DATA MINING ALGORITHMS FOR RANKING PROBLEMS

DATA MINING ALGORITHMS

FOR

RANKING PROBLEMS

By

Tianshi Jiao, M.Sc.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (2006)	McMaster University
COMPUTING AND SOFTWARE	Hamilton, Ontario

TITLE: DATA MINING ALGORITHMS FOR RANKING PROBLEMS

AUTHOR: Tianshi Jiao, M.Sc. (McMaster University)

SUPERVISOR: Dr. Jiming Peng, Dr. Tamás Terlaky

NUMBER OF PAGERS: xiv, 95

Abstract

Classification is the process of finding (or training) a set of models (or functions) that describe and distinguish data classes or concepts. That is for the purpose of being able to use the models to predict the unknown class labels of instances [12].

We deal with the ranking problem in this thesis. The ranking problem is a special case of the classification problem, where the class labels are ranks or ratings, represented by integers from 1 to q. The ranking problem can also be cast as the process of training a rank-prediction model that assigns each instance a rank that is "as close as possible" to the instance's actual rank [8]. Popular applications of the ranking problem include ranking the importance of web pages, evaluating the financial credit of a person, and ranking the risks of investments.

Two popular families of methods to solve ranking problems are Multi-Criteria Decision Aid (MCDA) methods and Support Vector Machines (SVMs). The performance of successful MCDA methods, such as UTilités Additives DIScriminantes (UTADIS) and Generalized UTilités Additives DIScriminantes (GUTADIS), is achieved by exploiting the background knowledge that describes the correlations between the attributes and the ranks. Unfortunately, the background knowledge is case-dependent, hence it is likely to be unavailable, inexact or difficult to be modeled in practice. This restricts the application of MCDA methods. SVMs, instead, do not require any background knowledge. Their good performance is achieved by keeping balance between minimizing the empirical loss and maximizing the separation margin. Normally, a multi-class Support Vector Machine Classifier is constructed by combining several binary Support Vector Machine Classifiers. In the SVM-based approach the ranking information is not used.

This thesis attempts to construct an efficient algorithm for ranking problems. We compare the properties of existing algorithms for ranking problems and propose a hybrid algorithm that combines the multi-class SVM (M-SVM) and the UTADIS model. In the new algorithm, the binary SVM classifiers are combined into a multi-class classifier based on the fuzzy voting technique. The optimal fuzzy voting strategy is searched by solving a Linear Program (LP). The new algorithm is called Fuzzy Voting based Support Vector Ranking (FVSVR) method. We also extend the idea of Fuzzy Voting from ranking problems to generic multi-class classification problems, which leads to a Fuzzy Voting based Support Vector Machine (FVSVM) method. The benefits of FVSVR and FVSVM are demonstrated by experimental results based on several databases of practical classification problems.

Acknowledgements

I would like to gratefully appreciate my supervisors Dr. Jiming Peng and Dr. Tamás Terlaky for their valuable comments and enthusiastic support. I also thank them for his very careful reading and insightful suggestions during the writing of this thesis, and their thoughtful guidance during my graduate study.

I am indebted to Dr. Tamás Terlaky, and Dr. Peng for the great optimization lessons they gave to me. I appreciate Dr. Terlaky, and Dr. Peng, for the inspirational optimization seminars they organized and for the great computation power of the laboratory they provided to us.

I sincerely thank Dr. Jiming Peng, Dr. Tamás Terlaky, Dr. Sanzheng Qiao and Dr. Ridha Khedri for their agreement to be a committee member and for their careful reading of, and helpful suggestions on, my thesis.

I thank all members of the Advanced Optimization Lab (AdvOL) for their friendly help and for the great working environment they made.

Special thanks also goes to my friends Wei Xu from McMaster University and Zhuo Zheng from Queens University for their helpful suggestions on my thesis.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

In this chapter we first give a brief introduction to the ranking problem and review two related concepts: multi-class classification and regression. Then, we describe the motivation of our research and outline the organization of the thesis.

1.1 Classification and Regression

As we have mentioned in the abstract, classification is the task of constructing a set of models to separate different classes. The applications of classification include pattern recognition, image segmentation, and natural language processing.

In classification, an instance (a sample) is usually represented by a used to predict an instance's target value, which is usually continuous [12]. In classification, the set of class labels $\mathcal{Y}$ is a set of discrete class labels such as $\{-1, 1\}$ or $\{A, B, C\}$. In regression, instead, $\mathcal{Y}$ is a continuous domain, such as $[-1, 1]$ or $(-\infty, +\infty)$.

1.2 The Prediction Loss

The training processes of both the classification problem and the regression problem can be cast as optimization problems. Specifically, the task of the training process is to find a set of optimal parameters for the classification/regression model (function) so that the expected risk of errors [4] is minimized.

We usually use a loss function [4], which determines the amount of loss when prediction errors take place on an instance, to evaluate the expected risk of errors of a prediction model. The most popular loss function for classification problems is the 0-1 loss function. Suppose that $l(\hat{y}, y)$ denotes the loss function of assigning a label $\hat{y}$ to a sample with an actual label y. The 0-1 loss function is:

$$l(\hat{y}, y) = \begin{cases} 1, & \text{if } \hat{y} \neq y, \\ 0, & \text{if } \hat{y} = y. \end{cases} \tag{1.1}$$

This means that each misclassification causes identical loss of value 1.

There are two popular loss functions for regression problems. One is

Chapter 1

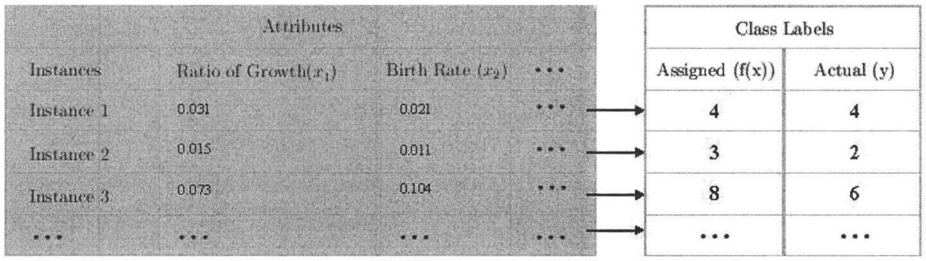
INTRODUCTION

In this chapter we first give a brief introduction to the ranking problem and review two related concepts: multi-class classification and regression. Then, we describe the motivation of our research and outline the organization of the thesis.

1.1 Classification and Regression

As we have mentioned in the abstract, classification is the task of constructing a set of models to separate different classes. The applications of classification include pattern recognition, image segmentation, and natural language processing.

In classification, an instance (a sample) is usually represented by a fixed number of attributes $x = [x_1, x_2, ..., x_n]$, together with a class label y. A instance is called labeled if its class label is known, otherwise unlabeled. Figure1.1 lists a few samples in the country risk classification problem.



Attributes					Class Labels	
Instances	Ratio of Growth(x_1)	Birth Rate (x_2)	•••		Assigned (f(x))	Actual (y)
Instance 1	0.031	0.021	•••	$\rightarrow$	4	4
Instance 2	0.015	0.011	•••	$\rightarrow$	3	2
Instance 3	0.073	0.104	•••	$\rightarrow$	8	6
•••	•••	•••	•••	$\rightarrow$	•••	•••

Figure 1.1: Samples in the country risk classification problem.

The process of classification has two steps: training and test/use. In the training step, a set of labeled instances (the training set) is provided. Based on the training set, a mapping from the attribute space $\mathcal{X}$ to the class label space $\mathcal{Y}$ is established by training. The training process can be formulated as an optimization problem that we will discuss in the next section. The resultant mapping f is called a classifier. For example, for $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$ we might have a binary classifier $f : \mathcal{X} \rightarrow \mathcal{Y}$. We call it a binary classifier since all the instances are in 2 classes, either in class "0" or in class "1", i.e., the the cardinality of $\mathcal{Y}$ is 2. The problem to construct such a binary classifier is called as the binary classification problem. Correspondingly, if the cardinality of $\mathcal{Y}$ is q, the problem is a q-class multi-class classification problem.

After the training step, the obtained classifier f may be used to assign a class label $\hat{y}$ to any instance x by $\hat{y} = f(x)$. The performance of this classifier is tested by another set of labeled instances (test set), and the prediction accuracy is assessed based on the comparison between the actual class labels and the class labels assigned by f. If the obtained accuracy is acceptable, this classifier can be used in practice. The whole classification process is demonstrated by Figure 1.2.

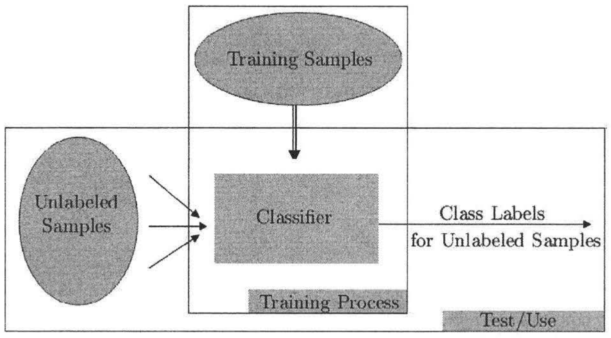


Figure 1.2: The classification system.

In the field of data mining, the regression problem is to train a regression function that assigns a value to an instance that is as close as possible to its actual value. The applications of regression include traffic flux estimation and air temperature prediction.

Similar to the classification problem, the regression problem is to find a function that describes or distinguishes certain concept. The difference between classification and regression is that, classification is to predict an instance's class label, which is usually discrete or nominal, while, regression is used to predict an instance's target value, which is usually continuous [12]. In classification, the set of class labels $\mathcal{Y}$ is a set of discrete class labels such as $\{-1, 1\}$ or $\{A, B, C\}$. In regression, instead, $\mathcal{Y}$ is a continuous domain, such as $[-1, 1]$ or $(-\infty, +\infty)$.

1.2 The Prediction Loss

The training processes of both the classification problem and the regression problem can be cast as optimization problems. Specifically, the task of the training process is to find a set of optimal parameters for the classification/regression model (function) so that the expected risk of errors [4] is minimized.

We usually use a loss function [4], which determines the amount of loss when prediction errors take place on an instance, to evaluate the expected risk of errors of a prediction model. The most popular loss function for classification problems is the 0-1 loss function. Suppose that $l(\hat{y}, y)$ denotes the loss function of assigning a label $\hat{y}$ to a sample with an actual label y. The 0-1 loss function is:

$$l(\hat{y}, y) = \begin{cases} 1, \text{ if } \hat{y} \neq y, \\ 0, \text{ if } \hat{y} = y. \end{cases} \tag{1.1}$$

This means that each misclassification causes identical loss of value 1.

There are two popular loss functions for regression problems. One is the least square loss function [10] defined as:

$$l_{LSL}(\hat{y}, y) = |\hat{y} - y|^2; \tag{1.2}$$

and the other one is the ε-intensive loss function [25] defined as:

$$l_\varepsilon(\hat{y}, y) = \begin{cases} 0, & \text{if } |\hat{y} - y| \leq \varepsilon, \\ |\hat{y} - y| - \varepsilon, & \text{if } |\hat{y} - y| > \varepsilon, \end{cases} \tag{1.3}$$

where ε is a constant and $|\hat{y} - y|$ denotes the distance between the assigned class label and the actual class label. Figure 1.3 depicts the ε-intensive loss function graphically. Both of these loss functions are distance-based ones,

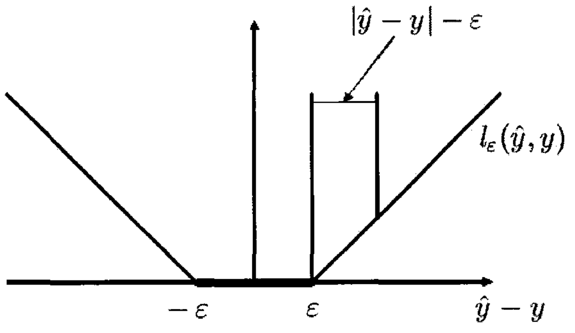


Figure 1.3: The ε-intensive loss.

since they are monotone functions of $|\hat{y} - y|$. Compared to the 0-1 loss function for classification, the distance-based loss functions can guide the optimization process of regression training so that the obtained regression function may assign a label value to an instance that is "as close as possible" to its actual label value.

1.3 The Ranking Problem

There are many applications where it is desirable to rank rather than to classify instances [6], for instance: ranking the importance of web pages, evaluating the financial credit rating of a person, and ranking the risk of investments.

The ranking problem is the task of learning a rank-prediction model that assigns an instance a discrete rank "as close as possible" to its actual rank [8]. In this thesis we define the ranking problem as a multi-class classification problem with ordinal class labels and a distance-based loss function. We can see that ranking problems are predication problems that share properties from both classification problems and regression problems. In ranking problems, the values to be predicted are termed as ranks or levels. They are discrete values as in classification problems. However, the loss functions of ranking problems are distance-based as in regression problems.

The special properties of ranking problems indicate that the algorithms for classification and regression are not quite suitable for ranking problems. The evidence for this argument is demonstrated in the remainder of our thesis.

1.4 Motivation and Organization

Two popular algorithms for ranking problems are the UTilités Additives DIScriminantes (UTADIS) [11] and the multi-class Support Vector Machine (SVM) methods [7]. We offer a brief discussion of the advantages and disadvantages of these two methods in this section to propose the motivation of our research.

The UTADIS model, which is a successful MultiCriteria Decision Aid (MCDA) method [29], incorporates the background knowledge in ranking problems by optimizing a set of piecewise-linear monotone criterion functions that represent the monotone correlations between the attributes and the ranks. Since generally the monotonicity assumption of the UTADIS model does not hold [26], its accuracy is not encouraging. The accuracy of GUTADIS model [26] is better than UTADIS, but its training process is time-consuming.

SVMs have good generalization performance for pairwise classification problems [13]. Normally, a multi-class Support Vector Machine classifier is constructed by combining several binary Support Vector Machine classifiers[13]. The problem with the popular "Max-Wins" voting combination method is that it does not take the ordinal relation among classes into consideration and it does not use the distance-based loss function. Thus this method is not good for ranking problems either.

The main purpose of this thesis is to construct an efficient algorithm for ranking problems assuming that the ranking information of classes is available. In particular, we discuss some issues on how to combine binary SVM classifiers. We propose a hybrid algorithm that takes the advantage of both of UTADIS and SVM, and combines binary SVM classifiers by "fuzzy voting". The same idea is extended to the generic multi-class classification.

The rest of this thesis is organized as follows. In Chapter 2, a review of the UTADIS and the GUTADIS models for ranking problems is presented. In Chapter 3, binary SVM, multi-class SVM and SVR are reviewed and discussed. We propose our hybrid algorithm for ranking problems and for generic multi-class classification problems in Chapter 4. Chapter 5 describes the process of our empirical experiments and reports the results. We conclude our work and suggest some further research directions in Chapter 6.

Chapter 2

UTADIS AND GUTADIS

In this chapter, we review the UTADIS and GUTADIS methods for ranking problems.

