Loan Default Prediction

A Project Report Submitted for the partial fulfillment of the Degree of Bachelor of Technology in Information Technology

Under

Maulana Abul Kalam Azad University of Technology, West Bengal

(Formerly known as West Bengal University of Technology)

Project submitted by

Rohit Prasad

Roll Number: 10400217047, Registration Number: 171040110280

Under the Guidance of:

Prof. Dr. Satyasaran Changdar

Department of Information Technology

Institute of Engineering & Management, Kolkata

Gurukul, Y-12, Block – EP, Sector -V, Salt Lake Electronics Complex,

Kolkata – 700091, West Bengal, India

JULY, 2021

A PROJECT REPORT ON

Loan Default Prediction

Presented to the faculty of Engineering,

Institute of Engineering & Management, Kolkata.

In partial fulfilment of requirements for the degree of

B.Tech (I.T. Engineering)

Submitted By,

Rohit Prasad

ROLL NO: 10400217047

Under the Guidance of

Prof. Dr. Satyasaran Changdar

Department of Information Technology, IEM

PREFACE

The thesis has been submitted towards the partial fulfilment of the requirement for the **AWARD** of the **DEGREE of BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY** of **INSTITUTE OF ENGINEERING & MANAGEMENT, MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY, KOLKATA.**

Dated: JULY, 2021.

Place: Salt Lake, Kolkata . . . . . .

(ROHIT PRASAD)

Declaration of Originality and Compliance of Academic Ethics

We hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of **Bachelor of Technology in Information Technology**.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name **: Rohit Prasad**

Exam Roll Number **:** 10400217047

Thesis Title **:** Loan Default Prediction

Date : July, 2021. .....

(ROHIT PRASAD)

CERTIFICATE OF RECOMMENDATION

We hereby recommend that the thesis entitled **"Loan Default Prediction"** prepared under my supervision and guidance by **Rohit Prasad** be accepted in partial fulfilment of the requirement for awarding the degree of **B.Tech** in **Information Technology** from **Institute of Engineering and Management, Kolkata, India**. The project, in our opinion, is worthy for its acceptance.

_____

PROF. Dr. SATYASARAN CHANGDAR

(THESIS SUPERVISOR)
Department of Information Technology, IEM, Kolkata, India.

COUNTER SIGNED BY

_____

PROF. DR. INDRANEEL MUKHOPADHYAY

(HEAD OF THE DEPARTMENT)
Department of Information Technology, IEM, Kolkata, India.

COUNTER SIGNED BY

_____

PROF. DR. SATYAJIT CHAKRABARTI

(DIRECTOR)
IEM, Kolkata, India.

ACKNOWLEDGEMENT

Every project big or small is successful largely due to the effort of a number of wonderful people who have always given their valuable advice or lent a helping hand. I sincerely appreciate the inspiration; support and guidance of all those people who have been instrumental in making this project a success.

I, **Rohit Prasad**, student of **Institute of Engineering and Management (IT Department)**, is extremely grateful to mentors for the confidence bestowed in me and entrusting my project entitled **"Loan Default Prediction"** with special reference.

At this juncture,I feel deeply honoured in expressing my sincere thanks to **Prof. Dr. Satyasaran Changdar** for making the resources available at right time and providing valuable insights leading to the successful completion of our project who even assisted me in completing the project.

I would also like to thank all the faculty members of **IEM,Kolkata** for their critical advice and guidance without which this project would not have been possible.

Last but not the least I place a deep sense of gratitude to my family members and friends who have been constant source of inspiration during the preparation of this project work.

DATE – JULY,2021 **NAME – ROHIT PRASAD**

PLACE – Kolkata, India.

ABSTRACT

Lending loans can be an arduous affair for banks and their management, and so it becomes very important to automate the process of loan lending. Thus it is very much necessary to predict whether a loan will be returned or not. We are here trying to create a prediction model that will perform loan default prediction using the Artificial Neural Networks (ANN) technique of Machine Learning and compare it with the Logistic Regression technique. We trained our model on a pre-recorded dataset. Our main aim was to predict whether or not the borrower would default and we aimed to achieve best possible results.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

Abbreviation	Meaning
ANN	Artificial Neural Network
LR	Logistic Regression
ROC	Receiver Operating Characteristic
TN	True Negative
TP	True Positive
FN	False Negative
FP	False Positive
ML	Machine Learning
BGD	Batch Gradient Descent
MSE	Mean Squared Error

CHAPTER 1

Introduction

Without a doubt, financial lending services hold a great amount of significance for a person, business or enterprise. per se services are required by an individual or a business to attain or accomplish their goals and to compete with the giants of their fields. Financial loans are a significant part of the first source of capital not only in the emerging economies but also within the developed cap-ital markets by both individuals and enterprises. As the lending growth by the financial firms and also the banks are considered because the key factor for inflation level and interest rate of any country which drives its economic process and depicts its financial condition. consistent with the mission statement of the study within the role of financial services, the economic growth of the important economy is the primary role of the financial firms. With such great importance andbenefits of financial lending comes some major issues and bottleneck problems. The foremost common and substantial issue within the domain of financial lending is that the fair and successful lending of loans while keeping the ratio of loan defaulters to the smallest amount is of minimal value. within the financial lend-ing, the danger of loan defaulters can never be neutralized but can be minimized

1.1 General

Along with the financial market, the credit business of the banking industry is also rapidly expanding with extreme competition. At the same time, the consumption method by the consumer has turned the consumer loan into a major competitive market. According to banking laws, “Consumer loan” is defined as personal credits provided by the financial institution which is paid back by the method of installment payment and is generally provided for personal or family consumption, to pay certain expenses, such as medical cost, education cost, traveling cost or to pay some other accumulated debt taken with some other purpose of consumption.

1.2 Consumer Loan

.Loan lending can be a difficult process and so automation is required. If a borrower does not return the sum of money he borrowed it is called a default case and the person is called a defaulter and thus the name Loan Default Prediction.

With an increase in the number of Loans, the number of default cases also increases. So, we must create an effective and accurate model to automatically predict the default cases. We must be very careful with prediction because the number of default cases is far less than normal cases. So the only accuracy should not be our only concern.

According to the study, during the year of 2017 in India, only the bad loans crossed the threshold of about 207 billion dollars equaling the percentage of about 9.6 for loan Author to whom correspondence should be addressed defaulter ratio. While the greatest number of loan defaulter cases were registered in Italy making the total of 16.4%for the loan defaulters [4]. To cope up with the issue of high ratio loan defaulters' lot of work has been done

1.3 Background Information

As within the past few decades, the choice making for financial lending has been pretty much influenced by by data sharing and technological advancements. While employing various credit scoring models that include FICO Scoring Model, Vantage score Model, Credit expert Credit Score. The technique of credit scoring is to gauge different credit attributes by analyzation and classification to an individual and enterprise profile to assess the credit decision or to estimate the credit worthiness. Only credit scoring isn't sufficient for the financial lending thanks to such an enormous number of loan defaulters. Some financial expert's judgmental reviews must be accompanied subjectively. Financial analysts not only rely on the credit scores but also on their experience regarding the historical successful and unsuccessful cases as well for better higher cognitive processes. Moreover, with such tremendous growth of financial lending and to improve the credit defaulter ratio, advanced statistical methods were introduced to fill the gap of underperforming credit scoring models. These advanced statistical models like genetic

programming and neural networks provided the alternative from the previous traditional statistical models which supported the logistic regression and discriminant analysis.

In earlier days, this research was done to assess risks associated with banks and other different financial institutions with the help of Logical Regression, Decision tree and manually. Its main aim was to help banks and other financial institutions to select loan applicants so that their loss can be reduced. The techniques were general modeling techniques. These techniques are used when less complex extensions exist. In the case of more complex extensions, these techniques fail. The results produced by using these techniques are in the form of binary i.e., 0's and 1's.

The datasets were trained under certain parameters and based on that trained data were tested. The outcome of this testing was plotted and received in the form of a slope. This slope is basically in the form of a straight line following the equation $y=mx+c$. The accuracy of this system was not much accurate due to which this system sometimes gets failed to produce the correct result. Also, various models that are used are gene algorithms (GA). There is also various kind of non-parametric models which are based on artificial technology.

1.4 Domain Overview

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of Computer Programs that can change when exposed to new data.

Supervised Machine Learning is the majority of practical machine learning uses supervised learning. Supervised learning is where to have input variables (X) and an output variable (y) and use an algorithm to learn the mapping function from the input to the output is $y = f(X)$. The goal is to approximate the mapping function so well that when you have new input data (X) that you can predict the output variables (y) for that data. Techniques of Supervised Machine Learning algorithms include logistic regression, multi-class classification, Decision Trees and support vector machines, etc.

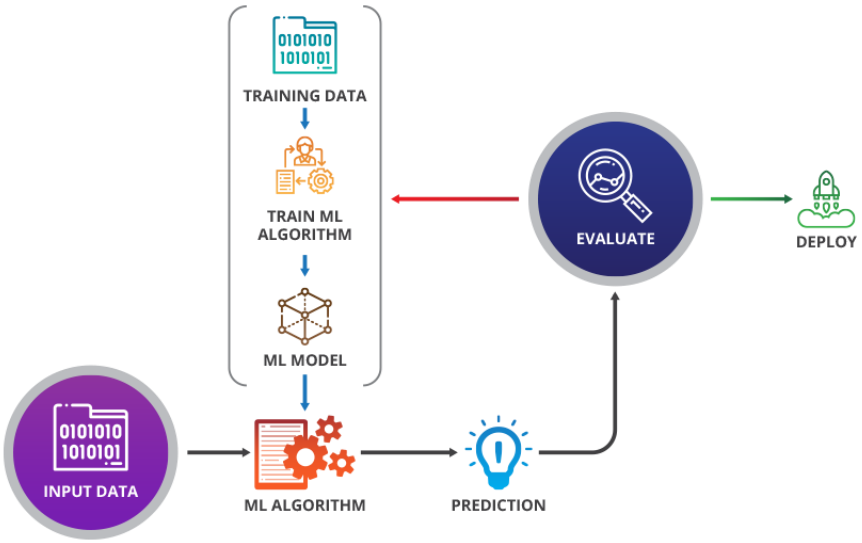


Figure 1. ML working

1.4.1 Artificial Neural Network

ANN is inspired by the human neural networks, wherein various interconnected neurons carry impulses to the brain. ANN uses activation units and nodes to reach a conclusion. The main purpose of ANN was to mimic the activity of the human brain. The human brain has several layers of neurons between the responding organs. These layers take electrical pulses as Input through Dendrites and process it according to the "algorithm" inside a Neuron and give output to Axon which takes it to the next layer. The function of the ANN is also the same. ANN is basically a modification of Regression techniques for solving more complex problems.

