2019

CREDIT RISK ANALYSIS USING MACHINE LEARNING AND NEURAL NETWORKS

Dhruv Dhanesh Thanawala
Michigan Technological University, dthanawa@mtu.edu

CREDIT RISK ANALYSIS USING MACHINE LEARNING AND NEURAL NETWORKS

By

Dhruv D. Thanawala

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Mathematical Sciences

MICHIGAN TECHNOLOGICAL UNIVERSITY

2019

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Mathematical Sciences.

Department of Mathematical Sciences

Report Co-advisor:	*Dr. Benjamin Ong*
Report Co-advisor:	*Dr. Gowtham*
Committee Member:	*Dr. Allan Struthers*
Department Chair:	*Dr. Mark Gockenbach*

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank Dr. Benjamin Ong for giving me this opportunity to work on a report. I am very grateful to Dr. Gowtham, who taught me project management and programming etiquettes. I would like to thank my committee member Dr. Allan Struthers for taking the time for reviewing my report. I am thankful to Dr. Anthony Pinar for teaching the Machine Learning Concepts. I want to thank Dr. Mark Gockenbach for supporting me.

I would like to thank my father (Mr. Dhanesh Thanawala), my mother (Mrs. Priti Thanawala), my brother (Mr. Arav Thanawala) and all family members for their blessings and supporting me morally. I also would like to thank Aesha for supporting me and being with me at each moment. I would like to thank Kushal, Akshay, Aditya, Shardool, Upendra, Tejas, and Amit for helping me, motivate me and always entertaining me.

I am thankful to Mr. Jason, who motivates me every time. I want to thank my fellow graduate students Jacob, Jake, Joseph, Prangya, Zazil, and Yasasya for helping me to learn Mathematics in depth. I am very grateful to the Mathematical Science department, all faculty members and staff at Michigan Technological University for encouragement and help.

Abstract

A key activity within the banking industry is to extend credit to customers, hence, credit risk analysis is critical for financial risk management. There are various methods used to perform credit risk analysis. In this project, we analyze German and Australian financial data from UC Irvine Machine Learning repository, reproducing results previously published in literature. Further, using the same dataset and various machine learning algorithms, we attempt to create better models by tuning available parameters, however, our results are at best comparable to published results.

In this report, we have explained the algorithms and mathematical framework that goes behind developing the machine learning models. We conclude with a discussion and comparision of summarizing the best approach to classify these datasets. K - Nearest Neighbors (KNN), Logistic Regression (LR), Naive Byaes Classification, Support Vector Machine (SVM), Classification Trees and Artificial Neural Networks (ANN) are the machine learning models used for this report.

Chapter 1

Introduction

1.1 Motivation

A significant activity of the banking industry is to extend credit to customers. Credit risk management evaluates available data and decides the credibility of a customer, with the intent of protecting the financial institution against fraud.

1.2 Datasets

The UC Irvine Machine Learning repository (UCI - ML) contains collection of datasets useful for evaluating machine learning algorithms. Two datasets rom the UCI - ML

repository were used for this project: the Australian credit dataset [1], and other is the German credit dataset[2]. The German dataset has 20 attributes and 1,000 instances, while the Australian dataset has 14 characteristics and 690 instances. The response variable is a binary decision, whether a customer is credible or not. Both datasets have some common attributes such as the credit score, the purpose of the loan and customer information (occupation, salary, age and account duration).

1.3 Methodology

We apply six machine learning classification algorithms. A summary of the methods are as follows:

1. K **- Nearest Neighbors (KNN)**

   The KNN algorithm is a non parametric technique that classifies unknown data points based on a majority votes of it's K nearest neighbors.

2. **Logistic Regression (LR)**

   Logistic Regression fits a logistic (sigmoid) function to the data. This is useful when the dependent variable is binary. To compute the weights, we will utilize the Iteratively Re-weighted Least Squares algorithm (IRLS).

3. **Naive Bayes Classification**

   This algorithm is based on the Bayes' Theorem. It computes the probability of each potential classification and selects the one with higher probability.

4. **Support Vector Machine (SVM)**

   SVM identifies hyperplanes to separate the data based on labels. To handle nonlinear data, projections using various kernels can be used.

5. **Classification Trees**

   We will implement two types of classification trees: Decision Tree and Random Forest. Decision Tree is similar to a flowchart and is very easy to visualize. Every terminal node represents the output. Random Forest is a collection of decision trees, where the majority vote is taken for prediction.

6. **Artificial Neural Networks (ANN)**

   Neural Network is an algorithm which learns from the data. The process of learning is similar to the human brain. Neural networks are used in many real-world application like classify images, predicting the weather, voice detection, etc.

We proceed to discuss the mathematical framework behind the above methods.

Chapter 2

K - Nearest Neighbors (KNN)

2.1 Mathematical Framework

In KNN, the data is classified by the majority number of K nearest neighbors. The most common metric for nearest distance is Minkowski distance.

$$\mathrm{d(x,y)} = \left(\sum_{i=1}^{k} (|x_i - y_i|)^q \right)^{\frac{1}{q}},$$

where often, one uses $q = 2$ to measure the Euclidean distance, or $q = 1$ to measure the Manhattan distance.

Consider the following example in which we have two classes: one is black circles

and the other is blue rectangles. Suppose we need to classify the data denoted by green triangle shown in fig. 2.1. Let's consider $K = 3$: we identify the three closest neighbors, in this case two blue rectangles and one black circle. Per the majority vote, we classified green triangle as a blue rectangle shown in fig. 2.2.

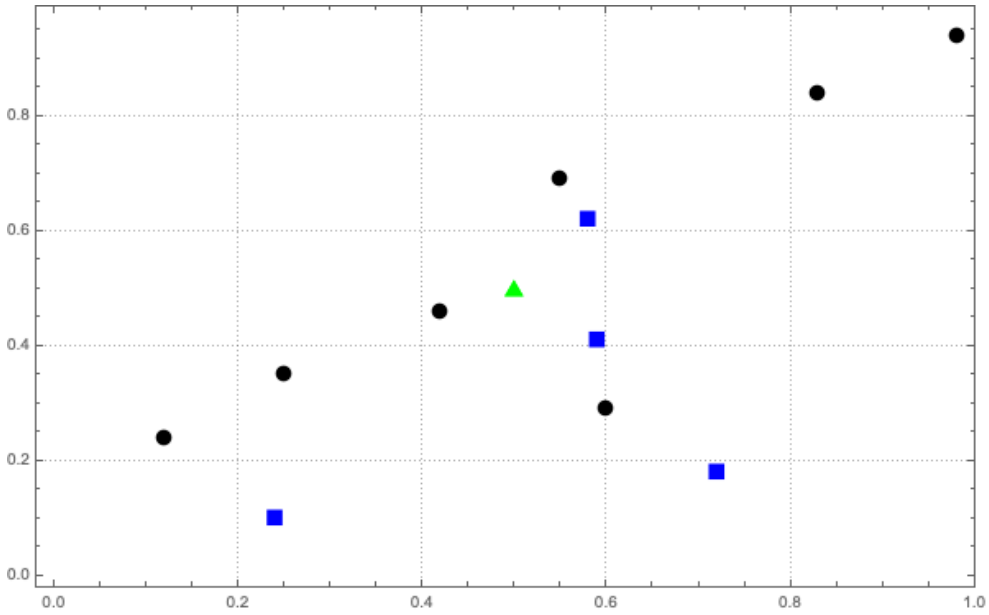

Figure 2.1: Random data generation of two classes: blue rectangles and black circles. The green triangle denotes the data we wish to classify.

Now in fig. 2.3 we can see that, if we choose $K = 5$, then we will have three black circles and two blue rectangles. So, we classified the triangle as a black circle. Therefore, choosing the correct K value is an essential task. The optimal k will always depend on the dataset. For binary classification, the value of K should be considered as an odd number so that there is no ambiguity.

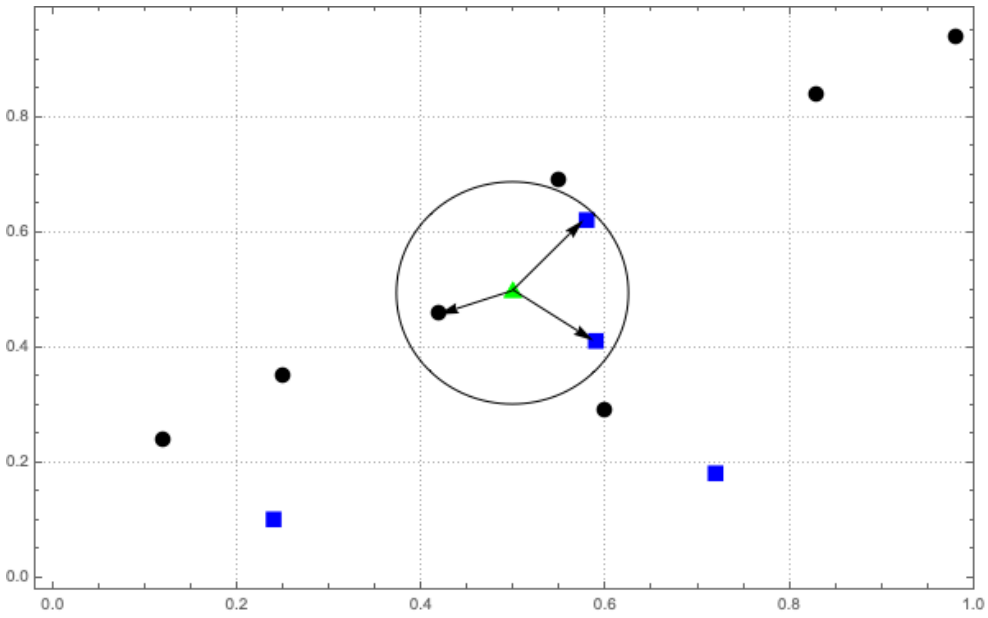

Figure 2.2: K - Nearest Neighbors example using $K = 3$. Here, the green triangle is classified, by majority vote, as a blue rectangle.

2.2 Results

After scaling and normalizing the data, the data is split into training and testing data using an 80% - 20% ratio: 80% of data are used for training, the rest of the data are used for testing; k-fold cross-validation is used to check model accuracy. Initially, I consider $K = 5$ and the Euclidean distance for both data sets; this model is reported as the pre-tuned model below. A grid search was then used to tune the parameters ($K = \{1, 3, 5, 7, 9, 11, 13, 15\}$ and $q = \{1, 2, 3, 4, 5, 6, 7\}$). After tuning, the best parameters for the German data are $K = 7$ and Euclidean distance ($q = 2$). For the Australian data, $K = 13$ and the Manhattan distance, $q = 1$, was optimal. The accuracy of the KNN model is reported in table 2.1.

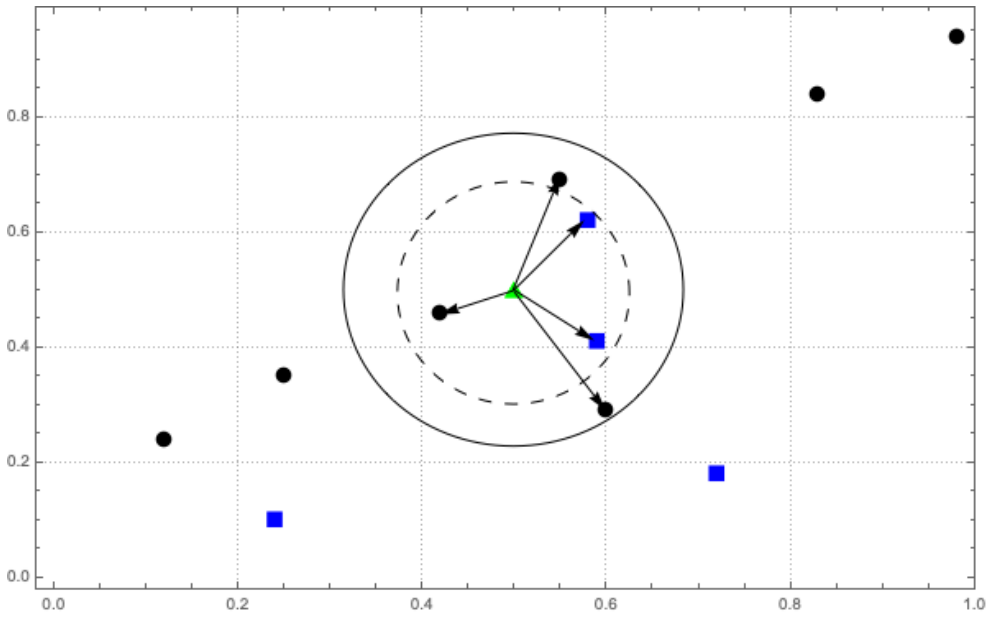

Figure 2.3: K - Nearest Neighbors example using $K = 5$. here, the green triangle is classified, by majority vote, as a black circle

Table 2.1
K - Nearest Neighbors Accuracy Table

Dataset	Mean Accuracy (%)	
	Before Tuning	After Tuning
Australian data	86	87
German data	74	75

Figures 2.4 and 2.5 are the confusion matrix of German data and Australian data respectively. We can see that the tuned KNN model predicts twenty-eight false positives results for German dataset; these customers are not credible, but the model predicts credible. The tuned KNN model predicts eight false positives for the Australian dataset.

