Rapport de Projet de Fin d'Étude

Sylvain ROY - sroy@ensisun.imag.fr

May 23, 2003

Contents

Avertissement

Mon projet de fin d'étude a été réalisé à l'université de Canterbury en Nouvelle Zélande sous la supervisation de Brent Martin. Dû aux contraintes d'une année à l'étranger, il a été couplé avec le stage de deuxième année pour former un stage de 6 mois.

Ce document présente les travaux réalisés pendant toute la durée du stage. Il est donc sensiblement plus long que ce que recommende le reglement de l'ENSIMAG. Les raisons en sont les suivantes.

Le travail présenté dans le chapitre 1 (*The Non-Nested Generalised Exemplars Algorithm*) a déjà été présenté dans le rapport de stage de deuxième année à l'issue des deux premiers mois. Cependant, un travail important a été fourni sur ce sujet aprés cette date. Au final, le chapitre 1 représente environ 4 mois de travail alors que les chapitre 2 (*Support Vector Machines*) et 3 (*Prescription of Warfarin*) ne représentent que deux mois. Il eut alors été plus logique de présenter *The Non-Nested Generalised Exemplars Algorithm* dans le cadre du stage de PFE et de garder *Support Vector Machines* et *Prescription of Warfarin* pour le stage 2A. L'ordre dans lequel les sujets ont été abordés n'a pas permis ce choix.

J'ai donc choisi de présenter la totalité de mon travail dans ce rapport. Le lecteur qui ne s'intresse qu'à l'évaluation de mon stage de fin d'étude peut survoler le chapitre 1 pour se consacrer plus longuement au reste du document. Ma soutenance de PFE à l'ENSIMAG devrait cependant porter sur la totalité de mon travail puisque je n'ai pas effectué de soutenance pour mon stage 2A.

Enfin, du point de vu de l'université de Canterbury, pour qui il n'y a qu'un seul stage de 6 mois, ce rapport présente la totalité de mes recherches. Il assure ainsi le second rôle de rapport de stage pour le département d'informatique de l'université de Canterbury.

Introdution

Francais

Le travail présenté dans ce rapport a été réalisé dans le cadre d'un stage de recherche de six mois dans le departement d'intelligence artificielle de l'université de Canterbury (Christchurch - Nouvelle Zélande). J'ai travaillé sous la supervisation de Brent Martin. Le travail réalisé s'est découpé en plusieurs parties distinctes.

Dans un premier temps, mes recherches se sont intéressées à l'*Instance-Based Learning* (Apprentissage basé sur les instances). J'ai en particulier cherché à développer un algorithme appelé *NNGE* qui se base sur une généralisation des instances à l'aide d'hyperrectangles. Ce travail s'inscrit dans la poursuite de précédentes recherches par Martin [5]. Une implantation de l'algorithme issue de ces recherches à été réalisée pour Weka [15], un ensemble d'outils "open source" consacré au *data mining* réalisé en Java. Ces travaux sont présentés dans le chapitre 1.

Dans un deuxième temps, mes travaux se sont portés sur le champs de l'apprentissage statistique et en particulier les algorithmes appelés *Support Vector Machines*. La très récente technique de *Sequence Minimum Optimization* (Sequence d'optimisation minimum) a été utilisée dans le but de résoudre le problème quadratique inherent de manière efficace. Le programme issu de ces travaux a lui aussi été implanté dans Weka [15]. Ces travaux sont présentés dans le chapitre 2.

Enfin, ce dernier algorithme a été utilisé pour analyser une base de données concernant la prescription de la drogue Warafarin à des patients malades du coeur. Le but étant de d'essayer d'automatiser la prescription et dans un deuxième temps de modéliser le taux d'INR dans le sang du patient. Ces travaux sont présentés dans le chapitre 3.

English

The work presented in this report has been done during a six month internship in the Intelligent Tutoring Lab in the department of Computer Science at the university of Canterbury (Christchurch, New Zealand). It has been supervised by Brent Martin. The work has been composed of several parts.

The first four months have been devoted to *NNGE*, an algorithm from the field of *Instance-Based Learning. NNGE* forms hyperrectangles in order to generalise instances. This work was based on previous research by Martin [5]. An implementation of the algorithm has been realised in Java and is now part of Weka [15] an open source Machine Learning workbemch. This is presented in chapter 1.

Then, my work focused on statistical learning and in particular a kind of algorithms called *Support Vector Machines.* The new technic of *Sequence Minimum Optimization* has been used for solving the inherent quadratic problem efficiently. Again, the resulting program has been implemented in Weka [15]. This is presented in chapter 2.

Eventually, this last algorithm has been used in order to analyse the Warfarin database. This database gathers information about one patient, who uses the Warfarin drug in order to control his INR. The goal here was to try to get an automatic prescription by different methods. This is presented in chapter 3.

Chapter 1

The Non-Nested Generalised Exemplars Algorithm

1.1 Introduction

The work presented in this paper has been realized within the framework of an internship which is a part of my studies at ENSIMAG (Grenoble, France). This internship has been conducted at the University of Canterbury (Christchurch, New Zealand) and was supervised by Brent Martin.

This paper is actually a draft of what might end up in a publication. More work is still necessary in order to reach this goal. Therefore, it has been written in a way which is quite different from what one would expect from a "rapport de stage 2A".

These researches are in the continuity of two previous publications in the IBL literature.
In [12], Salzberg describes a *Nearest Hyperrectangle Learning Method* which uses parallel axis hyperrectangles. He tested his *Nested Generalised Exemplars (NGE)* method with an implementation called *EACH* and obtained good results. Moreover, *NGE* has an important advantage for an algorithm within the framework of *Instance-Based Learning* (IBL) : it decreases the amount of work needed to classify a new example. IBL based algorithm are usually slow for classification since they have to work with a large dataset of training examples.
Although Salzberg obtained good results with *EACH*, Wettschereck and Dietterich [8] found that *EACH* performs poorly when compared to the standard *Nearest Neighbour* algorithm. They provided three hypothesis explaining these results :

- nested rectangles provide a poor bias
- overlapping rectangles provide a poor bias
- the incremental search of NGE needs improvement.

The following developments are based on these three hypothesis.

We describe *Non-Nested Generalised Exemplars* (*NNGE*), a method which uses parallel axis, non overlapping, hyperrectangles. *NNGE* has been implemented and tested on well-known Machine Learning database and produced good results when compared to other classic algorithms.

The “WEKA” ML workbench [2], a mature ML evaluation tool developed at the university of Waikato, was used for this work. The *NNGE* algorithm has been implemented on this platform.

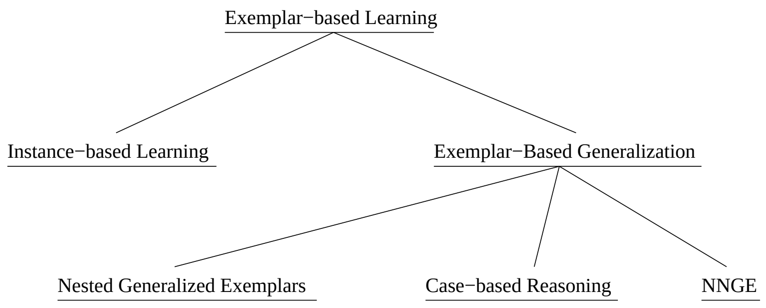

Figure 1.1: Salzberg's Exemplar-Based Learning tree

1.2 Exemplar-Based Generalization

In [12] Salzberg has proposed the hierarchy of the Figure 1.1 for the Exemplar-based learning works. Each theory in this family shares the property that it uses verbatim examples as the basis of learning (i.e. example are stored in memory without any modification).

Instance-based learning retains examples as point and never changes them. The classification of a new example is based on its similarity with one or several examples stored.

Our work, however, is based on a slightly different concept. In *Exemplar-based generalization*, the example stored in memory are not directly used for classifying new example. Instead, *generalised exemplars* are used for this operation. A *Generalised exemplar* is a higher level construction supposed to yield the concepts learnt. It is based on several examples stored in memory and can contain some additional information. *NNGE*, as *NGE*, is included in the *Exemplar-Based Learning* theory.

1.3 Generalised Exemplars

Here the term *exemplar* is used, following Salzberg's usage [12], to refer specifically to an instance that has been previously stored in computer memory in order to perform new classification. An *exemplar* may yield additional information (e.g. weight, reliability, etc). An *example* is either a training or a test instance shown to the system for the first time. Using *Generalised exemplar*, we design a group of examples gathered in one entity which is used as a whole in order to classify new instance. This follows Salzberg's usage.

In the following sections, we describe some advantages that *Generalised Exemplar* bring to *Case-Based Learning*.

1.3.1 Classification Performance

Martin [5] pointed out that large disjuncts tend to be under-represented in *Instance-based learning*'s algorithms. The problem appears when these algorithms are trained with few examples. Large disjunct are likely to have only a few examples in the training set. This may happen especially in the case of an oracle building a training set in order to represent every known rule with a small number of instances.

Martin suggests that *generalised exemplars* bring a solution for this problem. They can gather several examples from a same class in only one exemplar. An exemplar having a bigger influence on its neighborhood than the set of examples gives the disjunct a better representation. Incorrect generalised exemplars that have been created are not going to stay for long since a misclassified instance is likely to lead to re-consideration of the exemplar.

1.3.2 Classification Speed

Instance-Based learning algorithms have a very quick training speed. The amount of work during this operation being usually limited to a minimum : saving the new example. Although this is an advantage, it has a serious drawback. When trying to classify a new example, they look for the most similar learned exemplar. This may be long due to the necessity to go through the list of all the exemplars. Therefore, the classification operation is often very slow in Instance-Based Learning. In *Exemplar-Based Learning*, algorithms try to reduce the classification time by using a smaller set of generalised exemplars for classification. The idea is to transfer part of the work from the classification operation to the training operation by creating and updating a set of *generalised exemplars* during the training time. As a result, the classification operation is sped up.

1.3.3 Rule Induction

A desirable feature of a classifier is the ability to extract information from the data. This may be done via induction of rules which can be used to summarize data, perform new classification or even for analysis purpose.

1.4 Other Instance-Based Learning Issues

Instance-Based learning algorithms have to deal with several well known issues in order to perform well. As *Exemplar-Based Learning* is concerned by these issues, we describe them here.

