T.R.
BAHCESEHIR UNIVERSITY

DESIGNING AN EARLY WARNING SYSTEM FOR STOCK MARKET CRASHES BASED ON MACHINE LEARNING FORECASTING

Master Thesis

MURAT ACAR

İSTANBUL, 2010

T.R.
BAHCESEHIR UNIVERSITY

The Graduate School of Natural and Applied Sciences
Computer Engineering Graduate Program

DESIGNING AN EARLY WARNING SYSTEM FOR STOCK MARKET CRASHES BASED ON MACHINE LEARNING FORECASTING

Master Thesis

Murat ACAR

SUPERVISOR: ASSOC. PROF. DR. ADEM KARAHOCA

İSTANBUL, 2010

T.R.

BAHCESEHIR UNIVERSITY

The Graduate School of Natural and Applied Sciences

Computer Engineering Graduate Program

Name of the thesis: DESIGNING AN EARLY WARNING SYSTEM FOR STOCK MARKET CRASHES BASED ON MACHINE LEARNING FORECASTING

Name/Last Name of the Student: Murat Acar

Date of Thesis Defense: 07.06.2010

The thesis has been approved by The Graduate School of Natural and Applied Sciences.

Asst. Prof. Dr. F. Tunç Bozbura

Director

Signature

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Committee Members Signature

Assoc. Prof. Dr. Adem Karahoca (Supervisor)

Prof. Dr. Nizamettin Aydın

Asst. Prof. Dr. M. Alper Tunga

ACKNOWLEDGEMENTS

It is a pleasure to thank people who made this thesis possible; first and foremost, I would like to thank my family for always being supportive of my wife, son and daughter who have untiringly shown me patience and love. Without them, I never would have been able to get here.

I would like to express my deepest gratitude both personally and professionally, to my advisor, Assoc. Prof. Dr. Adem Karahoca, without whose guide and encouragement the completion of this study would be impossible. I will remember every word of his valuable advices through all my future studies.

Also I would like to add my thanks to committee members Prof. Dr. Nizamettin Aydın and Asst. Prof. Dr. M. Alper Tunga for carefully reading the thesis manuscript and their valuable suggestions.

I would like to thank Computer Engineering Graduate Program Academic Staff for making my years at Bahçeşehir University a great experience.

Finally, I would like to thank all my friends for their motivation.

Murat Acar

ABSTRACT

DESIGNING AN EARLY WARNING SYSTEM FOR STOCK MARKET CRASHES BASED ON MACHINE LEARNING FORECASTING

Acar Murat

The Graduate School of Natural and Applied Sciences, Computer Engineering Graduate Program

Supervisor: Assoc. Prof. Dr. Adem Karahoca

Jun 2010, 59 pages

In this study, we focus on building a financial early warning system (EWS) to predict stock market crashes by using stock market volatility and rising stock prices. The relation of stock market volatility with stock market crashes is analyzed empirically in this study. Also, Istanbul Stock Exchange (ISE) national 100 index data will be used to achieve better results from the point of modeling purpose. A stock market crash risk indicator is computed to predict crashes and to give an early warning signal. Various data mining classifiers are compared to obtain the best practical solution for the financial early warning system. ANFIS model is offered as a means to forecast stock market crashes efficiently. Besides, adaptive neuro fuzzy inference system (ANFIS) will be explained in detail as a training tool for the EWS. The empirical results show that the proposed adaptive neuro fuzzy (NF) model is successful thanks to ANFIS that includes both artificial neural network (ANN) learning ability and the fuzzy logic inference mechanism.

Keywords: Stock market crash; Economic crisis; Early warning system; Adaptive neuro fuzzy inference system; Data mining classifier

ÖZET

BORSA ÇÖKÜŞLERİNİ TAHMİN ETMEK İÇİN BİLGİSAYAR TABANLI BİR ERKEN UYARI SİSTEMİ TASARIMI

Acar Murat

Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Yüksek Lisans Programı

Danışman: Doç. Dr. Adem Karahoca

Haziran 2010, 59 sayfa

Bu çalışmada, borsadaki fiyat değişkenliği ve hisse senetlerinin fiyatları ile ilgili verileri kullanarak, olası borsa çöküşlerini tahmin etmek için bir finansal erken uyarı sistemi (FEUS) geliştirme üzerine odaklandık. Borsalardaki fiyat değişkenliği ile borsa çöküşleri arasındaki ilişki ampirik olarak analiz edilmiştir. Aynı zamanda, modelleme açısından daha iyi sonuçlar almak için, İstanbul Menkul Kıymetler Borsası (IMKB) Ulusal 100 Endeksi de kullanılmıştır. Borsadaki olası çöküşleri tahmin etmek ve bir erken uyarı sinyali verebilmek için bir risk göstergesi hesaplanmıştır. Finansal erken uyarı sistemi konusundaki en pratik çözümü bulmak için çeşitli veri madenciliği sınıflayıcıları birbirleriyle karşılaştırılmıştır. Adaptif Bulanık Yapay Sinir Ağı ile geliştirilen model borsadaki olası çöküşleri önceden etkli bir şekilde tahmin etmek için kullanılabilecek bir araç olarak önerilmektedir. Bunun yanı sıra, geliştirilen erken uyarı sisteminin eğitim aracı olan Adaptif Bulanık Yapay Sinir Ağı detaylı bir şekilde açıklanacaktır. Ampirik sonuçlar, ön görülen adaptif bulanık yapay sinir ağı modelinin, hem yapay sinir ağının öğrenme becerisi ve hem de bulanık mantık mekanizması dolayısıyla başarılı olduğunu göstermiştir.

Anahtar Kelimeler: Borsa çöküşü; Ekonomik Kriz; Erken Uyarı Sistemi; Adaptif Bulanık Yapay Sinir Ağı; Veri madenciliği sınıflayıcıları

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

Adaptive Neuro Fuzzy Inference System	:	ANFIS
Akaike Information Criterion	:	AIC
Area Under Curve	:	AUC
Artificial Neural Network	:	ANN
Bayes Network	:	BN
Bayesian Belief Network	:	BBN
Confusion Matrix	:	CM
Decision Tree	:	DT
Directed Acyclic Graph	:	DAG
Early Warning System	:	EWS
False Negatives	:	FN
False Positives	:	FP
Fuzzy Inference System	:	FIS
Gross Domestic Product	:	GDP
Istanbul Stock Exchange	:	ISE
Log Likelihood	:	LL
Logistic Discrimination	:	LD
Logistic Regression	:	LR
Matrix Laboratory	:	MATLAB
Maximum A Posteriori	:	MAP
Multilayer Perceptron	:	MLP
Neuro Fuzzy	:	NF
Radial Basis Function	:	RBF
Receiver Operating Characteristic	:	ROC
Root Mean Square Error	:	RMSE
Simple Logistic	:	SL
Support Vector Machine	:	SVM
Tree Augmented Naïve Bayes	:	TAN
True Negative Rate	:	TNR
True Negatives	:	TN
True Positive Rate	:	TPR
True Positives	:	TP
United States	:	US
Waikato Environment for Knowledge Analysis	:	WEKA

1. INTRODUCTION

Failures in financial systems may cause financial crises and then the latter may develop into economic fundamental crises that might not be always inevitable results. Economic crises are characterized by sharp falls in both asset prices and currency values. Failures could lead to a stock market crash that is often defined as a sharp dip in share prices of equities listed on the stock exchanges. Rising stock prices and excessive economic optimism may also cause a stock market crash. Although there is no a numerically specific definition of a stock market crash, it can be defined as double-digit percentage losses in a stock market index over a period of several days.

Stock market crashes can provoke recessions, lead to failures in the financial system or consume years of savings and pensions instantaneously. Testing for the existence of log-periodic behavior and attempting to forecast crashes are thus important for financial regulators, risk and portfolio managers, policy makers and financial institutions (Cajueiro, 2009). Generally, in any given field, crashes are extremely difficult to forecast accurately. Forecasting of crashes is one of the most popular research topics in finance. Many theoretical and empirical studies have been done to forecast crashes and many models have been developed to predict the occurrence of such crashes.

With increasing globalization and financial integration, crises in a country could make other countries highly vulnerable to shocks. The United States (US) subprime mortgage crisis also hit the Turkish economy in 2008. The Istanbul Stock Exchange (ISE) decreased from 54708 to 26864 in 2008 because of the rapid decrease in foreign markets and insufficient fresh money entrance. The ISE is the only securities exchange in Turkey. The ISE is a dynamic and growing emerging market with an increasing number of publicly traded companies, state-of-the-art technology and strong foreign participation. The ISE provides a transparent and fair trading environment not only for domestic participants, but also for foreign issuers and investors (http://www.ise.org). Iseri and et al. (2008)'s study indicates that the ISE has very high chaotic phenomena. So prediction on chaotic phenomena is very complex (Iseri, 2008).

Investors are intensely interested in market directions and possibilities of stock market crashes. Therefore behavior patterns of risky market days of this kind should be defined. Relationships among variables derived from the historical financial data should be discovered and a financial early warning system (EWS) should be constructed to forecast stock market crashes. Financial early warning systems have evolved considerably during the last decade thanks to data mining. Data mining is the automatization of the process of finding interesting patterns in datasets. Methodologies in data mining come from machine learning and statistics. Machine learning is connected to computer science and artificial intelligence and is concerned with finding relations and regularities in data that can be translated into general truths. The aim of machine learning is the reproduction of the data-generating process, allowing analysts to generalize from the observed data to new, unobserved cases (Giudici, 2003).

Early warning systems in finance are vital tools for monitoring and detecting events in financial markets to predict upcoming financial crises. The world financial crisis in 2008 has put an emphasis on the importance of prediction of crises in both academic and industrial senses. It's now more necessary to develop an efficient and predictive model to give early warning signals and to anticipate crises. From a policy perspective, EWS models that help to reliably anticipate financial crises constitute an important tool for policy makers if they are employed carefully and sensibly. Many financial crises over the past few decades had devastating social, economic and political consequences. Developing reliable EWS models therefore can be of substantial value by allowing policy makers to obtain clearer signals about when and how to take pre-emptive action in order to mitigate or even prevent financial turmoil. It should be stressed that EWS models cannot replace the sound judgment of the policy maker to guide policy, but they can play an important complementary role as a neutral and objective measure of vulnerability (Bussiere, 2006).

Forecasting simply means understanding which variables lead or help to predict other variables, when many variables interact in volatile markets. This means looking at the past to see what variables are significant leading indicators of the behavior of other variables. It also means a better understanding of the timing of lead–lag relations among many variables, understanding the statistical significance of these lead–lag

relationships, and learning which variables are the more important ones to watch as signals for further developments in other returns (McNelis, 2005).

In this study, the main motivation is developing reliable EWS. High stock market volatility and excessive stock prices make stock markets more risky. ISE national 100 index data was used to measure the dynamic change of volatility of ISE. A model was developed to predict the occurrence of such crises by using stock market volatility and rising stock prices or rising ISE national 100 index. Five variables as input variables and one variable as an output variable were included in the model. The output variable which is a crisis risk indicator represents the probability of a stock market crash. If the probability is strong, it should be interpreted as a warning signal that a stock market crash is more likely to happen. Adaptive neuro fuzzy inference system (ANFIS) is used in the model to give early warning signals and these signals help forecasting any stock market crash before it happens.

The rest of the study is organized as follows: Section 2 surveys the related works with developing EWS. In Section 3, variables and data mining methods in the model are explained. Respectively, Section 4 reports the results of data mining classifiers and detailed explanation of the ANFIS application. Finally, conclusions are drawn in Section 5.

2. LITERATURE REVIEW

There are various types of financial crises: currency crises, banking crises, sovereign debt crises, private sector debt crises, equity market crises. Most of early warning systems developed so far have tried to predict currency crises, banking crises or both. Previous early warning systems of financial crises have used methods that fall into two broad categories. One approach uses logit or probit models, whereas the other extracts signals from a range of indicators (Bussiere, 2006). The most remarkable papers in this field are written by Frankel and Rose (1996) and Kaminsky et al. (1998).

The advantage of logit or probit model is to represent all the information contained in the variables by giving the probability of the crisis. The disadvantage is that it cannot gauge the precise forecasting ability of each variable though it can give the significance level of each variable. In other words, the ability of the correct signal and false alarm for each variable can not be seen exactly from the model. On the other hand, the signal approach can show the contribution of each variable for the crisis prediction. Besides, it can also offer a summary indicator by calculating the conditional probability given the number of indicators used for signaling (Lin, 2006).

Frankel and Rose (1996) use a panel of annual data for over one hundred developing countries from 1971 through 1992 to characterize currency crashes. They define a currency crash as a large change of the nominal exchange rate that is also a substantial increase in the rate of change of the nominal depreciation. They examine the composition of the debt as well as its level, and a variety of other macroeconomic, external and foreign factors. Factors are significantly related to crash incidence, especially output growth, the rate of change of domestic credit, and foreign interest rates. A low ratio of foreign direct investment to debt is consistently associated with a high likelihood of a crash.

Kaminsky and et al. (1998) examines the empirical evidence on currency crisis and proposes a specific early warning system. This system involves monitoring the evolution of several indicators that tend to exhibit an unusual behavior in periods preceding a crisis. When an indicator exceeds a certain threshold value, this is

interpreted as a warning "signal" that a currency crisis may take place within the next 24 months. The threshold values are calculated so as to strike a balance between the risk of having many false signals and the risk of missing the crisis altogether. Also, since the group of indicators that are issuing signals would be identified, this helps provide information about the source(s) of problems that underlie a crisis.

Peltonen's study (2006) analyzes the predictability of emerging market currency crises by comparing the often used probit model to a multi-layer perceptron artificial neural network (ANN) model. The main result of the study is that both the probit and the ANN model are able to correctly signal crises reasonably well in-sample, and that the ANN model slightly outperforms the probit model. In contrast to the findings in the earlier literature on currency crises, the ability of the models to predict currency crises out-of-sample is found to be weak. Only in the case of the Russian crisis (1998) both models are able to signal its occurrence well in advance. In addition, certain economic factors are found to be related to the emerging market currency crises. These factors are the contagion effect, the prevailing de facto exchange rate regime, the current account and government budget deficits, as well as real gross domestic product (GDP) growth.