2.1 The UTADIS Model

2.1.1 Prior Knowledge in Ranking Problems

Prior knowledge (background knowledge in some references) refers to general information about the concepts that we are concerned about [16]. How to incorporate the backgróund knowledge into the construction of a classification model is an active research topic. In ranking problems, the most frequently used background knowledge is the correlation between the attributes and the class labels. For example, to predict the age of an abalone from its physical measurements [21], we may use the prior knowledge that a greater size indicates an older age. Imposing the prior knowledge on the classifier can make the classification result consistent with experts' knowledge and easy to understand. When the training set is relatively small, the background knowledge can help to control the complexity of the classification model so that the overfitting problem [22] is avoided.

In [11], the authors proposed UTADIS, a classification model that incorporates the background knowledge in function space. In the UTADIS model, the correlations between the attributes and the class labels are assumed to be monotone. The assumption is consistent with the background knowledge of the undertaking applications.

2.1.2 The Model of UTADIS

The idea of UTADIS is to train a utility function $U(x)$ to determine the class label of sample x. The concept of utility function comes from decision science [14], where the ranking problem is considered to make the best choice (the best class) from a list of choices (classes). An assumption is that any decision-maker unconsciously uses utility function:

$$U(x) = U(x_1, x_2, ..., x_n),$$

which aggregates his preferences in different aspects (attributes) of a choice, to rank all the optional choices [26].

Once we get the utility function, the class label of a sample x is decided by the following rules:

$$\begin{cases} \text{if } U(x) \geq \mu_1, \text{ then } x \in C_1, \\ \text{if } \mu_k \leq U(x) < \mu_{k-1}, \forall k, \ 2 \leq k \leq q-1, \text{ then } x \in C_k, \\ \text{if } U(x) < \mu_{q-1}, \text{ then } x \in C_q, \end{cases} \tag{2.1}$$

where the utility function $U(x)$ is usually normalized, i.e., $0 \leq U(x) \leq 1$, and $\mu_1, ..., \mu_{q-1}$ are the decision boundaries for the q classes.

Then, the main task of UTADIS training is to estimate such a utility function $U(x)$ and a set of decision boundaries $\{\mu_1, ..., \mu_{q-1}\}$. Thus, we have to define a classification function space as our search space. In the UTADIS model, $U(x)$ is searched in the following function space:

$$U(x^c) = \sum_i U_i(x^c), \tag{2.2a}$$

$$U_i(x^c) = U_i(x_i^c) = U_i^j \frac{x_i^{j+1} - x_i^c}{x_i^{j+1} - x_i^j} + U_i^{j+1} \frac{x_i^c - x_i^j}{x_i^{j+1} - x_i^j}, \tag{2.2b}$$

where $U(x)$ is the additive combination of a set of criterion functions: $\{U_i(x), \ i = 1, ..., q\}$, and $U_i(x)$ is a piecewise-linear monotone function that characterizes the correlation between the ith attribute and the class label. According to (2.2b), when x_i^j is fixed, the piecewise-linear function $U_i(x)$ is decided by the sequence of U_i^j, $j = 1, ..., r_i$, which are the variables to be optimized. It is obvious that searching for an optimal piecewise-linear function $U_i(x)$ is equivalent to searching for a set of optimal variables $\{U_i^j,\ j = 1, ..., r_i\}$. Figure 2.1 demonstrates a one-dimensional utility function decided by a monotone sequence of 6 points.

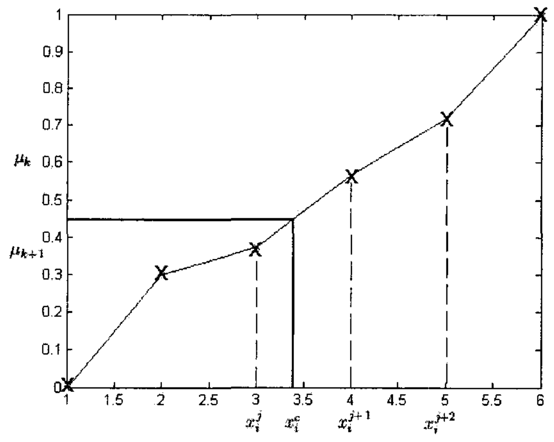


Figure 2.1: A piecewise monotone criterion function

Note that by using the additive form of the utility function, we impose another implicit assumption, i.e., each criterion (attribute) affects the utility function value independently. This assumption makes the model simple and easy to solve. It is typically not true in practice, though.

2.1.3 The Loss Function of UTADIS

The loss function of the UTADIS model is distance-based. To describe this, we need to introduce the "right range" for an instance. For an instance x in class k, its "right range" is the range: $[\mu_k, \mu_k - 1]$, where μ_k is the boundary values introduced in (2.1). According to (2.1), if the utility function value $U(x)$ is in the "right" range of x, the instance is correctly classified. Correspondingly, the classification loss of x is zero. If the value of $U(x)$ is out of the "right range" of x, the misclassification error is linearly punished. Figure 2.2 demonstrates the "right range" for instances in class k, denoted by C_k, and the misclassification errors when $U(x)$ is smaller than μ_i or greater than μ_{i-1}, denoted by $\sigma^+(c)$ and $\sigma^-(c)$ respectively. The loss function of UTADIS is quite similar with the ε-intensive loss (1.3).

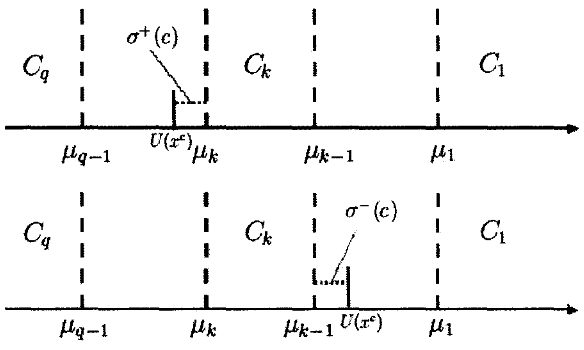


Figure 2.2: The "right range" for class k and misclassification errors.

Then the UTADIS training is to find the optimal estimation of the utility functions $U(x)$ and the set of the optimal boundary values $\{\mu_1, ..., \mu_{q-1}\}$, so that the total loss of the misclassified training instances is minimized. This gives the following LP problem.

2.1.4 The Linear Program for UTADIS Training

The UTADIS training is a Linear Program:

$$\min_{\sigma,\mu,U} \sum_{c \in C_k : k<q} \sigma^+(c) + \sum_{c \in C_k : k>1} \sigma^-(c) \tag{2.3a}$$

$$\text{s.t.} \sum_{i=1}^{n} U_i(x_i^c) - \mu_k + \sigma^+(c) \geq 0, \ \forall 1 \leq k \leq q-1, \ \forall c \in C_k, \tag{2.3b}$$

$$\sum_{i=1}^{n} U_i(x_i^c) - \mu_{k-1} - \sigma^-(c) \leq -\delta, \ \forall 2 \leq k \leq q, \ \forall c \in C_k, \tag{2.3c}$$

$$\mu_{k-1} - \mu_k \geq s, \ k = 2, \cdots, q-1, \tag{2.3d}$$

$$U_i^{j+1} - U_i^j \geq 0, \ \forall i = 1, \cdots, n, \ \forall j = 1, \cdots, r_i - 1, \tag{2.3e}$$

$$U_i^1 = 0, \ \forall i = 1, \cdots, n, \tag{2.3f}$$

$$\sum_{i=1}^{n} U_i^{r_i} = 1, \tag{2.3g}$$

$$\sigma^+(c) \geq 0, \ \sigma^-(c) \geq 0, \ \forall c, \tag{2.3h}$$

where n is the number of the attributes, q is the number of classes, and $U_i(x^c)$ is the criterion function defined by (2.2 b), $\forall c \in C_k$ is used to denote "for all the instances x^c whose class label are k ". Misclassification error for training instance x^c is captured by $\sigma^+(c)$ or $\sigma^-(c)$ in constraints (2.3 b,c), which is linearly punished in the objective function (2.3 a). The boundary values are defined in (2.3 d). Constraint (2.3 e) guarantees that the piecewise-linear criterion functions are monotone. Normalization constraints (2.3 f,g) define the contribution of each attribute to the resulting utility function value.

After the training step, the obtained $U(x)$ can predict the class label of unknown samples. For an unknown instance x, we compare the utility function value $U(x)$ with the boundary values, $\mu_1, ..., \mu_{q-1}$, and assign x a class label according to (2.1).

The prediction accuracy of the UTADIS model is usually not good, since both the naive monotone assumption and the independent assumption are not satisfied in many applications. These unrealistic assumptions should be removed or generalized.

2.2 The GUTADIS Model

To deal with the non-monotone criteria, Wang [26] introduced the GUTADIS model that extends the monotone assumption to unimodal.

In GUTADIS model, we still use the additive combination of criterion functions to estimate the utility function, which means we still impose the independent assumption. The loss function of GUTADIS is the same as the UTADIS model. The only difference is that we search the piecewise-linear unimodal function space for optimal criterion functions.

The optimization problem of GUTADIS training is:

$$\min_{\sigma,\mu,U} \sum_{c\in C_k:k<q} \sigma^+(c) + \sum_{c\in C_k:k>1} \sigma^-(c) \tag{2.4a}$$

$$\text{s.t.} \sum_{i=1}^{n} U_i(c) - \mu_k + \sigma^+(c) \geq 0,\ \forall 1 \leq k \leq q-1,\ \forall c \in C_k, \tag{2.4b}$$

$$\sum_{i=1}^{n} U_i(c) - \mu_{k-1} - \sigma^-(c) \leq -\delta,\ \forall 2 \leq k \leq q,\ \forall c \in C_k, \tag{2.4c}$$

$$\mu_{k-1} - \mu_k \geq s,\ \forall k = 2,\cdots,q-1, \tag{2.4d}$$

$$U_i^j - U_i^{j-1} \geq 0,\ \forall i \in I_m,\ \forall j = 2,\cdots,r_i, \tag{2.4e}$$

$$Y_{ij}(U_i^j - U_i^{j-1}) \geq 0,\ \forall i \in I_u,\ \forall j = 2,\cdots,r_i, \tag{2.4f}$$

$$Y_{ij} \geq Y_{i,j+1},\ \forall i \in I_u,\ \forall j = 2,\cdots,r_i - 1, \tag{2.4g}$$

$$0 \leq U_i^j \leq 1,\ \forall i = 1,\cdots,n,\ \forall j = 1,\cdots,r_i, \tag{2.4h}$$

$$Y_{ij} \in \{-1,1\},\ \forall i \in I_u,\ \forall j = 2,\cdots,r_i, \tag{2.4i}$$

$$\sigma^+(c) \geq 0,\ \sigma^-(c) \geq 0,\ \forall c, \tag{2.4j}$$

which is an integer nonlinear programming problem. Integer variable Y_{ij} is employed to control the increasing and decreasing of the criterion function. Constraints (2.4 f,g and i) exactly ensure that the sequence: $U_i^j,\ j = 1,...,r_i$ is a unimodal sequence.

The performance of GUTADIS model is much better than the original UTADIS model, because the unimodal assumption is much less restrictive than the monotonicity assumption. Nevertheless, there is one major concern for GUTADIS, i.e, the integer nonlinear programming problem (2.4) is typically intractable which excludes the algorithm from applications with a large data set.

Chapter 3

SUPPORT VECTOR MACHINES

In this chapter, we first review Support Vector Machine (SVM) for binary classification, also called Support Vector Classification (SVC). Then we discuss its extension to regression and multi-class classification

3.1 Binary SVM

In this section, we first introduce the hard margin linear SVM that forms the basis of our later discussion. Then we discuss the optimization model for SVM. At the end, we introduce soft margin SVM and nonlinear (kernel based) SVM as generalizations of the hard margin linear SVM.

3.1.1 Linear Hard Margin SVM

Binary classification is to train a model to separate two classes. We first consider the case where the classifier is a linear function, i.e., the separation hyperplane of the classifier can be written as:

$$w^T x + b = 0,$$

where w and b are the model parameters that determine the direction and position of the separation hyperplane. For any hyperplane $w^T x + b = 0$ that correctly splits the data points into two classes, we impose the following additional constraints on w and b

$$w^T x_1 + b = -1, \tag{3.1a}$$

$$w^T x_2 + b = +1, \tag{3.1b}$$

where x_1 and x_2 are the data points closest to the separate hyperplane in class 1 and class 2 respectively. The constraints actually imply:

$$w^T(x_2 - x_1) = 2,$$

and the separation hyperplane $w^T x + b = 0$ bisects the space (the margin) between the parallel hyperplanes defined by (3.1 a,b), as Figure 3.1 shows. Notice that if w is fixed and b is a variable, then $w^T x + b = 0$ defines a family of paralleled hyperplanes, in which only one hyperplane bisects the margin. In other words, by imposing the constraints, we only need to search an optimal w. When w is decided by the training process of the linear SVM, the value of b can be determined accordingly.

For any hyperplane $w^T x + b = 0$ correctly separating the data points of class 1 and class 2, we can define its margin as the distance between the parallel hyperplanes: $w^T x + b = 1$ and $w^T x + b = -1$, denoted by m in Figure 3.1. It follows that

$$\text{margin} = |\tfrac{w^T}{||w||}(x_1 - x_2)|,$$

(the projective length of $x_2 - x_1$ on direction w). From (3.1 a,b), we have that

$$\text{margin} = |\tfrac{w^T}{||w||}(x_1 - x_2)| = \tfrac{2}{||w||}.$$

It indicates that any pair of (w, b) satisfying constraints (3.1) decides an separation hyperplane and a margin. The size of the margin is $\frac{2}{||w||}$. It has been proved that essentially by requiring a larger margin (a small $||w||$), we can obtain a classification model with greater capacity of generalization [4]. The training of linear hard margin SVM is to search a function with a maximum margin in the linear function space under the constraints that all training samples are out of the margin. This is the so-called Max-Margin principle. Suppose we are given a set of n training samples $\{(x_i, y_i),\ x_i \in \mathbb{R}^n,\ y_i \in \{-1, 1\},\ i = 1, ..., n\}$, the following optimization problem describes the training process of linear hard

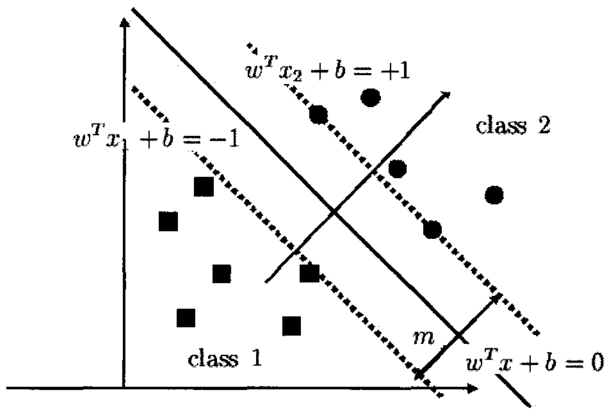


Figure 3.1: The margin between two classes.

margin SVM:

$$\min_{w,b} \frac{1}{2}||w||^2 \tag{3.2a}$$

$$\text{s.t. } y_i(w^T x_i + b) \geq 1,\ i = 1, ..., n. \tag{3.2b}$$

Constraints (3.2 b) guarantee that the training instance in both class 1 and class 2 are out of the margin. This is a convex linear constrained quadratic programming problem. In practice, we solve the dual problem that is given as follows:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{3.3a}$$

$$\text{s.t. } \sum_{i=1}^{n} \alpha_i y_i = 0, \tag{3.3b}$$

$$\alpha_i \geq 0,\ i = 1, ..., n, \tag{3.3c}$$

where α_i, $i = 1, ..., n$, are the Lagrange Multipliers of constraints (3.2 b). After solving the dual problem, w can be recovered by

$$w = \sum \alpha_i y_i x_i.$$

As a result of the training process, we get a linear decision function

$$f(z) = w^T z + b = \sum \alpha_i y_i x_i^T z + b.$$

For a new sample z, if $f(z) \leq 0$, we classify z into class 1, otherwise we classify it into class 2.

