2019-01-01

Forecasting Crashes, Credit Card Default, and Imputation Analysis on Missing Values by the use of Neural Networks

Jazmin Quezada
University of Texas at El Paso

FORECASTING CRASHES, CREDIT CARD DEFAULT,
AND IMPUTATION ANALYSIS ON MISSING VALUES
BY THE USE OF NEURAL NETWORKS

JAZMIN QUEZADA

Master's Program in Mathematical Sciences

APPROVED:

Maria Christina Mariani, Ph.D., Chair

Joe Guthrie, Ph.D.

Kristine M Garza, Ph.D.

Stephen Crites, Ph.D.
Dean of the Graduate School

to my

FATHER, MOTHER and TOBY

with love.

FORECASTING CRASHES, CREDIT CARD DEFAULT, AND IMPUTATION ANALYSIS ON MISSING VALUES BY THE USE OF NEURAL NETWORKS

by

JAZMIN QUEZADA

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Mathematical Sciences

THE UNIVERSITY OF TEXAS AT EL PASO

August 2019

Acknowledgements

I would like to express my deep-felt gratitude to our Amazing God, for his unconditional love and abundant grace he has shown me throughout this journey.

My absolute appreciation goes to Dr. Maria Christina Mariani of the Mathematics Sciences Department at The University of Texas at El Paso, for her advice, encouragement, enduring patience and constant support. For pushing me and challenging me as a woman to evolve in the world of mathematics, which has brought my thesis to full completion. A great recognition to Los Alamos National Laboratory members Claire McKay Bowen Ph.D. Postdoctoral and fellow Joanne R. Wendelberger from the Statistical Sciences Group, CCS-6; for the amazing opportunity working along them. Also my deepest gratitude to the United States Department of Education (DoE) for providing the funding for my internship with Los Alamos National Laboratory.

I would also like to thank other members of The University of Texas at El Paso faculty, Dr. Andrew Pownuk for his constant belief in me (though I was often doubting in my own abilities), always providing clear explanations when I was (hopelessly) lost, constantly driving me with energy. His response to my verbal thanks one day was a very modest, "It's my job." To Dr. Amy Wagler, from the Department of Mathematical sciences for the inspiration as a strong woman and her dedication as an educator, with such passion she shows and gives to her students in and out of classes. I wish all students the honor and opportunity to experience their abilities to showcase their knowledge in there area of speciality. Deepest appreciation to Dr. Joe Guthrie and Dr. Kristine Garza, for their time and support in being part of my committee on completion to this thesis. I also wish to thank The University of Texas at El Paso Mathematical Sciences Department professors and staff for all their hard work and dedication, for providing the means and resources throughout my educational experience at the University in completing my degree. Osei Tweneboah P.hD. Teaching Assistant Computational Sciences Program The University of

Texas at El Paso, and Masum Md Al Bhuiyan P.hD. Teaching Assistant Computational Science Program at The University of Texas at El Paso, for their support and help they offered throughout my research.

My warmest gratitude goes to my dearest Mother, and brothers for their love, encouragement and constant support. With my deepest love and praise goes to my Father, may your soul rest in peace. Thank you for engraving in me this grit of never giving up and knowing that with hard work dreams do come true. Finally, to all my friends and loved ones who have in great ways contributed to make this work successful.

Abstract

A neural network is a system of hardware and/or software patterned after the operation of neurons in the human brain. Neural networks, also called Artificial Neural Networks are a variety of deep learning technology, which also falls under the umbrella of artificial intelligence, or AI. Recent studies shows that Artificial Neural Network has the highest coefficient of determination (i.e. measure to assess how well a model explains and predicts future outcomes.) in comparison to the K-nearest neighbor classifiers, logistic regression, discriminant analysis, naive Bayesian classifier, and classification trees. In this work, the theoretical description of the neural network methodology and some practical applications which are based on real world data are presented. We used the Multilayer perceptron (often simply called neural network) to identify financial market crashes and also compute the credit card default payments of customers of a financial institution. The problem of detecting market crashes and credit card default payments were modeled as a special class of classification problem. The neural network technique is very efficient and robust compared to other classification techniques since it correctly discriminates with good accuracy.

Table of Contents

Chapter 1

Introduction

Stock price is the price of a single stock among the number of other stocks sold by a company listed in public offering, having this allows the public to own a portion of it. One of the major problems of stock market is its difficulty on the market index prediction; this is of practical interest for a number of reasons. For one, stock prices change because of supply and demand. Suppose, if many people are willing to buy a stock, then the price goes up as there is more demand. With this we identify the high interest on how important the stock market historical data has become, and more accessible as an investment tool; not only for investors but also for the general population. Furthermore, monitoring the movements in a stock index is relatively more manageable in terms of time, and less demanding in terms of the knowledge required from an average investor. These characteristics of stock indices have made them popular in many markets across the globe. Hence, predicting the trend in an index from past data could help investors, especially novices. With that we come across financial crashes and its unique phenomena in financial industries due to its high interest worldwide. However, stock index values are noisy and non-stationary, with trends and periodic changes introduced by numerous factors affecting each constituent stock. Due to the non-linear, volatile and complex nature of the stock market, it is quite difficult to forecast. However, data analysts believe that the future stock market price can be predicted using historical stock market prices and past data. Data analysis uses multiple charts and calculations to find trends in the historical stock market data, which aims to predict the direction of the future price. The S&P 500 or Standard & Poor's 500 Index is a market-capitalization-weighted index of the 500 largest U.S. publicly traded companies. The index is widely regarded as the best gauge of large-cap U.S. equities. We take the financial time

series forecasting and its tool function with Neural Networks on forecasting crashes and other data sets. However, one of the main problems in the time series forecasting with the use of neural networks is we find the presence of noisy data. Which may become very difficult to extrapolate. By using special interpolation techniques and deep learning, it is possible to generate extra noisy data with superior accuracy. The objective in the time series forecasting is to use historically observed values of a quantity of interest in order to devise a model of its behaviour, which can be used to forecast future values over a short or long term period. With applications such as data mining, the growth of artificial intelligence, machine learning, statistics and database systems. Time series forecasting may be applied to data sets from various fields such as physics, engineering, and finance. Applications of time series analysis in a financial context include credit rating analysis, bankruptcy prediction, stock selection, stock index prediction. [1] There are many reasons to explain this phenomenon, first the growth of quantitative analysis which moved from manual work to artificial intelligence, with current work on Deep learning achieving great progress on natural language processing and presently continues to have great performances in various ranges of sciences. Throughout time we have evolved, were now historical data is becoming of great meaningful value.

The stock market prediction can be divided into two main focusing categories: *data and model.* A general linear process is a mathematical model for time series data applied when the successive values r_t , are regarded as being generated based on a series of independent inputs, a_t. If the sequence of inputs is such that each is drawn (i.i.d.) from a normal distribution with mean zero and constant variance σ_a^2; the model is called a *white noise* process. This is a state-of-art method, which can provide a complex, nonlinear fit to the data, but at the same time it is good for guarding against over-fitting. This is very important when dealing with noisy financial time series data. The stock market is constantly changing all the time and large fluctuations happen daily in the stock exchange, which intuitively neural networks and their properties shows great capacities. The majority of related studies on the application of machine learning techniques such as neural networks

to forecasting stock market have focused on forecasting the direction of a given stock past movements. Artificial intelligence or AI could be the new solution to tackle the forecasting on the stock market. In May 2017, the latest version of Google AlphaGo2 beat the worlds best human Go player, Jie Ke, through the perfect deep learning method. (Google) With some introduction to steps taken on applications on forecasting a stock market crash starting of with optimization process applied, along with neural networks, deep learning and other supporting methodologies. Early research on stock market prediction was based on the Efficient Market Hypothesis (EMH) [2] and the random walk theory [3, 4, 5]. According to the theory of random walks and market efficiency, the future direction of a stock is no more predictable than the path of a series of cumulative random numbers.[6] Some researchers suggest that stock prices moves by the theory of random walk, that is that the future path of the price of a stock is not any more predictable than random numbers. However, stock prices do not follow random walks is heavily cited by authors papers that claim, considerable empirical evidence exists that show how the stock returns are to some extent predictable. This means that we can make the basic assumption that past behaviour of a stock price is rich in information, and may show signs of future behaviour. Some claim to have shown that history is repeated in patterns, and that some of the patterns tend to recur in the future. Since there are patterns, it is possible through analysis and modeling to develop an understanding of such patterns. These patterns can further be used to forecast the future behaviour of stock prices. To continue the analysis on a time series, we can see a time series may be observed as a stationary time series, it can be described by its mean, variance, and autocorrelation function or spectral density function [7]. A strictly stationary time series r_t requires that the joint distribution of $(r_1, ..., r_k)$, for an arbitrary positive integer k, be time-invariant. A time-invariant system refers to a system that has a time-dependent system function, that is not a direct function of time. Furthermore, time-dependent system function is a function of the time dependency input. However, as this notion of strict stationarity is difficult to verify empirically, weak stationarity (also known as second-order stationarity, covariance stationarity, or wide-sense stationarity) is

often used. In this case, the mean $E(r_t)$ does not depend on t, and the covariance between r_{t1} and r_{t2} only depends on the difference $t_2 - t_1$. An important aspect of the forecasting task is represented by the size of the horizon. If the one-step forecasting on a time series is already a challenging task, performing multi-step forecasting is more difficult because of additional complications, like accumulation of errors, reduced accuracy, and increased uncertainty and perhaps some missing values. The forecasting domain has been influenced, for a long time, by linear statistical methods such as ARIMA models. Around the late 1970s to 1980s, an increasingly amount of notice was made on how linear models are not always adaptable to many real world applications. It is recorded that around the same time period, some useful nonlinear time series models were proposed such as the bilinear model, and the threshold autoregressive model (see for review [23] and [25]). Within the last two decades, machine learning models have drawn attention and have established themselves as serious contenders to classical statistical models in the forecasting community. Some of these models, also called black-box or data-driven models, are examples of nonparametric nonlinear models which use only historical data to learn the stochastic (randomly determined; having a random probability distribution or pattern that may be analyzed statistically but may not be predicted precisely) dependency between the past and the future. On one of my first literature review on *Generalization of Backpropagation with application to Recurrent model by Werbos , P.J.* he found that Artificial Neural Networks (ANNs) outperform the classical statistical methods such as linear regression and Box-Jenkins approaches [29]. Which has concluded that ANNs can be successfully used for modeling and forecasting nonlinear time series, leading us to the motivation of this research in using the various Neural Networks methodologies available. In this study we have worked with two different problems, 1) we use machine learning techniques to predict Credit Default payment and Stock Market evolution and crashes (chapters 2-4), 2) How to impute missing values in a data set with imputations methods and along with neural network methodologies (chapter 5).

1.1 Motivation

Artificial neural networks are motivated by the learning capabilities of the human brain which consists of neurons interconnected by synapses. In fact at least theoretically they are able to learn any given mapping up to arbitrary accuracy [**?**, HSW89]. For as long as we have stock markets, the prediction of stock markets will only continue, with the increase of historical data available. Stock prices are considered to change quite frequently due to the financial domain and the factors affecting the company etc. Different kinds of financial time-series have been recorded and studied for decades. Nowadays, all transactions on a financial market are recorded, leading to a huge amount of data available, either for free in the Internet or commercially. Financial time-series analysis is of great interest to practitioners as well as to theoreticians, for making inferences and predictions. In the recent years, efforts have been put into applying machine learning to forecasting the stock market, however, most impressive work is the promising work it continues to show in various sciences (e.g. Health and Medicine field). For the remaining chapters we will see the following structure, chapter two introduction to time series and some of applications applied with some basic notions of time series modeling and the formalization of the forecasting task as an input-output problem. Chapter three we will introduce methodologies used for our research data, which will be talked about on chapters four and five. Finishing with chapter six with conclusions and remarks. The main goal of this study is to determine whether or not it is possible to make a profitable predictions using machine learning methodologies with its wide range of operations not only on financial data but on various of data sets.

Chapter 2

Time Series

In this chapter, we will give a summary of the background knowledge related to the Time Series along with its application on this thesis. The usage of time series models is two:

1. Obtain an understanding of the underlying forces and structure that produced the observed data.

2. Fit a model and proceed to forecasting, monitoring or even feedback and feedforward control.

The definition of Time Series is a sequence of observations on a variable measured at successive n points in time or over successive periods of time. It is mathematically defined as $x(t), t = 0, 1, 2, ...$ where t represents the time elapsed. The measurements taken during an event in a time series $x(t)$, are arranged in a proper chronological order. Where each observation can be continuous or discrete. In a continuous time series, observations are measured at every instance of time. Whereas a discrete time series, contains observations measured at discrete points of time. For example temperature readings, flow of a river, concentration of a chemical process etc. can be recorded as a continuous time series. On the other hand, population of a particular city, production of a company, and exchange rates between two different currencies may represent discrete time series. If the future values of a time series can be exactly determined using a mathematical function, then it is called deterministic. A time series whose future values can only be described by a probability distribution is said to be a statistical time series. The term stationary time series is used to denote a time series whose statistical properties are independent of time, as mentioned previously check [7] for a formal definition of stationary. This means that,

Figure 2.1: Dow Jones 100 year historical chart.

1. The process generating the data has a constant mean,

2. The variability of the time series is constant over time.

above we find the time series Dow Jones industrial Average stock market index for the last 100 years figure. A time series plot for a stationary time series will always exhibit a horizontal pattern. But simply observing a horizontal pattern is not sufficient evidence to conclude that the time series is stationary. As we can view on figure 2.1, although time series data generally can exhibit random fluctuations, a time series may also show gradual shifts or movements to relatively higher or lower values over a longer period of time.

If a time series plot exhibits this type of behavior, we say that a horizontal pattern exists. The trend of a time series can be identified by analyzing multi-year movements in historical data. Seasonal patterns are recognized by seeing the same repeating patterns over successive periods of time, some time series include a combination of a trend and seasonal pattern. For a horizontal pattern there exists data that fluctuate around a constant mean. With seasonal pattern exists if the time series plot exhibits a repeating pattern over successive periods. The successive periods are often one-year intervals, which is where the name seasonal pattern comes from. Finally, we look at cyclical pattern existing if the time series

plot shows an alternating sequence of points below and above the trend line lasting more than one year. The underlying pattern in the time series is an important factor in selecting a forecasting method. Thus, a time series plot should be one of the first things developed when trying to determine what forecasting method to use. Time series is used in various sectors nowadays such as economics, finance, environment, education and health [8]. One possible objective in analyzing a time series, especially in an economic context, is to predict accurately the future value of an observed time series. The trend is calculated by average, from long term movement using symmetric moving average. If it is a seasonal effect, it can be estimated using centered-moving average. A moving average is a calculation used to analyze data points, by introducing a series of averages from different parts of the complete data. The moving average applied repeatedly to a time series, can create artificial cycles to smoothen the data. We can write a time series as $\{x1, x2, ...xT\}$ or $xt, t = 1, 2, ...T$, we will treat x_t as a random variable as stated before. As for selecting a forecasting method, a key concept associated with measuring forecast accuracy is the forecast error, defined as *Forecast Error = Actual Value - Forecast.* Due to the positive and negative forecast errors tend to off-set one another, the mean error is likely to be small; thus, the mean error is not a very useful measurement of forecast accuracy. The *mean absolute error*, denoted as MAE, is a measurement of forecast accuracy that avoids the problem of positive and negative forecast errors offsetting one another. As you might expect given its name; MAE is the average of the absolute values of the forecast errors. Another measurement that avoids the problem of positive and negative forecast errors off-setting each other is obtained by computing the average of the squared forecast errors. This measure of forecast accuracy, referred to as the mean squared error, denoted as *MSE = average of the sum of squared forecast errors.* To make comparisons like these we need to work with relative or percentage error measures. *The mean absolute percentage error*, denoted *MAPE* is such measurement. To compute MAPE we must first compute the percentage error for each forecast. Unfortunately, there is no way to address the issue of accuracy associated with forecasts for future time periods. But if we select a forecasting method that works well for

the historical data, and we think that the historical pattern will continue into the future, we should obtain results that will ultimately be shown to get factual results. For every measure done, the average of the past values provides more accurate forecasts then using the most recent observation as the forecast for the next period. In general, if the underlying time series is stationary, the average of all the historical data will always provide the best results. With the various number of shifts occurring we get a new level to the series that arise, it will take longer time for the forecasting method that uses the average of all the historical data to adjust to the new level of the time series. But in most cases, the simple naive method adjusts very rapidly to the change n level because if uses the most recent observation available as the forecast.

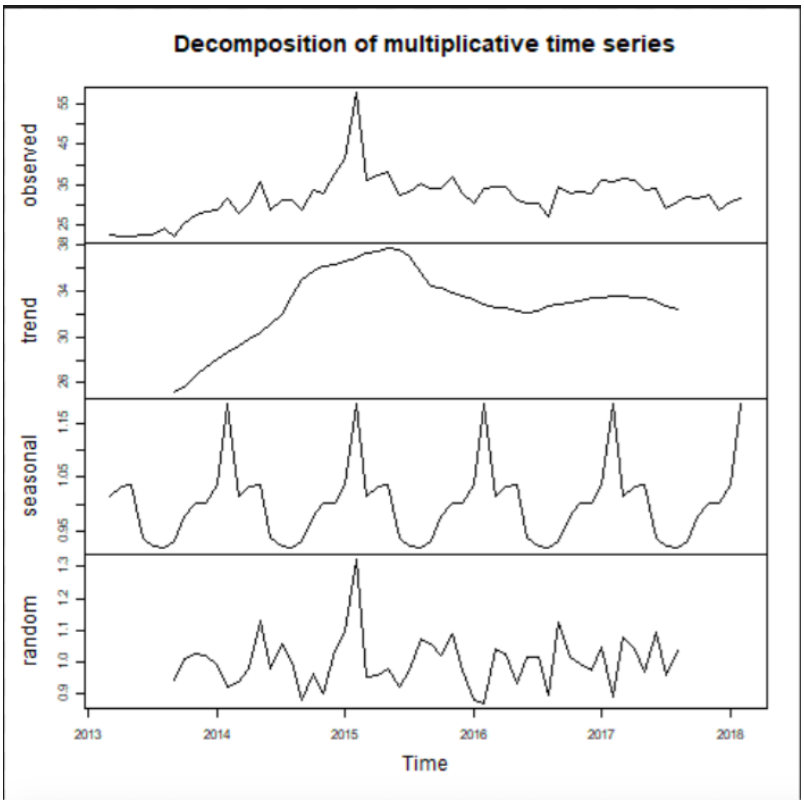


Figure 2.2: Decomposition of multiplicative time series.

The fitting of the Time series models can be an earnest undertaking. There are many

methods of model fitting, two main interpretations of the forecasting problem on the basis of historical data set exist. There are three main forecasting methods that are appropriate for a time series with a horizontal pattern which is found in our time series data: *moving averages, weighted moving averages,* and *exponential smoothing.* The objective of each of these methods is to "smooth out" the random fluctuations in the time series, usually referred to as smoothing methods. These methods are some of the simplest methods to use, they generally provide a high level of accuracy for the short-range of forecasts, such as forecasting for the next time period. For the *moving averages method*: let us quickly define function $f(x)$ and let it be some window size $s > 0$. The moving average of f is the function $Af(x)$ defined by:

$$Af(x) = \frac{1}{s}\int_{x-s/2}^{x+s/2} f(t)dt,$$

where A is the "averaging operator". This is an example of a functional transform: it takes one function f and puts out another function Af. The $1/s$ term ensures that if f is a constant, then Af is the same constant. We can then apply the moving average several times in order to improve the behaviour of a function. However, it is possible to make operators which work better in a single step. For example, consider the following triangular weighted moving average:

$$Wf(x) = \int_{x-s/2}^{x+s/2} w(x-t)f(t)dt,$$

where the weight function is

$$w(x) = \begin{cases} 4x/s^2 + 2/s & -s/2 \leq x \leq 0, \\ 2/s - 4x/s^2 & 0 \leq x \leq s/2. \end{cases}$$

Then if we apply this to the jump function, the outcome is piecewise quadratic:

$$Wf(x) = \begin{cases} 0 & < \ x < \ -s/2, \\ 2(x/s + 1/2)^2 & -s/2 \leq x \leq 0, \\ 1 - 2(x/s - 1/2)^2 & 0 \leq x \leq s/2 \\ 1 & x > s/2. \end{cases}$$

A more successful method, that turns any integrable function into a infinitely differentiable (i.e. perfectly smooth) is the Gaussian blur, noted as follows:

$$Gf(x) = \int_{-\infty}^{\infty} g(x-t)f(t)dt,$$

where g is just the Gaussian probability density function,

$$g(x) = \frac{1}{s\sqrt{2\pi}} e^{-\frac{x^2}{2s^2}}$$

for some constant $s > 0$. (i.e. this corresponds to an "infinite window") The usage of the average on the most recent k data values in the time series as the forecast for the next period. Mathematically, moving average forecast of the order k is as follows:

$$F_{t+1} = \sum_{k}^{t} \frac{(most\ recent\ k)}{k} = \frac{Y_t + Y_{t-1} + ... + Y_{t-k+1}}{k},$$

where F_{t+1} = forecast of the time series for period $t + 1$; Y_t = actual value of the time series in period t. The term *moving* averages is used because every time a new observation becomes available for the time series, it then will be replaced by the oldest observation in the equation and a new average is computed. The result of the average will change, or move, as new observations become available. To be able to use the moving averages to forecast a time series, we must first select the order, or number of time series values, to be included in the averaging. If only the most recent values of the time series are considered relevant,

then a small value of k is preferred. Now if more past values are considered relevant, now a larger value of k would work better. Recall that a time series with a horizontal pattern can shift to a new level over time. A moving average will adapt to the new level of the series and resume on providing forecasts in k periods. Furthermore, the smaller value of k will track shifts in a time series quickly rather than in the case of larger values of k will be more effective in smoothing out the random fluctuations over time.

On the second method for forecasting we have *weighted moving averages*, on this method we take each observation in the moving average calculation receives the same weight, involve it by selecting a different weight for each data value and then computing a weighted average of the most recent k values as the forecast. For most cases, the most recent observation receives the most weight, and the weight decreases for older data values. In technical analysis of financial data, a weighted moving average (WMA) has the specific meaning of weights that decrease in arithmetical progression.[4] In an n day WMA the latest day has weight n, the second latest $n-1$, etc., down to one.

$$WMA_M = \frac{np_M + (n-1)p_{M-1} + ... + 2p_{(M-n+2)} + p_{(M-n+1)}}{n + (n-1) + ... + 2 + 1}.$$

When we are calculating the WMA across successive values, we get the difference between the numerators of WMA_{M+1} and WMA_M is $np_{M+1} - P_M -, , , -p_{M-n+1}$. Denote the sum of $P_M + ... + p_{M-n+1}$ by the $Total_M$, then we have the following:

$$Total_{M+1} = Total_M + p_{M+1} - p_{M-n+1}$$

$$Numerator_{M+1} = Numberator_M + np_{M+1} - Total_M,$$

$$WMA_{M+1} = \frac{Numberator_{M+1}}{n + (n-1) + ... + 2 + 1}.$$

To follow with a quick example, WMAs can have different weights assigned based on the number periods used in the calculation. If we want a weighted moving average of four different prices, then the most recent weighting could be 4/10, the period before could have

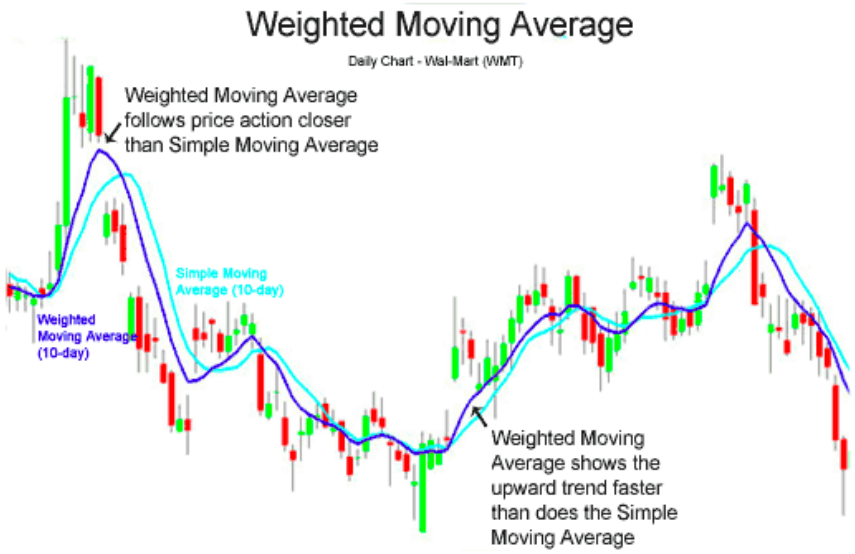


Figure 2.3: Weighted moving average.

a weight of 3/10, the period prior to that could have a weighting of 2/10, and so on. Now lets assume the following prices of 90,89,88,89, with the most recent price first. You would calculate this as

$$[(90 \times (4/10) + (89 \times (3/10)) + (88 \times (2/10)) + 89 \times (1/10))]$$

$$= 36 + 26.7 + 17.6 + 8.9 = 89.2.$$

Recall that we can change weights around to have better average according to information provided.