Until now, however, a few studies have been done on stock market crises. Kim and et al. (2004) study for modeling EWSs by training classifiers for the distinctive features of economic crises. An economic crisis always makes it possible to consider EWS as a pattern classifier between critical and normal economic situations. To find a better classifier for training EWSs, logistic discrimination (LD) model, decision tree (DT), support vector machine (SVM), neuro-fuzzy model (NF), and artificial neural networks (ANN) are considered among various classifiers. Each of these classifiers has its own strength and weakness, which might work either positively or negatively during training EWS. Kim defines five classifiers to compare in terms of their performances which are done by building EWS based on each classifier for Korean economy which experienced a severe economic crisis in 1997. As a concluding remark of his studies, ANN is suggested as a better classifier and is argued that its major drawback, overfitting might work positively for training EWS.

Levy (2008) also analyzes stock market crashes. He writes in his paper that stock market crashes are traumatic events that affect the lives of millions of people around the globe and have tremendous economic implications. Crashes are not only dramatic, but often completely unexpected. Levy suggests that spontaneous market crashes can be explained by a 'social phase transition' mechanism similar to statistical mechanics phase transitions. The analysis suggests that dramatic crashes are a robust and inevitable property of financial markets. It also implies that market crashes should be preceded by an increase in price volatility, as empirically observed. Thus market crashes are a fundamental and unavoidable part of our world. However, he thinks early warning systems can be developed that may help minimize the damages.

3. MATERIAL AND METHODS

We use ISE national 100 index data to measure the dynamic change of volatility of ISE in 2008. All variables which are used to estimate early warning signals are derived from ISE national 100 index (x_t) in Figure 3.1 by applying formulas numbered 3.2, 3.3, 3.4, and 3.5 that are mentioned below.

We use Kim and et al.'s detailed analysis to select input variables to measure the volatility and their analysis could be summarized as follows: It is expected that the stock market must have shown a sudden increase of volatility as it headed into the crisis. Indeed, five input variables are considered to measure such a sudden volatility increase. Input variables are Istanbul Stock Exchange national 100 index (x_t), daily rise and fall rate (p_t), ten-day moving average of rise and fall rate ($\overline{p}_t$), ten-day moving variance of rise and fall rate (s_t^2) and ratio of moving variance (r_t). The frequency and amplitude of p_t in Figure 3.2 reflects a sudden volatility increase of the stock market due to the upcoming crisis. The other variables $\overline{p}_t$ in Figure 3.3, s_t^2 in Figure 3.4 and r_t in Figure 3.5 add an additional dimension to the volatility analysis. Clearly, s_t^2 measures the amount of variation of p_t, and r_t is found to be very useful to obtain specific dates of sudden volatility increase. In fact, one can easily observe that s_t^2 starts to increase from September and there was an obvious signal or flag at September 16 by r_t (i.e. r_t exceeds 4 on that date) which is an early warning signal. The variable $\overline{p}_t$ is included since it is a variable to smooth out fluctuations over the recent 10 days and then it would help creating a stable indicator. Note that a rather short period of 10 days for moving average is taken into account in order to obtain the visibly clear non-stationary of p_t. These input variables are calculated by formulas presented as follows:

$$[x_t] \tag{3.1}$$

$$[p_t=(x_t-x_{t-1})/x_{t-1}] \tag{3.2}$$

$$[\overline{p}_t = \sum_{i=t-9}^{t} p_i/10] \tag{3.3}$$

$$[s_t^2=(1/10)\sum_{i=t-9}^{t}(pi-\overline{pt})^2] \quad (3.4)$$

$$[r_t=s_t^2/s_{t-1}^2] \quad (3.5)$$

The output variable or the stock market crash risk indicator in Figure 3.6 contains normalized results of the multiplication of Istanbul Stock Exchange national 100 index (x_t) and the ten-day moving variance of rise and fall rate (s_t^2) values:

$$[the\ result= x_t * s_t^2] \quad (3.6)$$

After normalization of the result values, we obtained five output values by applying the following rule:

If the result<0.2 then output=0.2,
Else If the result<0.4 then output=0.4,
Else If the result<0.6 then output=0.6,
Else If the result<0.8 then output=0.8,
Else output=1.

So the output variable contains five values which are 0.2, 0.4, 0.6, 0.8 and 1. The output value 1 means the most risky value in the stock market and if the output value is 1, a stock market crash likely happens in the very following days.

All input and output data is divided into three parts as training data set, testing data set and checking data set. Training data is used for model building, testing data is used for model validation and checking data is used for model evaluation.

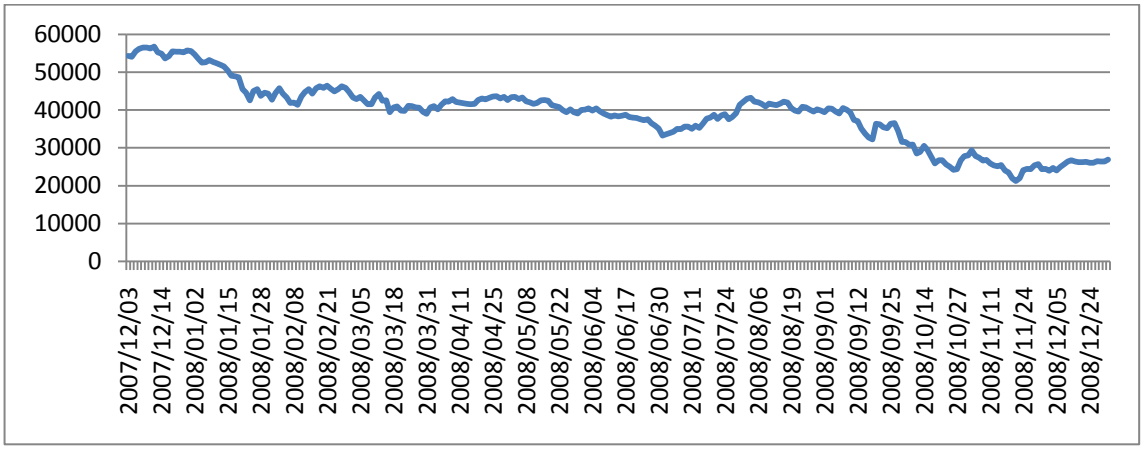

Figure 3.1 : Istanbul Stock Exchange national 100 index of 2008 (x_t)

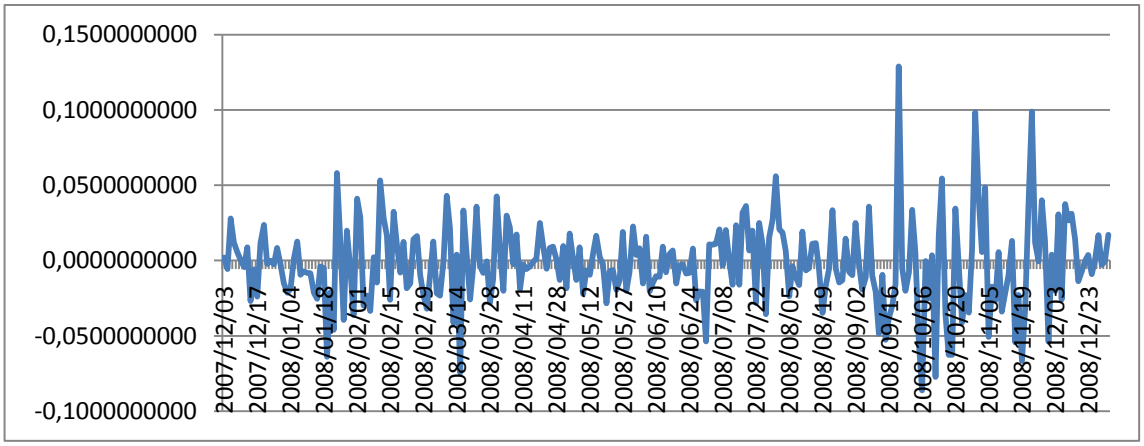

Figure 3.2 : Daily rise and fall rate of 2008 (p_t)

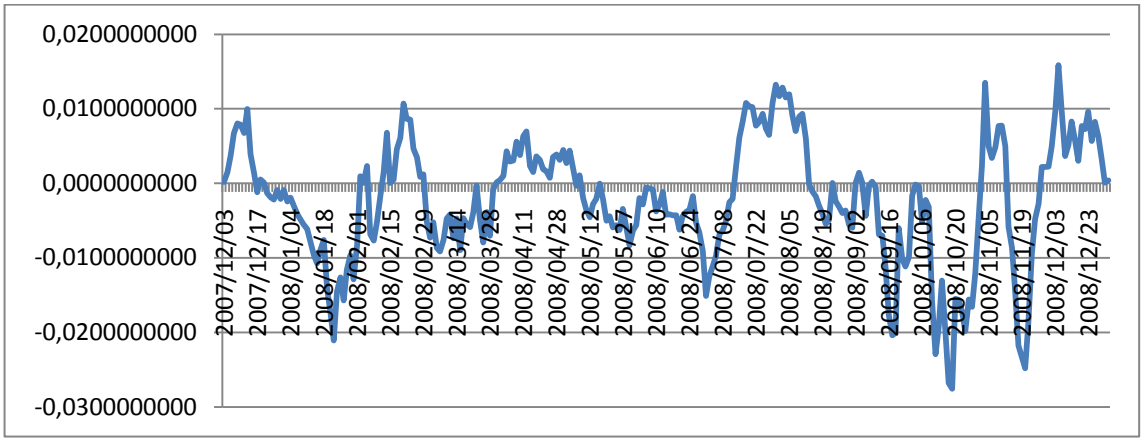

Figure 3.3 : Ten-day moving average of rise and fall rate of 2008 ($\overline{p}_t$)

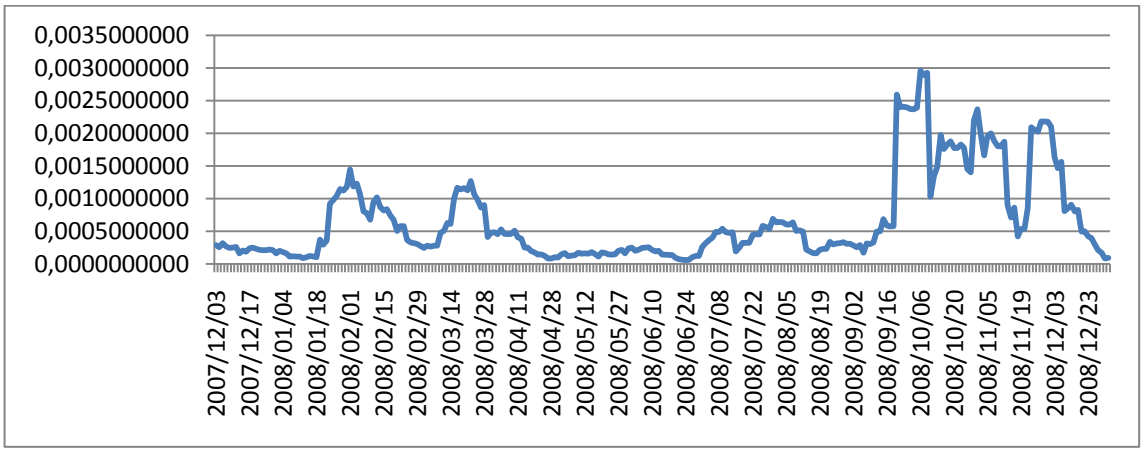

Figure 3.4 : Ten-day moving variance of rise and fall rate of 2008 (s_t^2)

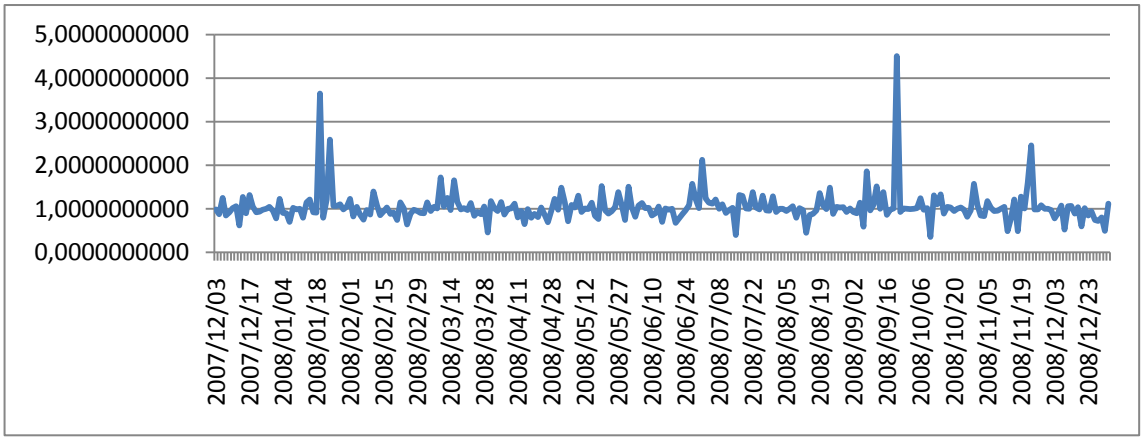

Figure 3.5 : Ratio of moving variance of 2008 (r_t)

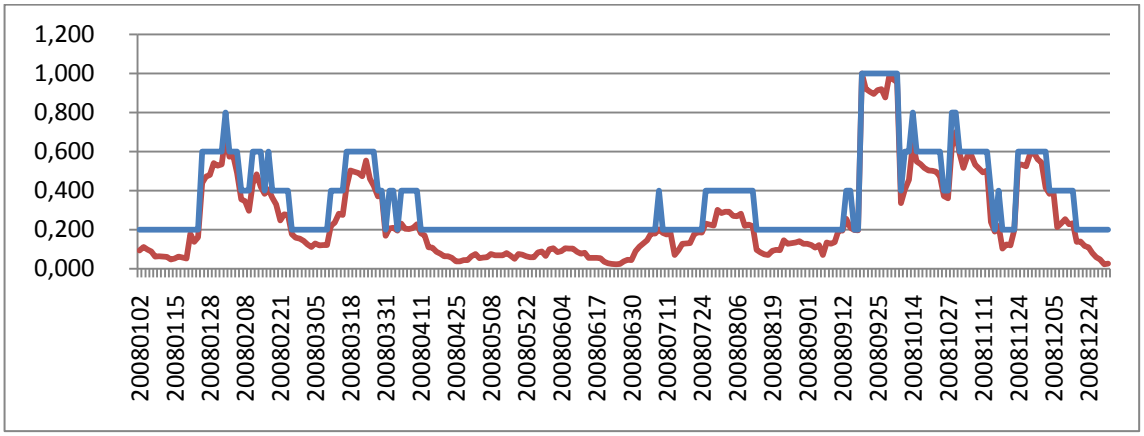

Figure 3.6 : Stock market crash risk indicator of 2008

In this section, definitions of data mining methods used in this study are given. The methods are general methods that are used for forecasting purposes. Following data mining methods are Adaptive neuro fuzzy inference system (ANFIS), Bayes network (BN), Logistic regression (LR), Multilayer perceptron (MLP), Radial basis function (RBF) network, Simple logistic (SL) and Bagging. Matlab and Weka as data mining tools were used. Weka doesn't include ANFIS. ANFIS was performed in Matlab and others were performed in Weka.