1.4.1 Feature Relevance

Algorithms that allow relevant and irrelevant features to influence the classification with the same effect suffer from an important weakness. Each added irrelevant feature exponentially increases the quantity of training examples needed to maintain the predictive accuracy of the algorithm [9]. This problem is known as "the curse of dimensionality".

Therefore, *IBL* algorithms should be able to treat differently features with different relevance. A feature that yields a lot of information about the final class of a new example should have more effect on the classification that a random feature.

Several kind of mechanisms can be found in the IBL literature for dealing with this problem. Here are a few of them:

- feature normalization : gives the same effect to each feature, whatever their scale is. (e.g A feature spread between 0 and 1000 doesn't have more effect that a feature spread between 0 and 1.)
- feature weight : gives an important effect to relevant features and tends to minimize the effect of the irrelevant ones.
- feature selection : selects relevant features for classification, discarding the rest.

1.4.2 Noise

The presence of noise in data might appear in two different ways. Either new examples can have noisy features or they can be presented with an incorrect class. Both of them increase the difficulty of learning. Again, solutions exist in the IBL's literature.

A possibility is to perform a selection over the exemplars used for classification (e.g. *IB3*).

Another, more flexible, possibility is to give a weight to each exemplar. This weight measures the reliability of the exemplar for classification. A noisy exemplar will be given a large weight (pushing new examples away) and a relevant exemplar will be given a small weight (pulling new examples close). Small weighted exemplars are preferred to large weighted exemplars for classification. Finally, noisy exemplars' effect tend to disappear as their weight increases.

1.5 Well-known Exemplar-Based Learning algorithms

In this section we give a small description of some well-known algorithm that are close to the algorithm presented in this paper. All of them are part of the *Instance-Based Learning* field.

1.5.1 Nearest Neighbour

Nearest neighbour is probably one of the most venerable machine learning algorithms. The entire training set is stored in memory. To classify a new example, the Euclidean distance is computed between the example and each stored training example and the new example is assigned the class of the nearest neighbour example.

1.5.2 k-Nearest Neighbour

K-nearest neighbour is a variation of the previous algorithm. It uses the K nearest neighbours to classify a query. A vote among the K nearest neighbours determines the class of the new example. This behaviour usually gives better results with noisy datasets.

1.5.3 The IB family

The *IB* family is actually a series of Nearest Neighbour like algorithms which are gradually improved in order to deal with irrelevant attributes, noise, classification speed. *IB1* through *IB3* are presented in [3] and *IB4* is presented in [14]. Here is a brief overview of their characteristics :

- *IB1* is an implementation of the *nearest neighbour* algorithm.
- *IB2* is an extension of *IB1* which tries to reduce storage requirements by saving only the misclassified training examples.
- *IB3* is an extension of *IB2* which tries to improve its poorly results with noisy examples. It maintains a classification record for each saved instances. Only the examples with good classification results are used to classify new example. Notice that *IB3* does save every training example, though only a subset of them are used to classify a new example.
- *IB4* adds a mechanism for dealing with irrelevant attributes.

1.5.4 Nested Generalised Exemplars

NGE (Nested Generalised Exemplars) is the closest ancestor of *NNGE* . It has been introduced by Salzberg in [12] and is intensively studied by Wettschereck and Dietterich in [8].

NGE is an exemplar-based learning algorithm. In *NGE*, an exemplar is a single training example, and a generalised exemplar is an axis parallel hyperrectangle that may covers several training examples. These hyperrectangles may overlap or nest. The algorithm grows hyperrectangles incrementally as training examples are processed.

1.6 The Non-Nested Generalized Exemplars algorithm

This chapter introduces Non-Nested Generalised Exemplars (*NNGE*), an algorithm that generalizes exemplars without nesting or overlap.

1.6.1 *NNGE*'s bias

NNGE is an extension of *NGE* [12], which performs generalization by merging exemplars, forming hyperrectangles in feature space that represent conjunctive rules with internal disjunction. *NNGE* forms a generalization each time a new example is added to the database, by joining it to one of its nearest neighbour of the same class. Unlike *NGE*, it does not allow hyperrectangles to nest or overlap. This is prevented in two ways. First, by testing that a generalization will not lead to overlapping or nesting before it performs it. Second, by modifying any generalizations that are later found to do so.

Following Dietterich and Wetteschereck attempt to define *NGE*'s bias, the bias of *NNGE* may be defined by : *find the minimum number of non-overlapped axis-parallel hyperrectangle that correctly classifies the training data.*

It is worth noticing that even with parallel axis hyperrectangles, *NNGE* bias leads to borders that are not axis parallel. Figure 1.2 shows a very simple example of a border between two hyperrectangles.

The border is composed of straight lines (between two edges) and parabolic segments (between an edge and a corner).

1.6.2 Motivation

Wettshereck and all [8] showed that *NGE* performs poorly compared to nearest neighbour. They found that construction of overlapping hyperrectangles should be avoided.

Figure 1.3 shows a drawback of nesting and overlapping. *NGE* is trained with three instances, the first two belong to the class *cross* while the other one to the class *circle*. As we can see, the first two instances of the class *cross* lead to the creation of a hyperrectangle which covers almost the whole range of possible examples. The next instance, of the class *circle*, falls inside the hyperrectangle and is considered by *NGE* as a nested hyperrectangle.

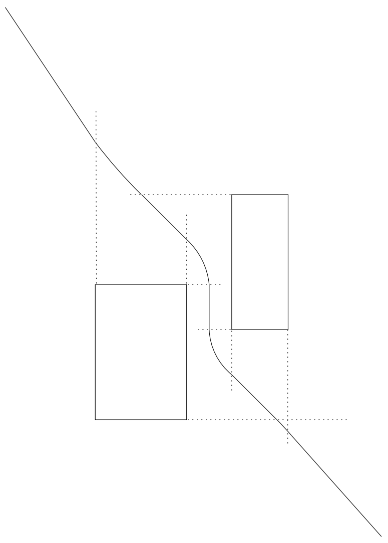

Figure 1.2: Border between two hyperrectangles.

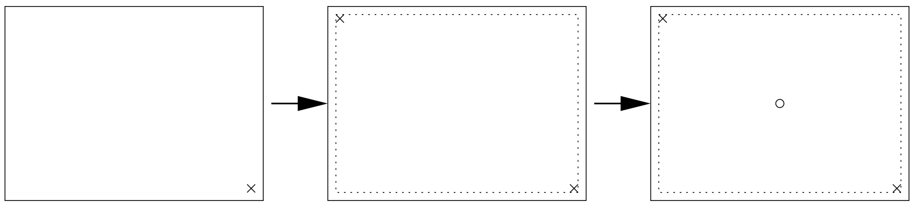

Figure 1.3: NGE's behaviour

The hyperrectangle of the class *cross* has a serious consequence ; it obstructs behaviour of the nearest neighbour of the classifier. As every new example will fall inside this hyperrectangle, the distance distance between the hyperrectangle and the new example is equal to zero. The only criteria left for classification is the heuristic developed by Salzberg. This consists of selecting the smallest hyperrectangle which includes the example. Unfortunately, this heuristic has got a short vision and lead to doubtful choices. The problem is over generalisation: examples close to a nested hyperrectangle belong to the outer one.

The Figure 1.4 shows that *NNGE* acts in a different way facing the same problem. The second example leads to a hyperrectangle covering the whole

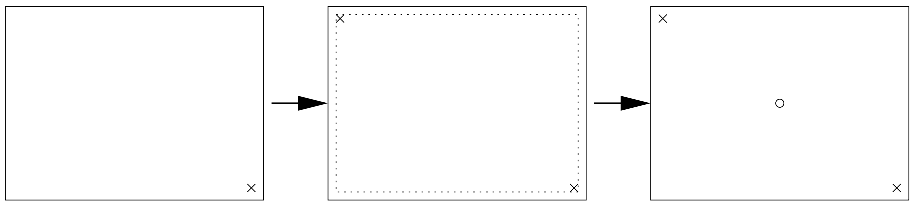
Figure 1.4: NNGE's behaviour

range of possible instance. The first instance falls inside a hyperrectangle with a different class. Then, *NNGE* 'prunes' the hyperrectangle in order to avoid any overlapping. Therefore, the nearest neighbour behaviour is conserved.

1.6.3 The *NNGE* Algorithm

Let us define some notations for the following. H will always be an hyperrectangle while E will be an example (for training or to classify). The i^{th} feature of the problem is noted f_i. Therefore, E_{f_i} is the value of the i^{th} feature of the example E and H_{upper,f_i}, H_{lower,f_i} are the upper and lower bounds of the hyperrectangle H. The maximum and minimum value of f_i are noted f_i^{max} and f_i^{min}.

E_p and E_n are two variables of each exemplar and w_{f_i} is the weight of the feature f_i. They are defined further.

```
Update($E$)
    adjust attribute ranges ($f_i^{max}$ and $f_i^{min}$)
    adjust attribute weights ($w_{f_i}$)

    Classify new example :
    $H^{closest} = closest\_Exemplar(E)$

    Adjust model :
    if($H^{closest}.class = E.class$)
        $H_p^{closest} = H_p^{closest} + 1$
    else
        $H_n^{closest} = H_n^{closest} + 1$
        if($H_{closest}$ overlaps $E$)
            $prune(H_{closest})$
    Generalize the new example :
    Try to generalize successively with one of the $n_{Max}$ closest Exemplars by order
    If none of these generalizations succeeded, create a new Exemplar with $E$
closest_Exemplar($E$)
    return the $H$ with minimum distance to $E$.
classify($E$)
    return $class(closest\_Exemplar(E))$
```

1.6.4 Classifying a new example

NNGE classifies new examples by determining the nearest hyperrectangle in the set of exemplars that has been learnt using an Euclidean distance function. This function has been slightly modified in order to include some new mechanisms. Yet, this function is similar to NGE's distance function :

$$D(E,H) = w_H * \sqrt{\sum_{i=1}^{m}\left(w_{f_i} \times \frac{d_{f_i}(E,H)}{f_i^{max} - f_i^{min}}\right)^2}$$

Where w_H and w_{f_i} are exemplar and feature weight (discussed below) and

d_{f_i} is defined by :

$$d_{f_i}(E,H) = \begin{cases} E_{f_i} - H_{upper,f_i} & \text{if } E_{f_i} > H_{upper,f_i} \\ H_{lower,f_i} - E_{f_i} & \text{if } H_{lower,f_i} > E_{f_i} \\ 0 & \text{otherwise} \end{cases}$$

However, in the special case of nominal feature, this last function is slightly different :

$$d_{f_i}(E,H) = \begin{cases} 0 & \text{if } E_{f_i} \in H_{f_i} \\ 1 & \text{otherwise} \end{cases}$$

Therefore, the distance between two nominal values is either 0 when they are equal or 1 when they are different. Compare to the numerical behaviour, this choice give the maximum possible distance to two different nominal values. We discuss w_{f_i} and w_H in the following sections.