Another most commonly used method we have *Exponential smoothing*, which also uses weighted average of the past time series values as a forecast, so we will similarities with the WMA method. It is a special case for the weighted moving averages method in which we select only one weight-by the weight for the most recent observation. The weights for the other data values are computed automatically and become smaller as the observations

move farther into the past. The Exponential smoothing equation as follows:

$$F_{t+1} = \alpha Y_t + (1 - \alpha) F_t$$

where, F_{t+1} = forecast of the time series for period $t + 1$, with Y_t= actual value of the time series in period t. Following with F_t = forecast of the time series for period t , α = smoothing constant $(0 \leq \alpha \leq 1)$. For forecast accuracy in the exponential smoothing calculations we use a smoothing constant of $\alpha = .2$. Although any value of α between 0 and 1 is acceptable, some values will yield better forecasts than others. Insight into choosing a good value for α can be obtained by rewriting the basic exponential smoothing model as follows:

$$F_{t+1} = \alpha Y_t + (1 - \alpha) F_t$$

$$F_{t+1} = \alpha Y_t + F_t - \alpha F_t,$$

$$F_{t+1} = F_t + \alpha (Y_t - F_t).$$

Therefore, we have the new forecast F_{t+1} is equal to the previous forecast F_t plus an adjustment, which is a smoothing constant α times the most recent forecast error, $Y_t - F_t$. Moveover, that the forecast in period $t+1$ is obtained by adjusting the forecast in period t by a fraction of the forecast error. As we have expressed some of the components of a Time Series, in practice a suitable model is fitted to a given time series and the corresponding parameters are estimated using the known data values. The procedure of fitting a time series to a proper model is termed as Time Series Analysis, which goes in hand with our data analysis and when analyzing a given set of data recommended adhering are the following steps: 1) exploratory data analysis and 2) confirmatory data analysis. It comprises methods that attempt to understand the nature of the series and is often useful for future forecasting and simulation. The simple exponential smoothing method requires little computation, and it is used when data pattern neither have not a cyclic variation nor trend in the historical data. With the *single exponential smoothing* method gets smoothing curve using only one

actual past value and one predicted past value, as follows:

$$S_t = \alpha y_{t-1} + (1-\alpha)S_{t-1}.$$

Where, S_t represents the predicted value (or smooth value) when $T = 1$, and S_{t-1} meaning that the smooth value when $T = t-1$. Then y_{t-1} represents the actual value of series when $T = t-1$, with α, ranges from 0 to 1, is the smoothing parameter. Which from there we get the representation of the smoothing of the series. One significant advantage of exponential smoothing method is computational efficient when dealing with mass of data. An extension to the exponential smoothing we have the *Double exponential smoothing* method. Double exponential allows forecasting values and anticipating a trend, the formulation for the Double Exponential method given by:

$$S_t = \alpha y_t + (1-\alpha)(S_{t-1} + T_(t-s)$$

$$T_t = \gamma(S_t + \ S_{t-1}) + (1-\gamma)T_{t-q}.$$

$$\hat{y}_t(k) = S_t + kT_t.$$

Where S_t and T_t are the smoothed level and trend; and α and γ are the smoothing parameters. We have y_t being the actual value of the time series in period t; and $\hat{y}_t(k)$ is the k step - ahead forecast made from the forecast origin t. We also extend this to *Triple Exponential smoothing*, has two models of computation, that is additive and multiplicative manner. The additive model calculations performed if the plot of the original data showed some seasonal fluctuations. As for multiplicative models are used when the plot of the original data showed seasonal fluctuations variations. Below find the equations used for additive model where L represents the level of estimate influenced by the amount of α, and T are the trend estimate influenced by the β, and S, and p represents the trend period, l is the length, Y is the actual data, as $\hat{y}$ means the value of the forecast for the coming

period; and f is for the seasonal estimate series.

$$L_t = \alpha(Y_t - S_{t-1}) + (1-\alpha)(K_{t-1} - T_{t-1})$$

$$T_t = \beta(L_t + L_{t-1}) + (1-\beta)T_{t-1|}.$$

$$S_t = \gamma(Y_t + L_t) + (1-\gamma)S_{t-l},$$

$$y_t \hat{+} p = L_t + pT_t + S_{t-l+p}.$$

2.1 Data Analysis

Data analysis is the process of exploring through the data set, cleaning, transforming and making supporting decision-making on further modeling. First we start by defining the type of data we are dealing with, which are two types. *Qualitative* data, which is represented either in verbal or narrative format. These types of data are collected through focus groups, interviews, opened ended questionnaire items, and other less structured situations. A simple way to look at qualitative data is to think of qualitative data in the form of words. *Quantitative* data is data that is expressed in numerical terms, in which the numeric values could be large or small. Numerical values may correspond to a specific category or label. As for when it comes to analyzing given set of data within an overall scientific investigation, adhering to the following two steps:

- exploratory data analysis
- confirmatory data analysis

The main purpose of the exploratory data analysis phase of data analysis is to discover important statistical properties in the given observations by carrying out simple graphical and numerical studies. As for the objective of the confirmatory data analysis stage is to confirm statistical in a rigorous fashion the absence or presence of certain properties in the

data. Once we have analysed the data we then move on with modeling of the time series data. Having two types of time series data, to univariate and multivariate data. One may classify multivariate methods with regard to whether they are atheoretical (unrelated to or lacking a theoretical basis), such as time-series models, or structural or theory-based. Multivariate models have more parameters than univariate ones. Every additional parameter is an unknown quantity and has to be estimated. This estimation brings in an additional source of error due to sampling variation. Outliers can have a more serious effect on multivariate than one univariate forecasts. Along with an important exploratory tool for modeling multivariate time series is the cross-correlation function (CCF). The CCF generalizes the autocorrelation function to the multivariate case. Thus, its main purpose is to find linear dynamic relationships in time series data that have been generated from stationary processes. In analogy to the univariate case, a multivariate process, X_t is called (covariance) stationary if,

$$1. EX_t = \mu$$

i.e the mean is a time-constant n -vector;

$$2. E(X_t - \mu)(X_t - \mu)^{'} = \sum,$$

i.e. the variance is a time-constant positive definite $n \times n$ matrix $\sum$

$$3. E(X_t - \mu)(X_{t+k} - \mu)^{'} = \Gamma(k),$$

i.e. the covariance is over time depend on lag only, with non-symmetric $n \times n$- matrices $\Gamma(k)$.

Recall that for a univariate time series, as the name suggests, is a series with a single time-dependent variable. As for Multivariate Time Series (MTS), it has more than one time-dependent variable. Each variable depends not only on its past values but also has some dependency on other variables. This dependency is used for forecasting future values.

For coming all of the information to this point we come across the importance re-framing of the time series data, which allows us to access the suite of standard linear and nonlinear machine learning algorithms on our problem. Along with preparing our data for forecasting, that is separation of data to *training data* and *test data*, as we move forward this will be elaborated on. This brings to the *sliding window method* for framing a time series modeling for forecast. In the sliding window method, a window of specified length, *Len,* moves over the data, sample by sample, and the statistic is computed over the data in the window. The output for each input sample is the statistic over the window of the current sample and the *Len* - 1 previous samples. In the first-time step, to compute the first *Len* - 1 outputs when the window does not have enough data yet, the algorithm fills the window with zeros. In the subsequent time steps, to fill the window, the algorithm uses samples from the previous data frame. The moving statistic algorithms have a state and remember the previous data. The application to the sliding window or lag method for multivariate data and multi-step forecasting. Which this becomes of importance when dealing with a multivariate data-set with missing values as that found in Los Alamos research project done at the being of summer. Coming along we see the important role of data assembling before moving forward with any other applications done, when dealing with missing values within the data-set; imputation methods are to be dealt with before moving forward with training data set and forecasting methods.

2.2 Missing values

In time series data, if there are missing values, there are two ways to deal with incomplete data :

- omit the entire record that does not contain information.
- Impute the missing information.

Before moving forward lets review why omitting data is not in our best interest, by

discarding any case that has a missing values, which may introduce bias or affect the representativeness of the results. Before moving forward with imputation methods, lets classify the time series according to composition. If we decompose the time series data with linear regression model for example we have $Y_t = m_t + s_t + \epsilon_t$. Where m_t stands for trend, s_t stands for seasonal, and ϵ_t stands for random variables. As mentioned before, the importance on looking at the type of trend the time data set produces, before moving forward with forecasting. Formal description on imputation, in statistics imputation is the process of replacing missing data with substituted values. When substituting for a data point, it is known as "unit imputation", when replacing a component of a data point, it is known as "item imputation". As for non-time-series specific imputation methods we have *mean imputation*, *median imputation*, and *mode imputation*; which calculate the appropriate measure and replaces $NA's$ (unavailable data) with the values. Imputation is a method to fill in the missing values with estimated ones. The objective is to employ known relationships that can be identified in the valid values of the data set to assist in estimating the missing values. As previously stated, Mean / Mode / Median imputation are one of the most frequently used methods. It consists of replacing the missing data for a given attribute by the mean or median (quantitative attribute) or mode (qualitative attribute) of all known values of that variable. For not only we encounter the idea of missing data but there is also different types of missing data. (MCAR) Missing Completely at Random; means that the probability that an observation is missing is not related to its value or to any other values in the data set. Missing at Random (MAR) means that the probability that an observation is missing is related to the values for some other observed variables. Finally, we have (MNAR) missing not at random, meaning that the probability that an observation is missing is related to its value. Depending on the proportion and the generating mechanism of missing data. We have three main problems that missing data causes: missing data can introduce a substantial amount of bias, make the handling and analysis of the data more arduous, and create reductions in efficiency. For imputation preserves all cases by replacing missing data with an estimated value based on other available information. Now applying

time to a data-set will likely increase the amount of missing data throughout time collecting. Once all missing values have been imputed (that is, filled in) the data set can then be analysed using standard techniques via neural networks for complete data. It is probably a popular misunderstanding that the goal of imputation is to predict individual missing values. This is popular because of hot deck imputation method (hot deck imputation is a method for handling missing data in which each missing value is replaced with an observed response from a similar unit), which the attempt to find the best match for each missing case. A better estimate for each missing value not necessarily leads to a better overall estimate for the parameters of interest. So here is a counterexample given by *Rubin (1996)*: suppose we have a coin that, in truth, is biased .6 heads and .4 tails. This known truth model A, whereas model B asserts that the coin has two heads. Using model A for creating imputations (i.e., future predictions) yields a hit rate (agreements between predictions and outcomes) of $.6.6 + .4.4 = .52$, whereas using model B for predictions yields a hit rate of .6. This does not mean that model B is better than model A for handling missing values. Filling in missing values using model B yields the invalid statistical inference that in the future all coin tosses will be heads, clearly inconsistent for the estimand $Q = fraction$ of tosses that are heads, whereas using model A yields consistent estimates for all such scientific estimands. In R, missing values are indicated by NAs. In classical regression (as well as most other models), R automatically excludes all cases in which any of the inputs are missing; this can limit the amount of information available in the analysis, especially if the model includes many inputs with potential missingness, leading us to a complete-case analysis. Missing outcomes in a regression can be handled easily by simply including the data vector, NAs and all. To decide how to handle missing data, it is helpful to know why they are missing. Here are the three main concepts on missing values, we have MCAR, MAR, and MANAR; to consider before moving forward on completing the data for further analysis. Rather than removing variables or observations with missing data, another approach is to fill in or impute missing values. A variety of imputation approaches can be used that range from extremely simple to rather complex. These methods keep the

full sample size, which can be advantageous for bias and precision; however, they can yield different kinds of bias. A single imputation strategy is used, the standard errors of estimates tend to be too low. The intuition here is that we have substantial uncertainty about the missing values, but by choosing a single imputation we in essence pretend that we know the true value with certainty. It is common to have missing data in several variables in an analysis, in which case one cannot simply set up a model for a single partially observed variable y given a set of fully observed X variables. More generally, we must think of the data-set as a multivariate outcome, any components of which can be missing. A direct approach to imputing missing data in several variables is to fit a multivariate model to all the variables that have missingness, thus generalizing the approach to allow the outcome Y as well as the predictors X to be vectors. The most difficult part on this approach is that it requires a lot of effort to set up a reasonable multivariate regression model. Most commonly the multivariate normal or t distribution for continuous outcomes, and a multinormal distribution for discrete outcomes.

Chapter 3

Methodologies

Before going into further explanation on forecasting methods, and how they may be classified as qualitative or quantitative data-sets. As mentioned on previous chapter *qualitative* methods generally involve the use of expert judgment to develop forecasts. Such methods are appropriate when historical data is on the variable being forecast are either not applicable or unavailable (Na). Quantitative forecasting methods can be used when (1) past information about the variable being forecast is available, (2) the information can be quantified, and (3) it is reasonable to assume that the pattern of the past will continue into the future. When all of these align a forecast can be developed using a time series method. The objective of time series analysis is to discover a pattern in the historical data or time series and then extrapolate the pattern into the future; the forecast is based solely on past values of the variable and/or on past forecast errors.

Stock market prediction has attracted much attention from both academia and business, and the question remains: "To what extent can the past history of a common stocks price be used to make meaningful predictions concerning the future price of the stock?" [2]. Neural networks can adapt to changing input so the network generates the best possible result without needing to redesign the output criteria. Using machine learning methods for time series forecasting, with the intuition behind this approach is that machine learning can provide complex, non-linear models, but without the difficulty in fitting such models encountered in the financial time series. Let us begin by talking about sequence problems. The simplest machine learning problem involving a sequence is a one to one problem, as shown in figure [3.1]. As we continue the process of chapter 3, we will continue introducing the machine learning methodologies applied to our three data-sets.

3.1 Simple Neuron

Lets start with the introduction to a **neuron** with a single scalar input and no bias is shown below. 3.1.

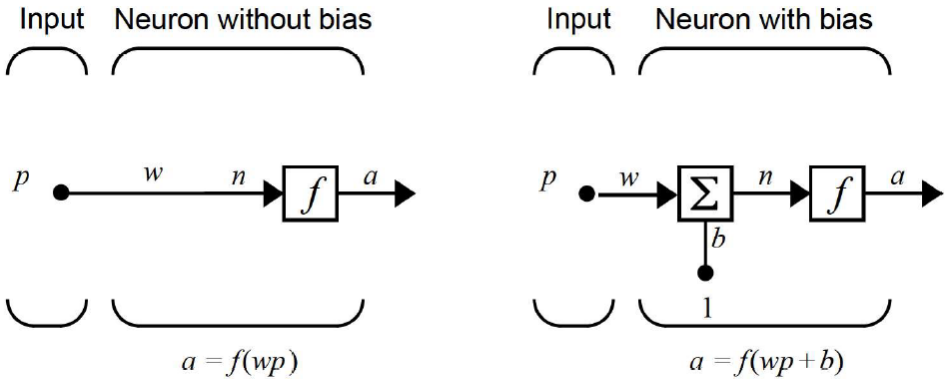


Figure 3.1: One to one system

The scalar input p is transmitted through a connection that multiplies its strength by the scalar weight w, to form the product $w \cdot p$, again a scalar. Here the weighted input $w \cdot p$ is the only argument of the transfer function f, which produces the scalar output a. The neuron on the right has a scalar bias, b. You may view the bias as simply being added to the product wp as shown by the summing junction or as shifting the function f to the left by an amount b. The bias is much like a weight, except that it has a constant input of 1.

The **transfer function** net input n, again a scalar, is the sum of the weighted input wp and the bias b. This sum is the argument of the transfer function f. Three of the most commonly used functions are shown below. Sigmoid transfer function is commonly used in backpropagation networks, in part because it is differentiable.

3.1.1 Neuron With Vector Input

A neuron with a single R-element input vector is shown below. Here are the individual element inputs

$$p_1, p_2, ..., p_R,$$

are multiplied by the weights

$$w_{1,1}, w_{1,2}, ..., w_{1,R}$$

and the weighted values are feed to the summing junction. Their sum is simply $W \cdot p$, the dot product of the (single row) matrix W and the vector p.

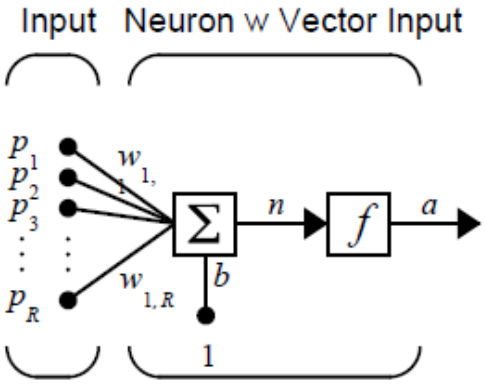


Figure 3.2: Neuron With Vector Input

The neuron has a bias b, which is summed with the weighted inputs to form the net input n. This sum, n, is the argument of the transfer function f.

$$n = w_{1,1}p_1 + w_{1,2}p_2 + ... + w_{1,R}p_R + b.$$

This expression can, of course, be written in as

$$n = W \cdot p + b$$

3.1.2 A Layer of Neurons

Here we have a one-layer network with R input elements and S neurons as follows. In this

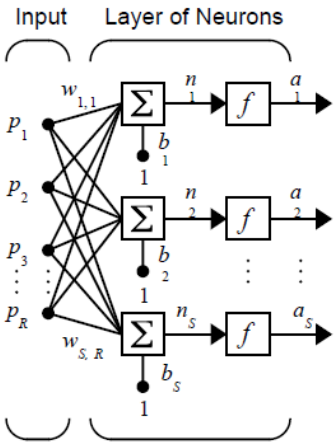


Figure 3.3: A Layer of Neurons

network, each element of the input vector p is connected to each neuron input through the weight matrix W. The i^{th} neuron has a sum that gathers its weighted inputs and bias, to form its own scalar output $n(i)$. The various $n(i)$ taken together form an S-element net, input vector n. Finally, the neuron layer outputs form a column vector a.

The input vector elements enter the network through the weight matrix W.

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ & & \dots & \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix}$$

The row indices on the elements of matrix W indicate the destination neuron of the weight, and the column indices indicate which source is the input for that weight. Thus, the indices say that the strength of the signal from the second input element to the first (and only) neuron is. The S neuron, R input one-layer network also can be drawn in abbreviated notation. Here p is an R length input vector, W is an $S \times R$ matrix, with a and b are

S length vectors. As defined previously, the neuron layer includes the weight matrix, the multiplication operations, the bias vector R, the sum and the transfer function boxes.

$$a = f(Wp + b). \tag{3.1}$$

3.1.3 Multiple Layers of Neurons

To move into a bigger network we can have several layers. Each layer has a weight matrix W, a bias vector b, and an output vector a. To distinguish between the weighted matrices, output vectors, etc.. For each of these layers in our figures, we append the number of the layer as a superscript to the variable of interest. You can see the use of this layer notation in the three-layer network shown below, and in the equations at the bottom of the Fig. 3.4.

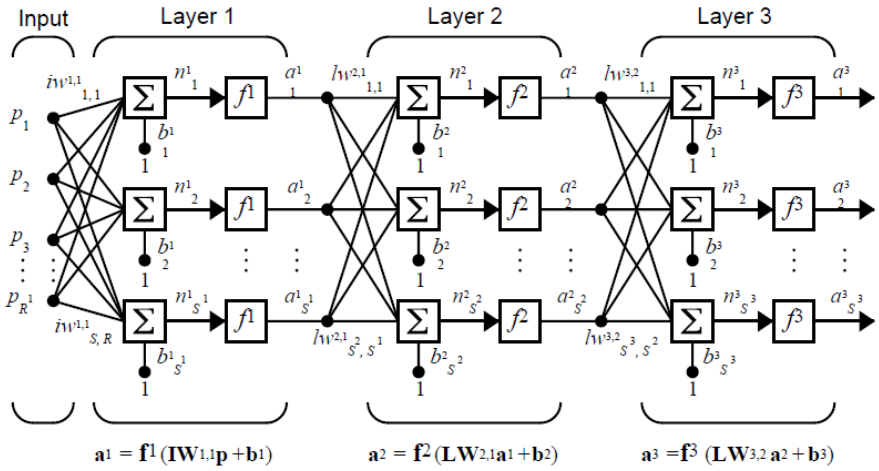


Figure 3.4: Multiple Layers of Neurons

The layers of a multilayer network play different roles, for a layer that produces the network output is called an output layer. All other layers are called hidden layers. The three-layer network shown earlier has one output layer (layer 3) and two hidden layers (layer 1 and layer 2). Some authors refer to inputs values as a fourth layer. We will not use that designation. The same three-layer network discussed previously also can be drawn

using our abbreviated notation.

We have the output of the neural network that can be calculated in the following way

$$y = a^3 = f^3(LW^{3,2}f^2(LW^{2,1}f^1(IW^{1,1}p + b^1) + b^2) + b^3), \tag{3.2}$$

where $IW^{1,1}$ is the weight matrix of the layer 1, $LW^{2,1}$ is the weight matrix of the layer 2, and $LW^{3,1}$ is the weight matrix of the layer 3. Biases of the layers are b^1, b^2, b^3. Multi-layer neural network are powerful models with non-convex objective functions. Although our convergence analysis does not apply to non-convex problems, we empirically found that Adam optimizer often outperforms other methods in such cases. Which later application will be shown along with a neural network model with two fully connected hidden layers with 1000/ (we can change number and analysis results) hidden units each and ReLU activation are used.

3.2 Rolling-Window Analysis of Time Series Models

Suppose that we have some data that for all periods in the sample [9].

1. Choose a rolling window size, m ;i.e., the number of consecutive observation per rolling window. The size of the rolling window depends on the sample size, T, and periodicity of the data. In general, we can use a short rolling window size for data collected in short intervals, and with a larger size for data collected during longer intervals. Longer rolling window sizes will tend to yield.

2. Choose a forecast horizon ,h. The forecast horizon depends on the application and periodicity of the data. The following illustrates how the rolling window partitions the data set.

3. If the number of increments between successive rolling windows is 1 period, then

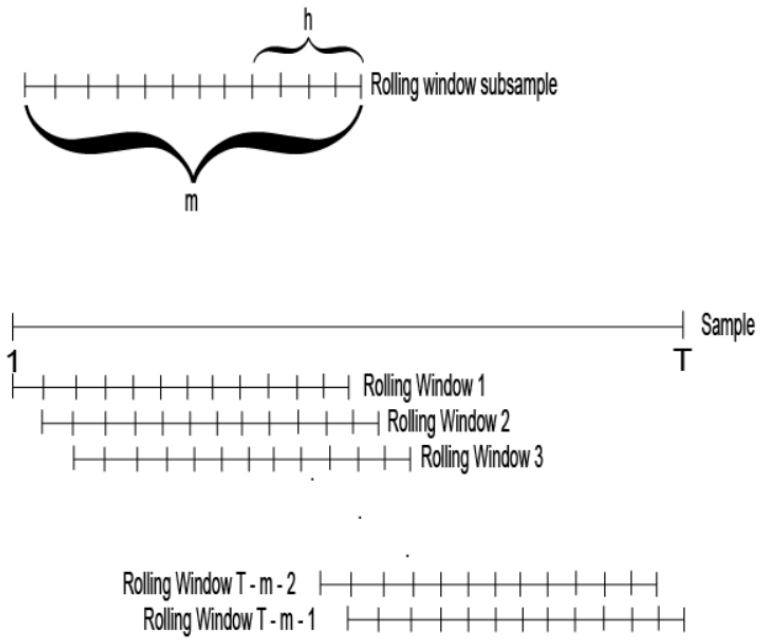


Figure 3.5: Rolling-Window for Time Series Analysis [10]

partition the entire data set into $N = T - m + 1$ subsamples. The first rolling window contains observations for the period 1 through m, second rolling window contains observations for period 2 through $m + 1$, and so on. The figure illustrates the partitions.

For the remaining chapters, we will work with data sets where our goal is to forecast a customer default on their credit card debt, along with the forecasting of a stock market crash on 9 different countries. The application of imputation on a missing values data set with our Educational data, and with further applications done with the simulation of educational data via neural networks

3.2.1 Sliding Window

Sliding window approach, is a window of suitable size, say $m \times n$ is chosen to perform a search over the target image a process. The whole cross-sectional data set is divided into

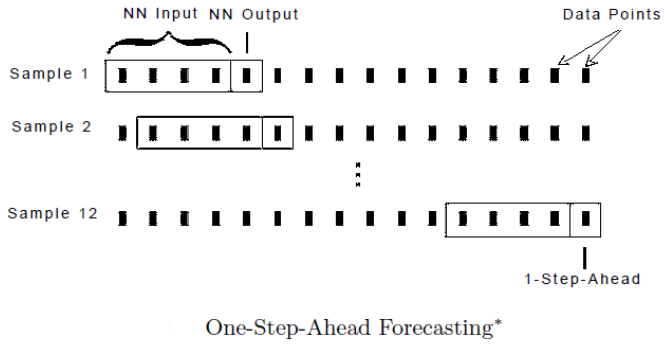


Figure 3.6: One-step ached forecasting

different training windows by specifying the window width. A model is trained using a training window and applied on the testing window to compute the performance for the first run. For the next run, the training window is slide to new set of training records and the process is repeated until all the training windows are used. Using this method we take average across the entire data set for metric performance. With neural networks input presentation through a sliding window allows to include all the local information conducive to a given output. Constructing a window classifier h_w that maps an input window of width w into an individual output value y. More precise, let $d = (w-1)/2$ be the "half-width" of the window. Then we have h_w that predicts $y_{i,t}$ using the window

$$\langle x_{i,t-d}, x_{i,t-d+1}, ..., x_{i,t}, .., x_{i,t+d-1}, x_{i,t+d} \rangle.$$

h_w the window classifier is trained by converting each sequential training example (x_i, y_i) into windows and then applying a standard supervised learning algorithm. Then a new sequence x is classified by converting it to windows, applying h_w to predict each y_t, then concatenating the $y_t's$ to form the forecasting sequence of y.

3.3 Optimization Methods

Deep learning\ machine learning, is an iterative process. With so many parameters to tune or methods to try, it is important to be able to train models fast, in order to quickly complete the iterative cycle. This is key in increasing the speed and efficiency of a machine learning algorithms. Hence, the importance of optimization algorithms such as stochastic gradient descent, backpropagation (gradient descent), and the Adam optimizer, along others come of extreme importance. These methods make it possible for our neural network to learn with greater accuracy.

3.3.1 Stochastic gradient descent and generalizations

If when, each step is taken after training on only 1 data point, this process is called stochastic gradient descent. Recent studies have shown that this method is not very good, because it often takes steps in the wrong direction and it will not converge to the global minimum; it will instead oscillate around the global minimum. With objective function that has the form of a sum:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w),$$

where the parameter w that minimizes $Q(w)$ is to be estimated. Each summand function of Q_i is typically associated with the i_{th} observation in the data set (used for training).

3.3.2 Backpropagation (Gradient descent) and generalizations

Through supervised learning of an artificial neural networks using gradient descent. Its value of the neural network can be computed. (3.2). Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural networks weights. Then we let x be a vector of input values and $\hat{y} = f(x, W, b)$ is a vector of the output values. For every initial values $x^{(1)}, x^{(2)}, ..., x^{(n)}$ it is possible to compare experimental output $y^{(1)}, y^{(2)}, ..., y^{(n)}$ and the values given by the neural network

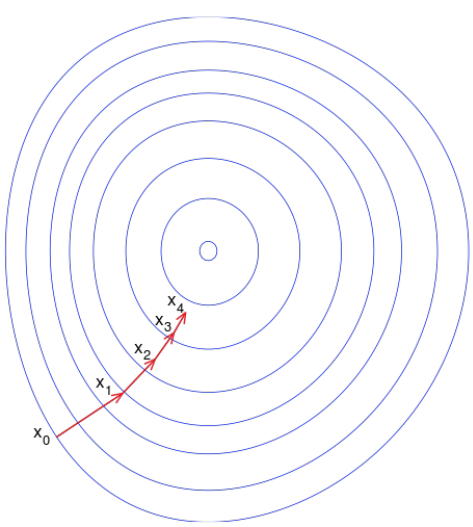


Figure 3.7: Graphical representation of the gradient decent method.

