of the thesis. For better understanding, I have listed all the equations needed by the algorithm.

5.1 The Learning Algotirithm

5.1.1 Input and Output

After long math proofs and analysis the algorithm is finally shown. It's in a pseudo-code format. The input requested from an Online Support Vector Regression is the following:

```
INPUTS
1. TrainingSet $\{\mathbf{x}_i, y_i, i = 1..l\}$
2. Wheights $\theta_i, i = 1..l$
3. Bias $b$
4. TrainingSet partition into SuppotSet(S) , ErrorSet(E)
   and RemainingSet(R)
5. Params: $\epsilon$, $C$, KernelType and KernelParams
6. R Matrix
7. New Sample c = $(\mathbf{x}_c, y_c)$
```

At the beginning of a training, the TrainingSet, Coefficients, SupportSet, ErrorSet, RemainingSet and R Matrix are empty, the Bias is set to 0.
Notice that all these values should be stored somewhere to train new values. In normal support vector regression training, only the TrainingSet, Weights and Bias are necessary.

```
OUTPUTS
1. New TrainingSet $\{\mathbf{x}_i, y_i, i = 1..l+1\}$
2. New Coefficients $\theta_i, i = 1..l+1$
3. New Bias $b$
4. New TrainingSet partition
5. New R Matrix
```

The output contains all the values given in input updated.

5.1.2 New Sample Training Pseudo-code

This is the pseudo-code of the training of a new sample.

```
NEW TRAINING ALGORITHM

1.    Add $(\mathbf{x}_c, y_c)$ at the TrainingSet
2.    Set $\theta_c = 0$
3.    Compute $f(\mathbf{x}_c)$ and $h(\mathbf{x}_c)$
4.    If $(|h(\mathbf{x}_c)| < \epsilon)$
4.1       Add NewSample to the RemainingSet and Exit
5.    Compute $h(\mathbf{x}_i), i = 1..l$

                                                  continues...
```

```
NEW TRAINING ALGORITHM (CONTINUATION)

6.          While (NewSample is not added into a set)
6.1             Update the values $\beta$ and $\gamma$
6.2             Find Least Variations($L_{c1}, L_{c2}, \mathbf{L}_s, \mathbf{L}_e, \mathbf{L}_r$)
6.3             Find Min Variation $\Delta\theta_c = \min(L_{c1}, L_{c2}, \mathbf{L}_s, \mathbf{L}_e, \mathbf{L}_r)$
6.4             Let Flag the case number that determinates $\Delta\theta_c$
                  $(L_{c1} = 1, L_{c2} = 2, \mathbf{L}_s = 3, \mathbf{L}_e = 4, \mathbf{L}_r = 5)$
6.5             Let $x_l$ the sample that determines $\Delta\theta_c$
6.6             Update $\theta_c, \theta_i, i = 1..l$ and $b$
6.7             Update $h(x_i), i \in E \cup R$

6.8             Switch Flag
6.8.1               (Flag = 1)
6.8.1.1                 Add NewSample to SupportSet
6.8.1.2                 Add NewSample to R Matrix
6.8.1.3                 Exit
6.8.2               (Flag = 2)
6.8.2.1                 Add NewSample to ErrorSet
6.8.2.2                 Exit
6.8.3               (Flag = 3)
6.8.3.1                 If $(\theta_l = 0)$
6.8.3.1.1                   Move Sample $l$ from Support to RemainingSet
6.8.3.1.2                   Remove Sample $l$ from R Matrix
6.8.3.1                 Else $[\theta_l = |C|]$
6.8.3.2.1                   Move Sample $l$ from Support to ErrorSet
6.8.3.2.2                   Remove Sample $l$ from R Matrix
6.8.4               (Flag = 4)
6.8.4.1                 Move Sample $l$ from Error to Support
6.8.4.2                 Add Sample $l$ to R Matrix
6.8.5               (Flag = 5)
6.8.5.1                 Move Sample $l$ from Remaining to SupportSet
6.8.5.2                 Add Sample $l$ to R Matrix
```

At the beginning, Online SVR tries to check if the new sample can be inserted in the RemainingSet. This is the best case, because it is fast and does not increase the complexity of the machine.
If this does not happen, it starts a cycle that finishes only when the new sample is added to Support or Error set (cases 1 and 2).
At each iteration, a sample migrates from one set to another. Each migration passes through SupportSet, so that at every iteration the R matrix (that is correlated to support samples) changes. Because this is a common and the most expensive operation, in the next sections it will be explained a method to efficiently update the R Matrix.

5.1.3 Formula Equations

In this section are reported the equations used in the algorithm.
For more details, please see the previous chapter.
At the beginning of the algorithm, computes f and h (the margin function):

$$f(\mathbf{x}_c) = \sum_{i=1}^{l}(\theta_i Q_{ic}) + b \tag{5.1}$$

$$h(\mathbf{x}_c) = f(\mathbf{x}_c) - y_c \tag{5.2}$$

At each iteration, β and γ, are useful for least variations calculus:

$$\beta = \begin{bmatrix} \beta_b \\ \beta_{s_1} \\ \vdots \\ \beta_{s_{l_s}} \end{bmatrix} = -\mathbf{R} \begin{bmatrix} 1 \\ Q_{s_1 c} \\ \vdots \\ Q_{s_{l_s} c} \end{bmatrix} \tag{5.3}$$

$$\gamma = \begin{bmatrix} \Delta Q_{n_1 c} \\ \vdots \\ \Delta Q_{n_{l_n} c} \end{bmatrix} + \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q_{n_1 s_1} & \cdots & Q_{n_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q_{n_{l_n} s_1} & \cdots & Q_{n_{l_n} s_{l_s}} \end{bmatrix} \beta \tag{5.4}$$

When the least variation is found, it is possible to update θ_c, $\theta_i|\ i \in S$, b and $h(\mathbf{x}_i)|\ i \in E \cup R$:

$$\theta_c = \theta_c + \Delta\theta_c \tag{5.5}$$

$$\theta_i = \theta_i + \Delta\theta_i \tag{5.6}$$

$$b = b + \Delta b \tag{5.7}$$

$$\begin{bmatrix} \Delta b \\ \Delta\theta_{s_1} \\ \vdots \\ \Delta\theta_{s_{l_s}} \end{bmatrix} = \beta\Delta\theta_c \tag{5.8}$$

$$h(\mathbf{x}_i) = h(\mathbf{x}_i) + \Delta h(\mathbf{x}_i) \tag{5.9}$$

$$\begin{bmatrix} \Delta h(\mathbf{x}_{n_1}) \\ \vdots \\ \Delta h(\mathbf{x}_{n_{l_n}}) \end{bmatrix} = \gamma\Delta\theta_c \tag{5.10}$$

5.1.4 Find Least Variations

Combining the considerations listed in previous chapters, we can write down the conditions for the moving of the samples.
Before analyzing all the possible moves, it's necessary to define the direction in which it is moving. For learning, it's the opposite direction of H_c, because we want to reduce this value until it reaches $|\epsilon|$:

$$\textsc{Direction} = sign(-H_c) \tag{5.11}$$

Now it's possible to define the moving steps. The first variation considered is that new sample reaches $|\epsilon|$ and consequently the SupportSet.
Variation value is:

```
$L_{C1}$ VARIATION

IF $((\theta_c > 0) \lor (\theta_c = 0 \land H_c < 0))$
    $L_{C1} = (-H_c - \epsilon)/\gamma_c$
ELSE
    $L_{C1} = (-H_c + \epsilon)/\gamma_c$
END
```

The second variation measures the distance of new sample from error set.
Variation value is:

```
$L_{C2}$ VARIATION

IF (DIRECTION $> 0$)
    $L_{C2} = -\theta_c + C$
ELSE
    $L_{C2} = -\theta_c - C$
END
```

The third variation measures the distance of each Support sample to Error or RemainingSet.
The Sample's direction depends on the direction of new sample and by the value of

β_i. If it is negative, the sample goes in the opposite direction.

Notice that the index i in β is not referred to the position in the vector but to the position inside the SupportSet.

Variation values are:

```
$L_S$ VARIATIONS

ForEach $i \in$ SupportSet
    If (Direction $* \beta_i > 0$)
        If ($H_i > 0$) [$H_i = \epsilon$]
            If ($\theta_i < -C$)
                $L_{S_i} = (-\theta_i - C)/\beta_i$
            ElseIf ($\theta_i <= 0$)
                $L_{S_i} = -\theta_i/\beta_i$
            Else [$\theta_i > 0$]
                $L_{S_i} =$ Direction*Inf
            End
        Else [$H_i = -\epsilon$]
            If ($\theta_i < 0$)
                $L_{S_i} = -\theta_i/\beta_i$
            ElseIf ($\theta_i <= C$)
                $L_{S_i} = (-\theta_i + C)/\beta_i$
            Else [$\theta_i > +C$]
                $L_{S_i} =$ Direction*Inf
            End
        End
    Else [Direction $* \beta_i < 0$]
        If ($H_i > 0$) [$H_i = -\epsilon$]
            If ($\theta_i < -C$)
                $L_{S_i} =$ Direction*Inf
            ElseIf ($\theta_i <= 0$)
                $L_{S_i} = (-\theta_i - C)/\beta_i$
            Else [$\theta_i > 0$]
                $L_{S_i} = -\theta_i/\beta_i$
            End
        Else [$H_i = \epsilon$]
            If ($\theta_i < 0$)
                $L_{S_i} =$ Direction*Inf
            ElseIf ($\theta_i <= C$)
                $L_{S_i} = -\theta_i/\beta_i$
            Else [$\theta_i > +C$]
                $L_{S_i} = (-\theta_i + C)/\beta_i$
            End
    End
End
```

The fourth variation measures the distance of each error sample to SupportSet. The samples's direction depends on the direction of the new sample and by the value of γ_i. If it is negative, the sample goes in the opposite direction.