The MATLAB high-performance language for technical computing integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include math and computation, algorithm development, data acquisition, modeling, simulation, and prototyping, data analysis, exploration, and visualization, scientific and engineering graphics, application development, including graphical user interface building. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. It allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran. The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation. MATLAB

features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. You can add on toolboxes for signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many other areas (Getting Started Guide, 2008). Fuzzy Logic Toolbox was used in this study. Fuzzy Logic Toolbox software is a collection of functions built on the MATLAB technical computing environment. It provides tools for you to create and edit fuzzy inference systems (FISs) within the framework of MATLAB (Fuzzy Logic Toolbox User's Guide , 2008).

The Weka workbench is a collection of state-of-the-art machine learning algorithms and data preprocessing tools. It is designed so that you can quickly try out existing methods on new datasets in flexible ways. It provides extensive support for the whole process of experimental data mining, including preparing the input data, evaluating learning schemes statistically, and visualizing the input data and the result of learning. As well as a wide variety of learning algorithms, it includes a wide range of preprocessing tools. This diverse and comprehensive toolkit is accessed through a common interface so that its users can compare different methods and identify those that are most appropriate for the problem at hand. Weka was developed at the University of Waikato in New Zealand, and the name stands for Waikato Environment for Knowledge Analysis (WEKA). The system is written in Java and distributed under the terms of the GNU General Public License. It runs on almost any platform and has been tested under Linux, Windows, and Macintosh operating systems—and even on a personal digital assistant. It provides a uniform interface to many different learning algorithms, along with methods for pre- and postprocessing and for evaluating the result of learning schemes on any given dataset (Witten, 2005).

3.1. ADAPTIVE NEURO FUZZY INFERENCE SYSTEM (ANFIS)

We use adaptive neuro fuzzy inference system (ANFIS) to model these input and output data. Fuzzy inference systems (FISs) are also known as fuzzy rule-based systems, fuzzy model, fuzzy expert system, and fuzzy associative memory. This is a major unit of a fuzzy logic system. The decision-making is an important part in the entire system. The FIS formulates suitable rules and based upon the rules the decision is made. This is mainly based on the concepts of the fuzzy set theory, fuzzy IF–THEN rules, and fuzzy reasoning. FIS uses "IF. . . THEN. . . " statements, and the connectors present in the rule statement are "OR" or "AND" to make the necessary decision rules. The basic FIS can take either fuzzy inputs or crisp inputs, but the outputs it produces are almost always fuzzy sets. When the FIS is used as a controller, it is necessary to have a crisp output. Therefore in this case defuzzification method is adopted to best extract a crisp value that best represents a fuzzy set.

The most important two types of fuzzy inference method are Mamdani and Takagi–Sugeno method.

We use Takagi–Sugeno method in this study. The Sugeno fuzzy model was proposed by Takagi, Sugeno, and Kang in an effort to formalize a system approach to generating fuzzy rules from an input–output data set. Sugeno fuzzy model is also known as Sugeno–Takagi model. A typical fuzzy rule in a Sugeno fuzzy model has the format

IF x is A and y is B THEN $z = f(x, y)$,

where A,B are fuzzy sets in the antecedent; $Z = f(x, y)$ is a crisp function in the consequent. Usually $f(x, y)$ is a polynomial in the input variables x and y, but it can be any other functions that can appropriately describe the output of the system within the fuzzy region specified by the antecedent of the rule. When $f(x, y)$ is a first-order polynomial, we have the *first-order* Sugeno fuzzy model. When f is a constant, we then have the *zero-order* Sugeno fuzzy model, which can be viewed either as a special case of the Mamdani FIS where each rule's consequent is specified by a fuzzy singleton, or a special case of Tsukamoto's fuzzy model where each rule's consequent is specified by a membership function of a step function centered at the constant. Moreover, a zero-order

Sugeno fuzzy model is functionally equivalent to a radial basis function (RBF) network under certain minor constraints. The first two parts of the fuzzy inference process, fuzzifying the inputs and applying the fuzzy operator, are exactly the same. The main difference between Mamdani and Sugeno is that the Sugeno output membership functions are either linear or constant. A typical rule in a Sugeno fuzzy model has the form

IF Input 1 = *x* AND Input 2 = *y*, THEN Output is $z = ax + by + c$.

For a zero-order Sugeno model, the output level *z* is a constant ($a = b = 0$). The output level z_i of each rule is weighted by the firing strength *wi* of the rule (Sivanandam, 2007).

Rule 1: If X is A_1 and Y is B_1, then $f_1 = p_1x + q_1y + r_1$
Rule 2: If X is A_2 and Y is B_2, then $f_2 = p_2x + q_2y + r_1$

The fuzzy reasoning mechanism is summarized in Figure 3.7. Weighted averages are used in order to avoid extreme computational complexity in defuzzification processes.

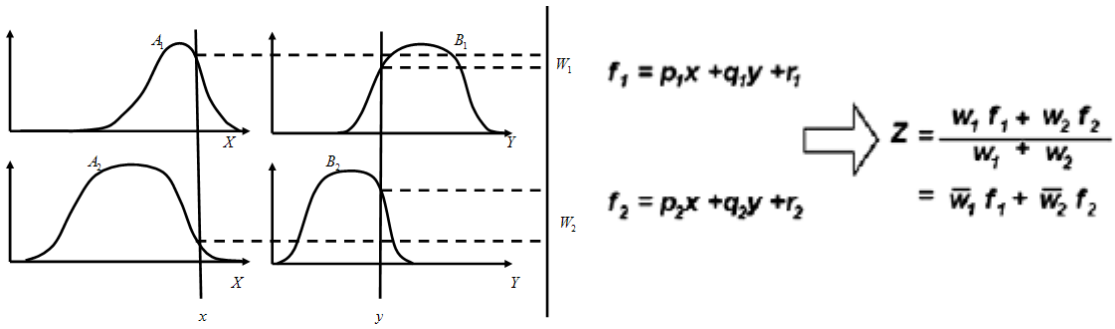

Figure 3.7 : First-order Sugeno fuzzy model

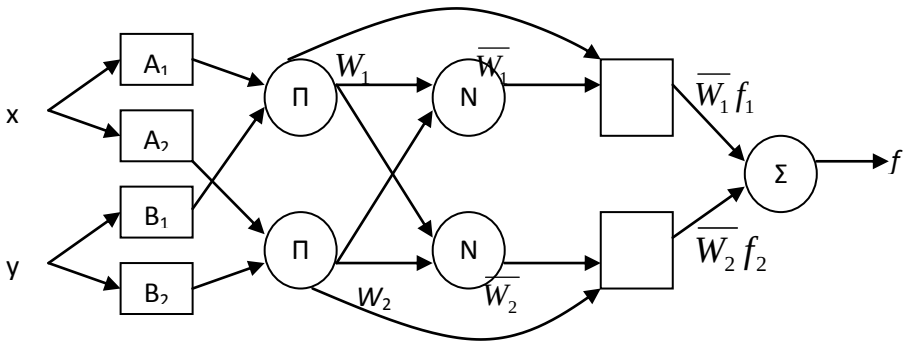

Figure 3.8 : ANFIS Architecture

3.1.1. ANFIS Algorithm

Layer 0: It consists of plain input variable set.

Layer 1: Each node in this layer generates a membership grade of a linguistic label. For instance, the node function of the i-th liode may be a generalized bell membership function:

$$\mu_{A_i}(x) = \frac{1}{1+\left[\left(\frac{x-c_i}{a_i}\right)^2\right]^{b_i}} \tag{3.7}$$

where x is the input to node i; A_i is the linguistic label (small , large, etc.) associated with this node; and $\{a_i, b_i, c_i\}$ is the parameter set that changes the shapes of the membership function. Parameters in this layer are referred to as the premise parameters.

Layer 2: The function is a T-norm operator that performs the firing strength of the rule, e.g., fuzzy conjunctives AND and OR. The simplest implementation just calculates the product of all incoming signals.

$$w_i = \mu A_i(x)\mu B_i(y), i = 1,2. \tag{3.8}$$

Layer 3: Every node in this layer is fixed and determines a normalized firing strength. It calculates the ratio of the jth rule's firing strength to the sum of all rules firing strength.

$$\bar{w}_i = \frac{w_i}{w_1 + w_2}, i = 1,2. \tag{3.9}$$

Layer 4: The nodes in this layer are adaptive and are connected with the input nodes (of layer 0) and the preceding node of layer 3. The result is the weighted output of the rule j.

$$\bar{w}_i f_i = \bar{w}_i(p_i x + q_i y + r_i) \tag{3.10}$$

where $\overline{w_i}$ is the output of layer 3, and $\{p_i, q_i, r_i\}$ is the parameter set. Parameters in this layer are referred to as the consequent parameters.

Layer 5: This layer consists of one single node which computes the overall output as the summation of all incoming signals.

Overall Output = $$\sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \quad (3.11)$$

The constructed adaptive network in Figure 3.8 is functionally equivalent to a fuzzy inference system in Figure 3.7. The basic learning rule of ANFIS is the back-propagation gradient descent which calculates error signals (the derivative of the squared error with respect to each node's output) recursively from the output layer backward to the input nodes. This learning rule is exactly the same as the back-propagation learning rule used in the common feed-forward neural networks (JANG, 1992,1993,1996).

3.2. BAYES NETWORK (BN)

A Bayesian network is a graphical model for probabilistic relationships among a set of variables. Over the last decade, the Bayesian network has become a popular representation for encoding uncertain expert knowledge in expert systems. More recently, researchers have developed methods for learning Bayesian networks from data. The techniques that have been developed are new and still evolving, but they have been shown to be remarkably effective for some data-modeling problems (Heckerman, 1997).

The Bayes classifiers produce probability estimates rather than predictions. For each class value, they estimate the probability that a given instance belongs to that class. Most other types of classifiers can be coerced into yielding this kind of information if necessary. For example, probabilities can be obtained from a decision tree by computing the relative frequency of each class in a leaf and from a decision list by examining the instances that a particular rule covers.

Probability estimates are often more useful than plain predictions. They allow predictions to be ranked, and their expected cost to be minimized. In fact, there is a strong argument for treating classification learning as the task of learning class probability estimates from data. What is being estimated is the conditional probability distribution of the values of the class attribute given the values of the other attributes. The classification model represents this conditional distribution in a concise and easily comprehensible form.

Naïve Bayes classifiers, logistic regression models, decision trees, and so on, are just alternative ways of representing a conditional probability distribution. Of course, they differ in representational power. Naïve Bayes classifiers and logistic regression models can only represent simple distributions, whereas decision trees can represent—or at least approximate— arbitrary distributions. However, decision trees have their drawbacks: they fragment the training set into smaller and smaller pieces, which inevitably yield less reliable probability estimates, and they suffer from the replicated subtree problem. Rule sets go some way toward addressing these shortcomings, but the design of a good rule learner is guided by heuristics with scant theoretical justification.

There is a statistically based alternative: a theoretically well-founded way of representing probability distributions concisely and comprehensibly in a graphical manner. The structures are called Bayesian networks. They are drawn as a network of nodes, one for each attribute, connected by directed edges in such a way that there are no cycles—a directed acyclic graph (DAG). In our explanation of how to interpret Bayesian networks and how to learn them from data, we will make some simplifying assumptions. We assume that all attributes are nominal and that there are no missing values. Some advanced learning algorithms can create new attributes in addition to the ones present in the data—so-called hidden attributes whose values cannot be observed. These can support better models if they represent salient features of the underlying problem, and Bayesian networks provide a good way of using them at prediction time. However, they make both learning and prediction far more complex and time consuming.

The way to construct a learning algorithm for Bayesian networks is to define two components: a function for evaluating a given network based on the data and a method for searching through the space of possible networks. The quality of a given network is measured by the probability of the data given the network. We calculate the probability that the network accords to each instance and multiply these probabilities together over all instances. In practice, this quickly yields numbers too small to be represented properly (called arithmetic underflow), so we use the sum of the logarithms of the probabilities rather than their product. The resulting quantity is the log-likelihood (LL) of the network given the data.

The nodes in the network are predetermined, one for each attribute (including the class). Learning the network structure amounts to searching through the space of possible sets of edges, estimating the conditional probability tables for each set, and computing the log-likelihood of the resulting network based on the data as a measure of the network's quality. Bayesian network learning algorithms differ mainly in the way in which they search through the space of network structures. Some algorithms are introduced below.

There is one caveat. If the log-likelihood is maximized based on the training data, it will always be better to add more edges: the resulting network will simply overfit. Various

methods can be employed to combat this problem. One possibility is to use cross-validation to estimate the goodness of fit. A second is to add a penalty for the complexity of the network based on the number of parameters, that is, the total number of independent estimates in all the probability tables. For each table, the number of independent probabilities is the total number of entries minus the number of entries in the last column, which can be determined from the other columns because all rows must sum to 1. Let K be the number of parameters, the log-likelihood, and N the number of instances in the data. Two popular measures for evaluating the quality of a network are the Akaike Information Criterion (AIC), AIC score = -LL + K, and the following *MDL metric* based on the MDL principle: MDL score **=** -*LL* **+** K/2log N. In both cases the log-likelihood is negated, so the aim is to minimize these scores.