Feature Normalization

In the distance function, the distance along an attribute is normalised with $(f_i^{max} - f_i^{min})$. This ensure that features with large scale do not have more importance than features with small scale. Either a feature with values spread between 0 and 2000 or a feature with values between 0 and 1 will have distance shrink in the interval $[0,1]$.

Another option, making the assumption that features' values follow a normal distribution, is to normalize the distance along an attribute with the standard deviation of the feature's values. However, we found that it is usually a poor choice and leads to slightly worse results.

Feature Weighting

Once features have been normalised, they have the same influence on the value of the distance function. However, as discussed in section 1.4.1, some features may be more relevant than others. Completely irrelevant features may even be present in the examples.

Giving a weight to features is a way to treat this problem. It gives a large weight to features that are relevant for classification and small weight to irrelevant features. Wettschereck et all give a general study of feature weighting methods for lazy algorithms in [16]. The methods tried for *NNGE* are largely inspired from this article and the original papers that introduced them.

Notice that in the framework of *NNGE* (i.e. an incremental learner) the method used must work with incomplete knowledge of the set of training examples : only the example already seen are known. Here are the methods implemented with *NNGE* :

NGE's Method : Salzberg [12] already used a feature weight system within *EACH*, his implementation of *NGE*. In *EACH*, features' weight are increased by setting $w_{f_i} = w_{f_i}(1 + \Delta_f)$ and decrease by setting $w_{f_i} = w_{f_i}(1 - \Delta_f)$. Assume that a new example E is misclassified by the hyperrectangle H. For each feature F_i, if E_{f_i} falls in the range of H_{f_i}, the weight of f_i (w_{f_i}) is increased; if E_{f_i} does not fall in the range of H_{f_i} the weight w_{f_i} is decreased. If E is correctly classified by H, then the feature weight is adjusted in the opposite direction.
However, Martin [5] found that this method is quite unstable and propose a slight modification. In this version, weight are adjusted only when a misclassification occurs. This is the version we tested with *NNGE*.

Mutual Information : The Mutual Information has been successfully used for feature weighting by Bakiri [6]. Wettschereck and Dietterich [8] improved the results of *NGE* by applying this method instead of the original one. Though, it is not really an incremental method, it appears possible to use it in the framework of our incremental learner. In this system, the feature weight is equal to the mutual information between the values of a feature and the class of the example. The mutual information will be 0 for a feature that doesn't give any information about the class and proportional to the log of the number of class for a feature that completely determines the class. Let us suppose that :

- $P(C = c)$ be the probability that the class of any training example equals c
- $P(f_j \in Q(i))$ be the probability that the value of feature j of any example falls into the interval $Q(i) = \left[\frac{i-1}{nIntervals}, \frac{i-1}{nIntervals}\right]$ $(i = 1, \ldots, nIntervals)$
- $P(C = c \wedge f_i \in Q(i))$ be the joint probability of these two events.

Then, the mutual information between feature f_i and the classification C is :

$$I(f_j, C) = \sum_{i=1}^{nIntervals} \sum_{c=1}^{nClasses} P(C = c \wedge f_i \in Q(i)) \ \log \left(\frac{P(C = c \wedge f_i \in Q(i))}{P(C = c)P(f_j \in Q(i))} \right)$$

For nominal features, intervals are replaced by distinct values of the feature. The probabilities are estimated from the training data and missing values are ignored. The use of mutual information raised some problems within the framework of our incremental learner. The algorithm does not have a full knowledge of the concepts to learn. In particular, it does not know what are the maximum and minimum values of each feature. Therefore, splitting the range of numerical feature in intervals cannot be done once for all the training. When a new example's value falls outside of the range defined by the previous examples on an attribute the intervals have to be re-defined. Then, the mutual information for this feature has to be computed from scratch.

IB4's method : Aha [14] uses another system in the design of IB4.

$$w_{f_i} = \max \left(\frac{CumulativeWeight_{f_i}}{WeightNormalizer_{f_i}} - 0.5, 0 \right)$$

Where $CumulativeWeight_{f_i}$ and $WeightNormalizer_{f_i}$ are updated with the following equations :

$$CumulativeWeight_{f_i} += \begin{cases} (1 - d_{f_i}(E, H)) \times (1 - \lambda(E, H)) & \text{if } class(E) = class(H) \\ d_{f_i}(E, H) \times (1 - \lambda(E, H)) & \text{otherwise} \end{cases}$$

$$WeightNormalizer_{f_i} += (1 - \lambda(E, H))$$

Where $\lambda(E, H)$ is the probability of the post probable class among E's class and H's class.

$CumulativeWeight_{f_i}$ is assumed to asymptote to half of the $WeightNormalizer_{f_i}$ for seemingly irrelevant features (i.e. the w_{f_i} is null).

It is worth noticing that this method adjusts a different set of feature weights for each class of the problem.

The mutual information has finally been chosen for feature weighting. Though it has drawbacks, the mutual information was clearly the system that led to the best results. The drawbacks are :

- The number of folds needed for computing has to be chosen arbitrarily. A poor choice can lead, very seldom, to incorrect weight values.
- The weights are computed in function of the training data only. The classifier performances are not taken into account.
- The weights computed are common for every class. (in comparison to IB4's feature weight system which has a set of feature weights for each class.)

However, the mutual information works very well with meaningless features. Those are usually given a very small weight, minimizing their influence on classification.

Exemplar reliability

In order to deal with noise, *NNGE* uses an exemplar weight system.

The idea is to favour exemplars that have given good classification. By giving a large weight to exemplars that have performed poorly *NNGE* pushes them away from new example. In contrast, by giving a small weight to exemplars that have done well it pulls them close to new example.

Records of the performance of each exemplar are kept for computing this weight. They are composed of : the number of correct classifications (p) and the number of incorrect classifications (n). Two functions have been tried for computing the exemplars' weight.

The first one defined by Salzberg [12] for *EACH* use the following equation : $w_H = \frac{p+n}{p}$

However, Martin [5] proposed another equation based on the following fact. When p is small compared to n, as it can happen at the beginning of learning session, $w_H = \frac{p+n}{p}$ is very large. Then, the exemplar is unlikely to ever make another prediction. *EACH* therefore considers only those exemplars that start well. Instead, Martin's proposed the following equation : $w_H = \frac{n}{p+n}$

Notice that for both of this system, the exemplars is assumed to give a correct classification for itself. Thus, p is initialised to 1. This avoid any division by 0 for computing w_H.

EACH system gave the best result and has therefore been chosen for the final version of *NNGE*.

1.6.5 Creating/Growing hyperrectangles

NNGE uses hyperrectangle as exemplars. For a numeric feature, the minimum and maximum values describe the range covered by the hyperrectangles. For a symbolic feature, a set of values is updated.

When generalizing an exemplar with a new example, the features that do not already overlap the new example are extended. For numeric features, either the minimum is decreased or the maximum is increased so that the range of values covers the new value. For symbolic feature, the value is added to the set of covered values.

1.6.6 Preventing overgeneralisation

Overgeneralisation is prevented in two ways :

First, when generalizing a hyperrectangle, *NNGE* will make sure that it is not going to overlap any other exemplar. In the case of overlapping, it either tries to generalize with the next closest exemplar or creates a new exemplar for this example according to the configuration of the learner.

Second, when a new example falls inside an exemplar of a different class it shrinks this exemplar so that it will no longer overlap the example. Here, one difficulty in the algorithm appears. In order to shrink the hyperrectangle, one feature must be chosen for cutting the hyperrectangle along this feature's axis.

When dealing with numeric features only, the following heuristic gives good results :

HEURISTIC A : Select the feature for which the misclassified example is the closest to the boundary along this feature's axis

Figure 1.5 illustrates this heuristic. The rectangle is the exemplar that covers all the examples represented by a cross. The circle is a new misclassi-

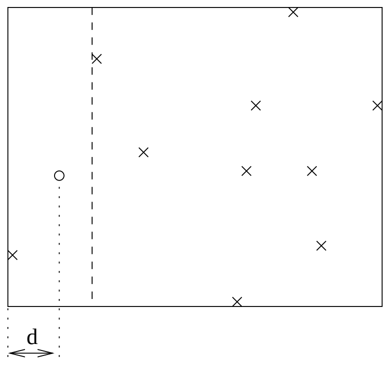

Figure 1.5: pruning heuristic ; the case of numeric features.

fied example. *NNGE* will cut the rectangle perpendicularly to the feature's axis which minimize d, creating two new rectangles. One is composed of only one example and the other is drawn by the dashed border.

This heuristic is based on the observation that in the case of fuzzy border, pushing the exemplar away from the border is probably the action which makes the more sense. Generalised exemplars will gather examples far from the border where there is no real problem for classification. Ungeneralised or lightly generalised exemplars will stand along the border where more accurate classification is needed.

When dealing with nominal features only, this heuristic does not make sense anymore. Because nominal features do not have any order, any nominal feature's value is "on the border" of the hyperrectangle.

Moreover, cutting a hyperrectangle along a nominal feature is a different operation. The hyperrectangle is not cut in two part like in the case of a numeric feature. The number of feature values is finite and each value can be removed individually. Therefore, cutting a hyperrectangle along a nominal feature consists only of removing a specific value along the given axis. We used the following heuristic :

HEURISTIC B : Select the feature for which the feature-value to be removed matches the hyperrectangle the least well

In other words, *NNGE* counts for each feature the number of examples in the exemplar that have the same value as the conflicting example. The feature which has the smallest count will be cut.