Notice that the index i in γ is not referred to the position in the vector but to the position in the ErrorSet.

Variation values are:

```
$L_E$ VARIATIONS

ForEach $i \in$ ErrorSet
    If (Direction*$\gamma_i > 0$)
        If ($\theta_i > 0$) [$\theta_i = C$]
            If ($H_i < -\epsilon$)
                $L_{E_i} = (-H_i - \epsilon)/\gamma_i$
            Else
                $L_{E_i} =$ Direction*Inf
            End
        Else [$\theta_i - = C$]
            If ($H_i < \epsilon$)
                $L_{E_i} = (-H_i + \epsilon)/\gamma_i$
            Else
                $L_{E_i} =$ Direction*Inf
            End
        End
    End
    Else [Direction*$\gamma_i < 0$]
        If ($\theta_i > 0$) [$\theta_i = C$]
            If ($H_i > -\epsilon$)
                $L_{E_i} = (-H_i - \epsilon)/\gamma_i$
            Else
                $L_{E_i} =$ Direction*Inf
            End
        Else [$\theta_i - = C$]
            If ($H_i > \epsilon$)
                $L_{E_i} = (-H_i + \epsilon)/\gamma_i$
            Else
                $L_{E_i} =$ Direction*Inf
            End
        End
    End
End
```

The last variation measures the distance of each remaining sample to SupportSet. The samples's direction depends on the direction of the new sample and by the value of γ_i. If it is negative, the sample goes in the opposite direction.

Notice that the index i in γ is not referred to the position in the vector but to the position in the RemainingSet.

Variation values are:

```
$L_R$ VARIATIONS

ForEach $i \in$ ErrorSet
    If (Direction*$\gamma_i > 0$)
        If ($H_i < -\epsilon$)
            $L_{R_i} = (-H_i - \epsilon)/\gamma_i$
        ElseIf ($H_i < +\epsilon$)
            $L_{R_i} = (-H_i + \epsilon)/\gamma_i$
        Else
            $L_{R_i} =$ Direction*Inf
        End
    Else [Direction*$\gamma_i < 0$]
        If ($H_i > +\epsilon$)
            $L_{R_i} = (-H_i + \epsilon)/\gamma_i$
        ElseIf ($H_i > -\epsilon$)
            $L_{R_i} = (-H_i - \epsilon)/\gamma_i$
        Else
            $L_{R_i} =$ Direction*Inf
        End
    End
End
```

It's easy to notice that not all the conditions seem probable, but for instability problems on floating points operations, in practice, these are all used.
In the next chapters we will be shown how to take advantage of these enlarged conditions.

5.1.5 Update of the R Matrix

As seen in the chapter 4, R Matrix is defined as:

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q_{s_1 s_1} & \cdots & Q_{s_1 s_{l_s}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q_{s_{l_s} s_1} & \cdots & Q_{s_{l_s} s_{l_s}} \end{bmatrix}^{-1} \tag{5.12}$$

In the algorithm above, we see that it is very inefficient at each iteration to rebuild the matrix, due to the high complexity of the matrix inversion (about $O(n^2 log(n))$). To avoid this problem, we have developed a method, specific for this type of matrix that is more efficient (about $O(s^2)$, where s is the number of SupportSet samples). Now, we can add or remove a sample from R Matrix with this method [1].

Add First Sample To R

The first sample should be updated in this way:

$$\mathbf{R} = \begin{bmatrix} -Q_{11} & 1 \\ 1 & 0 \end{bmatrix} \tag{5.13}$$

Add Other Samples To R

The other samples are more expensive to update because of its computation time:

$$\mathbf{R}_{NEW} = \begin{bmatrix} & & & 0 \\ & \mathbf{R} & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & 0 \end{bmatrix} + \frac{1}{\gamma_i} \begin{bmatrix} \beta \\ 1 \end{bmatrix} \begin{bmatrix} \beta^T & 1 \end{bmatrix} \tag{5.14}$$

If the new sample is added to the SupportSet, γ_i is definite as:

$$\gamma_i = Q_{CC} + \begin{bmatrix} 1 & Q_{CS_1} & \cdots & Q_{CS_{l_s}} \end{bmatrix} \beta \tag{5.15}$$

Otherwise, if the new sample is moved from Error or RemainingSet to SupportSet, one should recompute also β and γ_i as follows:

$$\beta = -\mathbf{R} \begin{bmatrix} 1 \\ Q_{is_1} \\ \vdots \\ Q_{is_{l_s}} \end{bmatrix} \tag{5.16}$$

$$\gamma_i = Q_{ii} + \begin{bmatrix} 1 & Q_{iS_1} & \cdots & Q_{iS_{l_s}} \end{bmatrix} \beta \tag{5.17}$$

Remove a Sample From R

The remove procedure delete the line and the column of the erased sample and update the others:

$$\begin{gathered} I = [1 \cdots (i-1) \;\; (i+1) \cdots (l_s+1)] \\ \mathbf{R}_{NEW} = R_{I,I} - \frac{\mathbf{R}_{I,i}\mathbf{R}_{i,I}}{R_{i,i}} \end{gathered} \tag{5.18}$$

5.2 The Unlearning Algorithm

This section will show the unlearning algorithm. It is very similar to learning, so only the differences between the two algorithms will be discussed.

5.2.1 Input and Output

The learning and unlearning algorithms are very similar to each other.
The first is a cycle where the h_c value of the new sample decreases until it does not reach its set. The second is a cycle where the θ_c value decreases until it doesn't become 0.
The input required by in the unlearning version of the algorithm is:

INPUT

1. TrainingSet $\{\mathbf{x}_i, y_i, i = 1..l\}$
2. Wheights $\theta_i, i = 1..l$
3. Bias b
4. TrainingSet partition into SuppotSet(S) , ErrorSet(E) and RemainingSet(R)
5. Params: $\epsilon, C,$ KernelType and KernelParam
6. R Matrix
7. Sample to Remove Index (c)

It is easy to notice that the input is the same of the learning algorithm, except for the fact that the sample should be removed

```
OUTPUT
1. New TrainingSet $\{\mathbf{x}_i, y_i, i = 1..l+1\}$
2. New Coefficients $\theta_i, i = 1..l+1$
3. New Bias $b$
4. New TrainingSet partition
5. New R Matrix
```

The outputs are the values given in input updated.

5.2.2 Sample Forgetting Pseudo-code

This is the pseudo-code for forgetting a sample.

```
FORGETTING ALGORITHM

1.   If ($c \in$ RemainingSet)
1.1      Remove Sample from RemainingSet
1.1      Remove Sample from TrainingSet
1.1      Exit
2.   If ($c \in$ SupportSet)
2.1      Remove Sample from SupportSet
3.   If ($c \in$ ErrortSet)
3.1      Remove Sample from ErrorSet
4.   Compute $h(\mathbf{x}_i), i = 1..l$

                                        continues...
```

```
FORGETTING ALGORITHM (CONTINUATION)

5.          WHILE (SAMPLE C IS NOT REMOVED)
5.1             UPDATE THE VALUES $\beta$ AND $\gamma$
5.2             FIND LEAST VARIATIONS $(L_c, \mathbf{L}_s, \mathbf{L}_e, \mathbf{L}_r)$
5.3             FIND MIN VARIATION $\Delta\theta_c = \min(L_c, \mathbf{L}_s, \mathbf{L}_e, \mathbf{L}_r)$
5.4             LET FLAG THE CASE NUMBER THAT DETERMINATES $\Delta\theta_c$
                  $(L_c = 1, \mathbf{L}_s = 2, \mathbf{L}_e = 3, \mathbf{L}_r = 4)$
5.5             LET $x_l$ THE SAMPLE THAT DETERMINES $\Delta\theta_c$
5.6             UPDATE $\theta_c, \theta_i, i = 1..l$ AND $b$
5.7             UPDATE $h(\mathbf{x}_i), i \in E \cup R$

5.8             SWITCH FLAG
5.8.1               (FLAG = 1)
5.8.1.1                 REMOVE SAMPLE FROM TRAININGSET AND EXIT
5.8.2               (FLAG = 3)
5.8.2.1                 IF $(\theta_l = 0)$
5.8.2.1.1                   MOVE SAMPLE $l$ FROM SUPPORT TO REMAININGSET
5.8.2.1.2                   REMOVE SAMPLE $l$ FROM R MATRIX
5.8.2.1                 ELSE $[\theta_l = |C|]$
5.8.2.2.1                   MOVE SAMPLE $l$ FROM SUPPORT TO ERRORSET
5.8.2.2.2                   REMOVE SAMPLE $l$ FROM R MATRIX
5.8.3               (FLAG = 4)
5.8.3.1                 MOVE SAMPLE $l$ FROM ERROR TO SUPPORT
5.8.3.2                 ADD SAMPLE $l$ TO R MATRIX
5.8.4               (FLAG = 5)
5.8.4.1                 MOVE SAMPLE $l$ FROM REMAINING TO SUPPORTSET
5.8.4.2                 ADD SAMPLE $l$ TO R MATRIX
```

At the beginning, Online SVR check the set of the sample. If it's in the RemainingSet, it is easy to remove.

Otherwise it should be removed from its sample and start a cycle that end only when the θ_c values becomes equal to 0.

5.2.3 Find Least Variations

During the unlearning process, the direction depends on the θ_c value, because we want that it becomes 0.

$$\textsc{Variation} = sign(-\theta_c) \tag{5.19}$$

In unlearning algorithm there aren't L_{C1} and L_{C2} variations. They are replaced by a new distance, L_C, that measures the difference of θ_c from 0.

$$\boxed{\begin{array}{c} L_C \text{ VARIATION} \\ \\ L_C = -\theta_c \end{array}}$$

The variations of Support, Error and RemainingSet are always the same, the only difference is the Direction and this depends on θ_c.