Note that in the dual problem of the training process and in the test or use process, the samples are referenced only as inner product: $x_i^T x_j$ or $x_i^T z$. This is important for the kernel trick, which will be discussed in Section 3.1.3.

3.1.2 Soft Margin SVM

In practice, a few outliers, denoted by x_i and x_j in Figure 3.2, may cause constraint (3.2b) infeasible. This implies that the two classes cannot be separated by a hyperplane. To deal with the infeasible cases, Vapnik [7] proposed the relaxed separation constraints as below:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \; i = 1, ..., n.$$

The violatings of constraint (3.2b) are captured in slack variables $\xi_i \geq 0$; $i = 1, ..., n$, which are penalized in the' objective function via a regularization con-

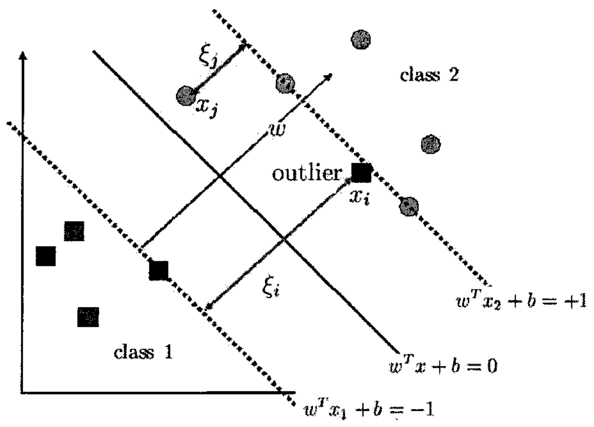


Figure 3.2: A linearly inseparable case.

stant C, chosen as a priori. As a consequence, we get the following quadratic programming problem:

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \xi_i \tag{3.4a}$$

$$\text{s.t. } y_i(w^T x_i + b) \geq 1 - \xi_i,\ i = 1, ..., n, \tag{3.4b}$$

$$\xi_i \geq 0,\ i = 1, ..., n. \tag{3.4c}$$

The constant C controls the tradeoff between the training errors (denoted by $\sum_{i=1}^{n} \xi_i$, the summation of the violatings) and the complexity of the separation function (denoted by $\frac{1}{2}||w||^2$). It has been proved that the expected risk, which indicates the probability of misclassification for unseen samples, is small if we keep both the training errors and the function complexity small [4].

The dual problem of (3.4) is:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{3.5a}$$

$$\text{s.t. } C \geq \alpha_i \geq 0, \; i = 1, ..., n, \tag{3.5b}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0. \tag{3.5c}$$

Again,the samples are referenced only as inner product here.

3.1.3 Nonlinear SVM

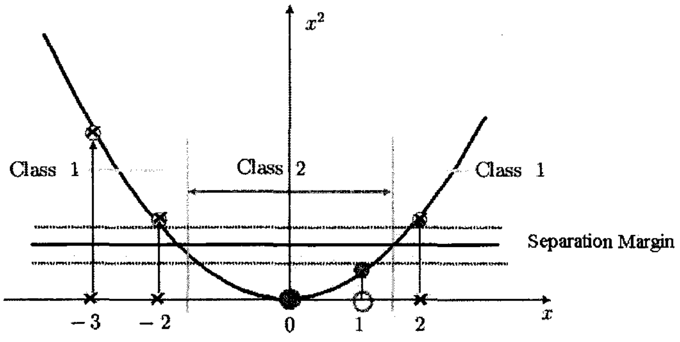


Figure 3.3: A mapping $\phi(x) = \{x, x^2\}$.

Up to now, the separation model we discussed is the linear function, whose usability is quite limited. To allow for more general separation boundary, first, the input vectors: $(x_1, ..., x_n)^T$ can be nonlinearly transformed into a high-dimension feature space by a mapping $\phi : \mathbb{R}^n \to \mathbb{R}^k, k > n$, then a linear separation can be searched in the feature space. Figure 3.3 demonstrates an separation using a mapping from one-dimension space to two-dimension space, where the instances in class 1 (-3,-2 and 2) and the instances in class 2 (0 and 1) are inseparable in the one-dimensional space but are separable in the two-dimensional space. This shows that mapping a data set from so-called input space to the feature space allows us to find a linear separation relatively easily.

As mentioned before in both the training and the test processes of the linear SVM, the samples are referenced only as inner product: $x^T z$. After the mapping, the inner product of samples in feature space becomes $\phi(x)^T \phi(z)$. So the dual problem of SVM training in the feature space is:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \Phi(x_i)^T \Phi(x_j) \tag{3.6a}$$

$$\text{s.t. } C \geq \alpha_i \geq 0,\ i = 1, ..., n, \tag{3.6b}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0. \tag{3.6c}$$

Correspondingly the decision function of the classifier is:

$$f(z) = w^T z + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \Phi(x_{t_j})^T \Phi(z) + b.$$

To make the calculation more efficient, we can rewrite $\phi(x)^T \phi(z)$ into the form of a kernel function, i.e., $K(x, z) = \phi(x)^T \phi(z)$. The dual problem for SVM training is rewritten as:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{3.7a}$$

$$\text{s.t. } C \geq \alpha_i \geq 0, \ i = 1, ..., n, \tag{3.7b}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0. \tag{3.7c}$$

The corresponding decision function can be constructed by:

$$f(z) = w^T z + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} K(x_{t_j}, z) + b.$$

Note that the map $\phi(x)$ does not appear in the final optimization formula. Actually, by selecting a kernel function, we implicitly select the mapping function $\phi(x)$. Every mapping function $\phi(x)$ has a corresponding kernel function: $K_\phi(x, z) = \phi(x)^T \phi(z)$, but not every bivariate function $K(x, z)$ has a corresponding mapping function $\phi(x)$. For example, function $K(x, z) = x^2 + 0 * z$ does not has such a $\phi(x)$. It has been proved that any bivariate function $K(x, z)$ satisfying Mercer's Condition [4] has a corresponding mapping function. Such a bivariate function $K(x, z)$ can thus be considered as a kernel function.

In conclusion, the good performance of SVMs lies in two important features: One is the balance between the training error and the complexity of the classification model, controlled by the parameter C, which minimizes the expected risk. The other is the kernel technique that makes the SVMs flexible and efficient.

3.2 Support Vector Regression (SVR)

The Max-Margin principle and kernel technique in SVM can be extended to regression. In ε-SVR [25], we are looking for a margin that contains most of the data points, instead of a margin to separate two classes in SVM. Figure 3.4 shows a linear ε-SVR, where the dark area is the the ε-tube (the "margin") of the linear regression function.

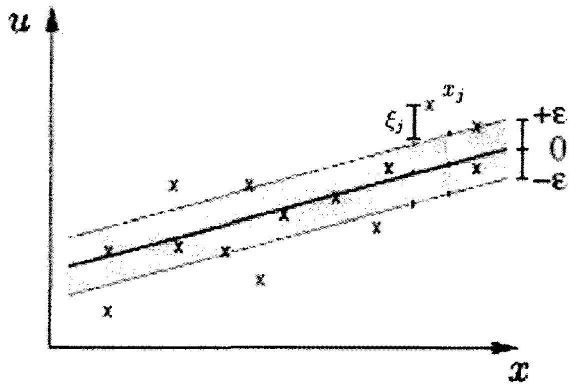


Figure 3.4: Support vector regression (linear case).

Similar to the discussion of SVM above, we consider the linear regression case of SVR first, then we extend the result to the nonlinear case.

Compared to the least square regression [10], which uses the least square loss function (1.2), ε-SVR uses the ε-insensitive loss function (1.3). They are compared in Figure 3.5, where u is the target value of an instance; $\hat{u}$ is the regression function value of the instance; $l(u, \hat{u})$ is the loss function. Intuitively,

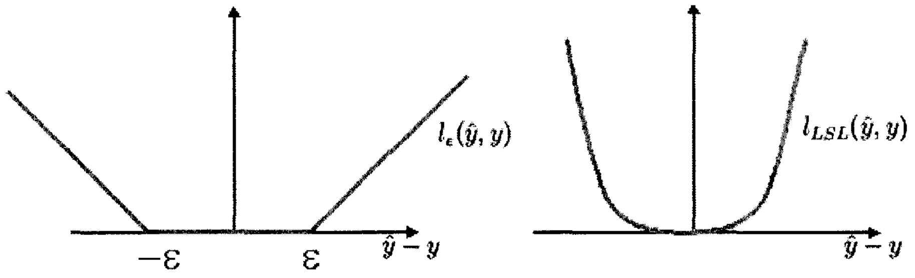


Figure 3.5: The comparison of the ε-insensitive loss and the least square loss.

by using ε-insensitive loss, errors less than ε are ignored; errors greater than ε are linearly penalized.

The ε-insensitive loss function is designed so that the ideas of margin and kernel can be used conveniently. Given a data set $x_1, ..., x_n$ with target values $u_1, ..., u_n$, and a priori chosen ε, the ε-SVR training is a quadratic programming problem [25]:

$$\min_{w,b,\xi,\xi^*} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \tag{3.8a}$$

$$\text{s.t. } u_i - w^T x_i - b \leq \varepsilon + \xi_i, \ i = 1, ..., n, \tag{3.8b}$$

$$-u_i + w^T x_i + b \leq \varepsilon + \xi_i^*, \ i = 1, ..., n, \tag{3.8c}$$

$$\xi_i \geq 0; \ \xi_i^* \geq 0 \ i = 1, ..., n. \tag{3.8d}$$

The errors greater than ε are captured by ξ_i and ξ_i^* that are punished in the objective function. Similar to SVM, C is a parameter to control the amount of influence of the training errors.

By solving the kerneled dual problem:

$$\max_{\alpha,\alpha^*} - \varepsilon \sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)u_i - \frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)K(x_i, x_j) \tag{3.9a}$$

$$\text{s.t. } \sum_{i=1}^{n}(\alpha_i^* - \alpha_i) = 0, \tag{3.9b}$$

$$0 \le \alpha_i \le \frac{C}{n},\ 0 \le \alpha_i^* \le \frac{C}{n},\ i = 1, ..., n, \tag{3.9c}$$

we obtain the values of α_i and α_i^*, which are both zero if x_i does not contribute to the error function. The obtained regression function is

$$f(z) = \sum_{j=1}^{s}(\alpha_{t_j} - \alpha_{t_j}^*)K(x_{t_j}, z) + b.$$

Jinbo Bi and Kristin Bennett [2] proposed a geometric approach demonstrating the relation between ε-SVR and binary SVC. They showed that ε-SVR can be regarded as a binary SVC in dual space.

ε-SVR can be used to solve ranking problems. The algorithm is described as follows:

- Setting the Parameters: Select the kernel function K and decide the values for C and ε.

- Training Process:

  Input: $(x_i, u_i),\ i = 1, ..., n$, where

  x_i is the attributes of the ith instances in the training set;

  u_i is the ranks of the ith instances in the training set;

  Step 1:

  Solve (3.9) to obtain the parameters $(\alpha_i, \alpha_i^*,\ i = 1, ..., n)$ of regression function.

  Step 2:

  $$f(z) \equiv \sum_{j=1}^{s} (\alpha_{t_j} - \alpha_{t_j}^*) K(x_{t_j}, z) + b$$

- Test or Use: For any unlabeled instance x^c, rank(x^c) =round($f(x^c)$), where round(x) rounds x to the nearest integer.

3.3 Multi-Class SVM (M-SVM)

The SVM [7] was originally proposed for binary classification, as discussed earlier. To extend SVM efficiently for the multi-class cases is an on-going research issue. The M-SVM classifiers are computationally more expensive than the binary SVM classifiers. The typical way to construct a multi-class SVM classifier is combining several binary SVM classifiers. There are three different strategies by which the binary classifiers are combined: "one-against-one" [18], "one-against-all" [3] and Directed Acyclic Graph Support Vector Machines (DAGSVM) [23].

For a q-class classification problem, the "one-against-all" strategy of M-SVM constructs q binary classifiers. The ith classifier is trained by taking the instances in the ith class as positive instances, and taking all the other instances as negative ones, i.e., the ith SVM training solves the following problem:

$$\min_{w^i,b^i,\xi^i} \frac{1}{2}||w^i||^2 + C\sum_{j=1}^{n} \xi_j^i \tag{3.10a}$$

$$\text{s.t. } (w^i)^T\phi(x_j) + b^i \geq 1 - \xi_j, \text{ if } y_j = i, \tag{3.10b}$$

$$(w^i)^T\phi(x_j) + b^i \leq -1 + \xi_j, \text{ if } y_j \neq i, \tag{3.10c}$$

$$\xi_j^i \geq 0. \tag{3.10d}$$

The decision functions of binary classifiers is obtained by

$$f_i(x) = (w^i)^T\phi(x) + b^i.$$

After obtaining the decision functions, the class label for any unlabeled instance x is assigned according to the following rule:

$$\text{class of x} = f(x) = \text{argmax}_{i=1,\dots,q}(f_i(x)).$$

The "one-against-one" strategy constructs $q(q-1)/2$ binary classifiers where each one is trained on data from two classes. To train the classifier separating the ith and the jth classes, the following optimization problem is solved:

$$\min_{w^{ij},b^{ij},\xi^{ij}} \frac{1}{2}\|w^{ij}\|^2 + C\sum_t \xi_t^{ij}, \tag{3.11a}$$

$$\text{s.t. } (w^{ij})^T\phi(x_t) + b^{ij} \geq 1 - \xi_t^{ij}, \text{ if } y_t = i, \tag{3.11b}$$

$$(w^{ij})^T\phi(x_t) + b^{ij} \leq -1 + \xi_t^{ij}, \text{ if } y_t = j, \tag{3.11c}$$

$$\xi_t^{ij} \geq 0. \tag{3.11d}$$

The decision functions of binary classifiers is obtained by

$$f_{ij}(x) = (w^{ij})^T\phi(x) + b^{ij}.$$

After obtaining the decision functions, the class label for any unlabeled instance x is decided by the "Max-Wins" voting method [13], which we will discuss later.

The DAGSVM [23] is similar to the "one-against-one" method, except that it uses a rooted binary Directed Acyclic Graph (DAG) to organize the $q(q-1)/2$ binary classifiers obtained from the SVM training (3.10). The binary classifiers are embraced as the nodes of the graph. Given an unknown sample

x, we traverse a path from the start node of the graph to one of q end node of the graph by evaluating $q-1$ binary classifiers. The label of the end node decides the class label of the instance.

Obviously, M-SVM can be used to solve ranking problems. Taking "one-against-one" strategy as an example, we describe the algorithm as follows.

- Setting the Parameters: Decide the kernel function K and the parameter C for the binary classifiers.

- Training Process:

  Input: $(x_i, y_i),\ i = 1, ..., n$, where

  x_i is the attributes of the ith instances in the training set;

  u_i is the ranks of the ith instances in the training set;

  Step 1:

  Solve (3.11) to obtain the parameters $(w^{ij},\ b^{ij},\ i < j)$ of the binary classifiers.