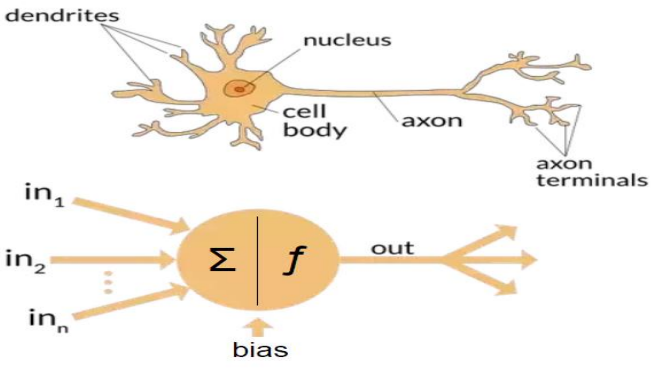


Figure 2. ANN similarity with human neuron

Neural Networks come in handy when we have a non-linear classification boundary, Logistic regression in these cases becomes too complex to apply.

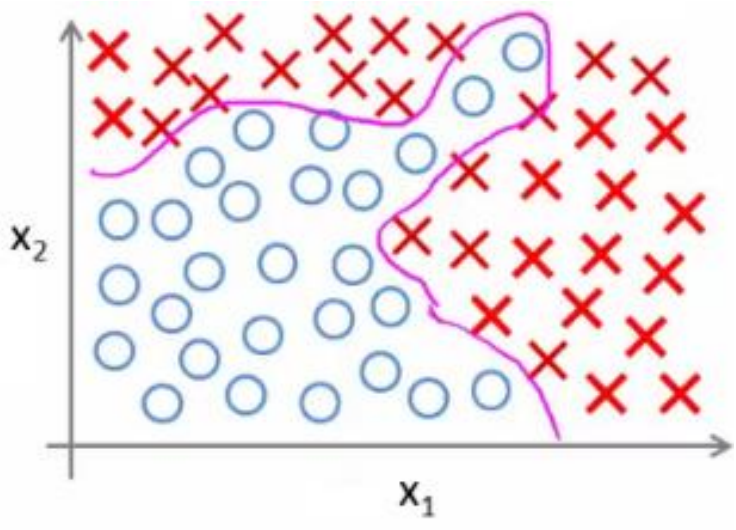


Figure 3. ANN model training

This happens mainly because it is –

1. It is infeasible to determine the polynomial variables.

2. Computational complexity also increases.

All these issues are handled by Neural Networks. With the advent of faster processors, the demand of ANN in model development is increasing rapidly.

Instead of passing the input variables into the logit function once, we repeatedly "feed-forward", the output of one layer into another just like biological neurons. The in-between layers are called "hidden" layers and the outermost layer is called the "output" layer.

The ANN consists of three steps:

1. Deciding the computation graph

2. Forward Propagation

3. Back Propagation

1.4.1.1 Forward propagation

aAs the name proposes, the information is taken care of the forward way through the system. Each shrouded layer acknowledges the info information, forms it according to the enactment capacity and goes to the progressive layer.

So as to create some yield, the information ought to be taken care of the forward way as it were. The information ought not stream backward heading during yield age else it would frame a cycle and the yield would never be produced. Such system arrangements are known as feed-forward system. The feed-forward system helps in forward engendering. At every neuron in a covered up or yield layer, the preparing occurs in two stages:

Preactivation: it is a weighted entirety of information sources for example the direct change of loads w.r.t to inputs accessible. In light of this amassed whole and actuation work the neuron settles on a choice whether to pass this data further or not.

Actuation: the determined weighted total of information sources is passed to the initiation work. An initiation work is a numerical capacity which adds non-linearity to the system. There are four regularly utilized and famous enactment capacities — sigmoid, hyperbolic tangent(tanh), ReLU and Softmax.

1.4.1.2 Back propagation

In AI, backpropagation could likewise be a generally utilized calculation in preparing feedforward neural systems for administered learning. Speculations of backpropagation exist for other counterfeit neural systems (ANNs), and for capacities by and large – a classification of calculations spoken conventionally as "backpropagation"In fitting a neural system, backpropagation processes the slope of the misfortune work with pertinence the loads of the system for one information yield model, and does so effectively, not at all like a guileless direct

calculation of the inclination with significance each weight exclusively. This proficiency makes it possible to utilize angle strategies for preparing multilayer systems, refreshing loads to constrict misfortune; inclination drop, or variations like stochastic slope plummet, are regularly utilized. The backpropagation calculation works by registering the inclination of the misfortune work with pertinence each weight by the chain rule, figuring the slope each layer in turn, emphasizing in reverse from the last layer to dodge repetitive computations of halfway terms inside the chain rule; this might be a case of dynamic programming

1.5 Motivation AND Problem Definition

Banks are essential bodies of the society which not only function a secure vault for storing money but also help people in time of need by providing loans and credit. Banks provide loans supporting the credit history and credibility of the applicant. However, the speed of loan default is increasing exponentially. per Forbes, say that just about 40 percent of borrowers are expected to neglect their student loans by 2023. Considering the statistics, rather than making money from loan interest, banks will suffer an enormous financial loss. In order the prevent the loss, it's important to own a system in situ which is able to accurately predict the loan defaulters even before approving the loan.

With the rise in “Consumer loan” provided by financial institutions, the scenario of loan default is additionally increasing. With excessive competition between different financial institutions to draw in more consumers, it became important to not deny the loan or just provide the loan to a consumer. To provide a loan, it's important to grasp whether the patrons are ready to pay back the provided loan or not. Although financial institutes provide “Consumer Loan” with extreme caution and after various affirmations, still there are some cases within which the consumer isn't ready to pay the loan back. These form of cases are what we loan default cases. These cases can vary from extremely minor cases to major cases leading to scams. a number of these cases may even lead the financial organization on the verge of bankruptcy. Thus, it became extremely important to predict whether a loan default scenario can occur with some particular case or not on the idea of previous similar quite consumer’s history whether or not they were defaulters or not.

1.6 Objective

Loans default will cause huge loss for the banks, in order that they pay much attention on this issue and apply various method to detect and predict default behaviours of their customers.In this Project, we are discussing the method of loan default prediction with machine learning algorithms using ANN.

CHAPTER 2

Literature Survey

This section tells about the previous works in Loan Default Prediction and their results.

Uzair Aslam, Hafiz Ilyas Tariq Aziz, Asim Sohail, and Nowshath Kadhar Batcha described Loan Default Prediction using Vector Machines. When compared to other classifiers it was noted in study that support vector machines outperformed most of them when it came to effectiveness and computational execution where large datasets with numerous explanatory variables were involved. They achieved the accuracy of about 81%. However, the major disadvantage highlighted in the mentioned paper is that when there are less than 10 explanatory attributes the accuracy of SVM is reduced, and other classifiers such as QDA, and Logistic Regression outperformed SVM.

Dominic M. Obarel, Gladys G. Njorogel and Moses M. Murayal. The purpose of this research was to analyze individual loan defaults in Kenya using the logistic regression model. The data used in this study was obtained from the equity bank of Kenya. A random sample of 1000 loan applicants whose loans had been approved by equity bank of Kenya during this period was obtained. Data obtained was on the credit history, purpose of the loan, loan amount, nature of the saving account, employment status, sex of the applicant, age of the applicant, security used when acquiring the loan and the area of residence of the applicant (rural or urban). This study employed a quantitative research design, it deals with individual loans defaults as group characteristics of a borrower. The data was pre-processed by seeding using R- Software and then split into training dataset and test data set. The train data was used to train the logistic regression model by employing Supervised machine learning. The model had an accuracy of 0.7727 with the train data and 0.7333 with the test data. The logistic regression model showed a precision of 0.8440 and 0.8244 with the train and test data respectively. The main disadvantage of this model was that it has a high false positives rate.

Hafiz Ilyas Tariq, Asim Sohail, Uzair Aslam, and Nowshath Kadhar Batcha [3], The purpose of this study was to provide a comprehensive research and to develop a model to predict the loan defaults. This study used methodologies named KDD, CRISP-DM and SEMMA. While in the experimentation phase, three different data mining techniques were applied for the proposed model and their performances were evaluated on various parameters. Based on these parameters, the best method was selected, explained and suggested because of its significant characteristics regarding the prediction of the loan defaults in the financial sector. This system has accuracy about 78% and it fails because its ROC score and area were not good.

Dr. LALIT MOHAN GOEL described about Neural Network approach to Loan Default prediction. In this paper, the author introduced an approach to implement a neural network model which will be used for loan default prediction. The prediction is performed by taking into consideration the financial and personal details provided by the potential debtor. The proposed neural network model was trained and tested using the dataset provided by the Lending Club bank. Before training the neural network, Principal Component Analysis was performed on the dataset in order to reduce the dimensionality of the dataset.

Charles Kwofie, Caleb Owusu-Ansah and Caleb Boadi, This study examines the performance of logistic regression in predicting probability of default using data from a microfinance company. A logistic regression analysis was conducted to predict default status of loan beneficiaries using 90 sampled beneficiaries for model building and 30 out of sample beneficiaries for prediction. Age, marital status, gender number of years of education, number of years in business and base capital were used as predictors. The predictors that were significant in the model were marital status, number of years in business and base capital. This sysem achieved accuracy of about 91% but the major disadvantage was that the explained variability in the response variable in the logistic regression was very weak.

Pidikiti Supriya , Myneedi Pavani , Nagarapu Saisushma [6] discussed about Loan Default prediction using machine learning models. Along with the financial market, the credit business of the banking industry is also rapidly expanding with extreme competition. At the same time,

the consumption method by the consumer has turned the consumer loan into a major competitive market due to which every day many people apply for loans, for a variety of purposes. But not all the applicants are reliable, and not everyone can be approved. Every year, there are cases where people do not repay the bulk of the loan amount to the bank which results in huge financial loss. The risk associated with making a decision on a loan approval is immense. In this paper, the author gathered loan data from the Lending Club website and use machine learning techniques on this data to extract important information and predict if a customer would be able to repay the loan or not. In other words, the goal is to predict if the customer would be a defaulter or not.

Semiu A, Akanmu Abdul Rehman Gilal, This Study developed a model for predicting Loan Default among social lending Patrons, specifically the small business owners, using boosted decision tree model. The United States small business administration (Usba) publicly available Loan Administration Dataset Of 27 features and 899164 data instances was used in 80:20 ratios for the training and testing of the model. This system recorded an accuracy of about 93%. The main disadvantage of this sytem was that it fails when there are complex data extensions and is much time taking model.

JAYADEV M and RAVI VADLAMANI, tried to train the model using machine learning and set the classifier as LSVM and performance metric as RMSE. The limitation was that it lost performance where there were less than 10 explanatory variables. JAYADEV M tried again to train the model using ANN. In this the classifier was Neural Network with accuracy was 93%. Its limitation was that it requiresd larger training sets to give accurate result. Also accuracy should not be the only deciding factor here.