5.3 Stabilization

Stabilization is a technique ideated to correct the errors generated by floating point operations (that have finite precision) and after it has been extended in order to permit online parameters estimation.
It's made up by two parts:

1. *samples correction*: when a sample is outside its set can be corrected moving it in the right one

2. *stabilization procedure*: can be invoked after a training and it checks that all the samples are in the right set. If this doesn't happen, each wrong sample is forgotten and trained again. This is the stabilization procedure:

```
STABILIZATION PROCEDURE

1       For each sample s in TrainingSet
1.1         If s doesn't verify KKT conditions
1.1.1           Forget s
1.1.2           Train s
```

Sample correction is inserted into the learning/unlearning algorithm. Otherwise the stabilize procedure is a function that can be called anytime, for two main purposes:

1. be sure that all the elements are in the right sample, in order to avoid precision errors

2. change one or more parameters and stabilize the SVR. When a parameter changes, many samples automatically exit from their set. Using this procedure is not necessary train the machine from the beginning, but it's sufficient to stabilize the samples.

Chapter 6

Complexity Analysis

This chapter lists the complexity measure of the OnlineSVR training algorithm. The complexity is defined in terms of time and space, based on the number of trained elements.

6.1 Time Complexity

Now the time complexity of the algorithm will be formally definite. Like every good complexity analysis, we compute the value of the complexity in the worst case ($O(f(x))$).
In the table shown below, on the right is visible the complexity of each instruction.

```
NEW TRAINING ALGORITHM                                              COMPLEXITY

1.    ADD $(\mathbf{x}_c, y_c)$ AT THE TRAININGSET                              $O(c)$
2.    SET $O_c = 0$                                                          $O(c)$
3.    COMPUTE $f(\mathbf{x}_c)$ AND $h(\mathbf{x}_c)$                           $O(n)O(kernel)$
4.    IF $(|h(\mathbf{x}_c)| < \epsilon)$
4.1       ADD NEWSAMPLE TO THE REMAININGSET AND EXIT                    $O(c)$
5.    COMPUTE $h(\mathbf{x}_i), i = 1..l$                                     $O(n^2)O(kernel)$
```

```
NEW TRAINING ALGORITHM (CONTINUATION)                                   COMPLEXITY

6.         WHILE (NEWSAMPLE IS NOT ADDED INTO A SET)                              $O(n*5)$
6.1            UPDATE THE VALUES $\beta$ AND $\gamma$                               $O(n*l_S)O(kernel)$
6.2            FIND LEAST VARIATIONS $(L_{c1}, L_{c2}, \mathbf{L}_s, \mathbf{L}_e, \mathbf{L}_r)$   $O(n)$
6.3            FIND MIN VARIATION $\Delta O_c = \min(L_{c1}, L_{c2}, \mathbf{L}_s, \mathbf{L}_e, \mathbf{L}_r)$   $O(c)$
6.4            LET FLAG THE CASE THAT DETERMINATES $\Delta O_c$
                 $(L_{c1} = 1, L_{c2} = 2, \mathbf{L}_s = 3, \mathbf{L}_e = 4, \mathbf{L}_r = 5)$
6.5            LET $x_l$ THE SAMPLE THAT DETERMINES $\Delta O_c$
6.6            UPDATE $O_c, O_i, i = 1..l$ AND $b$                               $O(l_s)$
6.7            UPDATE $h(\mathbf{x}_i), i \in E \cup R$                          $O(n)O(kernel)$

6.8            SWITCH FLAG
6.8.1              (FLAG = 1)
6.8.1.1                ADD NEWSAMPLE TO SUPPORTSET                            $O(c)$
6.8.1.2                ADD NEWSAMPLE TO R MATRIX                              $O(l_S^2)O(kernel)$
6.8.1.3                EXIT
6.8.2              (FLAG = 2)
6.8.2.1                ADD NEWSAMPLE TO ERRORSET                              $O(c)$
6.8.2.2                EXIT
6.8.3              (FLAG = 3)
6.8.3.1                IF $(O_l = 0)$
6.8.3.1.1                  MOVE SAMPLE $l$ FROM S TO E                        $O(c)$
6.8.3.1.2                  REMOVE SAMPLE $l$ FROM R MATRIX                    $O(l_S^2)$
6.8.3.1                ELSE $[O_l = |C|]$
6.8.3.2.1                  MOVE SAMPLE $l$ FROM S TO E                        $O(c)$
6.8.3.2.2                  REMOVE SAMPLE $l$ FROM R MATRIX                    $O(l_S^2)$
6.8.4              (FLAG = 4)
6.8.4.1                MOVE SAMPLE $l$ FROM E TO S                            $O(c)$
6.8.4.2                ADD SAMPLE $l$ TO R MATRIX                             $O(l_S^2)O(kernel)$
6.8.5              (FLAG = 5)
6.8.5.1                MOVE SAMPLE $l$ FROM R TO S                            $O(c)$
6.8.5.2                ADD SAMPLE $l$ TO R MATRIX                             $O(l_S^2)O(kernel)$
```

6.2 Space complexity

Space complexity is very easy to compute:

SPACE COMPLEXITY

1. TrainingSet $\{\mathbf{x}_i, y_i, i = 1..l\}$	$O(n)$
2. Wheights $O_i, i = 1..l$	$O(n)$
3. Bias b	$O(c)$
4. TrainingSet partition into SuppotSet(S) , ErrorSet(E) and RemainingSet(R)	$O(n)$
5. R Matrix	$O(l_S^2)$
6. β and γ	$O(n)$
7. KernelMatrix (Optional)	$O(n^2)$

6.3 Considerations

It is easy to see that the algorithm does not seem very efficient.
In fact, if we consider the worst case, to add a new sample we need:

$$O(5n) * O(n * l_s) * O(kernel)$$

that becomes:

$$O(n^3) * O(kernel)$$

when all the training samples are support samples.
The complexity of the kernel operations can easily be avoided by saving all the kernel values in a matrix. This reduce time complexity, yet adds to the space complexity the factor $O(n^2)$. Other considerations can be done: the complexity $O(n^3)$ can seem to high compared to other support vector for regression algorithms, but, in practice, this does not happen. In the average case, the algorithm has complexity $O(c * n^2)$, as shown in the test section.
The speed of learning depends mostly on the number of support vectors, that can influence significatively performances.

Chapter 7

Tests

In this chapter are shown the results of the test computed using Online Support Vector Regression algorithm.

7.1 Samples Number and Iterations Count

In the following test we compare the number of samples trained versus the number of iterations.
The TrainingSet is extracted from the AutoMPG database that is contained in the UCI Machine Learning Repository [11].
The training parameters used are: ϵ=0.1, C=1, KernelType=RadialBasisFunction, KernelParam=10.

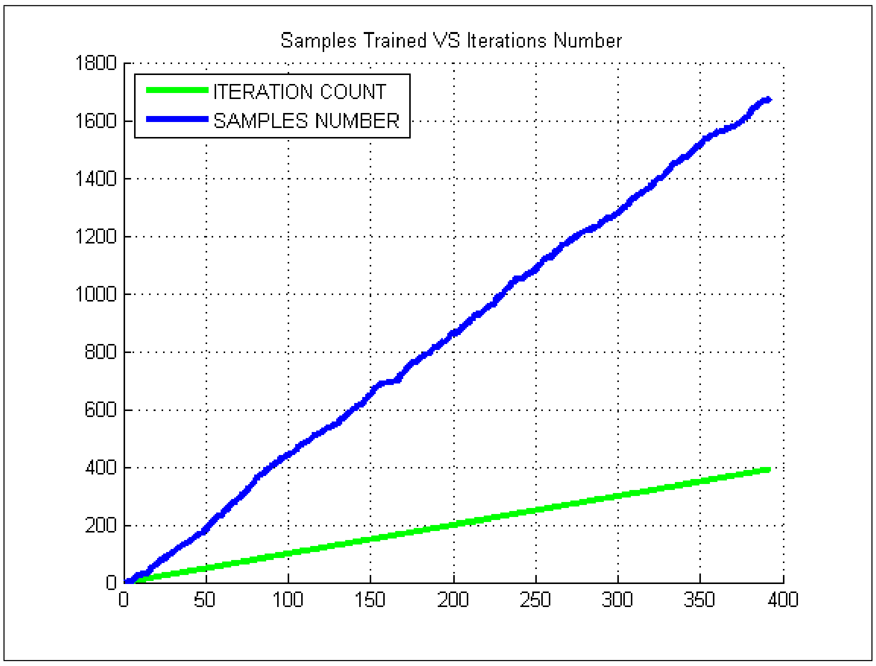


Figure 7.1: Samples trained compared with iteration count

In the test their relation seems be linear. After a number of samples trained, SVR should begin to predict the samples correctly, so the number of iterations decrease. By using bad parameters, the machine doesn't learn, and the iteration count probably will be higher.

7.2 OnlineSVR and LibSVM

Another important test is the speed: if the online algorithm is too slow, is it would be preferred to do a batch training for each time. So, we have compared the Online Support Vector Regression Algorithm with one of the more used support vector software: LibSVM.
The TrainingSet is extracted from the AutoMPG database; the training parameters used are: ϵ=0.1, C=1, KernelType=RadialBasisFunction, KernelParam=10.