  Step 2:

  Construct the decision functions of the classifiers by:

$$f_{ij}(x) \equiv \begin{cases} (w^{ij})^T\phi(x) + b^{ij}, & \text{if } i < j \\ -(w^{ji})^T\phi(x) - b^{ji} = -f_{ji}(x), & \text{if } i > j. \end{cases}$$

- Test or Use:

  Input: Any unlabeled instance x^c.

  Step 1: Calculate $V_i(x^c) = \sum\limits_j s(f_{ij}(x^c))$, where

$$s(t) = \begin{cases} 1, & \text{if } t > 0 \\ 0, & \text{if } t \leq 0. \end{cases}$$

.

  Step 2: $\text{Rank}(x^c) = \text{argmax}_{i=1,\dots,q}(V_i(x))$

All the described multi-class SVM methods: one-against-one, one-against-all, and DAGSVM, are solving the ranking problem by treating it as generic multi-class classification problem and neglecting the ranking information. Although the DAGSVM uses a list of natural numbers as class labels and uses the list to construct the Directed Acyclic Graph (DAG), the choice of the class order in the list is arbitrary [23]. Thus all of the described multi-class SVM methods can be improved for ranking problems by taking the ranking information into consideration.

Chapter 4

HYBRID ALGORITHMS

In this chapter, we first discuss the advantages and disadvantages of the current methods for ranking problems. Based on the discussion, we propose to use fuzzy voting in our hybrid algorithm for ranking problems. At the end, we apply the idea of fuzzy voting to generic multi-class classification problems.

4.1 The Hybrid Algorithm for Ranking Problems

4.1.1 Comparison of Existing Methods

Both UTADIS-based algorithms and SVM-based algorithms are candidates for ranking problems. We simply compare these two families of methods.

The UTADIS model comes from decision science, where it is usually assumed that the prior knowledge about the correlations between attributes and class labels is available and easy to be characterized by function models. However, the assumption is not true in general. The prior knowledge may be unavailable or inexact for most ranking problems. This explains why the prediction performance of UTADIS is not good [26].

As for the GUTADIS model, although its prediction performance is good, its computational complexity is extremely high, because a time-consuming Mixed Integer Program (MIP) needs to be solved in the GUTADIS model. This prevents the algorithm from applications with a large data set.

We can see that the performance of UTADIS-based algorithms is restricted by the assumptions made, and it is hard to manage the tradeoff between the training errors and the complexity of the classification model.

The SVM-based methods, instead, provide the convenience to control the tradeoff, and they do not need any prior knowledge or assumptions. These are important reasons why SVM-based methods are among the most popular classification methods.

The multi-class SVM (M-SVM) is designed for generic multi-class classification problems, whose loss function is the 0-1 loss defined by (1.1), that is, there is no difference among the losses of any misclassification errors; each misclassification causes identical loss of "1". This loss function is not a good choice for ranking problems, since in ranking problems some misclassification errors are more serious than others. For example, Table 4.1 lists the empiri-

	Predicted by M-SVM, average accuracy: 80.43%							
Class	1	2	3	4	5	6	7	8
1	100.00	0	0	0	0	0	0	0
2	0	100.00	0	0	0	0	0	0
3	0	35.29	58.82	5.88	0	0	0	0
4	0	0	3.57	82.14	14.29	0	0	0
5	0	0	3.03	9.09	81.82	6.06	0	0
6	0	0	0	8.33	33.33	58.33	0	0
7	0	0	0	0	33.33	0	66.67	0
8	0	0	0	0	66.67	0	33.33	0

Table 4.1: M-SVM for the Country Risk Classification problem

cal prediction accuracy of M-SVM method for the Country Risk Classification problem. The number in row i and column j denotes the probability of a sample in class i being misclassified to class j. We notice that the number in row 8 and column 5 is 66.60%, which means it is much possible that a "CC-D" (class 8) country is misclassified as an "BB" (class 5) country. This kind of misclassification errors is more serious than misclassifying a "CC-D"(class 8) country to "CC"(class 7). Thus, this kind of misclassifications should be penalized heavier in loss function so that the loss function can guide the training process of the classifier to avoid them. It also indicates that the loss function used in ranking problems should be distance-based loss function, such as ε-intensive loss, defined in (1.3). Motivated by this observation, we propose to modify the M-SVM method so that it can handle the distance-based loss functions.

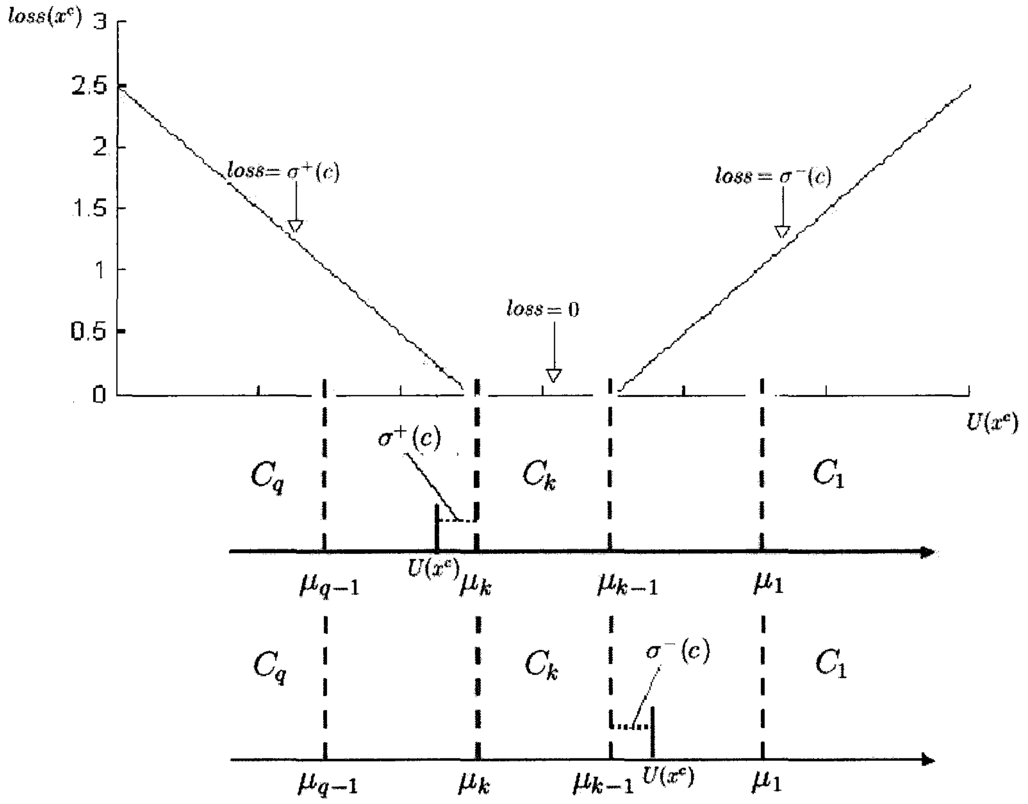


Figure 4.1: The loss function of the UTADIS model.

One interesting issue we would like to discuss is the SVR. If we consider the class labels in ranking problems as target values in regression problems, we can use ε-SVR to solve ranking problems. Suppose the class labels are represented by natural numbers, i.e., $\mathcal{Y} = \{1, 2, ..., q\}$, the loss function of ε-SVR with $\varepsilon = 0.5$ is equivalent to that of the UTADIS model with fixed $\mu_i = i + 0.5$, $i = 1, ..., q-1$. The comparison of Figure 4.1 and Figure 3.5 shows the equivalence clearly. Since ε-SVR takes the advantage of SVM methods and embraces the distance-based loss function, it seems that ε-SVR is a better choice for ranking problems than either UTADIS or M-SVM.

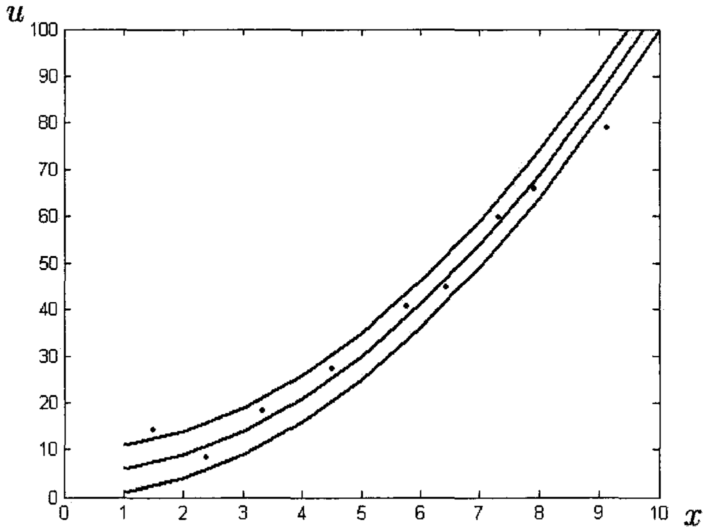


Figure 4.2: A smooth function estimated by ε-SVR.

Unfortunately, the empirical results (listed in Chapter 5) indicate that the prediction performance of ε-SVR for ranking problems is not very good.

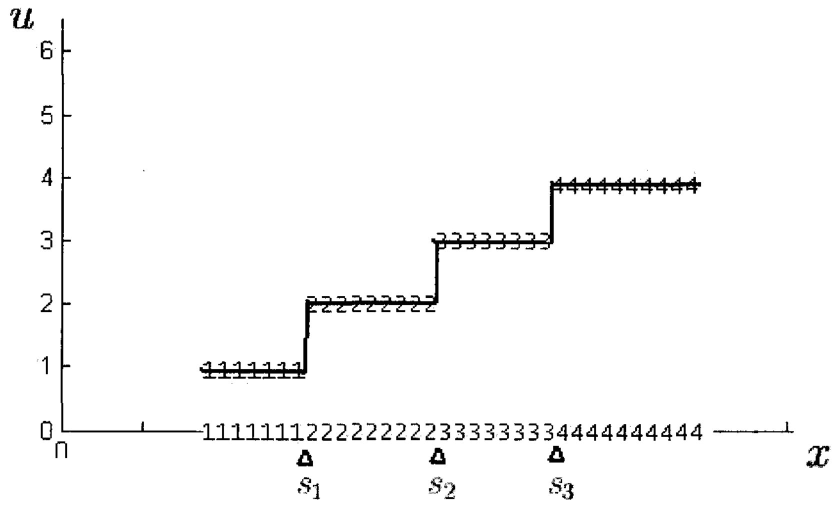


Figure 4.3: A stair-like function need to be estimated in ranking problem.

The reason is that the ε-SVR is designed to predict continuous real numbers. It is used to construct a smooth function whose ε-tube covers most of the data points, as shown in Figure 4.2. However, when we use SVR to solve ranking problems, the functions that need to be estimated are typically not very smooth since the target values (the ranks) are discrete. The fact can be illuminated by a trivial one dimensional ranking problem as shown in Figure 4.3. In this problem, each pair of adjacent classes can be separated by several points (denoted by s_1,s_2, and s_3). If we try to use SVR, we have to estimate a stair-like function shown by the dark line, since the data points are concentrated in several layers. SVR can hardly estimate such a stair-like function well. The UTADIS model, instead, can estimate such a stair-like function by a piecewise-linear monotone function. This observation motivates us to combine the SVM method with the UTADIS model to achieve a better algorithm for ranking problems.

4.1.2 Construction of Binary Classifiers

In our hybrid algorithm for ranking problems, we first construct $q-1$ binary classifiers where the ith classifier is trained by taking the instances from the first i classes as positive instances, and the rest instances as negative ones, that is, the ith SVM training solves the following optimization problem:

$$\min_{w^i,b^i,\xi^i} \frac{1}{2}\|w^i\|^2 + C\sum_{j=1}^{n}\xi_j^i \tag{4.1a}$$

$$\text{s.t. } (w^i)^T\phi(x_j) + b^i \geq 1 - \xi_j, \ \forall y_j \leq i, \tag{4.1b}$$

$$(w^i)^T\phi(x_j) + b^i \leq -1 + \xi_j, \ \forall y_j > i, \tag{4.1c}$$

$$\xi_j^i \geq 0. \tag{4.1d}$$

The reason why we use these $q-1$ classifiers can be attributed to the specific properties of ranking problems. These $q-1$ classifiers are the smallest collection of binary classifiers containing all necessary separation boundaries.

4.1.3 Fuzzy Voting Combination of Binary Classifiers

After we construct $q-1$ binary SVM classifiers, for each instance c, which is represented by x^c in training set, we calculate its decision function values of the $q-1$ classifiers, and take the values as new attributes of the instance, i.e., we calculate

$$v_i^c = f_i(x^c) = (w^i)^T\phi(x^c) + b_i;\ i = 1, ..., q-1, \tag{4.2}$$

where, $f_i(x)$ is the decision function of the ith binary classifier, and obtained v_i^c is the ith new attribute of c. This is actually a transformation from the original space $\mathcal{X}$ to a new space $\mathcal{F}$. Now each instance c in the training set is represented by $q-1$ new attributes v_i^c, $i = 1, ..., q-1$ in the transformed data set. In the next step, we use the transformed data set and the original class labels (the ranks) to train the UTADIS model.

By using the UTADIS model, we typically assume that the correlations between the new attributes v_i, $i = 1, ..., q-1$ and class labels y are monotone. We can see the assumption is reasonable by examining Figure 4.4, which demonstrates the semi-monotone correlations between the decision function values v_1, v_2 and the class label y in the application of Country Risk Classification.

The following LP problem is proposed to train the UTADIS-based com-

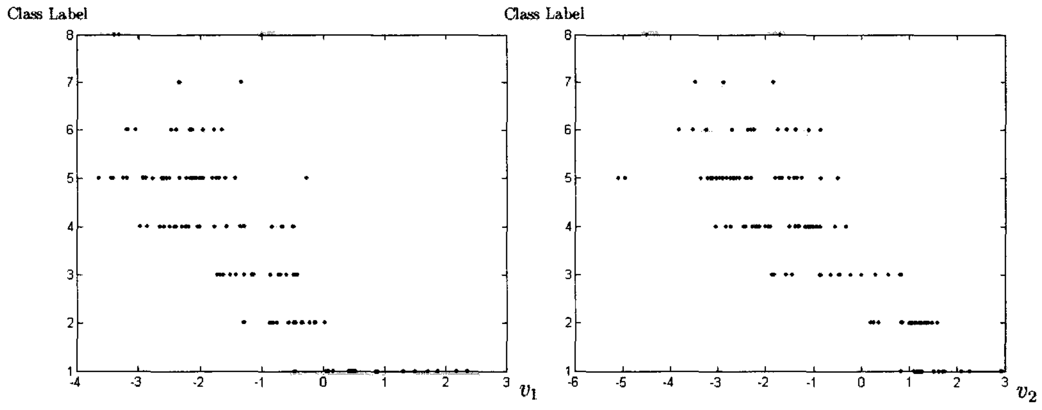


Figure 4.4: The semi-monotone relation between v_1 v_2 and the class label y.

bination:

$$\min_{\sigma,\mu,U} \sum_{c\in C_k:k<q} \sigma^+(c) + \sum_{c\in C_k:k>1} \sigma^-(c) \tag{4.3a}$$

$$\text{s.t.} \sum_{i=1}^{q-1} U_i(f_i(x^c)) - \mu_k + \sigma^+(c) \geq 0,\ \forall 1 \leq k \leq q-1,\ \forall c \in C_k, \tag{4.3b}$$

$$\sum_{i=1}^{q-1} U_i(f_i(x^c)) - \mu_{k-1} - \sigma^-(c) \leq -\delta,\ \forall 2 \leq k \leq q,\ \forall c \in C_k, \tag{4.3c}$$

$$\mu_{k-1} - \mu_k \geq s,\ \forall k = 2, \cdots, q-1, \tag{4.3d}$$

$$U_i^{j+1} - U_i^j \geq 0,\ \forall i = 1, \cdots, q-1,\ \forall j = 1, \cdots, r_i - 1, \tag{4.3e}$$

$$U_i^1 = 0, U_i^{r_i} = 1,\ \forall i = 1, \cdots, q-1, \tag{4.3f}$$

$$\sigma^+(c) \geq 0,\ \sigma^-(c) \geq 0,\ \forall c, \tag{4.3g}$$

where q is the number of classes, and thus the number of attributes is $q-1$,

C_k represents the kth class, $f_i(x^c)$ is the ith new attribute of instance c, which is actually the decision function value of the ith binary classifier on instance x^c.