Figures 2.6 and 2.7 show the decision boundary plot of the KNN algorithm applied to both data sets. The largest two principal components, as found by Principal

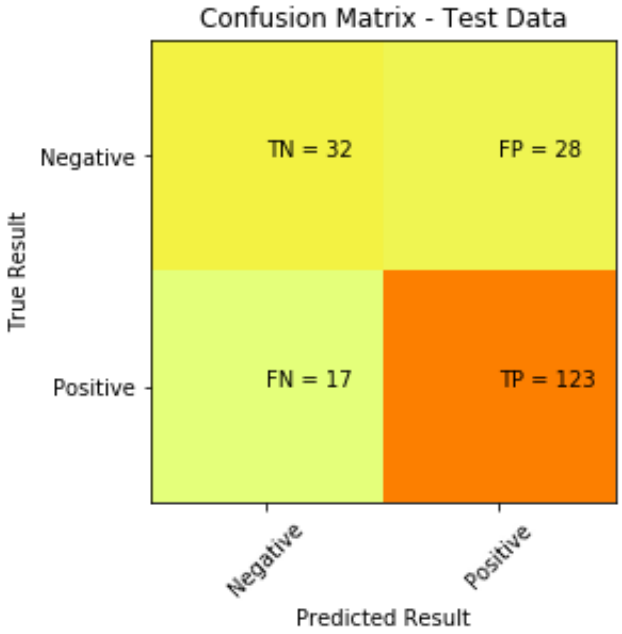

Figure 2.4: Confusion matrix of K - Nearest Neighbors using German dataset. There is an unacceptably large number of false positives.

Component Analysis (PCA), are used to generate the plot [3].

Figures 2.8 and 2.9 are the learning curves of this machine-learning method applied to the German and Austrialina data set respectively. A learning curve is a plot of model learning performance over experience or time – it helps identify whether the model is over-fitting or under-fitting. In this curve, y-axis is the accuracy and x-axis is the number of training samples. For this method, both the curves show that the model is neither over-fitting nor under-fitting because the learning curves do not exhibit high bias or high variance.

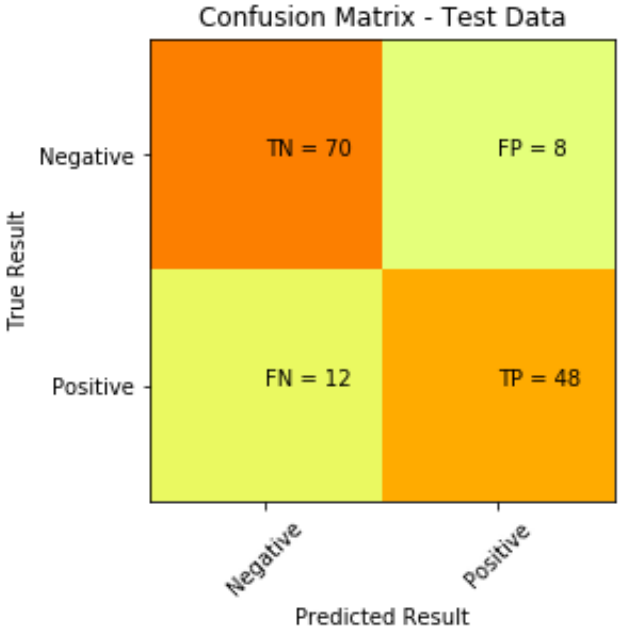

Figure 2.5: Confusion matrix of K - Nearest Neighbors using Australian dataset. There is a small number of false positives.

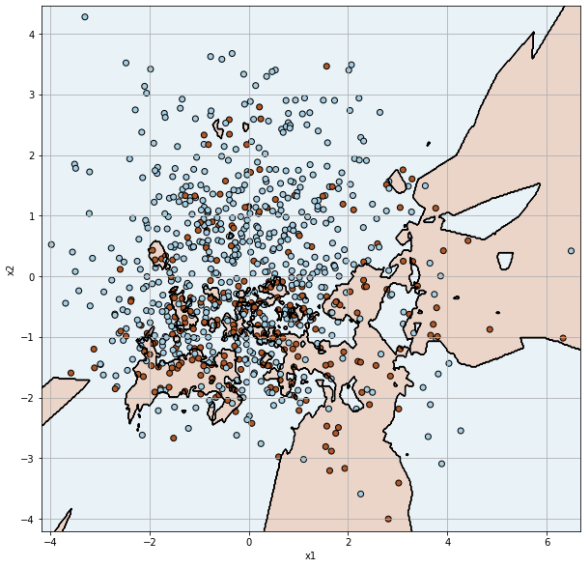

Figure 2.6: Decision boundary plot using K - Nearest Neighbors of German dataset. Observe that the decision boundaries are very complicated for this model.

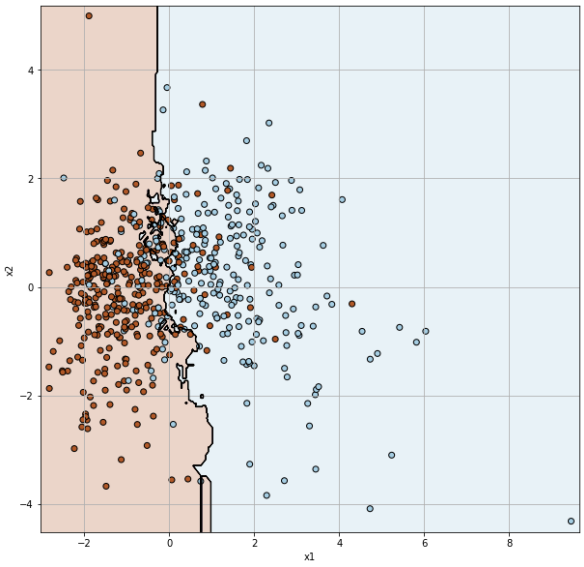

Figure 2.7: Decision boundary plot using K - Nearest Neighbors of Australian dataset. The decision boundary for this data set is a lot simpler, as compared with the decision boundary for the Australian data set.

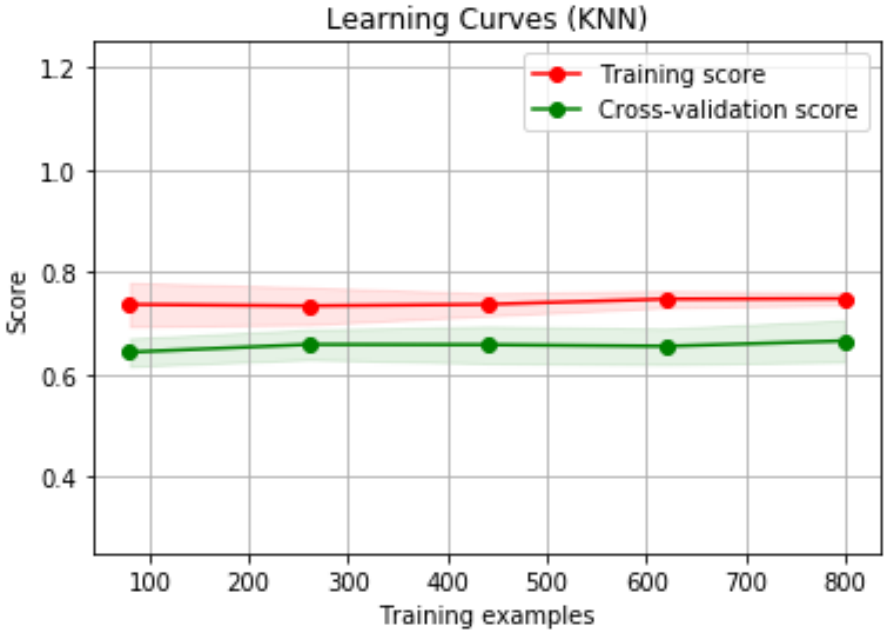

Figure 2.8: Learning curve of K - Nearest Neighbors using the German dataset. No high bias or high variance is observed, indicating that the KNN model is not over-fitting or under-fitting this German data set.

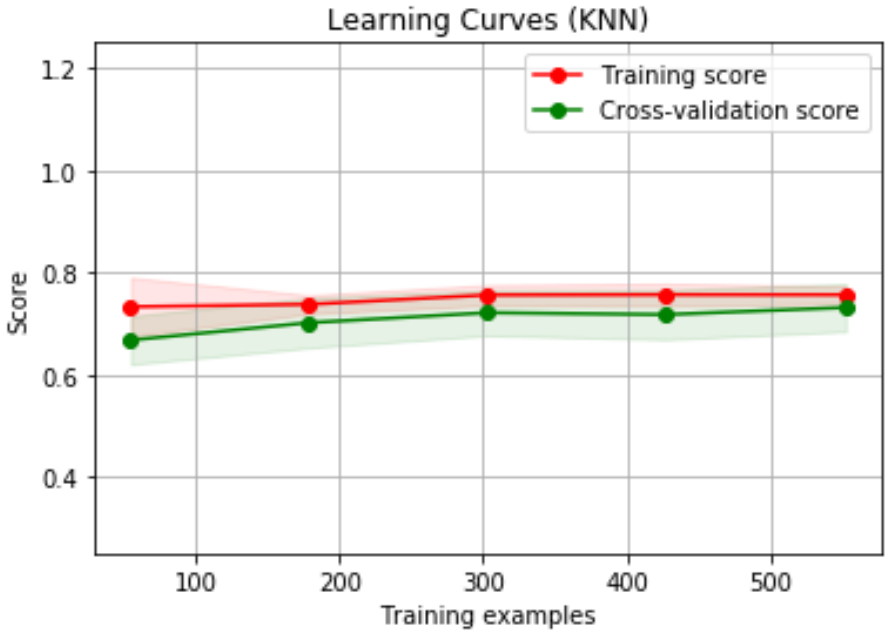

Figure 2.9: Learning curve of K - Nearest Neighbors using Australian dataset. No high bias or high variance is observed, indicating that the KNN model is not over-fitting or under-fitting this Austraian data set.