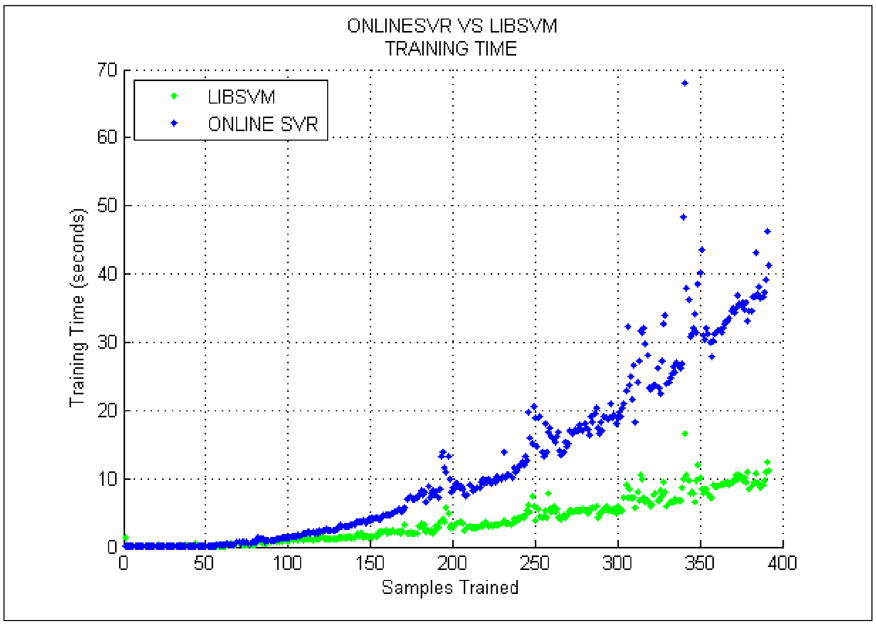


Figure 7.2: OnlineSVR and LibSVM compared in a batch training

LibSVM appears faster than Online Support Vector Regression algorithm. But the OnlineSVR's main purpose is to optimize the speed of a new sample trained, not the total speed of the training.

So, we can compare the exact time to add a new sample using the two methods:

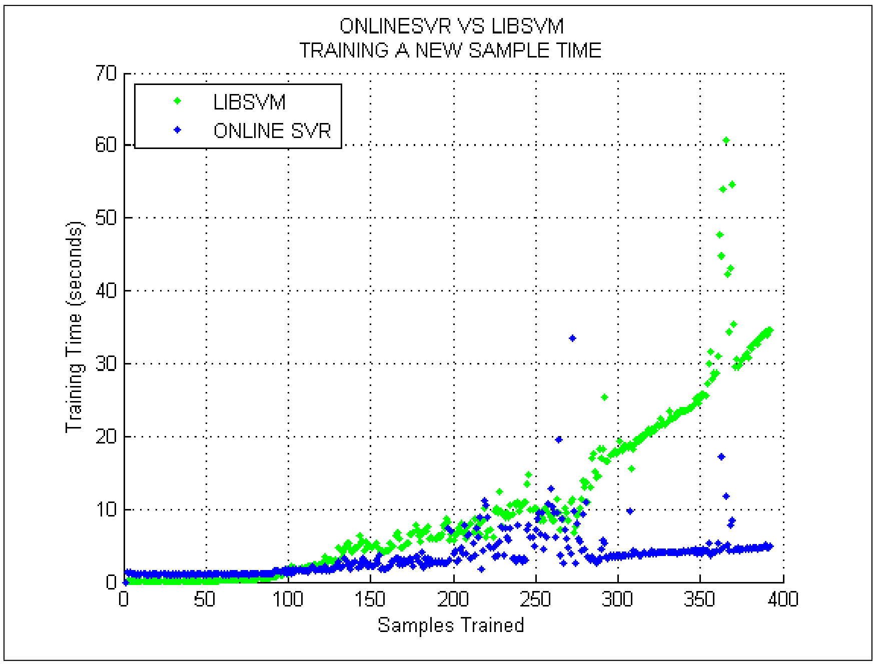


Figure 7.3: OnlineSVR and LibSVM compared in an online training

We can see that OnlineSVR is quicker than LibSVM in incremental learning.

7.3 Stabilization

One of the more interesting characteristics of OnlineSVR is stabilization. It corrects floating-point errors, but has another important application: if a parameter changes, with stabilization it is possible to stabilize the SVR without restarting the training. Is it faster to restart the training or use stabilization algorithm? To answer to this question, we prepared a test where, starting from a trained OnlineSVR, we changed all the parameters and compared the results.
The table shows the parameters of the SVR used during the stabilization test.

Training	ϵ	C	KernelType	KernelParam
Default	0.01	3	RBF	50
Increasing C	0.01	10	RBF	50
Decreasing C	0.01	1	RBF	50
Increasing ϵ	0.1	3	RBF	50
Decreasing ϵ	0.0001	3	RBF	50
Increasing KernelParam	0.01	3	RBF	100
Decreasing KernelParam	0.01	3	RBF	10

Table 7.1: Training parameters of the stabilization test

7.3.1 Stabilizing C

The C parameter influences Support and ErrorSet. From these results, it seems that it should be used with an increment of speed. When used in an application, it could be changed to avoid overfitting and underfitting.

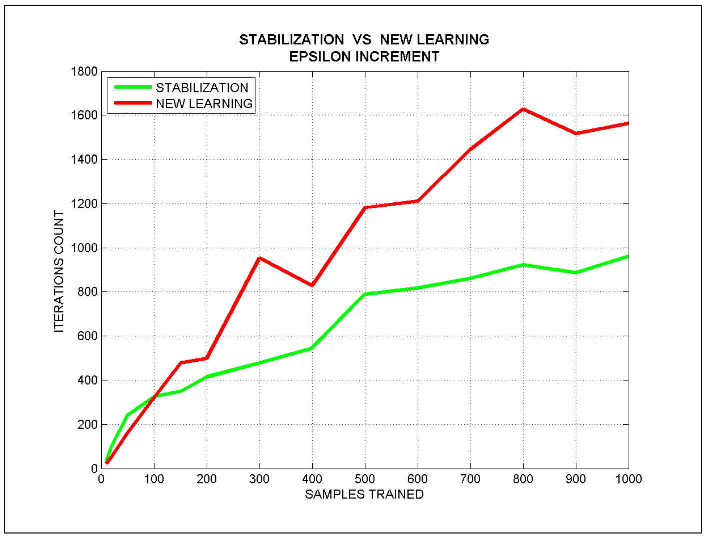


Figure 7.4: Stabilization and new learning time when C is increased

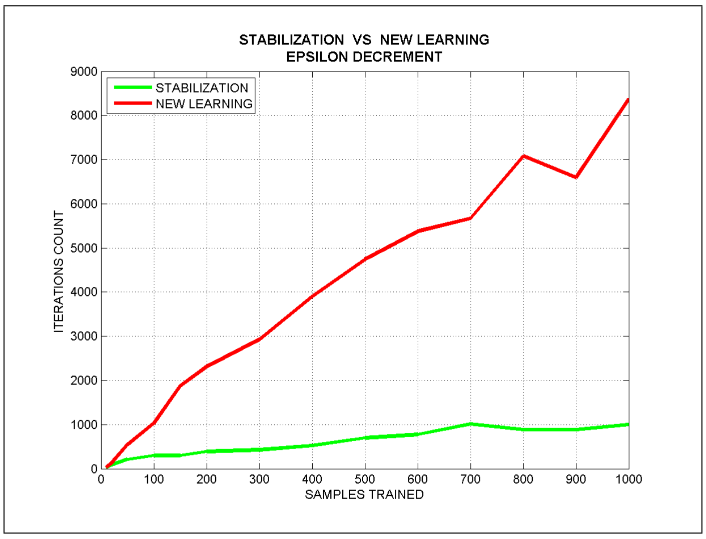


Figure 7.5: Stabilization and new learning time when C is decreased

7.3.2 Stabilizing Epsilon

ϵ is more critical than C because it influences all the samples. In fact, the number of iterations needed is higher and, sometimes the algorithm speed is lower than if we were to restart the training.