An interesting way to understand why we use the UTADIS model to combine these binary classifiers is that we actually proposed a "fuzzy" voting method instead of the "Max-Wins" voting method to combine the binary classifiers. In other words, when a classifier "votes" an instance c for or against a class, the classifier uses a real number in $[0, 1]$ as its "fuzzy" vote instead of the exact vote in $\{0, 1\}$ in "Max-Wins".

In this algorithm, the "fuzzy" vote of the ith classifier for samples c is actually $U_i(f_i(x^c))$. Recall that $f_i(x^c)$ is the decision function value of the ith classifier on sample c, and U_i is a piecewise-linear monotone function, which normalizes $f_i(x^c)$ to some value between 0 and 1. In brief, the "fuzzy" vote of the ith classifier is its decision function value normalized by function U_i. The votes from $q - 1$ binary classifiers are summed up to get the value of the utility function: $U(x^c) = \sum_i U_i(f_i(x^c))$. This value can be considered as the total votes on the instance c, which determines the rank of the instance.

In the optimization of UTADIS training, we search the optimal criterion functions $U_i(t)$, $i = 1, ..., q-1$, which are actually the optimal voting strategies of the binary classifiers, so that the loss of training errors, defined by (4.3 a), is minimized.

The benefits of this "fuzzy" voting are obvious. First, the additional information from the decision function values of the binary classifiers can help to improve the prediction performance. Second the ε-intensive loss is minimized in UTADIS training.

In conclusion, we have the following hybrid algorithm for ranking problems:

- Setting the Parameters: Decide the kernel function K and the value of C for the binary classifiers.

- Training Process:

  Input: $(x_i, y_i),\ i = 1, ..., n$, where

  x_i is the attributes of the ith instances in the training set;

  u_i is the ranks of the ith instances in the training set;

  Step 1:

  Solve (4.1) to obtain the parameters $(w^i,\ b^i,\ i = 1, ..., q-1)$ of the $q-1$ binary classifiers.

  Step 2:

Construct the decision functions of the classifiers by: $f_i(x) \equiv (w^i)^T \phi(x) + b^i$;

Step 3:

Construct a new data matrix $F : n \times (q-1)$, whose entities $F(i,j)$ is calculated by: $F(i,j) = f_j(x_i)$.

Step 4:

Taking F as the new attribute matrix of training set, solve (4.3) to construct the piecewise-linear criterion functions $U_i(t)$ and obtain the boundary values: $\xi_i,\ i = 1, ..., q-1$.

- Test or Use:

Input: Any unlabeled instance x^c

Step 1:

Calculate the summation of the fuzzy votes by:

$$U(x^c) = \sum_i U_i(f_i(x^c)). \tag{4.4}$$

Step 2:

Compare $U(x^c)$ with the boundary values $\xi_i,\ i = 1, ..., q-1$ to decide the rank of instance x^c.

We name our hybrid algorithm for ranking problems as "Fuzzy Voting Support Vector Ranking (FVSVR)". Empirical results (in Chapter 5) show that FVSVR achieves a promising performance in accuracy and it is more efficient than UTADIS and GUTADIS methods.

In the first stage of FVSVR training, the ranking information is used to group the classes, so that only q-1 binary classifiers are constructed. That make FVSVR more efficient than M-SVM in the first stage.

While, in the decision process, FVSVR has to solve a additional LP problem (4.3) to achieve the fuzzy voting functions. The LP problem, which has $4m + (r + 2)(q - 1) - 1$ constraints and $2m + (r + 1)(q - 1)$ valuables, is usually an easier problem than the LP problem of UTADIS training (2.3), which has $4m + nr + q - 1$ constraints and $2m + (r + 1)n$ valuables, because in most of the practical applications, the attribute number n is greater than the class number q.

Since both the QP problem in binary SVM training and the LP problem in UTADIS training are easy to solve, the FVSVR, as a combination of them, is efficient. This augment is supported by experiments in Chapter 5.

4.2 Using the Quadratic Loss Function

In the LP problem (4.3), the misclassification errors are linearly punished. Other loss functions can also be employed into this optimization problem. We here try to use the quadratic loss function, which results in the following Quadratic Programming (QP) problem.

$$\min_{\sigma,\mu,U} \sum_{c\in C_k:k<q} (\sigma^+(c))^2 + \sum_{c\in C_k:k>1} (\sigma^-(c))^2 \tag{4.5a}$$

$$\text{s.t.} \sum_{i=1}^{q-1} U_i(f_i(x^c)) - \mu_k + \sigma^+(c) \geq 0,\ \forall 1 \leq k \leq q-1,\ \forall c \in C_k, \tag{4.5b}$$

$$\sum_{i=1}^{q-1} U_i(f_i(x^c)) - \mu_{k-1} - \sigma^-(c) \leq -\delta,\ \forall 2 \leq k \leq q,\ \forall c \in C_k, \tag{4.5c}$$

$$\mu_{k-1} - \mu_k \geq s,\ \forall k = 2,\cdots,q-1, \tag{4.5d}$$

$$U_i^{j+1} - U_i^j \geq 0,\ \forall i = 1,\cdots,q-1,\ \forall j = 1,\cdots,r_i-1, \tag{4.5e}$$

$$U_i^1 = 0, U_i^{r_i} = 1,\ \forall i = 1,\cdots,q-1, \tag{4.5f}$$

$$\sigma^+(c) \geq 0,\ \sigma^-(c) \geq 0,\ \forall c, \tag{4.5g}$$

The only difference between (4.3) and (4.5) is in the objective functions. Experimental results in Chapter 5 shows that the effect of quadratic loss function is insignificant.

4.3 Fuzzy Voting for Generic M-SVM

In the "fuzzy" voting method proposed in the first section of this chapter, the normalized decision function values of the $q-1$ binary classifiers are used as the "fuzzy" votes. The rank of a sample c is decided by the summation of these "fuzzy" votes. This idea can be extended to the generic multi-class SVM (M-SVM) to improve the current combination methods.

As we have mentioned before, there are three different combination strategies by which the binary SVM classifiers are combined into multi-class classifiers: "one-against-one" [18], "one-against-all" [3] and DAGSVM [23]. In the following part of this section, we will propose our "fuzzy" voting method for the "one-against-one" strategy, which is called "Fuzzy Voting Support Vector Machine (FVSVM)". In the following discussion, we denote C_{ij} as the binary classifier that separate class i and class j, denote C_i as the ith class, and denote c as an instance.

4.3.1 Fuzzy Voting Combination for "One-Against-One" Strategy

Recall the algorithm we mentioned in Chapter 3, the "one-against-one" strategy constructs $q(q-1)/2$ binary classifiers each of which is trained on data from two classes. The "Max-Wins" voting method [13] is used to combine the binary classifiers into multi-class classifiers.

In the "Max-Wins" voting method, we calculate

$$V_i(x^c) = \sum_{j=1,\ldots,q, j \neq i} s(f_{ij}(x^c)) \tag{4.6}$$

for each instance c and each class i, where x^c denotes the data point of the sample c. $f_{ij}(x)$ is the decision function of the binary classifier that is trained by taking the samples from class i as positive ones and the samples from class j as negative ones. Thus, there are two decision functions of a classifier $C_{ij}(x)$: $f_{ij}(x)$ and $f_{ji}(x)$, and obviously, $f_{ij}(x) = -f_{ji}(x)$. In (4.6), function $s(t)$ maps the decision function value to $0, 1$ as follows:

$$s(t) = \begin{cases} 1, & \text{if } t > 0 \\ 0, & \text{if } t \leq 0. \end{cases} \tag{4.7}$$

In brief, (4.6) means that when classifier C_{ij} decides that a sample c is in class i rather than in class j ($f_{ij}(x^c) > 0$), the classifier contributes 1 as its vote to $V_i(x^c)$, otherwise it contributes 0. For each class i and each instance c, $V_i(x^c)$, then, is the summation of the votes from the binary classifiers in favor of the decision that the instance c is in class i. This is the "voting" process.

Then, the class label of sample c is decided by:

$$\text{Class Label of } c = argmax_{i=1,\ldots,q} V_i(x^c), \tag{4.8}$$

i.e., we decide that instance c belongs to class i if class i gets the maximum votes from the binary classifiers. This is what "Max-wins" voting means. We call this voting strategy as the "exact" voting since the votes from classifiers are either exact 1 or exact 0.

Similar as the approach made for ranking problems, we propose our "fuzzy" voting instead of the "exact" voting to combine the binary classifiers for the generic multi-class classification problems. We replace the function $s(t)$ in (4.6) by a piecewise-linear monotone function $U_{ij}(t)$ that normalizes (maps) the decision function value $f_{ij}(x)$ to a real value between 0 and 1, i.e., the "fuzzy" version of (4.6) is

$$V_i(x^c) = \sum_{j=1,\ldots,q,j\neq i} U_{ij}(f_{ij}(x^c)). \tag{4.9}$$

For an instance c, when classifier C_{ij} votes it for or against the class i, the classifier uses its normalized decision function value $U_{ij}(f_{ij}(x^c))$ instead of "0 or 1" as its "fuzzy" vote. For each class i and each instance c, $V_i(x^c)$, again, is the summation of the votes from the binary classifiers in favor of the decision that the instance c is in class i.

Note that for an instance c, any binary classifier C_{ij} contributes two votes: v_1 for class i and v_2 for class j. In exact voting, $v_1, v_2 \in \{0, 1\}$ and $v_1+v_2 = 1$. In "fuzzy" voting, $v_1 = U_{ij}(f_{ij}(x)) \in [0, 1]$, $v_2 = U_{ji}(f_{ji}(x)) \in [0, 1]$, and we still want to keep the constraint: $v_1 + v_2 = 1$, so that the weights (authorities) of the binary classifiers in voting are identical, i.e., we want

$$U_{ij}(f_{ij}(x)) + U_{ji}(f_{ji}(x)) = 1, \ \forall i \neq j. \tag{4.10}$$

Since we know that $f_{ij}(x) = -f_{ji}(x)$, we can prove that if we impose the following constraint:

$$U_{ij}(t) = 1 - U_{ji}(-t), \ \forall i \neq j \tag{4.11}$$

on the normalization functions $U_{ij}(t)$, then (4.10) is satisfied. The proof is trivial:

$$\begin{aligned} U_{ij}(f_{ij}(x)) + U_{ji}(f_{ji}(x)) =& 1 - U_{ji}(-f_{ij}(x)) + U_{ji}(f_{ji}(x)) && \text{(4.12a)} \\ =& 1 - U_{ji}(f_{ji}(x)) + U_{ji}(f_{ji}(x)) && \text{(4.12b)} \\ =& 1. && \text{(4.12c)} \end{aligned}$$

Now the remaining problem is how to build the normalization functions $U_{ij}(t)$. Similar to the hybrid algorithm proposed before, we construct a LP to search the optimal normalization functions $U_{ij}(t)$, $j \neq i$, so that the loss of training errors is minimized. The optimization problem is:

$$\begin{aligned} &\min_{\xi, U} \sum_{c,j} \xi_j^c && \text{(4.13a)} \\ &\text{s.t. } V_i(x^c) - V_j(x^c) + \xi_j^c \geq 0, \ \forall c \in C_i, \ \forall i \neq j, && \text{(4.13b)} \\ &V_i(x^c) = \sum_{j=1,\dots,q, j\neq i} U_{ij}(f_{ij}(x^c)), \ \forall i = 1, \dots, q, && \text{(4.13c)} \end{aligned}$$

$$U_{ij}^{l} \geq U_{ij}^{l-1},\ \forall i \neq j,\ \forall l = 2, \cdots, r, \tag{4.13d}$$

$$U_{ij}^{l} = 1 - U_{ji}^{r-l},\ \forall i \neq j,\ \forall l = 1, \cdots, r-1, \tag{4.13e}$$

$$U_{ij}^{1} = 0,\ U_{ij}^{r} = 1,\ \forall i \neq j, \tag{4.13f}$$

$$\xi_j^c \geq 0, \tag{4.13g}$$

where $U_{ij}(t)$ in (4.13 c) is a piecewise-linear function. Suppose the decision function value $f_{ij}(x^c)$ falls in interval $[t_{ij}^l, t_{ij}^{l+1}]$, $U_{ij}(f_{ij}(x^c))$ is calculated by:

$$U_{ij}(f_{ij}(x^c)) = \frac{t_{ij}^{l+1} - f_{ij}(x^c)}{t_{ij}^{l+1} - t_{ij}^{l}} U_{ij}^{l} + \frac{f_{ij}(x^c) - t_{ij}^{l}}{t_{ij}^{l+1} - t_{ij}^{l}} U_{ij}^{l+1}. \tag{4.14}$$

According to the "Max-Wins" principle (4.8), for an instance $c \in C_i$, if $V_i(x^c) \neq \max\limits_{j=1,\ldots,q} V_j(x^c)$, a misclassification error takes place. We use ξ_j^c in (4.13 b) to capture these misclassification errors, which are linearly penalized in the objective function (4.13 a). Constraint (4.13 d) guarantees that the sequence U_{ij}^l, $l = 1, \ldots, r$ is monotone. We import the constraint (4.13 e) so that the normalization functions satisfies (4.11). Also, constraints (4.13 d and f) guarantee that the range of the normalization function is $[0, 1]$

By solving this LP, we get the piecewise-linear monotone normalization functions $U_{ij}(t), i \neq j$, which can be used to assign class labels to unlabeled samples.

In summary, the key issue of our "fuzzy" voting methods is to use the normalized decision function value as the "fuzzy" votes instead of the 0-1 votes used in exact "Max-Wins" voting. The normalization (mapping) functions $U_{ij}(t), i \neq j$ are decided by the LP(4.13). Thus, our "FVSVM" algorithm is described as follows.

- Setting the Parameters: Decide the kernel function K and the value of C for the binary classifiers.

- Training Process:

  Input: $(x_i, y_i),\ i = 1, ..., n$, where

  x_i is the attributes of the ith instances in the training set;

  u_i is the ranks of the ith instances in the training set;

  Step 1:

  Solve (3.11) to obtain the parameters $(w^{ij}, b^{ij},\ i < j)$ of the binary classifiers.