HAN, CHOI AND KIM, set classifier as Logistic Regression with accuracy as 86%. Its limitation was that it lost performance in datasets with large number of variables due to on linearity relation between them.

Manjeet Kumar, Vishesh Goel, Tarun Jain, Sahil Singhal, Dr. Lalit Mohan Goel [10], tried to train the model using ANN, they used “Adam” optimizer, had 2 hidden layers of 20 nodes each and had set the number of epochs to 1000. They achieved a 93% accuracy but their model was overtrained and did not do well in test dataset.

Edinam Agbemava, Israel Kofi Nyarko, Thomas Clarkson Adade and Albert K. Bediako [11], The objective of this research is to predict loan defaults by customers in the microfinance sector and to develop a model that links these factors to credit default by customers in the sector. Data from a microfinance institution based in Accra Ghana was used. A binomial logistic regression analysis was fitted to a data of 548 customers who were granted credit from January 2013 to December 2014. The results of the study revealed that six factors: X3 (Marital Status); X7 (Dependents); X11 (Type of Collateral or Security); X13(Assessment); X15 (Duration); and X16 (Loan Type) were statistically significant in the prediction of loan default payment with a predicted default rate of 86.67%. To increase its accuracy, it is therefore suggested that microfinance institutions adopt among others, the default risk model to ascertain the level of risk since it's relatively efficient and cost effective. There should also be up to date training for loan officers of microfinance institutions in order to improve on their assessment skills and methodology.

CHAPTER 3

Proposed Methodology

3.1 Existing System

In the existing system, the method to assess risks associated with banks and other different financial institutions is done with the help of Logical Regression, Decision tree and manually. Its main aim was to help banks and other financial institutions to select loan applicants so that their loss can be reduced. The techniques were general modeling techniques. These techniques are reduced when less complex extensions exist. The results produced by using these techniques are in the form of binary i.e., 0's and 1's. The datasets were trained under certain parameters and based on the data, trained data were tested. The outcome of this testing was plotted and received in the form of a slope. This slope is basically in the form of a straight line following the equation y=mx+c. The accuracy of this system is not accurate due to which the result obtained from this system is not correct. Also, various models that are used are gene algorithms (GA). There are also various kinds of non-parametric models which are based on artificial technology.

3.1.1 Issues in Existing System

- The biggest issue in the existing system is its accuracy due to which it fails to produce correct results.
- This existing system fails when there are complex data extensions.
- Its ROC score is not much good and its ROC curve's area is not better than the ANN system.
- The false positives rate of the system increases due to which it sometimes produces false or incorrect results.
- The time taken by the system to produce the result is also very high.

3.2 Proposed System

We would divide the dataset into 2 parts, (1) to train data and (2) to validate the data. We would first prepare our data for training and then we would train it and then validate it. We first imported all the necessary libraries. We would use ANN for Loan Default Prediction in this project and compared it with the existing Logistic Regression Algorithm. We would try to show our comparison based on 3 factors: (a) Accuracy, (b) roc score and (c) roc curve for both training and testing.

The existing system is using machine learning algorithms like logistic regression, the decision tree model are slow and does not provide much accuracy. They lack in calculation with complex data. The system fails when the data is large and complex.

Our proposed methodology uses ANN (Artificial Neural Networks). This algorithm has high computational power, high speed and much more accuracy than the existing system. This can be used to perform large and complex calculations and deal with complex data.

3.2.1 Proposed System Architecture

In the architecture used in the prediction of the loan default.First, the data is gathered from different sources. These are known as raw data. They are gathered from three different sources. Data preprocessing and cleaning is done to remove unnecessary data and the void data. Data is converted into a standard form so that data can be processed further. Data is divided into two parts that is training data and test data. 70% of the data is used for training and 30% of the data is used for validation.

Model is trained using the set of training data. ANN algorithm is used to train the model. The training is done in three different layers. Data is tested to classify as defaulters or non-defaulters.

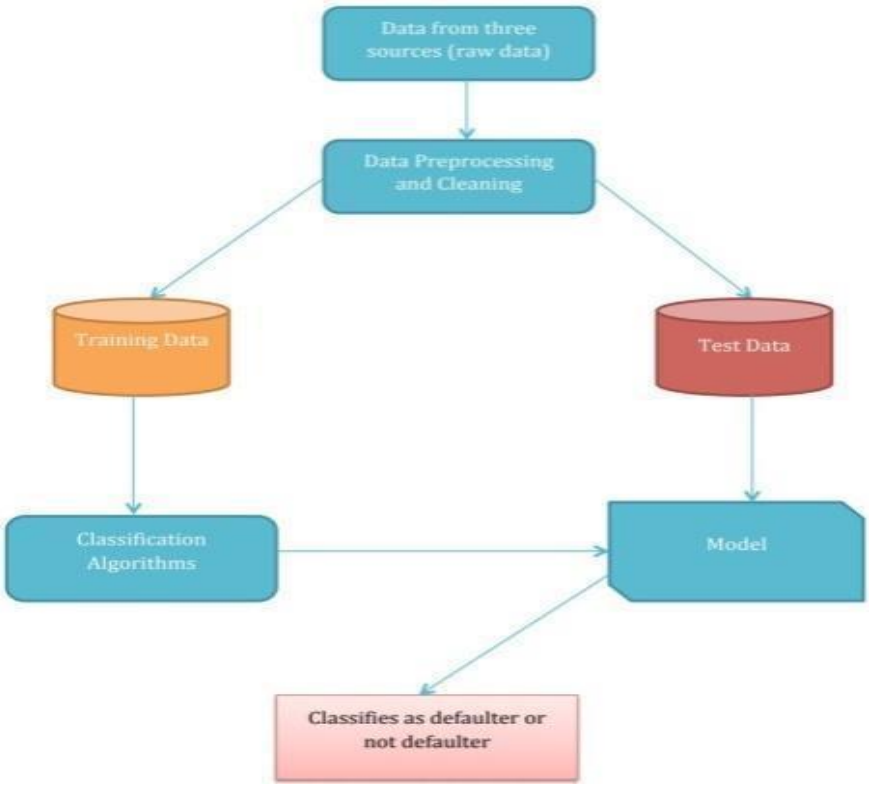


Figure 4. Proposed System architecture

3.2.2 Data Flow Diagram

The data set that we have is raw and must be prepared before fitting a model to it.

The raw data is first processed by label encoding the variables, and then alignment of test and train data is done. High mutually correlated variables are then removed and at last missing value treatment is done. The data is now prepared for a model. Now any model can be fit into this data.

First, we fit the LR model, calculate its accuracy and roc curve.Then, ANN model is fit in and we calculate the accuracy and roc curve.Their results are then compared.

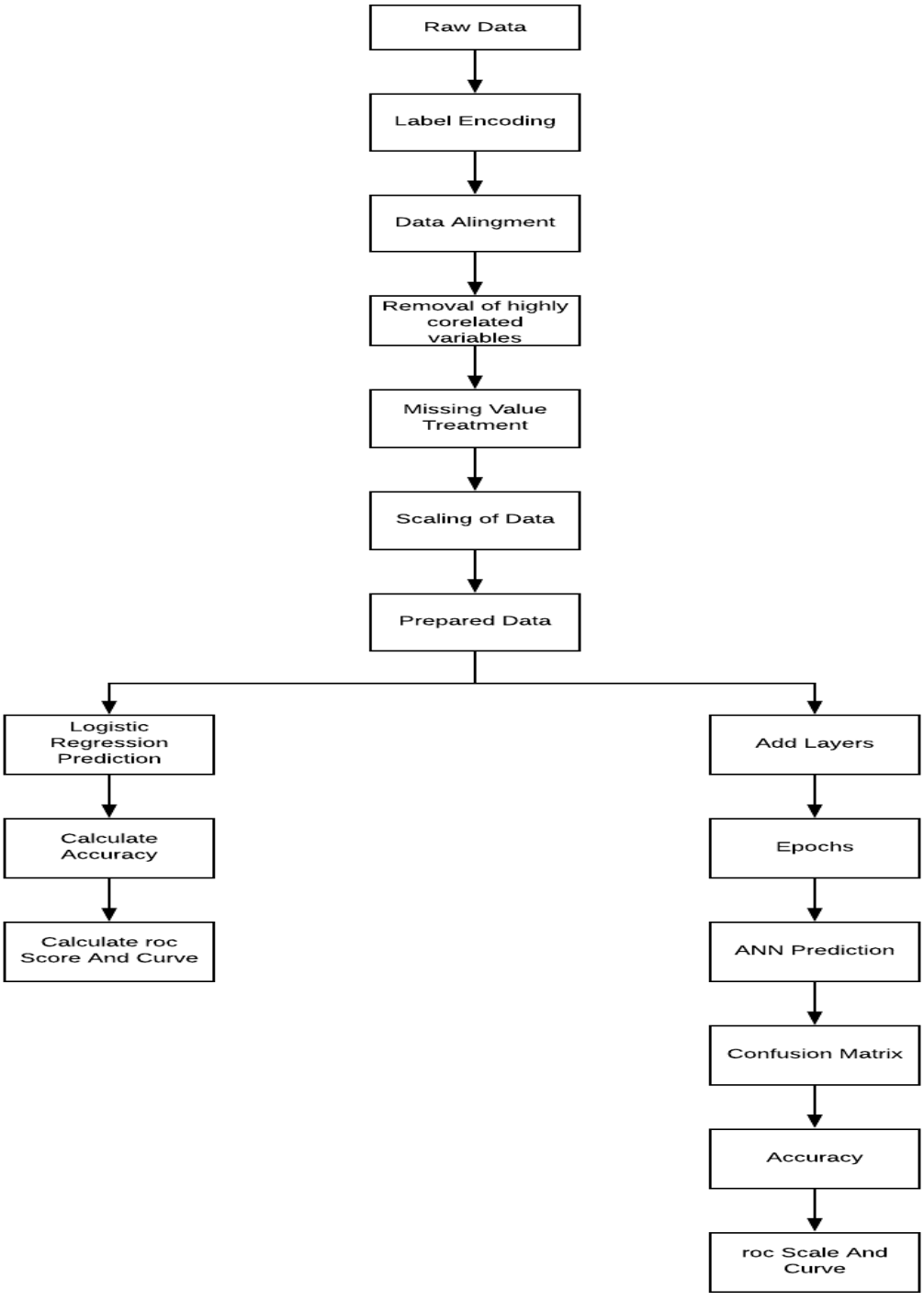


Figure 5.DATA FLOW DIAGRAM

CHAPTER 4

Requirement Analysis

4.1 Functional Requirements

- **Purpose**

  To predict the loan default. The system tells whether there are chances for loan default or not.

- **Inputs**

  Details of customer are provided as the input to the system. The parameters such as income, previous loan details, credit score etc are provided to the system as input.