Chapter 3

Logistic Regression

3.1 Mathematical Framework

Suppose we have N training samples (x_i, y_i) where the response variable, y_i, is a binary variable, i.e., $y_i \in \{0, 1\}$. The log-odds then becomes,

$$\ln\left(\frac{p\left(y_i \mid x_i\right)}{1 - p\left(y_i \mid x_i\right)}\right) = w^T x_i, \tag{3.1}$$

where $p\left(y_i \mid x_i\right)$ is the posterior probability and w is the weight vector that we seek. Logistic Regression (LR) models are fitted via maximum likelihood. Solving eq. (3.1)

for the class posterior probability,

$$\begin{aligned} Pr\left(y_i = 1\right) &= p\left(y_i \mid x_i\right) = \frac{e^{w^T x_i}}{1 + e^{w^T x_i}}, \\ Pr\left(y_i = 0\right) &= 1 - p\left(y_i \mid x_i\right) = \frac{1}{1 + e^{w^T x_i}}. \end{aligned} \tag{3.2}$$

We recall Bernoulli RVs (random variables) are discrete RVs with only two values, typically 0 and 1. The probability mass function is completely specified with one parameter: the "probability of success", $p = Pr(X = 1)$,

$$pr(k) = p^k (1-p)^{(1-k)}, \quad k \in \{0, 1\}.$$

We note that for binary classification, each y_i can be modelled as a Bernoulli RV. Since y_i is Bernoulli RV, the (joint) likelihood is

$$\mathcal{L}(w) = Pr\left(\{y_1, y_2, \ldots\} \mid w, \{x_1, x_2, \ldots\}\right) = \prod_{i=1}^{N} p^{y_i} \left(1-p\right)^{(1-y_i)}.$$

Hence,

$$\begin{aligned} \ln \mathcal{L}(w) &= \sum_{i=1}^{N} \left[y_i \ln(p) + (1 - y_i) \ln(1-p)\right] \\ &= \sum_{i=1}^{N} \left[y_i \ln\left(\frac{e^{w^T x_i}}{1 + e^{w^T x_i}}\right) + (1 - y_i) \ln\left(1 - \frac{e^{w^T x_i}}{1 + e^{w^T x_i}}\right)\right] \\ &= \sum_{i=1}^{N} \left[y_i w^T x_i - \ln\left(1 + e^{w^T x_i}\right)\right]. \end{aligned} \tag{3.3}$$

The task is to maximize the (log) likelihood, eq. (3.3). This is equivalent to minimizing the negative log likelihood (NLL)

$$\mathrm{NLL}(w) = -\sum_{i=1}^{N} \left[y_i w^T x_i - \ln\left(1 + e^{w^T x_i}\right) \right].$$

Gradient descent is a first-order minimization method that uses information about the gradient. The update formula looks like

$$w^{n+1} = w^n - \gamma \; g(w)\big|_{w^n}.$$

where γ is a stepsize to be computed, and the gradient satisfies

$$g(w) = \nabla_w \mathrm{NLL}(w) = X^T (p - y). \tag{3.4}$$

Here, X is $N \times (p+1)$ matrix of x_i values, p the vector of fitted probabilities has elements $p_i = p\,(x_i, w)$, y is a vector of y_i values.

Newton's method is a second-order optimization method which uses the gradient and the Hessian (curvature). The update formula looks like

$$w^{n+1} = w^n - \tilde{\gamma}\; H^{-1}(w)\big|_{w^n} \;\; g\big|_{w^n}, \tag{3.5}$$

where $\tilde{\gamma}$ is a stepsize to be computed, the gradient is defined above in eq. (3.4), and

the Hessian H satisfies,

$$H = \nabla^2 \mathrm{NLL}(w) = X^T S X. \tag{3.6}$$

Here, S is a $N \times N$ diagonal matrix with entries $s_i = p_i(1 - p_i)$. Both methods converge, however, Newtons method is faster since it takes curvature into account. Substituting the definitions of H, eq. (3.6) and g eq. (3.4) into eq. (3.5), we get

$$\begin{aligned} w^{n+1} &= w^n - H^{-1} g \\ &= w^n - (X^T S_n X)^{-1} X^T (y - p_n) \\ &= (X^T S_n X)^{-1}((X^T S_n X) w^n - X^T (y - p_n)) \\ &= (X^T S_n X)^{-1} X^T (S_n X w^n + y - p_n) \\ &= (X^T S_n X)^{-1} X^T S_n z, \end{aligned} \tag{3.7}$$

where we define the working response or adjusted response,

$$z_n = X w^n + S_n^{-1} (y - p_n).$$

3.2 Iteratively Re-weighted Least Squares (IRLS)

We have seen previously, eq. (3.7) that after each iteration, p_n changes which in turn changes S_n and z_n. Therefore we update the weights, s_n, after every iteration, and solve a weighted least squares problem. The algorithm for finding the weights is

summarized in algorithm 1.

Algorithm 1 Iteratively Reweighted Least Squares (IRLS)

$w = 0$
while not converged **do**
  for $i = 1 : N$ **do**
    $p_i = \dfrac{e^{w^T x_i}}{1 + e^{w^T x_i}}$;
    $s_i = p_i(1 - p_i)$;
    $z_i = w^T x_i + \dfrac{y_i - p_i}{s_i}$
  end for
  $S = diag(s_1 s_2 ... s_N)$
  $w = (X^T S X)^{-1} X^T S z$
end while=0

Predictions with the Sigmoid Function: So for a new data point x, compute $p(y_i \mid w, x)$. If $p(y_i \mid w, x) \geq 0.5$ then classify x as Class A. If $p(y_i \mid w, x) < 0.5$ then classify x as Class B.

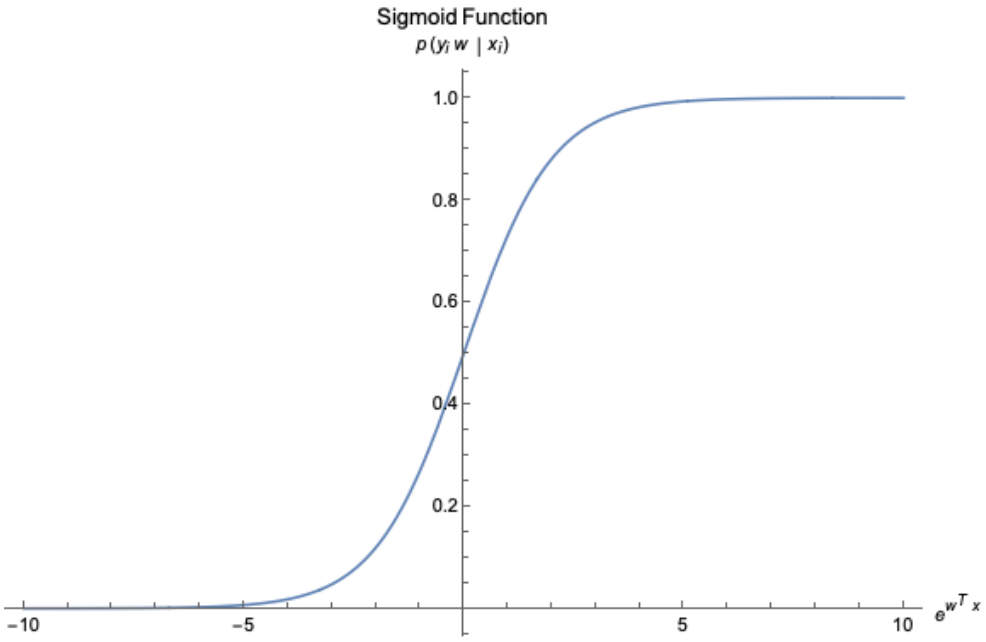

Figure 3.1: Sigmoid Function

3.3 Results

We present the results of the logistic model in table 3.1. In algorithm 1, we iterate until w converges to within 10^{-4}. As previously observed for KNN classification, we are able to generate better models for the Australian data set. The accuracy for the logistic model is marginally better than the KNN model.

Table 3.1
Accuracy of the Logistic Model. The logistic model has marginally better performance compared to the KNN model.

data set	Mean Accuracy Before Tuning (%)
Australian Data	87
German Data	77

Figures 3.2 and 3.3 are the confusion matrix of German data and Australian data respectively. For both datasets, the logistic model has more false positive compared with the KNN model.

Figures 3.4 and 3.5 show the classification boundary plot for the logistic model. Since we are using a linear regression model, the boundary that separates the data is a straight line. For the German data set, this differs significantly from the classification boundary obtained by the KNN model.

Figures 3.6 and 3.7 show the learning curves for this Logistic model. As we increase

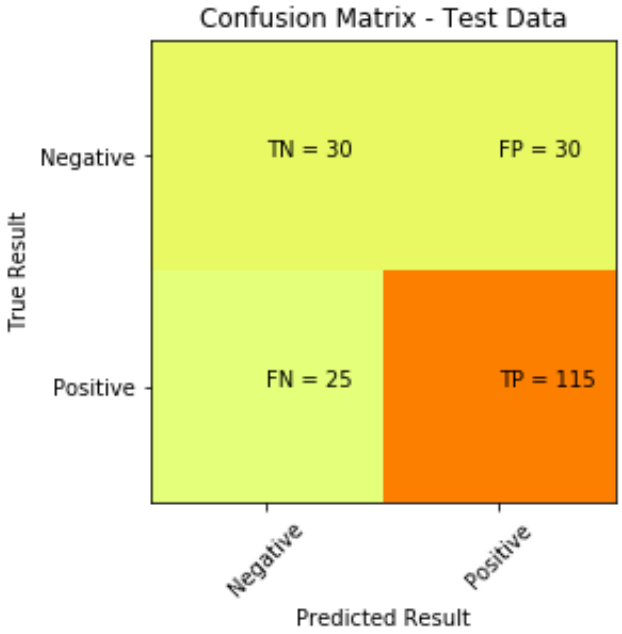

Figure 3.2: Confusion matrix of Logistic Regression using German data set. There is an unacceptably large number of false positives.

the training samples, the model is learning at a slow rate, and in the end, there is a little gap between the training curve and cross-validation curve; so we can say that this model looks a good fit model because it does not show high bias or high variance.

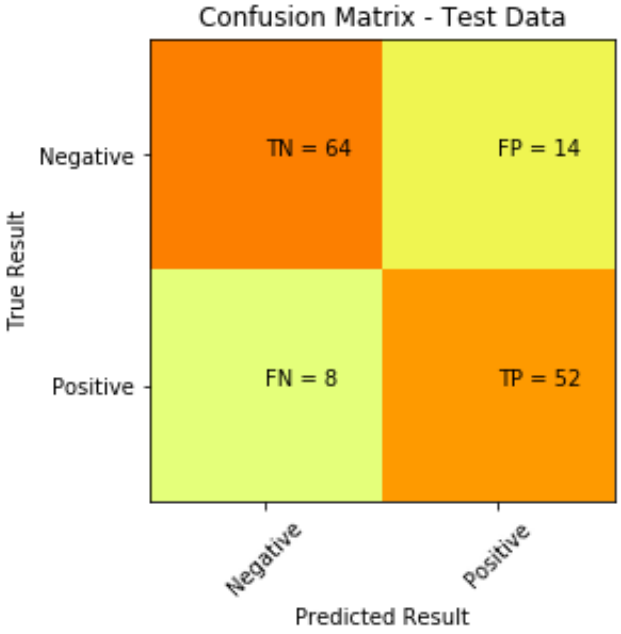

Figure 3.3: Confusion matrix of Logistic Regression using Australian data set. There is an unacceptably large number of false positives.

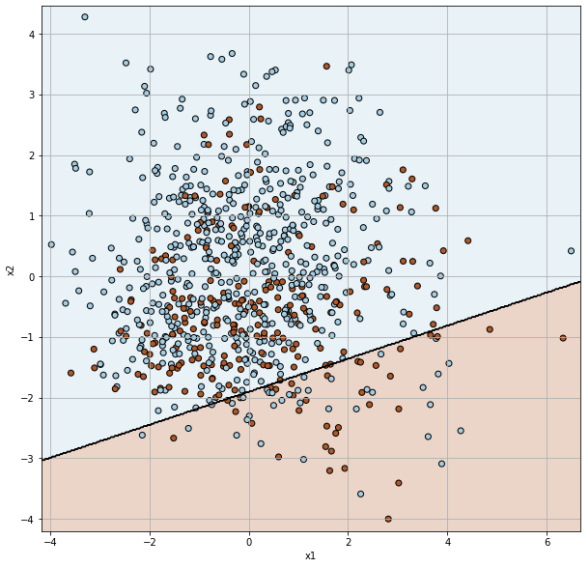

Figure 3.4: Decision boundary plot of the Logistic omodel of the German data set. Since linear regression is used, the decision boundary is a straight line.

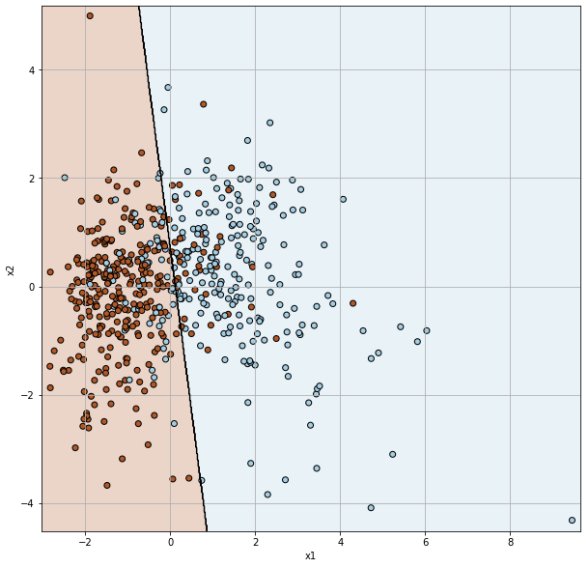

Figure 3.5: Decision boundary plot using Logistic Regression of the Australian data set.