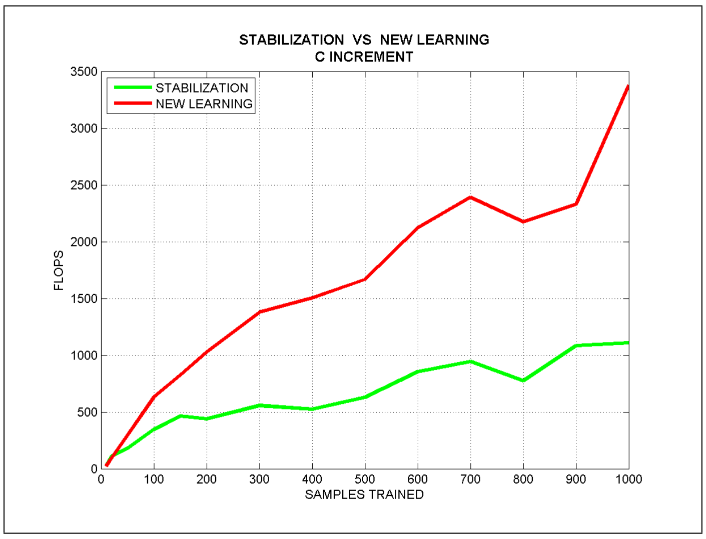


Figure 7.6: Stabilization and new learning time when ϵ is increased

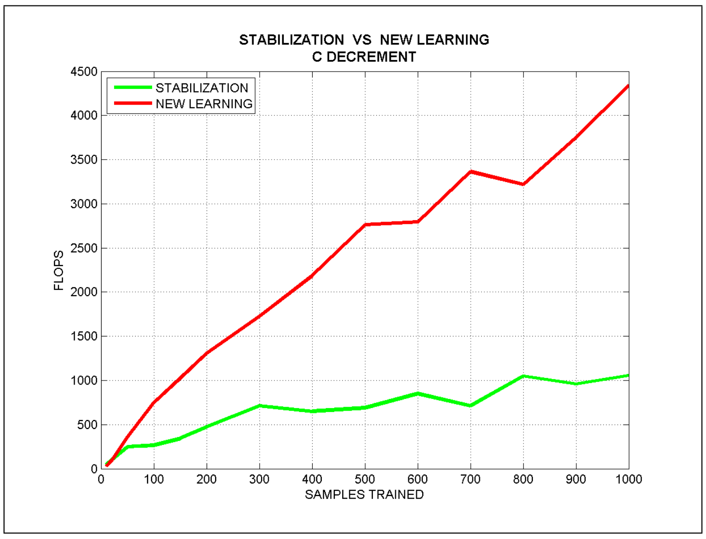


Figure 7.7: Stabilization and new learning time when ϵ is decreased

7.3.3 Stabilizing KernelParam

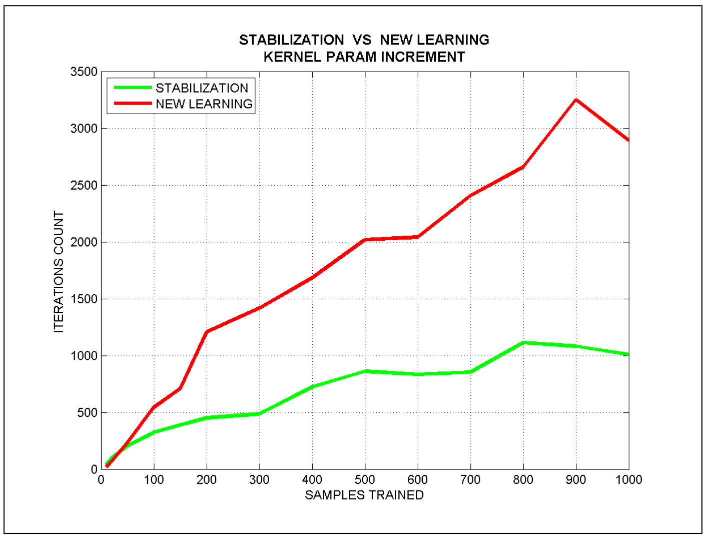


Figure 7.8: Stabilization and new learning time when kernel param is increased

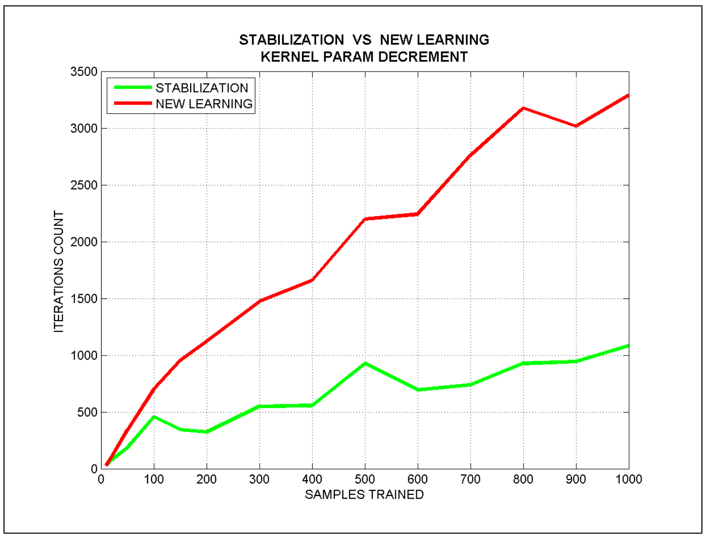


Figure 7.9: Stabilization and new learning time when kernel param is decreased

It's easy to see that stabilization seems more efficient than starting the training from the beginning. However, even if stabilization works better when there are little variations, so that the samples don't change a lot.

Another interesting thing is that stabilization works even if we change the kernel type.

This means that it is possible build a SVR that in the beginning has a linear kernel, and later can be converted in a RBF that offers a more flexible solution.

7.4 Robot arm test

We have given importance to the capacity of learning online over a practical case. In Genoa, there is a robot named James at D.I.S.T. which is at the moment made up by a head and an arm:

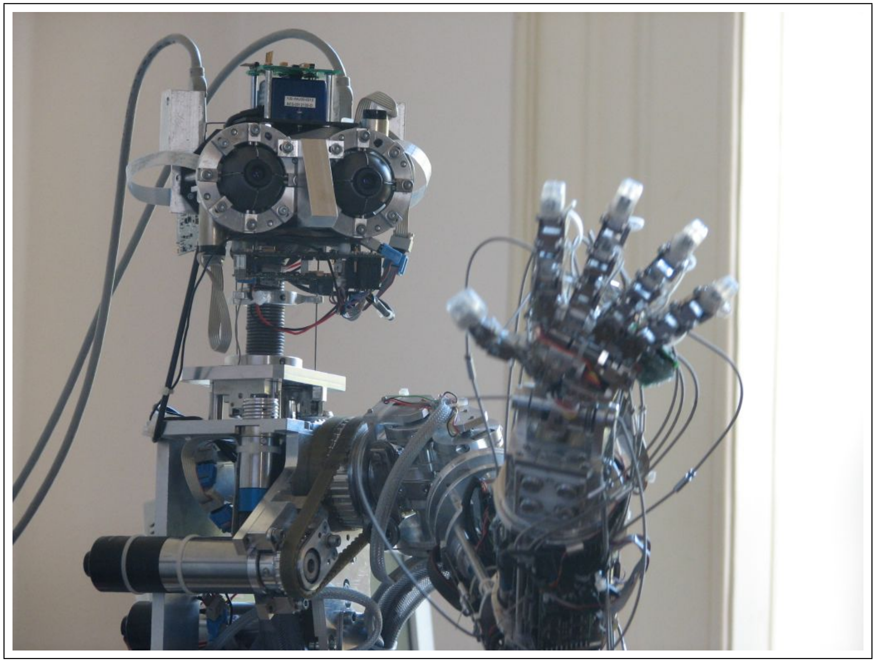


Figure 7.10: The James robot

The arm has seven degrees of freedom, approximately as in humans. At the moment, the arm is controlled by microprocessor based cards that give the right impulse for the motors to be able to move this arm in the desired direction. The idea of this experiment is to dynamically train the OnlineSVR in order to simulate the controller. In practice, given the desired position we wish to reach, certain voltages are tried out in order to complete the necessary movement of an arm.
Because every SVR is only able to predict one result at a time, we need one for each motor in the arm. In this test we have experimented with three degrees of freedom, and therefore we have used three motors.

7.4.1 The training scheme

Training is performed with the micro-cotroller enabled: for a certain period of time the SVR registers the voltage needed to generate each motor movement. Once the

SVR is finally trained, the micro-controller is substituted by the SVR and it will generates the new tension values.
This experiment has many risks: if the SVR doesn't learn the function well enough, it sends out incorrect voltages and this in turn could cause damage to the robot due to any wrong movements. For this reason we have, first substitute the micro-controller, simulated the training with an input already collected from the robot, to make sure the results were satisfactory.
Here are a summary scheme:

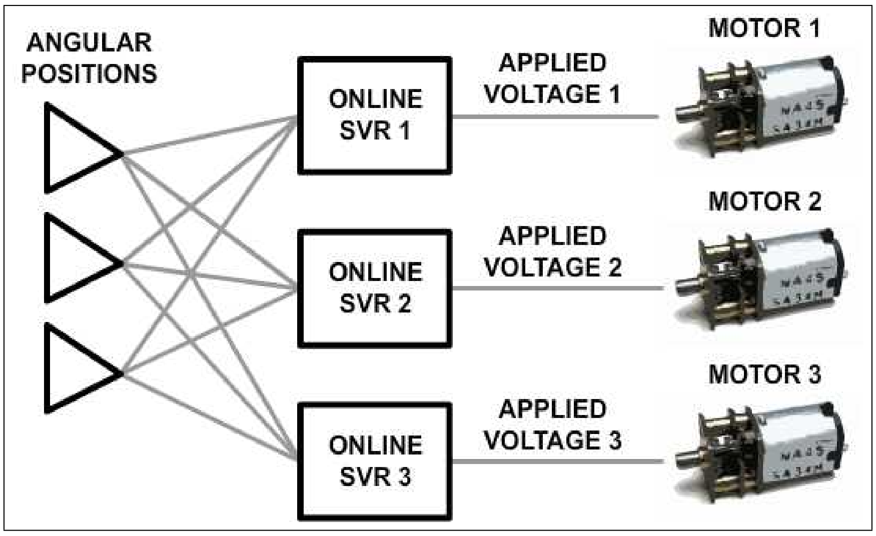


Figure 7.11: The training scheme

Every SVR is separated from the others and generates an independent result. Input data (x, y, z) correspond to the three angles for the arm movement that each motor of the arm should compute. The voltages (output) range from -12V to 12V with an acceptable margin of error of about 1V. We have chosen to use the gaussian kernel and the study of the correct parameters has been determined upon through cross validation.

7.4.2 Results obtained

To measure the training worth we have observed the mean and standard deviation of the error. If the algorithm learns then with time mean and the standard deviation should decrease significantly.