- **Output**

  The system predicts the chances of loan default. It tells whether there are chances for loan default or not.

4.2 Non-Functional Requirements

- **Accuracy**

  This system provides more accuracy than any of the existing models. The output generated is accurate. The system can easily generate output for large and complex data.

- **Speed**

  System provides great speed for the computation of the data. Output generated is very fast. System works at high speed even for complex and large data.

- **Platform Independent**

  System works for any operating systems which provide ease of use to the users.

- **Usability**

  It provides ease of use. System is very user-friendly .It is less complicated to work on.

4.3 Hardware Requirements

- Computer system
- 16, 32 GB Ram
- Intel i3 and above processor
- Server Overhead
- Port 8000
- Internet connectivity

4.4 Software Requirements

- Notepad++ or any Code Editor
- Anacoda Navigator
- Jupyter Notebook
- MS-Excel
- Python
- MongoDB 2.6
- Client emvironment may be Windows or Linux
- Web Browser

CHAPTER 5

Modules and their Implementation

The model is divided into 3 main modules which are (1) Data Collection & Preparation, (2) Fitting model using LR and (3) Fitting model using ANN.

5.1 Data Collection & Preparation

After the collection of information from different loan applicants, what is collectively formed is some raw data that cannot be directly given to the model to be trained and tested. Instead, the data must thoroughly be checked and aligned in order to create symmetry between both the "test" as well as "train" data. A sample data is collected and worked upon here to be fit in the model. The data used here is a ".csv" file which is a tabular data. There are 2 sets of data which are used to train and test individually. The train and test data are named "Application_train" and "Application_test" respectively. The columns in these data represent the variables while the rows represent the details of the applicants. The train data has 307511 rows and 122 columns while the test data has 48744 rows and 122 columns. The train data has 1 extra column than in test data. The extra column is the "**target"** variable which is used for validation while training the data. The data is structured but unorganized and consists of values of different types including strings, integer and float. The values also range from very low to very high and also includes negative integers and float values.

SK_ID_C	NAME_C	CODE_G	FLAG_O	FLAG_O	CNT_CHI	AMT_INC	AMT_CR	AMT_AN	AMT_GC	NAME_T	NAME_IN	NAME_EI	NAME_F	NAME_H	REGION_	DAYS_B	DAYS_EI	DAYS_R	DAYS_IC	OWN_CA	FLAG_M	FLAG_EI	FLAG_W	FLAG_CI	FLAG_PI	FLAG_EI	OCCUPA	CNT_FAI	R
100001	Cash loa	F	N	Y	0	135000	568800	20561	450000	Unaccon	Working	Higher ec	Married	House / a	0.0189	-19241	-2329	-5170	-812		1	1	0	1	0	1		2	
100005	Cash loa	M	N	Y	0	99000	222768	17370	180000	Unaccon	Working	Seconda	Married	House / a	0.0358	-18064	-4469	-9118	-1623		1	1	0	1	0	0	Low-skill	2	
100013	Cash loa	M	Y	Y	0	202500	663264	69777	630000		Working	Higher ec	Married	House / a	0.0151	-20038	-4458	-2175	-3503	5	1	1	0	1	0	0	Drivers	2	
100028	Cash loa	F	N	Y	2	315000	2E+06	49019	2E+06	Unaccon	Working	Seconda	Married	House / a	0.0264	-13976	-1866	-2000	-4208		1	1	0	1	1	0	Sales sta	4	
100038	Cash loa	M	Y	N	1	180000	625500	32067	625500	Unaccon	Working	Seconda	Married	House / a	0.01	-13040	-2191	-4000	-4262	16	1	1	1	1	0	0		3	
100042	Cash loa	F	Y	Y	0	270000	959688	34601	810000	Unaccon	State ser	Seconda	Married	House / a	0.0252	-18604	-12009	-6116	-2027	10	1	1	0	1	1	0	Drivers	2	
100057	Cash loa	M	Y	Y	2	180000	499221	22118	373500	Unaccon	Working	Higher ec	Married	House / a	0.0228	-16685	-2580	-10125	-241	3	1	1	0	1	0	0	High skill	4	
100065	Cash loa	M	N	Y	0	166500	180000	14220	180000	Unaccon	Working	Higher ec	Single / n	With pare	0.0051	-9516	-1387	-5063	-2055		1	1	1	1	1	0	Core staf	1	
100066	Cash loa	F	N	Y	0	315000	364896	28958	315000	Unaccon	State ser	Higher ec	Married	House / a	0.0462	-12744	-1013	-1686	-3171		1	1	0	1	0	0	Core staf	2	
100067	Cash loa	F	Y	Y	1	162000	45000	5337	45000	Family	Working	Higher ec	Civil marri	House / a	0.0186	-10395	-2625	-8124	-3041	5	1	1	1	1	1	0	Sales sta	3	
100074	Cash loa	F	N	Y	0	67500	675000	25448	675000	Unaccon	Pensione	Seconda	Married	House / a	0.0031	-23670	365243	-7490	-4136		1	0	0	1	1	0		2	
100090	Cash loa	F	N	Y	0	135000	261621	16848	216000	Unaccon	Working	Seconda	Widow	House / a	0.008	-15524	-3555	-7833	-3985		1	1	0	1	0	0	Laborers	1	
100091	Cash loa	F	N	Y	0	247500	296280	23540	225000	Unaccon	Working	Seconda	Civil marri	House / a	0.0186	-12278	-329	-6031	-4586		1	1	0	1	0	0		2	
100092	Cash loa	F	Y	Y	0	90000	360000	18536	360000	Unaccon	Working	Seconda	Married	House / a	0.0145	-19687	-3578	-10673	-3183	3	1	1	0	1	0	0	Core staf	2	
100106	Revolving	M	N	Y	0	180000	157500	7875	157500	Spouse,	Working	Seconda	Civil marri	House / a	0.0313	-12091	-1830	-1042	-4221		1	1	0	0	0	0	Laborers	2	
100107	Cash loa	M	Y	Y	0	180000	296280	21690	225000	Unaccon	Working	Seconda	Civil marri	House / a	0.0326	-13563	-1007	-5719	-4044	14	1	1	0	1	0	1	Laborers	2	

Screenshot 1. (Sample Data Set)

This raw data must be organized before being fetched to the model, this is because, all machine learning algorithm are based on statistics and mathematics. This data contains string values that are not understood by the machine. Also highly unstable values and empty cells will affect the outcome of the prediction very largely.

So, to prepare this data, several data cleaning and preparation techniques as well as algorithms must be used. Treatment of missing values, deletion of empty columns and conversion string values to mathematical values must be done. Also, highly correlated values must be removed and alignment of test and train data must be done, so that they don't cause problem.

5.1.1 Label Encoding

Label encoding does convert the values of the variables from human-readable form to numeric form which is machine understandable. Label encoding converts categorical values to numerical values. It is part of scikit library in python. Label Encoding is of 2 types: (1)Integer Encoding and (2)One-Hot Encoding. We perform **"One-Hot Encoding"** on our data.

5.1.1.1 One-Hot Encoding

One-Hot encoding is part of label encoding where the categorical values are converted into numerical values without making any hierarchy. This makes it easier for the machine because hierarchy can affect the model predictions. One-hot encoding takes a column which has categorical data, which has been label encoded, and then splits the column into multiple columns. The numbers are replaced by 1s and 0s, depending on columns and values. For example, we have a column named color that has many values but only 3 unique values, Red, Green, and Blue. We'll get three new columns, one for each color — Red, Green, and Blue. For rows which have the first column value as Red, the 'Red' column will have a '1' and the other two columns will have '0's. Similarly, for rows which have the first column value as Green, the 'Green' column will have a '1' and the other two columns will have '0's.

color	color_red	color_blue	color_green
red	1	0	0
green	0	0	1
blue	0	1	0
red	1	0	0

Figure 6 (One-Hot Encoding)

One-Hot encoding is performed on our model to convert the categorical values to binary columns. The implementation is done using the **LabelEncoder()** function and **transform()** functions. We must also maintain the count of the variables that have been Label Encoded. This is also increase the number of columns in the data frame but will make our data machine understandable.

```
# Create a label encoder object
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le_count = 0

# Iterate through the columns
for col in train:
    if train[col].dtype == 'object':
        # If 2 or fewer unique categories
        if len(list(train[col].unique())) <= 2:
            # Train on the training data
            le.fit(train[col])
            # Transform both training and testing data
            train[col] = le.transform(train[col])
            # Keep track of how many columns were label encoded
            le_count += 1

print('%d columns were label encoded.' % le_count)
```

```
3 columns were label encoded.
```

Screenshot 2. (Label Encoding implementation)

5.1.2 Data Alignment

After one-hot encoding, the train data and test data frames must be aligned so that the column and rows have same configuration. This is done because our model which is trained on train data will be trained on a certain configuration of column and when it is tested, the result will not be effective if the configuration of columns in not same in test data as well.

Data Alignment is nothing but a sort of inner join of the axes in the data frame. After data alignment the rows and columns of both the test and training data frames will have same configuration and will maintain indices. This also includes, that if one of the columns is missing in any one of the data frames, it will add a column and fill any custom value that is given in the function in case of right and left joines.

TB1 - Left

	Product	Sales in millions	Profit	Store_location
1	Kitchen Utensils	25	10	Boston
2	Gardening	35	15	NYC

TB2 - Right

	Product	Sales in millions	Loss	Store_location
0	Kitchen Utensils	35	7	Somerville
1	Switches	35	10	Bridgewater
2	Monitors	70	8	Trenton

Figure 7 (A sample table before alignment)

Align - Inner Join and axis = 1(Columns)

TB1

Product	Sales in millions	Store_location
Kitchen Utensils	25	Boston
Gardening	35	NYC

TB2

Product	Sales in millions	Store_location
Kitchen Utensils	35	Somerville
Switches	35	Bridgewater
Monitors	70	Trenton

Figure 8. (Sample table after inner join alignment)

We will perform inner join in our data frame so that we don't have to deal with null or assumed value. We perform this operation using the align() function of the pandas library. We give the parameters of join type as "inner join" and axis as 1.

```
train_labels = train['TARGET']

# Align the training and testing data, keep only columns present in both dataframes
train, test = train.align(test, join = 'inner', axis = 1)

# Add the target back in
train['TARGET'] = train_labels

print('Training Features shape: ', train.shape)
print('Testing Features shape: ', test.shape)
```

```
Training Features shape:  (307511, 240)
Testing Features shape:  (48744, 239)
```

Screenshot 3. (Implementation of Data Alignment)

5.1.3 Removal of High Mutually Correlated Variables

Data correlation is nothing but a way to find out the relationship among multiple variables. Correlation can help understand which variables affect the model the most. Correlation is a common term in statistics. There are many methods to calculate the correlation.