$\hat{y}^{(1)}, \hat{y}^{(2)}, ..., \hat{y}^{(n)}$. The goal of the learning process is minimization of the error.

$$E = \sum_{i=1}^{n} \|y^{(i)} - \hat{y}^{(i)}\|^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} (y_j^{(i)} - \hat{y}_j^{(i)})^2 \tag{3.3}$$

The gradient descent is the simplest case of finding minimum error.

We use the basic formulation and structure of the gradient descent,

$$x_{i+1} = x_i - \gamma \nabla E(x_i),$$

where γ is some constant.

By iterating through several passes and updating the weights in a direction that moves the total error between target, and predicted errors towards the local minimum error of the gradient surface. In presenter problem x is the vector of parameters (i.e. the weight matrix and the bias matrix). In order to find minimum of the error E, it is continent to find derivative with respect to the weight matrix W and bias vector b.

The Partial derivatives of error with respect to weight matrix are as follows,

$$\frac{\partial E}{\partial w_{i,j}} = \sum_{i=1}^{n}\sum_{j}^{m}\frac{\partial}{\partial w_{i,j}}(y_j^{(i)} - \hat{y}_j^{(i)})^2 = \sum_{i=1}^{n}\sum_{j}^{m}2(y_j^{(i)} - \hat{y}_j^{(i)})\frac{\partial}{\partial w_{i,j}}(y_j^{(i)} - \hat{y}_j^{(i)}) =$$

$$\frac{\partial E}{\partial w_{i,j}} = -\sum_{i=1}^{n}\sum_{j}^{m}2(y_j^{(i)} - \hat{y}_j^{(i)})\frac{\partial \hat{y}_j^{(i)}}{\partial w_{i,j}}.$$

Partial derivatives of error with respect to bias,

$$\frac{\partial E}{\partial b_i} = \sum_{i=1}^{n}\sum_{j}^{m}\frac{\partial}{\partial b_i}(y_j^{(i)} - \hat{y}_j^{(i)})^2 = \sum_{i=1}^{n}\sum_{j}^{m}2(y_j^{(i)} - \hat{y}_j^{(i)})\frac{\partial}{\partial b_i}(y_j^{(i)} - \hat{y}_j^{(i)}) =$$

$$\frac{\partial E}{\partial b_i} = -\sum_{i=1}^{n}\sum_{j}^{m}2(y_j^{(i)} - \hat{y}_j^{(i)})\frac{\partial \hat{y}_j^{(i)}}{\partial b_i}.$$

Derivatives $\frac{\partial \hat{y}_j^{(i)}}{\partial w_{i,j}}$, $\frac{\partial \hat{y}_j^{(i)}}{\partial b_i}$ can be found by using the formulas below. The output values y can be calculated by using the formula (3.1)

$$p^{(k+1)} = f^{(k)}(W^{(k)}p^{(k)} + b^{(k)}),$$

where $p^{(k+1)}$ are values in the layer $k+1$, $W^{(k)}$ is the weight matrix of the layer k, $b^{(k)}$ is a bias of the layer k, $p^{(k)}$ is an input value of the layer k, and $f^{(k)}$ is an activation function of the layer k.

$$\frac{\partial p^{(k+1)}}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}}f^{(k)}(W^{(k)}p^{(k)} + b^{(k)}) =$$

$$= f'^{(k)}(n^{(k)})\left(\frac{\partial W^{(k)}}{\partial w_{i,j}}p^{(k)} + W^{(k)}\frac{\partial p^{(k)}}{\partial w_{i,j}}\right),$$

where $n^{(k)} = W^{(k)}p^{(k)} + b^{(k)}$.

If $w_{i,j}$ is in $W^{(k)}$ then $\dfrac{\partial p^{(k)}}{\partial w_{i,j}} = 0$ and

$$\frac{\partial p^{(k+1)}}{\partial w_{i,j}} = f'^{(k)}(n^{(k)})\frac{\partial W^{(k)}}{\partial w_{i,j}}p^{(k)}.$$

If $w_{i,j}$ is not in $W^{(k)}$ then $\dfrac{\partial W^{(k)}}{\partial w_{i,j}} = 0$ and

$$\frac{\partial p^{(k+1)}}{\partial w_{i,j}} = f'^{(k)}(n^{(k)})W^{(k)}\frac{\partial p^{(k)}}{\partial w_{i,j}}.$$

If $w_{i,j}$ is in $W^{(k-1)}$ then $\dfrac{\partial p^{(k-1)}}{\partial w_{i,j}} = 0$ and

$$\frac{\partial p^{(k)}}{\partial w_{i,j}} = f'^{(k-1)}(n^{(k-1)})\frac{\partial W^{(k-1)}}{\partial w_{i,j}}p^{(k-1)},$$

then

$$\frac{\partial p^{(k+1)}}{\partial w_{i,j}} = f'^{(k)}(n^{(k)})W^{(k)}\frac{\partial p^{(k)}}{\partial w_{i,j}} = f'^{(k)}(n^{(k)})W^{(k)}f'^{(k-1)}(n^{(k-1)})\frac{\partial W^{(k-1)}}{\partial w_{i,j}}p^{(k-1)}.$$

If $w_{i,j}$ is not in $W^{(k-1)}$, then $\dfrac{\partial W^{(k-1)}}{\partial w_{i,j}} = 0$,

$$\frac{\partial p^{(k)}}{\partial w_{i,j}} = f'^{(k-1)}(n^{(k-1)})W^{(k-1)}\frac{\partial p^{(k-1)}}{\partial w_{i,j}}.$$

Now it is necessary to find the derivative $\dfrac{\partial p^{(k-1)}}{\partial w_{i,j}}$, by using similar calculations.

Similar calculations can be applied to be bias.

$$\frac{\partial p^{(k+1)}}{\partial b_i} = \frac{\partial}{\partial b_i}f^{(k)}(W^{(k)}p^{(k)} + b^{(k)}) =$$

$$= f'^{(k)}(n^{(k)})\left(W^{(k)}\frac{\partial p^{(k)}}{\partial b_i} + \frac{\partial b^{(k)}}{\partial b_i}\right),$$

where $n^{(k)} = W^{(k)}p^{(k)} + b^{(k)}$.
If b_i is in $b^{(k)}$ then $\dfrac{\partial p^{(k)}}{\partial b_i} = 0$ and

$$\frac{\partial p^{(k+1)}}{\partial b_i} = f'^{(k)}(n^{(k)})\frac{\partial b^{(k)}}{\partial b_i}.$$

If b_i is not in $b^{(k)}$ then $\dfrac{\partial b^{(k)}}{\partial b_i} = 0$ and

$$\frac{\partial p^{(k+1)}}{\partial b_i} = f'^{(k)}(n^{(k)})W^{(k)}\frac{\partial p^{(k)}}{\partial b_i}.$$

If b_i is in $b^{(k-1)}$ then $\dfrac{\partial p^{(k-1)}}{\partial b_i} = 0$ and

$$\frac{\partial p^{(k)}}{\partial b_i} = f'^{(k-1)}(n^{(k-1)})\frac{\partial b^{(k-1)}}{\partial b_i},$$

$$\frac{\partial p^{(k+1)}}{\partial b_i} = f'^{(k)}(n^{(k)})W^{(k)}\frac{\partial p^{(k)}}{\partial b_i} = f'^{(k)}(n^{(k)})W^{(k)}f'^{(k-1)}(n^{(k-1)})\frac{\partial b^{(k-1)}}{\partial b_i}.$$

If $b_{i,}$ is not in $b^{(k-1)}$ then $\dfrac{\partial b^{(k-1)}}{\partial b_i} = 0$ and

$$\frac{\partial p^{(k)}}{\partial b_i} = f'^{(k-1)}(n^{(k-1)})W^{(k-1)}\frac{\partial p^{(k-1)}}{\partial b_i}.$$

Similarly, we can calculate the derivative of $\dfrac{\partial p^{(k-1)}}{\partial b_i}$.

3.3.3 Least square method

To apply the ordinary least squares (OLS) method, we apply the below formula to find the equation:

$$m = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2},$$

$$b = \bar{y} - m * \bar{x}.$$

Where x = independent variables and $\bar{x}$ = average of independent variables, with y= dependent variables, and $\bar{y}$ = average of dependent variables. Lets solidify the basic concepts in least squares regression, suppose we have some simple dataset, $\{(x_i, y_i), i = 1, ..., n\}$, where x_i and y_i are real numbers. Say our model of y is related to x which is given by

$$y = f(x; w) + e,$$

$$f(x; w) = w^{'}\phi(x).$$

Where $\phi : \mathcal{R} \rightarrow \mathcal{R}^d$ is a specified function which maps to x to a d-dimensional feature vector, $\phi(x) = (\phi_1(x), ..., \phi_d(x))^{'}$; w is a d-dimensional parameter vector $w = (w_1, ..., w_d)^{'}$; e is the prediction error, which we do not model explicitly. We will use $w^{'}$ to denote the transpose of any vector w. In order to determine the least squared prediction error,

$$J(w) = \frac{1}{n}\sum_i (y_i - f(x_i; w))^2.$$

Solution to this problem is $\hat{w} = (X^{'}X)^{-1}X^{'}y$, where $X = (\phi(x_1), ..., \phi(x_n))^{'}$ is a $n \times d$ matrix whose first row is $\phi_1(x_1), ..., \phi_d(x_1)$ and the last row is given by $\phi_1(x_n, ..., \phi_d(x_n)$; The output vector y is defined as y is defined as $y = (y_1, y_2, ..., y_n)^{'}$. Assumption made on matrix $(X^{'}X)$ is invertible so that the problem is well-posed, (i.e. there exists a unique minimizer). This holds true for feature vectors $\phi(x_1), ..., \phi(x_n)$ associated with the training examples span the $d - dimensional$ feature space. As the feature vectors are long and the number of training points n is small. Now, for the estimate of $\hat{w}$ the resulting prediction errors $\hat{e}_i = y_i - f(x_i; \hat{w})$ should be "uncorrelated" with features:

$$\frac{1}{n}\sum_i \hat{e}_i \phi_k(x_i) = 0, k = 1, ..., d.$$

As these conditions are obtained by taking the derivative of $J(w)$ with respect to each w_i, $i = 1, .., d$, and setting them to zero. Make a note that the prediction error needs not to be zero mean unless one of the features is a constant, i.e., say $\phi_1 = 1$ for all x, so that

$$\frac{1}{n}\sum_i \hat{e}\phi_1(x_i) = \frac{1}{n}\sum_i \hat{e}_i = 0.$$

3.3.4 Adam optimizer

The Adam optimization algorithm is an extension to stochastic gradient descent, some of Adams advantages are that the magnitudes of parameter updates are invariant to rescaling of the gradient, its stepsizes are approximately bounded by the stepsize hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing. The procedure that has the adam optimizer at better performance is due to its update network weights iterative based in training data. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. (Diederik Kingma from OpenAI and Jimmy Ba from University of Toronto 2015). These authors describe Adam having two advantages over other extensions of stochastic gradient descent;

- **Adaptive Gradient Algorithm** (AdaGrad) which means that it maintains a per-parameter learning rate that improves the performance on the problems with sparse gradients. (e.g. natural language processing to most recent progress in the computer vision problems).

- **Root Mean Square Propagation** (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients form the weight (e.g. in how they quickly changing). Meaning that the algorithm does well on online and non-stationary problems (noisy). Adam also makes use of the average of the second moments of the gradients (the uncentered variance).

Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters β_1 and β_2 control the decay rates of these moving averages.

With the following requirements for Adam:

$$\alpha : Stepsize$$

$\beta_1, \beta_2 \in [0, 1)$, is the exponential decay rates for the moment estimates. $f(\theta)$:The stochastic objective function with parameters θ. With θ_0 : Initial parameter vector, then, $m_0 \overrightarrow{0}$ (Initialize 1^{st} moment vector), $v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector) and $t \leftarrow 0$ (Initialize timestep), $g_t \leftarrow \bigtriangledown_\theta f_t(\theta_{t-1})$, we get the gradients w.r.t. at timestep t. Then we have exponentially decaying average of past gradients m_t, similar to momentum. As m_t and v_t are initialized as vector of 0's, they are biased towards zero. Especially when the decay rates are small (i.e. β_1 and β_2 are close to 1).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \rightarrow$$

the first moment (the mean).

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \rightarrow$$

the second moment (the uncentered variance).

Adam: averages of gradient, or squared gradients. For bias correction we have the following:

$$m_t = (1 - \beta_1) f^{'}(\theta_t) + \beta_1 m_{t-1},$$

$$v_t = (1 - \beta_2) f^{'}(\theta_t)^2 + \beta_2 v_{t-1}$$

$$\hat{m}_t = \frac{m_t}{(1 - (1 - \beta_1)^t)}$$

$$\hat{v}_t = \frac{v_t}{(1-(1-\beta_2)^t)}.$$

Followed by the Adam update rule,

$$p_t = \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}},$$

$$\theta_{t+1} = \theta_t - p_t.$$

By counteracting these biases in Adam we have the following, $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ and $\hat{v} = \frac{v_t}{1-\beta_2^t}$, *Adam* $\theta_{T+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$. Using large models and data sets, we demonstrate Adam can efficiently solve practical deep learning problems.

3.3.5 Epochs

In terms of artificial neural networks (AI), an epoch refers to one cycle through the full training data set. Its a point of time to mark the important and noteworthy period in the history. It also means the marking of time period that brought out lot of new developments and change usually, training a neural network takes more than a few epochs. In other words, feeding a neural network the training data for more than one epoch in different patterns, we hope for a better generalization when given a new "unseen" input (test data). The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training data set. One epoch means that each sample in the training data set has had an opportunity to update the internal model parameters. It may not make much sense in the starting that passing the entire data set through a neural network once is not enough. And we do need to pass the full data set multiple times to the same neural network to improve learning process. But lets keep in mind that we are using a limited data set and to optimise the learning and the graph we are using Gradient Descent which is an iterative process. So, updating the weights with single pass or one epoch is not enough. For hyperparameters on log scale, we have for example, a typical sampling

of the learning rate would look as follows : $learning\_rate = 10 * *uniform(-6, 1)$. That is, we are generating a random number from a uniform distribution, but then raising it to the power of 10. The number of epochs is traditionally large, often hundreds or thousands, allowing the learning algorithm to run until the error from the model has been sufficiently minimized. You may see examples of the number of epochs in the literature and in tutorials set to 10, 100, 500, 1000, and larger.

3.4 Activation functions with Neural Networks

Lets quickly review what usage of an activation functions has along with neural networks, for a brief definition of what an activation function is, Its just a thing function that you use to get the output of node. It is also known as Transfer Function. This comes of importance because it is used to determine the output of neural network, like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).

$$Y = \sum(weight * input) + bias,$$

The value of Y can be anything ranging from $-\infty$ to $+\infty$. The neuron really doesn't know the bounds of the value.

3.4.1 Sigmoid or Logistic Activation Function

The Sigmoid Function curve looks like a S-shape.

The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

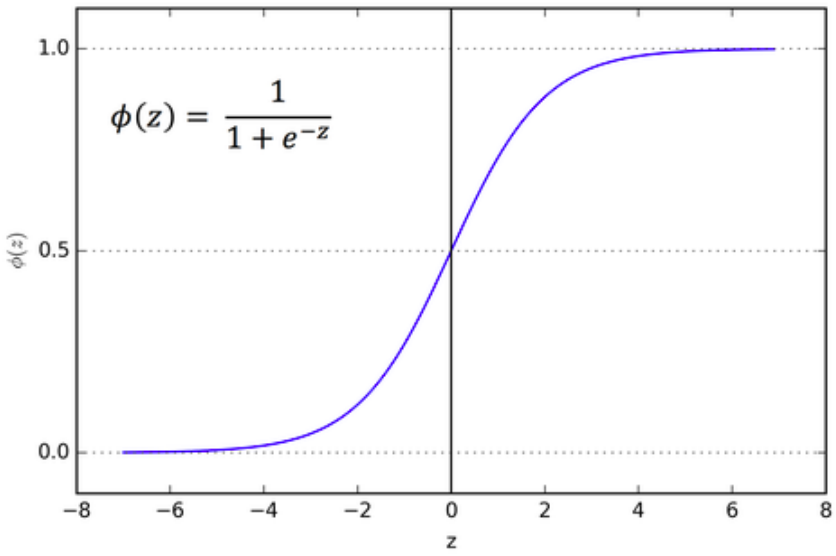


Figure 3.8: Sigmoid Function

3.4.2 Tanh or Hyperbolic tangent Activation function

tanh is also like logistic sigmoid but studies have shown it to be better, the range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped). The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph. The function is differentiable, and the function is also monotonic while its derivative is not monotonic. The tanh function is mainly used classification between two classes.

3.4.3 ReLu (Rectified Linear Unit) Activation Function

The ReLU is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning methodologies, because a model that uses it is easier to train and often achieves better performance. A node or unit that implements this activation function is referred to as a rectified linear activation unit, or ReLU for short. Often, networks that use the rectifier function for the hidden layers

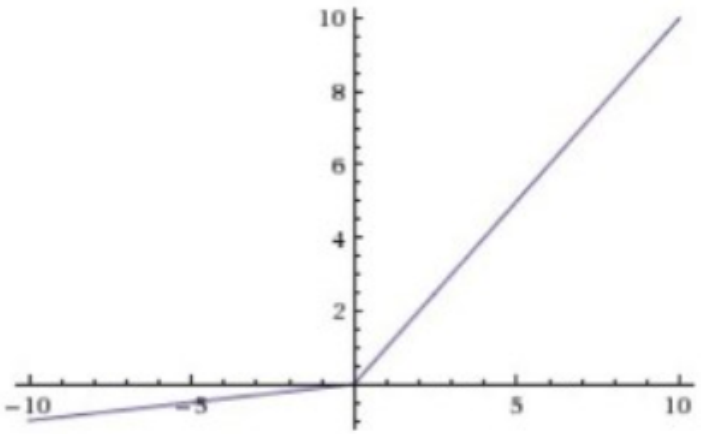


Figure 3.9: LReLU function

are referred to as rectified networks. The rectified linear activation function is a simple calculation that returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less.

3.4.4 LReLU Activation function

Leaky ReLU is a modification of ReLU which replaces the zero part of the domain in $[-\infty, 0]$ by a low slope, as shown in figure and formula below. The motivation for using LReLU instead of ReLU is that constant zero gradients can also result in slow learning, as when a saturated neuron uses a sigmoid activation function. Furthermore, some of them may not even activate.

$$f(x) = \begin{cases} 0.01x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \qquad f'(x) = \begin{cases} 0.01 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$$

3.5 Supervised Learning

The majority of practical machine learning (ML) uses supervised learning. Supervised learning is where you have input variables (X) and an output variable (y) and use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X).$$

The intention of supervised machine learning is to build a model that makes predictions based on evidence in the presence of uncertainty. A supervised learning algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions for the response to new data, often we find the usage of classification and regression techniques to develop predictive models. A concise explanation of classification techniques to foresee discrete responses, for example; whether an email is genuine or spam, or whether a tumor is cancerous or benign. Classification models classify input data into categories. Typical applications include medical imaging, speech recognition, and credit scoring. [11] Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

The goal is to approximate the real underlying mapping so well that when you have new input data (X), you can predict the output variables (y) for the data. To have a quick look of the process of supervised learning data set here is a contrived example. where each row is an observation comprised of one input variable (X) and one output variable to be predicted (y).

The process of an algorithm learning from the training data set can be thought of as a teacher supervising the learning process. Recall that we know the correct answers, the

Table 3.1: Sample data

1	X;time	y;measure
2	1	100
3	2	110
4	3	108
5	4	115
6	5	120

algorithm iteratively makes predications on the training data and is corrected by making updates. The learning stops when the algorithm achieves an acceptable level of performance. Furthermore, supervised learning problems can be further grouped into regression and classification problems.

1. *Classification*: A classification problem is when the output variable is category, such as red and blue or disease and no disease to binary representation of 1 (yes) , 0 (no).

2. *Regression:* A regression problem is when the output variable is a real value, such as dollars or weight. The contrived example above is a regression problem.

Now lets look at the sequence of numbers from pervious table, given a sequence of numbers for a time series data set we can restructure the data to look like a supervised learning problem. We may do this by using pervious time steps as input variables and use the next time step as the output variable. As our pervious table shows that we can restructure this time series data set as a supervised learning problem by using the value at the previous time step to predict the value at the next time-step. Re-organizing the time series data set as this manner, the data will look as follows:

Before moving on, some observation to be made on the transformation of the data set:

- We can see that the previous time step is input (X) and the next time step is the output (y) in our supervised learning problem.
- The order between the observations is preserved, and must continue to be preserved when using this data set to train a supervised model.

Table 3.2: Sample data with missing values

1	X	y
2	?	100
3	100	110
4	110	108
5	108	115
6	115	120
7	120	?

- Notice we have no pervious value that we can use to predict the first value in the sequence, then we will delete this row as we cannot use it.

- Finally, observe that we do not have a known next value to predict for the last value in the sequence. We may want to delete this value while training our supervised model. Note that we still use pervious data we simply preserve for training propose.

The use of using prior time steps to forecast the next time step is called in some literature sliding, and in statistics and time series analysis, this is called a lag or lag method. Which was perviously stated used as a forecasting method. The number of pervious time steps are called the lag value. This is the bias for how we can turn any time series data set into a supervised learning problem. Furthermore, we must notice a few more things:

- This work can turn a time series into either a regression or classification supervised learning problem for real-valued or labeled time series values.

- Once a time series data set is prepared in this manner, any of the standard linear and nonlinear machine algorithms may be applied, as long as the order of the rows is preserved. - The lag value can be increased to include more previous time steps. - The lag method can be used on a time series that has more than one value, or so-called multivariate time series.

3.5.1 Unsupervised Learning

Unsupervised learning is concerned with finding patterns and structure in unlabeled data. Examples of unsupervised learning include clustering, dimensionality reduction, and generative modeling. The unsupervised machine learning algorithms infer to patterns from a dataset without reference to known, or labeled, outcomes. Unlike supervised machine learning, unsupervised machine learning methods cannot be directly applied to a regression or a classification problem because you have no idea what the values for the output data might be, making it impossible for you to train the algorithm the way you normally would. Unsupervised leaning can instead be used for discovering the underlying structure of the data. There are some useful application of unsupervised machine learning algorithms which learning purports to uncover previously unknown patterns to data, but most of the time these patterns are poor approximations of what supervised machine learning can achieve. Additionally, since you do not know what the outcomes should be, there is no way to determine how accurate they are, making supervised machine learning more applicable to real-world problems. Leading to having supervised learning is the optimal technique.

We now find ourselves with gathered information what in application to methodologies applied to our stock market and educational datasets. Now we move forward on exploring the structure of the methods used on our datasets via artificial neural networks.

3.6 Data Splitting

One of the main requirements in Machine learning is to build computational models with a high ability to generalize well the extracted knowledge. The learning goal, the ability to predict depends on the number of data points we have, the 'noise' in the data, and our knowledge about relevant features. The reason we must divide our data into a training and test dataset is that the point of machine learning is to make accurate predictions about new data we have not seen. We can guard against overfitting in two ways: we can use less expressive models with fewer parameters, or we can collect more data so that the

likelihood that the noise appears patterned decreases. Indeed, when we increase the size of the training data set.

The simpler model has more bias but is less dependent on the particular realization of the When training (e.g. artificial neural networks), poor generalization is often characterized by over-training. A common method is to avoid over-training then one holds-out cross-validation. In most of the applications, simple random sampling is used. Nevertheless, there are several sophisticated statistical sampling methods suitable for various types of datasets. In the case of supervised learning, a computational model is trained to predict outputs of an unknown target function. The target function is represented by a finite training dataset T of examples of inputs and the corresponding desired outputs: $T = [\ x1,\ d1], ..., [\ xn,\ dn]$, where $n > 0$ is the number of ordered pairs of input/output samples (patterns). At the end of the training process, the final model should predict correct outputs for the input samples from T, but it should also be able to generalize well to previously unseen data. Cross-validation techniques [Refaeilzadeh et al., 2009; Picard and Cook, 1984] belong to conventional approaches used to ensure good generalization and to avoid over-training. K-fold cross-validation maximizes the use of the data. The basic idea is to divide the dataset T into two subsets one subset is used for training while the other subset is left out and the performance of the final model is evaluated on it. K-fold divides data randomly into k folds (subsets) of equal size, then we train the model $k-1$ folds (i.e. use one fold for testing). Repeat this process k times so that all folds are used for testing, which then we compute the average performance on the k test sets. This effectively uses all the data for both training and testing, typically $k = 10$ is used.

The problem of appropriate data splitting can be handled as a statistical sampling problem. These sampling methods can be divided into the following categories based on their principles, goals and algorithmic and computational complexity:

- Simple random sampling (SRS)

- Trial-and error methods

- Stratified sampling

- Systematic sampling

... and others.

Some of the methods are simple and widely used, although they suffer from high variance of the model performance. For classification problems, we we measure the performance of a model in terms of its error rate: percentage of incorrectly classified instances in the data set. Building a model, we want to use it to classify new data. Hence we are chiefly interested in model performance on new (unseen) data. The re-substitution error (error rate on the training set) is a bad predictor of performance on new data. The model was build to account for the training data, so might overfit it, i.e., not generalize to unseen data. Moveover, the more data available, more training which equals better model. The more test data, the more accurate the error estimate.

3.7 Classification method

Machine learning is one of the major ways of classifying the myriad sciences, studies how to automatically learn to make accurate predictions based on past observations. As we have mention before raw data will likely be seldom complete. Knowing that Artificial Neural Networks require complete set of data for an accurate classification, after imputation and data completion we may move forward on classifying data.

Classification methods based on Neural Networks, take a single neuron (processing element- PE) with inputs and outputs. Once more we have a set of training observation $(x_1, y_1), ..., (x_n, y_n)$ that we can use to build a classifier. With classifying examples into given sets of categories.