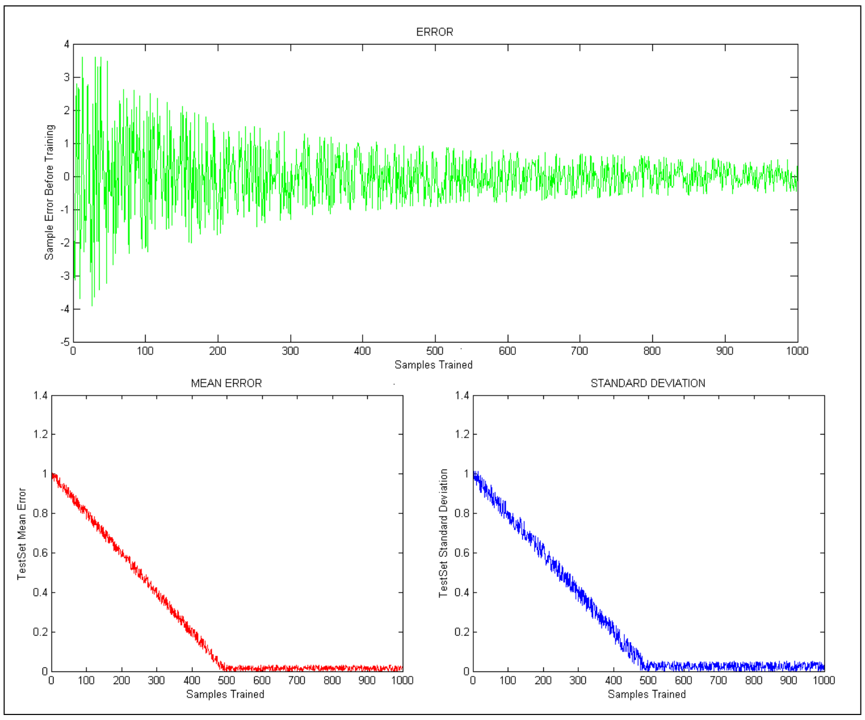


Figure 7.12: The first graph shows the error of a sample before learning, the second one the mean error and the third one the standard deviation

The first is a graph of the errors committed by SVR before that it trains the sample. It should decrease gradually when the machine learn something.
The graph of the mean shows, how, with the augmentation of the number of elements used, the mean error (calculated on a TestSet prepared separately), decreases with time, up to its stabilization. After a certain point, even if we add new examples, the result does not change significantly.
The standard deviation shows, instead, when the error deviates from the mean. A high value implies that the training did not go as planned, because the machine is not reliable and could make big mistakes. If the value is low, however we can be assured that whichever mistake it makes will not cost us more than the mean error itself. In many cases, this number is more important than the mean of the error.

Here are the result obtained from the first motor:

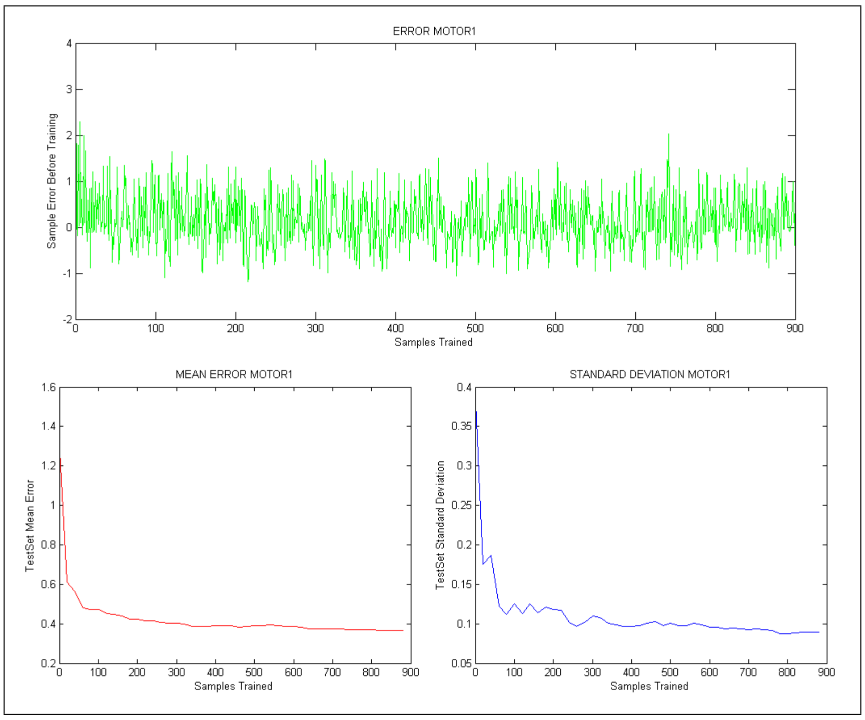


Figure 7.13: Motor Arm 1 results

The second motor:

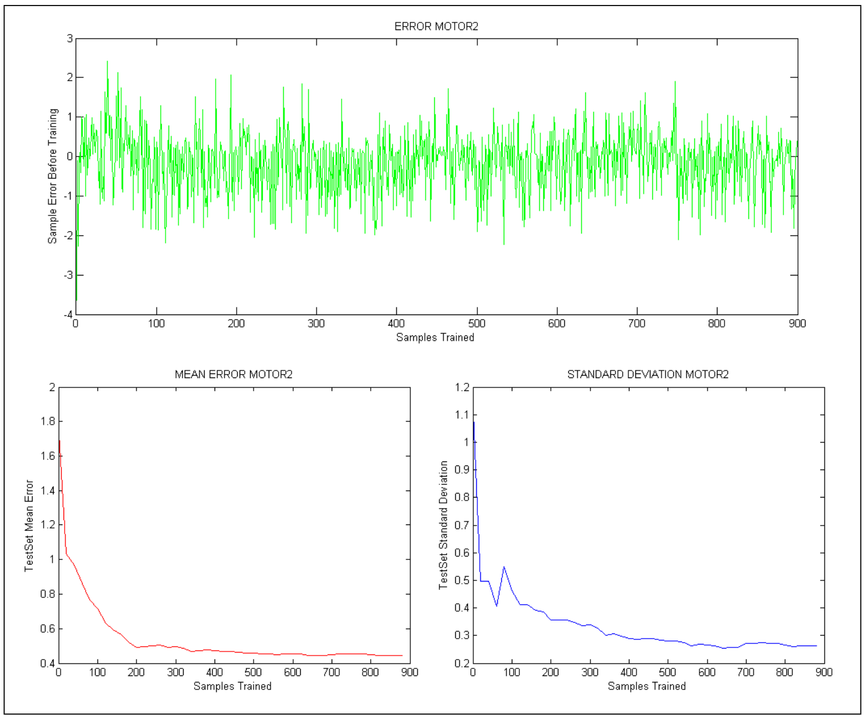


Figure 7.14: Motor Arm 2 results

And the third motor:

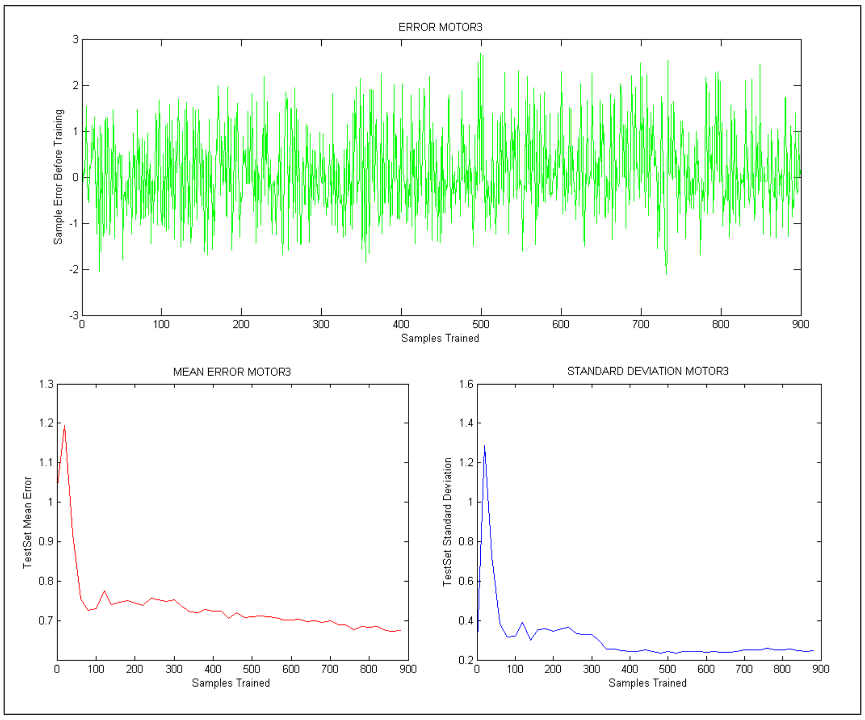


Figure 7.15: Motor Arm 3 results

Unfortunately the machine seems not to learn very well. This is noted in the observation of the graphs and can be because of many factors:

1. The most important is surely that of variance; where the error is too high, there is no minimum amount of reliance on the error made. However, after descending minimally, after 30 elements it stabilizes.

2. Even the mean error remains high, even if the medium is of 0.5 which is half the amount of error permitted, unfortunately there are many little things which cause an error of 2.5-3 volts, errors which could cause serious damages to the robot.