The Formula used is:-

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

It is very important to know which variables are highly correlated with the "**Target**" variable because this will give an insight as to what are the variables that will affect the prediction the most. Correlation is of 3 types mainly: (1)Positive, (2)Negative and (3)Zero. In our case positive and negative correlations have the same effect on the model while Zero correlated variables have no effect on the model.

The correlation between the **target** variable and all other variables are calculated using the **corr()** function of pandas library. Then all the highly correlated variables (both with positive and negative correlation) must be separated to identify what are affecting the model the most. Then all mutual correlation must be found out among these variables.

```
correlations=train.corr()
```

```
print('\nPositive correlation\n')
print(correlations['TARGET'].sort_values(ascending=False).head(15))
print('\nNegative correlation\n')
print(correlations['TARGET'].sort_values(ascending=False).tail(15))
```

```
Positive correlation

TARGET                                                1.000000
DAYS_BIRTH                                            0.078239
DAYS_EMPLOYED                                         0.074958
REGION_RATING_CLIENT_W_CITY                           0.060893
REGION_RATING_CLIENT                                  0.058899
NAME_INCOME_TYPE_Working                              0.057481
DAYS_LAST_PHONE_CHANGE                                0.055218
CODE_GENDER_M                                         0.054713
DAYS_ID_PUBLISH                                       0.051457
REG_CITY_NOT_WORK_CITY                                0.050994
NAME_EDUCATION_TYPE_Secondary / secondary special     0.049824
FLAG_EMP_PHONE                                        0.045982
```

Screenshot 4. (Implementation of correlation with target variable)

If the variables have high mutual correlation, i.e, greater than 0.7(both positive and negative), then the variable that has lower correlation with the target variable must be dropped. This is done to stabilize the model and to avoid the problem called as “Multicollinearity”.

Multicollinearity happens when one variable in a multiple regression model can be linearly predicted from the others with a high degree of accuracy. This can lead to misleading results or skewed results.

A user-defined function named get_hi_corr() in implemented where the mutually correlated variables are treated and the variable with high mutual correlation and lower value of correlation with target variable is dropped. Correlation is not the best approach but it gives us an idea about the importance of that feature/ variable in the model.

```
def get_hi_corr(df,target):
    y=0.7
    lst=[]
    for i in range(2,121):
        for j in range (1,i):
            x=df.iloc[i][j]
            if(x>=y or x<=(-y)):
                m=np.abs(target[df.index[i]])
                n=np.abs(target[df.index[j]])
                print(m,n)
                if(m<=n):
                    if df.index[i] not in lst and df.index[j] not in lst:
                        lst.append(df.index[i])
                else:
                    if df.index[j] not in lst and df.index[i] not in lst:
                        lst.append(df.index[j])
                print(df.index[i],df.index[j],df.iloc[i][j])
    print(lst)
    return lst
```

```
#Dropping one of the variables with high correlation
label_to_drop=get_hi_corr(correlations,correlations['TARGET'])
```

Screenshot 5. (Implementation of dropping of mutually correlated variables)

5.1.4 Removal of any anomalies

With huge sets of data comes few anomalies. An anomaly is nothing but a defect or error in the data variables or their values. They can be several in the data set and it is important for them to be treated.

For example, consider a column named age where the age of the customer is saved, and by mistake the age of the person is filled in alphabetical form or any absurd value instead of numeric. So this is an anomaly and must be treated.

Anomalies are generally random and there is no such algorithm to treat them. So, manually one must discover all anomalies and remove them with proper logic.

```
anomaly_in_days=train[train['DAYS_EMPLOYED']==365243]
```

```
n=len(anomaly_in_days.index)
```

```
m=len(anomaly_in_days[anomaly_in_days['TARGET']==1].index)
```

```
(m/n)*100
```

```
5.399646043269405
```

```
no_anomaly=train[train['DAYS_EMPLOYED']!=365243]
```

```
n=len(no_anomaly.index)
m=len(no_anomaly[no_anomaly['TARGET']==1])
(m/n)*100
```

```
8.65997453765215
```

```
#This implies that anomalous days have lower precentage of default than non anomalous.
#We need to replace these values with NaN values
```

Screenshot 6. (Removal of Anomaly 1)

```
lst=[]
for i in range(0,198):
    lst.append(X.columns[i].replace(' ','_'))
lst2=[]
for i in range(0,198):
    lst2.append(lst[i].replace(':',''))
lst3=[]
for i in range(0,198):
    lst3.append(lst2[i].replace('/',''))
lst4=[]
for i in range(0,198):
    lst4.append(lst3[i].replace(',',''))
```

```
d={}
j=0
for i in X.columns:
    d[lst4[j]]=X[i]
    j=j+1
x=pd.DataFrame(d)
```

Screenshot 7 (Removal of Anomaly 2)

5.1.5 Missing Value Treatment

5.1.5.1 Deletion of variables that have maximum cells empty

Data collection can be a very difficult process and one of the major way of collecting data is through forms. Consumers fill up their details in designed forms and the data is collected through that process. Forms can contain various fields that become columns or variables in data sets. Due to many fields some of them may be unnecessary or absurd for some consumers. So their values remain empty in most of the cases. Unfortunately, these empty cells can affect the model and its accuracy. So these empty cells must be treated.

The variables that have more than 2/3rd of their cells as empty must be deleted in order to provide no threat to the model and its prediction accuracy.

```
train.count()[train.count()<307511]
```

```
train.drop(labels=['COMMONAREA_MEDI','NONLIVINGAPARTMENTS_AVG'],axis=1,inplace=True)
```

```
train.count()[train.count()<307511]
```

Screenshot 8. (Deletion of variables having maximum empty cells)

5.1.5.2 Imputation

Imputation is nothing but filling in the empty cells with assumed values. It is not the perfect but a very effective way of handling missing values. The main objective here is to employ known relationships that can be identified in the valid values of the data set to assist in estimating the missing values. Mean, Median and Mode Imputations are 3 main strategies that are commonly used to fill in the missing cells. It is mainly filling in the missing values with mean/median/mode of all known values of the column.

The Formula for the same is:-

$$M_o = l + \left(\frac{f_1 - f_0}{2f_1 - f_0 - f_2}\right)h$$

We will use median imputation strategy in our dataset. It is nothing but filling in the empty cells of the column with median values of the known values of the variable. We use **Imputer()** and **imputer.transform()** functions to implement our imputation. The strategy used is **"Median"** and the strategy used in imputing categorical values after transforming is **"most_frequent"**.

```
# Median imputation of missing values
imputer = Imputer(strategy = 'median')

# Fit on the training data
imputer.fit(train)

# Transform  training  data
train = imputer.transform(train)

#imputing the caegorical variables

imputer=Imputer(strategy = 'most_frequent')
imputer.fit(temp_train)
temp_train=imputer.transform(temp_train)
```

Screenshot 9. (Implementation of Imputation)

5.1.6 Feature Scaling

Real world dataset contains features that highly vary in magnitude, unit and range. Normalization should be performed when the scale of a feature is meaningless and not should normalize when the scale is meaningful.

Feature Scaling is a step of Data Pre Processing which is applied to independent variables or columns of data. It basically helps to limit the data within a particular range. It also helps in speeding up the calculations in an algorithm.

If the data limits are extreme, the values can affect the performance of the model. So, it is very much necessary to set limits and normalize the data. We must use any effective method for feature scaling. We use the formula given in the figure below to perform scaling.

$$X_{new} = \frac{X_i - X_{mean}}{\text{Standard Deviation}}$$

We use the **mean()** and **std()** functions to calculate the mean and standard deviation respectively. The **fit()** function in the sklearn.preprocessing package is used to perform feature scaling.

```
train=pd.DataFrame(train,columns=features)
for col in train.columns:
    mean = train[col].mean()
    std = train[col].std()
    train[col] = (train[col] - mean) / std
temp_train=pd.DataFrame(temp_train,columns=temp_features)
train=pd.concat([train,temp_train],axis=1)
```

Screenshot 10. (Implementation of Feature Scaling)

5.1.7 Data Splitting

The model prediction can only be verified if we check if our results are accurate or not. This can be done by performing validation on the training data itself. So, the train data set must be split so that one part of it can be used to train the model while the other part is used for validation.

We have split our train data into a 7:3 ratio where 70% of the data will be used for training the model and 30% will be used for validating the results. We will achieve this using **train_test_split()** from the sklearn.model_selection library. We give the test size as 0.3.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, Y, test_size=0.3, random_state=101)
from sklearn.linear_model import LogisticRegression
```

Screenshot 11. (Implementation of data splitting)

5.2 Logistic Regression Model

Now, when our data is ready, we must fit it into a model using any algorithm. Since, the existing system uses LR model, we must fit our data into a LR model so that we can later compare the results with our ANN model and confirm which is better.

We need to fit our LR model to a set of data in order to estimate the parameters β_0 and β_1. In a linear regression we mentioned that the straight line fitting the data can be obtained by minimizing the distance between each dot of a plot and the regression line.

5.2.1 Comparison of Logistic Regression vs. Linear Regression

Linear regression results can vary from number to number and they have a high range. So, basically the results prediction is not that efficient.

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n,$$

where T denotes the transpose, so that xiTβ is the inner product between vectors xi and β.Often these n equations are stacked together and written in matrix notation as

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

LR results are only in the range from 0 to 1. So, it is basically a type of probability test, where is predicts the probability of an event.

$$P = \frac{e^{a+bX}}{1+e^{a+bX}}$$

Logistic regression is different from linear regression in a way such that linear regression makes a linear graph but LR makes a curved graph in the plot. Linear regression was used earlier but LR provides way better results.

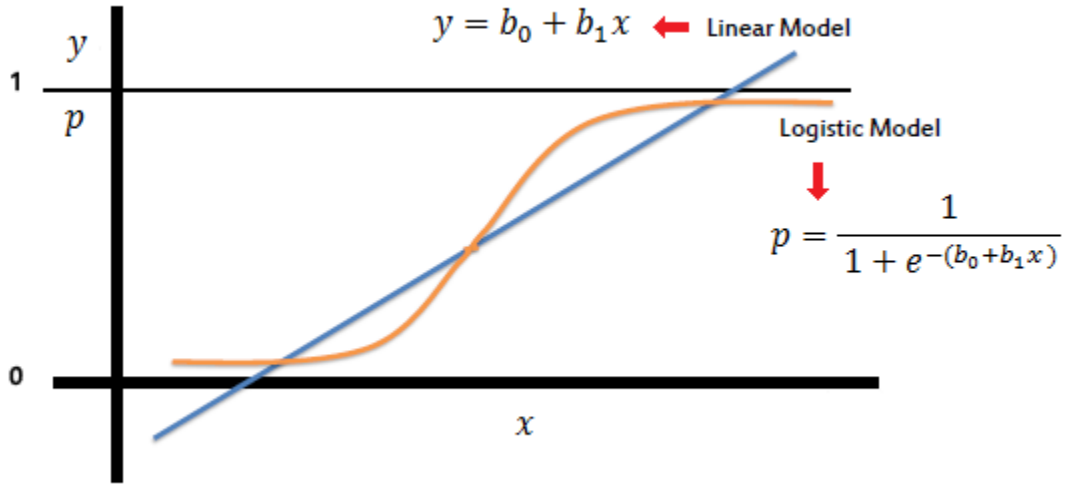


Figure 9. (Logistic regression vs Linear Regression Graph comparison)

5.2.2 Fitting LR model into data set

LR model is fit into the data set using the **LogisticRegresion()** function of the **sklearn.linear_model** package. The result is stored in a variable named "logreg". We also set the weight for our model to be predicted so that the results are scaled. We give the weight as every 0 has a weight of 11 and every 1 has a weight of 120 where 1 represents "person will default". This is done because the default cases are lesser in number than people who do not default. Without proper weight balancing the results will be skewed.

Once the LogisticRegression() function has been implemented. The model is fit using the **fit()** function. It is done using logreg.fit(). After fiting the model, we start predicting the results using **predict()** function. The predict() function will start predicting the probability of 1's and 0's in the data set and then verification will be done on the basis of confusion matrix.