Examples of Classification problems we have the following:

- fradu detection

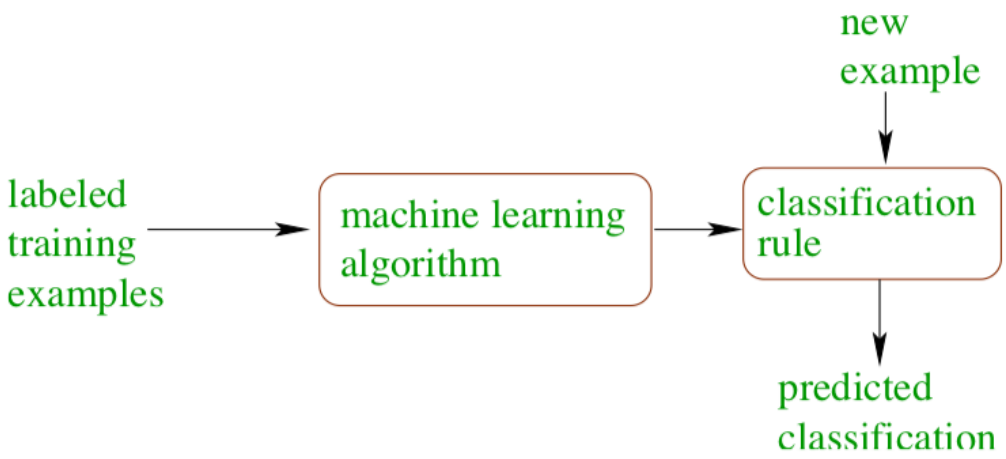


Figure 3.10: Classification.

- optical character recognition
- machine vision (e.g., face detection)
- market segmentation (e.g., predict if customer will respond to promotion)
- bioinformatics (e.g., classify proteins according to their function)

... along with others.

Classification is the most widely used Machine learning technique that involves separating the data into different segments which are non-overlapping. Hence classification is the process of finding a set of models that describe and distinguish class label of the data object. Classification can be performed on structured or unstructured data. Classification is a technique where we categorize data into a given number of classes. The main goal of a classification problem is to identify the category/class to which a new data will fall under. Building an accurate classifier is always what we are working towards, for good test performance, we need 1) enough training examples, 2) good performance on training set and 3) classifier that is not "too complex" (Occams razor). With classifiers having to be "as simple as possible, but no simpler". We have "simplicity" closely related to prior expecta-

tion. Before moving forward a few terminologies needed when dealing with classification method.

- Classier: An algorithm that maps the input data to specific category.
- Classification model: A classification model tries to draw some conclusion from the input values given for training. It will predict the class labels/categories for the new data.
- Feature: A feature is an individual measurable property of a phenomenon being observed.
- Binary Classification: Classification task with two possible outcomes. Eg: Gender classification(Male/Female)
- Multiclass classification: Classification with more than two classes. In multi class classification each sample is assigned to one and only one target label. eg: An animal can be cat or dog but not both at the same time
- train the classifier: All classifiers in sciit-learn uses a fit (X, y) method to fit the model(training) for the given train data X and train label y.
- Predict the target: Given an unlabeled observation X, the predict(X) returns the predicted label y.

3.7.1 Linear Regression

Let us first start with the supervised learning via **Linear Regression** for classification method. Start off with stating *training data:* $\{(x_1, g_1), (x_2, g_2), ..., (n_N, g_N)\}$, with the feature vector $X = (X_1, X_2, ..., X_p)$,where each variable X_j is quantitative. The response variable G is categorical. $G \in \mathcal{G} = \{1, 2, ..., K\}$, from a predictor $G(x)$ to predict G based on X. For a quick simple and most commonly used example we have email spam G has

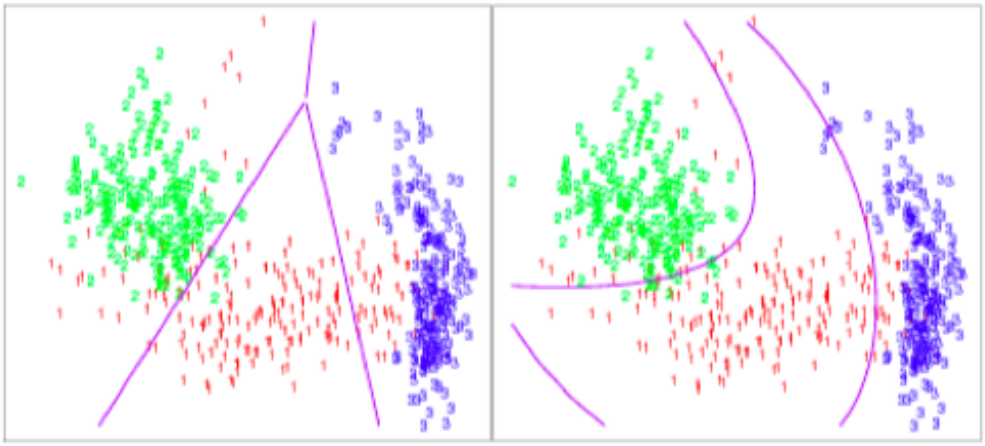


Figure 3.11: Linear Regression Classification problem.

only two values, say 1 denoting a useful email and 2 denoting a junk email. X is a 57-dimensional vector, each element being the relative frequency of a word or a punctuation mark. $G(x)$ divides the input space (feature vector space) into a collection of regions, each labeled by one class.

On the figure above the left plot shows some data from three classes, with linear decision boundaries found by linear discriminant analysis. The right plot shows quadratic decision boundaries. These boundaries were obtained by fining linear boundaries in the five-dimensional space $X_1, X_2, X_{12}, X_1^2, X_2^2$. Linear inequalities in this space are quadratic inequalities in the original space. For linear methods we have two class problem, the decision boundary between the two classes is a hyperplane in the feature vector space. A hyperplane in the p dimensional input space is the set:

$$\left\{ x : \alpha_0 + \sum_{j=1}^{p} \alpha_j x_j + 0 \right\}.$$

Table 3.3: Logistic regression

	notation	range equivalents
standard probability	p	0, 0.5, 1
odds	p/q	0, 1, $+\infty$
log odds (logit)	log(p/q)	$(-\infty, 0, +\infty)$

The two regions separated by a hyperplane:

$$\left\{ x : \alpha_0 + \sum_{j=1}^{p} \alpha_j x_j > 0 \right\},$$

and

$$\left\{ x : \alpha_0 + \sum_{j=1}^{p} \alpha_j x_j < 0 \right\}.$$

3.7.2 Logistic Regression

As for the **Logistic regression** we have the type of regression model that is used for predicting the result of categorical (a variable that can have a limited number of categories) dependent variable based on one or more predictor variables. Name is somewhat misleading The Logistic regression model is used to determine the impact of multiple independent variables presented simultaneously to predict membership of one or other of the two dependent variable categories. Name is somewhat misleading, but it is really a technique for classification, not regression. The "Regression" comes from the fact that we fit a linear model to the feature space. Which involves a more probabilistic view of classification. Quickly lets go over the different ways of expressing probability, lets consider a two-outcome probability space, where:

$$p(O_1) = p,$$

$$p(O_2) = 1 - p = q.$$

We can express probability of O_1 as:

Then we have the following functions from probability to log odds,

logit function:

$$z = log\left(\frac{p}{1-p}\right),$$

logistic function:

$$p = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}.$$

Using a logistic regression model we have a model which consists of a vector β in d-dimensional feature space. For point x in feature space, we have β to convert it into a real number z in the range $-\infty$ to $+\infty$ then,

$$z = \alpha + \beta \cdot x = \alpha + \beta_1 x_1 + ... + \beta_d x_d.$$

We map z to the range 0 to 1 using the logistic function $p = \frac{1}{(1+e^{-z})}$. Training a logistic regression model we take fourth the need to optimize β so the model gives the best possible reproduction of training set labels. This is usually done by numerical approximation of maximum likelihood, and on really large dataset, we may use the gradient descent. As we can see the logistic regression can be considered a special case of linear regression models. A logistic regression model specifies that an appropriate function of the fitted probability of the event is a linear function of the observed values of the available explanatory variables. The Logistic regression model has one major advantage of being able to produce a simple probabilistic formula of classification, on the contrary a weakness is that it cannot properly deal with problems of non-linear and interactive effects of explanatory variables.

For instance, a sophisticated machine learning program could classify flowers based on photographs. Our aspiration is more modest, we're going to classify an example of a Iris flowers based on the length and width measurements of their sepals and petals. The Iris genus entails about 300 species, but our program will only classify the following three:

- Iris Setosa

- Iris Virginica
- Iris Versicolor

This data set, iris_training.csv, is a plain text file that stores tabular data formatted as comma-separated values (CSV). Use the head -n5 command to take a peak at the first five entries:

```
120,4,setosa,versicolor,virginica
6.4,2.8,5.6,2.2,2
5.0,2.3,3.3,1.0,1
4.9,2.5,4.5,1.7,2
4.9,3.1,1.5,0.1,0
```

From this view of the data set, we notice the following:

1. The first line is a header containing information about the data set:
    - There are 120 total examples. Each example has four features and one of three possible label names.
2. Subsequent rows are data records, one example per-line, where:
    - The first four fields are features: these are characteristics of an example. Here, the fields hold float numbers representing flower measurements.
    - The last column is the label: this is the value we want to predict. For this data set we have an integer value of 0, 1, or 2 that corresponds to a flower name.

Each label is associated with string name (for example, "setosa"), but machine learning typically relies on numeric values (binary). Here we find ourselves with more than two variables to classify. The label numbers are mapped to a named representation, such as:

- 0: Iris Setosa

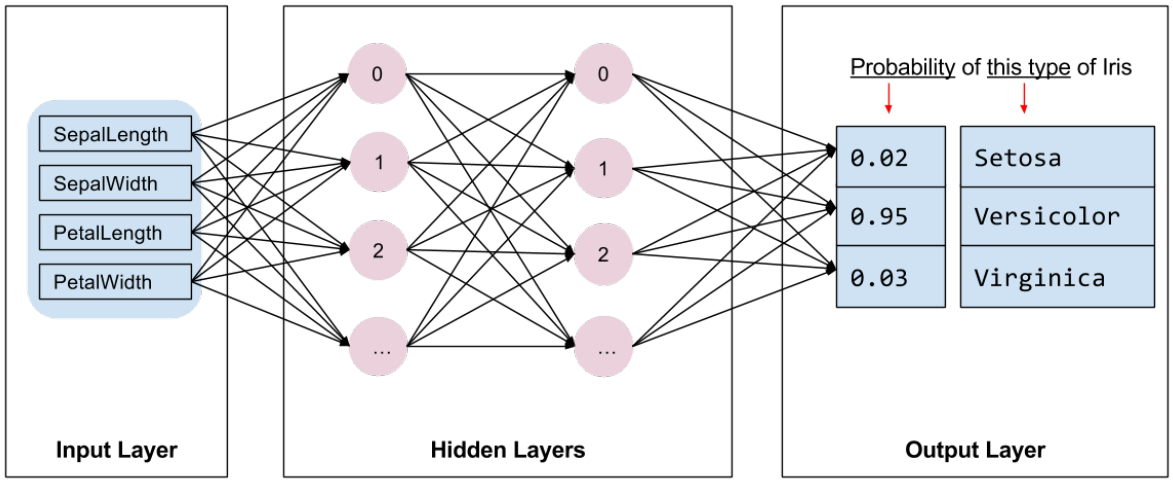


Figure 3.12: A neural network with features, hidden layers, and predictions.

- 1: Iris Versicolor
- 2: Iris Virginica

We need to select such algorithm that we may model and train. There are countless of algorithms that machine learning has made various designs for recognizing patterns to make intelligent decision for the input data. The major challenge for machine learning technique is behavior of inputs which should be trained during observed examples. Hence they are trained with all possible inputs to produce efficient and sensitive output. In this case it is possible to use a neural network to solve the Iris classification problem. Neural networks can find complex relationships between features and the label. It is a highly-structured graph, organized into one or more hidden layers. Each hidden layer consists of one or more neurons. There are several categories of neural networks and this program uses a dense network, or fully-connected neural network: the neurons in one layer receive input connections from every neuron in the previous layer. For example, Figure 3.2 3.7.2 illustrates a dense neural network consisting of an input layer, two hidden layers, and an output layer [12]:

Python code based on Tensor-flow may be found in *Appendix A*.

3.8 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) most popular neural network model being used for image classification problem. The Convolutional Neural Network is a feed-forward neural network and like the traditional architecture of a neural network including input layers, hidden layers and output layers, convolutional neural network also contains these features and the input of the layer of convolution are the output of the previous layer of convolution or pooling. Of course, they still have some unique features such as pooling layers, full connection layers, etc. The number of hidden layers in a CNN is more than that in a traditional neural network, which to some extent, shows that the capability of the neural network. The more the hidden layers are, the higher feature it can extract and recognize from the input. With the activation function and conditioning is performed by applying multiple convolutional filters in parallel to separate time series which allows for the fast processing of data and the exploitation of the correlation structure between the multivariate time series. The practical benefit is that having fewer parameters greatly improves the time it takes to learn as well as reduces the amount of data required to train the model. Instead of a fully connected network of weights from each pixel, a CNN has just enough weights to look at a small patch of the image, which allows for deeper learning process. Its like reading a book by using a magnifying glass; eventually, you read the whole page, but you look at only a small patch of the page at any given time. Convolutional neural networks were developed with the idea of local connectivity. Each node is connected only to a local region in the input. The spatial extent of this connectivity is referred to as the receptive field of the node. The local connectivity is achieved by replacing the weighted sums from the neural network with convolutions. In each layer of the convolutional neural network, the input is convolved with the weight matrix (also called the filter) to create a feature map. In other words, the weight matrix slides over the input and computes the dot product between the input and the weight matrix. Note that as opposed to regular neural networks, all the values in the output feature map share the same weights. This means that

all the nodes in the output detect exactly the same pattern. The local connectivity and shared weights aspect of CNNs reduces the total number of learnable parameters resulting in more efficient training. The intuition behind a convolutional neural network is thus to learn in each layer a weight matrix that will be able to extract the necessary, translation-invariant features from the input.

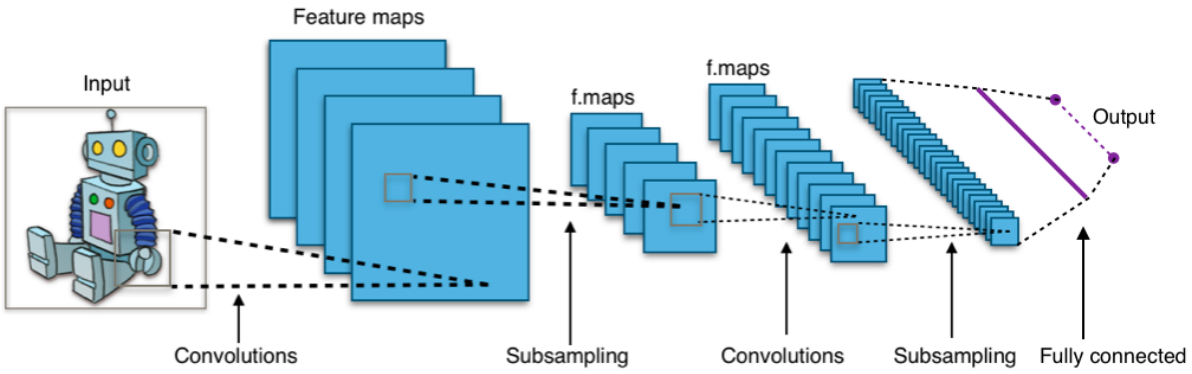


Figure 3.13: Typical CNN architecture.

CNNs are regularized versions of multilayer perceptrons, the multilayer perceptrons usually refer to fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks make them prone to overfitting data. Typical ways of regularization includes adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

Lets consider a 256×256 image, CNN can efficiently scan it chunk by chunk, say

a 5×5 window. The 5×5 window slides along the image (usually left to right, and top to bottom), as shown below. Example, a stride length of 2 means the 5×5 sliding window moves by 2 pixels at a time until it spans the entire image. A convolution is a weighted sum of the pixel values of the image, as the window slides across the whole image. Turns out, this convolution process throughout an image with a weight matrix produces another image (of the same size, depending on the convention). Convolving is the process of applying a convolution, as the sliding-window methodology happens in the convolution layer of the neural network. A typical CNN has multiple convolution layers. Each convolutional layer typically generates many alternate convolutions, so the weight matrix is a tensor of $5 \times 5 \times n$, where n is the number of convolutions. For example, say we have an image which goes through a convolution layer on a weight matrix of $5 \times 5 \times 64$. It then generates 64- convolutions by sliding a 5×5 window. Therefore, this model has $5 \times 5 \times 64 (= 1,600)$ parameter, which is remarkably fewer parameters than a fully connected network, $256 \times 256 = 65,536$.

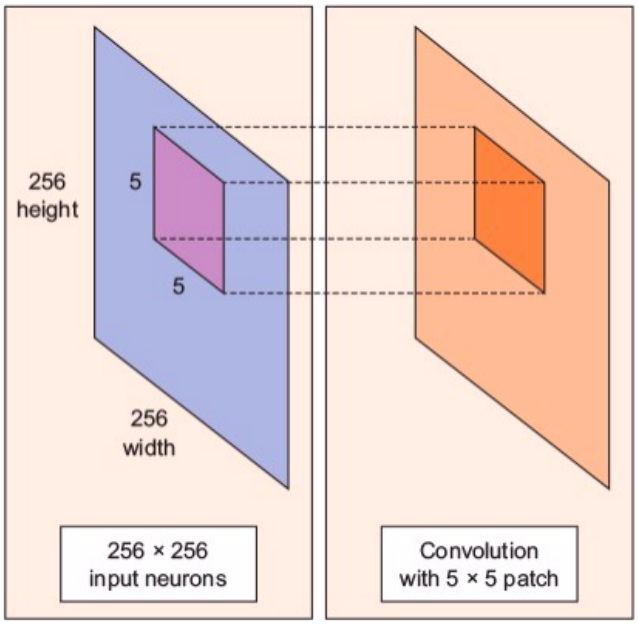


Figure 3.14: CNN classifier.

This leads us to the best part of the CNN, which is that the number of parameters is independent of the size of the original image. You can run the same CNN on a 300×300 image, and the number of parameters will not change in the convolution layer. The CNN model will learn a function that maps a sequence of past observations as input to an output observation. As such, the sequence of observations must be transformed into multiple examples from which the model can learn. On a later chapter the application of the CNN will be displayed on our data.

3.9 Recurrent Neural Network (RNN)

A **Recurrent neural network** (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence, they are networks with loops in them, allowing information to persist. This allows it to exhibit temporal dynamic behavior. Unlike feedforward (memeoryless) neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. The main advantage of Recurrent Neural Networks is that RNN can model sequence of data (i.e. time series) so that each sample can be assumed to be dependent on previous ones. The next value in the series is dependent on the previous value in the series. As the Recurrent Neural Network remembers the past and its decisions are influenced by what it has learned from the past. RNNs learn similarly while training, in addition, they remember things learnt from prior input(s) while generating output(s), its part of the network. The RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular NN, but also by a hidden (layer) state vector representing the context based on prior input(s)/output(s). So, the same input could produce a different output depending on previous inputs in the series. As shown on the figure below, a Recurrent Neural Network, with a hidden state that is meant to carry pertinent information from one input item in the series to others.

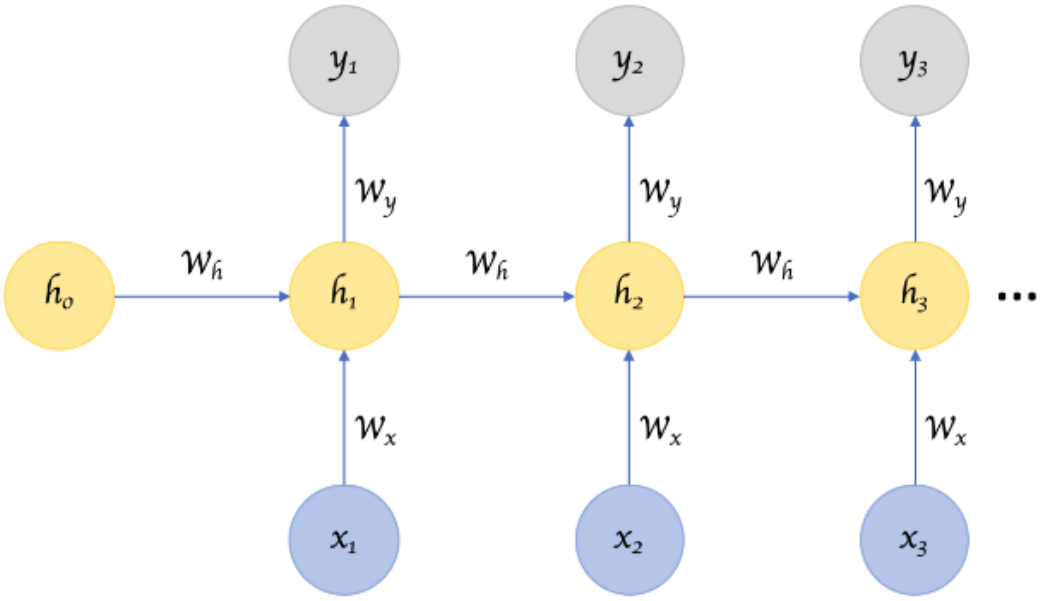


Figure 3.15: Figure 3.7: Simple Recurrent Neural Network

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data. In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning. The list goes on. For simplicity lets start of with Elman and Jordan networks known as "simple recurrent networks" (SRN). That can be represented by diagram above.

Elman network:

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y). \end{aligned}$$

Jordan network:

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h y_{t-1} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y). \end{aligned}$$

RNNs are very powerful, because they combine two properties: - Distributed hidden state that allows them to store a lot of information about the past efficiently. - Non-linear

dynamics that allows them to update their hidden state in complicated ways. The core reason that recurrent nets are more exciting is that they allow us to operate over sequences of vectors: Sequences in the input, the output, or in the most general case both. The type of artificial neural network that the RNNs comes along with adds additional weights to the network to create cycles in the network graph in an effort to maintain an internal state. The promise of adding state to neural networks is that they will be able to explicitly learn and exploit context in sequence prediction problems, such as problems with an order or temporal component.

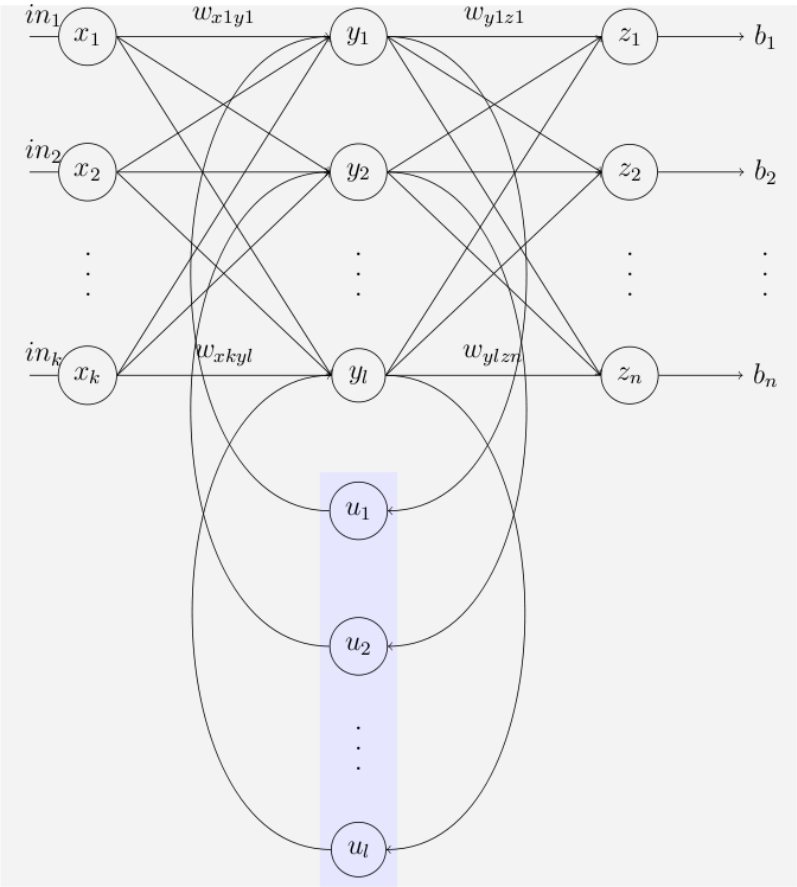


Figure 3.16: Recurrent neural network

Recurrent neural networks are deterministic. So think of the hidden state of an RNN as the equivalent of the deterministic probability distribution over hidden states in a linear

dynamical system or hidden Markov model. Recall that the Hidden Markov Models have a discrete one of-N hidden state. RNNs could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects. Furthermore, with a RNN we can process a sequence of vectors x by applying a recurrence formula at every time step denoted by: $h_t = f_W(h_{t-1}, x_t)$, with h_t = new state, f_W = some function with parameters W, h_{t-1}= old state, and x_t= input vector at some time step.

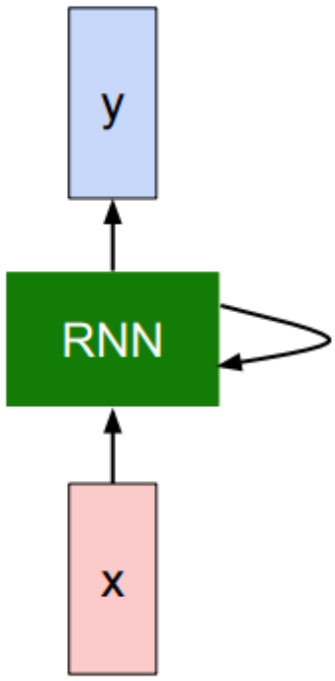


Figure 3.17: State consists of a single "hidden vector" h

Then we have the following RNN format:

$$h_t = f_W(h_{t-1}, x_t)$$

$$\downarrow$$

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t),$$

$$y_t = W_{hy}h_t.$$

Chapter 4

Forecasting of Data sets

4.1 Classification of Credit Card Default

For a brief introduction to our data set, regarding the default of credit card problem, this case of information of customers default payments come from Taiwan. We will us default binary result of classification - credible or not credible clients. Our indicates that payment date was on October of 2005; Taiwan bank collected a cash and credit card issuer. There is a total of 25,000 observations, 5529 observations (22.12%) are the card holders with default payment. The data set will be set employed as a binary variable, default payment (Yes = 1, No=0), as the response variable. Within our data set we have used the following of 23 variables as explanatory variables denoted as:

- X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
- X2: Gender (1 = male; 2 = female).
- Education (1 = graduate school; 2 = university; 3= high school; 4 = others).
- X4: Marital status (1 = married; 2 = single; 3 = others).
- X5: Age (year).
- X6-X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005 ; . . .; X11 = the repayment

status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.