  Step 2:

  Construct $q * (q-1)$ decision functions by:

$$f_{ij}(x) \equiv \begin{cases} (w^{ij})^T\phi(x) + b^{ij}, & \text{if } i < j \\ -(w^{ji})^T\phi(x) - b^{ji} = -f_{ji}(x), & \text{if } i > j; \end{cases} \tag{4.15}$$

  Step 3:

Calculate the decision function value for the instances in training set, and reserve the decision function value of the ith instance as the ith row of the matrix $F : (m \times q(q-1))$.

Step 4:

Taking matrix F as the new attribute matrix of training set, solve (4.13) to construct the piecewise-linear normalization functions $U_{ij}(t)$.

- Test or Use

  Input:

  Any unlabeled instance x^c.

  Step 1: $V_i(x^c) = \sum\limits_j U_{ij}(f_{ij}(x^c))$

  Step 2: Class Label of $x^c = argmax_{i=1,\ldots,q}(V_i(x))$

The described algorithm is called Fuzzy Voting based Support Vector Machine (FVSVM). Empirical results of FVSVM are presented in Chapter 5. It is clear that FVSVM is a little more time-consuming than SVM since they construct the same set of binary SV classifiers and an additional LP problem (4.13) has to be solved in the decision process of FVSVM. But the LP problem we proposed is very easy to solve. In practice, the running time of FVSVM training for each problem is roughly twice of that of M-SVM.

Chapter 5

COMPUTATIONAL RESULTS AND EXPERIMENTS

The purpose of our experiment is to compare the prediction performance of our hybrid algorithms proposed in Chapter 4 with the current algorithms for both ranking problems and generic multi-class classification problems. The performance is compared on several practical classification problems. In this chapter, we first describe the computational environment and the procedure of our experiments. Then, for each classification problem, we briefly introduce the application background, the data source and the properties of the problem. Following this, we propose our empirical results of the candidate algorithms on this problem. The benefits of our approach are demonstrated by the empirical results.

5.1 The Computation Environment

5.1.1 The Solvers

Both the LP problems ((2.3) for UTADIS training, (4.3) for Fuzzy Voting based Support Vector Ranking (FVSVR) and (4.13) for Fuzzy Voting based Support Vector Machine (FVSVM)) and the QP problem (4.5) are solved using SeDuMi v 1.1 [24], which is one of the best software package for optimization over symmetric cones. It was developed by Jos F. Sturm, and the Advanced Optimization Lab at McMaster University continues the development and maintenance of SeDuMi. The developers, Oleksandr Romanko and Imre Pólik, both are Ph.D. students of Dr. Tamás Terlaky.

SeDuMi is not designed specifically for LP, but in practice I found it is more robust and efficient than the LP solver provided by MATLAB.

The Mix-integer nonlinear programming problem (2.4) for GUTADIS training is solved by MINLP [19], which implements a branch and bound algorithm for nonlinearly constrained mix-integer programming problem. MINLP was developed by Roger Flétcher and Sven Leyffer at the University of Dundee.

The SVM solver we used is LIBSVM [5], which is a simple, easy-to-use, and efficient software for SVM classification and regression. It can solve binary SVM, multi-class SVM, ν-SVM, one-class-SVM, ε-SVR, and ν-SVR. It also provides an automatic model selection tool for SVM classification. The package was developed and maintained by Chih-Chung Chang and Chih-Jen Lin [5] from National Taiwan University. The simple MATLAB interface for LIBSVM we used in our experiment was provided by Jun-Cheng Chen, Kuan-Jen Peng, Chih-Yuan Yang, and Chih-Huai Cheng from the National Taiwan University.

Another popular data mining package we used to test the performance of M-SVM is "Weka" [15], which is an open source data mining software package. We used its SMO algorithm, which implements John Platt's Sequential Minimal optimization algorithm for training a SVM classifier. The prediction accuracy of SMO for each problem is listed where we present our empirical result.

We designed several MATLAB 7.0 programs to manage the whole process of our experiment. The tasks include the file operations, the data preprocessing, the construction and maintenance of the matrices and vectors, the organization of the cross validation process and the visualization of the experimental results. The source code of the programs can be found in Appendix I.

Most of our programs are executed on an IBM PC with an P-IV 1.7G CPU and 256M memory, except that we used the "MINLP" solver provided by NEOS [9] to solve the Mix-integer nonlinear programming in GUTADIS training.

5.2 The Process of Experiments

5.2.1 Data Pre-processing

Most of the databases in our experiment are stored as ".txt" files. They are transferred to matrices that can be processed conveniently. Such attributes as the ID of the instances are ruled out to avoid the overfitting problem. Nominal attributes are converted to binary numeric attributes. All the attributes are normalized to the range of $[-1, 1]$. The class labels are represented by natural numbers: $\{1, ..., q\}$, where q is the number of classes. In ranking problems the order of the natural number indicates the order of the classes. However, in generic multi-class Classification problems, these natural numbers are just symbols of the classes.

After data pre-processing, we get a matrix $X : n \times m$, which stores the attribute values of the instances in the database, and a vector $Y : n \times 1$ that stores the class label of the instances. They are inputs of the candidate classification algorithms.

5.2.2 k-fold Cross Validation

The k-fold cross validation method [17], which we utilized to evaluete the power of our algorithms is widely used to evaluate the prediction ability of classification models. In this validation method, the data set is randomly partitioned into k subsets, with the samples in each classes evenly distributed in the k sets. The validation procedure is then repeated k times. Each time, one of the k subsets is left out as the test set, while the others are used as training set. After running the training procedure on the training set, the obtained classifier is tested on the test set and the test errors are recorded. When all the k validation procedures are finished, the average accuracy is calculated as the approximation of the predication accuracy of the classifier.

We can see that each validation procedure takes a $\frac{k-1}{k}$ fraction of the data set as training set. On the one hand, a smaller k indicates a smaller proportion of the training set, hence indicates an underestimated classifier with poorer generalization ability. On the other hand, a larger k indicates a validation procedure consuming more time and computational power. In this thesis, we choose $k = 10$, a widely used trade-off, which means 90 percent of the samples is used as training set and 10 percent is used as test set at each validation phase.

5.3 Databases and Experimental Results for Ranking Problems

The experiment is to compare the prediction performance of M-SVM, FVSVR, UTADIS, GUTADIS, and SVR for ranking problems.

We test the candidate algorithms with a group of ranking problems. One of them is the country risk classification problem. We employ it mainly because the UTADIS and GUATDIS models have been successfully applied to this problem. The other two are the computer performance estimation problem and the auto-mpg estimation problem. Both of them are from the UCI Machine Learning Repository [21].

5.3.1 Experiments with the Country Risk Classification Problem

The problem of country risk classification originates from international business. The country risk of a country refers to the risk that the country may not repay its international debt. The estimation of the country risk can help to make the decision of loans and investments to a certain country. The estimation is usually done by constructing a multi-class classifier that ranks the country risks of different countries in terms of their economic and political status.

The database of this problem has two sources: the economic and political data of the countries come from the Would Development Indicators (WDI) [28] database created by The Would Bank. This database records 575 attributes for 207 countries. Xijun Wang [26] selected 40 attributes and 69 countries from the online limited version of WDI as the data set of the Country Risk Classification problem. The country risk levels (class labels) of these 69 countries are provided by Standard and Poors [1].

This problem is a typical ranking problem, since the class labels of this problem: from AAA (class 1) to CC-D (class 8), are ordinal and the misclassification losses are distance-based.

Before we test the SVM-based algorithms on this data set, we have to decide the penalty factor C and kernel function $K(x, z)$ for the binary classifiers. In the whole procedure of our experiment, we use the "RBF" kernel [4] with fixed $\gamma = 0.2$ as our kernel function, since how to select a good kernel is not the topic of this project. For each candidate algorithm, we do the 10-fold Cross Validation 11 times. Each time, we test different penalty factors: $C = 2^{-i}, i = -2, -1, 0, 1, ..., 8$. We pick the best prediction performance from

	Predicted by M-SVM							
Class	1	2	3	4	5	6	7	8
1	20	0	0	0	0	0	0	0
2	0	22	0	0	0	0	0	0
3	0	6	10	1	0	0	0	0
4	0	0	1	23	4	0	0	0
5	0	0	1	3	27	2	0	0
6	0	0	0	1	4	7	0	0
7	0	0	0	0	1	0	2	0
8	0	0	0	0	2	0	1	0

Table 5.1: Confusion matrix of M-SVM on the Country Risk Classification database

the 11 tries as predication performance of the algorithm.

The best prediction performance of M-SVM on the Country Risk Classification database, achieved by trying $C = 2^6$, is demonstrated by the Confusion Matrix X in Table 5.1, where the natural number x_{ij} in row i and column j indicates the number of instances whose actual class label is i but it is clas-

	Predicted by M-SVM, accuracy: 80.43%							
Class	1	2	3	4	5	6	7	8
1	100.00	0	0	0	0	0	0	0
2	0	100.00	0	0	0	0	0	0
3	0	35.29	58.82	5.88	0	0	0	0
4	0	0	3.57	82.14	14.29	0	0	0
5	0	0	3.03	9.09	81.82	6.06	0	0
6	0	0	0	8.33	33.33	58.33	0	0
7	0	0	0	0	33.33	0	66.67	0
8	0	0	0	0	66.67	0	33.33	0

Table 5.2: Performance of M-SVM on the Country Risk Classification database

sified to class j. For example, the seventh row of the matrix X means that there are 3 instances in class 7, 2 of which are correctly classified (to class 7) and 1 of which is misclassified to class 5.

We divide the numbers in each row of X by the sum of this row, the total number of instances in this class, to get a matrix R preseneted in Table

	Predicted by FVSVR, accuracy: 81.16%							
Class	1	2	3	4	5	6	7	8
1	85.00	15.00	0	0	0	0	0	0
2	0	100.00	0	0	0	0	0	0
3	0	29.41	58.82	11.76	0	0	0	0
4	0	0	3.57	89.29	7.14	0	0	0
5	0	0	0	12.12	87.88	0	0	0
6	0	0	0	0	41.67	58.33	0	0
7	0	0	0	0	0	66.67	33.33	0
8	0	0	0	0	0	33.33	33.33	33.33

Table 5.3: Performance of FVSVR on the Country Risk Classification database

5.2. The entry r_{ij} in matrix R is the correct/incorrect classification rate from class i to class j. In the following part of this chapter we present this error rate matrix instead of the confusion matrix as the experiment result, since it is more convenient for us to compare the performance of different algorithms based on the misclassification rate. The "accuracy" listed on the top of the table is the average accuracy of the algorithm, which is calculated as follows:

$$\text{AveAccuracy} = \sum_i \frac{n_i}{n}, \tag{5.1}$$

where, n_i is the number of correctly classified instances at the ith validation, and n is the total number of instances.

	Predicted by FVSVR, accuracy: 80.43%							
Class	1	2	3	4	5	6	7	8
1	75.00	10.00	10.00	5.00	0.00	0.00	0.00	0.00
2	0.00	90.91	9.09	0.00	0.00	0.00	0.00	0.00
3	0.00	5.88	70.59	23.53	0.00	0.00	0.00	0.00
4	0.00	0.00	3.57	82.14	14.29	0.00	0.00	0.00
5	0.00	0.00	3.03	0.00	90.91	6.06	0.00	0.00
6	0.00	0.00	0.00	0.00	33.33	66.67	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	66.67	33.33	0.00
8	0.00	0.00	0.00	0.00	0.00	33.33	0.00	66.67

Table 5.4: Performance of FVSVR with quadratic loss on the Country Risk Classification database

We can see from Table 5.2 that although the accuracy of M-SVM is good (80.43% by LIBSVM and 81.16 % by SMO in Weka), it allows some

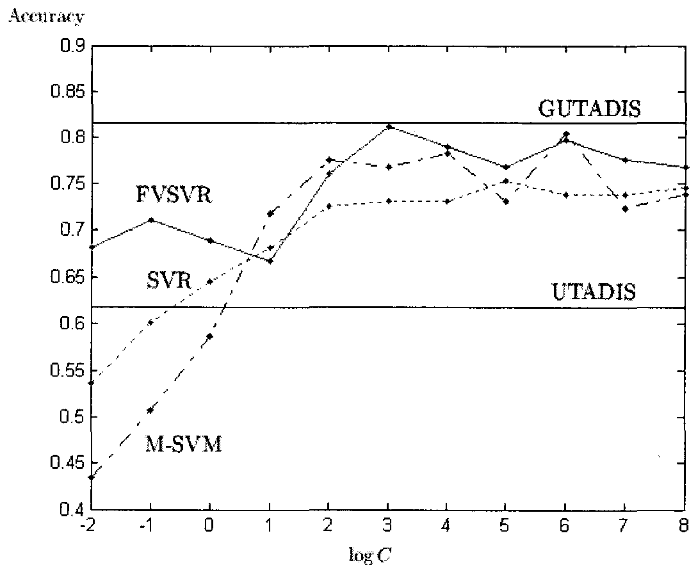


Figure 5.1: The accuracy of candidate algorithms with different panelty factor C.

Algorithms	M-SVM	FVSVR	UTADIS	GUTADIS	SVR
Training Time	1.602	2.415	3.623	24.57	3.472

Table 5.5: Training time of candidate algorithms (seconds per training)

serious misclassifications. For example, the misclassification rate from class 8 to class 5 is as great as 66.67%. These serious misclassifications can be avoided by employing the distance-based loss function. We can see the effect of the

the distance-based loss function from the experimental results of FVSVR in Table 5.3. The prediction performance of FVSVR with quadratic loss function is reported by Table 5.4.

The performance of the UTADIS model is not as good as that of M-SVM and FVSVR, (61.59% in average). Although the prediction accuracy of GUTADIS is pretty good (81.16% in average), it is quite time-consuming in computation. The prediction accuracy of ε-SVR is 75.36% in the average, which is not very competitive. The accuracy of candidate algorithms is compared by Figure 5.1. We can see that FVSVR achieves the best performance (81.16%).

In the first stage of the training process of FVSVR, we construct only 7 binary SV classifiers, compared by 28 binary classifiers in M-SVM. In the decision process, we solve a LP problem with 614 constraints and 332 variables compared by the LP problem in the original UTADIS model which has 2599 constraints and 2356 variables. The training times of the candidate algorithms are listed in Table 5.5, which shows that FVSVR is as efficient as M-SVM, UTADIS and SVR. And it is much more efficient than GUTADIS.

	M-SVM, accuracy: 64.59%				
Class	1	2	3	4	5
1	48.39	51.61	0.00	0.00	0.00
2	8.20	81.97	6.56	1.64	1.64
3	3.45	34.48	37.93	17.24	6.90
4	0.00	32.00	36.00	16.00	16.00
5	0.00	0.00	7.94	4.76	87.30

Table 5.6: Performance of M-SVM on the Computer Hardware database

5.3.2 Experiments with the Computer Performance Estimation Problem

This problem is to estimate the relative performance capabilities of computers in terms of their cycle time, memory size, etc.

We treat this problem as a five-levels ranking problem by grouping the relative performance of the items into 5 ranks: $(0, 20), [20, 40), [40, 60), [60, 80), [80, Max)$. We have 209 instances in 5 ranks. Each instance has 7 numeric attributes including the class label.