```
from sklearn.linear_model import LogisticRegression
```

```
logreg=LogisticRegression(class_weight={0:11,1:120})
```

```
logreg.fit(X_train,y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py
to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
LogisticRegression(C=1.0, class_weight={0: 11, 1: 120}, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

```
predictions=logreg.predict(X_test)
```

Screenshot 11. (Implementation of LR model fit)

5.2.3 Confusion Matrix

A confusion matrix or error matrix is a table that is often used to describe the performance of a classification model on a set of data for which the values are known. It allows the visualization of the performance of an algorithm.It allows very easy identification of confusion among classes. Many performance measures are identified from the confusion matrix.

The number of correct and incorrect predictions are easily visualized through a confusion matrix. It shows how the model is confused by the predictions and the actual values. It also gives the type of error given by the model.

The various terms of a confusion matrix are:

(1) **True Positive (TP)**: The results under TP show the number of predicted positive results that are actually positive in the validation data.
(2) **True Negative (TN)**: The results under TN show the number of predicted negative results that are actually negative in the validation data.
(3) **False Positive (FP)**: The results under FP show the number of predicted positive results that are actually negative in the validation data.
(4) **False Negative (FN)**: The results under TP show the number of predicted negative results that are actually positive in the validation data.

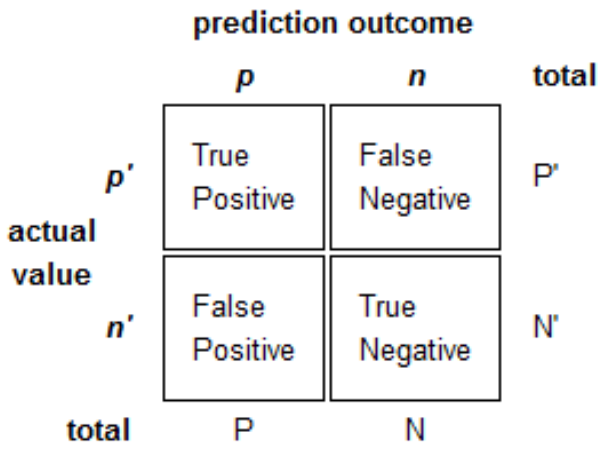


Figure 10. (Confusion matrix representation)

Our real problem is prediction of False Positives. The number of FP results should be minimum because we cannot let a defaulter get away from our prediction.

Confusion matrix is implemented using the **confusion_matrix()** function of the **sklearn.metrics** package. The confusion matrix is made using the validation data which was split from the train data, and the predictions.

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, predictions)
```

Screenshot 12. (Implementation of confusion matrix in LR model)

5.2.3 Accuracy and ROC curve

After all the predictions and the confusion matrix, it is necessary to calculate the accuracy of our predictions. The accuracy is nothing but the percentage of successful predictions of the model.

The accuracy is calculated using the **accruracy_score()** function of the **sklearn.metrics** library. The accuracy score will give an insight as to how much the model is accurate.

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test, predictions)
```

Screenshot 13. (Implementation of accuracy in LR model)

5.2.3.1 ROC curve

Receiver Operating Characteristic(ROC) curve is nothing but a way of presenting the rate of FP against the rate of TP over a graph. Each point in the ROC curve represents sensitivity pair for a particular decision threshold.

The better the curve is the better are the prediction results. The area of the curve is the **ROC Score**. More is the area the better trained model it is. If the curve is a straight line, it means that the model is not trained at all.

We calculate the roc score and make a roc curve after getting the confusion matrix. We use function **roc_auc_score()** function of the sklearn.metrics library. Using this score we plot a graph using **"seaborn"** library. We plot the graph using the **plot()** function of the seaborn library.

We calculate the roc score and plot the curve for both the development and validation process.

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
logit_roc_auc2=roc_auc_score(y_test,predictions)
print(logit_roc_auc2)
```

```
confusion_matric=confusion_matrix(y_test,logreg.predict(X_test))
acc=(confusion_matric[0][0]+confusion_matric[1][1])/(confusion_matric[0][0]+confusion_matric[0][1]+confusion_matric[1][0]+con
print("Percentage accuracy is ", acc)
fpr, tpr, thresholds = roc_curve(y_test,probability[:,1])
plt.figure()
plt.plot(fpr, tpr, label='LR (area = %0.2f)' % logit_roc_auc2)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic(Validation)')
plt.legend(loc="lower right")
plt.savefig('Log_ROC2')
```

Screenshot 14. (Implementation of roc curve in LR model)

5.3 ANN Model

When the data is prepared, any model can be fit into it. Already LR model is fit into the data so that the outputs can be compared. It is now the turn to fit in ANN model into the data.

ANN basically creates a kind of black box, because we can't really know the probability of each prediction. It just predicts classes in the form of 0 and 1, but because of its multi layers, it gives highly efficient results.

To fit the ANN model we must follow certain process. We use "**keras"** library for ANN model.

5.3.1 Model Selection

There are various models in ANN but we will use **Sequential Model**.

Sequential Model in ANN

The sequential model is a linear stack of layers. Sequential model can be created by passing a list of layer instances to the constructor.

Specifying the input shape

The model needs to know what input shape it should expect. For this reason, the first layer in a Sequential model needs to receive information about its input shape. There are several possible ways to do this:

- Pass an input_shape argument to the first layer. This is a shape tuple. In input_shape, the batch dimension is not included.
- Some 2D layers, such as Dense, support the specification of their input shape via the argument input_dim, and some 3D temporal layers support the arguments input_dim and input_length.

- If you ever need to specify a fixed batch size for your inputs, you can pass a batch_size argument to a layer. If you pass both batch_size=32 and input_shape=(6, 8) to a layer, it will then expect every batch of inputs to have the batch shape (32, 6, 8).

Compilation

Before training a model, configuration of learning process is to be done, via the compile method. It receives three arguments:

- An optimizer. This could be the string identifier of an existing optimizer or an instance of the Optimizer class.
- A loss function. This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function or it can be an objective function.
- A list of metrics. A metric could be the string identifier of an existing metric or a custom metric function.

Training

These models are trained on Numpy arrays of input data and labels. For training a model, fit function is typically used.

We use the **Sequencial()** function to implement the select the type of model.

We then add layers to the model, where each layer defines: (1)Number of activation units, (2)type of activation function and (3)no of inputs from previous layer.

5.3.1.1 Activation Units

Activation unit is nothing but the number of outputs from the previous layer. These units become the input for the next layer. These activation units also help in performing forward and backward propagation and yielding a prediction.

5.3.1.2 Activation Function

Activation functions are really important for an Artificial Neural Network to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response variable. They introduce non-linear properties to our Network. Their main purpose is to convert a input signal of a node in a A-NN to an output signal. That output signal now is used as a input in the next layer in the stack.

If we do not apply a activation function then the output signal would simply be a simple linear function. A linear function is just a polynomial of **one degree.** Now, a linear equation is easy to solve but they are limited in their complexity and have less power to learn complex functional mappings from data. A Neural Network without activation function would simply be a **Linear regression Model,** which has limited power and does not performs good most of the times.

5.3.2.1.1 Sigmoid Activation function

It is a activation function of form:

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

Its Range is between 0 and 1. It is a S -shaped curve. It is easy to understand and apply but it has major reasons which have made it fall out of popularity -

- Vanishing gradient problem.
- Its output isn't zero centered. It makes the gradient updates go too far in different directions. 0 < output < 1, and it makes optimization harder.
- Sigmoids saturate and kill gradients.
- Sigmoids have slow convergence.

5.3.1.3 Number of Inputs

The number of variables in the data frame are the number of inputs in the first layer of the ANN model. After the first layer, the output units from the previous layer serve as the input for the next layer.

We implement the sequential model by first adding layers. We add a total of 3 layers and select "sigmoid" as the activation function. The final layer yields only 1 activation unit and that is the out of the sequential model.

```
from keras.utils import to_categorical
from keras import models
from keras import layers
from keras.metrics import Recall
```