A third possibility is to assign a prior distribution over network structures and find the most likely network by combining its prior probability with the probability accorded to the network by the data. This is the "Bayesian" approach to network scoring. Depending on the prior distribution used, it can take various forms. However, true Bayesians would average over all possible network structures rather than singling out a particular network for prediction. Unfortunately, this generally requires a great deal of computation. A simplified approach is to average over all network structures that are substructures of a given network. It turns out that this can be implemented very efficiently by changing the method for calculating the conditional probability tables so that the resulting probability estimates implicitly contain information from all subnetworks. The details of this approach are rather complex.

The task of searching for a good network structure can be greatly simplified if the right metric is used for scoring. Recall that the probability of a single instance based on a network is the product of all the individual probabilities from the various conditional probability tables. The overall probability of the dataset is the product of these products for all instances. Because terms in a product are interchangeable, the product can be rewritten to group together all factors relating to the same table. The same holds for the log-likelihood, using sums instead of products. This means that the likelihood can be optimized separately for each node of the network. This can be done by adding, or

removing, edges from other nodes to the node that is being optimized—the only constraint is that cycles must not be introduced. The same trick also works if a local scoring metric such as AIC or MDL is used instead of plain log-likelihood because the penalty term splits into several components, one for each node, and each node can be optimized independently.

3.2.1. BAYES Algorithm

One simple and very fast learning algorithm, called K2, starts with a given ordering of the attributes (i.e., nodes). Then it processes each node in turn and greedily considers adding edges from previously processed nodes to the current one. In each step it adds the edge that maximizes the network's score. When there is no further improvement, attention turns to the next node. As an additional mechanism for overfitting avoidance, the number of parents for each node can be restricted to a predefined maximum. Because only edges from previously processed nodes are considered and there is a fixed ordering, this procedure cannot introduce cycles. However, the result depends on the initial ordering, so it makes sense to run the algorithm several times with different random orderings.

The Naïve Bayes classifier is a network with an edge leading from the class attribute to each of the other attributes. When building networks for classification, it sometimes helps to use this network as a starting point for the search. This can be done in K2 by forcing the class variable to be the first one in the ordering and initializing the set of edges appropriately.

Another potentially helpful trick is to ensure that every attribute in the data is in the Markov blanket of the node that represents the class attribute. A node's Markov blanket includes all its parents, children, and children's parents. It can be shown that a node is conditionally independent of all other nodes given values for the nodes in its Markov blanket. Hence, if a node is absent from the class attribute's Markov blanket, its value is completely irrelevant to the classification. Conversely, if K2 finds a network that does not include a relevant attribute in the class node's Markov blanket, it might help to add an edge that rectifies this shortcoming. A simple way of doing this is to add an edge

from the attribute's node to the class node or from the class node to the attribute's node, depending on which option avoids a cycle.

A more sophisticated but slower version of K2 is not to order the nodes but to greedily consider adding or deleting edges between arbitrary pairs of nodes (all the while ensuring acyclicity, of course). A further step is to consider inverting the direction of existing edges as well. As with any greedy algorithm, the resulting network only represents a local maximum of the scoring function: it is always advisable to run such algorithms several times with different random initial configurations. More sophisticated optimization strategies such as simulated annealing, tabu search, or genetic algorithms can also be used.

Another good learning algorithm for Bayesian network classifiers is called tree augmented Naïve (TAN) Bayes. As the name implies, it takes the Naïve Bayes classifier and adds edges to it. The class attribute is the single parent of each node of a Naïve Bayes network: TAN considers adding a second parent to each node. If the class node and all corresponding edges are excluded from consideration, and assuming that there is exactly one node to which a second parent is not added, the resulting classifier has a tree structure rooted at the parentless node—this is where the name comes from. For this restricted type of network there is an efficient algorithm for finding the set of edges that maximizes the network's likelihood based on computing the network's maximum weighted spanning tree. This algorithm is linear in the number of instances and quadratic in the number of attributes.

All the scoring metrics that we have described so far are likelihood based in the sense that they are designed to maximize the joint probability Pr[a1, a2, . . ., an] for each instance. However, in classification, what we really want to maximize is the conditional probability of the class given the values of the other attributes—in other words, the conditional likelihood. Unfortunately, there is no closed-form solution for the maximum conditional-likelihood probability estimates that are needed for the tables in a Bayesian network. On the other hand, computing the conditional likelihood for a given network and dataset is straightforward—after all, this is what logistic regression does. Hence it

has been proposed to use standard maximum likelihood probability estimates in the network, but the conditional likelihood to evaluate a particular network structure.

Another way of using Bayesian networks for classification is to build a separate network for each class value, based on the data pertaining to that class, and combine the predictions using Bayes's rule. The set of networks is called a Bayesian multinet. To obtain a prediction for a particular class value, take the corresponding network's probability and multiply it by the class's prior probability. Do this for each class and normalize the result (Witten, 2005).

In Bayesian statistics, the parameters are considered to be random variables, and the data are considered to be known. The parameters are regarded to come from a distribution of possible values, and Bayesians look to the observed data to provide information on likely parameter values. Let θ represent the parameters of the unknown distribution. Bayesian analysis requires elicitation of a prior distribution for θ, called the prior distribution, p(θ). In the field of data mining, huge data sets are often encountered; therefore, the prior distribution should be dominated by the overwhelming amount of information to be found in the observed data.

Once the data have been observed, prior information about the distribution of θ can be updated by factoring in the information about θ contained in the observed data. This modification leads to the posterior distribution, p(θ|X), where X represents the entire array of data. The posterior distribution of θ given the data is proportional to the product of the likelihood and the prior.

A common estimation method is to choose the posterior mode, the value of θ that maximizes p(θ|X) for an estimate, in which case we call this estimation method the maximum a posteriori (MAP) method. The Bayesian MAP classification is optimal; that is, it achieves the minimum error rate for all possible classifiers. The MAP classifier may be expressed as $\theta_{MAP} = \arg\max_{\theta} p(X|\theta)p(\theta)$. Bayes' theorem is given by

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.12}$$

where A and B are events.

The posterior odds ratio represents a measure of the strength of evidence in favor of a particular classification and is calculated as

$$\frac{p(\theta_c|\mathbf{X})}{p(\overline{\theta}_c|\mathbf{X})} = \frac{p(\mathbf{X}|\theta_c)p(\theta_c)}{p(\mathbf{X}|\overline{\theta}_c)p(\overline{\theta}_c)} \tag{3.13}$$

where θ_c represents a particular classification of the unknown target variable. A value greater than 1 indicates that the posterior distribution favors the positive classification; a value less than 1 represents evidence against the positive classification (e.g., churn = true). The value of the posterior odds ratio may be interpreted as indicating roughly the proportion of evidence provided by the posterior distribution in favor of the positive classification against the negative classification.

In general, the number of probabilities that would need to be calculated to find the MAP classification would be on the order of k^m, where k is the number of classes for the target variable and m is the number of predictor variables. However, we may make the simplifying assumption that the predictor variables are conditionally independent given the target value. Two events A and B are said to be conditionally independent if for a given event C, $p(A \cap B|C) = p(A|C)p(B|C)$.

Thus, the naive Bayes classification is

$$\theta_{NB} = \arg\max_{\theta} \prod_{i=1}^{m} p(X_i = x_i|\theta)p(\theta) \tag{3.14}$$

When the conditional independence assumption is valid, the naive Bayes classification is the same as the MAPclassification. When using naive Bayes classification, far fewer probabilities need to be estimated, just km probabilities rather than k^m for the MAP classifier, in other words, just the number of predictor variables times the number of distinct values of the target variable. Bayesian classification can be extended from

categorical to continuous predictor variables, provided that we know the relevant probability distribution.

Bayesian belief networks (BBNs) are designed to allow joint conditional independencies to be defined among subsets of variables. BBNs, also called Bayesian networks or Bayes nets, take the form of a directed acyclic graph, where directed means that the arcs are traversed in one direction only, and acyclic means that no child node cycles back up to any progenitor. The nodes represent variables, and the arcs represent the (directed) dependence among the variables.

In general, node A is a parent or immediate predecessor of node X, and node X is a descendant of node A, if there exists a directed arc from node A to node X. The intrinsic relationship among the variables in a Bayesian network is as follows: Each variable in a Bayesian network is conditionally independent of its nondescendants in the network, given its parents. The Bayesian network represents the joint probability distribution by providing that (1) a specified set of assumptions regarding the conditional independence of the variables, and (2) the probability tables for each variable, given its direct predecessors. For each variable, information regarding both (1) and (2) is provided (Larose, 2005).

3.3. LOGISTIC REGRESSION (LR)

Linear regression is used to approximate the relationship between a continuous response variable and a set of predictor variables. However, the response variable is often categorical rather than continuous. For such cases, linear regression is not appropriate, but the analyst can turn to an analogous method, logistic regression (LR), which is similar to linear regression in many ways. Logistic regression refers to methods for describing the relationship between a categorical response variable and a set of predictor variables.

Logistic regression assumes that the relationship between the predictor and the response is nonlinear. In linear regression, the response variable is considered to be a random variable $Y = \beta_0 + \beta_1 x + \varepsilon$ with conditional mean $\pi(x) = E(Y|x) = \beta_0 + \beta_1 x$. The conditional mean for logistic regression takes on a different form from that of linear regression. Specifically,

$$\pi(x) = \frac{e^{\beta_0+\beta_1 x}}{1 + e^{\beta_0+\beta_1 x}} \tag{3.15}$$

Curves of this form are called sigmoidal because they are S-shaped and therefore nonlinear. The minimum for π(x) is obtained at

$$\lim_{a\to-\infty} [e^a/(1 + e^a)] = 0 \tag{3.16}$$

and the maximum for π(x) is obtained at

$$\lim_{a\to\infty} [e^a/(1 + e^a)] = 1 \tag{3.17}$$

Thus, π(x) may be interpreted as the probability that the positive outcome is present for records with X = x, and 1 − π(x) may be interpreted as the probability that the positive outcome is absent for such records. The variance of ε is π(x) [1 − π(x)], which is the variance for a binomial distribution, and the response variable in logistic regression Y = π(x) + ε is assumed to follow a binomial distribution with probability of success π(x). The logit transformation is as follows:

$$g(x) = \ln \frac{\pi(x)}{1 - \pi(x)} = \beta_0 + \beta_1 x \tag{3.18}$$

Noclosed-form solution exists for estimating logistic regression coefficients. Thus,we must turn to maximum likelihood estimation, which finds estimates of the parameters for which the likelihood of observing the observed data is maximized.

A saturated model is a model that which contains as many parameters as data points and so predicts the response variable perfectly with no prediction error.We may then look upon the observed values of the response variable to be the values predicted by the saturated model. To compare the values predicted by our fitted model (with fewer parameters than data points) to the values predicted by the saturated model, we use

$$\text{deviance} = -2 \ln \left[\frac{\text{likelihood of the fitted model}}{\text{likelihood of the saturated model}} \right] \tag{3.19}$$

The resulting hypothesis test is called a likelihood ratio test. The deviance represents the error left over in the model, after the predictors have been accounted for. As such, it is analogous to the sum of squares error in linear regression. To determine whether a particular predictor is significant, find the deviance of the model without the predictor, and subtract the deviance of the model with the predictor, thus:

$$\begin{aligned} G &= \textit{deviance}(\text{model without predictor}) - \textit{deviance}(\text{model with predictor}) \\ &= -2 \ln \left[\frac{\text{likelihood without predictor}}{\text{likelihood with predictor}} \right] \end{aligned} \tag{3.20}$$

The test statistic G follows a chi-square distribution with 1 degree of freedom, assuming that the null hypothesis is true that $\beta_1 = 0$.

Odds may be defined as the probability that an event occurs divided by the probability that the event does not occur. The odds ratio (OR) is defined as the odds that the response variable occurred for records with x = 1 divided by the odds that the response variable occurred for records with x = 0. Conveniently, odds ratio = e^{β_1} . The odds ratio

is sometimes used to estimate the relative risk, defined as the probability that the response occurs for x = 1 divided by the probability that the response occurs for x = 0.

The slope coefficient β_1 may be interpreted as the change in the value of the logit for a unit increase in the value of the predictor, $\beta_1 = g(x + 1) - g(x)$. The coefficient b_1 represents the estimated change in the log odds ratio for a unit increase in the predictor. In general, for a constant c, the quantity cb_1 represents the estimated change in the log odds ratio for an increase of c units in the predictor.

Zero cells play havoc with the logistic regression solution, causing instability in the analysis and leading to possibly unreliable results. Rather than omitting the categories with zero cells, we may try to collapse the categories or redefine them somehow, in order to find some records for the zero cells. The logistic regression results should always be validated using either the model diagnostics and goodnessof-fit statistics, or the traditional data mining cross-validation methods (Larose, 2005).

3.4. MULTILAYER PERCEPTRON (MLP)

Neural network proponents used a different approach for nonlinear classification: they connected many simple perceptron-like models in a hierarchical structure. This can represent nonlinear decision boundaries.

Human and animal brains successfully undertake very complex classification tasks—for example, image recognition. The functionality of each individual neuron in a brain is certainly not sufficient to perform these feats. How can they be solved by brain-like structures? The answer lies in the fact that the neurons in the brain are massively interconnected, allowing a problem to be decomposed into subproblems that can be solved at the neuron level. This observation inspired the development of networks of artificial neurons—neural nets.

Consider the simple datasets in Figure 3.9. Figure 3.9(a) shows a two dimensional instance space with four instances that have classes 0 and 1, represented by white and black dots, respectively. No matter how you draw a straight line through this space, you will not be able to find one that separates all the black points from all the white ones. In other words, the problem is not linearly separable, and the simple perceptron algorithm will fail to generate a separating hyperplane (in this two-dimensional instance space a hyperplane is just a straight line). The situation is different in Figure 3.9(b) and Figure 3.9(c): both these problems are linearly separable. The same holds for Figure 3.9(d), which shows two points in a one-dimensional instance space (in the case of one dimension the separating hyperplane degenerates to a separating point).