3. If the training doesn't improve the results, retrain it with 30 elements or 1000 brings us to many similar errors.

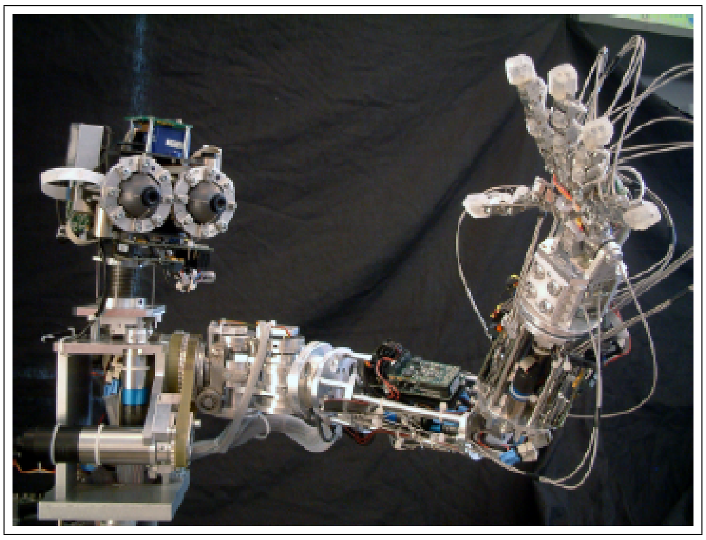


Figure 7.16: The results are not as good as we would

After a more accurate analysis of the problem we realized that the input data were somewhat suspicious. For example two positions of the robot seemed identical:

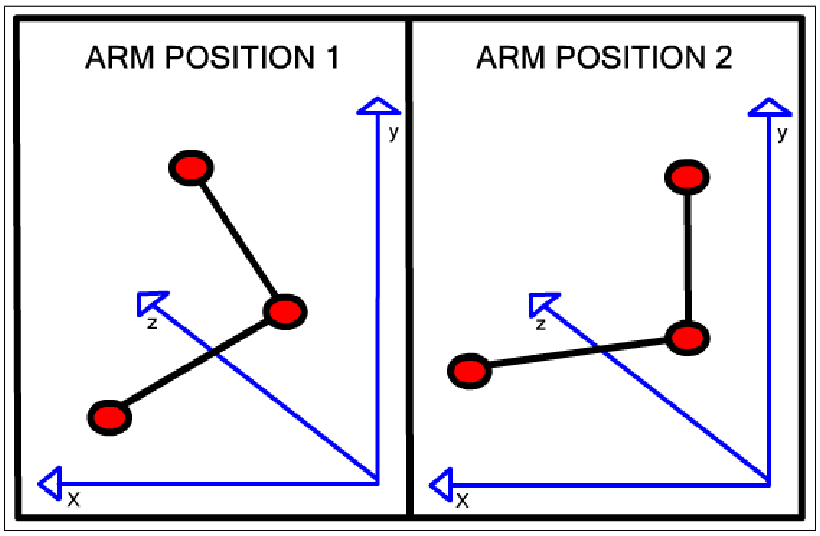


Figure 7.17: Noisy problem: these two arm positions for the robot are the same.

This surely is the main problem. The data suffers of a noise level that is larger than our tolerance and this determines the incapability to find a good solution to the problem.

Chapter 8

Conclusions

8.1 Analysis of the work done

In this thesis the online version of support vector machine for regression algorithm has been analyzed. This algorithm has been viewed from an theoretical an implementation point of view. It has been compared with one of the most frequently used libraries of batch SVM. Thanks to the different implementations in Matlab and C++ it has been possible to construct, a simple version to give a better understanding to those researcher who want to study in depth this algorithm and an efficient version which allows the development of real applications.
The online version is more efficient than the batch when used for incremental learning, which can do the "unlearning".
All errors found in the article [1] have been corrected in this thesis and, thanks to the stabilization technique, it is possible to change dynamically the SVR parameters, like ϵ, C and the kernel type.
A few other problems remain, related to, from the quadratic complexity of the algorithm, which does not permit an increment of the number of samples trained without a great loss of performance.

8.2 Future development

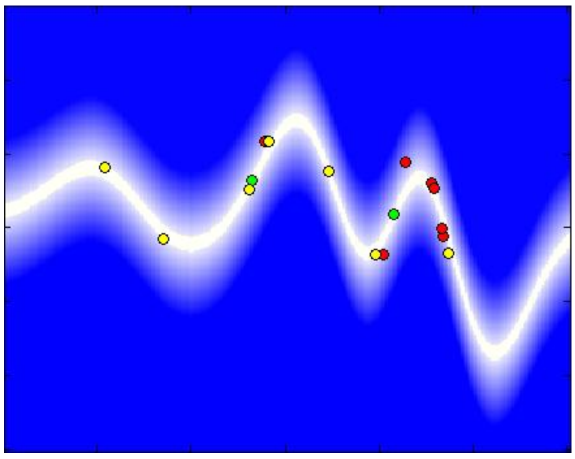
This algorithm has been completed and is current functioning. One of the possible further developments could be that of search functional heuristics which would permit us to choose what are the best examples to use for training, in order to render the operations of learning and unlearning more efficient.
Also, many algorithms already exists for online learning, few can have also the unlearning feature. This permits the opportunity to create new techniques purposely studied to make use of this useful operation, which, up until now, have been left in a dark a little.
Thanks to the stabilization, it is possible to dynamically change the parameters so that other than the only advancement of a machine which can learn new input, it would also be able to automatically determinate which wold be the best parameters during the training phase.

Appendix A

Software Developed

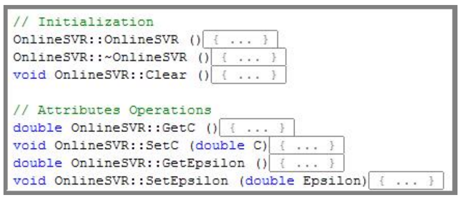
A.1 Matlab Implementation

This is the prototype of the algorithm. It is not very efficient, but it is useful for testing and it has some special features, like the plot of the results; it can also build a video of a the training procedure.
It is available for Linux and Windows.



A.2 C++ Implementation

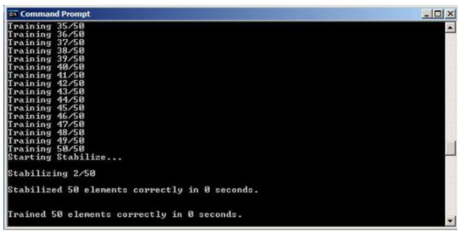
This is the efficient implementation of the algorithm. It has two levels of execution, one is to increment speed, the other reduce the memory utilized. Also, included are “cross validation” and “leave one out” for parameters estimation.



```
// Initialization
OnlineSVR::OnlineSVR ()[ { ... } ]
OnlineSVR::~OnlineSVR ()[ { ... } ]
void OnlineSVR::Clear ()[ { ... } ]

// Attributes Operations
double OnlineSVR::GetC ()[ { ... } ]
void OnlineSVR::SetC (double C)[ { ... } ]
double OnlineSVR::GetEpsilon ()[ { ... } ]
void OnlineSVR::SetEpsilon (double Epsilon)[ { ... } ]
```

A.3 Command Line Interface

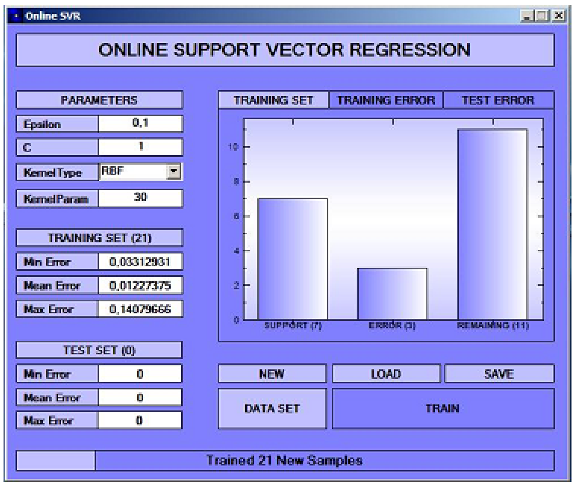
It is based on C++ implementation and is very useful used with script languages. Available for Windows and Linux.



```
Command Prompt
Training 35/50
Training 36/50
Training 37/50
Training 38/50
Training 39/50
Training 40/50
Training 41/50
Training 42/50
Training 43/50
Training 44/50
Training 45/50
Training 46/50
Training 47/50
Training 48/50
Training 49/50
Training 50/50
Starting Stabilize...

Stabilizing 2/50

Stabilized 50 elements correctly in 0 seconds.


Trained 50 elements correctly in 0 seconds.
```

A.4 Windows Interface

It is very useful for manual estimation of the parameters. It also plots the errors of the training and test sets.



PARAMETERS	
Epsilon	0,1
C	1
Kernel Type	RBF
KernelParam	30

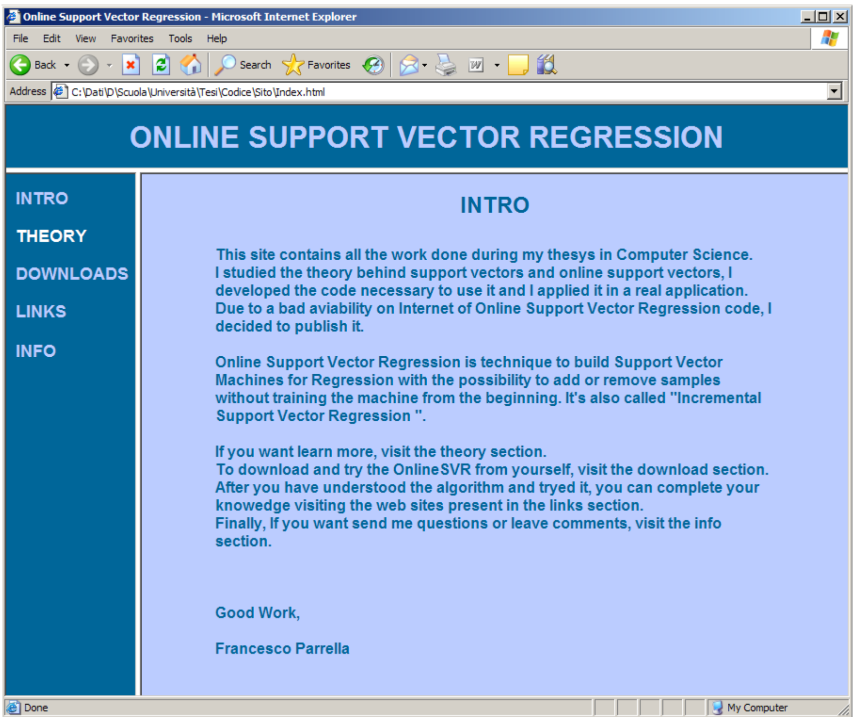
TRAINING SET (21)	
Min Error	0,03312931
Mean Error	0,01227375
Max Error	0,14079666

TEST SET (0)	
Min Error	0
Mean Error	0
Max Error	0

Trained 21 New Samples

A.5 OnlineSVR Web Site

It contains all the documentation of the algorithm and the code produced. It is available on http://onlinesvr.altervista.org.