```
Using TensorFlow backend.
```

```
model = models.Sequential()
```

```
model.add(layers.Dense(90, activation = "sigmoid", input_shape=(198, )))
```

```
model.add(layers.Dense(45, activation = "sigmoid", input_shape=(90, )))
```

```
model.add(layers.Dense(1, activation = "sigmoid"))
```

Screenshot 15. (Implementation of sequential model)

5.3.2 Model Compilation

Once the layers are added and model is prepared, we must compile the model using **compile()** function. We must decide the optimizer type, loss and metrics. After model compilation it is ready to be fit in. Model compilation requires some parameters, they are:

5.3.2.1 Optimizer

During the training process, we tweak and alter the parameters (weights) of our model to do and minimize that loss function, and make our predictions as correct and optimized as possible. But how exactly does one do that? How does one change the parameters of your model, by what quantity, and when?

This is where optimizers are available. They tie together the loss function and model parameters by updating the model in response to the output of the loss function. In simpler terms, optimizers shape and mold your model into its most accurate possible form by futzing with the weights. The loss function is that the guide to the terrain, telling the optimizer when it's getting the correct or wrong direction.

For a useful mental model, you'll be able to consider a hiker trying to induce down a mountain with a blindfold on. It's impossible to grasp which direction to travel in, but there's one thing she will be able to know: if she's taking place (making progress) or intensifying (losing progress). Eventually, if she keeps taking steps that lead her downwards, she'll reach the bottom.

Similarly, it's impossible to grasp what your model's weights should be right from the beginning. But with some trial and error supported the loss function (whether the hiker is descending), you'll be able to find yourself getting there eventually.

There are many types of optimizer and me must select the one which will yield the best results with our data. So we select **"bgd"** as the optimizer in our data set.

5.3.2.1.1 Batch Gradient Descent (BGD) Optimizer

Batch gradient descent could be a variation of the gradient descent algorithm that calculates the error for every example within the training dataset, but only updates the model in the end training examples are evaluated. One cycle through the whole training dataset is named a training epoch. Therefore, it's often said that batch gradient descent performs model updates at the tip of every training epoch.

Upsides

1. Fewer updates to the model means this variant of gradient descent is more computationally efficient than stochastic gradient descent.

2. The decreased update frequency ends up in a more stable error gradient and should end in a more stable convergence on some problems.

3. The separation of the calculation of prediction errors and therefore the model update lends the algorithm to data processing based implementations.

Downsides

1. The more stable error gradient may end in premature convergence of the model to a less optimal set of parameters.

2. The updates at the tip of the training epoch require the extra complexity of accumulating prediction errors across all training examples.

3. Commonly, batch gradient descent is implemented in such the simplest way that it requires the whole training dataset in memory and available to the algorithm.

4. Model updates, and successively training speed, may become very slow for big datasets.

5.3.2.2 Metrics

Metric is a function that is used to judge the performance of the model. It is a parameter on which the performance of our model depends. There can be many metrics such as accuracy, recall and time, but we use **"recall"** as our metrics.

5.3.2.2.1 Recall

Recall attempts to answer the question as to what proportion of actual positives was identified correctly?

Mathematically, recall is defined as follows:

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

For example, for a text search on a collection of documents, recall is that the variety of correct results divided by the amount of results that ought to are came back.

In binary classification, recall is termed sensitivity. It is viewed because the likelihood that a relevant document is retrieved by the question.

It is trivial to attain recall of 100% by returning all documents in response to any question. Therefore, recall alone isn't enough however one has to live the amount of non-relevant documents conjointly, as an example by conjointly computing the exactitude.

5.3.2.3 Loss Function

Loss function is used to optimize the parameters of our prediction. We must optimize the weights of the parameters in order to avoid loss. The loss is calculated using loss function by matching the target value and predicted value by a neural network. There are various types of loss functions and we must select the best one according to our data. So, according to our data **"mean squared error"** may be the best loss function and so we use the same.

5.3.2.3.1 Mean Squared Error

In statistics, the mean squared error (MSE) of an estimator (of a procedure for estimating an unobserved quantity) measures the common of the squares of the errors — that's, the common squared difference between the estimated values and what's estimated. MSE may be a risk function, admire the arithmetic mean of the squared error loss. the very fact that MSE is sort of

always strictly positive (and not zero) is thanks to randomness or because the estimator doesn't account for information that would produce a more accurate estimate.

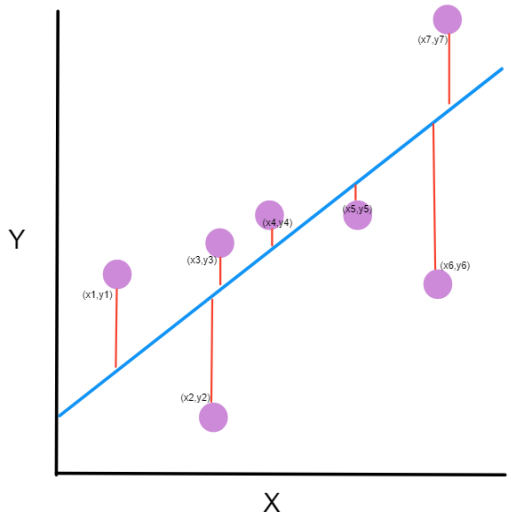


Figure 11 (mean squared error graph)

- The purple dots are the points on the graph. Each point has an x-coordinate and a y-coordinate.

- The blue line is our prediction line. This is a line that passes through all the points and fits them in the best way. This line contains the predicted points.

- The red line between each purple point and the prediction line are the errors. Each error is the distance from the point to its predicted point.

Let's define a mathematical equation that will give us the mean squared error for all our points.

$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

So, we implement the model compilation using the compile() function by giving the optimizer as "bgd", metrics as "recall()" and loss as "mean_squared_error".

```
model.compile(
 optimizer = "sgd",
 loss = "mean_squared_error",
 metrics = [Recall()]
)
```

Screenshot 16. (Implementation of model compilation)

5.3.3 Fitting ANN Model

Before fitting our model, we must set the weights of the classes. We set the weight for our model to be predicted so that the results are scaled. We give the weight as every 0 has a weight of 11 and every 1 has a weight of 120 where 1 represents "person will default". This is done because the default cases are lesser in number than people who do not default. Without proper weight balancing the results will be skewed.

Then we fit in our model using **fit()** function. We must set the no of **epochs** and **batch size.**

Epoch is nothing but a full cycle of forward propagation and backward propagation combined. We set the number of epochs to 30, i.e. 30 times there will be a complete cycle of forward and backward propagation. Our model keeps getting trained as these epochs are running.

Batch Size is nothing but the number of training examples in one epoch. We set the batch size as 32, i.e. 32 examples will be considered through 1 epoch.

The third and final parameter of the fit function is the weight. We have already decided the weights of the classes and will pass them in this function.

```
weight={0:11,1:120}
results = model.fit(
 X_train, y_train,
 epochs= 30,#30
 batch_size = 32,#1024
 class_weight=weight
)
```

Screenshot 17. (Implementation of ANN model fit)

5.3.4 Confusion Matrix

As defined earlier, a confusion matrix is an error matrix that tells us the errors that our model makes during predictions.

Confusion matrix is implemented using the **confusion_matrix()** function of the **sklearn.metrics** package. The confusion matrix is made using the validation data which was split from the train data, and the predictions.

```
prediction=model.predict_classes(X_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(confusion_matrix(y_test,prediction))
```

Screenshot 18. (Implementation of confusion matrix in ANN model)

5.3.5 Accuracy and ROC curve

After all the predictions and the confusion matrix, it is necessary to calculate the accuracy of our predictions. The accuracy is nothing but the percentage of successful predictions of the model.

The accuracy is calculated using the **accruracy_score()** function of the **sklearn.metrics** library. The accuracy score will give an insight as to how much the model is accurate.

5.3.5.1 ROC curve

Receiver Operating Characteristic(ROC) curve is nothing but a way of presenting the rate of FP against the rate of TP over a graph. Each point in the ROC curve represents sensitivity pair for a particular decision threshold.

The better the curve is the better are the prediction results. The area of the curve is the **ROC Score**. More is the area the better trained model it is. If the curve is a straight line, it means that the model is not trained at all.

We calculate the roc score and make a roc curve after getting the confusion matrix. We use function **roc_auc_score()** function of the sklearn.metrics library. Using this score we plot a graph using **"seaborn"** library. We plot the graph using the **plot()** function of the seaborn library.

We calculate the roc score and plot the curve for both the development and validation process.

```
confusion_matric=confusion_matrix(y_test,prediction)
acc=(confusion_matric[0][0]+confusion_matric[1][1])/(confusion_matric[0][0]+confusion_matric[0][1]+confusion_matric[1][0]+confus
print("Percentage accuracy is ", acc)
fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test))
plt.figure()
plt.plot(fpr, tpr, label='ANN (area = %0.2f)' % logit_roc_auc2)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic(Validation)')
plt.legend(loc="lower right")
plt.savefig('Log_ROC2')
```

Screenshot 19. (Implementation of roc curve and accuracy in ANN model)

CHAPTER 6

Results and Outputs

6.1 Result comparison of LR vs ANN

Table 1 (Table : Comparison of results in LR model vs ANN model)

Feature	LR Model	ANN Model
Accuracy	0.6982	0.7182
TP	59612	57565
TN	4802	5006
FP	2561	2357
FN	25729	27326
ROC score	0.6771	0.7674

6.2 Outputs

	precision	recall	f1-score	support
0	0.96	0.70	0.81	84891
1	0.16	0.65	0.26	7363
micro avg	0.70	0.70	0.70	92254
macro avg	0.56	0.68	0.53	92254
weighted avg	0.90	0.70	0.77	92254

Screenshot 20. (Classification report of LR Model)

	precision	recall	f1-score	support
0	0.96	0.72	0.82	84891
1	0.16	0.63	0.26	7363
micro avg	0.71	0.71	0.71	92254
macro avg	0.56	0.68	0.54	92254
weighted avg	0.89	0.71	0.78	92254

Screenshot 21. (Classification report of LR Model)

```
array([[59612, 25279],
       [ 2561,  4802]], dtype=int64)
```

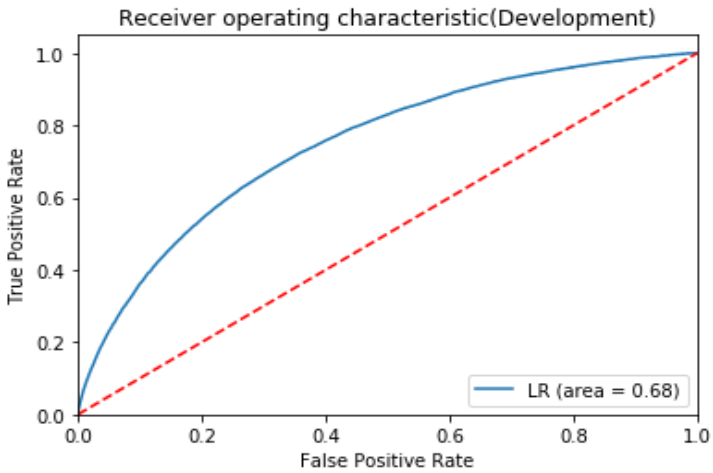
Screenshot 22. (Confusion Matrix of LR model)

```
[[57565 27326]
 [ 2357  5006]]
```

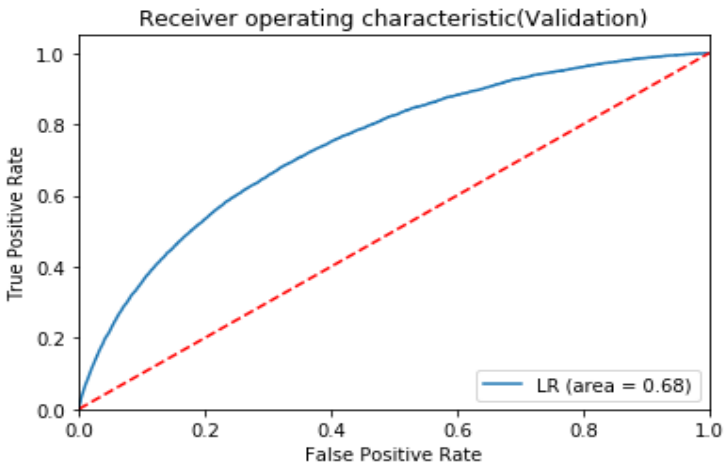
Screenshot 23. (Confusion Matrix of ANN model)

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 90)                17910
_________________________________________________________________
dense_2 (Dense)              (None, 45)                4095
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 46
=================================================================
Total params: 22,051
Trainable params: 22,051
Non-trainable params: 0
_________________________________________________________________
```