Figure 3.9(a) represents a logical XOR, where the class is 1 if and only if both attributes have the same value. Figure 3.9(b) represents logical AND, where the class is 1 if and only if both attributes have value 1. Figure 3.9(c) represents OR, where the class is 0 only if both attributes have value 0. Figure 3.9(d) represents NOT, where the class is 0 if and only if the attribute has value 0. Because the last three are linearly separable, a perceptron can represent AND, OR, and NOT. Indeed, perceptrons for the corresponding datasets are shown in Figure 3.9(f) through (h) respectively. However, a simple perceptron cannot represent XOR, because that is not linearly separable. To

build a classifier for this type of problem a single perceptron is not sufficient: we need several of them.

Figure 3.9(e) shows a network with three perceptrons, or units, labeled A, B, and C. The first two are connected to what is sometimes called the input layer of the network, representing the attributes in the data. As in a simple perceptron, the input layer has an additional constant input called the bias. However, the third unit does not have any connections to the input layer. Its input consists of the output of units A and B (either 0 or 1) and another constant bias unit. These three units make up the hidden layer of the multilayer perceptron (MLP). They are called "hidden" because the units have no direct connection to the environment. This layer is what enables the system to represent XOR. Closer inspection of the behavior of the three units reveals that the first one represents OR, the second represents NAND (NOT combined with AND), and the third represents AND. Together they represent the expression (a1 OR a2) AND (a1 NAND a3), which is precisely the definition of XOR.

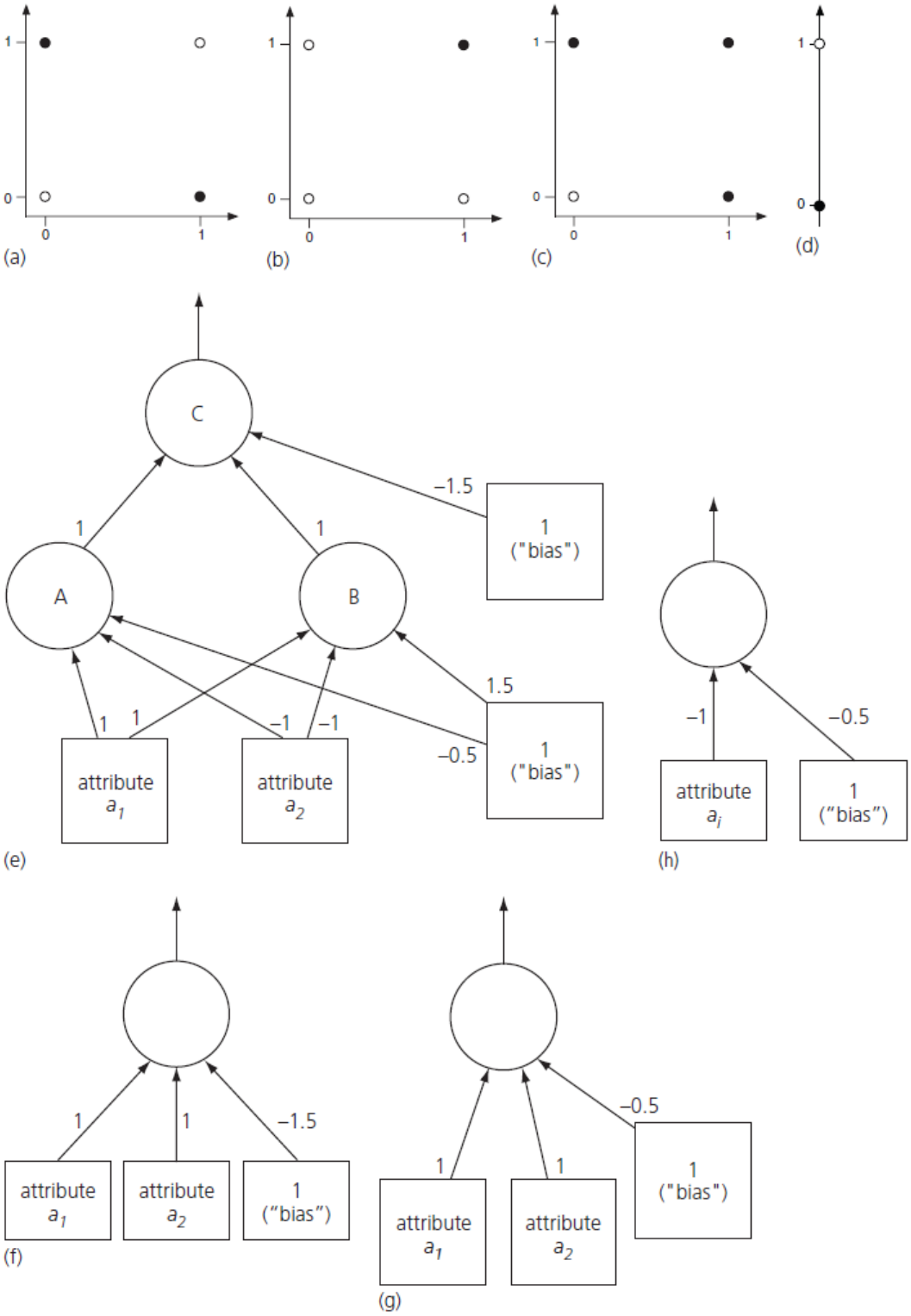

Figure 3.9 : Example datasets and corresponding perceptrons

There are two aspects to how to learn a multilayer perceptron: learning the structure of the network and learning the connection weights. It turns out that there is a relatively simple algorithm for determining the weights given a fixed network structure. This algorithm is called backpropagation. However, although there are many algorithms that attempt to identify network structure, this aspect of the problem is commonly solved through experimentation—perhaps combined with a healthy dose of expert knowledge. Sometimes the network can be separated into distinct modules that represent identifiable subtasks (e.g., recognizing different components of an object in an image recognition problem), which opens up a way of incorporating domain knowledge into the learning process. Often a single hidden layer is all that is necessary, and an appropriate number of units for that layer are determined by maximizing the estimated accuracy.

Backpropagation

Suppose that we have some data and seek a multilayer perceptron that is an accurate predictor for the underlying classification problem. Given a fixed network structure, we must determine appropriate weights for the connections in the network. In the absence of hidden layers, the perceptron learning rule can be used to find suitable values. But suppose there are hidden units. We know what the output unit should predict, and could adjust the weights of the connections leading to that unit based on the perceptron rule. But the correct outputs for the hidden units are unknown, so the rule cannot be applied there.

It turns out that, roughly speaking, the solution is to modify the weights of the connections leading to the hidden units based on the strength of each unit's contribution to the final prediction. There is a standard mathematical optimization algorithm, called gradient descent, which achieves exactly that. Unfortunately, it requires taking derivatives, and the step function that the simple perceptron uses to convert the weighted sum of the inputs into a 0/1 prediction is not differentiable. We need to see whether the step function can be replaced with something else.

Step function: if the input is smaller than zero, it outputs zero; otherwise, it outputs one. We want a function that is similar in shape but differentiable. A commonly used replacement is the sigmoid function, and it is defined by

$$f(x)=\frac{1}{1+e^{-x}} \tag{3.21}$$

Learning a multilayer perceptron is closely related to logistic regression. To apply the gradient descent procedure, the error function—the thing that is to be minimized by adjusting the weights—must also be differentiable. Multilayer perceptrons are usually trained by minimizing the squared error of the network's output, essentially treating it as an estimate of the class probability. (Other loss functions are also applicable. For example, if the likelihood is used instead of the squared error, learning a sigmoid-based perceptron is identical to logistic regression.)

Gradient descent exploits information given by the derivative of the function that is to be minimized—in this case, the error function. As an example, consider a hypothetical error function that happens to be identical to $x^2 + 1$. The X-axis represents a hypothetical parameter that is to be optimized. The derivative of $x^2 + 1$ is simply 2x. The crucial observation is that, based on the derivative, we can figure out the slope of the function at any particular point. If the derivative is negative the function slopes downward to the right; if it is positive, it slopes downward to the left; and the size of the derivative determines how steep the decline is. Gradient descent is an iterative optimization procedure that uses this information to adjust a function's parameters. It takes the value of the derivative, multiplies it by a small constant called the learning rate, and subtracts the result from the current parameter value. This is repeated for the new parameter value, and so on, until a minimum is reached.

The learning rate determines the step size and hence how quickly the search converges. If it is too large and the error function has several minima, the search may overshoot and miss a minimum entirely, or it may oscillate wildly. If it is too small, progress toward the minimum may be slow. Note that gradient descent can only find a local minimum. If the function has several minima— and error functions for multilayer

perceptrons usually have many—it may not find the best one. This is a significant drawback of standard multilayer perceptrons compared with, for example, support vector machines.

To use gradient descent to find the weights of a multilayer perceptron, the derivative of the squared error must be determined with respect to each parameter— that is, each weight in the network.

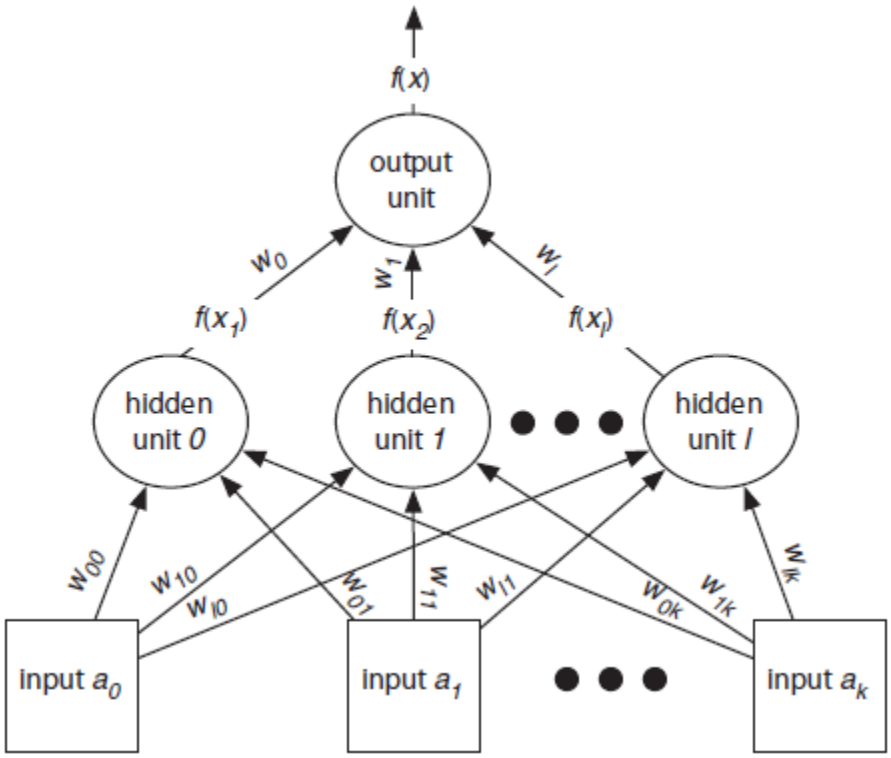

Figure 3.10 : Multilayer perceptron with a hidden layer

Suppose $f(x_i)$ is the output of the ith hidden unit, w_{ij} is the weight of the connection from input j to the ith hidden unit, and w_i is the weight of the ith hidden unit to the output unit. The situation is depicted in Figure 3.10. f(x) is the output of the single unit in the output layer. Putting everything together yields an equation for the derivative of the error function with respect to the weights w_{ij}:

$$\frac{dE}{dw_{ij}} = (y - f(x))f'(x)w_i f'(x_i)a_i. \tag{3.22}$$

As before, we calculate this value for every training instance, add up the changes associated with a particular weight w_{ij}, multiply by the learning rate, and subtract the outcome from the current value of w_{ij}. This derivation applies to a perceptron with one hidden layer. If there are two hidden layers, the same strategy can be applied a second time to update the weights pertaining to the input connections of the first hidden layer, propagating the error from the output unit through the second hidden layer to the first one. Because of this error propagation mechanism, this version of the generic gradient descent strategy is called backpropagation.

We have assumed that weights are only updated after all training instances have been fed through the network and all the corresponding weight changes have been accumulated. This is batch learning, because all the training data is processed together. But exactly the same formulas can be used to update the weights incrementally after each training instance has been processed. This is called stochastic backpropagation because the overall error does not necessarily decrease after every update and there is no guarantee that it will converge to a minimum. In both variants of backpropagation, it is often helpful to standardize the attributes to have zero mean and unit standard deviation. Before learning starts, each weight is initialized to a small, randomly chosen value based on a normal distribution with zero mean.

Like any other learning scheme, multilayer perceptrons trained with backpropagation may suffer from overfitting—especially if the network is much larger than what is actually necessary to represent the structure of the underlying learning problem. Many modifications have been proposed to alleviate this. A very simple one, called early stopping, works like reduced-error pruning in rule learners: a holdout set is used to decide when to stop performing further iterations of the backpropagation algorithm. The error on the holdout set is measured and the algorithm is terminated once the error begins to increase, because that indicates overfitting to the training data. Another method, called weight decay, adds to the error function a penalty term that consists of the squared sum of all weights in the network. This attempts to limit the influence of irrelevant connections on the network's predictions by penalizing large weights that do not contribute a correspondingly large reduction in the error.

Although standard gradient descent is the simplest technique for learning the weights in a multilayer perceptron, it is by no means the most efficient one. In practice, it tends to be rather slow. A trick that often improves performance is to include a momentum term when updating weights: add to the new weight change a small proportion of the update value from the previous iteration. This smoothes the search process by making changes in direction less abrupt. More sophisticated methods use information obtained from the second derivative of the error function as well; they can converge much more quickly. However, even those algorithms can be very slow compared with other methods of classification learning.

A serious disadvantage of multilayer perceptrons that contain hidden units is that they are essentially opaque. There are several techniques that attempt to extract rules from trained neural networks. However, it is unclear whether they offer any advantages over standard rule learners that induce rule sets directly from data—especially considering that this can generally be done much more quickly than learning a multilayer perceptron in the first place.