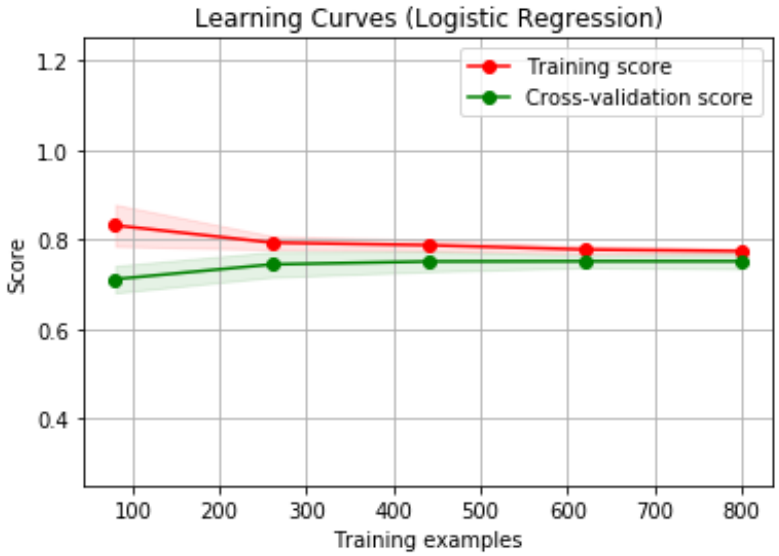

Figure 3.6: Learning curve of Logistic Regression using the German data set. No high bias or high variance is observed, indicating that the LR model is not over-fitting or under-fitting this German data set.

Figure 3.7: Learning curve of Logistic Regression using the Australian data set. No high bias or high variance is observed, indicating that the LR model is not over-fitting or under-fitting this Australian data set.

Chapter 4

Naive Bayes Classification

4.1 Introduction

Naive Bayes Classification algorithm is based on Bayes' theorem. Bayes' theorem reads,

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A)\,P(B \mid A)}{P(B)},$$

where, $P(A)$ is the probability of event A occurring, $P(B)$ is the probability of event B occurring, $P(A \mid B)$ is the probability of A given B, $P(B \mid A)$ is the probability of B given A, and $P(A \cap B))$ is the probability of both A and B occurring. In Naive Bayes, we assume that the features are independent of each other. The classifier proceeds to calculate the probability of each class and selects the class that has the

greatest probability.

4.2 Mathematical Framework

Suppose we view a data set, X, in terms of feature vectors, i.e., our $N \times p$ matrix has p feature vectors,

$$X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \tilde{x}_1 & \tilde{x}_2 & \dots & \tilde{x}_p \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

The conditional probability is given by

$$P(y = k \mid X) = \frac{P(X \mid y = k)\, P(y = k)}{P(X)}, \text{ for } k = 1, \dots, K, \quad j = 1, \dots, p,$$

where $P(y = k \mid X)$ is the posterior probability, $P(y = k)$ is the prior probability, and $P(X \mid y = k)$ is a likelihood. Using the chain rule, the likelihood $P(X \mid y = k)$

can be decomposed as

$$\begin{aligned} P(\tilde{x}_1,\, \tilde{x}_2, \ldots, \tilde{x}_p \mid y = k) =& P(\tilde{x}_1 \mid \tilde{x}_2, \ldots, \tilde{x}_p, y = k) \times \\ & P(\tilde{x}_2 \mid \tilde{x}_3, \ldots, \tilde{x}_p, y = k) \times \cdots \\ & P(\tilde{x}_{p-1} \mid \tilde{x}_p, y = k)\, P(\tilde{x}_p \mid y = k). \end{aligned} \tag{4.1}$$

Using the naive independence assumption,

$$P(\tilde{x}_j \mid \tilde{x}_{j+1}, \ldots, \tilde{x}_p) = P(\tilde{x}_j \mid y = k),$$

eq. (4.1) can be expressed as,

$$P(X \mid y = k) \;=\; \prod_{j=1}^{p} P(\tilde{x}_j \mid y = k),$$

Therefore the posterior probability can be calculated as,

$$P(y = k \mid X) = \frac{P(y = k) \prod_{j=1}^{p} P(\tilde{x}_j \mid y = k)}{P(X)}.$$

The prior probability $P(X)$ is constant therefore,

$$P(y = k \mid X) \propto P(y = k) \prod_{j=1}^{p} P(\tilde{x}_j \mid y = k).$$

Now, if we consider the data set which has $\tilde{x}_1, \ldots, \tilde{x}_p$ and two classes k_1 and k_2 then,

$$\begin{aligned}
P(y = k_1 \mid \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_p) &\propto P(y = k_1) \prod_{j=1}^{p} P(\tilde{x}_j \mid y = k_1) \\
&\propto P(y = k_1) \prod_{j=1}^{p} \mathcal{N}(\tilde{x}_j \mid \mu_{\tilde{x}_j}, \sigma_{\tilde{x}_j}) \\
P(y = k_2 \mid \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_p) &\propto P(y = k_2) \prod_{j=1}^{p} P(\tilde{x}_j \mid y = k_2) \\
&\propto P(y = k_2) \prod_{j=1}^{p} \mathcal{N}(\tilde{x}_j \mid \mu_{\tilde{x}_j}, \sigma_{\tilde{x}_j})
\end{aligned}$$

Where, $\mathcal{N}(\tilde{x}_j \mid \mu_{\tilde{x}_j}, \sigma_{\tilde{x}_j}) = \frac{1}{\sqrt{2\pi\sigma^2_{\tilde{x}_j}}} e^{-(\tilde{x}_j - \mu_{\tilde{x}_j})^2 \,/\, 2\,\sigma^2_{\tilde{x}_j}}$ is Normal (Gaussian) distribution with the mean μ_{x_i} and the standard deviation $\sigma_{\tilde{x}}$. If $P(y = k_1 \mid \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_p) > P(y = k_2 \mid \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_p)$, then the classifier predicts that sample data are in Class k_1.

4.3 Results

So, after applying this method, we get the accuracy shown in table 4.1. Figures 4.1 and 4.2 are the confusion matrix of German data and Australian data respectively. It seems that it has very less false-positives than Logistic Regression method which are 27 and 8 for German and Australian data respectively.

Figures 4.3 and 4.4 are the decision boundary plot which separates the data with a

Table 4.1
Accuracy of the Naive Bayes model. The Naive Bayes model has lesser accuracy compared to Logistic model.

Dataset	Mean Accuracy (%)	
	Before Tuning	After Tuning
Australian data	86	87
German data	74	75

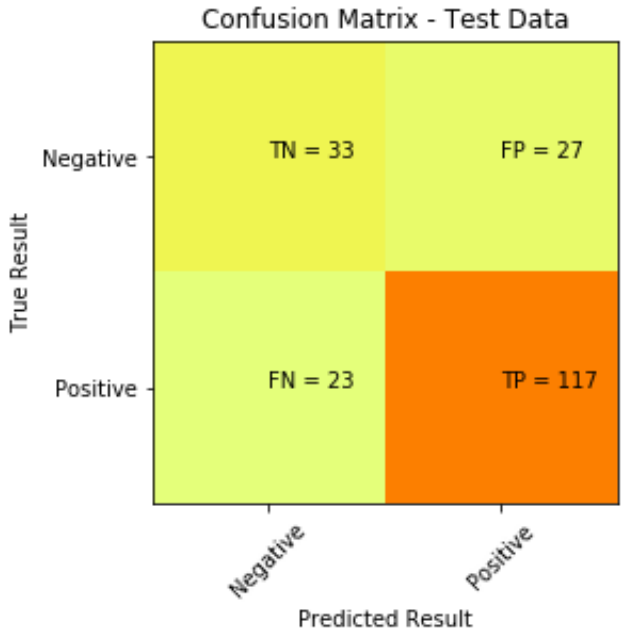

Figure 4.1: Confusion matrix of Naive Bayes using German data set. There is an unacceptably large number of false positives.

curve.

Figures 4.5 and 4.6 are the learning curves for the German and Australian data respectively. In Figure 4.5, there is no gap between the training curve and cross-validation curve, which shows that the model is underfitting, and in fig. 4.6, the gap is very low, which shows that the model is little bit underfitting.

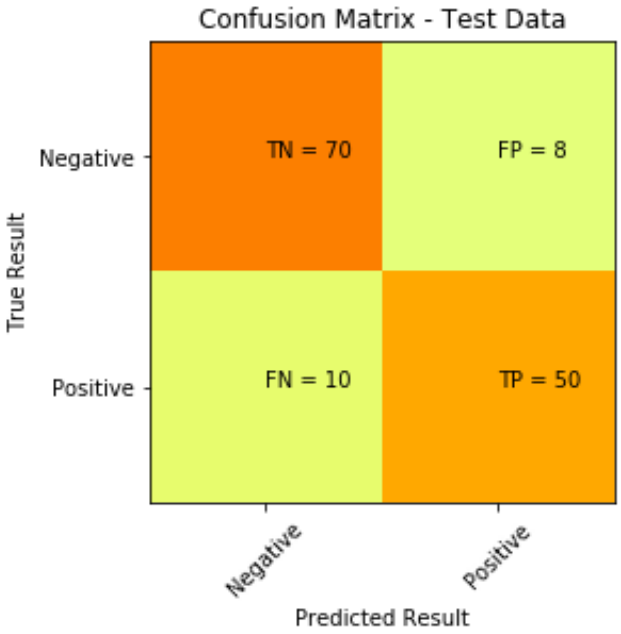

Figure 4.2: Confusion matrix of Naive Bayes using Australian data set. There is a small number of false positives.

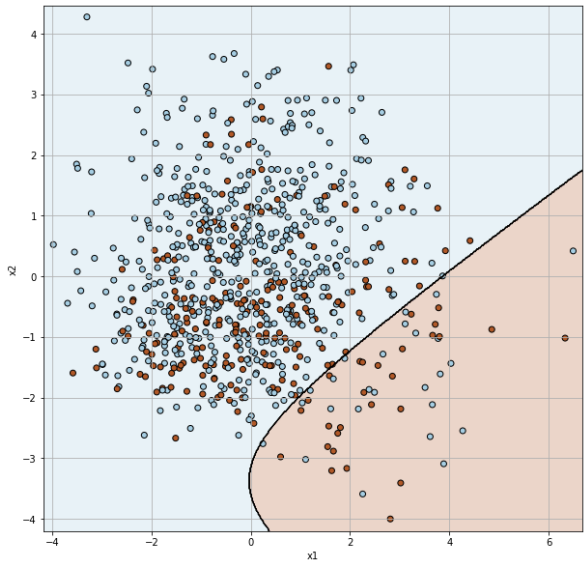

Figure 4.3: Decision boundary plot using Naive Bayes of German data set. Observe that the decision boundary is a curve for this model.

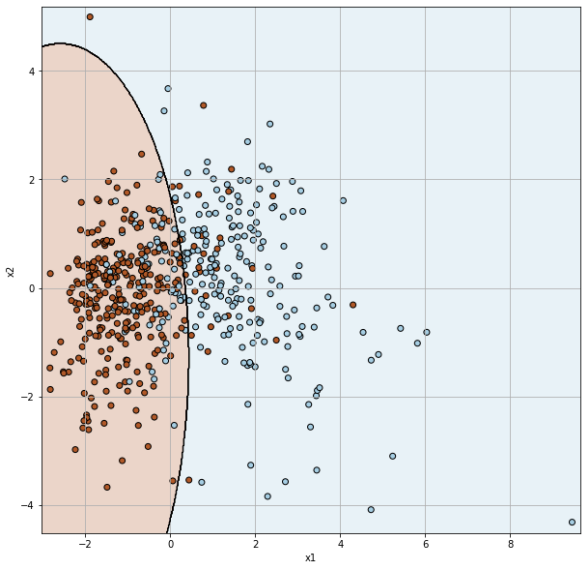

Figure 4.4: Decision boundary plot using Naive Bayes of Australian data set. Observe that the decision boundary is a curve for this model.