The problem is more complicated when working with both numerical and nominal features. The following heuristic has the enormous advantage of being valid on both kind of features :

HEURISTIC C : Select the feature such that cutting along this feature will minimize the number of examples removed from the hyperrectangle.

This heuristic corresponds to the previous one when used with nominal features only. Unfortunately, for numeric features a comparison of this heuristic with a random choice of the feature to cut showed that it performs pretty poorly.

Eventually, the heuristic D, which is nothing else that an attempt to keep good results with both numerical and nominal features, has been selected.

HEURISTIC D : Select the best numerical feature with heuristic A. Then select the best feature to cut from the set composed of the best numerical feature and of all the nominal features with heuristic C.

1.7 Evaluation

1.7.1 Test domains

Five datasets have been used for testing Nnge. They were all obtained from the from the UCI machine learning data repository [7]. The following paragraphs summarize each dataset used.

- **Iris:** This is perhaps the best known database to be found in the pattern recognition literature. Each instance has 4 numeric attributes. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

- **C-heart:** The dataset has 303 examples. The examples in this database represent heart disease patients described by 13 attributes. Attributes are both numeric and nominal. The class identifies the presence of heart disease in the patient. Some values are missing.

- **Led24:** This artificial domain contains examples representing the set of ten decimal digits on an LED display. Each example contains 24 boolean attributes - the seven segments of an LED display and seventeen irrelevant attributes - and the class - a integer in the interval $[1, 10]$. Furthermore, a noise is added: each attribute, excluding the class, has a 10% chance of being inverted. There is no missing values.

- **Waveform:** As Led24, this domain is artificial. Each example represents an artificially created waveform described by 21 numeric values, with 10% noise added. The dataset has three classes. There is no missing value.

- **Breast-Cancer:** This is one of three domains provided by the Oncology Institute that has repeatedly appeared in the machine learning literature. This data set includes 201 instances of one class and 85 instances of another class. The instances are described by 9 attributes, some of which are linear and some are nominal. There is missing value.

1.7.2 NNGE vs other IBL algorithms

The following table gives the results of tests on *NNGE* and three other *Instance-Based Learning* algorithms. The first number is the accuracy of the classification (percentage of correctly classified examples). The second one is the number of exemplars generated divided by the number of examples of the dataset. The tests have been done using 10 fold cross validation.

Several couples of values for *NNGE* correspond to experiments with different values of the maximum number of attempts of generalization.

	IB3	*IB4*	*NGE*	*Nearest Neighbour*	*NNGE*
iris			95.3/12	95.3/100	**96.6/6.6**
c − heart	77.9/18.6			76.2/100	**82.2/25.4** *or* 77.9/20.4
led24	45.8/25.8	**68.9/21.6**		48/100	68.2/37.8
waveform	**74.7/14.1**			72/100	74/24.8
breast − cancer			**77.6/8**	64.3/100	67.5/32.5

These tests need to be improved. As we do not have any implementation of *IB3* and *NGE*, the results above have been collected on the original publication. Although the experiments have been reproduced as close as possible as the original one with *Nearest Neighbour* and *NNGE*, the results might be slightly different. More serious experiments are planned in order to have a good comparison of *NNGE* with these algorithms.

However, it is still possible to see that *NNGE* performs quit well.

1.7.3 Nnge

The final version of *NNGE* lets the user choose the number of attempts at generalisation. It corresponds to the number of times the algorithm will try to generalise a new example with an existing exemplar. It starts with the closest exemplar and then tries with the next closest one until it either succeeds or reaches this limit value.

This parameter is mainly influential for the memory requirement of *NNGE*. The bigger it is, the less exemplars *NNGE* creates. However, it has influence on the classification performances, too.

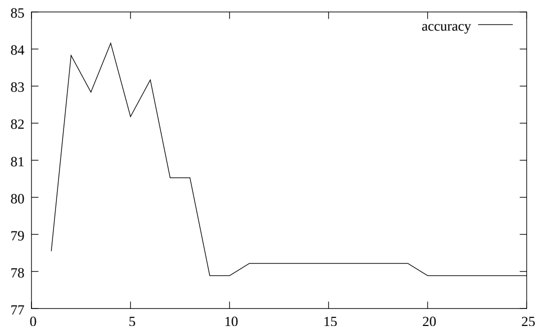

Figure 1.6: accuracy as a function of the number of attempts of generalization

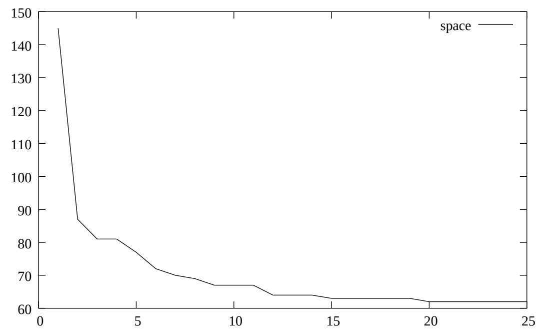

Figure 1.7: space as a function of the number of attempts at generalization. The y axis corresponds to the number of exemplars generated. The data set has 303 instances.

The Figure 1.6 and 1.7 are the results of test made with the *Cleveland Heart Disease* dataset from the *UCI-Repository* [7] which is composed of 303 examples. Though specific to this dataset, they are really general in their shapes and give a good idea of the effect of this parameter on classification performance (Figure 1.6) and space requirements (Figure 1.7).

It is worth noting that five attempts of generalisation seems to be a good value in most of the cases.

1.8 Perspectives - Conclusion

NNGE's performance showed that it performs well compared to state of the art algorithm like *IB3*.

The algorithm obtains good classification results. Further, it creates a number of exemplars which is usually around a third of the number of training examples. Therefore, the classification time is smaller, at the cost of a slightly longer training time.

It is worth noting that a batch algorithm realised for this research lead to results that were only slightly better. A batch algorithm knows all training examples from the beginning and therefore can learn the concept with full knowledge of the data. Then, this algorithm does not have to prune any hyperrectangles since it never overgeneralises. The fact that the results are not a lot better leads us to think that *NNGE*'s search algorithm is efficient.

NNGE has really good behaviour with noisy datasets and datasets containing meaningless features. This is mainly due to the use of the Mutual Information as a feature weight system.

However, *Nnge* still suffers from one important weakness: It does not handle missing values in a satisfactory way.
The basic heuristic, which consists of replacing missing value with a specific value, is not really satisfactory for several reasons.

First, it assumes knowledge of the data that may be incorrect: This operation should be done only if a missing value yields information.

Second, this heuristic is not completely defined for real value attributes. The choice of a numeric value to use for missing numeric values is still undefined.

Another solution is to accept hyperrectangle with missing borders, covering the whole range of value along a feature. However, this lead to inconsistent situations in *Nnge*'s bias.

This last problem needs further research in order to determine a proper way to deal with missing values.

Other future research might also include another shape for Exemplars (hyper-ellipse, convex hull).

Chapter 2

Support Vector Machines

2.1 Introduction

In the last few years, there has been a surge of interest in Support Vector Machines. SVMs have been shown to give good generalization performance on a wide variety of problems such as handwritten character recognition, face detection, pedestrian detection and text categorization.

Vladimir Vapnik invented Support Vector Machines in 1979 [1]. In its simplest, linear form, an SVM is a hyperplane that separates a set of positive examples from a set of negative examples with maximum margin (see figure 2.1). In the linear case, the margin is defined by the distance of the hyperplane to the nearest of the positive and negative examples.

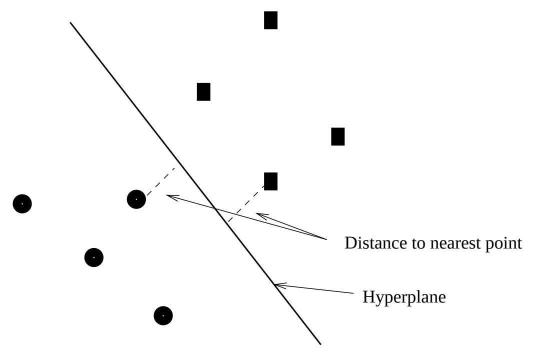

Figure 2.1: A linear Support Vector Machine

Further improvements led to deal with non-linear cases, error. Eventually, SVM has been extended to perform regression.

In this chapter, the theory of SVM for regression is briefly explained. Then SMO, an efficient algorithm for training SVM, is introduced. Although SMO is supposed to be easy to implement, it has been found that small mistakes, lacks of precision or inconsistencies get it harder. Therefore, the last section discusses these issues.

2.2 Theory

2.2.1 The linear case

Suppose we are given training data $\{(x_1, y_1), ..., (x_l, y_l)\} \subset \mathcal{X} \times \mathbb{R}$, where $\mathcal{X}$ denotes the space of input patterns - for instance, $\mathbb{R}^d$. In ε-SV regression (Vapnick [1]), the goal is to find a function $f(x)$ that has at most ε deviation from the actually obtained targets y_i for all the training data, and at the same time, is as flat as possible. In other words, we do not care about errors as long as they are less than ε, but will not accept any deviation larger than this.

Let us begin with the case of linear functions f, taking the form:

$$f(x) = \langle w, x \rangle + b \text{ with } w \in \mathcal{X}, b \in \mathbb{R} \tag{2.1}$$

Flatness in the case of (2.1) means that one seeks small w. One way to ensure this is to minimize $\|w\|^2$. The ε-insensitivity is expressed by the following inequations $f(x_i) - \varepsilon \leq y_i \leq f(x_i) + \varepsilon$. Then, the problem can be written as a convex optimization problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|w\|^2 \\ \text{subject to} & \left\{ \begin{array}{lll} y_i - \langle w, x_i \rangle - b & \leq & \varepsilon \\ \langle w, x_i \rangle + b - y_i & \leq & \varepsilon \end{array} \right. \end{array} \tag{2.2}$$

The tacit assumption in (2.2) is that a function f, which approximates all pairs (x_i, y_i) with ε precision, actually exists. One can introduce slack variables ξ_i, ξ_I^* to cope with infeasible constraints of the optimization problem. This gives us the formulation stated in [1].