Although multilayer perceptrons are the most prominent type of neural network, many others have been proposed. Multilayer perceptrons belong to a class of networks called feedforward networks because they do not contain any cycles and the network's output depends only on the current input instance. Recurrent neural networks do have cycles. Computations derived from earlier input are fed back into the network, which gives them a kind of memory (Witten, 2005).

3.5. RADIAL BASIS FUNCTION (RBF) NETWORK

Another popular type of feedforward network is the radial basis function (RBF) network. It has two layers, not counting the input layer, and differs from a multilayer perceptron in the way that the hidden units perform computations. Each hidden unit essentially represents a particular point in input space, and its output, or activation, for a given instance depends on the distance between its point and the instance—which is just another point. Intuitively, the closer these two points, the stronger the activation. This is achieved by using a nonlinear transformation function to convert the distance into a similarity measure. A bell-shaped Gaussian activation function, whose width may be different for each hidden unit, is commonly used for this purpose. The hidden units are called RBFs because the points in instance space for which a given hidden unit produces the same activation form a hypersphere or hyperellipsoid. (In a multilayer perceptron, this is a hyperplane.)

The output layer of an RBF network is the same as that of a multilayer perceptron: it takes a linear combination of the outputs of the hidden units and— in classification problems—pipes it through the sigmoid function.

The parameters that such a network learns are (a) the centers and widths of the RBFs and (b) the weights used to form the linear combination of the outputs obtained from the hidden layer. A significant advantage over multilayer perceptrons is that the first set of parameters can be determined independently of the second set and still produce accurate classifiers.

One way to determine the first set of parameters is to use clustering, without looking at the class labels of the training instances at all. The simple k-means clustering algorithm can be applied, clustering each class independently to obtain k basis functions for each class. Intuitively, the resulting RBFs represent prototype instances. Then the second set of parameters can be learned, keeping the first parameters fixed. This involves learning a linear model (e.g., linear or logistic regression). If there are far fewer hidden units than training instances, this can be done very quickly.

A disadvantage of RBF networks is that they give every attribute the same weight because all are treated equally in the distance computation. Hence they cannot deal effectively with irrelevant attributes—in contrast to multilayer perceptrons. Support vector machines share the same problem. In fact, support vector machines with Gaussian kernels (i.e., "RBF kernels") are a particular type of RBF network, in which one basis function is centered on every training instance, and the outputs are combined linearly by computing the maximum margin hyperplane. This has the effect that only some RBFs have a nonzero weight—the ones that represent the support vectors (Witten, 2005).

3.6. SIMPLE LOGISTIC (SL)

The Simple logistic (SL) model is widely used for technology forecasting. Many new forecasting models were proposed based on the simple logistic model and include innovations such as the Bass diffusion model and extended logistic model. The most important characteristic of simple logistic model is that it is symmetric about the point of inflection. This feature indicates that the process which will happen after the point of inflection is the mirror image of the process that happened before the point. The model for the simple logistic curve is controlled by three coefficients, a, b, and L is expressed as

$$y_t = \frac{L}{1 + ae^{-bt}} \tag{3.23}$$

where y_t is the value of interest, L is the maximum value of y_t, a describes the location of the curve, and b controls the shape of the curve. To estimate the parameters for a and b, the equation of the simple logistic model is transformed into a linear function using natural logarithms. The linear model is expressed as

$$Y_t = \ln(y_t/L - y_t) = -\ln(a) + bt \tag{3.24}$$

where the parameter a and b are then estimated using a simple linear regression (Trappey, 2008).

When the outcome, or class, is numeric, and all the attributes are numeric, linear regression is a natural technique to consider. This is a staple method in statistics. The idea is to express the class as a linear combination of the attributes, with predetermined weights:

$$x = w_0 + w_1a_1 + w_2a_2 + ... + w_ka_k \tag{3.25}$$

where x is the class; a_1, a_2, . . ., a_k are the attribute values; and w_0, w_1, . . ., w_k are weights.

The weights are calculated from the training data. Here the notation gets a little heavy, because we need a way of expressing the attribute values for each training instance. The first instance will have a class, say $x^{(1)}$, and attribute values $a_1^{(1)}$, $a_2^{(1)}$, . . ., $a_k^{(1)}$, where the superscript denotes that it is the first example. Moreover, it is notationally convenient to assume an extra attribute a_0 whose value is always 1.

The predicted value for the first instance's class can be written as

$$w_0 a_0^{(1)} + w_1 a_1^{(1)} + w_2 a_2^{(1)} + \ldots + w_k a_k^{(1)} = \sum_{j=0}^{k} w_j a_j^{(1)} \tag{3.26}$$

This is the predicted, not the actual, value for the first instance's class. Of interest is the difference between the predicted and the actual values. The method of linear regression is to choose the coefficients w_j—there are k + 1 of them—to minimize the sum of the squares of these differences over all the training instances. Suppose there are n training instances; denote the i_{th} one with a superscript (i). Then the sum of the squares of the differences is

$$\sum_{i=1}^{n} \left(x^{(i)} - \sum_{j=0}^{k} w_j a_j^{(i)} \right)^2 \tag{3.27}$$

where the expression inside the parentheses is the difference between the i_{th} instance's actual class and its predicted class. This sum of squares is what we have to minimize by choosing the coefficients appropriately.

Linear regression is an excellent, simple method for numeric prediction, and it has been widely used in statistical applications for decades. Of course, linear models suffer from the disadvantage of, well, linearity. If the data exhibits a nonlinear dependency, the best-fitting straight line will be found, where "best" is interpreted as the least mean-squared difference. This line may not fit very well. However, linear models serve well as building blocks for more complex learning methods.

3.7. BAGGING

Combining the decisions of different models means amalgamating the various outputs into a single prediction. The simplest way to do this in the case of classification is to take a vote (perhaps a weighted vote); in the case of numeric prediction, it is to calculate the average (perhaps a weighted average). Bagging and boosting both adopt this approach, but they derive the individual models in different ways. In bagging, the models receive equal weight, whereas in boosting, weighting is used to give more influence to the more successful ones—just as an executive might place different values on the advice of different experts depending on how experienced they are.

To introduce bagging, suppose that several training datasets of the same size are chosen at random from the problem domain. Imagine using a particular machine learning technique to build a decision tree for each dataset. You might expect these trees to be practically identical and to make the same prediction for each new test instance. Surprisingly, this assumption is usually quite wrong, particularly if the training datasets are fairly small. This is a rather disturbing fact and seems to cast a shadow over the whole enterprise! The reason for it is that decision tree induction is an unstable process: slight changes to the training data may easily result in a different attribute being chosen at a particular node, with significant ramifications for the structure of the subtree beneath that node. This automatically implies that there are test instances for which some of the decision trees produce correct predictions and others do not.

Returning to the preceding experts analogy, consider the experts to be the individual decision trees.We can combine the trees by having them vote on each test instance. If one class receives more votes than any other, it is taken as the correct one. Generally, the more the merrier: predictions made by voting become more reliable as more votes are taken into account. Decisions rarely deteriorate if new training sets are discovered, trees are built for them, and their predictions participate in the vote as well. In particular, the combined classifier will seldom be less accurate than a decision tree constructed from just one of the datasets. (Improvement is not guaranteed, however. It can be shown theoretically that pathological situations exist in which the combined decisions are worse.)

The effect of combining multiple hypotheses can be viewed through a theoretical device known as the bias–variance decomposition. Suppose that we could have an infinite number of independent training sets of the same size and use them to make an infinite number of classifiers. A test instance is processed by all classifiers, and a single answer is determined by majority vote. In this idealized situation, errors will still occur because no learning scheme is perfect: the error rate will depend on how well the machine learning method matches the problem at hand, and there is also the effect of noise in the data, which cannot possibly be learned. Suppose the expected error rate were evaluated by averaging the error of the combined classifier over an infinite number of independently chosen test examples. The error rate for a particular learning algorithm is called its bias for the learning problem and measures how well the learning method matches the problem. This technical definition is a way of quantifying the vaguer notion of bias: it measures the "persistent" error of a learning algorithm that can't be eliminated even by taking an infinite number of training sets into account. Of course, it cannot be calculated exactly in practical situations; it can only be approximated.

A second source of error in a learned model, in a practical situation, stems from the particular training set used, which is inevitably finite and therefore not fully representative of the actual population of instances. The expected value of this component of the error, over all possible training sets of the given size and all possible test sets, is called the variance of the learning method for that problem. The total expected error of a classifier is made up of the sum of bias and variance: this is the bias–variance decomposition. Combining multiple classifiers decreases the expected error by reducing the variance component. The more classifiers that are included, the greater the reduction in variance.

Of course, a difficulty arises when putting this voting method into practice: usually there's only one training set, and obtaining more data is either impossible or expensive. Bagging attempts to neutralize the instability of learning methods by simulating the process described previously using a given training set. Instead of sampling a fresh, independent training dataset each time, the original training data is altered by deleting some instances and replicating others. Instances are randomly sampled, with replacement, from the original dataset to create a new one of the same size. This

sampling procedure inevitably replicates some of the instances and deletes others. The term bagging stands for bootstrap aggregating. Bagging applies the learning scheme—for example, a decision tree inducer—to each one of these artificially derived datasets, and the classifiers generated from them vote for the class to be predicted.

The algorithm is summarized as follows (Han, 2006).

Bagging Algorithm: The bagging algorithm—create an ensemble of models (classifiers or predictors) for a learning scheme where each model gives an equally-weighted prediction.

Input:

D, a set of d training tuples;
k, the number of models in the ensemble;
a learning scheme (e.g., decision tree algorithm, backpropagation, etc.)

Output: A composite model, M*.

Method:

(1) for i = 1 to k do // create k models:
(2) create bootstrap sample, D_i, by sampling D with replacement;
(3) use D_i to derive a model, M_i;
(4) endfor

To use the composite model on a tuple, X:

(1) if classification then
(2) let each of the k models classify X and return the majority vote;
(3) if prediction then
(4) let each of the k models predict a value for X and return the average predicted value;

Instead of obtaining independent datasets from the domain, bagging just resamples the original training data. The datasets generated by resampling are different from one another but are certainly not independent because they are all based on one dataset. However, it turns out that bagging produces a combined model that often performs

significantly better than the single model built from the original training data, and is never substantially worse.

Bagging can also be applied to learning methods for numeric prediction— for example, model trees. The only difference is that, instead of voting on the outcome, the individual predictions, being real numbers, are averaged. The bias–variance decomposition can be applied to numeric prediction as well by decomposing the expected value of the mean-squared error of the predictions on fresh data. Bias is defined as the mean-squared error expected when averaging over models built from all possible training datasets of the same size, and variance is the component of the expected error of a single model that is due to the particular training data it was built from. It can be shown theoretically that averaging over multiple models built from independent training sets always reduces the expected value of the mean-squared error.

Bagging helps most if the underlying learning method is unstable in that small changes in the input data can lead to quite different classifiers. Indeed it can help to increase the diversity in the ensemble of classifiers by making the learning method as unstable as possible. For example, when bagging decision trees, which are already unstable, better performance is often achieved by switching pruning off, which makes them even more unstable. Another improvement can be obtained by changing the way that predictions are combined for classification. As originally formulated, bagging uses voting. But when the models can output probability estimates and not just plain classifications, it makes intuitive sense to average these probabilities instead. Not only does this often improve classification slightly, but the bagged classifier also generates probability estimates—ones that are often more accurate than those produced by the individual models. Implementations of bagging commonly use this method of combining predictions.

Bagging is a prime candidate for cost-sensitive classification because it produces very accurate probability estimates from decision trees and other powerful, yet unstable, classifiers. However, a disadvantage is that bagged classifiers are hard to analyze.

A method called MetaCost combines the predictive benefits of bagging with a comprehensible model for cost-sensitive prediction. It builds an ensemble classifier using bagging and uses it to relabel the training data by giving every training instance the prediction that minimizes the expected cost, based on the probability estimates obtained from bagging. MetaCost then discards the original class labels and learns a single new classifier—for example, a single pruned decision tree—from the relabeled data. This new model automatically takes costs into account because they have been built into the class labels! The result is a single cost-sensitive classifier that can be analyzed to see how predictions are made.

3.8. COMPARING DATA MINING METHODS

We often need to compare two different learning methods on the same problem to see which the better one to use is. It seems simple: estimate the error using cross-validation (or any other suitable estimation procedure), perhaps repeated several times, and choose the scheme whose estimate is smaller. This is quite sufficient in many practical applications: if one method has a lower estimated error than another on a particular dataset, the best we can do is to use the former method's model. However, it may be that the difference is simply caused by estimation error, and in some circumstances it is important to determine whether one scheme is really better than another on a particular problem. This is a standard challenge for machine learning researchers. If a new learning algorithm is proposed, its proponents must show that it improves on the state of the art for the problem at hand and demonstrate that the observed improvement is not just a chance effect in the estimation process.

This is a job for a statistical test that gives confidence bounds, the kind we met previously when trying to predict true performance from a given test-set error rate. If there were unlimited data, we could use a large amount for training and evaluate performance on a large independent test set, obtaining confidence bounds just as before. However, if the difference turns out to be significant we must ensure that this is not just because of the particular dataset we happened to base the experiment on. What we want to determine is whether one scheme is better or worse than another on average, across all possible training and test datasets that can be drawn from the domain. Because the amount of training data naturally affects performance, all datasets should be the same size: indeed, the experiment might be repeated with different sizes to obtain a learning curve (Witten, 2005).

Using training data to derive a classifier or predictor and then to estimate the accuracy of the resulting learned model can result in misleading overoptimistic estimates due to overspecialization of the learning algorithm to the data. Instead, accuracy is better measured on a test set consisting of class-labeled tuples that were not used to train the model. The accuracy of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. In the pattern recognition literature, this is

also referred to as the overall recognition rate of the classifier, that is, it reflects how well the classifier recognizes tuples of the various classes.