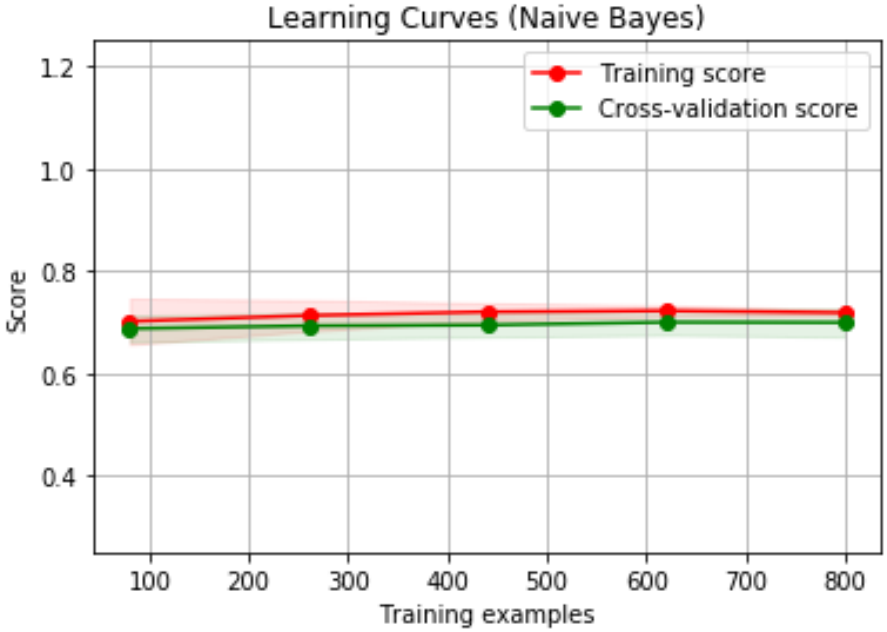

Figure 4.5: Learning curve of Naive Bayes using the German data set. Very small gap is observed as increases the samples, indicating that the Naive Bayes model is under-fitting this German data set.

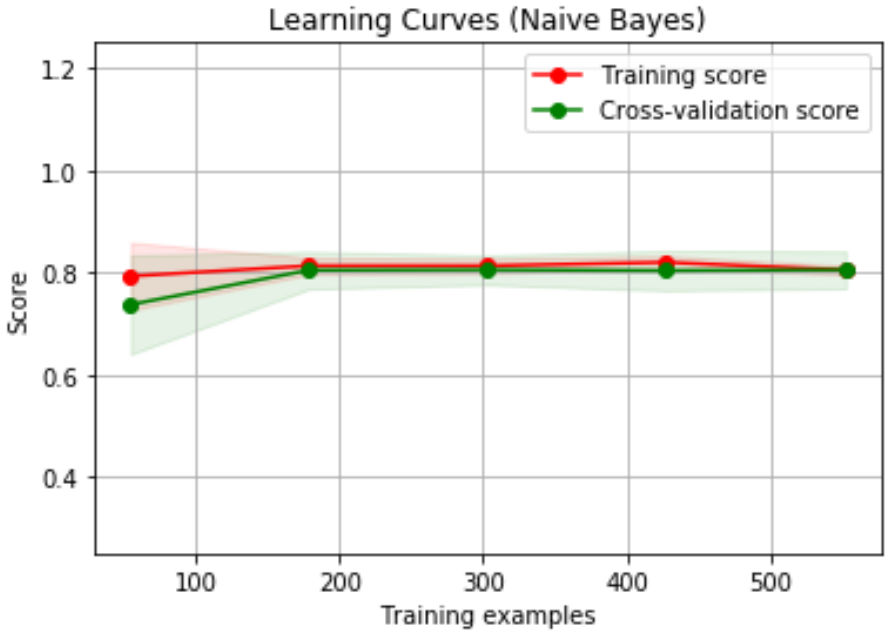

Figure 4.6: Learning curve of Naive Bayes using the Australian data set. No gap is observed as increases the samples, indicating that the Naive Bayes model is under-fitting this Australian data set.

Chapter 5

Support Vector Machine (SVM)

5.1 Linear SVM

Support Vector Machine is a supervised machine learning algorithm. The idea is to select the optimal hyperplane that partitions the data. There are indeed many hyperplanes that can partition the data, see fig. 5.1; the SVM method selects the optimal hyperplane, which we define below. First, the equation of a hyperplane is given by

$$w^T x + b = 0, \tag{5.1}$$

where w is a weight vector, x is the input vector and b is the bias. The optimal hyperplane we wish to find is the one which maximizes the margin to the hyperplane,

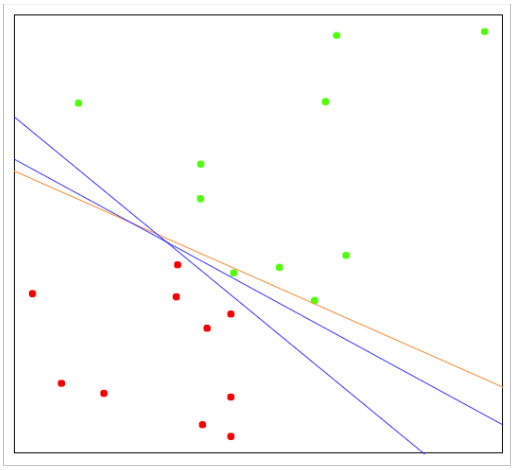

Figure 5.1: Example of the possible hyperplanes

that is, finding a hyperplane that maximizes the distance to the nearest training-data point of any class.

The distance between a data point x_i to a hyperplane defined by eq. (5.1) is given by

$$\frac{|w^T x_i + b|}{\|w\|}. \tag{5.2}$$

We can scale w arbitrarily without changing its direction, so without loss of generality, we can scale w so that the numerator of eq. (5.2) is one. Hence, the distance between the data point x_i to a hyperplane will be given by $\frac{1}{\|w\|}$. The data point(s) in Class "+1" closest to the hyperplane will obey $w^T x_i + b = 1$, and the points in Class "−1" closest to the hyperplane will obey $w^T x_i + b = -1$. An example of the optimal margin

is given in fig. 5.2. The total width between the hyperplanes and the two classes is

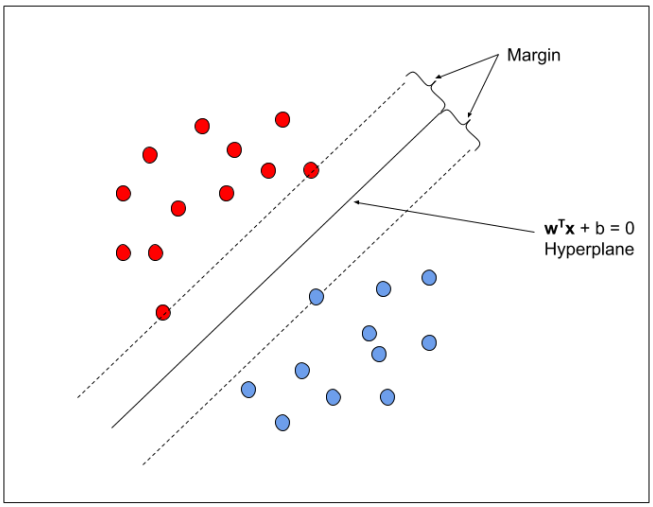

Figure 5.2: Example of the optimal margin

thus given by

$$\frac{1}{\|w\|} + \frac{1}{\|w\|} = \frac{2}{\|w\|}.$$

If the data is linearly separable, the following must hold true:

$$\begin{aligned} w^T x_i + b \geq 1, \qquad & \forall x_i \in C_{+1}, \\ w^T x_i + b \leq -1, \qquad & \forall x_i \in C_{-1}. \end{aligned} \tag{5.3}$$

We note that maximizing the margin is equivalent to minimizing $\|w\|$. For convenience, we minimize $\frac{1}{2}\|w\|^2$. If we desire that all the samples are correctly classified,

i.e., eq. (5.3) is satisfied, then we wish to satisfy

$$y_i(w^T x_i + b) \geq 1, \quad i = 1, \ldots, N.$$

The constrained minimization problem can then be posed:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 \\ &\text{subject to} \quad y_i(w^T x_i + b) \geq 1, \quad i = 1, \ldots, N. \end{aligned}$$

Using Lagrange multipliers to find the minima,

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i (y_i(w^T x_i + b) - 1). \tag{5.4}$$

Due to the inequality constraints, the solution to this must meet the following Karush-Kuhn-Tucker (KKT) conditions,

$$\begin{aligned} &\alpha_i \geq 0, \quad i = 1, \ldots, N; \\ &\alpha_i(y_i(w^T x_i + b) - 1) = 0, \quad \forall i \\ &\frac{\partial \mathcal{L}}{\partial w} = w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0 \implies w = \sum_{i=1}^{N} \alpha_i y_i x_i \\ &\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i = 0 \implies \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{5.5}$$

Substituting these KKT conditions, eq. (5.5), into the Lagrangian equation, eq. (5.4),

leads to the Wolfe dual problem,

$$\begin{aligned}
&\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k \, y_i \, y_k \, x_i^T x_k \\
&\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, N.
\end{aligned}$$

The dual problem is a maximization problem of a cost function quadratic in α. So, we'll minimize the negative of cost function,

$$\begin{aligned}
&\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k \, y_i \, y_k \, x_i^T x_k - \sum_{i=1}^{N} \alpha_i \\
&\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, N.
\end{aligned}$$

To put this in standard QP form, we'll express it in matrix/vector form,

$$\min_{\alpha} \frac{1}{2} \alpha^T \begin{bmatrix} y_1 \, y_1 \, x_1^T \, x_1^T & y_1 \, y_2 \, x_1^T \, x_1^T & \cdots & y_1 \, y_N \, x_1^T \, x_N^T \\ y_2 \, y_1 \, x_2^T \, x_1^T & y_2 \, y_2 \, x_2^T \, x_1^T & \cdots & y_2 \, y_N \, x_2^T \, x_N^T \\ \vdots & \vdots & \ddots & \vdots \\ y_N \, y_1 \, x_N^T \, x_1^T & y_N \, y_2 \, x_N^T \, x_1^T & \cdots & y_N \, y_N \, x_N^T \, x_N^T \end{bmatrix} \alpha - 1^T \, \alpha,$$

$$\text{subject to } \; y^T \, \alpha = 0, \quad 0 \leq \alpha \leq \infty.$$

This minimization problem can be solved by any QP solver. In Python, the `CVXOPT` library has QP solvers, and MATLAB has the `quadprog()` function.

5.2 Non-Linear SVM (Kernel Trick)

Suppose the data is non-linearly separable. SVM can still be applied to separate data with the use of a kernel trick – a kernel is used to transform the data to a higher dimensions which is then separable with hyperplanes.

Radial Basis Functions (RBF) and polynomial kernels are two popular kernel functions used in the SVM community.

† RBF Kernel: $K(x_i,\, x_k) = e^{-\gamma \|x_i - x_k\|^2}$

† d-th degree Polynomial Kernel: $K(x_i,\, x_k) = \left(1 + x_i^T x_k\right)^d$

We can rewrite the Wolfe dual minimization problem using the kernel functions,

$$\begin{aligned} &\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \, \alpha_k \, y_i \, y_k \, K(x_i, x_k) - \sum_{i=1}^{N} \alpha_i \\ &\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, N \end{aligned} \tag{5.6}$$

5.3 Results

By applying Kernel SVM to German and Australian credit data sets, the accuracies are obtained shown in table 5.1. The default parameters for SVM is set to be polynomial kernel with degree 2. The other tuning values for degree are from 1 to 7 and for kernel, RBF is also considered as different kernel. However, for Australian data set, it gives even good accuracy by setting the polynomial kernel with degree 1. While for German data, it improves the accuracy with RBF kernel.

Figures 5.3 and 5.4 are the confusion matrix of German data and Australian data respectively. It seems that it has more false-positives which are 33 and 19 for German and Australian data respectively. This model has good accuracies, but due to more number of false-positives, this model is risky for these data sets.

Table 5.1
Accuracy of the Support Vector Machine model. The SVM model has marginally better performance compared to the Naive Bayes model.

Dataset	Mean Accuracy (%)	
	Before Tuning	After Tuning
Australian data	81	87
German data	72	77

Figures 5.5 and 5.6 are the decision boundary plot of this method. We can see that in German data set the data are separated by the Gaussian curve, while a straight line separates Australian data because for this data set the poly kernel with degree 1 is used.