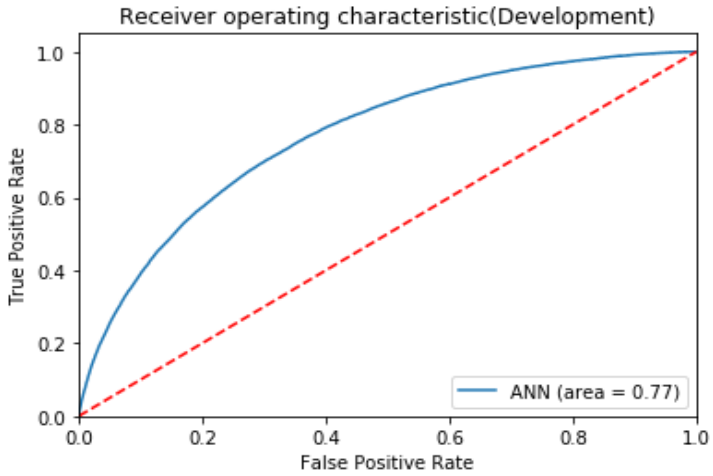
Screenshot 24. (ANN Sequential Model summary)



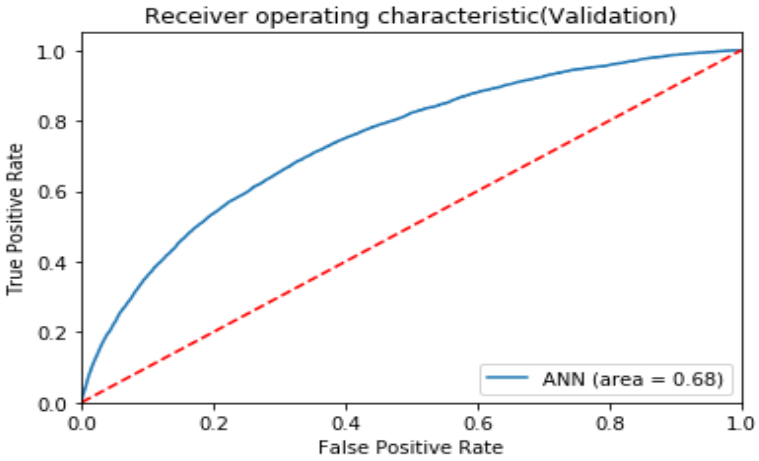
Screenshot 25. (ROC Development curve for Logistic Regression Model)



Screenshot 26. (ROC Validation curve for Logistic Regression Model)



Screenshot 28. (ROC Development curve for ANN Model)



Screenshot 29. (ROC Development curve for ANN Model)

CHAPTER 7

Conclusion

We have successfully predicted the “Default Cases”. We have created this model with 3 hidden layers and 30 epochs. We have achieved an accuracy of about 72% and a roc score of about 77% which is fairly adequate using ANN which is better than the accuracy of about 70% and roc score of about 69% from the Logistic Regression Algorithm.

Our only drawback is that our results cannot be explained because ANN creates a sort of black box, so we cannot know what variables affect our model the most and what is the probability of each prediction.

Thus, it is concluded that we get better results using ANN than Logistic Regression.

CHAPTER 8

Future Scope

Our system provides a way of analyzing borrowers and detects the defaulters. It can help the banks in giving loans to the trustworthy customers. This technique can be used by bank to sanction loans and take a correct decision. This reduces the effort.

In the future, the accuracy of the model can be increased by preparing the datasets according to the data sets of other regions or countries. Also the time complexity of the whole prediction can be also decreased.

The total number of False Positive predictions must also be reduced, by using better data sets and increasing the number of epochs or batch size.

A better optimizer which best suites the data can also be custom designed, which will increase the accuracy and roc score.

CHAPTER 9

Appendix 1

Code for data processing

```
train=pd.read_csv('application_train.csv')
# Create a label encoder object
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le_count = 0
# Iterate through the columns
for col in train:
    if train[col].dtype == 'object':
        # If 2 or fewer unique categories
        if len(list(train[col].unique())) <= 2:
            # Train on the training data
            le.fit(train[col])
            # Transform both training and testing data
            train[col] = le.transform(train[col])
            # Keep track of how many columns were label encoded
            le_count += 1
print('%d columns were label encoded.' % le_count)
train = pd.get_dummies(train)
correlations=train.corr()
def get_hi_corr(df,target):
    y=0.7
    lst=[]
    for i in range(2,121):
        for j in range (1,i):
            x=df.iloc[i][j]
            if(x>=y or x<=(-y)):
```

```
            m=np.abs(target[df.index[i]])
            n=np.abs(target[df.index[j]])
            print(m,n)
            if(m<=n):
               if df.index[i] not in lst and df.index[j] not in lst:
                  lst.append(df.index[i])
            else:
               if df.index[j] not in lst and df.index[i] not in lst:
                  lst.append(df.index[j])
            print(df.index[i],df.index[j],df.iloc[i][j])
   print(lst)
   return lst
#Dropping one of the variables with high correlation
label_to_drop=get_hi_corr(correlations,correlations['TARGET'])
train.drop(labels=label_to_drop[:len(label_to_drop)-2],axis=1,inplace=True)
train.count()[train.count()<307511]
train.drop(labels=['COMMONAREA_MEDI','NONLIVINGAPARTMENTS_AVG'],axis=1,inpl
ace=True)
# Median imputation of missing values
imputer = Imputer(strategy = 'median')
# Fit on the training data
imputer.fit(train)
# Transform  training  data
train = imputer.transform(train)
#imputing the caegorical variables
imputer=Imputer(strategy = 'most_frequent')
imputer.fit(temp_train)
temp_train=imputer.transform(temp_train)
train=pd.DataFrame(train,columns=features)
for col in train.columns:
   mean = train[col].mean()
   std = train[col].std()
   train[col] = (train[col] - mean) / std
temp_train=pd.DataFrame(temp_train,columns=temp_features)
train=pd.concat([train,temp_train],axis=1)
```

Appendix 2

Code for LR model fit

```
from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression(class_weight={0:11,1:120})
logreg.fit(X_train,y_train)
predictions=logreg.predict(X_test)
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, predictions)
from sklearn.metrics import accuracy_score
accuracy_score(y_test, predictions)
probability=logreg.predict_proba(X_test)
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
logit_roc_auc2=roc_auc_score(y_test,predictions)
print(logit_roc_auc2)
confusion_matric=confusion_matrix(y_test,logreg.predict(X_test))
acc=(confusion_matric[0][0]+confusion_matric[1][1])/(confusion_matric[0][0]+confusion_matri
c[0][1]+confusion_matric[1][0]+confusion_matric[1][1])
print("Percentage accuracy is ", acc)
fpr, tpr, thresholds = roc_curve(y_test,probability[:,1])
plt.figure()
plt.plot(fpr, tpr, label='LR (area = %0.2f)' % logit_roc_auc2)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic(Validation)')
plt.legend(loc="lower right")
plt.savefig('Log_ROC2')
```

Appendix 3

Code for ANN model fit

```
from keras.utils import to_categorical
from keras import models
from keras import layers
from keras.metrics import Recall
model = models.Sequential()
model.add(layers.Dense(90, activation = "sigmoid", input_shape=(198, )))
model.add(layers.Dense(45, activation = "sigmoid", input_shape=(90, )))
model.add(layers.Dense(1, activation = "sigmoid"))
model.compile( optimizer = "sgd", loss = "mean_squared_error", metrics = [Recall()] )
weight={0:11,1:120}
results = model.fit( X_train, y_train, epochs= 30, batch_size = 32, class_weight=weight)
prediction=model.predict_classes(X_test)
print(confusion_matrix(y_test,prediction))
logit_roc_auc2=roc_auc_score(y_test,prediction)
print(logit_roc_auc2)
confusion_matric=confusion_matrix(y_test,prediction)
acc=(confusion_matric[0][0]+confusion_matric[1][1])/(confusion_matric[0][0]+confusion_matri
c[0][1]+confusion_matric[1][0]+confusion_matric[1][1])
fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test))
plt.figure()
plt.plot(fpr, tpr, label='ANN (area = %0.2f)' % logit_roc_auc2)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic(Validation)')
plt.legend(loc="lower right")
plt.savefig('Log_ROC2')
```

CHAPTER 10

References

[1] *Uzair Aslam, Hafiz Ilyas Tariq Aziz, Asim Sohail, and Nowshath Kadhar Batcha, An Empirical Study on Loan Default Prediction Models, Journal of Computational and Theoretical Nanoscience, Vol. 16, 3483–3488, (2019)*

[2] *Dominic M. Obarel, Gladys G. Njorogel and Moses M. Murayal, Department of Physical Sciences, Chuka University, (2019)*

[3] *Hafiz Ilyas Tariq, Asim Sohail, Uzair Aslam, and Nowshath Kadhar Batcha, Loan Default Prediction Model Using Sample, Explore, Modify, Model, and Assess, Journal of Computational and Theoretical Nanoscience, Vol. 16, 3489–3503, (2019)*

[4] *Manjeet Kumar, Vishesh Goel, Tarun Jain, Sahil Singhal and Dr. Lalit Mohan Goel, Neural Network approach to Loan Default Prediction, Bharati Vidyapeeth's College of Engineering, New Delhi, (2018)*

[5] *Charles Kwofie, Caleb Owusu-Ansah and Caleb Boadi, Predicting the Probability of Loan-Default, School of Business, University of Ghana, (2018)*

[6] *Pidikiti Supriya , Myneedi Pavani , Nagarapu Saisushma, Namburi Vimala Kumari, K Vikas, Loan Default Prediction using Machine Learning Models, (2017)*

[7] *Semiu A and Akanmu Abdul Rehman Gilal, Decision Tree Model for Predicting Loan Default, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-9 Issue-1, (2019)*

[8] *Jayadev M and Ravi Vadlamani, Predicting Educational Loan Defaults, Indian Institute of Management Bangalore, (2018)*

[9] *Han, Choi and Kim, Loan Default Prediction using Logistic Regression, Department of Accountancy, Ho Polytechnic, Ghana, (2018)*

[10] *Manjeet Kumar, Vishesh Goel, Tarun Jain, Sahil Singhal, Dr. Lalit Mohan Goel, Model using ANN, , Bharati Vidyapeeth's College of Engineering, New Delhi, (2018)*

[11] *Edinam Agbemava, Israel Kofi Nyarko, Thomas Clarkson Adade and Albert K. Bediako, Department of Marketing, Ho Polytechnic, Ghana, (2017)*