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ \text{subject to} & \left\{ \begin{array}{lll} y_i - \langle w, x_i \rangle - b & \leq & \varepsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i & \leq & \varepsilon + \xi_i^* \\ \xi, \xi* & \geq & 0 \end{array} \right. \end{array} \tag{2.3}$$

The constant $C > 0$ determines the trade off between the flatness of f and the amount up to which deviations larger than ε are tolerated. This formulation uses an *varepsilon*-insensitive loss function. described by

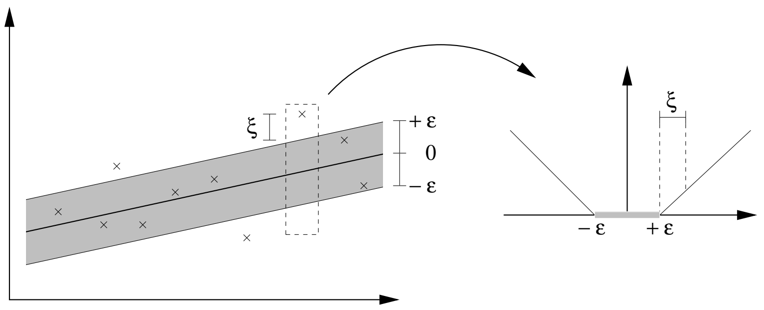

Figure 2.2: The soft margin loss.

$$|\vartheta|_\varepsilon := \begin{cases} 0 & \text{if } |\vartheta| \leq \varepsilon \\ |\vartheta| - \varepsilon & \text{otherwise} \end{cases} \tag{2.4}$$

The figure (2.2) depicts the situation graphically. Only the points outside the shaded region contribute to the cost.

It turns out that the optimization problem (2.3) can be solved more easily in its dual formulation. The Lagrangian of the problem is

$$\begin{aligned} L := \ & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) - \sum_{i=1}^{l}\alpha_i(\varepsilon + \xi_i - y_i + \langle w, x_i\rangle + b) \\ & - \sum_{i=1}^{l}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle w, x_i\rangle - b) - \sum_{i=1}^{l}(\eta_i\xi_i + \eta_i^*\xi_i^*) \end{aligned} \tag{2.5}$$

where $\alpha_i, \alpha_i^*, \eta_i, \eta_i^*$ are the Lagrange multiplier and are subject to $\alpha_i, \alpha_i^*, \eta_i, \eta_i^* \geq 0$. It follows from the saddle point condition that the partial derivatives of L with respect to the primal variables (w, b, ξ_i, ξ_i^*) have to vanish for optimality.

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{l}(\alpha_i^* - \alpha_i) = 0 \tag{2.6}$$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)x_i = 0 \tag{2.7}$$

$$\frac{\partial L}{\partial \xi_i^{(*)}} = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0 \tag{2.8}$$

Substituting (2.7), (2.8) and (2.8(in (2.5) yields the dual optimization problem.

$$\begin{array}{ll} \text{maximize} & \left\{ \begin{array}{l} -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle x_i, x_j\rangle \\ -\varepsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) \end{array} \right. \\ \text{subject to} & \left\{ \begin{array}{lcl} \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) & = & 0 \\ \alpha_i, \alpha_i^* & \in & [0, C] \end{array} \right. \end{array} \tag{2.9}$$

The equation (2.1) can be rewritten as follows

$$w = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)x_i \text{ and therefore } f(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\langle x_i, x\rangle + b \tag{2.10}$$

The Karush-Kuhn-Tucker conditions state that that at the optimal solution the product between dual variables and constraints has to vanish. In our case, it means

$$\begin{array}{c} \alpha_i(\epsilon + \xi_i - y_i + \langle w, x_i\rangle + b) = 0 \\ \alpha_i(\epsilon + \xi_i + y_i + \langle w, x_i\rangle - b) = 0 \end{array} \tag{2.11}$$

and

$$\begin{array}{c} (C - \alpha_i)\xi_i = 0 \\ (C - \alpha_i^*)\xi_i^* = 0 \end{array} \tag{2.12}$$

It gives us some useful conclusions. First, $\alpha_i\alpha_i^* = 0$ since both constraints in (2.11) cannot be equal to zero at the same time. Secondly, only samples (x_i, y_i) with corresponding $\alpha_i^{(*)} = C$ lie outside the ε-insensitive tube around f. Finally, for $\alpha_i^{(*)} \in]0, C[$ we have $\xi_i^{(*)} = 0$ which give us a way to compute b with (2.11).

2.2.2 The non-linear case

In order to deal with non-linear regression one can simply map $\mathcal{X}$ into some feature space $\mathcal{F}$ and then apply the SV regression. Let us call this mapping $\phi : \mathcal{X} \to \mathcal{F}$.

The problem is that one usually wants to use a feature space $\mathcal{F}$ of higher dimension. Working in this space lead to more computing and might even be computationally infeasible.

However, one can notice that the optimization problem depends of the data through the scalar product only. Therefore, if it exists a function k - called a kernel - such as $k(x_i, x_j) := \langle \phi(x_i), \phi(x_j) \rangle$, the computing can be done in the original feature space $\mathcal{X}$. This avoid the expensive computing in $\mathcal{F}$ which might be of really high dimension. Furthermore, we don't even need to know ϕ. The modification of the linear case is then straightforward and leads to :

$$\begin{array}{ll} \text{maximize} & \left\{ \begin{array}{l} -\frac{1}{2} \sum\limits_{i,j=1}^{l} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) k(x_i, x_j) \\ -\varepsilon \sum\limits_{i=1}^{l} (\alpha_i + \alpha_i^*) + \sum\limits_{i=1}^{l} y_i (\alpha_i - \alpha_i^*) \end{array} \right. \\ \text{subject to} & \left\{ \begin{array}{lcl} \sum\limits_{i=1}^{l} (\alpha_i - \alpha_i^*) & = & 0 \\ \alpha_i, \alpha_i^* & \in & [0, C] \end{array} \right. \end{array} \tag{2.13}$$

with

$$w = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \phi(x_i) \text{ and therefore } f(x) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) k(x_i, x) + b \tag{2.14}$$

2.2.3 Kernels

One can now choose a kernel without knowing about ϕ. However, a function must satisfy Mercer's condition in order to be suitable as a kernel. It means that if

$$\int_{\mathcal{X} \times \mathcal{X}} k(x, x') f(x) f(x') \, dx \, dx' \geq 0 \text{ for all } f \in \mathcal{L}_2(\mathcal{X}) \tag{2.15}$$

holds we can write $k(x, x')$ as a dot product in some feature space. Here is a list of kernels that have been implemented.

- Polynomial regression of order p:

$$k(x, x') = \langle x, x' \rangle^p \text{ with } p \in \mathbb{N} \tag{2.16}$$

- Polynomial regression of order p with lower order terms:

$$k(x,x') = (\langle x,x' \rangle + 1)^p \text{ with } p \in \mathbb{N} \tag{2.17}$$

- RBF-kernel:

$$k(x,x') = \exp^{-\frac{\|x-x'\|^2}{2\sigma^2}} \text{ with } \sigma \in \mathbb{R} \tag{2.18}$$

  It is worth to notice that $dim(\mathcal{F}) = \infty$ here.

2.3 Sequence Minimum Optimization

Although simplified in its last form (2.13), the optimization problem is still hard to solve. The number of dual variables (α_i, α_i^*) is equal to twice the number of training instances. As a simple Machine Learning dataset is usually composed of a large number of instances, it is necessary to have a quick and efficient way to solve the quadratic problem. SMO has been proposed by Platt [10] as a fast algorithm for training Support Vector Machines for Pattern recognition. Smola modified the algorithm for training Support Vector Machines for regression in [13]. Finally, Keerthi & all proposed in [11] a improved version of Smola's algorithm that they found to be much faster. This last version is the one which has been implemented and which is presented below.

The main idea of SMO is to decompose the optimization problem in to smaller problems that are solved sequentially. This was already done by many algorithm using *chuking*. However, SMO goes further since it breaks the quadratic problem in to a series of smallest possible QP problems. That is, only two pairs of dual variables are computed at each iteration. The key point is those small problems can be solved analytically, which avoids using a time consuming numerical QP optimization.

In the case of a regression, SMO is solving the problem (2.13). At each iteration, the problem is solved for only two indices, say (i,j). Other dual variables are considered constant.

Let us defined φ_i, the error made by the current hypothesis at sample x_i

$$\varphi_i := y_i - f(x_i) = y_i - \left[\sum_{j=1}^{l} (\alpha_j - \alpha_j^*) k(x_i, x_j) + b \right] \tag{2.19}$$

and the shorthand v_i

$$v_i = y_i - \sum_{a \neq i,j} (\alpha_a - \alpha_a^*) K_{ia} - b = \varphi + (\alpha_i^{old} - \alpha_i^{*old}) K_{ii} + (\alpha_j^{old} - \alpha_j^{*old}) K_{ij} \quad (2.20)$$

where the superscript *old* indicates the value from the previous iteration. The first constraint of (2.13) gives us another constant :

$$\gamma := (\alpha_j - \alpha_j^*) + (\alpha_i - \alpha_i^*) = (\alpha_j^{old} - \alpha_j^{*old}) + (\alpha_i^{old} - \alpha_i^{*old}) \quad (2.21)$$

Then, (2.13) restricted to (i, j) can be rewritten as follows:

$$\begin{array}{ll} \text{maximize} & \left\{ \begin{array}{l} -\frac{1}{2} \begin{pmatrix} \alpha_j - \alpha_j^* \\ \alpha_i - \alpha_i^* \end{pmatrix}^\mathsf{T} \begin{pmatrix} K_{jj} & K_{ji} \\ K_{ij} & K_{ii} \end{pmatrix} \begin{pmatrix} \alpha_j - \alpha_j^* \\ \alpha_i - \alpha_i^* \end{pmatrix} \\ +v_j(\alpha_j - \alpha_j^*) + v_i(\alpha_i - \alpha_i^*) - \varepsilon(\alpha_j + \alpha_j^* + \alpha_i + \alpha_i^*) \end{array} \right. \\ \text{subject to} & \left\{ \begin{array}{l} (\alpha_j - \alpha_j^*) + (\alpha_i - \alpha_i^*) = \gamma \\ \alpha_j, \alpha_j^*, \alpha_i, \alpha_i^* \in [0, C] \end{array} \right. \end{array} \quad (2.22)$$

Because we have $\alpha_i \alpha_i^* = \alpha_j \alpha_j^* = 0$, only one multiplier for each indice is actually modified, the other being null. Therefore, we have 4 different cases: $(\alpha_i, \alpha_j), (\alpha_i, \alpha_j^*), (\alpha_i^*, \alpha_j), (\alpha_i^*, \alpha_j^*)$.