- X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.

- X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .; X23 = amount paid in April, 2005.

The delinquency crisis of credit card debt increased in recent years in Taiwan. As observed on variables from data were collected year of 2005, based on Taiwan recent studies they were expecting a peak in the third quarter of year 2006 (crisis on credit card debt) [13]. In order for Taiwan to increase their market share, the card-issuing banks of Taiwan over issued credit cards to unqualified candidates. Another factor that would have to be dealt with is the fact that most card holders, irrespective of their repayment ability and lead to an overused credit card, this accumulates to a heavy credit card and cash debts. This crisis (crash) leads to major importance to business and banks, on having the ability predict customers' credit risk, and reduce the damage and uncertainty. Such methods are describe to be statistical methods, which are used to classifying applicants for credit into "good" and "bad" classes. Throughout the growth of Artificial Intelligence and machine learning these types of models/methods have become increasingly important with the dramatic growth in consumer credit in the past years.

Moveover, for the default data, we use the logistic model by defaulting on the credit card debt. For example, the probability of default given *balance* may be written as:

$$P_r(default = Yes|balance),$$

we will denote for not the above probability by $p(balance)$ for convenience. Moving on, we now see how the modeling in relationship between X and Y with gathered information from pervious chapters. As for the binary classification problem,

$$p(X) = Pr(Y = 1|X)$$

for X, recall perviously stated that the linear regression model

$$p(X) = \beta_0 + \beta_1 X,$$

where the goal is to use the default = Yes to predict *balance.*
For a quick intuition on the modeling perspective, we copy $p(X)$ using a function that gives outputs between 0 and 1 for all values of X, where

$$p(X) = \beta_0 + \beta_1 X.$$

We use the logistic function,

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}.} \tag{4.1}$$

To fit a model in equation (4.1) we use least square method, after some manipulation of equation (4.1),

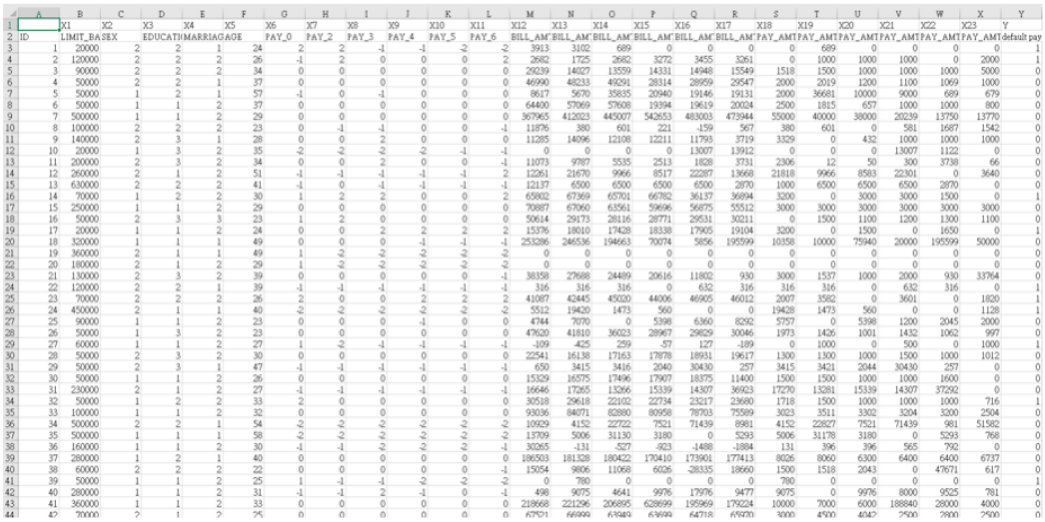
$$\frac{p(X)}{1 - p(X)} = e^{\beta_0+\beta_1 X} \tag{4.2}$$

Taking the log of equation (4.2) gives us,

$$log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X,$$

where the left hand side is called the log-odds or logit. With further production estimating the coefficients we note that:

- β_0 and β_1 in equation (4.1) are unknown and must be estimated using the training



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1		X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	Y
2	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default pay
3	1	20000	2	2	1	24	2	2	-1	-1	-2	-2	3913	3102	689	0	0	0	0	689	0	0	0	0	1
4	2	120000	2	2	2	26	-1	2	0	0	0	2	2682	1725	2682	3272	3455	3261	0	1000	1000	1000	0	2000	1
5	3	90000	2	2	2	34	0	0	0	0	0	0	29239	14027	13559	14331	14948	15549	1518	1500	1000	1000	1000	5000	0
6	4	50000	2	2	1	37	0	0	0	0	0	0	46990	48233	49291	28314	28959	29547	2000	2019	1200	1100	1069	1000	0
7	5	50000	1	2	1	57	-1	0	-1	0	0	0	8617	5670	35835	20940	19146	19131	2000	36681	10000	9000	689	679	0
8	6	50000	1	1	2	37	0	0	0	0	0	0	64400	57069	57608	19394	19619	20024	2500	1815	657	1000	1000	800	0
9	7	500000	1	1	2	29	0	0	0	0	0	0	367965	412023	445007	542653	483003	473944	55000	40000	38000	20239	13750	13770	0
10	8	100000	2	2	2	23	0	-1	-1	0	0	-1	11876	380	601	221	-159	567	380	601	0	581	1687	1542	0
11	9	140000	2	3	1	28	0	0	2	0	0	0	11285	14096	12108	12211	11793	3719	3329	0	432	1000	1000	1000	0
12	10	20000	1	3	2	35	-2	-2	-2	-2	-1	-1	-1	0	0	0	13007	13912	0	0	0	13007	1122	0	0
13	11	200000	2	3	2	34	0	0	2	0	0	-1	11073	9787	5535	2513	1828	3731	2306	12	50	300	3738	66	0
14	12	260000	2	1	2	51	-1	-1	-1	-1	-1	2	12261	21670	9966	8517	22287	13668	21818	9966	8583	22301	0	3640	0
15	13	630000	2	2	2	41	-1	0	-1	-1	-1	-1	12137	6500	6500	6500	6500	2870	1000	6500	6500	6500	2870	0	0
16	14	70000	1	2	2	30	1	2	2	0	0	2	65802	67369	65701	66782	36137	36894	3200	0	3000	3000	1500	0	1
17	15	250000	1	1	2	29	0	0	0	0	0	0	70887	67060	63561	59696	56875	55512	3000	3000	3000	3000	3000	3000	0
18	16	50000	2	3	3	23	1	2	0	0	0	0	50614	29173	28116	28771	29531	30211	0	1500	1100	1200	1300	1100	0
19	17	20000	1	1	2	24	0	0	2	2	2	2	15376	18010	17428	18338	17905	19104	3200	0	1500	0	1650	0	1
20	18	320000	1	1	1	49	0	0	0	-1	-1	-1	253286	246536	194663	70074	5856	195599	10358	10000	75940	20000	195599	50000	0
21	19	360000	2	1	1	49	1	-2	-2	-2	-2	-2	0	0	0	0	0	0	0	0	0	0	0	0	0
22	20	180000	2	1	2	29	1	-2	-2	-2	-2	-2	0	0	0	0	0	0	0	0	0	0	0	0	0
23	21	130000	2	3	2	39	0	0	0	0	0	-1	38358	27688	24489	20616	11802	930	3000	1537	1000	2000	930	33764	0
24	22	120000	2	2	1	39	-1	-1	-1	-1	-1	-1	316	316	316	0	632	316	316	316	0	632	316	0	1
25	23	70000	2	2	2	26	2	0	0	2	2	2	41087	42445	45020	44006	46905	46012	2007	3582	0	3601	0	1820	1
26	24	450000	2	1	1	40	-2	-2	-2	-2	-2	-2	5512	19420	1473	560	0	0	19428	1473	560	0	0	1128	1
27	25	90000	1	1	2	23	0	0	0	-1	0	0	4744	7070	0	5398	6360	8292	5757	0	5398	1200	2045	2000	0
28	26	50000	1	3	2	23	0	0	0	0	0	0	47620	41810	36023	28967	29829	30046	1973	1426	1001	1432	1062	997	0
29	27	60000	1	1	2	27	1	-2	-1	-1	-1	-1	-109	425	259	-57	127	-189	0	1000	0	500	0	1000	1
30	28	50000	2	3	2	30	0	0	0	0	0	0	22541	16138	17163	17878	18931	19617	1300	1300	1000	1500	1000	1012	0
31	29	50000	2	3	1	47	-1	-1	-1	-1	-1	-1	650	3415	3416	2040	30430	257	3415	3421	2044	30430	257	0	0
32	30	50000	1	1	2	26	0	0	0	0	0	0	15329	16575	17496	17907	18375	11400	1500	1500	1000	1000	1600	0	0
33	31	230000	2	1	2	27	-1	-1	-1	-1	-1	-1	16646	17265	15266	15339	14307	36923	17270	13281	15339	14307	37292	0	0
34	32	50000	1	2	2	33	2	0	0	0	0	0	30518	29618	22102	22734	23217	23680	1718	1500	1000	1000	1000	716	1
35	33	100000	1	1	2	32	0	0	0	0	0	0	93036	84071	82880	80958	78703	75589	3023	3511	3302	3204	3200	2504	0
36	34	500000	2	2	1	54	-2	-2	-2	-2	-2	-2	10929	4152	22722	7521	71439	8981	4152	22827	7521	71439	981	51582	0
37	35	500000	1	1	1	58	-2	-2	-2	-2	-2	-2	13709	5006	31130	3180	0	5293	5006	31178	3180	0	5293	768	0
38	36	160000	1	1	2	30	-1	-1	-2	-2	-2	-1	30265	-131	527	-923	-1488	-1884	131	396	396	565	792	0	0
39	37	280000	1	2	1	40	0	0	0	0	0	0	186503	181328	180422	170410	173901	177413	8026	8060	6300	6400	6400	6737	0
40	38	60000	2	2	2	22	0	0	0	0	0	-1	15054	9806	11068	6026	-28335	18660	1500	1518	2043	0	47671	617	0
41	39	50000	1	1	2	25	1	-1	-1	-2	-2	-2	0	780	0	0	0	0	780	0	0	0	0	0	1
42	40	280000	1	1	2	31	-1	-1	2	-1	0	-1	498	9075	4641	9976	17976	9477	9075	0	9976	8000	9525	781	0
43	41	360000	1	1	2	33	0	0	0	0	0	0	218668	221296	206895	628699	195969	179224	10000	7000	6000	188840	28000	4000	0
44	42	70000	2	1	2	25	0	0	0	0	0	0	67501	66999	63949	63699	64718	69970	3000	4500	4047	2500	2800	2400	0

Figure 4.1: Sample data set.

data.

- Using the method of least squares to fit the coefficients as shown previously [11].

On the figure 4.1 a figure on the credit card clients, for data visuals.

Data was randomly divided into two groups, one; for model training and the other; to validate the model. The training data is based on error rates, in our research we will show how the artificial neural network (ANN) is the best models in the classification methods, along with other CNN and RNN. We have the following Mathematica code, Mathematica as other softwares, has been growing in machine learning \ AI community. It is able to read the data from the csv file.

Appropriate code in Mathematica is given below.

```
DataMatrix = Import["defaultofcreditcardclients.csv"];
NumberOfRows = Dimensions[DataMatrix][[1]];
TrainingSet = Table[DataMatrix[[i, 1 ;; 23 ]] -> DataMatrix[[i, 24 ]],
     {i, 1, NumberOfRows}];
```

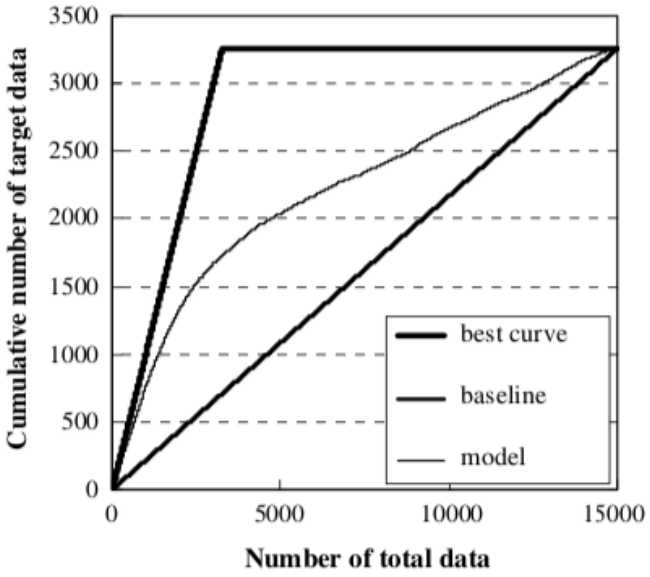


Figure 4.2: Logistic Regression chart [13].

```
Prediction = Classify[TrainingSet, Method -> "NeuralNetwork"]
Prediction[{20000, 2, 2, 1, 24, 2, 2, -1, -1, -2, -2, 3913, 3102, 689, 0, 0, 0,
   0, 689, 0, 0, 0, 0}]
Prediction[{50000, 1, 2, 1, 57, -1, 0, -1, 0, 0, 0, 8617, 5670, 35835, 20940,
  19146, 19131, 2000, 36681, 10000, 9000, 689, 679}]
```

The command was given by: DataMatrix = Import["defaultofcreditcardclients.csv"] . After reading csv file and stored appropriate information into matrix. Next we move onto our TrainingSet= Table[DataMatrix[[i,1;;23]]-> DataMatrix[[i,24]],{i,1,NumberOfRows}], the training-set contains a data in the format readable by the function **classify.** **Prediction = Classify[TrainingSet]**, For the output of the function *classify*, can be used for predictions.

On the figure above we see a chart of the logistic regression, with its classification accuracy we have the following information for the *Error rate* on the training data we have 0.20% of accuracy and 0.18% on validation of data. Then we have,

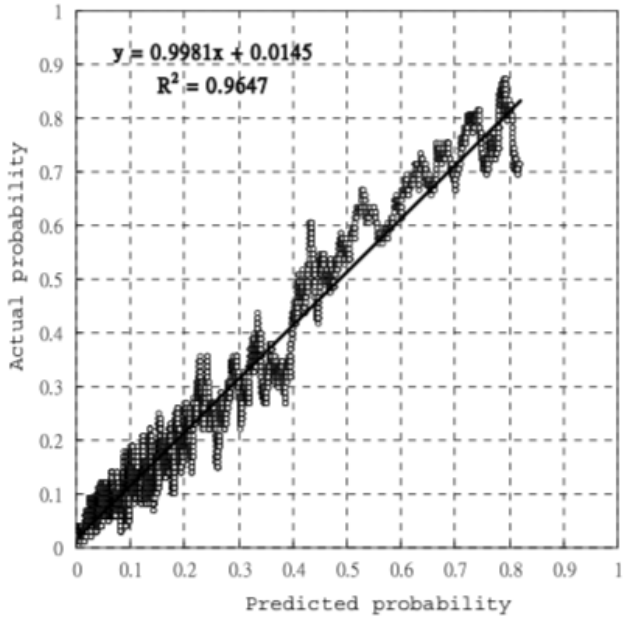


Figure 4.3: Scatter plot diagram of artificial neural networks [13].

Predition[{20000,2,2,1,24,2,2,-1,-1,-2,-1,3913,3102,689,0,0,0,0,689,0,0,0,0}]

providing *output* :1 (correct). To continue with different training-set we get,

Predition[{50000,1,2,1,57,-1,0,-1,-1,0,0,0,8617,5670,35835,20940,19146,19131, 2000,36681,10000,9000,689,679}]

with *output*:0 (correct).

To move forward with the application of Artificial neural network (ANN) we have the following scatter plot to represent our data prediction.

Artificial neural networks perform classification more accurately than the others [13]. In the predictive accuracy of probability of default, artificial neural networks have shown the best performance based on R2 (0.9647, close to 1), regression intercept (0.0145, close to 0), and regression coefficient (0.9971, close to 1). The predictive default probability produced by ANN is the only one that could be used to represent real probability of default.

As the perspective from risk control, when it comes to estimating the default is more

Table 4.1: Classification accuracy [13]

method	$(\frac{ErrorRate}{Training})$	Validation
K-nearest neighbor	0.18	0.45
Logistic regression	0.20	0.44
Neural networks	0.19	0.54
Naïve Bayes	0.21	0.21

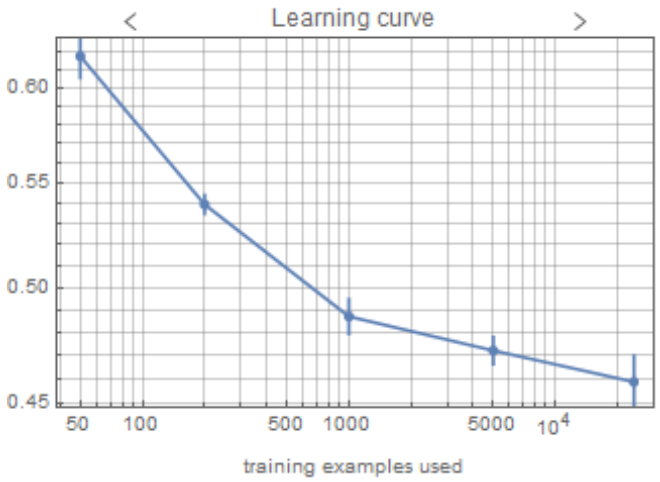


Figure 4.4: Learning curve for neural network (NetworkDepth=8, MaxTrainingRounds=10, Accuracy=0.818 ± 0.007, Loss=0.459 ± 0.011).

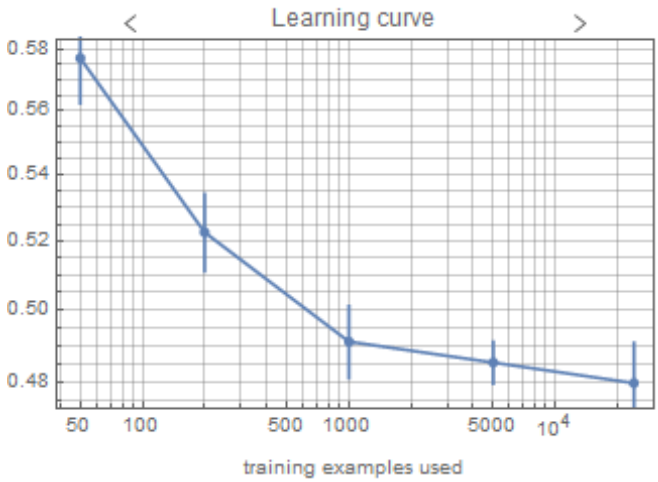


Figure 4.5: Learning curve for logistic regression (L2Regularization=0.1, OptimizationMethod=LBFGS, Accuracy=0.807 ± 0.007, Loss=0.48 ± 0.011)

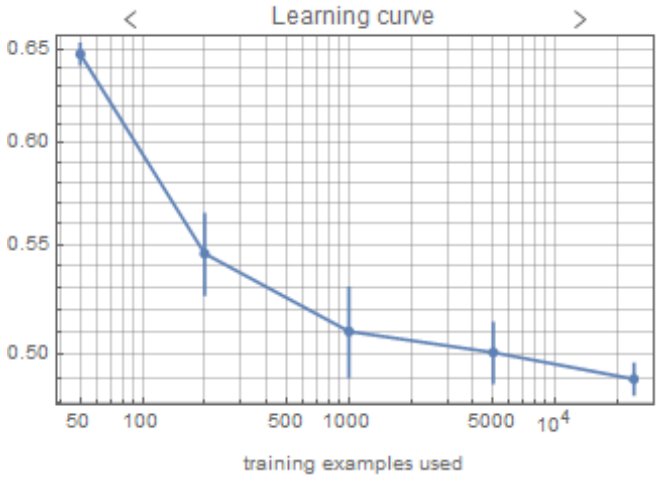


Figure 4.6: Learning curve for random forest (FeatureFraction=$\frac{1}{\sqrt{26}}$, LeafSize=5, TreeNumber=50, DistributionSmoothing=0.5, Accuracy=0.817 ± 0.007, Loss=0.49 ± 0.0064

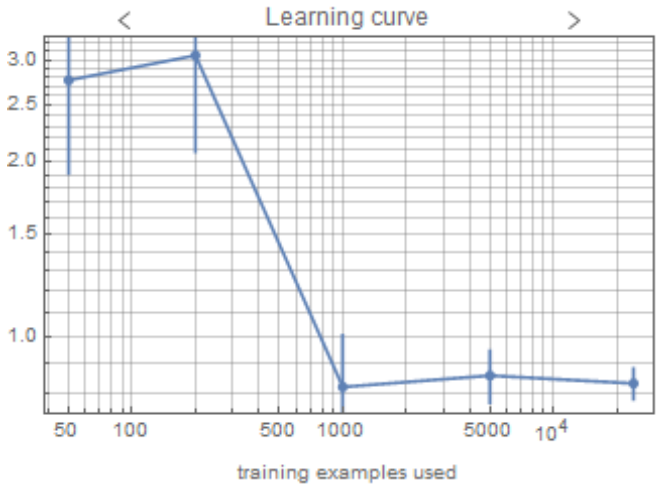


Figure 4.7: Learning curve for Naive Bayes (SmoothingParameter=0.2, Accuracy=0.776 ± 0.0012, Loss=0.830 ± 0.050)

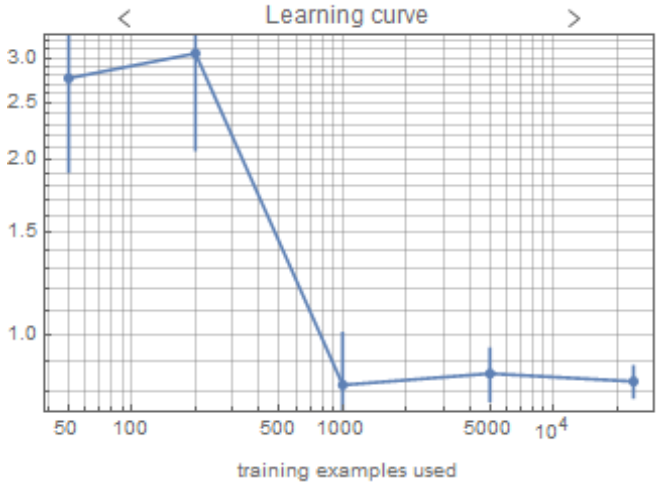


Figure 4.8: Learning curve for K-Nearest Neighbors (NeighborsNumber=100, DistributionSmoothing=0.5, NearestMethod=Scan, Accuracy=0.782 ± 0.021, Loss=0.497 ± 0.027)

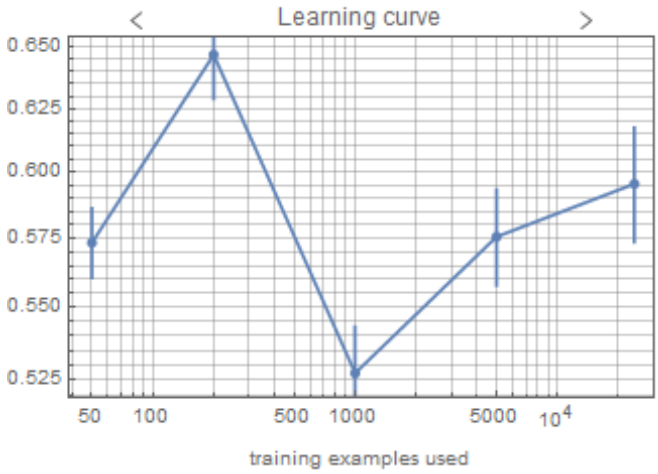


Figure 4.9: Learning curve for Support Vector Machine (KernelType=RadialBasisFunction, GammaScalingParameter=0.0182614, SoftMarginParameter=0.1, PolynomialDegree=3, BiasParameter=1, MulticlassStrategy=OneVersusOne, Accuracy=0.795 $\pm$ 0.003, Loss=0.527 $\pm$ 0.016)

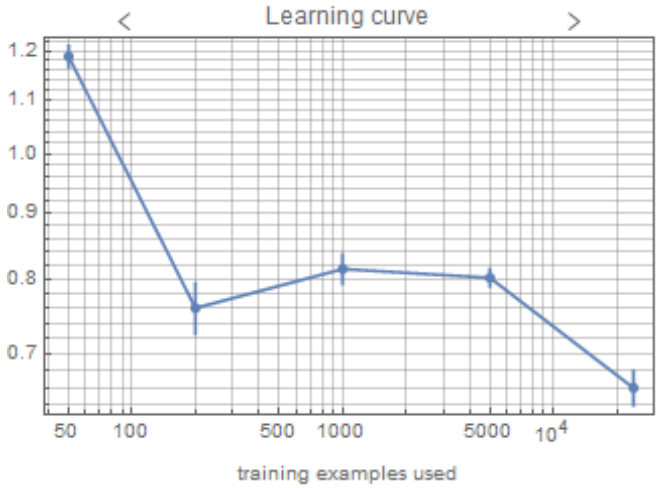


Figure 4.10: Learning curve for Decision Tree (DistributionSmoothing=1, FeatureFraction=1, Accuracy=0.747 $\pm$ 0.008, Loss=0.66 $\pm$ 0.020)

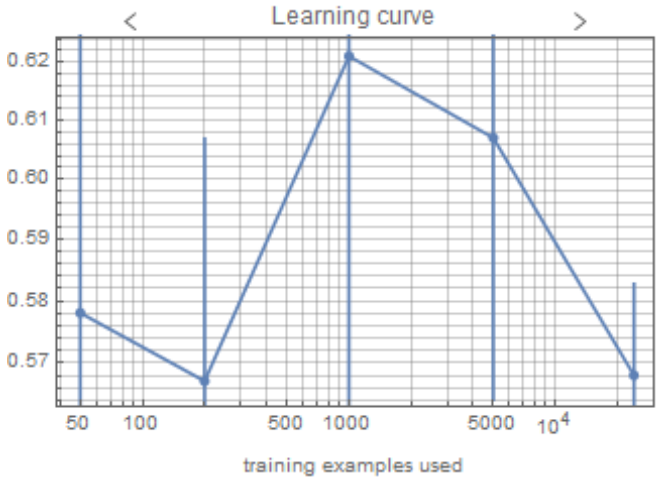


Figure 4.11: Learning curve for Markov model (Order=1, MinimumTokenCount=0, Accuracy=0.761 ± 0.0011, Loss=0.568 ± 0.0‘5)

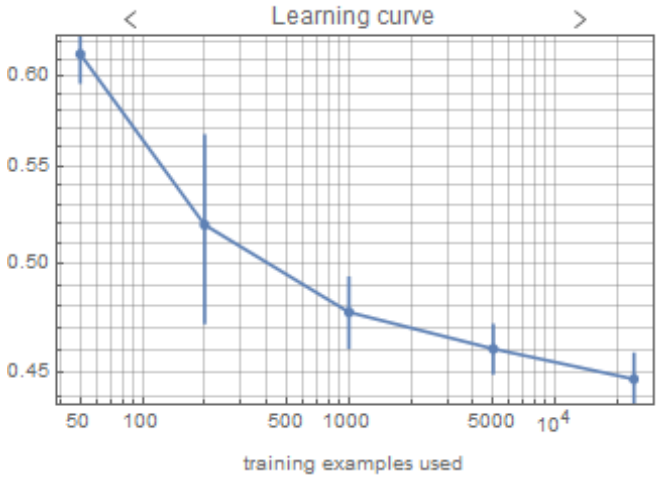


Figure 4.12: Learning curve for Gradient boosted trees (BoostingMethod=Gradient, MaxTrainingRounds=50, LeavesNumber=500, LearningRate=0.1, MaxDepth=6, LeafSize=35, L1Regularization=0, L2Regularization=0, Accuracy=0.82 ± 0.007, Loss=0.447 ± 0.0011)

meaningful than classifying clients into binary results to - risky and non-risky. Therefore, artificial neural networks should be employed to score clients instead of other data mining techniques, such as logistic regression.