We can also speak of the error rate or misclassification rate of a classifier,M, which is simply 1-Acc(M), where Acc(M) is the accuracy of M. If we were to use the training set to estimate the error rate of a model, this quantity is known as the resubstitution error. This error estimate is optimistic of the true error rate (and similarly, the corresponding accuracy estimate is optimistic) because the model is not tested on any samples that it has not already seen.

The confusion matrix (CM) is a useful tool for analyzing how well your classifier can recognize tuples of different classes. A confusion matrix for two classes is shown in Figure 3.11. Given m classes, a confusion matrix is a table of at least size m by m. An entry, $CM_{i,j}$ in the first m rows and m columns indicates the number of tuples of class i that were labeled by the classifier as class j. For a classifier to have good accuracy, ideally most of the tuples would be represented along the diagonal of the confusion matrix, from entry $CM_{1,1}$ to entry $CM_{m,m}$, with the rest of the entries being close to zero. The table may have additional rows or columns to provide totals or recognition rates per class.

		Predicted class	
		C_1	C_2
Actual class	C_1	true positives	false negatives
	C_2	false positives	true negatives

Figure 3.11 : A confusion matrix for positive and negative tuples

Given two classes, we can talk in terms of positive tuples versus negative tuples. True positives (TP) refer to the positive tuples that were correctly labeled by the classifier, while true negatives (TN) are the negative tuples that were correctly labeled by the classifier. False positives (FP) are the negative tuples that were incorrectly labeled. Similarly, false negatives (FN) are the positive tuples that were incorrectly labeled. These terms are useful when analyzing a classifier's ability.

Sensitivity is also referred to as the true positive (recognition) rate (that is, the proportion of positive tuples that are correctly identified), while specificity is the true negative rate (TNR) (that is, the proportion of negative tuples that are correctly identified). The counts in a confusion matrix can also be expressed in terms of percentages. The true positive rate (TPR) or sensitivity is defined as the fraction of positive examples predicted correctly by the model:

$$sensitivity\ (TPR) = TP/(TP + FN) \tag{3.28}$$

Similarly, the true negative rate (TNR) or specificity is defined as the fraction of negative examples predicted correctly by the model.

$$specificity\ (TNR) = TN/(TN + FP) \tag{3.29}$$

Accuracy is a function of sensitivity and specificity:

$$accuracy = sensitivity\frac{pos}{(pos+neg)} + specificity\frac{neg}{(pos+neg)} \tag{3.30}$$

Precision determines the fraction of records that actually turns out to be positive in the group which the classifier has declared as a positive class. The higher the precision is, the lower the number of false positive errors committed by the classifier.

$$precision = TP/(TP + FP\ ) \tag{3.31}$$

Recall is defined:

$$recall = TP/(TP + FN) \tag{3.32}$$

F-score, which is defined as the harmonic mean of recall and precision:

$$F\_score = \frac{recall \times precision}{(recall + precision)/2} \tag{3.33}$$

Correctness is the percentage of correctly classified instances. RMSE denotes the root mean square error (RMSE) for the given dataset and method of classification. Correctness and RMSE values show important variety. Lower RMSE systems tend to make less incorrect classifications than the others and it indicates reliability in further testing of data.

In classification problems, it is commonly assumed that all tuples are uniquely classifiable, that is, that each training tuple can belong to only one class. Yet, owing to the wide diversity of data in large databases, it is not always reasonable to assume that all tuples are uniquely classifiable. Rather, it is more probable to assume that each tuple may belong to more than one class. How then can the accuracy of classifiers on large databases be measured? The accuracy measure is not appropriate, because it does not take into account the possibility of tuples belonging to more than one class.

Rather than returning a class label, it is useful to return a probability class distribution. Accuracy measures may then use a second guess heuristic, whereby a class prediction is judged as correct if it agrees with the first or second most probable class. Although this does take into consideration, to some degree, the nonunique classification of tuples, it is not a complete solution.

ROC curves are a useful visual tool for comparing two classification models. The name ROC stands for Receiver Operating Characteristic (ROC). ROC curves come from signal detection theory that was developed during World War II for the analysis of radar images. An ROC curve shows the trade-off between the true positive rate or sensitivity (proportion of positive tuples that are correctly identified) and the false-positive rate (proportion of negative tuples that are incorrectly identified as positive) for a given model. That is, given a two-class problem, it allows us to visualize the trade-off between the rate at which the model can accurately recognize ‘yes’ cases versus the rate at which it mistakenly identifies ‘no’ cases as ‘yes’ for different “portions” of the test set. Any increase in the true positive rate occurs at the cost of an increase in the false-positive rate. The area under the ROC curve is a measure of the accuracy of the model.

In order to plot an ROC curve for a given classification model, M, the model must be able to return a probability or ranking for the predicted class of each test tuple. That is, we need to rank the test tuples in decreasing order, where the one the classifier thinks is most likely to belong to the positive or 'yes' class appears at the top of the list. Naive Bayesian and backpropagation classifiers are appropriate, whereas others, such as decision tree classifiers, can easily be modified so as to return a class probability distribution for each prediction. The vertical axis of an ROC curve represents the true positive rate. The horizontal axis represents the false-positive rate. An ROC curve for M is plotted as follows. Starting at the bottom left-hand corner (where the true positive rate and false-positive rate are both 0), we check the actual class label of the tuple at the top of the list. If we have a true positive (that is, a positive tuple that was correctly classified), then on the ROC curve, we move up and plot a point. If, instead, the tuple really belongs to the 'no' class, we have a false positive. On the ROC curve, we move right and plot a point. This process is repeated for each of the test tuples, each time moving up on the curve for a true positive or toward the right for a false positive.

Figure 3.12 shows the ROC curves of two classification models. The plot also shows a diagonal line where for every true positive of such a model, we are just as likely to encounter a false positive. Thus, the closer the ROC curve of a model is to the diagonal line, the less accurate the model. If the model is really good, initially we are more likely to encounter true positives as we move down the ranked list. Thus, the curve would move steeply up from zero. Later, as we start to encounter fewer and fewer true positives, and more and more false positives, the curve cases off and becomes more horizontal.

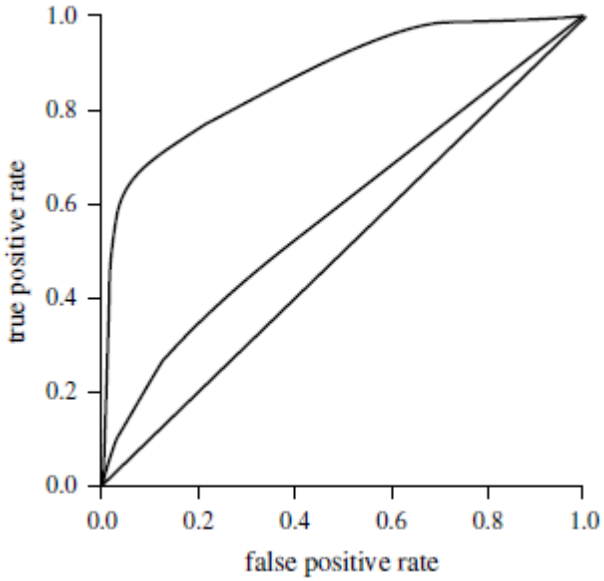

Figure 3.12 : The ROC curves of two classification models

To assess the accuracy of a model, we can measure the area under the curve (AUC). Several software packages are able to perform such calculation. The closer AUC is to 0.5, the less accurate the corresponding model is. A model with perfect accuracy will have an area of 1.0 (Han, 2006).

4. FINDINGS

This study compares various data mining classifiers to obtain the best practical solution for the financial early warning system and offers ANFIS as a means to efficient stock market crashes forecasting.

We should again underline that it is possible to foresee whether the market is close to a stock market crash by monitoring the market's volatility. Market volatility is expected to peak just before the market crash. The input variables are taken into account in this study mainly because they can effectively measure the volatility change of stock market. This effect may be pronounced in advance, as illustrated by numerical simulation, and in confirmation with the empirical findings. The value of the stock market crash risk indicator closely shows the possibility of an upcoming crisis. The possibility of a crisis is as high as the value.

Rule based models and decision tree derivatives have high level of precision, however they demonstrate poor robustness when the dataset is changed. In order to provide adaptability of the classification technique, neural network based alteration of fuzzy inference system parameters is necessary. The results prove that, ANFIS method combines both precision of fuzzy based classification system and adaptability (back propagation) feature of neural networks in classification of data. Receiver operating characteristics analysis conveys information about performance from all possible combinations and of misclassification costs and class distributions. The Receiver Operating Characteristic (ROC) curve, which is obtained by altering threshold level, is typically used to visualize the performance and robustness of the method. The ROC curve indicates how the prediction rate changes as the thresholds are varied to generate more or fewer false alarms. The ROC curve is a plot of prediction accuracy against the false positive probability that tradeoffs prediction accuracy against the analyst workload. The ROC curves for selected methods are illustrated on Fig. 4.1.

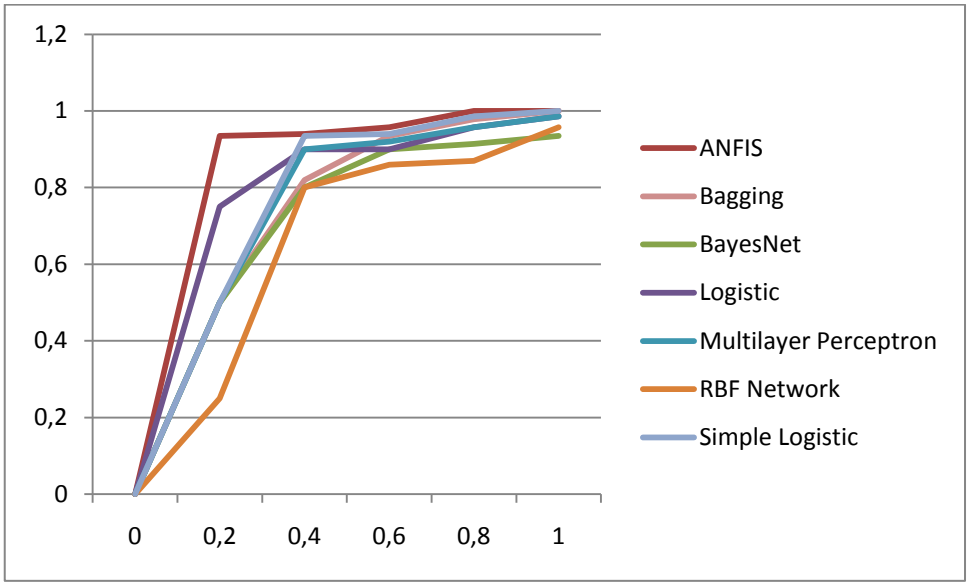

Figure 4.1 : The ROC curves of selected methods

In this study, various data mining classifiers were used. The empirical results in Table 4.1 show that the proposed ANFIS model is the most successful one.

Table 4.1 : Test results for the selected methods

Method	Sensitivity	Precision	Recall	F-Measure	ROC Area
ANFIS	0.9664	0.9492	0.9664	0.9574	0.9886
Bagging	0.7466	0.7156	0.7466	0.7296	0.958
BayesNet	0.8098	0.9084	0.8098	0.8412	0.967
Logistic	0.8986	0.9166	0.8986	0.9068	0.988
MultilayerPerceptron	0.8526	0.8696	0.8526	0.8582	0.996
RBFNetwork	0.7474	0.8038	0.7474	0.7664	0.8442
SimpleLogistic	0.8722	0.902	0.8722	0.8846	0.994

The ANFIS approach uses Gaussian functions for fuzzy sets and linear functions for the rule outputs. The initial parameters of the network are the mean and standard deviation of the membership functions (antecedent parameters) and the coefficients of the output linear functions (consequent parameters). The ANFIS learning algorithm is then used to obtain these parameters. This learning algorithm is a hybrid algorithm consisting of the gradient descent and the least-squares estimate. Using this hybrid algorithm, the rule parameters are recursively updated until an acceptable error is reached. Iterations have two steps, one forward and one backward. In the forward pass, the antecedent

parameters are fixed, and the consequent parameters are obtained using the linear least-squares estimate. In the backward pass, the consequent parameters are fixed, and the output error is back-propagated through this network, and the antecedent parameters are accordingly updated using the gradient descent method.

In the designing of ANFIS model in Figure 4.2, the number of membership functions, the number of fuzzy rules, and the number of training epochs are important factors to be considered. If they are not selected appropriately, the system will overfit the data or will not be able to fit the data. Adjusting mechanism works using a hybrid algorithm combining the least squares method and the gradient descent method with a mean square error method. ANFIS creates membership functions for each variable input.

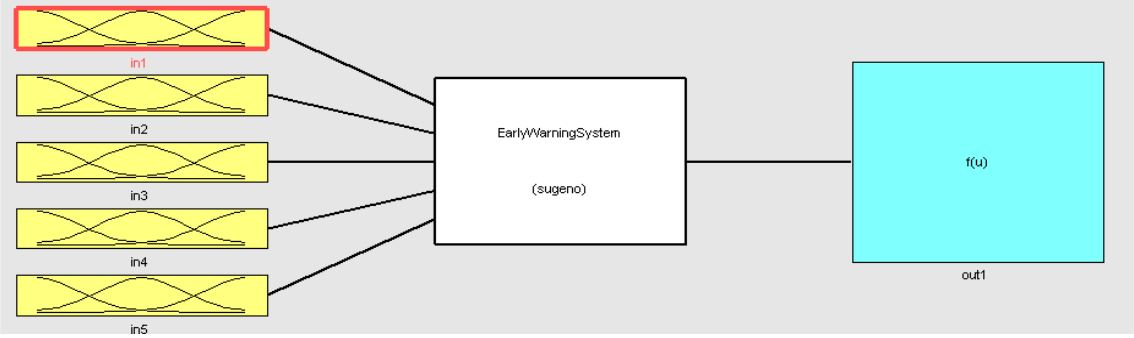

Figure 4.2 : FIS Model

The aim of the training process in Figure 4.3 is to minimize the training error between the ANFIS output and the actual objective. This allows a fuzzy system to train its features from the data it observes, and implements these features in the system rules.