The performance of M-SVM, FVSVR and FVSVR with quadratic loss function for this problem is presented in Table 5.6, Table 5.7 and Table 5.8

respectively. Since the correlations between the attributes and the ranks are

	FVSVR, accuracy: 67.94%				
Class	1	2	3	4	5
1	58.06	41.94	0.00	0.00	0.00
2	11.48	70.49	11.48	6.56	0.00
3	0.00	20.69	51.72	20.69	6.90
4	0.00	16.00	32.00	44.00	8.00
5	0.00	0.00	4.76	7.94	87.30

Table 5.7: Performance of FVSVR on the Computer Hardware database

	FVSVR, accuracy: 67.94%				
Class	1	2	3	4	5
1	58.06	38.71	3.23	0.00	0.00
2	8.20	70.49	14.75	6.56	0.00
3	0.00	17.24	48.28	27.59	6.90
4	0.00	16.00	32.00	48.00	4.00
5	0.00	0.00	6.35	6.35	87.30

Table 5.8: Performance of FVSVR with quadratic loss on the Computer Hardware database

monotone, the obtained accuracy of UTADIS and GUTADIS (62.20%) is as good as M-SVM (64.59 %by LIBSVM and 63.94 % by SMO in Weka) but still worse than FVSVR (67.94%). The accuracy of ε-SVR is 59.41% which is not good. The accuracy is compared by Figure 5.3 We can see that the FVSVR

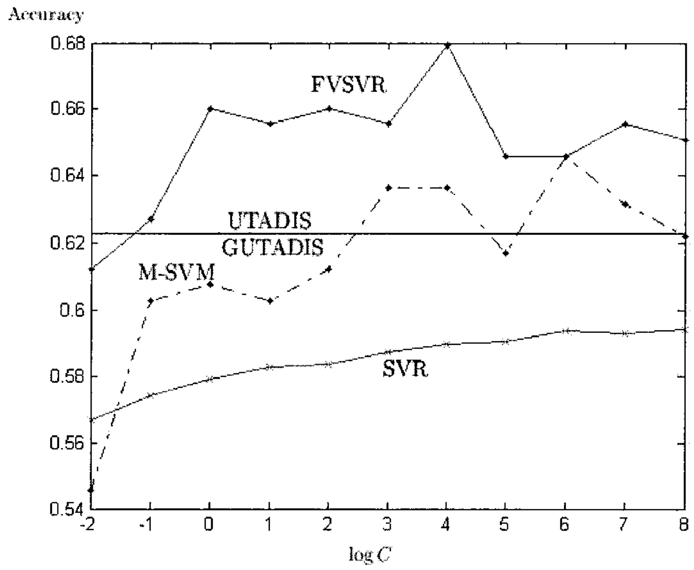


Figure 5.2: The accuracy of candidate algorithms with different panelty factor C.

achieves the best performance (67.94%).

Algorithms	M-SVM	FVSVR	UTADIS	GUTADIS	SVR
Training Time	0.1156	0.8412	1.7525	6.6850	0.2983

Table 5.9: Training time of candidate algorithms (seconds per training)

In the first stage of the training process of FVSVR, we construct only 4 binary SV classifiers, compared by 10 binary classifiers in M-SVM. In the decision process, we solve a LP problem with 871 constraints and 450 variables compared by the LP problem in the original UTADIS model which has 1146 constraints and 730 variables. The training times of the candidate algorithms are listed in Table 5.9, which shows that FVSVR is a little time-consuming than SVM and SVR, but is more efficient than UTADIS and GUTADIS.

5.3.3 Experiments with the Auto-mpg Estimation Problem

	Predicted by M-SVM, accuracy: 64.32%					
Class	1	2	3	4	5	6
1	83.02	16.98	0	0	0	0
2	13.27	70.41	14.29	2.04	0	0
3	1.28	17.95	51.28	25.64	2.56	1.28
4	0	2.60	15.58	70.13	9.09	2.60
5	0	0	1.79	16.07	55.36	26.79
6	0	0	2.78	5.56	41.67	50.00

Table 5.10: Performance of M-SVM on the Auto-mpg database

The Auto-mpg database concerns city-cycle fuel consumption in Miles Per Gallon, to be predicted in terms of such attributes as horsepower and

	Predicted by FVSVR, accuracy: 66.08%					
Class	1	2	3	4	5	6
1	81.13	18.87	0	0	0	0
2	11.22	71.43	17.35	0	0	0
3	1.28	12.82	61.54	21.79	2.56	0
4	0	0	22.08	64.94	11.69	1.30
5	0	0	1.79	19.64	57.14	21.43
6	0	0	2.78	0	41.67	55.56

Table 5.11: Performance of FVSVR on the Auto-mpg database

weight of a vehicle.

We treat this problem as a six-levels ranking problem by grouping the MPGs into 6 levels: $(0, 15), [15, 20), [20, 25), [25, 30), [30, 35), [35, Max)$. We have 398 instances in 6 ranks. Each instance has 8 attributes including the class label.

The performance of M-SVM, FVSVR and FVSVR with quadratic loss are presented in Tables 5.10 5.11 and 5.12 respectively.

Since the background knowledge about the correlations between the attributes and the ranks is unavailable in this problem, to test the performance

	Predicted by FVSVR, accuracy: 67.84					
Class	1	2	3	4	5	6
1	83.02	16.98	0.00	0.00	0.00	0.00
2	13.27	71.43	15.31	0.00	0.00	0.00
3	0.00	15.38	61.54	20.51	2.56	0.00
4	0.00	0.00	22.08	62.34	14.29	1.30
5	0.00	0.00	1.79	14.29	66.07	17.86
6	0.00	0.00	2.78	5.56	27.78	63.89

Table 5.12: Performance of FVSVR with quadratic loss on the Auto-mpg database

of the UTADIS model, we have to arbitrarily assume that each attribute has an increasing correlation with the rank of MPGs, that is a greater attribute value indicates a higher rank. Thus, the obtained accuracy of UTADIS is not good (52.76%). The accuracy of ε-SVR on this database is 64.82% that is a little bit better than M-SVM (64.32% by LIBSVM and 63.47% by SMO in Weka), but still worst than FVSVR (67.84%). The GUTADIS method achieves an accuracy of 63.47%. The accuracy is compared by Figure 5.2

In the first stage of the training process of FVSVR, we construct only

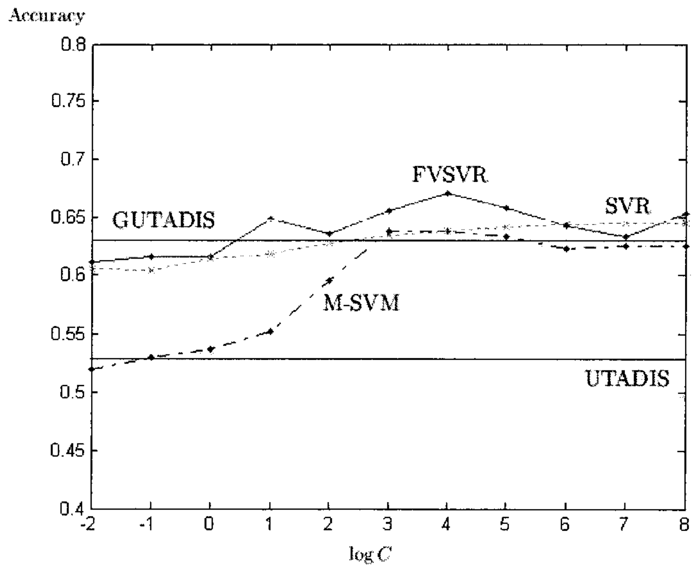


Figure 5.3: The accuracy of candidate algorithms with different panelty factor C.

Algorithms	M-SVM	FVSVR	UTADIS	GUTADIS	SVR
Training Time	0.3305	1.8426	3.5823	34.3292	1.6123

Table 5.13: Training time of candidate algorithms (seconds per training)

5 binary SV classifiers, compared by 15 binary classifiers in M-SVM. In the decision process, we solve a LP problem with 1636 constraints and 836 variables compared by the LP problem in the original UTADIS model which has 1954

constraints and 1160 variables. The training times of the candidate algorithms are listed in Table 5.13, which shows that FVSVR is a little time-consuming than SVM and SVR, but is more efficient than UTADIS and GUTADIS.

5.4 Databases and Experimental Results for Generic Multi-class Classification Problems

The experiment is to compare the prediction performance of M-SVM and FVSVM for generic multi-class classification problems.

The M-SVM and FVSVM are compared with several practical problems which has been used in [13] to test the performance of SVM-based methods. They are the glass identification problem, the DNA identification problem, the vowel recognition problem, the vehicle recognition problem, the iris identificaiton problem, the wine identificaiton problem, the segment recognition problem, and the satimage identification problem. They are generic multi-class classification problems, that is, the ranking information of classes are unavailable or unclear. All of them are from the UCI Machine Learning Repository. [21].

5.4.1 Experiments with the Glass Identification Problem

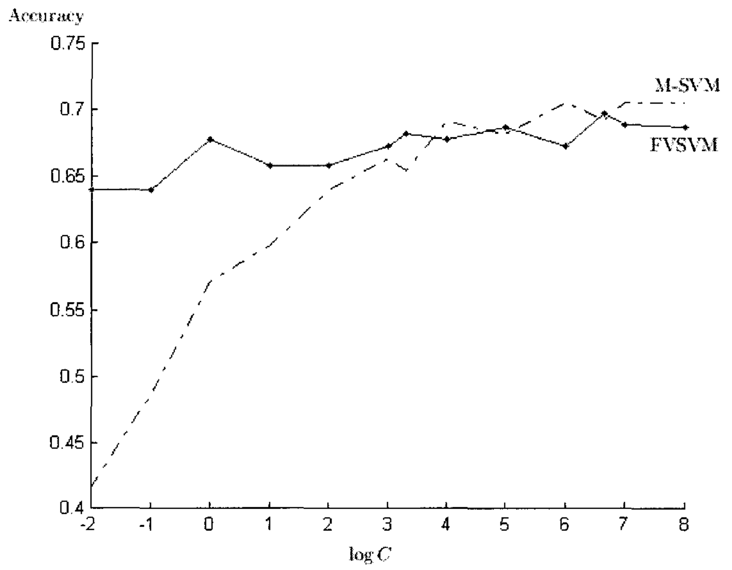


Figure 5.4: The comparison of M-SVM and FVSVM with glass.

The study of classification of types of glass was motivated by criminological investigations. At the scene of the crime, the glass left can be used as evidence, if it is correctly identified. So the task of glass identification is to identify the type of glass by its oxide content (i.e., Na, Fe, K, etc).

In this database, we have 214 instances in 6 classes. Each instance has 10 attributes including the class label.

The performances of M-SVM and FVSVM are compared and demonstrated in Figure 5.4, which shows the prediction accuracy of these two algorithms with different values of the penalty factor C. We can see when C is small (less than 2^2) the prediction accuracy of M-SVM (indicated by the dash-dot line) is terrible. Meanwhile the prediction accuracy of FVSVM (indicated by the solid line) is always acceptable. When C reaches it optimal value. M-SVM and FVSVM achieves similar accuracy.

We justify this result by the following discussion. Remind that the "fuzzy voting" technique improves the performance of the original M-SVM method by extracting more information from the binary classifiers. When C is far away from its optimal value, the binary separation boundaries may not be optimal, then the "fuzzy voting" technique can use the additional information from the decision function to adjust the combination of the binary separations so that the obtained multi-class classifier achieves a better performance. However, when C is close to its optimal value, the binary separations are nearly optimal, the binary separation boundaries themselves are representative enough to construct the multi-class separation. Then, the information from the decision functions is usually redundant, and thus FVSVM can hardly do a better job then M-SVM in this situation.

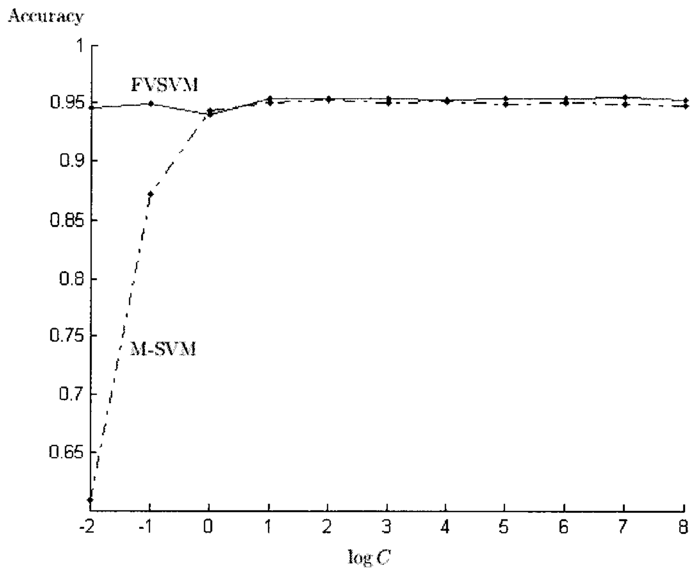


Figure 5.5: The comparison of M-SVM and FVSVM with DNA.

5.4.2 Experiments with the DNA Identification Problem

In this problem, we have 2000 training instances and 1186 testing instances from 3 classes. Each of them has 181 binary attributes including the class label.

The comparason of M-SVM and FVSVM is demonstrated by Figure 5.5, from which, we can see that FVSVM is much more robust.

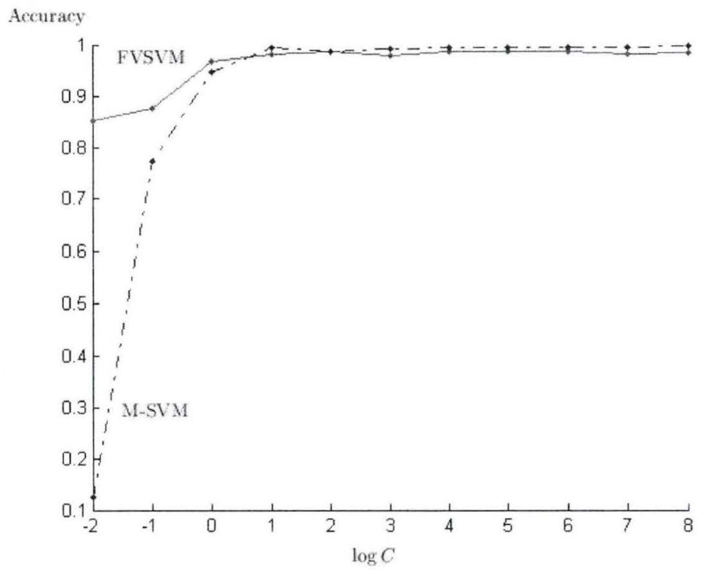


Figure 5.6: The comparison of M-SVM and FVSVM with vowel.

5.4.3 Experiments with the Vowel Recognition Problem

In this problem, we have 528 instances from 11 classes. Each of them has 11 attributes including the class label.

The comparason of M-SVM and FVSVM is demonstrated by Figure 5.6, from which, we can see that FVSVM is more robust.