4.2 Crashes in Stock Market

The collapse of the Thai baht in July 1997 was followed by an unprecedented financial crisis in East Asia around 1997-1999, from which some of these economies are still struggling to recover. From this time frame we will showcase the forecasting of the crashes for 9 of those countries that suffered from the devaluation in Thailand, from the foreign exchange market pressure, along with the downfall of Thai baht. Not only did the collapse of Thai baht affect other Asian countries, it also had an impact on Latin America, Caribbean countries, and United States. The countries that will be showcased in our application are the following:

- Thailand 97'-01'
- Brasil 93'-01'
- Mexico 91'-01'
- Argentina 96'-01'
- Hong Kong 97'-01
- Philippines 97'-01'
- Turky 97'-01'
- USA 97'-01'

Of forecasting with the methods of neural networks, those which we have spoken about on earlier chapters, ANN, RNN, and CNN.

The $S\&P$ 500 index as mention back on chapter 1, is an index of the market-capitalization-weighted index of the 500 largest U.S. publicly traded companies, measured by market capitalization. According to the details available on TSX *website*[1], the constituent stocks currently represent three industrial groups: Industrial, Mining, and Oil&Gas. This index

is the basis for the most highly traded futures contract in United States. Market capitalization (market cap or capitalized value) is a measurement of corporate or economic size equal to the share price times the number of shares outstanding of a public company. A float-adjusted index is an index that weighs the component securities by the relative capitalization of only those shares that are available to the public for trading, rather than the total shares outstanding. A majority of Standard & Poors equity indices are market capitalization weighted and float-adjusted, so each stocks weight in the index is proportional to its float-adjusted market value. The second group of indices are equal weighted, where each stock is weighted equally in the index. A third, and newer, group of indices are weighted by other factors, such as the attributes used to choose stocks. The simplest capitalization weighted index can be thought of as a portfolio consisting of all available shares of the stocks in the index. As one might track this portfolios value worth in dollar terms, it would be an unwieldy value. For example, the $S\&P$ 500 market value is roughly 11 trillion. The scaling is done by dividing the portfolio market value by a factor, usually called the *index divisor*.

$$Index\ Value = \frac{\sum_i P_i Q_i}{Index\ Divisor},$$

where P_i is the stock of index i and Q_i is the number of the shares of the stock i, used in the index calculations. Using data from individual stocks to forecast the future value of the overall index, in the short term (i.e., within the span of a few days). In order to be able to measure the performance/quality of the models and methods on a data set different than the exact set used for training purposes, each set of possible two-day end-of-day closing value differences for the component stocks constituting the index is broken into two sets, one used for training purposes, and one used for testing purposes as mentioned before in our data preprocessing step. It is worth mentioning that in statistical learning applications, during a validation process, there is a random decomposition of data into testing and training. However, as this work is focusing on time-series financial data, in order to be able to perform predictions of future movement in the series, we have chosen to select a

certain number of beginning observations as our training set and the remainder of the data set as our testing data. By applying machine learning methodologies to these data sets, knowing when most high and low frequency occurred, to forecast this information is many in the stock market industries to novice. More and More people are devoted to the study of the prediction and as Artificial Intelligence, Deep learning continues to grow the ability to forecast is becoming easier and easier for us to make stock prediction by using different neural networks. In the following two sections we will showcase two neural networks on forecasting the stock price movement. We set the opening price, high price, low price, closing price and volume of stock deriving from the internet as input of the architecture and then run and test the program.

4.3 Forecasting via Convolutional Neural Networks

We start off with forecasting with convolutional neural network that we had introduced back in chapter 3. Traditionally, the input of a convolutional neural network is often a 2D image, but that does not mean that we cannot use the model to help us make predictions. There are two ways to preprocess the data, one we can convert the 1D-input data into a 2D matrix and the other way is to take advantage of 1D function to help us do the computation of convolution. We chose the latter to help us run and test the experiment. Then, due to the stock data belonging to 1D time series data, a 1D function is applied to do the convolution and set five features including volume, high price, low price, closing price and volume as input [14]. The convolutional neural network (CNN) may be defined the following way by using (software) TensorFlow.

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, \\
      activation='relu', input_shape=(n_steps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
```

```
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Our task it to do the binary classification instead of regression, so the output of our model only consists of two values–one or zero, to show whether the stock price movement will be up or down. When people are considering on buying a stock they will do when the result of forecasting is a 1 and look to sell once price is posted. On the contrary, if forecast is 0, one my decided to sell, or do other financial moves to not lose any equity. To minimize our computing time, we have vectorized the input data, meaning that we will be training our model at one time. The input form may be defined by the following matrix,

$$X = (x^{(1)}, x^{(2)}, ..., x^{(n)})$$

n indicates the number of sample inputs.

The output form similarly,

$$Y = (y^{(1)}, y^{(2)}, ..., y^{n})$$

as y as our output value it will make it a Boolean value: 1 and 0. Adopting the min-max normalization to keep value at range of 0 to 1.

$$x^* = \frac{x - min}{max - min},$$

Max indicates the maximum value of the features and min indicates the minimum of the features.

Some sample prediction by using convolutional neural network is shown in the Fig. **??**.

As perviously stated information on accuracy (chapter 2-3), it is possible to apply by several different metrics to find our accuracy percentage. In this thesis the Mean Square Error will be applied (MSE). Appropriate code in TensorFlow is given below.

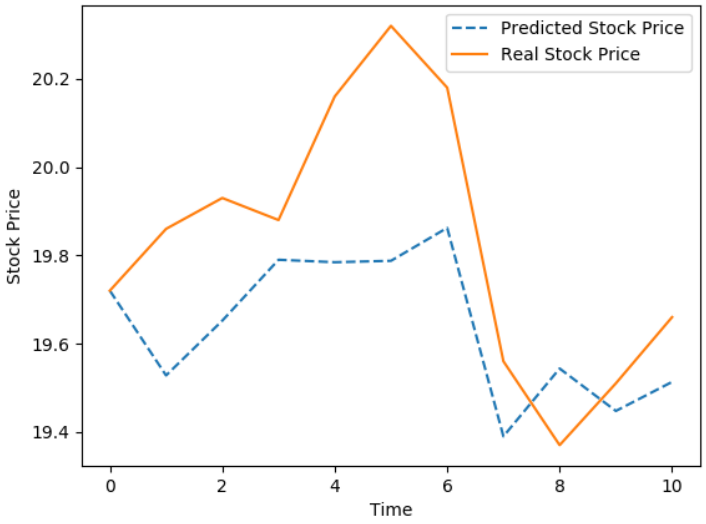


Figure 4.13: Sample prediction of crash for SP 500 (high) by using convolutional neural network.

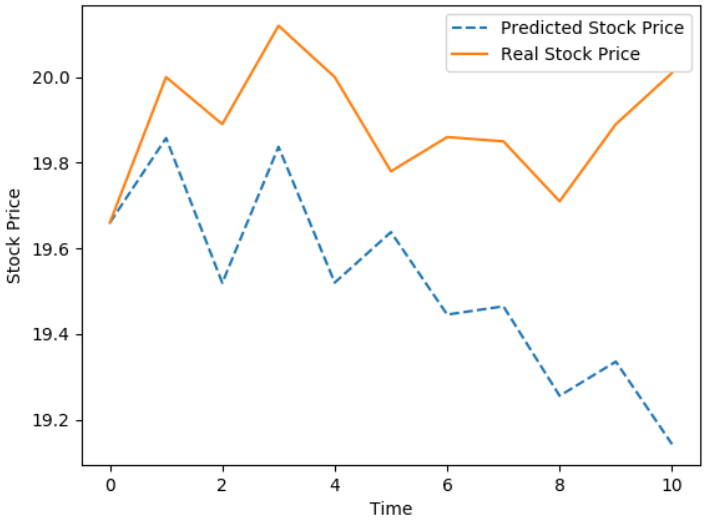


Figure 4.14: Forecast Crash for SP 500 (close) by using convolutional neural network.

We will use "teaching" number to train the model, with firstly vector of stock data will be input into the stock prediction model and we will use the conv1d function to do the computation of convolution. It is necessary to re-introduce the activation function. Traditionally, the activation function most frequently used can be concluded as followings: *Tanh*, *Relu*, *Sigmoid* and *Lrelu*, as mention on earlier chapter. To solve the problem of gradient vanishing, we choose the Relu and Lrelu function to do the computation of activation. The ReLU function and LReLU function can be defined as followings:

$$Relu\ (x) = \begin{cases} 0 & (x \leq 0) \\ x & (x > 0) \end{cases}$$

and

$$LRelu\ (x_i) = \begin{cases} a_i, x_i & (x \leq 0) \\ x_i & (x > 0) \end{cases}.$$

Where x and x_i both are results of the computation of convolution in the convolutional layer a_i is a hyper-parameter with a very small value and usually we choose 0.01 as its initial value (i.e. it can also be fine-tuned to get the best result while training).

The final value the has been forecast will then be compared, and we will compute its average error. Goal in hand during the training period is to minimize its error in order to improve its accuracy. Here is were we apply optimization method with loss function, first computing the result of the softMax function used in the fully connected layer, denoted as follows:

$$y_i = \frac{e^{a_i}}{\sum_{k=1}^{C} e^{a_k}}\ , i = 2.$$

With i indicating the number of classifications done, and a is the output of the connected layer, finally C is the number of a an e of natural logarithm. With Error function computed as the following:

$$J(W) = -\sum_{k=1}^{n}\sum_{i=1}^{C} t_{ki} log(y_{ki}),$$

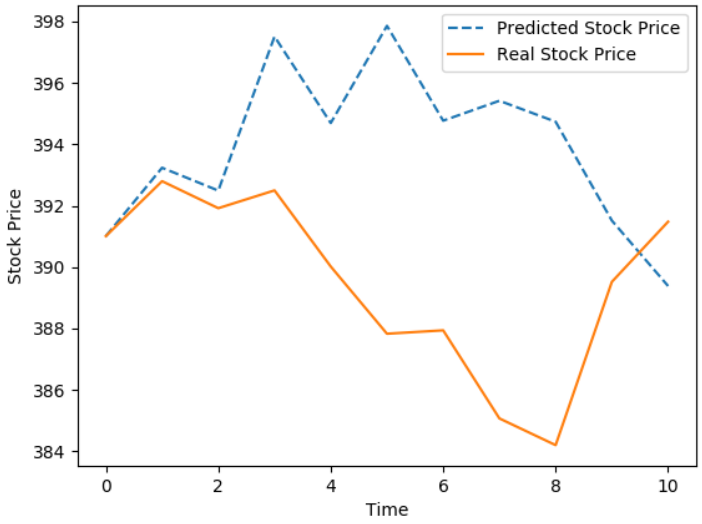


Figure 4.15: Forecast Crash for SPC USA(close) by using convolutional neural network.

n is the number of the samples, and C is the number of the output of the fully connected layer, t_{ki} is the probability that samples k belongs to class i, y_{ki} then indicates the probability that models predict sample k which belongs to class I and W for the weight coefficient of filters. Now in order for us to minimize the loss, we use what we have introduction in our optimization methods sections, backpropagation algorithm, based on gradient descent (check earlier chapter for review on topics). Taking ADAM algorithm proposed in to do the computation of the optimization and we use the following equation to update the parameter W.

$$W^{'} = W - \eta \cdot \frac{\partial J(W)}{\partial W}$$

Where $W^{'}$ is the value which will be updated after each gradient descent iteration, and η is the learning rate set to 0.0001 in experiment.

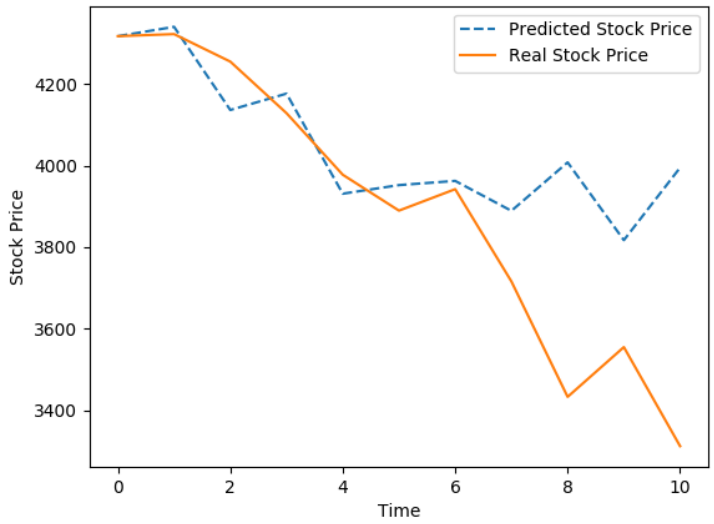


Figure 4.16: Forecasting Crash for XU100 Turkey (close) by using convolutional neural network.

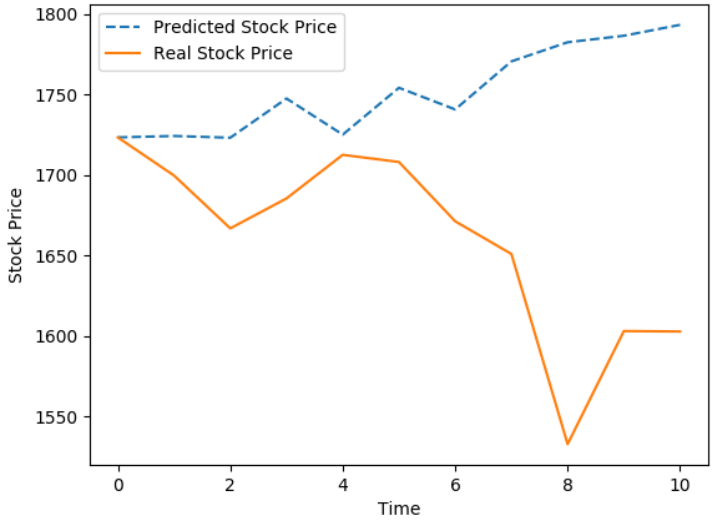


Figure 4.17: Forecasting Crash for Nasdaq USA (close) by using convolutional neural network.

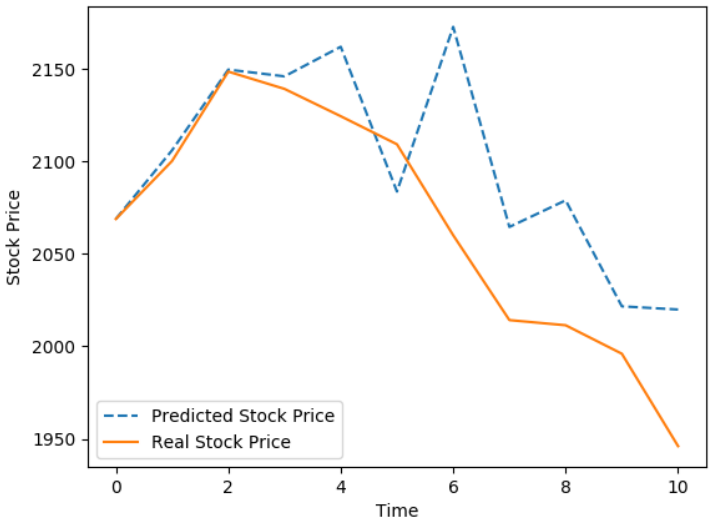


Figure 4.18: Forecasting Crash for PSI Philipines (close) by using convolutional neural network.

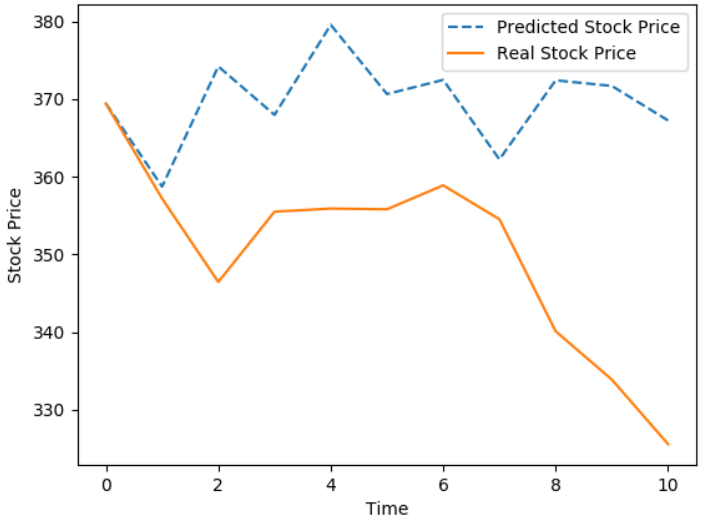


Figure 4.19: Forecasting Crash for SETI Thailand (close) by using convolutional neural network.

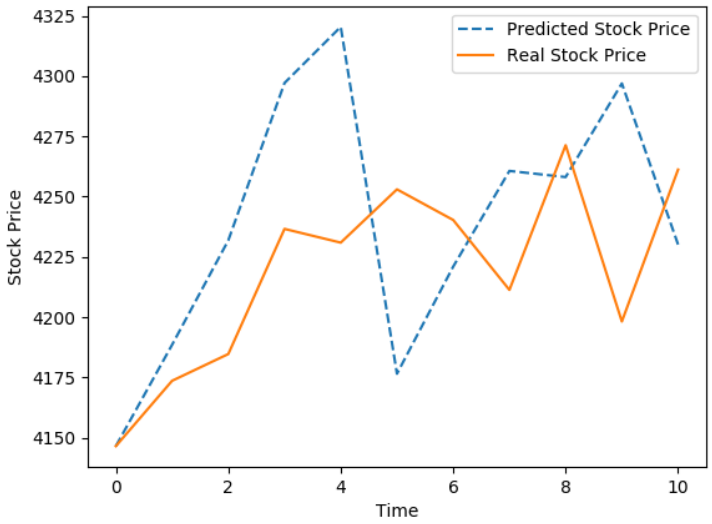


Figure 4.20: Forecast Crash for HSI HongKong (close) by using convolutional neural network.

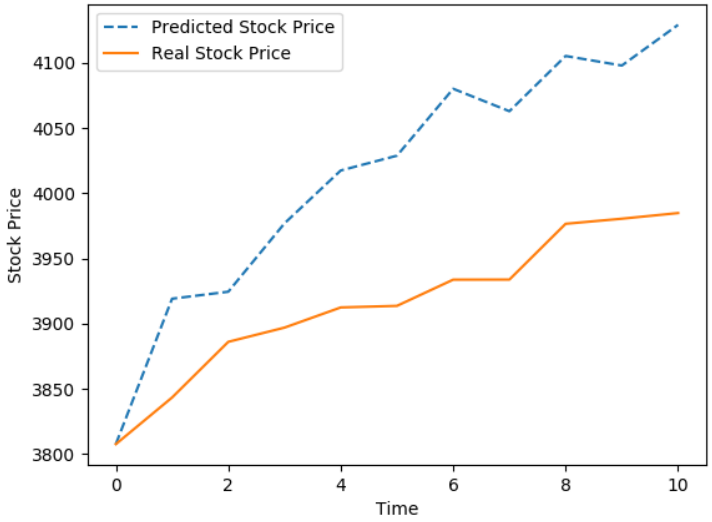


Figure 4.21: Forecast Crash for IGPA Chile (close) by using convolutional neural network.

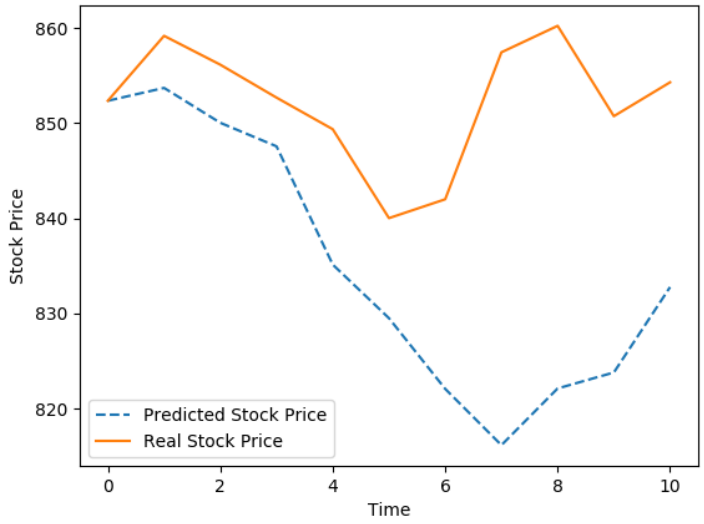


Figure 4.22: Forecast Crash for MERV Arg (high) by using convolutional neural network.

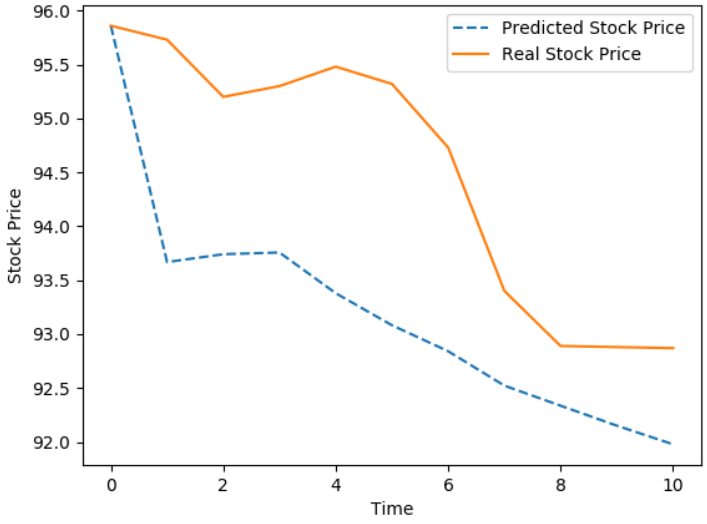


Figure 4.23: Forecast Crash for SPC 77 (low) by using convolutional neural network.

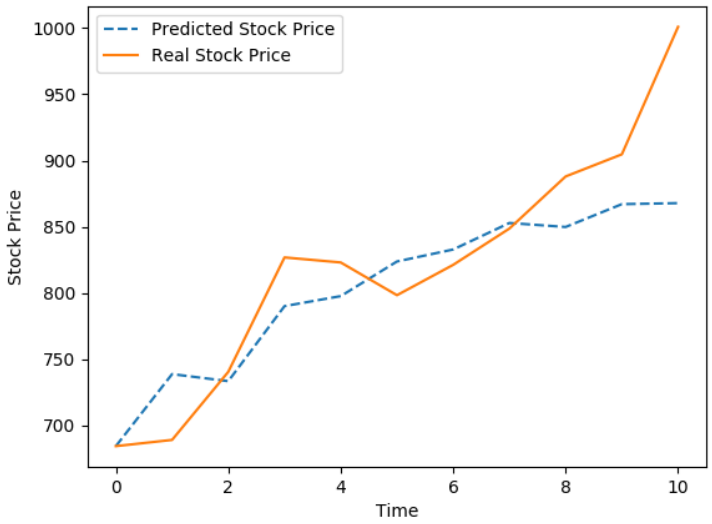


Figure 4.24: Forecast Crash for BVSP Bras (close) by using convolutional neural network.

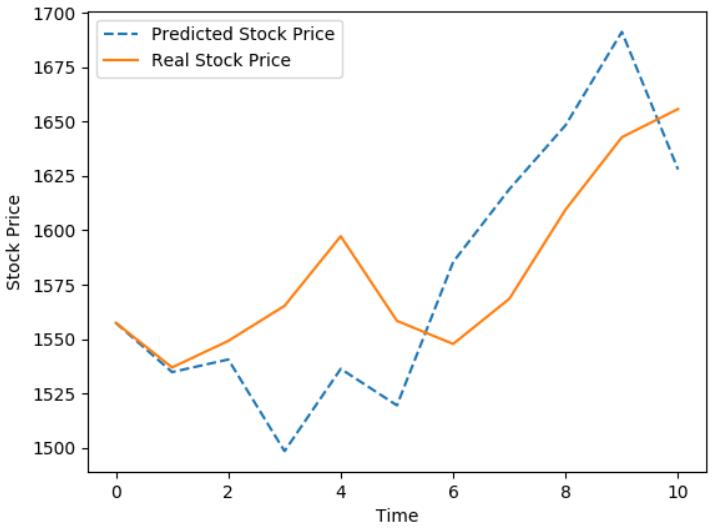


Figure 4.25: Forecast Crash for MXX Mexico (close) by using convolutional neural network.

4.4 Forecasting via Recurrent Neural Networks

In the choice of model we have well known Feed-forward neural network, which allows signals to travel one way only: from input to output and there are no feedback (loops), the output of any layer does not affect that same layer. In Recurrent neural network, it allow signals to traveling in both directions by introducing loops in the network. There is a feedback (loops) and feedback networks are powerful, dynamic and can get extremely complicated. Their state is changing continuously until they reach an equilibrium point. In stock price prediction we need a dynamic feedback network where we can follow the unpredictable trend of the stock price dynamically. That is why we chose recurrent neural network (RNN) for us to evolve it on our forecasting method. A recurrent network is a class of advanced artificial neural network (ANN) that involves directed cycles in memory. In a recurrent neural network; its connections between nodes form a directed graph along a sequence, which allows exhibiting dynamic temporal behavior for a time sequence. Suppose one wants to predict the next word in a sentence or to forecast the next day stock price etc. by using machine learning methodologies [15]. The simplest form has an input layer which receives the input, a hidden layer where the activation function is applied and an output layer where one finally receives the output, covered in chapter 3.