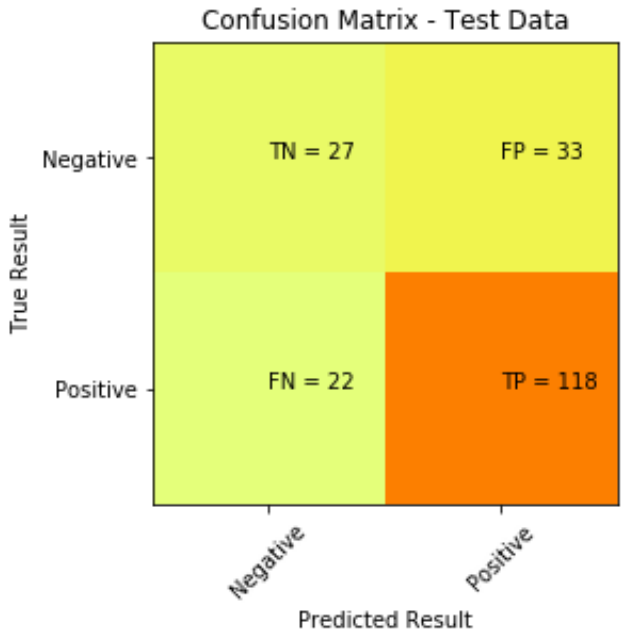

Figure 5.3: Confusion matrix of Support Vector Machine using German data set. There is an unacceptably large number of false positives.

Figures 5.7 and 5.8 are the learning curves for the German and Australian data respectively. In fig. 5.7 the training score is very high, however the validation score is not improving, which shows that the SVM model for german data is over-fitting. In fig. 5.8 the gap between the training score and validation score is very low which shows under-fitting.

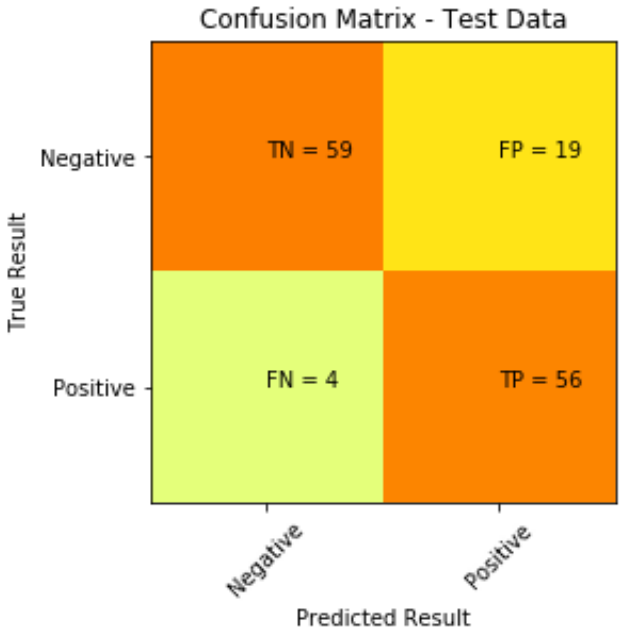

Figure 5.4: Confusion matrix of Support Vector Machine using Australian data set. There is an unacceptably large number of false positives.

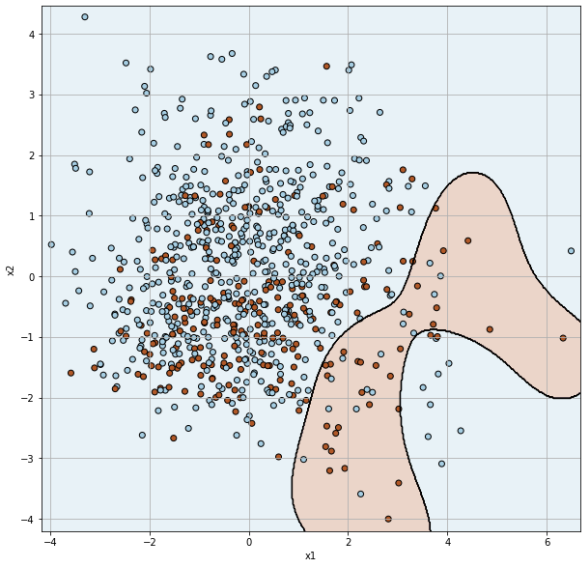

Figure 5.5: Decision boundary plot using Support Vector Machine of German data set. Observe that the decision boundary is a curve for this model.

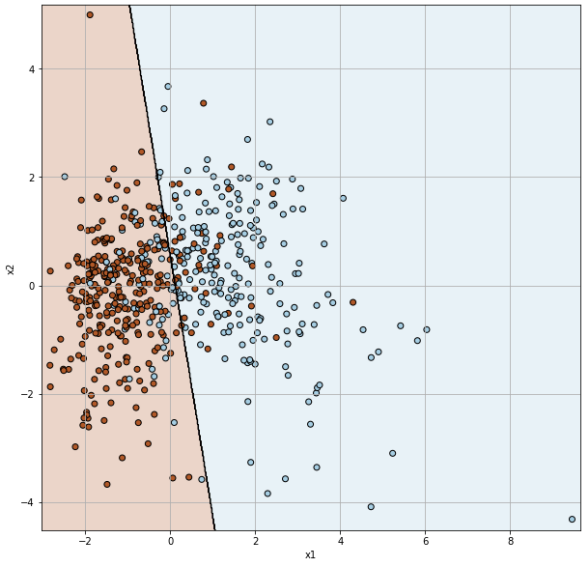

Figure 5.6: Decision boundary plot using Support Vector Machine of Australian data set. Observe that the decision boundary is a straight line for this model.

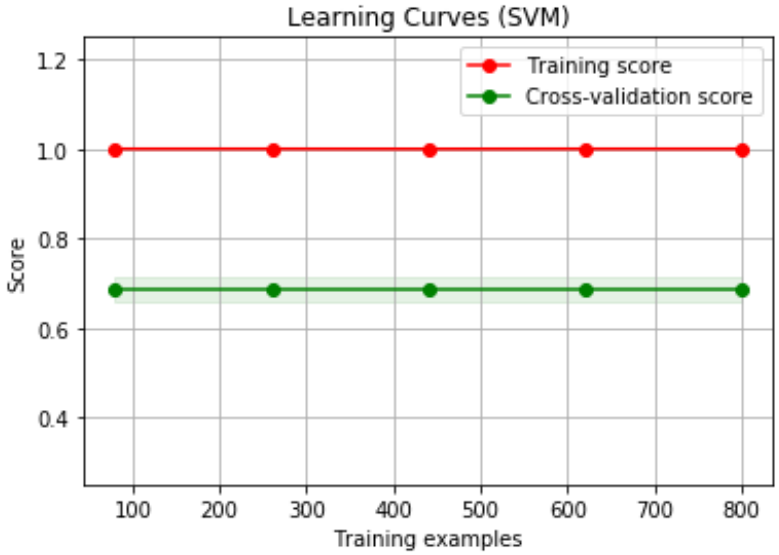

Figure 5.7: Learning curve of Support Vector Machine using the German data set. Training score is very high and cross validation score is not improving, indicating that the Support Vector Machine model is over-fitting this German data set.

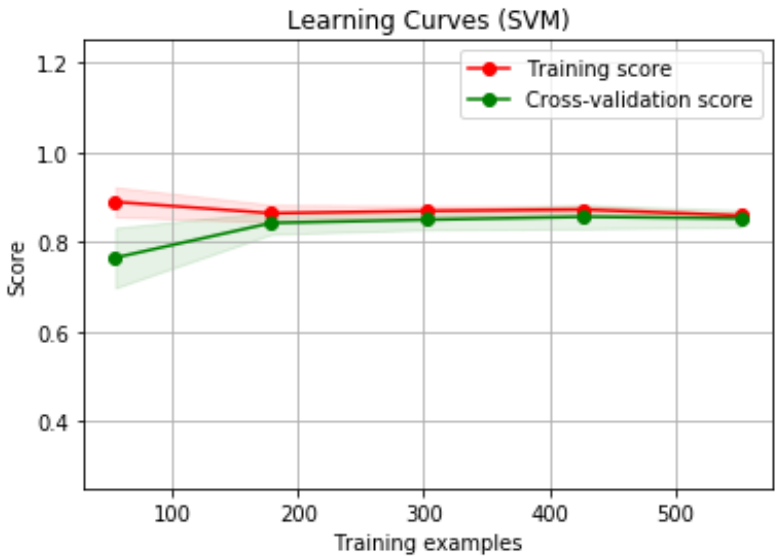

Figure 5.8: Learning curve of Support Vector Machine using the Australian data set. No gap is observed as increases the samples, indicating that the Support Vector Machine model is under-fitting this Australian data set.

Chapter 6

Classification Trees

6.1 Decision Tree

6.1.1 Introduction

A decision tree can be viewed as a flow chart, where each node represents the features of objects belonging to the node, each split represents a decision, and each terminal node represents the output. Decision tree classification can be performed using the ID3 (Iterative Dichotomiser 3) algorithm, which was invented by Ross Quinlan []. The ID3 algorithm uses entropy as a metric. CART (Classification And Regression Trees) is another algorithm to generate decision trees, which uses the Gini index as

a metric to calculate information gain. A typical decision tree is shown in fig. 6.1.

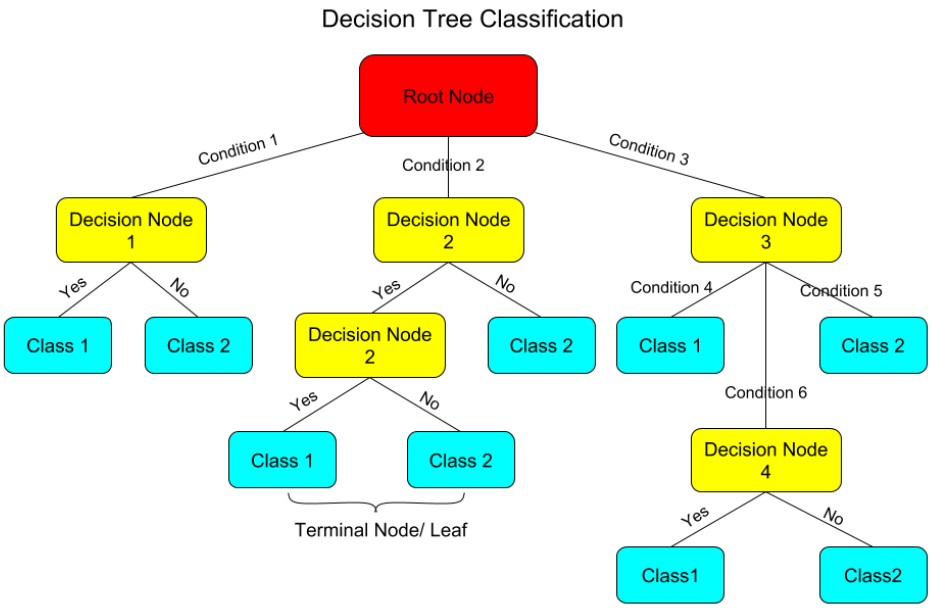

Figure 6.1: Decision Tree consists of three types of nodes; Root Node, Decision Node, and Terminal Node.

6.1.2 Mathematical Framework

Suppose we view a data set in terms of feature vectors, i.e., our $N \times p$ matrix has p feature vectors,

$$X = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \tilde{x}_1 & \tilde{x}_2 & \dots & \tilde{x}_p \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

The ID3 algorithm for finding the decision tree proceeds as follows.

1. Set X to be the entire data set.

2. Compute the entropy of X, where the entropy is defined as

$$H(X) = \sum_{c \in C} -p(c)\ \log_2 p(c). \tag{6.1}$$

   For our credit risk analysis, the response variable has binary values, $C = \{$Yes, No$\}$, and $p(c)$ is the respective probability.

3. Calculate the weighted sum entropy of a particular feature.

$$H(X \mid \tilde{x}) = \sum_{x} p(\tilde{x}) \sum_{c \in C} -p(c \mid \tilde{x})\ \log_2 p(c \mid \tilde{x})$$

4. Calculate information gain for each feature and partition X using the maximum gain.

$$IG(X, x) = \sum_{c \in C} -p(x)\ \log_2 p(x) - \sum_{x} p(x) \sum_{c \in C} -p(c \mid x)\ \log_2 p(c \mid x)$$

5. Repeat steps 2-5 with X as each child node until we get the desired decision tree.

The CART algorthm is similar except that the entropy computation, eq. (6.1) is replaced by computing the gini index,

$$G(X) = \sum_{c \in C} p(c)\,(1 - p(c))$$

6.1.3 Results

Table 6.1 shows the accuracy of this model. For both data sets, I used Gini as a metric in the start; then after using the Grid Search method, it was concluded that entropy metric is best for both data sets.