5.4.4 Experiments with the Vehicle Recognition Problem

In this problem, we have 846 instances from 4 classes. Each of them has 19 attributes including the class label. The comparason of M-SVM and FVSVM

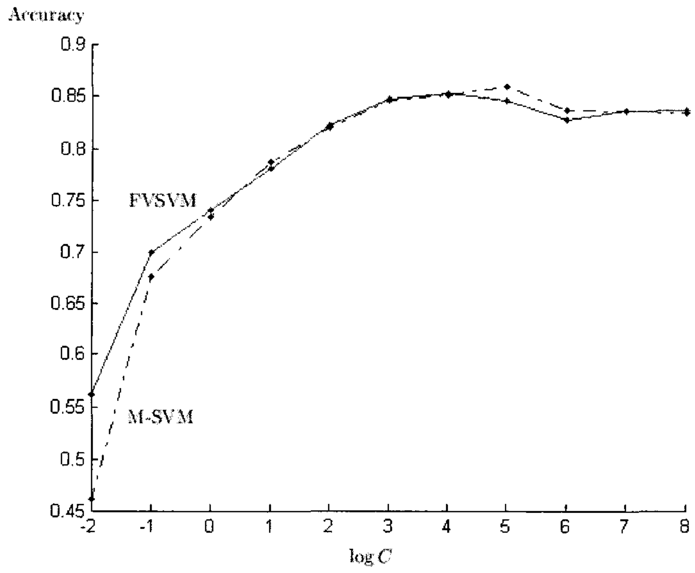


Figure 5.7: The comparison of M-SVM and FVSVM with vehicle.

is demonstrated by Figure 5.7, from which, we can see that FVSVM is a little more robust.

5.4.5 Experiments with the Iris Identificaiton Problem

In this problem, we have 150 instances from 3 classes. Each of them has 5 attributes including the class label. The comparason of M-SVM and FVSVM

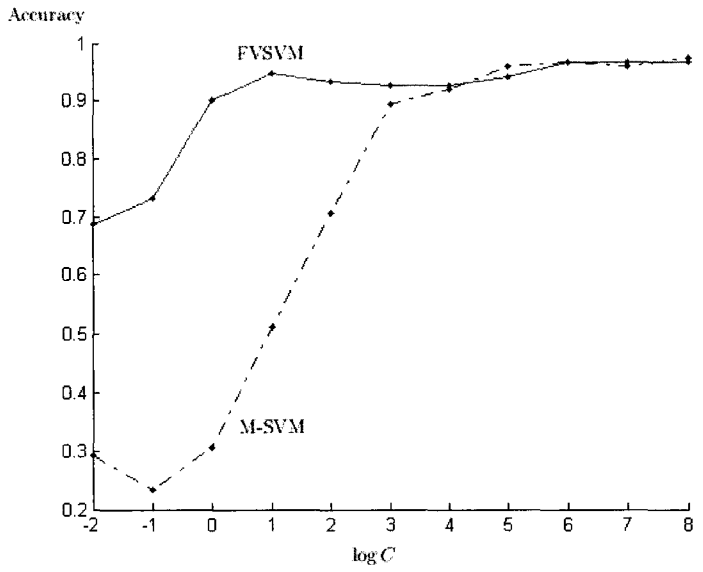


Figure 5.8: The comparison of M-SVM and FVSVM with iris.

is demonstrated by Figure 5.8, from which, we can see that FVSVM is much more robust.

5.4.6 Experiments with the Wine Identificaiton Problem

In this problem, we have 178 instances from 3 classes. Each of them has 14 attributes including the class label. The comparason of M-SVM and FVSVM is demonstrated by Figure 5.9, from which, we can see that FVSVM is much more robust.

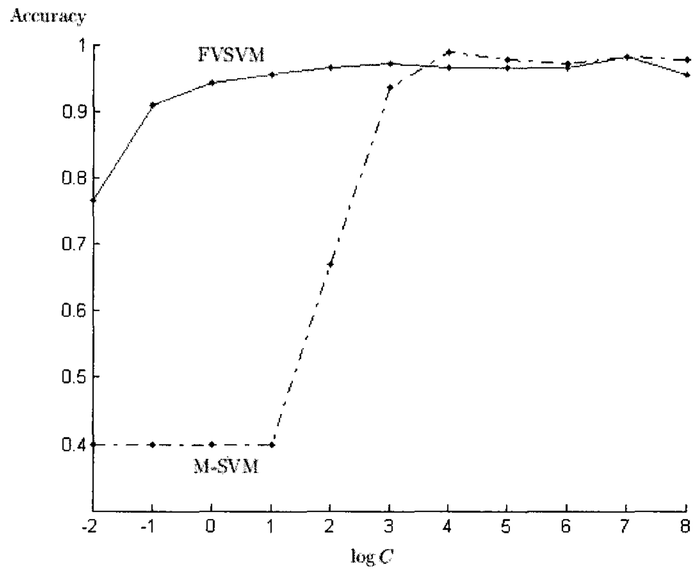


Figure 5.9: The comparison of M-SVM and FVSVM with wine.

5.4.7 Experiments with the Segment Recognition Problem

In this problem, we have 2310 instances from 7 classes. Each of them has 20 attributes including the class label. The comparason of M-SVM and FVSVM is demonstrated by Figure 5.10, from which, we can see that FVSVM and M-SVM have similar performance.

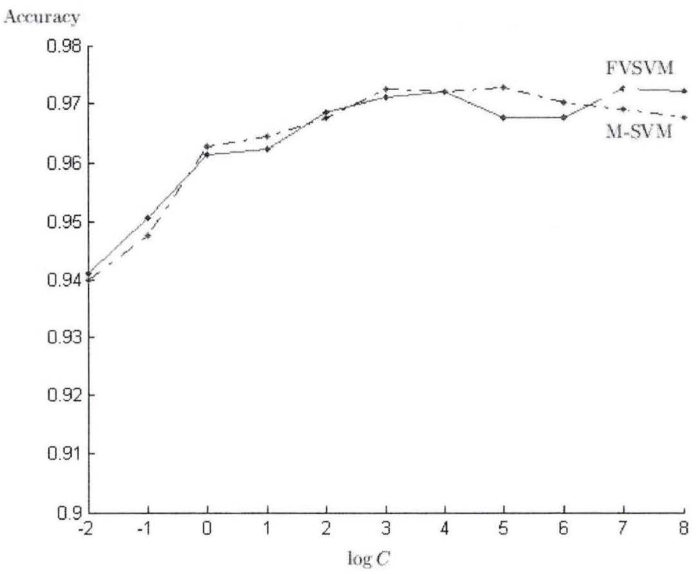


Figure 5.10: The comparison of M-SVM and FVSVM with segment.

5.4.8 Experiments with the Satimage Recognition Problem

In this problem, we have 4435 training instances and 2000 testing instances from 6 classes. Each of them has 37 binary attributes including the class label.

The comparason of M-SVM and FVSVM is demonstrated by Figure 5.11, from which, we can see that FVSVM is more robust.

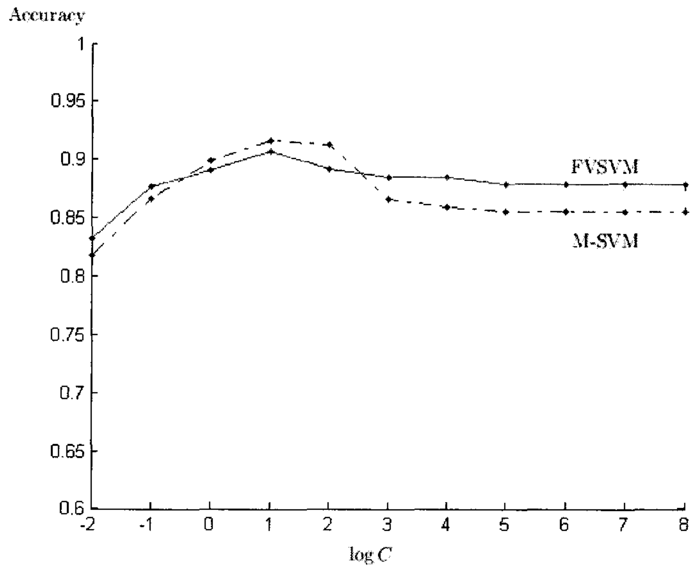


Figure 5.11: The comparison of M-SVM and FVSVM with satimage.

5.4.9 Time Complexity

It is obviously that FVSVM is more time-consuming than SVM since they use the same set of binary SV classifiers and an additional LP problem has to be solved in the decision process of FVSVM. In practice, the running time of FVSVM training for each problem is roughly triple of that of M-SVM. So our FVSVM is a promising trade-off between the robusticity and the time-complexity of the M-SVM method.

Chapter 6

CONCLUSIONS AND FUTURE WORK

In this thesis, we compare UTADIS based methods with Support Vector Machine based methods for ranking problems. We conclude that although the multi-class SVM (M-SVM) has good performance for generic multi-class classification problems, it fails to control the serious misclassifications for ranking problems. This is mainly due to the fact that M-SVM is not driven by distance-based loss, hence it is unable to distinct the serious misclassifications from the others. We propose a hybrid algorithm that combines M-SVM and UTADIS to solve this problem. In this new algorithm, binary SVM classifiers are combined into a multi-class classifier by the fuzzy voting technique instead of the exact voting technique in M-SVM. Therefore, the new algorithm is named as Fuzzy Voting based Support Vector Ranking (FVSVR) method. Empirical results on the databases of three typical ranking problems show that the FVSVR method achieves better performance in practice than M-SVM, UTADIS and SVR. We also extend the idea of Fuzzy Voting from ranking problems to generic multi-class classification problems, which results in the so called Fuzzy Voting based Support Vector Machine (FVSVM) method. Our empirical results show that FVSVM is insensitive to the choice of the penalty factor C.

There are still a few issues we need to explore for both the FVSVR and FVSVM methods. First, as we have mentioned before, there are three strategies by which binary SVM classifiers are combined to multi-class classifiers: "one-against-one", "one-against-all", and "DAG". The FVSVM method we proposed is just the fuzzy voting version of the "one-against-one" strategy. The same idea can be extended to the "one-against-all" and the "DAGSVM" strategies.

Secondly, in FVSVR and FVSVM the optimization problems of searching the binary SVM classifiers and that of searching the optimal voting are two separate steps. It is an interesting problem to explore how these two steps can be combined together into a single optimization problem.

Bibliography

[1] S. Alexe, P. L. Hammer, A. Kogan and M. A. Lejeune, A Nonrecursive Regression Model for Country Risk Rating, *Technical Report*, Rutgers University, Center for Operations Research, Piscataway, New Jersey, 2003.

[2] J. Bi and K. Bennett, A Geometric Approach to Support Vector Regression, *Neurocomputing, 55*, pp. 79-108, 2003.

[3] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Müller, E. Säckinger, P. Simard, and V. Vapnik, Comparison of Classifier Methods: A Case Study in Handwritten Digit Recognition *ICPR*, pp. 77-87, 1994.

[4] C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, *Data Mining and Knowledge Discovery, 2*, pp. 121-167, 1998.

[5] C. Chang and C. Lin, User Manual for LIBSVM, 2004. http://www.csie.ntu.edu.tw/˜cjlin/libsvm/ / / Accessed on Dec 2005

[6] W. Cohen, R. E. Schapire and Y. Singer, Learning to Order Things, *Advances in Neural Information Processing Systems 10, Morgan Kaufmann*, 1998.

[7] C. Cortes, V. Vapnik, Support-Vector Networks, *Machine Learning, 20*, pp. 273-297, 1995

[8] K. Crammer and Y. Singer, Pranking with Ranking, *In Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, 2001. Link: http://citeseer.ist.psu.edu/crammer01pranking.html

[9] J. Czyzyk, M. Mesnier, and J. Morè, The neos server, *IEEE Journal on Computational Science and Engineering 5*, pp. 68-75, 1998. http://www-neos.mcs.anl.gov/

[10] G. E. Dallal, Introduction to Simple Linear Regression, http://www.tufts.edu/~gdallal/slr.htm

[11] M. Doumpos and C. Zopounidis, Multicriteria Decision Aid Classification Methods, *Vol. 73 of Applied Optimization*, Kluwer Academic Publishers, Norwell, MA, U.S.A. 2002.

[12] J. Han and M. Kamber, *Data Mining, Concepts and Techniques*, The

Morgan Kaufmann Series in Data Management Systems, pp. 279-281, 2001.

[13] C. Hsu and C. Lin, A Comparison of Methods for Multi-class Support Vector Machines, *IEEE Transactions on Neural Networks, 13*, pp. 415-425, 2002.

[14] R. L. Keeney and H. Raiffa, Decisions with Multiple Objectives, Preferences and Value Tradeoffs, *Cambridge University Press*, Cambridge, UK, 1993. http://www.boosting.org/papers/CohSchSin98.pdf

[15] R. Kirkby and E. Frank, Weka Explorer User Guide for Version 3.4, 2005. http://www.cs.waikato.ac.nz/ml/weka/

[16] A. Knobbe, A. Schipper and P. Brockhausen, *Domain Knowledge and Data Mining Process Decisions*, Enabling End-User Datawarehouse Mining, Contract No. IST-1999-11993, pp. 4-5, 2000.

[17] R. A. Kohavi, Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection, *IJCAI*, pp. 1137-1145. 1995.

[18] U. Krebel, Pairwise Classification and Support Vector Machines, in B. Scholkopf, C.J.C. Burges, *Advances in Kernel Methods: Support Vector Learning*, pp. 255-268, MIT Press, 1999.

[19] S. Leyffer, *User Manual for MINLP BB*, University of Dundee, Dundee, U.K. 1999. http://www.gamsworld.org/minlp/

[20] D. Michie, D. J. Spiegelhalter and C. C. Taylor, Statlog Database, *Neural and Statistical Classification*, Englewood Cliffs, N.J. Prentice Hall, 1994, Database from:
http://www.niaad.liacc.up.pt/statlog/

[21] D. J. Newman, S. Hettich, C. L. Blake and C. J. Merz, UCI Repository of Machine Learning Databases, Irvine, CA, University of California Irvine, Department of Information and Computer Science 1998, Database from:
http://www.ics.uci.edu/~mlearn/MLRepository.html

[22] J. Peng, Data Mining: Concepts and Algorithms, *Lecture Notes of CS6TF3* McMaster University, Hamilton, On, Canada, 2004.

[23] J. C. Platt, N. Cristianini and J. Shawe-Taylor, Large Margin DAGs for Multiclass Classification, *Advances in Neural Information Processing Systems 12*, pp. 547-553, MIT Press, 2000.

[24] I. Pólik, *SeDuMi 1.1 R2, User Manual* Advanced Optimization Lab, McMaster U. 2005. http://sedumi.mcmaster.ca/

[25] A. J. Smola and B. Schölkopf, A Tutorial on Support Vector Regression, *Statistics and Computing*, 14, pp. 199-222, 2004.

[26] X. Wang, Country Risk Classification and Multicriteria Decision-Aid. *M.Sc. Thesis*, Dept. Computing and Software, McMaster Univ. Hamilton, ON, Canada, 2004.

[27] J. Weston and C. Watkins, Multi-Class Support Vector Machines, in M.Verleysen, Editor, *Proceedings of ESANN99, Brussels*, 1999.

[28] The World Bank, World Development Indicators on CD-ROM, IBRD World Bank, Washington, D.C. 2002.

[29] C. Zopounidis, and M. Doumpos, Multi-criteria Decision Aid in Financial Decision Making, *methodologies and literature review, Journal of Multi-Criteria Decision Analysis 11*, pp. 167C186, 2002.