Online Support Vector Regression - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\Dati\D\Scuola\Università\Tesi\Codice\Sito\Index.html
ONLINE SUPPORT VECTOR REGRESSION
INTRO
THEORY
DOWNLOADS
LINKS
INFO
Done
My Computer

INTRO

This site contains all the work done during my thesys in Computer Science.
I studied the theory behind support vectors and online support vectors, I developed the code necessary to use it and I applied it in a real application.
Due to a bad aviability on Internet of Online Support Vector Regression code, I decided to publish it.

Online Support Vector Regression is technique to build Support Vector Machines for Regression with the possibility to add or remove samples without training the machine from the beginning. It's also called "Incremental Support Vector Regression ".

If you want learn more, visit the theory section.
To download and try the OnlineSVR from yourself, visit the download section.
After you have understood the algorithm and tryed it, you can complete your knowedge visiting the web sites present in the links section.
Finally, If you want send me questions or leave comments, visit the info section.

Good Work,

Francesco Parrella

Appendix B

Parameters Estimation

In this appendix are reported the citations written by many researchers. They suggest how to estimate the parameters, a big problem in real applications.

B.1 C Estimation

”However, it is critical here, as in any regularization scheme, that a proper value is chosen for C, the penalty factor. If it is too large, we have a high penalty for nonseparable points and we may store many support vectors and overfit. If it is too small, we may have underfitting.”
Alpaydin (2004)

”...the coefficient C affects the trade-off between complexity and proportion of non-separable samples and must be selected by the user.”
Cherkassky and Mulier (1998)

”Selecting parameter C equal to the range of output values [6]. This is a reasonable proposal, but it does not take into account possible effect of outliers in the training data.” ”Our empirical results suggest that with optimal choice of ϵ, the value of regularization parameter C has negligible effect on the generalization performance (as long as C is larger than a certain threshold analytically determined from the training data).”

Cherkassky and Ma (2002b)

"In the support-vector networks algorithm one can control the trade-off between complexity of decision rule and frequency of error by changing the parameter C,..."
Cortes and Vapnik

"There are a number of learning parameters that can be utilized in constructing SV machines for regression. The two most relevant are the insensitivity zone e and the penalty parameter C, which determines the trade-off between the training error and VC dimension of the model. Both parameters are chosen by the user."
Kecman (2001)

The parameter C controls the trade off between errors of the SVM on training data and margin maximization (C = [infinity] leads to hard margin SVM).
Rychetsky (2001)

"The parameter C controls the trade-off between the margin and the size of the slack variables."
Shawe-Taylor and Cristianini (2004)

"[Tuning the parameter C] In practice the parameter C is varied through a wide range of values and the optimal performance assessed using a separate validation set or a technique known as cross-validation for verifying performance using only a training set."
Shawe-Taylor and Cristianini (2004)

"...the parameter C has no intuitive meaning."
Shawe-Taylor and Cristianini (2004)

"The factor C in is a parameter that allows one to trade off training error vs. model complexity. A small value for C will increase the number of training errors,

while a large C will lead to a behavior similar to that of a hard-margin SVM."
Joachims (2002)

"In this paper a method based on the characteristics of the kernel, the range of output values and the size of the e-insensitive zone, is proposed."
Jordaan

"Let us suppose that the output values are in the range [0, B]. [...] a value of C about equal to B can be considered to be a robust choice."
Mattera and Haykin (1999)

"Support Vector Machines use a regularization parameter C to regulate the trade-off between the complexity of the model and the empirical risk of the model. Most of the techniques available for determining the optimal value of C are very time consuming. For industrial applications of the SVM method, there is a need for a fast and robust method to estimate C. In this paper a method based on the characteristics of the kernel, the range of output values and the size of the e-insensitive zone, is proposed."
Jordaan (2002)

B.2 Epsilon Estimation

"Smola and Kwok proposed asymptotically optimal e - values proportional to noise variance, in agreement with general sources on SVM. The main practical drawback of such proposals is that they do not reflect sample size. Intuitively, the value of e should be smaller for larger sample size than for small sample size (with same noise level)." "Optimal setting of e requires the knowledge of noise level. The noise variance can be estimated directly from training data, i.e. by fitting very flexible (high-variance) estimator to the data. Alternatively, one can first apply least-modulus regression to the data, in order to estimate noise level." "Similarly, Mattera and Haykin propose to choose e - value so that the percentage of SVs in the SVM re-

gression model is around 50% of the number of samples. However, one can easily show examples when optimal generalization performance is achieved with the number of SVs larger or smaller than 50%." "Smola and Kwok proposed asymptotically optimal e - values proportional to noise variance, in agreement with general sources on SVM The main practical drawback of such proposals is that they do not reflect sample size. Intuitively, the value of e should be smaller for larger sample size than for small sample size (with same noise level)."
Cherkassky and Ma

"For an SVM the value of e in the e-insensitive loss function should also be selected. ϵ has an effect on the smoothness of the SVM's response and it affects the number of support vectors, so both the complexity and the generalization capability of the network depend on its value. There is also some connection between observation noise in the training data and the value of ϵ. Fixing the parameter ϵ can be useful if the desired accuracy of the approximation can be specified in advance."
Horvth (2003)

"There are a number of learning parameters that can be utilized in constructing SV machines for regression. The two most relevant are the insensitivity zone ϵ and [...] Both parameters are chosen by the user. [...] An increase in ϵ means a reduction in requirements for the accuracy of approximation. It also decreases the number of SVs, leading to data compression."
Kecman (2001)

"Under the assumption of asymptotically unbiased estimators we show that there exists a nontrivial choice of the insensitivity parameter in Vapniks e-insensitive loss function which scales linearly with the input noise of the training data. This finding is backed by experimental results."
Smola Optimal Choice of e-Loss for Support Vector Machines

"The value of epsilon determines the level of accuracy of the approximated function.

It relies entirely on the target values in the training set. If epsilon is larger than the range of the target values we cannot expect a good result. If epsilon is zero, we can expect overfitting. Epsilon must therefore be chosen to reflect the data in some way. Choosing epsilon to be a certain accuracy does of course only guarantee that accuracy on the training set; often to achieve a certain accuracy overall, we need to choose a slightly smaller epsilon."

"Parameter ϵ controls the width of the e-insensitive zone, used to fit the training data. The value of e can affect the number of support vectors used to construct the regression function. The bigger e, the fewer support vectors are selected. On the other hand, bigger e-values results in more flat estimates. Hence, both C and e-values affect model complexity (but in a different way)."

"A robust compromise can be to impose the condition that the percentage of Support Vectors be equal to 50%. A larger value of ϵ can be utilized (especially for very large and/or noisy training sets)..."
Mattera and Haykin, 1999

"the optimal value of ϵ scales linearly with σ."
Learning with Kernels

B.3 KernelParam Estimation

"For classification problems, the optimal σ can be computed on the basis of Fisher discrimination. And for regression problems, based on scale space theory, we demonstrate the existence of a certain range of σ, within which the generalization performance is stable. An appropriate σ within the range can be achieved via dynamic evaluation. In addition, the lower bound of iterating step size of σ is given."
Wang, et al., 2003.

"the optimal value of ϵ scales linearly with σ."

Learning with Kernels

Bibliography

C.1 Incremental Support Vector Machines

[1] Accurate Online Support Vector Regression,
Ma, Theiler, Perkins, Article, 2003
*T*his is the article where I started my research.

[2] Incremental and Decremental Support Vector Machine Learning,
Cauwenbergh, Poggio, Article, 2001
*S*tart point of all incremental support vector algorithms.

[3] Incremental Support Vector Learning: Analysis, Implementation and Applications,
Laskov, Gehl, Krauger, Muller, Article, 2001
*A*rticle rich of demostrations about incremental classification. It includes also some implementation tricks for an efficient implementation.

[4] Online SVM Learning: from Classification to Data Description and Back,
Tax, Laskov, Article, 2003
*U*seful to understand how the algorithm converge.

C.2 Support Vector Machines

[5] A tutorial on support vector regression,
Smola, Schoelkopf, Article, 1998
*S*tart point for support vector regression.

[6] Support Vector Machines,

Taylor and Cristianini, Book, 2000

*T*heory and concepts abot support vector machines.

[7] Support Vector Machines for Classification and Regression,

Gunn, Article, 1998

*C*omplete article about support vector machines: contains theory, examples and the matlab code of the implemenetation.

[8] LubSVM - A Library for Support Vector Machines,

Chang and Lin, Web Site, 2006

*h*ttp://www.csie.ntu.edu.tw/ cjlin/libsvm/

[9] Support Vector Machines,

Grudic, Article, 2004

*I*ntroduction to support vector machines.

[10] Support Vector Machines,

Maniezzo, Article, 2004

*I*ntroduction to support vector machines.

C.3 Other

[11] UCI Machine Learning Repository,

Various authors, Web Site, 2004

*h*ttp://www.ics.uci.edu/ mlearn/MLRepository.html

[12] Wikipedia,

Various authors, Online Repository, 1998

*h*ttp://en.wikipedia.org/wiki/Learning

[13] SVM Parameters,

Various authors, Web Site, 2004

*h*ttp://www.svms.org/parameters/