In TensorFlow Recurrent neural network can be defined in the following way:

```
# Initialising the RNN
regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True, \\
          input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
```

```
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a third LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

# Adding the output layer
regressor.add(Dense(units = 1))
```

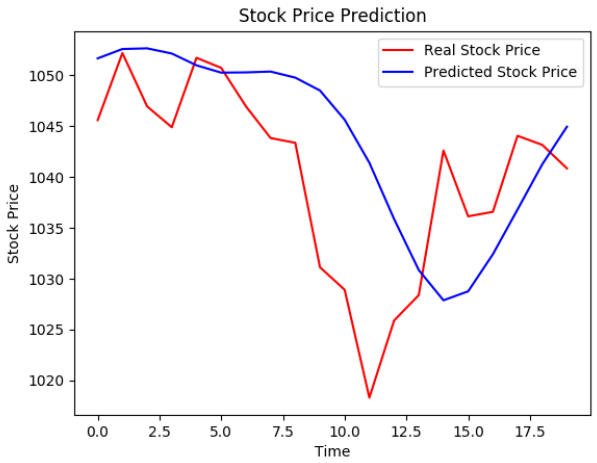


Figure 4.26: Validation of the model for SP 500 by using Recurrent Neural Network (RNN) for 100 epochs.

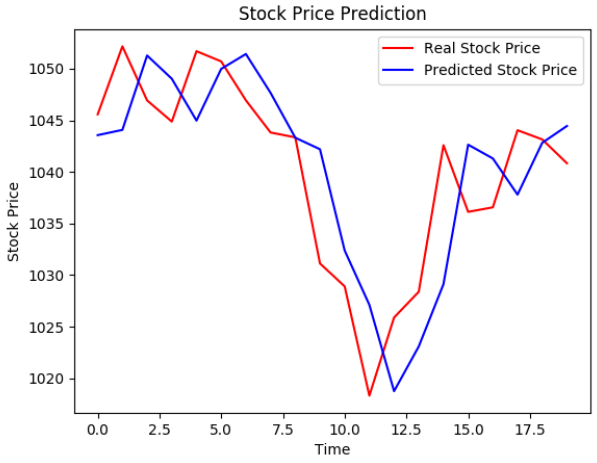


Figure 4.27: Validation of the model for SP 500 by using Recurrent Neural Network (RNN) for 400 epochs.

4.4.1 SP 500 methods with multiple input files via RNN

As research has progressed it was observed that it is possible to consider multiple input data files. In presented example we have, 4 different input files were used: SP500_50-05-Close.csv, SP500_50-05-High.csv, SP500_50-05-Open.csv, SP500_50-05-Low.csv, normally we run each separately and by considering running these multiple inputs we save computational time and work load is increased. For a quick sample results for the different number of epochs is given below. In theory, it is possible to use arbitrary number of input files which increase accuracy of the perditions, and allows us to take into account a lot of dependency in the input data, which are not available for the calculations with single data file. An argument can be made that it is possible to create one mathematical model for all stocks that are available on the market. It is also possible to include additional information: Natural disasters, international and national political situation, foreign exchange etc. With any extra information it may increase accuracy of predictions, and help mold a comprehensive mathematical model of financial market. A precise model of financial market can be applied to simulate different economical strategy and find optimal solution for given situation on

the market.

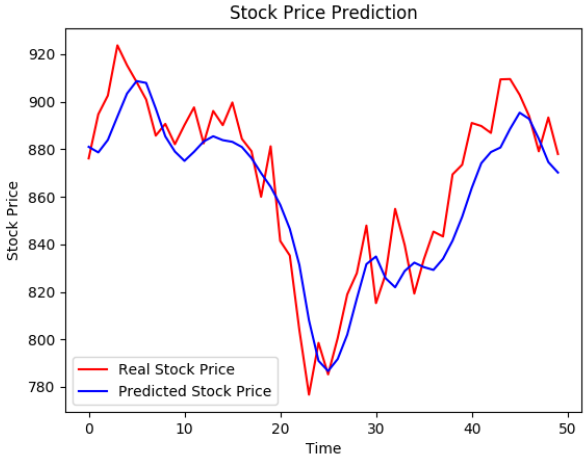


Figure 4.28: Validation of the model for SP 500 (close) by using Recurrent Neural Network (RNN) for 400 epochs.

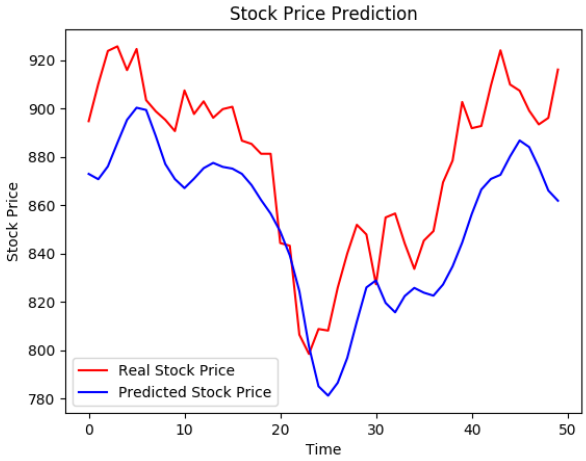


Figure 4.29: Validation of the model for SP 500 (high) by using Recurrent neural network (RNN) for 400 epochs.

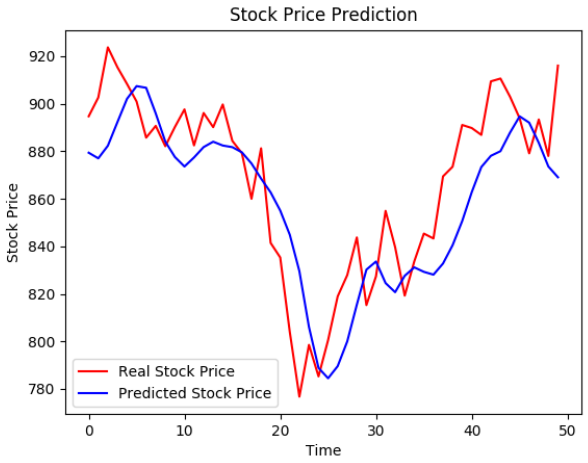


Figure 4.30: Validation of the model for SP 500 (open) by using Recurrent neural network (RNN) for 400 epochs.

Chapter 5

Educational Data

5.1 Los Alamos National Laboratory

Artificial Intelligence is one of the scientific areas with greater diffusion and application in the last years and continues to show tremendous growth. As missing data being of a common problem in statistical analysis and with an everyday growth it is more common to find tools for industrial, commercial or academic that use intelligent techniques in the resolution of critical and recurrent problems. At this moment is where the collaboration occurred with Claire McKay Bowen, Ph.D. (Postdoctoral Research Associate) Statistical Sciences Group (CCS-6), and fellow Joanne R. Wendelberger, Ph.D Lead of the Statistical Sciences Group, (CCS-6) from Los Alamos National Laboratory (LANL), and myself. My research topic started with the time series and its forecasting on a financial data set, to forecasting crashes and credit card default in multivariate time series via the usage of machine learning methodologies. From here, Claire and Joanne came up with a tentative outline for my summer project. The title of my LANL project, **Tile**: Analyzing the Effects of Missing Data in Time Series with Machine Learning Algorithms, with **Problem**: Educational and career data sets tend to contain missing data, which is exacerbated the longer the data collection/tracking is (e.g. K-12 to college to graduation). The computational statistical group (CCS-6) from LANL, provided all the resources that I needed for my project to reach full success. Chapter 5 will be dedicated to showcasing the summer project done for Los Alamos National Lab. With great appreciation to Los Alamos and especially to Claire and Joanne for their support and amazing inspiration for future work.

Method Name	Category	Software	Reference
Mean impute (`mean`)	Mean		Little and Rubin (1987)
Expectation-Maximization (`EM`)	EM		Dempster et al. (1977)
EM with Mixture of Gaussians and Multinomials	EM		Ghahramani and Jordan (1994)
EM with Bootstrapping	EM	`Amelia II`	Honaker et al. (2011)
K-Nearest Neighbors (`knn`)	K-NN	`impute`	Troyanskaya et al. (2001)
Sequential K-Nearest Neighbors	K-NN		Kim et al. (2004)
Iterative K-Nearest Neighbors	K-NN		Caruana (2001); Brás and Menezes (2007)
Support Vector Regression	SVR		Wang et al. (2006)
Predictive-Mean Matching (`pmm`)	LS	`MICE`	Buuren and Groothuis-Oudshoorn (2011)
Least Squares	LS		Bø et al. (2004)
Sequential Regression Multivariate Imputation	LS		Raghunathan et al. (2001)
Local-Least Squares	LS		Kim et al. (2005)
Sequential Local-Least Squares	LS		Zhang et al. (2008)
Iterative Local-Least Squares	LS		Cai et al. (2006)
Sequential Regression Trees	Tree	`MICE`	Burgette and Reiter (2010)
Sequential Random Forest	Tree	`missForest`	Stekhoven and Bühlmann (2012)
Singular Value Decomposition	SVD		Troyanskaya et al. (2001)
Bayesian Principal Component Analysis	SVD	`pcaMethods`	Oba et al. (2003); Mohamed et al. (2009)
Factor Analysis Model for Mixed Data	FA		Khan et al. (2010)

Figure 5.1: Table of imputation methods

5.2 Introduction to missing data

As missing data is common problem in real-world settings which has lead to significant attention in the statistical literature. There are flexible frameworks based on formal optimization to impute missing values. Research has shown that these frameworks can readily incorporate various predictive models such as K-nearest neighbors, Support Vector Machines, and Decision tree based methods, and can be adapted for multiple imputation. With missing data phenomena is arguably the most common issue encountered by machine learning practitioners when analyzing real-world data (raw data). As we know that for many statistical models and machine learning algorithms rely on complete data sets, it becomes extremely important to handle the missing values appropriately. Below you will find a table of imputation methods for deeper perception on methodologies and their references.

There are some machine learning studies done, that have shown some algorithms naturally account for missing data and there is no need for preprocessing. In particular, CART and K-means have been adapted for problems with missing data. But as for many other situations, missing values need to be imputed prior to any statistical analyses on the complete

data set. Going forward lets assume that we are given data,

$$X = \{x_1, ..., x_n\}$$

with missing entries $x_{id}, (i, d) \in \mathcal{M}$. The objective is to impute the values of the missing data that mirror the underlying complete data as closely as possible. From this point, one may apply pattern recognition using machine learning methods on the imputed data and results should be complementary to the complete data given.

5.3 Data description

On this study data we started with a subset data description,which lead us with an initial simulation of data before moving forward. Our multivariate data had some dependent variables and Independent variables, with the total of 14 variables, some discrete others continuous. Below figure for visual of data description. So before moving forward with our imputation, we needed a simulated dataset. In order to s simulated a dataset, the sample data used for prediction is given by csv file of 1000 random generated students. For a more complex data we have simulated a data-set of 1000 students with name for columns for input given by :

- **Gender**: female=1; male=0.
- **Race**: white=1; African-American (black)=2; Asian =3; American Indian or Alaskan Native =4, Native Hawaiian or other Pacific Islander , ethnicity: non-hispanic white =1; hispanic or latino =2 ; non-hispanic Black =3; Chinese=4; European=5 ; Arab =6 ; Indigenous=7; Filipino = 8.
- Immigrant: Yes=1, No=0
- High school-age=numeric from 14-15

- Rank
- Dual-credit, Ap credit
- Combined SAT and ACT scores
- Parents highest education received
- Family gross income

The variable that will be used for forecasting will be the *output of degree completion.*

In order to test effectiveness of the methods presented in the Educational data I generated test data with different properties. It is also possible to predict missing data by using different mathematical techniques along with statistical tools. I effectively used existing statistical software for missing data (MICE, missForest, MI,). The results for methods well executed in this chapter. As research shows that in order to predict missing data it is possible to apply machine learning techniques (neural networks), this application of simulation and prediction will also be showcased below. Through the usage of neural networks a module was developed to simulate the process described above. We simulated independent paths of our model using different time steps for the data sets.

5.4 Imputation methods

As for the imputation packages found in R, we have the following used on our Educational missing data-set, **MICE** (Multivariate Imputation via chained equations) is one of the commonly used packages in R. Creating multiple imputations as compared to a single imputation (such as the mean) it takes care of uncertainly in missing values. MICE assumes that the missing data are Missing at Random (MAR), which means that the probability that a value is missing depends only on observed value and can be predicted using them. It imputes data on a variable by variable basis by specifying an imputation model per

Variable Name	Variable Type	Description or Coding
Dependent Variable		
Degree completion	Time-varying	Dichotomous variable indicating whether a student received a STEM degree by postsecondary public institution for an academic year
Independent Variables		
Student Characteristics		
Gender	Time-invariant	M = Male, F = Female
Gifted/Talented	Time-varying	00, 01
Immigrant	Time-invariant	00, 01
Race/ethnicity	Time-invariant	*New Ethnic Origin*: 1 = Hispanic or Latino origin, 2 = Not Hispanic or Latino origin, 3 = Not answered *Race*: 1 = White, 2 = African-American/Black, 4 = Asian, 5 = American Indian or Alaskan native, 6 = International, 7 = Unknow or Not Reported, 8 = Native Hawaiian or Other Pacific Islander
Family gross income	Time-invariant	Continuous variable indicating total parental income
Parent education	Time-invariant	*Highest grade level mother completed*: 1 = Elementary, 2 = High school, 3 = College or beyond, 4 = Unknown *Highest grade level father completed*: 1 = Elementary, 2 = High school, 3 = College or beyond, 4 = Unknown
Age at enrollment	Time-invariant	Continuous variable indicating age at enrollment
Academic Preparation for College		
Combined SAT (or ACT) score	Time-invariant	Combined scores from the SAT Mathematics and Evidenced-based Reading and Writing (EBRW) tests
High school rank	Time-invariant	1 = Accepted and ranked in top 10% of high school graduating class, 2 = Accepted and ranked in 11-25% of high school graduating class, 3 = Others
Dual credit	Time-invariant	Dichotomous indicator of whether a student ever earned dual credits
AP credit	Time-invariant	Dichotomous indicator of whether a student ever earned AP credits
College Experience		
First-year GPA	Time-invariant	Calculated first-year college GPA
GPA change	Time-invariant	The difference between first-year calculated GPA and last-year cumulative GPA

Figure 5.2: Data description

variable.

Amelia, for a quick history on why this name, Amelia Earhart, was the first female aviator to fly solo across the Atlantic Ocean. History says, she got mysteriously disappeared (missing) while flying over the Pacific ocean, hence the name to solve missing value problems. This package also performs multiple imputation which generates imputed data sets. It is enabled with bootstrap based EMB algorithm which makes it faster and robust to impute many variables including cross sectional, time series data. Amelia makes the following assumptions, 1.) All variables in a data set have multivariate normal distribution, it uses mean and covariances to summarize data. 2.) Missing data is random in nature (MAR). It takes m bootstrap samples and applies EMB algorithm to each sample. The m estimates of mean and variances will be different. Finally, the first set of estimates are used to impute first set of missing values using regression, then second set of estimates are used for second set and so on.

missForest was the third method applied to our Educational data and was the method that was found most success in. As the name suggests, missForest is an implementation of random forest algorithm. It is a non-parametric imputation method applicable to various variable types. Non-parametric method does not make explicit assumptions about functional form of f (any arbitrary function). Instead, it tries to estimate f such that it can be as close to the data points without seeming impractical. missForest works by building a random forest model for each variable, then it uses the model to predict missing values in the variable with the help of observed values.

Another package found in R, was **mi** (multiple imputation with diagnostics) this package provides several features for dealing with missing values. Like other packages, it also builds multiple imputation models to approximate missing values and uses predictive mean matching method. Predictive mean matching (pmm) for numeric variables. For each observation in a variable with missing value, we observe (from available values) with the closest predictive mean to that variable. From there the observed value from this match is then used as the imputed value, *mi* uses Bayesian version of regression models to handle

issue of separation. The imputation model specification is similar to regression output in R, it also automatically detects irregularities in data such as high collinearity among variables, and also it adds noise to imputation process to solve the problem of additive constraints.

Finally, we have **PCA (principal component analysis);** this imputation of incomplete continuous or categorical datasets, handles missing values in exploratory multivariate analysis, an estimate the number of dimensions for the factorial analysis of mixed data by cross-validation. The family of methods adapts the PCA algorithm to consider explicitly the missing values. These procedures return scores for both variables and individuals using an incomplete data set. Ipca : The iterative PCA method (Kiers 1997) also known as the EM-PCA algorithm. Providing the scores and loadings minimizing the least squares criterion on the observed entries, $W \circ \ (X - \hat{X})^2$, with $w_{ik} = 0$ if x_{ik} is missing and 1 otherwise and $\circ$ denotes the elementwise product. The minimization is achieved through an iterative procedure: missing values are replaced by random values, and then PCA is applied on the completed data set, and the missing values are then updated by the fitted values ($\hat{X}_s = U\Lambda V^T$) using a predefined number of dimensions S. This procedure is repeated until reached convergence. The method is implemented in the function imputePCA of the R package missMDA.

5.5 MICE

The R package mice imputes incomplete multivariate data by chained equations. The software mice 1.0 appeared in the year 2000 as an S-PLUS library, and in 2001 as an R package. mice 1.0 introduced predictor selection, passive imputation and automatic pooling. This article documents mice 2.9, which extends the functionality of mice 1.0 in several ways. The usage of mice 2.9, the analysis of imputed data is made completely general, whereas the range of models under which pooling works is substantially extended. mice 2.9 adds new functionality for imputing multilevel data, automatic predictor selection, data handling, post-processing imputed values, specialized pooling routines, model selection tools, and diagnostic graphs. Imputation of categorical data is improved in order to bypass problems caused by perfect prediction. Special attention is paid to transformations, sum scores, indices and interactions using passive imputation, and to the proper setup of the predictor matrix [16].

Sample R code.

```
library(missForest)
library(mice)
library(randomForest)
MissingData<-read.csv(file='data-education-50-r-5-mis.csv')
imputed_Data <- mice(MissingData,m=10, maxit = 50, method = 'pmm', seed =
600)
print(imputed_Data)
completeData <- complete(imputed_Data,action = 1)
write.csv(completeData,'data-education-50-r-5-mis-imputation.csv')
```

Table 5.1: Data for MICE

gender	race	ethnicity	immigrant	highschool_age
1	7	8	2	14.5
0	2	2	0	14
0	1	1	0	14
1	6	6	2	13.8
0	1	1	0	14
0	1	1	0	14
1	5	6	2	13.8
0	2	2	0	14
1	5	6	2	13.8
0	3	3	1	14
0	2	2	0	14
1	6	6	2	13.8
0	4	4	1	14
1	6	7	2	15
1	5	6	1	13.8
0	4	4	1	14
0	3	3	0	14
1	5	5	1	14
0	2	3	0	14
0	2	3	0	14

Table 5.2: Missing data for MICE

	gender	race	ethnicity	immigrant	highschool_age
1	1	7	8	2	14.5
2	0	2	2	0	14
3	NA	1	1	0	14
4	1	6	6	2	13.8
5	0	1	1	NA	14
6	NA	1	1	0	14
7	1	5	6	2	13.8
8	NA	2	2	0	14
9	1	5	6	2	13.8
10	0	3	3	NA	14
11	0	NA	NA	0	14
12	1	6	6	2	13.8
13	NA	4	4	1	14
14	1	6	7	2	NA
15	1	5	6	1	13.8
16	0	4	4	1	14
17	0	3	3	0	14
18	NA	5	NA	1	14
19	0	2	3	0	14
20	0	2	3	0	14

Table 5.3: Data after imputation for MICE

	X	gender	race	ethnicity	immigrant	highschool_age
1	1	1	7	8	2	14.5
2	2	0	2	2	0	14
3	3	0	1	1	0	14
4	4	1	6	6	2	13.8
5	5	0	1	1	0	14
6	6	0	1	1	0	14
7	7	1	5	6	2	13.8
8	8	0	2	2	0	14
9	9	1	5	6	2	13.8
10	10	0	3	3	0	14
11	11	0	1	1	0	14
12	12	1	6	6	2	13.8
13	13	0	4	4	1	14
14	14	1	6	7	2	15
15	15	1	5	6	1	13.8
16	16	0	4	4	1	14
17	17	0	3	3	0	14
18	18	1	5	6	1	14
19	19	0	2	3	0	14
20	20	0	2	3	0	14

5.6 Amelia

Amelia II "multiply imputes" missing data in a single cross-section (such as a survey), from a time series (like variables collected for each year in a country), or from a time-series-cross-sectional data set (such as collected by years for each of several countries). Amelia II implements our bootstrapping-based algorithm that gives essentially the same answers as the standard IP or EM is approaches, is usually considerably faster than existing approaches and can handle many more variables. The program also generalizes existing approaches by allowing for trends in time series across observations within a cross-sectional unit, as well as priors that allow experts to incorporate beliefs they have about the values of missing cells in their data. Amelia II also includes useful diagnostics of the fit of multiple imputation models. The program works from the R command line or via a graphical user interface that does not require users to know R. Amelia, multiple imputation involves imputing m values for each missing cell in your data matrix and creating m "completed" data sets. (Across these completed data sets, the observed values are the same, but the missing values are filled in with different imputations that reflect our uncertainty about the missing data.) After imputation, Amelia will then save the m data sets. Once this process is done we can then apply appropriate statistical method complete m data sets, and use a simple procedure to combine the results. Under normal circumstances, you only need to impute once and can then analyze the m imputed data sets as many times and for as many purposes as you wish. The advantage of Amelia is that it combines the comparative speed and ease-of-use of the algorithm with the power of multiple imputation, to let you focus on your substantive research questions rather than spending time developing complex application-specific models for non-response in each new data set. Unless the rate of missingness is exceptionally high, m=5 (default) will usually be adequate. Other methods of dealing with missing data, such as listwise deletion, mean substitution, or single imputation, are in common circumstances biased, inefficient, or both. When multiple imputation works properly, it fills in data in such a way as to not change any relationships in the data but

which enables the inclusion of all the observed data in the partially missing rows. Amelia II is a newer version of program, and follows same purpose as the first version of Amelia by James Honaker, Anne Joseph, Gary King, Kenneth Scheve, and Naunihal Singh [17].

Sample R code for generating sample missing data.

```
library(missForest)
library(mice)
MyDataInput <- read.csv(file="data-education-50-r.csv", header=TRUE, sep=",")
MyDataInput.mis <- prodNA(MyDataInput, noNA = 0.1)
print("Create data-education-50-r-mis.csv with missing data")
write.csv(MyDataInput.mis,'data-education-50-r-mis.csv')
```

R code for finding missing data.

```
library(Amelia)
MissingData<-read.csv(file='data-education-50-r-mis.csv')
imputed_Data <- amelia(MissingData,m=5, parallel = "multicore", noms = "Species")
print(imputed_Data)
print("Save data after imputation to data-education-50-r-mis-imputation.csv")
write.amelia(imputed_Data, file.stem = "data-education-50-r-mis-imputation.csv")
```

5.7 missForrest

missForest is a nonparametric imputation method for basically any kind of data. It can cope with mixed-type of variables, nonlinear relations, complex interactions and high dimensionality ($p >> n$). It only requires the observation (i.e. the rows of the data frame supplied to the function) to be pairwise independent. The algorithm is based on random forest and is dependent on its R implementation randomForest by Andy Liaw and Matthew Wiener. Put simple (for those who have skipped the previous paragraph): for each variable

Table 5.4: Data for Amelia

gender	race	ethnicity	immigrant	highschool_age
1	7	8	2	14.5
0	2	2	0	14
0	1	1	0	14
1	6	6	2	13.8
0	1	1	0	14
0	1	1	0	14
1	5	6	2	13.8
0	2	2	0	14
1	5	6	2	13.8
0	3	3	1	14
0	2	2	0	14
1	6	6	2	13.8
0	4	4	1	14
1	6	7	2	15
1	5	6	1	13.8
0	4	4	1	14
0	3	3	0	14
1	5	5	1	14
0	2	3	0	14
0	2	3	0	14

Table 5.5: Sample Missing Data for Amelia

	gender	race	ethnicity	immigrant	highschool_age
1	1	7	8	2	14.5
2	0	2	2	0	14
3	NA	1	1	0	14
4	1	6	6	2	13.8
5	0	1	1	NA	14
6	NA	1	1	0	14
7	1	5	6	2	13.8
8	NA	2	2	0	14
9	1	5	6	2	13.8
10	0	3	3	NA	14
11	0	NA	NA	0	14
12	1	6	6	2	13.8
13	NA	4	4	1	14
14	1	6	7	2	NA
15	1	5	6	1	13.8
16	0	4	4	1	14
17	0	3	3	0	14
18	NA	5	NA	1	14
19	0	2	3	0	14
20	0	2	3	0	14

Table 5.6: Data after imputation for Amelia

	X	gender	race	ethnicity	immigrant	highschool_age
1	1	1	7	8	2	14.5
2	2	0	2	2	0	14
3	3	0.399867	1	1	0	14
4	4	1	6	6	2	13.8
5	5	0	1	1	0.291427	14
6	6	0.120429	1	1	0	14
7	7	1	5	6	2	13.8
8	8	-0.01615	2	2	0	14
9	9	1	5	6	2	13.8
10	10	0	3	3	0.890275	14
11	11	0	1.805109	1.731756	0	14
12	12	1	6	6	2	13.8
13	13	0.256045	4	4	1	14
14	14	1	6	7	2	14.17264
15	15	1	5	6	1	13.8
16	16	0	4	4	1	14
17	17	0	3	3	0	14
18	18	0.702625	5	4.300979	1	14
19	19	0	2	3	0	14
20	20	0	2	3	0	14

missForest ts a random forest on the observed part and then predicts the missing part. The algorithm continues to repeat these two steps until a stopping criterion is met or the user specied maximum of iterations is reached. For further details see Stekhoven and Buhlmann [18]. To understand the remainder of this user guide it is important to know that missForest is running iteratively, continuously updating the imputed matrix variable-wise, and is assessing its performance between iterations. This assessment is done by considering the difference(s) between the previous imputation result and the new imputation result. As soon as this difference (in case of one type of variable) or differences (in case of mixed-type of variables) increase the algorithm stops. missForest provides the user with an estimate of the imputation error. This estimate is based on the out-of-bag (OOB) error estimate of random forest. Stekhoven and Buhlmann [18] showed that this estimate produces an appropriate representation of the true imputation error.