Table 6.1
Accuracy of the Decision tree model. The Decision Tree model has marginally weaker performance compared to the SVM model.

Dataset	Mean Accuracy (%)	
	Before Tuning	After Tuning
Australian data	84	85
German data	68	69

Figures 6.4 and 6.5 are the decision boundary plot of this method. The data is separated with some partitions in both the data sets.

Figures 6.6 and 6.7 are the learning curves. From these curves we can say that the model overfits the data because in the end, as the training samples increases the training score is constant however, the cross-validation is decreasing.

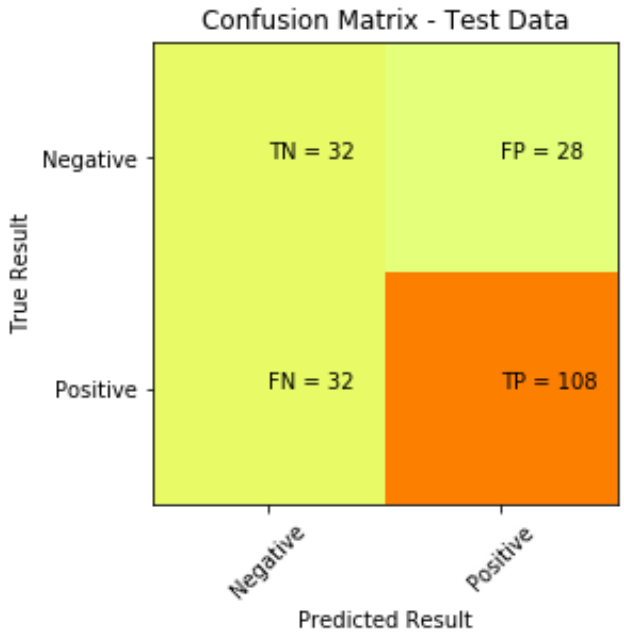

Figure 6.2: Confusion matrix of Decision Tree using German data set. There is an unacceptably large number of false positives.

6.2 Random Forest Classification

6.2.1 Methodology

The Random Forest Classification is an algorithm which collects decision trees using the random subsets of features and choosing the majority vote among them for the classification. This method minimizes overfitting and increases the overall accuracy. Figure 6.8 shows how the random forest classification looks. So, for a given data each tree will make a vote for the prediction, and the majority vote will be considered as the prediction.

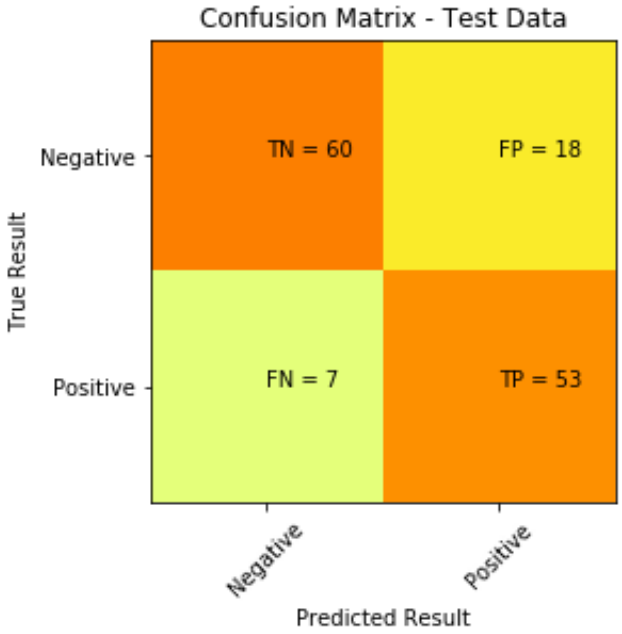

Figure 6.3: Confusion matrix of Decision Tree using Australian data set. There is an unacceptably large number of false positives.

6.2.2 Results of Random Forest

After applying this method to Australian and German Credit data set, the results are obtained as shown in table 6.2. Initially, I consider ten trees and gini metric. For tuning I consider gini and entropy as metric, and the number of trees are {10, 20, 30, 40, 50, 75, 100, 150}. But after grid search it turns out that, for the Australian data 40 trees and entropy metric are best, while for German data 30 trees and entropy metric gives better accuracy.

Figures 6.11 and 6.12 are the decision boundary plots, and figs. 6.9 and 6.10 are the confusion matrices. This method has very good accuracy and also it has less number

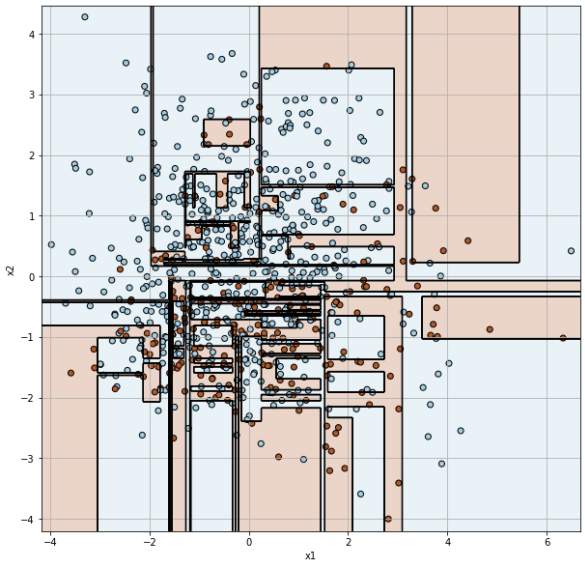

Figure 6.4: Decision boundary plot using Decision Tree of German data set. Observe that the decision boundary separates the data with the partitions for this model.

of false-positives as compared to SVM and Decision Tree methods. Figures 6.13 and 6.14 are learning curves. From the learning curves we can see that there is no high bias or high variance. Therefore this model is a good fit model.

Table 6.2
Accuracy of the Random Forest Model. The Random Forest model has marginally better performance compared to the Decision Tree model.

Dataset	Mean Accuracy (%)	
	Before Tuning	After Tuning
Australian data	87	89
German data	72	75

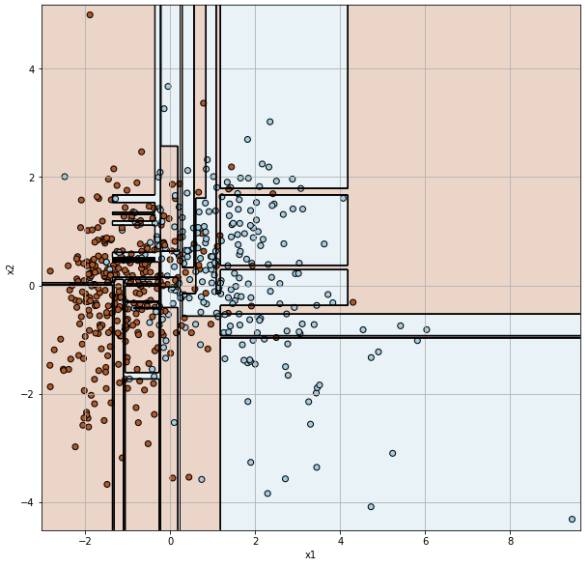

Figure 6.5: Decision boundary plot using Decision Tree of Australian data set. Observe that the decision boundary separates the data with the partitions for this model.

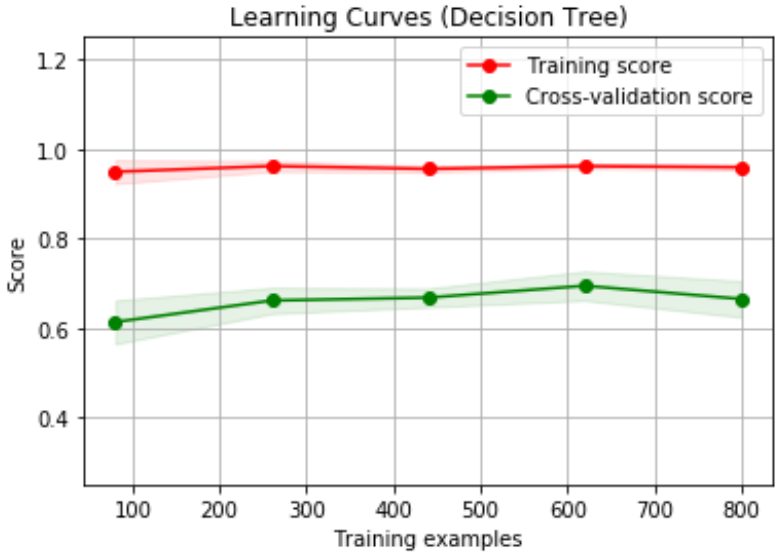

Figure 6.6: Learning curve of Decision Tree using the German data set. Training score is very high and cross validation score is decreasing after increasing training samples, indicating that the Decision Tree model is overfitting this German data set.

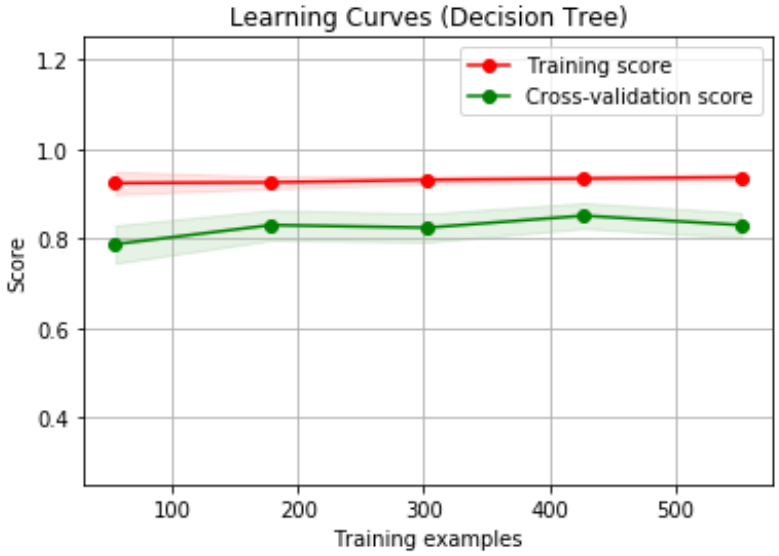

Figure 6.7: Learning curve of Decision Tree using the German data set. Training score is very high and cross validation score is decreasing after increasing training samples, indicating that the Decision Tree model is overfitting this Australian data set.

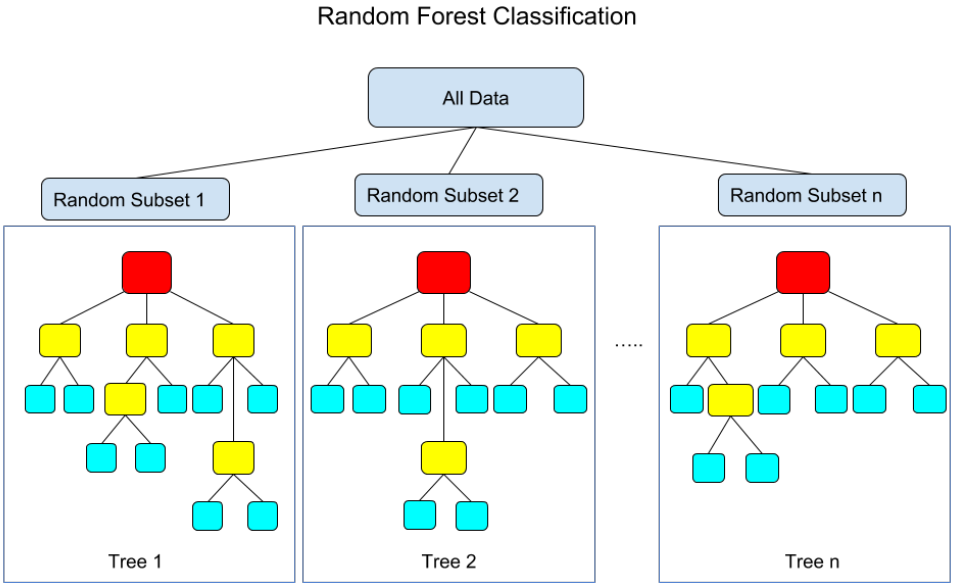

Figure 6.8: Random Forest Classification

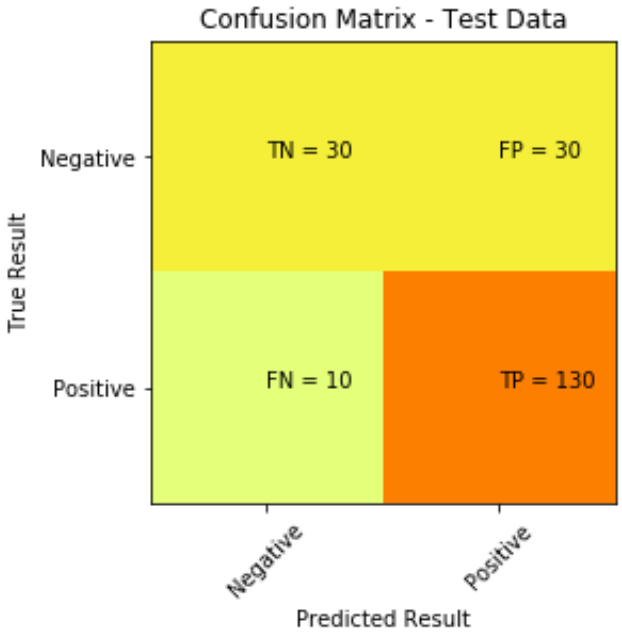

Figure 6.9: Confusion matrix of Random Forest using German data set. There is an unacceptably large number of false positives.