The figure (2.3) depicts the constraints of (2.13) on the two multipliers being optimized. One can see that SMO must find an optimum on a diagonal segment.

The table (2.3) gives the values of L and H the maximum and minimum of $\alpha_i^{(*)}$ based on the constraints in (2.13).

Finally, using the summation constraint (2.13) can be rewritten as follow. (Terms independent of $\alpha_i^{(*)}$ have been ignored.)

$$\begin{array}{ll} \text{maximize} & \left\{ \begin{array}{l} -\frac{1}{2}(\alpha_i - \alpha_i^*)^2 (K_{jj} + K_{ii} - 2K_{ji}) - \varepsilon(\alpha_i + \alpha_i^*)(1 - s) \\ +(\alpha_i - \alpha_i^*)(v_i - v_j - \gamma(K_{ji} - K_{jj}) \end{array} \right. \\ \text{subject to} & \left\{ \ \alpha_i, \alpha_i^* \in [L, H] \right. \end{array} \quad (2.23)$$

Where $s = 1$ for α_j, α_i and α_j^*, α_i^* and 0 otherwise.

This optimization problem is easily solved analytically. Let us defined the shorthand $\eta = K_{ii} + K_{jj} - 2 * K_{ij}$. If the kernel respects Mercer's condition,

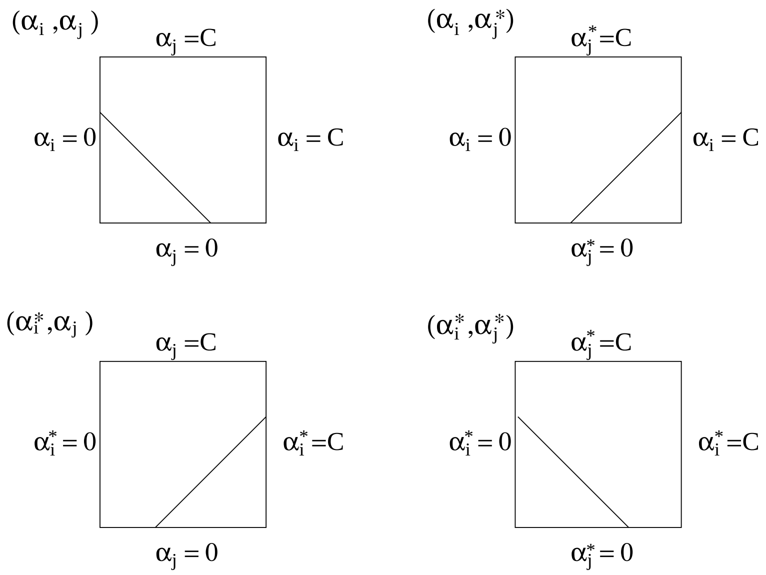

Figure 2.3: The constraints of the two Lagrange multipliers. The second constraint in (2.13) causes the Lagrange multipliers to lie in the box. The first constraint causes them to lie on a diagonal line.

	α_j	α_j^*
α_i	$L = \max(0, \gamma - C_j)$ $H = \min(C_i, \gamma)$	$L = \max(0, \gamma)$ $H = \min(C_i, C_j^* + \gamma)$
α_i^*	$L = \max(0, -\gamma)$ $H = \min(C_i^*, -\gamma + C_j)$	$L = \max(0, -\gamma - C_j^*)$ $H = \min(C_i^*, -\gamma)$

Figure 2.4: $\alpha_j^{(*)} \in [0, C_j^{(*)}]$ yields $\alpha_i^{(*)} \in [L, H]$

α_i, α_j	$\alpha_i = \frac{v_i - v_j - \gamma(K_{ji} - K_{jj})}{\eta} = \alpha_i^{old} + \frac{\varphi_i - \varphi_j}{\eta}$
α_i, α_j^*	$\alpha_i = \frac{v_i - v_j - \gamma(K_{ji} - K_{jj}) - 2\varepsilon}{\eta} = \alpha_i^{old} + \frac{\varphi_i - \varphi_j - 2\varepsilon}{\eta}$
α_i^*, α_j	$\alpha_i^* = \frac{v_j - v_i + \gamma(K_{ji} - K_{jj}) - 2\varepsilon}{\eta} = \alpha_i^{*old} + \frac{\varphi_i - \varphi_j + 2\varepsilon}{\eta}$
α_i^*, α_j^*	$\alpha_i^* = \frac{v_j - v_i + \gamma(K_{ji} - K_{jj})}{\eta} = \alpha_i^{*old} - \frac{\varphi_i - \varphi_j}{\eta}$

Figure 2.5: Iteration on $\alpha_i^{(*)}$ when $\eta > 0$

α_i, α_j	$\psi(\alpha_i) = \alpha_i(\varphi_i - \varphi_j)$
α_i, α_j^*	$\psi(\alpha_i) = \alpha_i(\varphi_i - \varphi_j - 2\varepsilon)$
α_i^*, α_j	$\psi(\alpha_i^*) = -\alpha_i^*(\varphi_i - \varphi_j + 2\varepsilon)$
α_i^*, α_j^*	$\psi(\alpha_i^*) = -\alpha_i^*(\varphi_i - \varphi_j)$

Figure 2.6: Objective function to maximize on $\alpha_i^{(*)} \in L, H$ when $\eta = 0$.

we have $\eta \geq 0$. Then two cases are possible. First, $\eta > 0$ and the new value of $\alpha_i^{(*)}$ can be computed using table (2.3). Second, $\eta = 0$ and the new value of $\alpha_i^{(*)}$ is either H or L. The value which maximizes the objective function expressed in (2.3) is chosen.

Finally, the following equation - derived using (2.21) - is useful to update $\varphi_i^{new} - \varphi_j^{new}$.

$$\varphi_i^{new} - \varphi_j^{new} = \varphi_i^{old} - \varphi_j^{old} - \eta((\alpha_i^{new} - \alpha_i^{*new}) - (\alpha_i^{old} - \alpha_i^{*old})) \qquad (2.24)$$

2.4 Implementation

SMO for Support Vector Regression has been implemented in Weka ans is already available in the latest version. The algorithm is called SMOreg and is in the package `weka.classifiers.functions.supportVector` along with SMO a similar algorithm which performs classification.

SMOreg has been implemented with three different kernels:

- Polynomial kernel : $K(x, y) = \langle x, y \rangle^e$
- Polynomial kernel with low order terms : $K(x, y) = (\langle x, y \rangle + 1)^e$

- RBF kernel : $K(x, y) = e^{-\gamma \langle x-y, x-y \rangle^2}$

In order to improve the speed of the algorithm, a cache has been implemented for the kernels. That is, a kernel evaluation is computed only once.

The data are usually normalized before being used with SMO. This is necessary in order to ensure a good behaviour of SMO with all scale of value. However, an option allows to choose between normalization, standardization or nothing.

The algorithm has the following options:

- -S value : The value of ε
- -C value : The value of C
- -E value : The value of e if not RBF kernel
- -G value : The value of γ if RBF kernel
- -N 0,1,2 : Whether to 0=normalize, 1=standardize, 2=neither
- -F : Turn on feature-space normalization (only for non-linear polynomial kernels)
- -O : Use lower-order terms (only for non-linear polynomial kernels)
- -R : Use RBF kernel
- -A value : The size of the kernel cache
- -T value : The tolerance parameter.
- -P value : the epsilon round-off error

2.5 Results

SMOreg has been tested on a large number of dataset during its development. The datasets from the UCI repository [7] have mainly been used. It performed very well compared to the other algorithms in Weka and is often the most accurate algorithm without any special setting.

It is worth noticing that SMOreg is quite fast. Though the intensive computing, the Sequence Minimum Optimization performs the optimization at great speed.

Finally, two datasets, used for the first time, have been used with SMO. The next chapters present the results of these experiments.

2.6 Problems met during implementation

Our implementation of SMO is based on the first improvement presented by Shevade et all [11]. Therefore, the following papers have been used in the process of understanding the algorithm:

- the first presentation of SMO by Platt in [10]
- the first presentation of SMO for regression by Smola in [13] and [4].
- improvement of SMO for regression by Shevade et all in [11].

In [11], Shevade et all propose an improvement of the algorithm introduced by Smola in [13]. Therefore, they present the same pseudo-code with some modifications in some critical parts. (They also corrected some small mistakes of Smola's pseudo-code.)

As a consequence, their pseudo-code is sometimes based on Smola's notation and sometimes on their own. Then a line of pseudo-code like

```
update deltaphi
```

is sometime quite hard to interpret. What is deltaphi ? In which paper is it defined ? The situation is even worse when different papers define a variable in a different way.

As a result, one has to make all the calculus from scratch by himself to figure out what to do. In a way, this is fine: it is probably the best way to understand the algorithm.

In this section, those ill-documented calculus are presented. The pseudo-code of the algorithm, using the notation of this paper, is presented in annexe. In the next sub-section, ill-documented peaces of code are detailled.

It is worth to note that in the following, like in the paper listed above, i and j correspond respectively to the indices 2 and 1 in the pseudo-code.

2.6.1 maximizing the objective function when $\eta = 0$

```
% We assume that eta > 0. Otherwise one has to repeat the complete
% reasonning similarly (i.e. compute objective functions at L and H
% and decide which one is the largest.)
```

The case eta = 0 does not happen often. However, since it may happen the algorithm has to cope with this situation. This claculus doesn't appear in either [4] or [11].

The objective function is defined by the equation (82) of [4]:

$$\psi = -\frac{K_{ii} + K_{jj} - 2K_{ij}}{2}(\alpha_i - \alpha_i^*)^2 - \varepsilon(\alpha_i + \alpha_i^*)(1-s) + (\alpha_i - \alpha_i^*)(v_i - v_j - \gamma(K_{ij} - K_{jj})$$

Then the equation (80) of the [4] and the fact that $\eta = -(K_{ii} + K_{jj} - 2K_{ij}) = 0$ lead to

$$\psi = -\varepsilon(\alpha_i + \alpha_i^*)(1-s) + (\alpha_i - \alpha_i^*)(\varphi_i - \varphi_j)$$

As for the case where $\eta > 0$ four cases are possible, there are resumed in the table 2.3. The cases $\eta = 0$ and $\eta > 0$ can be treated together in the code since the four cases are similar.