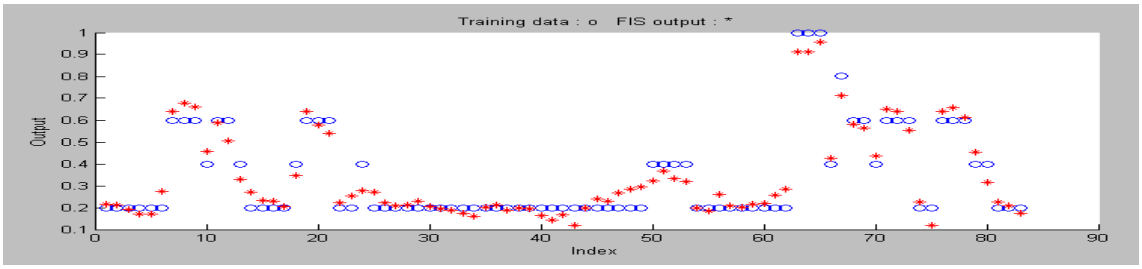

Figure 4.3 : Training plot

As seen on Figure 4.4 and Figure 4.5, the results indicate that ANFIS has a pretty good means to model an EWS. 78 of 83 test samples were correctly classified. In Figure 4.4, the vertical axis denotes the test output, whereas the horizontal axis shows the index of the testing data instances.

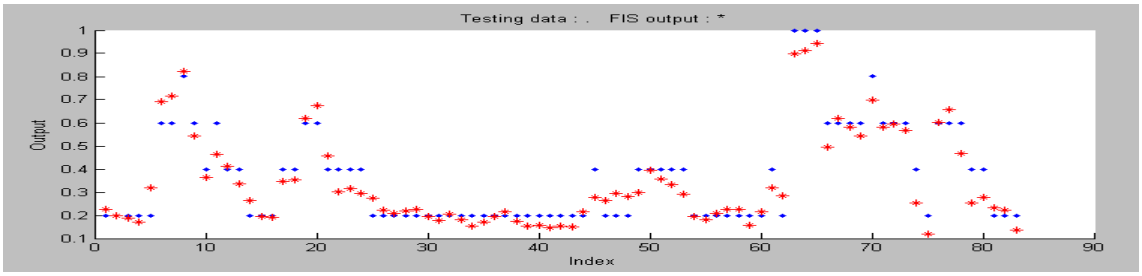

Figure 4.4 : Testing plot

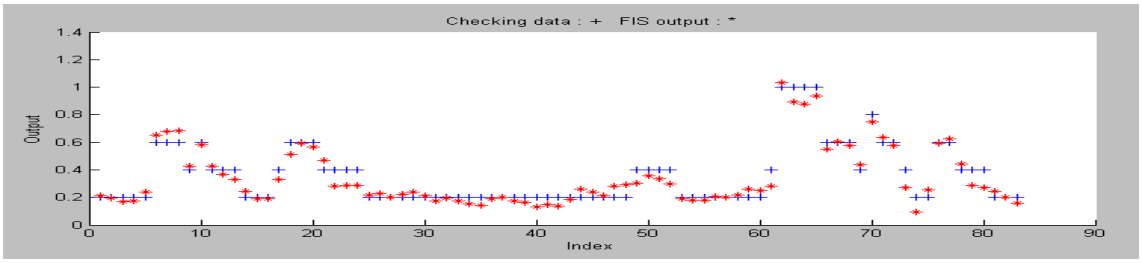

Figure 4.5 : Checking plot

The plots in Figure 4.3, Figure 4.4 and Figure 4.5 show that the proposed neuro fuzzy model is successful. Average training error is 0.0498, average testing error is 0.0653 and average checking error is 0.0587. The result of experiment shows that the accuracy rate of the neuro fuzzy model is approximately 95%.

We prefer Sugeno-type for computational efficiency. As seen on Figure 4.6, Takagi and Sugeno's fuzzy if-then rules are used in the model. The output of each rule is a linear combination of input variables and a constant term. The final output is the weighted average of each rule's output. The basic learning rule of the proposed network is based on the gradient descent and the chain rule. Takagi and Sugeno's fuzzy is a fuzzy system with crisp functions in consequent, which perceived proper for multifaceted applications. Due to crisp consequent functions, ANFIS method requires a rather uncomplicated form of scaling implicitly. The ANFIS has the advantage of good

applicability as it can be interpreted as local linearization modeling and that form of state estimation is straightforwardly applicable to different systems.

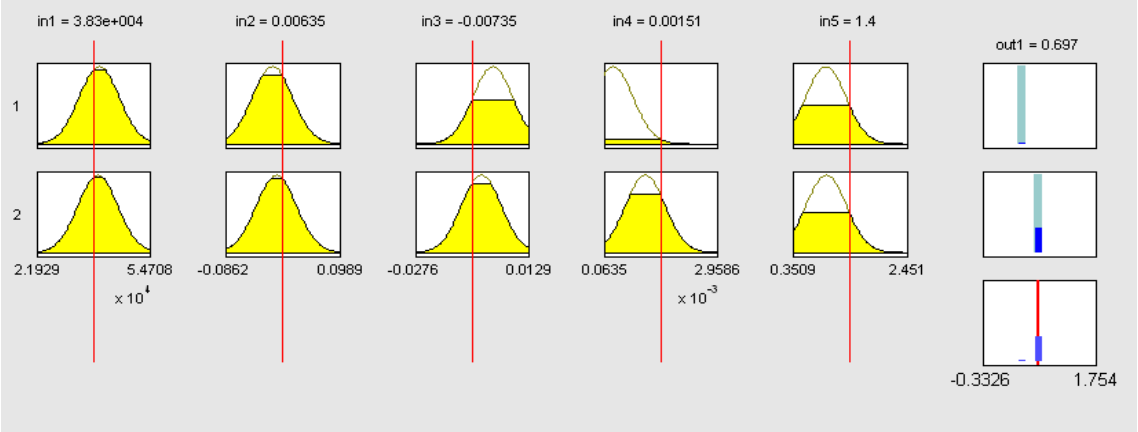

Figure 4.6 : FIS Rules

Figure 4.7 shows the model used in this study for predicting stock market crashes. As mentioned before, 5 inputs are fed into ANFIS model and one variable output is obtained at the end. The last node (rightmost one) calculates the summation of all outputs.

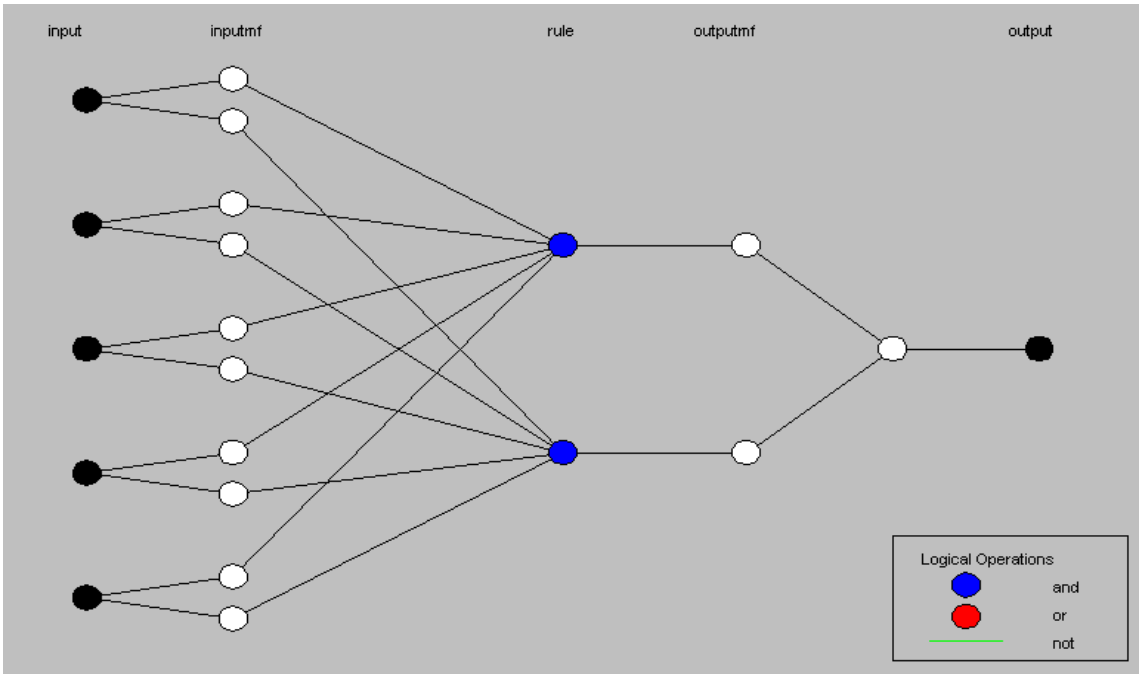

Figure 4.7 : FIS Model Structure

5. CONCLUSIONS

In this study, we presented a new methodology of early warning system for stock market crashes and this new methodology is able to provide excellent early warning information. We have constructed early warning systems by using various data mining classifiers. The empirical results show that the proposed neuro fuzzy model is the most successful.

ISE national 100 index data is non-linear and ANFIS can model non-linear system successfully. Consequently, the model learns patterns from the dataset and these patterns can help us decide upcoming stock market crashes and so the model is capable of indicating crash risks effectively.

One disadvantage of the ANFIS method is that the complexity of the algorithm is high when there are more than a number of inputs fed into the system. However, when the system reaches an optimal configuration of membership functions, it can be used efficiently against large datasets. Based on the accuracy of the results of the study, it can be stated that the ANFIS models can be used as an alternative to current financial early warning systems to predict stock market crashes.

ANFIS we used in this study is a part of a new and progressive technology, data mining and data mining is among the fastest increasing business technologies in the world. Early warning systems based on data mining are especially becoming integral parts of our daily lives. Early warning systems can reduce or eliminate losses that are caused by disasters. Our capability for forecasting future economic, social, and political discontinuities is much less well developed (Ayres, 2000). Therefore, we believe that scientists should put emphasis on early warning systems more than usual.

REFERENCES

Ayres R. U., 2000, On forecasting discontinuities, Technological Forecasting and Social Change 65, 81–97.

Bussiere M., & Fratzscher M., 2006, Towards a new early warning system of financial crises, Journal of International Money and Finance, 25 953-973.

Cajueiro, D. O., & Tabak, B. M., 2009, Werneck, F. K., Can we predict crashes? The case of the Brazilian stock market, Physica A 388 16031609.

Frankel, J. A., & Rose, A. K., 1996, Currency crashes in emerging markets: An empirical treatment, Journal of International Economics, vol. 41(3-4), pages 351-366, November.

Giudici, P., 2003, Applied data mining, John Wiley & Sons Ltd, ISBN: 0-470-84679-8 (Paper).

Han J., & Kamber M., 2006, Data mining: Concepts and techniques, Morgan Kaufmann Publishers, ISBN: 978-1-55860-901-3.

Heckerman D., 1997, Bayesian networks for data mining, Data Mining and Knowledge Discovery 1, 79–119.

Iseri M., & Caglar H., & Caglar N., 2008, A model proposal for the chaotic structure of Istanbul stock exchange, Chaos, Solitons and Fractals 36 1392–1398.

http://www.ise.org/.

JANG, J-SR., 1996, Input Selection for ANFIS learning, Proceedings of the IEEE International Conference on Fuzzy Systems, pp.1493-1499.

JANG, J-SR., 1993, ANFIS: Adaptive-network-based fuzzy inference system, IEEE Trans. Syst. Man Cybernet, Vol.23, No.3, pp.665–685.

JANG, J-SR., 1992, Self-learning fuzzy controllers based on temporal back propagation, IEEE Trans Neural Networks, Vol.3, No.5, pp.714–723.

Kaminsky, G., & Lizondo, S., & Reinhart, C., 1998, Leading indicators of currency crisis, International Monetary Fund Staff Papers, March Volume 45, Number 1.

Kim, T. Y., & Oh, K. J., & Sohn, I., & Hwang, C., 2004, Usefulness of artificial neural networks for early warning system of economic crisis, Expert Systems with Applications 26 583–590.

Larose, D. T., 2005, Data mining methods and models, John Wiley & Sons, Inc., ISBN: 978-0-471-66656-1.

Levy, M., 2008, Stock market crashes as social phase transitions, Journal of Economic Dynamics & Control 32 137–155.

Lin C., & Khan H. A., & Wang Y., & Chang R., 2006, A new Approach to modeling earlywarning systems for currency crises : Can a machine-learning fuzzy expert system predict the currency crises effectively?, CIRJE-F-411, April.

MATLAB, 2008, Getting started guide, The MathWorks, Inc., Twelfth printing, October.

MATLAB, 2008, Fuzzy logic toolbox user's guide, The MathWorks, Inc., Revised for Version 2.2.8 (Release 2008b), October.

McNelis P. D., 2005, Neural networks in finance: Gaining predictive edge in the market, Elsevier Academic Press, ISBN: 0-12-485967-4.

Peltonen, T. A., 2006, Are emerging market currency crises predictable? A test, European Central Bank, Working Paper Series, No. 571 / January.

Sivanandam, S. N., & Sumathi, S., & Deepa, S. N., 2007, Introduction to fuzzy logic using matlab, Springer, ISBN: 10 3-540-35780-7.

Trappey C. V., & Wu H., 2008, An evaluation of the time-varying extended logistic, simple logistic, and Gompertz models for forecasting short product lifecycles, Advanced Engineering Informatics 22 421–430

Witten, I. H., & Frank E., 2005, Data mining, Morgan Kaufmann Publishers, ISBN: 0-12-088407-0.

CURRICULUM VITAE

Personal

Name/Last Name	: Murat ACAR
E-mail	: muratacar@gmail.com
Place/Date of Birth	: İzmir - 1973
Marital Status	: Married
Military Service	: Completed
Smoking	: No
Driving Licence	: B Class

Experience

2007 – ... Istanbul Stock Exchange Takasbank Inc., ISTANBUL
Project Manager

1999 – 2007 Istanbul Stock Exchange Takasbank Inc., ISTANBUL
Software Specialist

Education

1991–1996 Ege University, Izmir
Faculty of Engineering B.Sc. in Computer Engineering

1988–1991 Ataturk Lisesi, Izmir