Sample R code.

```
library(mice)
library(randomForest)
library(missForest)
MissingData<-read.csv(file='data-education-50-r-mis.csv')
imputed_Data <- missForest(MissingData)
print(imputed_Data)
print("Save data after imputation to data-education-50-r-mis-imputation.csv")
write.csv(imputed_Data$ximp,'data-education-50-r-mis-imputation.csv')
```

Table 5.7: Description of columns

1	gender
2	race
3	ethnicity
4	immigrant
5	highschoolage
6	highschool rank
7	dual credit
8	ap credit
9	combined sat
10	parent mother
11	parent father
12	gross income
13	degree completion

Table 5.8: Sample input data

0	1	1	0	14	2	0	0	0	1	1	37368	0
0	3	4	1	14	2	0	0	1	1	1	59473	1
1	6	7	2	15	3	1	1	2	3	3	88947	2
1	7	8	2	14.5	3	1	1	2	4	4	100000	2
1	6	7	2	15	3	1	1	2	3	3	92631	2
0	4	4	1	14	2	0	0	1	1	1	63157	1
0	3	3	1	14	2	0	0	1	1	1	55789	1
0	2	3	0	14	2	0	0	0	1	1	48421	0
1	7	8	2	14.5	3	1	1	2	4	4	96315	2
0	2	2	0	14	2	0	0	0	1	1	41052	0
0	1	1	0	14	1	0	0	0	1	1	30000	0
1	5	6	1	13.8	2	1	1	1	2	1	77894	1
1	5	6	2	13.8	2	1	1	2	2	2	81578	2
0	3	3	0	14	2	0	0	0	1	1	52105	0
1	5	5	1	14	2	1	1	1	2	1	74210	1
1	4	5	1	14	2	1	1	1	1	1	70526	1
1	6	7	2	15	3	1	1	2	2	2	85263	2
0	2	2	0	14	2	0	0	0	1	1	44736	0
1	4	5	1	14	2	1	1	1	1	1	66842	1
0	1	1	0	14	1	0	0	0	1	1	33684	0

Table 5.9: Sample data with missing values.

1	0	1	1	0	14	2	0	0	0	1	1	37368	0
2	0	NA	4	1	14	2	0	0	1	1	1	59473	1
3	1	6	7	2	15	3	1	1	2	3	3	88947	2
4	1	7	NA	2	14.5	3	1	1	2	4	4	100000	2
5	1	6	7	2	15	NA	1	NA	2	3	3	92631	2
6	0	4	4	1	14	2	NA	0	1	NA	1	63157	1
7	0	3	3	1	14	NA	0	0	NA	NA	1	55789	1
8	0	2	3	0	14	2	NA	0	0	NA	1	48421	0
9	1	7	8	2	14.5	3	NA	1	2	4	4	96315	2
10	0	NA	2	0	14	2	0	0	0	1	1	41052	0
11	0	1	1	0	14	1	0	0	0	1	1	30000	0
12	1	5	NA	1	13.8	2	1	1	1	2	1	77894	1
13	1	5	6	2	13.8	2	1	1	2	2	2	81578	2
14	0	3	3	0	14	NA	0	0	0	1	1	52105	0
15	1	5	5	1	NA	2	1	1	1	2	1	74210	1
16	1	4	5	1	14	NA	1	1	1	1	1	70526	1
17	1	6	7	2	15	3	1	1	2	2	2	85263	2
18	0	2	2	0	14	2	0	0	0	1	1	44736	0
19	1	4	5	1	14	NA	1	1	1	1	1	NA	1
20	0	1	1	0	14	1	NA	0	0	1	1	33684	0

Table 5.10: Data after imputation

1	1	0	1	1	0	14	2	0	0	0	1	1	37368	0
2	2	0	3.493	4	1	14	2	0	0	1	1	1	59473	1
3	3	1	6	7	2	15	3	1	1	2	3	3	88947	2
4	4	1	7	7.53	2	14.5	3	1	1	2	4	4	100000.0	2
5	5	1	6	7	2	15	2.933	1	1	2	3	3	92631	2
6	6	0	4	4	1	14	2	0.25	0	1	1.05	1	63157	1
7	7	0	3	3	1	14	1.997	0	0	0.553	1.04	1	55789	1
8	8	0	2	3	0	14	2	0	0	0	1.03	1	48421	0
9	9	1	7	8	2	14.5	3	1	1	2	4	4	96315	2
10	10	0	1.576	2	0	14	2	0	0	0	1	1	41052	0
11	11	0	1	1	0	14	1	0	0	0	1	1	30000	0
12	12	1	5	5.242	1	13.8	2	1	1	1	2	1	77894	1
13	13	1	5	6	2	13.8	2	1	1	2	2	2	81578	2
14	14	0	3	3	0	14	1.932	0	0	0	1	1	52105	0
15	15	1	5	5	1	13.941	2	1	1	1	2	1	74210	1
16	16	1	4	5	1	14	1.985	1	1	1	1	1	70526	1
17	17	1	6	7	2	15	3	1	1	2	2	2	85263	2
18	18	0	2	2	0	14	2	0	0	0	1	1	44736	0
19	19	1	4	5	1	14	1.965	1	1	1	1	1	70331.232	1
20	20	0	1	1	0	14	1	0.04	0	0	1	1	33684	0

Similar results it is possible to get by using different software packages. Appropriate code in R is presented in the next sections.

5.8 MI

mi package in R has several features that allows the user to get inside the imputation process and evaluate the reasonableness of the resulting models and imputations. These features include: choice of predictors, models, and transformations for chained imputation models; standard and binned residual plots for checking the fit of the conditional distributions used for imputation. With plots for comparing the distributions of observed and imputed data. In addition, mi uses Bayesian models and weakly informative prior distributions to construct more stable estimates of imputation models. The goal is to have a demonstration package that (a) avoids many of the practical problems that arise with existing multivariate imputation programs, and (b) demonstrates state-of-the-art diagnostics that can be applied more generally and can be incorporated into the software of others [19].

R code.

```
library(mi)
MissingData<-read.csv(file='data-education-50-r-mis.csv')
imputations  <- mi(MissingData)
summary(imputations )
mi2stata(imputations ,m=5, file='data-education-50-r-mis-imputation.csv')
```

Table 5.11: Sample data with missing values (10%) for MI.

1	0	2	2	0	14	2	0	0	0	1	1	44482	0
2	1	7	7	2	15	3	1	1	2	3	3	92758	2
3	0	1	1	0	14	1	0	0	0	1	1	30000	0
4	0	1	1	0	14	2	0	0	0	1	1	37241	0
5	1	5	6	2	13.8	2	1	1	2	2	1	78275	2
6	1	6	7	2	15	3	1	1	2	2	2	85517	2
7	1	6	7	2	15	3	1	1	2	3	3	87931	2
8	0	2	2	0	14	2	0	0	0	1	1	42068	0
9	1	NA	6	2	13.8	2	1	1	2	2	2	83103	2
10	0	1	2	0	14	2	0	0	0	1	1	39655	0
11	0	1	1	0	14	1	0	0	0	1	1	34827	0
12	0	3	3	1	14	2	0	0	1	1	1	54137	1
13	1	5	6	2	13.8	2	1	1	2	2	2	80689	2
14	0	2	2	0	14	2	0	0	0	1	1	46896	0
15	0	4	4	1	14	2	0	0	1	1	1	63793	1
16	0	3	4	1	14	2	0	0	1	1	1	NA	1
17	1	5	6	1	13.8	2	1	1	1	2	1	75862	1
18	1	7	8	2	14.5	3	1	1	2	4	4	97586	2
19	0	4	4	1	14	2	0	0	1	1	1	61379	1
20	1	4	5	1	14	2	1	1	1	1	1	66206	1
21	1	5	5	1	14	2	1	1	1	2	1	73448	1
22	1	7	8	2	14.5	3	1	1	2	4	4	100000	2
23	0	3	3	1	14	2	0	0	1	1	1	56551	1
24	0	2	3	0	14	2	0	0	0	1	1	49310	0
25	1	4	5	1	14	2	1	1	1	1	1	71034	1

Table 5.12: Sample data after imputation (10%) for MI.

0	2	2	0	14	2	0	0	0	1	1	44482	0
1	7	7	2	15	3	1	1	2	3	3	92758	2
0	1	1	0	14	1	0	0	0	1	1	30000	0
0	1	1	0	14	2	0	0	0	1	1	37241	0
1	5	6	2	13.8	2	1	1	2	2	1	78275	2
1	6	7	2	15	3	1	1	2	2	2	85517	2
1	6	7	2	15	3	1	1	2	3	3	87931	2
0	2	2	0	14	2	0	0	0	1	1	42068	0
1	6	6	2	13.8	2	1	1	2	2	2	83103	2
0	1	2	0	14	2	0	0	0	1	1	39655	0
0	1	1	0	14	1	0	0	0	1	1	34827	0
0	3	3	1	14	2	0	0	1	1	1	54137	1
1	5	6	2	13.8	2	1	1	2	2	2	80689	2
0	2	2	0	14	2	0	0	0	1	1	46896	0
0	4	4	1	14	2	0	0	1	1	1	63793	1
0	3	4	1	14	2	0	0	1	1	1	58965	1
1	5	6	1	13.8	2	1	1	1	2	1	75862	1
1	7	8	2	14.5	3	1	1	2	4	4	97586	2
0	4	4	1	14	2	0	0	1	1	1	61379	1
1	4	5	1	14	2	1	1	1	1	1	66206	1
1	5	5	1	14	2	1	1	1	2	1	73448	1
1	7	8	2	14.5	3	1	1	2	4	4	100000	2
0	3	3	1	14	2	0	0	1	1	1	56551	1
0	2	3	0	14	2	0	0	0	1	1	49310	0
1	4	5	1	14	2	1	1	1	1	1	71034	1

5.9 Finding Missing Data with Neural Network

Due to the great interest in deep learning in the last decade, it is especially important to establish unified tools for practitioners to process missing data with arbitrary neural networks. Artificial neural networks use non-linear mathematical equations to successively develop meaningful relationships between input and output variables through a learning process. We applied back propagation networks to classify data along with other optimization methods. The structure of back propagation networks is typically composed of an input layer, one or more hidden layers, and an output layer, each consisting of several neurons. ANNs can easily handle the non-linear and interactive effects of explanatory variables. The major drawback of ANNs is they cannot result in a simple probabilistic formula of classification. Artificial neural networks use non-linear mathematical equations to successively develop meaningful relationships between input and output variables through a learning process.

Theoretically the methodology for feeding neural networks with missing data, by having the model uncertainty on missing attributes by probability density functions, which eliminates the need of direct completion (imputation) by single values.

5.9.1 Layer for processing missing data

The methodology for feeding neural networks with missing data, representation of missing data point is denoted by (x, J), where $x \in \mathbb{R}^D$ and $J \subset \{1, ..., D\}$ the set of attributes with missing values. Upon each missing point (x, J) we associate the subspace consisting all of the points which coincide with x on known coordinates $J^{'} = \{1, ..., N\}/J$:

$$S = Aff[x, J] = x + span(e_J),$$

where $e_J = [e_j]_{j \in J}$ and e_j is the j^{th} vector in $\mathbb{R}^D$.

Assumption made is that the values that at the missing attributes come from the un-

known D-dimensional probability distribution F. Then we can model the unobserved values of (x, J) by restricting F to subspace $S = Aff[x, J]$. Now it is possible that the values of incomplete data point (x, J) are described by the conditional density function [20]

$$F_S : S \to \mathbb{R}$$

given by:

$$F_S(x) = \begin{cases} \frac{1}{\int F(s)ds} F(x), & for\ x \in S, \\ 0, & otherwise. \end{cases}$$

5.9.2 Predicting One Missing Variable with Neural Network

To process probability density functions (representing missing data points) by neural networks, we generalize the neurons activation function. We define the generalized response (activation) of a neuron $n : \mathbb{R}^D \to \mathbb{R}$ on F_S as the mean output:

$$n(F_S) = E[n(x)|x \sim F_S] = \int n(x) F_S(x) dx.$$

From the neurons response we move back to same step size as before first layer while the rest of network architecture can remain unchanged. Basic requirement is the ability of computing expected value with respect to F_S.

Recall that the ReLU neuron is given by,

$$ReLU_{w,b}(x) = max(w^T x + b, 0),$$

where $w \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are the bias.

With TRUE vales being $= 0, 0, 1, 1, 1$. As we can see our neural network was able to forecast to the closest value of given TRUE value to data. TensorFlow code is provided,

below.

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.utils import to_categorical

#read in training data
train_df1 = pd.read_csv('data-education-1000-r.csv')
train_df2 = pd.read_csv('data-education-1000-r-predict.csv')

#create a dataframe with all training data except the target column
train_X = train_df1.drop(columns=['degree_completion'])
train_X_Predict = train_df2.drop(columns=['degree_completion'])

#create a dataframe with only the target column
train_y = train_df1[['degree_completion']]

#create model
model = Sequential()

#get number of columns in training data
n_cols_x = train_X.shape[1]
n_cols_y = train_y.shape[1]

#add model layers
model.add(Dense(10, activation='relu', input_shape=(n_cols_x,)))
model.add(Dense(10, activation='relu'))
```

```
model.add(Dense(n_cols_y))

#compile model using mse as a measure of model performance
model.compile(optimizer='adam', loss='mean_squared_error')

early_stopping_monitor = EarlyStopping(patience=3)

#train model
model.fit(train_X, train_y,epochs=2000, verbose=1)

print("prediction")
print(model.predict(train_X_Predict))
```

Summary of the accuracy of prediction is given below.

100 epochs

Number of Errors=476

Number of rows=1000

Number of correct answers in percent=52.4

Number of errors in percent=47.6

200 epochs

number of Errors=184

Number of rows=1000

Number of correct answers in percent=81.6

Number of errors in percent=18.4

500 epochs

Number of Errors=100

Number of rows=1000

Number of correct answers in percent=90.0

Number of errors in percent=10.0

1000 epochs

Number of Errors=0

Number of rows=1000

Number of correct answers in percent=100.0

Number of errors in percent=0.0

Chapter 6

Conclusions

6.1 Neural Networks

I've presented applications of neural networks, a modern machine learning method, to the problem of prediction in customer credit card default, with 9 different countries stock market and the SP500. With our last topic consisted of imputation methods and finding the missing values with the use of Neural Networks. The results are promising, but could still be improved. One important lesson learned from the preliminary experiments is that preprocessing of the data is crucial for obtaining good results. The results also contain some findings which are counter-intuitive and should be investigated further. There are many avenues for future work. The most obvious one is to use more than one previous data point in the prediction, or even a bigger observation made the bigger our data-set is the more usage we seem to have over the training and forecasting stage with application of different methodologies; to those that are increasing along this topic of forecasting data. All of this should help improve performance of models. The experiment could be improved by including stocks not based on their individual performance, but based on their ability to fit the residual error remaining from the other stocks. More data preprocessing (e.g. smoothing) could help. We note that a different way to use recurrent neural network, Long-short term memory (LSTM), would be as a layer on top of a regular financial model. For example, an RNN could use the parameters of a linear autoregression model as inputs, rather than raw data, and predict a non-linear effect on the future data, promising work.

As for general approach for adapting neural networks to process incomplete data, which

is able to train on data set containing only incomplete samples. For our strategy in introducing input layer for processing missing data, which can be used for a wide range of networks and does not require their extensive modifications. The experiments confirm its practical usefulness in various tasks and for diverse network architectures. In particular, it gives comparable results to the methods, which require complete data in training. With test effectiveness of importance on testing methods presented in this thesis, I generated test data with different properties. It is also possible to predict missing data by using different mathematical techniques along with statistical tools, another observation made on how this field of work is in high interest of research. For I effectively used existing statistical software for missing data (MICE, missForest, MI, and Amelia). Due to short time, unfortunately was not able to fully get PCA working, more complex algorithm on incomplete data. The PCA method shows to be a promising approach for a large scale of missing data.

As research shows that in order to predict missing data it is possible to apply machine learning techniques (neural networks). Which on my coming I presented this via neural networks with the TensorFlow, and with appropriate numerical results. The presented algorithms in work are not limited to presented applications. Many algorithms can be extended to different problems with hidden information or to complete data when it comes to our forecasting methods.

6.2 Future plans

In the future, I plan to prepare extensive report which present effectiveness of different existing statistical tools (MICE, missForest, MI etc.) for forecasting missing data, including appropriate visualizations. Hidden information is especially important in cyber security, modeling of uncertainty, in the negotiations, military problems, and safety analysis. Continuing to work on different neural networks and other machine learning techniques for forecasting data and missing data. As experienced first hand at Los Alamos National Lab. the growth on this Artificial Intelligence \ Deep learning there is great affirmation that

many may be applied to multivariate time series data, along with incompleteness or missing values. I will integrated all of my work done in a special computational framework, which allows one to do very large scale research together with appropriate documentation.

Chapter 7

Appendix

7.1 The Iris Classification Problem by Using Tensorflow

Code below is based on the following reference [12].

```
from __future__ import absolute_import, division, print_function
import os
import matplotlib.pyplot as plt
import tensorflow as tf

tf.enable_eager_execution()
print("TensorFlow version: {}".format(tf.__version__))
print("Eager execution: {}".format(tf.executing_eagerly()))
# column order in CSV file
column_names = ['sepal_length', 'sepal_width', 'petal_length',
      'petal_width', 'species']
feature_names = column_names[:-1]
label_name = column_names[-1]
print("Features: {}".format(feature_names))
print("Label: {}".format(label_name))

train_dataset_url = "https://storage.googleapis.com/download.
```

```
      tensorflow.org/data/iris_training.csv"
train_dataset_fp = tf.keras.utils.get_file(
      fname=os.path.basename(train_dataset_url),
                                          origin=train_dataset_url)
print("Local copy of the dataset file: {}".format(train_dataset_fp))
class_names = ['Iris setosa', 'Iris versicolor', 'Iris virginica']
batch_size = 32
train_dataset = tf.data.experimental.make_csv_dataset(
    train_dataset_fp,
    batch_size,
    column_names=column_names,
    label_name=label_name,
    num_epochs=1)
features, labels = next(iter(train_dataset))
print(labels)
print(features)
plt.scatter(features['petal_length'].numpy(),
            features['sepal_length'].numpy(),
            c=labels.numpy(),
            cmap='viridis')
plt.xlabel("Petal length")
plt.ylabel("Sepal length");
def pack_features_vector(features, labels):
"""Pack the features into a single array."""
features = tf.stack(list(features.values()), axis=1)
return features, labels

train_dataset = train_dataset.map(pack_features_vector)
```

```
features, labels = next(iter(train_dataset))
print(features[:5])
model = tf.keras.Sequential([
  tf.keras.layers.Dense(10, activation=tf.nn.relu, input_shape=(4,)),
  tf.keras.layers.Dense(10, activation=tf.nn.relu),
  tf.keras.layers.Dense(3)
])
predictions = model(features)
print(predictions[:5])
print(tf.nn.softmax(predictions[:5]))
print("Prediction: {}".format(tf.argmax(predictions, axis=1)))
print("    Labels: {}".format(labels))
def loss(model, x, y):
y_ = model(x)
return tf.losses.sparse_softmax_cross_entropy(labels=y, logits=y_)
l = loss(model, features, labels)
print("Loss test: {}".format(l))
def grad(model, inputs, targets):
with tf.GradientTape() as tape:
loss_value = loss(model, inputs, targets)
return loss_value, tape.gradient(loss_value, model.trainable_variables)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
global_step = tf.Variable(0)
loss_value, grads = grad(model, features, labels)
print("Step: {}, Initial Loss: {}".format(global_step.numpy(),
                                          loss_value.numpy()))
optimizer.apply_gradients(zip(grads, model.trainable_variables), global_step)
print("Step: {},         Loss: {}".format(global_step.numpy(),
```

```
                                                loss(model, features, labels).numpy()))
# Note: Rerunning this cell uses the same model variables
from tensorflow import contrib
tfe = contrib.eager
# keep results for plotting
train_loss_results = []
train_accuracy_results = []
num_epochs = 201
for epoch in range(num_epochs):
epoch_loss_avg = tfe.metrics.Mean()
epoch_accuracy = tfe.metrics.Accuracy()
# Training loop - using batches of 32
for x, y in train_dataset:
# Optimize the model
loss_value, grads = grad(model, x, y)
optimizer.apply_gradients(zip(grads, model.trainable_variables),global_step)
# Track progress
epoch_loss_avg(loss_value)  # add current batch loss
# compare predicted label to actual label
epoch_accuracy(tf.argmax(model(x), axis=1, output_type=tf.int32), y)
# end epoch
train_loss_results.append(epoch_loss_avg.result())
train_accuracy_results.append(epoch_accuracy.result())
if epoch % 50 == 0:
print("Epoch {:03d}: Loss: {:.3f}, Accuracy: {:.3%}".format(epoch,
epoch_loss_avg.result(),
epoch_accuracy.result()))
fig, axes = plt.subplots(2, sharex=True, figsize=(12, 8))
```

```
fig.suptitle('Training Metrics')
axes[0].set_ylabel("Loss", fontsize=14)
axes[0].plot(train_loss_results)
axes[1].set_ylabel("Accuracy", fontsize=14)
axes[1].set_xlabel("Epoch", fontsize=14)
axes[1].plot(train_accuracy_results);
test_url = "https://storage.googleapis.com/download.
      tensorflow.org/data/iris_test.csv"
test_fp = tf.keras.utils.get_file(fname=os.path.basename(test_url),
                                  origin=test_url)
test_dataset = tf.data.experimental.make_csv_dataset(
    test_fp,
    batch_size,
    column_names=column_names,
    label_name='species',
    num_epochs=1,
    shuffle=False)
test_dataset = test_dataset.map(pack_features_vector)
test_accuracy = tfe.metrics.Accuracy()
for (x, y) in test_dataset:
  logits = model(x)
  prediction = tf.argmax(logits, axis=1, output_type=tf.int32)
  test_accuracy(prediction, y)
print("Test set accuracy: {:.3%}".format(test_accuracy.result()))
tf.stack([y,prediction],axis=1)
predict_dataset = tf.convert_to_tensor([
    [5.1, 3.3, 1.7, 0.5,],
    [5.9, 3.0, 4.2, 1.5,],
```

```
    [6.9, 3.1, 5.4, 2.1]
])
predictions = model(predict_dataset)
for i, logits in enumerate(predictions):
class_idx = tf.argmax(logits).numpy()
p = tf.nn.softmax(logits)[class_idx]
name = class_names[class_idx]
print("Example {} prediction: {} ({:4.1f}%)".format(i, name, 100*p))
```

References

[1] H. Yang, I. King, L. Chan, and K. Huang, "Financial time series prediction using non-fixed and asymmetrical margin setting with momentum in support vector regression," *Neural Information Processing: Research and Development (Studies in Fuzziness and Soft Computing)*, vol. 152, pp. 334–350, 2004.

[2] E. Fama, "The behavior of stock-market prices," *The Journal of Business*, vol. 38, no. 1, pp. 34–105, 1965. [Online]. Available: http://www.jstor.org/stable/2350752

[3] E. Fama, L. Fisher, M. Jensen, and R. Roll, "The adjustment of stock prices to new information," *International Economic Review*, vol. 10, no. 1, pp. 1–21, 1969.

[4] E. Fama, "Effient capital markets: Ii," *The Journal of Finance*, vol. 46, no. 5, pp. 1575–1617, 1991.

[5] ——, "Random walks in stock market prices," *Financial Analysts Journal*, vol. 51, no. 1, pp. 75–80, 1995.

[6] R. David, "Machine-learning classification techniques for the analysis and prediction of high-frequency stock direction," Ph.D. dissertation, University of Iowa, 2014.

[7] G. Box, G. M. Jenkins, and G. C. Reinsell, *Time Series Analysis: Forecasting and Control, 3rd ed.* Englewood Cliffs, 1994.

[8] M. C. Mariani, B. M. A. Masum, and O. K. Tweneboah, "Estimation of stochastic volatility by using ornstein-uhlenbeck type models," *Physica A: Statistical Mechanics and its Applications*, vol. 491, pp. 167–176, 2018.

[9] E. Zivot and J. Wang, *Modeling Financial Time Series with S_PLUS. 2nd ed.* Springer Science+Business Media, 2006.

[10] Rolling-window analysis of time-series models. [Online]. Available: https://www.mathworks.com/help/econ/rolling-window-estimation-of-state-space-models.html

[11] Math works, introducing machine learning. [Online]. Available: https://www.mathworks.com

[12] The iris classification problem. [Online]. Available: https://www.tensorflow.org/tutorials/eager/custom_training_walkthrough

[13] I. Yeh and C. Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert Systems with Applications*, vol. 36, no. 2, pp. 2473–2480, 2009.

[14] S. Chen and H. He, "Stock prediction using convolutional neural networks," in *IOP Conference Series : Materials Science and Engineering*, 2018.

[15] I. Jahan, "Stock price prediction using recurrent neural networks," Master's thesis, North Dakota State University of Agriculture and Applied Sciences, 2018.

[16] S. Buuren and K. Groothuis-Oudshoorn, "mice: Multivariate imputation by chained equations in r," *Journal of Statistical Software*, vol. 45, no. 3, 2011.

[17] J. Honaker, G. King, and M. Blackwell. Amelia ii: A program for missing data. [Online]. Available: https://gking.harvard.edu/amelia

[18] D. Stekhoven and P. Buehlmann, "Missforest - nonparametric missing value imputation for mixed-type data," *Bioinformatics*, vol. 28, no. 1, pp. 112–118, 2012.

[19] Y.-S. Su, A. Gelman, J. Hill, and M. Yajima, "Multiple imputation with diagnostics (mi) in r: Opening windows into the black box," *Journal of Statistical Software*, vol. 45, no. 2, pp. 1–31, 2011.

[20] M. Smieja, L. Struski, J. Tabor, B. Zielinski, and P. Sputnik, "Processing of missing data by neural networks," in *32nd Conference on Neural Information Processing Systems*, NeurIPS 2018, Montral, Canada, 2018.

Curriculum Vitae

Jazmin Quezada an El Paso, Texas native received a Bachelor's and Master's degree in Mathematics from University of Texas at El Paso. Her scientific interests are related to Machine Learning \AI and forecasting of missing and hidden data by using different machine learning methodologies. She presented her scientific work on several scientific workshops. In June 2019, participation in a internship at the Los Alamos National Laboratories.