2.6.2 Computing L and H

```
compute L, H (w.r.t. alpha1, alpha2)
compute L, H (w.r.t. alpha1', alpha2)
compute L, H (w.r.t. alpha1, alpha2')
compute L, H (w.r.t. alpha1', alpha2')
```

Those lines refer to the results in tables 2.3 and 2.3. The value of η decides which table has to be used.

2.6.3 Updating deltaphi

```
update deltaphi
```

This can be done using equation (83) of [4]. However, it is important to note that the definitions of η in this equation and in the pseudo-code are different: $\eta = -eta$.

2.6.4 Updating the cache and the alphas' arrays

```
Update f-cache[i] for i in I_0 using new Lagrange multipliers
```

`f-cache[k]` yields $F_k = d_k - \sum_l (\alpha_l - \alpha_l^*)\langle z_l, z_k\rangle$ (defined in [11]). Then, considering that $\alpha_i, \alpha_i^*, \alpha_j, \alpha_j^*$ are the only alphas whose values are different, a few lines of computing lead to:

$$F_k - F_k^{old} = ((\alpha_j^{old} - \alpha_j^{*old}) - (\alpha_j - \alpha_j^*))K_{jk} + ((\alpha_i^{old} - \alpha_i^{*old}) - (\alpha_i - \alpha_i^*))K_{ik}$$

2.6.5 Computing F_2

```
compute F2 = F_i2 and set f-cache[i2] = F2
```

This can be done easily with the definition of F_i in [11]:

$$F_k = d_k - \sum_l (\alpha_l - \alpha_l^*)K(x_k, x_l)$$

2.6.6 A complete implementation

Our implementation of SMO is open source. It is part of Weka, a Java software that can be downloaded for free at `http://www.cs.waikato.ac.nz/~ml/`. Two algorithms are actually implemented:

- SMO for classification in weka.classifiers.functions.supportVector.SMO.java
- SMO for regression in weka.classifiers.functions.supportVector.SMOreg.java. (discussed here)

Chapter 3

Prescription of Warfarin

3.1 Presentation of the problem

Drug prediction has long been an interesting and yet frustrating area of research in the medical field. The variations in patient metabolism, the interactions between modification and a number of other external effects causes many difficulties for the accurate prediction of the effects of medication. All of these difficulties are experienced when trying to predict Warfarin, which is primarily used as an anticoagulant in patients with artificial heart valves, to maintain a consistent thickness of the blood.

Currently, the blood thickness levels of patients on Warfarin are tested every few weeks, or even days. Their results are sent to their cardiologist or their family doctor, who uses a set of guidelines or an ad-hoc approach in an attempt to accurately prescribe the drug, aiming to maintain a target range for their blood reading. Inaccurate prediction of the effects and therefore prescriptions of the drug can cause a number of undesirable consequences, potentially resulting in clotting on the patient's valve if the blood becomes too thick, or alternatively, thinning the blood too much, resulting in serious hemorrhaging.

Within this framework, two tasks have been considered. The first one consisted of in learning to prescribe the Warfarin drug. That is, the prescription of the doctor is considered as the *correct answer*. Therefore, it is important to note that the algorithm learnt how to imitate the doctor and not how to optimize the prediction.

The second task took the opposite point of view. The algorithm was asked to predict the next blood reading of the patient given that previous blood readings and prescriptions. This time, the algorithm is asked to learn how the patient's metabolism is reacting to the drug. Although more useful, this problem proved to be much harder.

3.2 The dataset

The Warfarin dataset is the first *real world* dataset that has been tested with our implementation of a *Support Vector Regression*. As usual, nothing is so easy in the real world. The Warfarin dataset quickly showed that things were going to be harder…

Concretely, the dataset was composed of two photocopies of what were probably already photocopies (of photocopies…) of the records of the prescriptions/blood readings of one patient. Any entry in this dataset was composed of three fields:

- Date: the date when the entry has been made
- INR: International Normalised Ratio. It gives an information on the current thickness of the blood.
- Warfarin: the prescription made by the doctor with the following information.

The records spread on a duration of 8 years, with a period between two records being between a day and a few weeks. However, 10 months was missing.

Here are some characteristics of the dataset:

- the dataset was noisy. (It is worth noticing that wherever you are in the world, doctors manage to have an awful writing!)
- some values were missing.
- the number of instances was quite small (160 entries).

In order to use this dataset, some transformations have been applied on the dataset.

First, the missing values have been replaced by hand. Our implementation of the *Support Vector Regression* could have done a similar operation, but without any knowledge on the data, resulting in lost consistency. A missing date has been replaced such as being right in the middle between the previous and the next record's date. A missing prediction has been replaced by the prediction in the closest situation in the dataset. No INR value were missing.

Finally, the format of the dataset has been changed. Considering that the information of the short term history of the patient may yield useful information, one might want to have records yielding those values. Then, the record's format has been changed with the following fields:

- Warfarin: current Warfarin prescription. (This attribute has been removed in the second task.)
- INR: current INR value

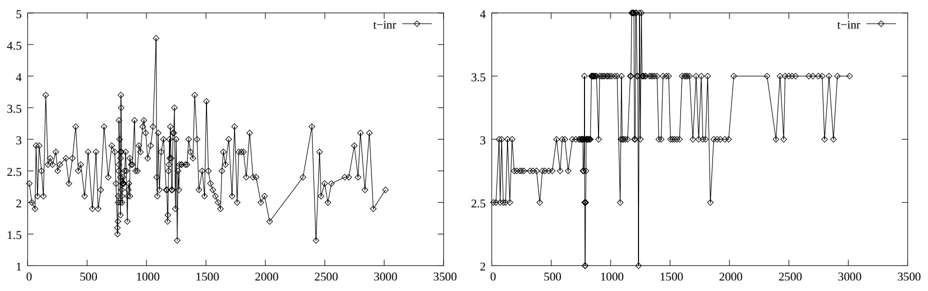

Figure 3.1: INR and Warfarin predictions evolutions in time (days).

- date: date of the current record
- Warfarin1: Warfarin prescription of the previous record.
- INR1: INR value of the previous record.
- deltaT1: Time (in days) since the previous record.
- Warfarin2: Warfarin prescription of the record before the previous one.
- INR2: INR value of the record before the previous one.
- deltaT2: Time (in days) since the record before the previous one.

The figure 3.1 despictes the evolutions of the INR and the predictions on the whole duration of the dataset.

The figure 3.2 despictes the correlation between the INR and the prescription. A large INR usually leads to a small prescription and a small INR to a large prescription. However, the correlation between the INR and the prescription is only 0.000193!. Things are not so simple...

3.3 Experiments

The experiments have been done with 10-fold cross validations. As well as SMOreg, 8 other algorithms available in Weka have been tested. When necessary, a short optimization of the parameters has been done.

Two values were used to evaluate performance.

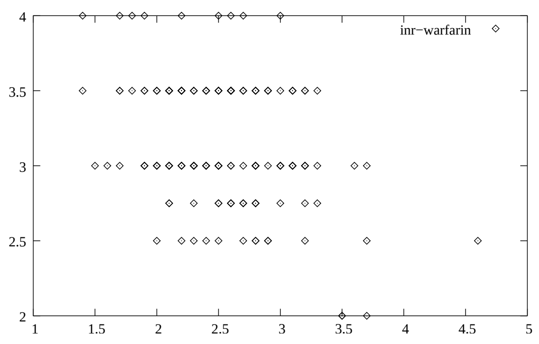

Figure 3.2: The correlation INR - warfarin prescriptions.

The *Correlation Coefficient* gives a measure of how much the deviations of two variables match. The closer to one the better. A value of 0 means that predictions and expected values don't match at all.

The *Relative Absolute Error* is a ratio of the *mean absolute error* of the learning algorithm over the *mean absolute error* obtained by predicting the mean of the training data. The lower the value the better the performance of the classifier compared to just predicting the mean. A score of 100% indicates the same performance as predicting the mean. In other words:

$$\text{Relative Absolute Error} = \frac{\sum_{i=1}^{n} |y_i - f_i|}{\sum_{i=1}^{n} |y_i - \overline{y_i}|}$$

where y_i if the expected value for the i^{th} instance, f_i the prediction and $\overline{y_i}$ the mean of the y_i.

3.3.1 Task1: Prediction of the Prescription

In this task, the algorithms are supposed to learn to imitate the doctor. Given the short term historic of the patient and his current state, they must predict what would be the prescription.

It is interesting to notice that the parameters ϵ may be chosen easily. The Warfarin's prescription are all in the set: $\{2.0, 2.5, 3.0, 3.5, 3.75, 4.0\}$. Therefore, an error of 0.25 doesn't really matter. Since the data are normalized, it gives $\epsilon = 0.1$.

algorithm	**Correlation Coefficient**	**Relative Absolute Error**
SMOreg (RBF, $\gamma = .1, \epsilon = .1, C = 3$)	**0.8025**	49.63 %
SMOreg (poly, $e = 2, \epsilon = .1, C = .1$)	0.7934	49.55 %
SMOreg (default options)	0.7851	**48.5754 %**
LinearRegression	0.7789	51.12 %
PaceRegression	0.7717	52.32 %
IBk (5 neighbours)	0.7365	53.88 %
NeuralNetwork	0.7106	60.12 %
LeastMedSq	0.6857	54.64 %
LWL	0.6462	70.26 %
IB1	0.6111	61.18 %
UnivariateLinearRegression	0.4875	75.54 %

The results are definitely very satisfactory. Not only SMOreg gives accurate predictions but it outperformed all the other algorithms. Actually, even with the default configuration, SMOreg is still the most accurate one.

3.3.2 Task2: Prediction of the INR

In this second task, the algorithms have to learnt to predict the INR for the current record, knowing the two previous record. The current Warfarin's prediction, though, isn't available. This appears to be a much harder problem.

algorithm	**Correlation Coefficient**	**Relative Absolute Error**
IBk (12 neighbours)	**0.402**	**90.02 %**
SMOreg (poly, $e = 2.2, \epsilon = 1^{-3}, C = 2$)	0.3687	94.83 %
SMOreg (default options)	0.3542	91.39 %
LinearRegression	0.3206	92.61 %
LeastMedSq	0.3128	93.66 %
PaceRegression	0.296	94.55 %
LWL	0.278	95.85 %
IB1	0.2486	129.20 %
NeuralNetwork	0.1954	114.53 %
UnivariateLinearRegression	0.0534	100.26 %

Once again, SMOreg performed quite well compared to the others. However, the old worthy *k-nearest neighbour* showed that it is not going to retire soon. It outperformed SMOreg, though it needed not less that the 12 nearest neighbours for that.

When not compared to the other algorithms, the results were disappointing. The *Relative Absolute Error* is not far below 100 %. Then, the algorithm performed not much better that using the means as a prediction.

Chapter 4

Appendix

4.1 pseudo-code for SMO regression

```
target = desired output vector
point = training point matrix
f-cache = cache vector for F_i values (contains d_i - w . z_i)

procedure takeStep(i1, i2)
  if ( i1 == i2) return 0
  alpha1, alpha1' = Lagrange multipliers for i1
  F1 = f-cache[i1]
  k11 = kernel(point[i1], point[i1])
  k12 = kernel(point[i1], point[i2])
  k22 = kernel(point[i2], point[i2])
  eta = -2*k12 + k11 + k22
  gamma = alpha1 - alpha1' + alpha2 - alpha2'

  % We assume that eta > 0. Otherwise one has to repeat the complete
  % reasonning similarly (i.e. compute objective functions at L and H
  % and decide which one is the largest.)

  case1 = case2 = case3 = case4 = finished = 0
  deltaphi = F1 - F2
  while !finished
    % This loop is passed at most three times
    % Case variables needed to avoid attempting small changes twice
    if (case1 == 0) &&
       (alpha1 > 0 || (alpha1' == 0 && deltaphi > 0)) &&
       (alpha2 > 0 || (alpha2' == 0 && deltaphi < 0))
         compute L, H (w.r.t. alpha1, alpha2)
         if (L < H)
            a2 = alpha2 - (deltaphi / eta)
            a2 = min(a2, H)
            a2 = max(L, a2)
            a1 = alpha1 - (a2 - alpha2)
            update alpha1, alpha2 if change is larger than some eps
         else
            finished = 1
```

```
          endif
          case1 = 1
     else if (case2 == 0) &&
        (alpha1 > 0 || (alpha1' == 0 && deltaphi > 2*epsilon)) &&
        (alpha2' > 0 || (alpha2 == 0 && deltaphi > 2*epsilon))
          compute L, H (w.r.t. alpha1, alpha2')
          if (L < H)
             a2 = alpha2' + ((deltaphi - 2*epsilon)/eta)
             a2 = min(a2, H)
             a2 = max(L, a2)
             a1 = alpha1 - (a2 - alpha2')
             update alpha1, alpha2' if change is larger than some eps
          else
             finished = 1
          endif
          case2 = 1
     else if (case3 == 0) &&
        (alpha1' > 0 || (alpha1 == 0 && deltaphi < -2*epsilon)) &&
        (alpha2 > 0 || (alpha2' == 0 && deltaphi < -2*epsilon))
          compute L, H (w.r.t. alpha1', alpha2)
          if (L < H)
             a2 = alpha2' - ((deltaphi + 2*epsilon)/eta)
             a2 = min(a2, H)
             a2 = max(L, a2)
             a1 = alpha1' + (a2 - alpha2)
             update alpha1', alpha2 if change is larger than some eps
          else
             finished = 1
          endif
          case3 = 1
     else if (case4 == 0) &&
        (alpha1' > 0 || (alpha1 == 0 && deltaphi < 0)) &&
        (alpha2' > 0 || (alpha2 == 0 && deltaphi > 0))
          compute L, H (w.r.t. alpha1', alpha2')
          if (L < H)
             a2 = alpha2' - deltaphi/eta
             a2 = min(a2, H)
             a2 = max(L, a2)
```

```
            a1 = alpha1' + (a2 - alpha2')
            update alpha1', alpha2' if change is larger than some eps
         else
            finished = 1
         endif
         case4 = 1
    else
       finished = 1
    endif
    update deltaphi
  endwhile
  if changes in alpha1('), alpha2(') are larger than some eps
      Update f-cache[i] for i in I_0 using new Lagrange multipliers
      Store the changes in alpha, alpha' array
      Update I_0, I_1, I_2, I_3
      Compute (i_low, b_low) and (i_up, b_up) by applying the conditions
      mentionned above, using only i1, i2 and indices in I_0
      return 1
  else
      return 0
  endif
endprocedure



procedure examineExample(i2)
  alpha2, alpha2' = Lagrange mutipliers for i2
  if (i2 in in I_0)
     F2 = f-chace[i2]
  else
     compute F2 = F_i2 and set f-cache[i2] = F2
     % Update (b_low, i_low) or (b_up, i_up) using (F2, i2)...
     if (i2 is in I_1)
        if (F2+epsilon < b_up)
           b_up = F2+epsilon, i_up = i2
        elseif (F2-epsilon > b_low)
           b_low = F2-epsilon, i_low = i2
        end if
```

```
   elseif ( (i2 is in I_2) && (F2+epsilon > b_low))
      b_low = F2+epsilon, i_low = i2
   elseif ( (i2 is in I_3) && (F2-epsilon < b_up))
      b_up = F2-epsilon, i_up = i2
   endif
endif
% check optimality using current b_low and b_up and, if
% violated, find an index i1 to joint optimization with i2...
optimilaty = 1
case 1: i2 is in I_0a
        if (b_low-(F2-epsilon) > 2*tol)
           optimality = 0; i1 = i_low;
           % For i2 in I_0a choose the better i1...
           if ((F2-epsilon)-b_up > b_low-(F2-epsilon))
              i1 = i_up;
           endif
        elseif ((F2-epsilon)-b_up > 2*tol)
           optimality = 0; i1 = i_up;
           % For i2 in I_0a choose the better i1...
           if (b_low-(F2-epsilon) > (F2-epsilon)-b_up)
              i1 = i_low;
           endif
        endif

case 2: i2 is in I_0b
        if (b_low-(F2+epsilon) > 2*tol)
           optimality = 0; i1 = i_low;
           % For i2 in I_0b choose the better i1...
           if ((F2+epsilon)-b_up > b_low-(F2+epsilon))
              i1 = i_up;
           endif
        elseif ((F2+epsilon)-b_up > 2*tol)
           optimality = 0; i1 = i_up;
           % For i2 in I_0b choose the better i1...
           if (b_low-(F2+epsilon) > (F2+epsilon)-b_up)
              i1 = i_low;
           endif
        endif
```

```
  case 3: i2 is in I_1
          if (b_low-(F2+epsilon) > 2*tol)
             optimality = 0; i1 = i_low;
             % For i2 in I_1choose the better i1...
             if ((F2+epsilon)-b_up > b_low-(F2+epsilon))
                i1 = i_up;
             endif
          elseif ((F2-psilon)-b_up > 2*tol)
             optimality = 0; i1 = i_up;
             % For i2 in I_1 choose the better i1...
             if (b_low-(F2-epsilon) > (F2-epsilon)-b_up)
                i1 = i_low;
             endif
          endif

  case 4: i2 is in I_2
          if ((F2+epsilon)-b_up > 2*tol)
             optimality = 0; i1 = i_up;
          endif

  case 5: i2 is in I_3
          if (b_low-(F2-epsilon) > 2*tol)
             optimality = 0; i1 = i_low;
          endif

  if (optimality == 1)
     return 0
  if takeStep(i1, i2))
     return 1
  else
     return 0
  endif
endprocedure
```

```
% main routine for modification 1 of Shevade et all
procedure main

  set alpha and alpha' to wero for every example
  set I_1 to contain all the examples
  Choose any example i from the training set
  set b_up = target[i]+epsilon
  set b_low = target[i]-epsilon
  i_up = i_low = i
  while (numChanged > 0 || examineAll)
    numChanged = 0
    if (examineAll)
      loop I over all the training examples
            numChanged += examineExample(I)
    else
      loop I over I_0
            numChanged += examineExample(I)
            % it is easy to check if optimality  on I_0 is attained...
            if (b_up > b_low - 2 * tol) at any I
                exit the loop after setting numChanged = 0
    endif
    if (examineAll == 1)
      examineAll = 0;
    elseif (numChanged ==0)
      examineAll = 1;
    endif
  endwhile
endprocedure
```

Bibliography

[1] *The Nature of Statistical Learning Theory.* Springer, 1995.

[2] *Data mining : Practical machine learning tools and techniques with java implementations.* Morgan Kaufmann Publishers, 2000.

[3] David W. Aha & Dennis Kibler & Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 1991.

[4] Bernhard Sch olkopf Alex J.Smola. A tutorial on support vector regression. Technical report, NeuroCOLT2, 1998.

[5] Martin B. Instance-based learning : Nearest neighbour with generalisation. Master's thesis, University of waikato, 1995.

[6] G. Bakiri. *Converting English Text To Speech : A Machine Learning Approach.* PhD thesis, Oregon State University, 1991.

[7] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[8] Wettschereck D. & Dietterich G. An experimental comparison of the nearest-neighbour and nearest-hyperrectangle algorithms. *Machine Learning*, 1995.

[9] D. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 1995.

[10] John C. Platt. Sequential minimal optimization : A fast algorithm for training support vector machines. Technical report, Microsoft Research, 1998.

[11] C. Bhattacharyya & K.R.K Murthy S.K. Shevade, S.S. Keerthi. Improvements to smo algorithm for svm regression. Technical report, Control Division, Dept of Mechanical and Production Engineering, National University of Singapore, 1999.

[12] Salzberg S.L. A nearest hyperrectangle learning method. *Machine Learning*, 1991.

[13] Alexander J. Smola. *Learning with kernels.* PhD thesis, University of Berlin, 1998.

[14] Aha D. W. Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies*, 1991.

[15] The weka team. Weka, machine learning project, 2003.

[16] Mohri T. Wettschereck D., Aha D. W. A review and comparative evaluation of feature weighting methods for lazy learning algorithms. *Artificial Intelligence Review*, 1997.