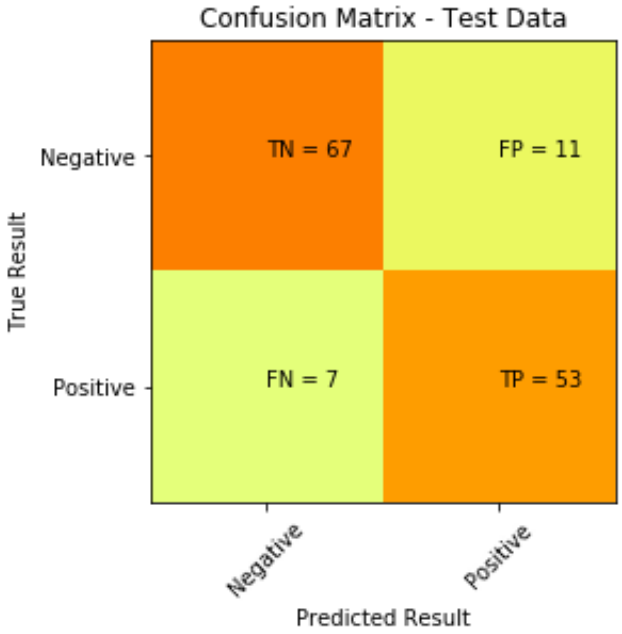

Figure 6.10: Confusion matrix of Random Forest using Australian data set. There is a little large number of false positives.

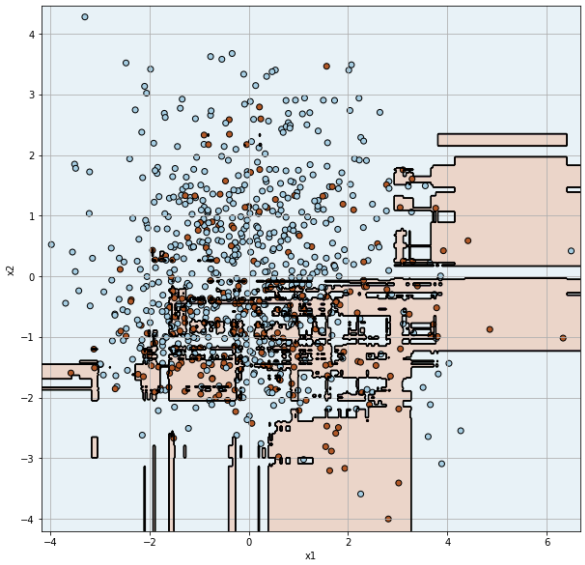

Figure 6.11: Decision boundary plot using Random Forest of German data set. Observe that the decision boundaries are very complicated for this model.

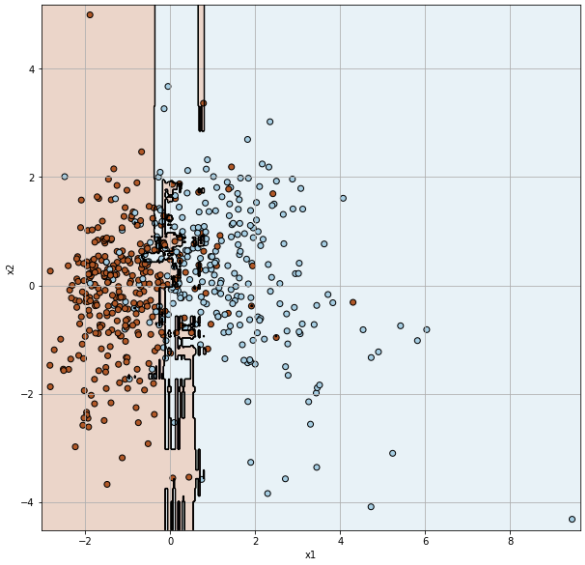

Figure 6.12: Decision boundary plot using K - Nearest Neighbors of Australian data set. The decision boundary for this data set is a lot simpler, as compared with the decision boundary for the German data set.

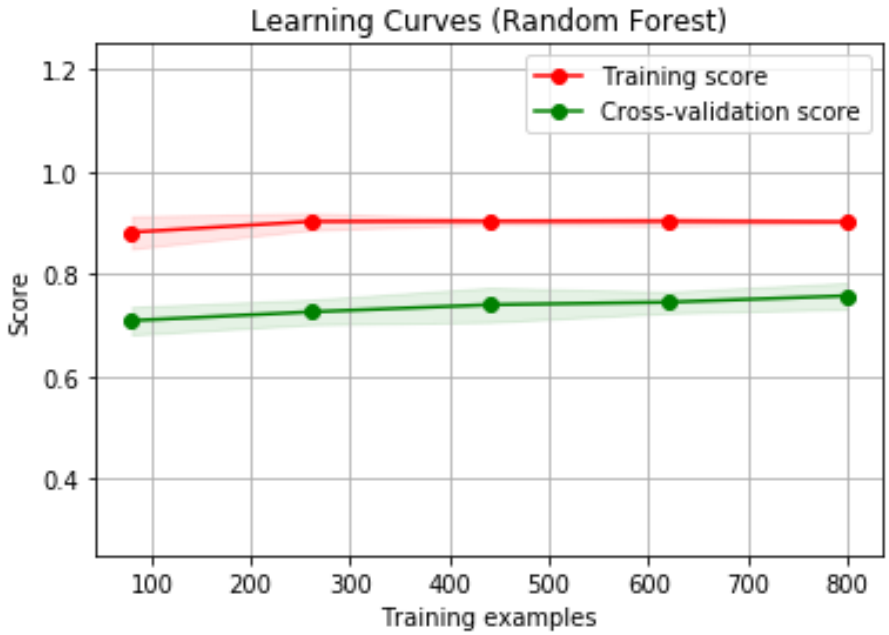

Figure 6.13: Learning curve of Random Forest using the German data set. No high bias or high variance is observed, indicating that the KNN model is not over-fitting or under-fitting this German data set.

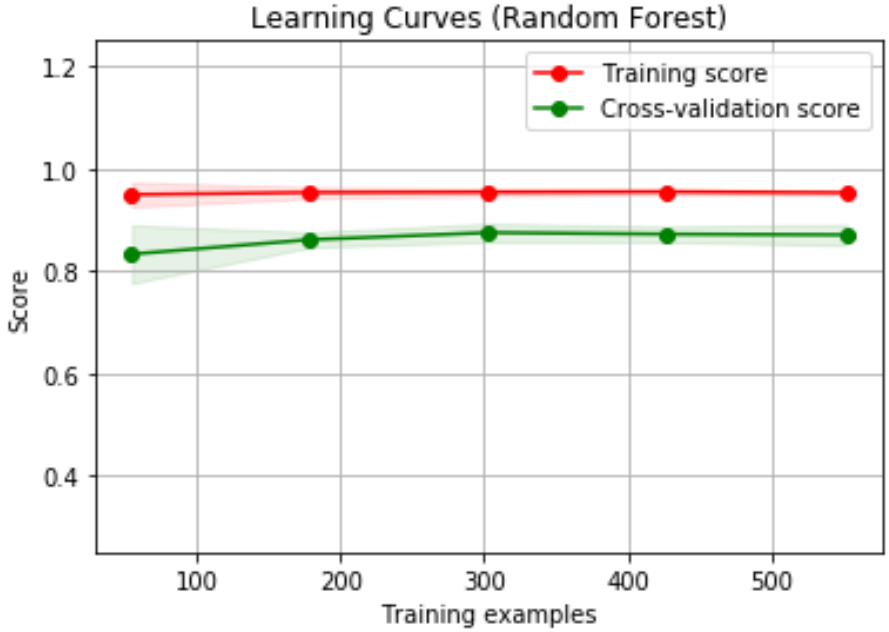

Figure 6.14: Learning curve of Random Forest using Australian data set. No high bias or high variance is observed, indicating that the KNN model is not over-fitting or under-fitting this Austraian data set.

Chapter 7

Artificial Neural Networks

7.1 Introduction

The process of learning in an artifical neural network is similar to the the learning process in a human brain. Neural networks are used in many real-world application, for example, to classify images, predict weather and in voice detection.

A perceptron is a single-layer neural network; a multi-layer neural network has more than one perceptron. In fig. 7.1, we can see the example of a simplified perceptron. This perceptron takes input data $x = (x_1, x_2)$, applies weights and a bias to the data, and outputs the sign of the resulting calculation.

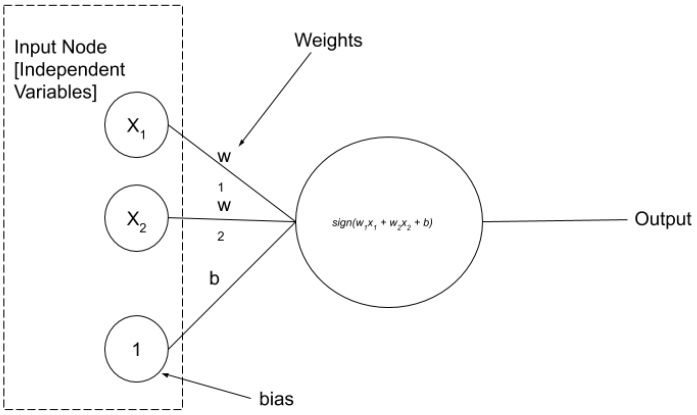

Figure 7.1: Simplified Perceptron

From chapter 5, the operation of the perceptron in fig. 7.1 is akin to classifying data based on the hyperplane defined by w and b. If the data is not linearly separable, see **??**, this perceptron is not a good classifier.

To classify the non-linear data in fig. 7.2, we need another hyperplane. Two coupled perceptrons can be used to generate two hyperplanes. Figure 7.3, gives a cartoon view of how nonlinear data can classified with two perceptrons. We will explore this mathematically shortly.

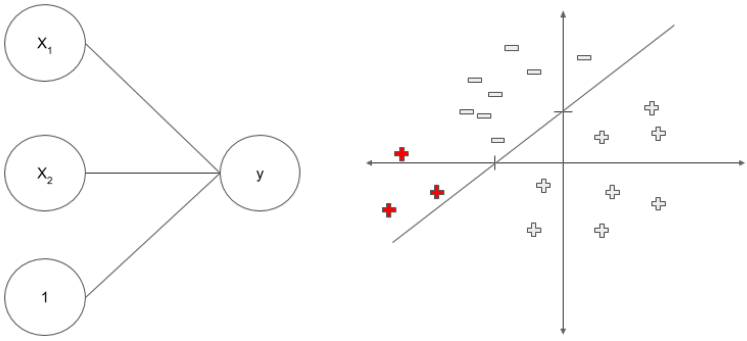

Figure 7.2: Single Perceptron Hyperplane

7.2 Basic Neural Network Model

A hidden layer is a layer between input and output. Given an input data, x, with p features, a hidden layer constructs M linear combinations of the input variables, also known as activations. Assume a constant value (often 1) has been added to the input vectors to accommodate the bias – bias is the intercept added to the linear equation. The M linear combinations of the input variables are given by

$$a_j^{(1)} = \sum_{i=1}^{p} w_{ji}^{(1)} x_i = w_j^{(1)T} x, \quad j = 1, 2, \ldots, M.$$

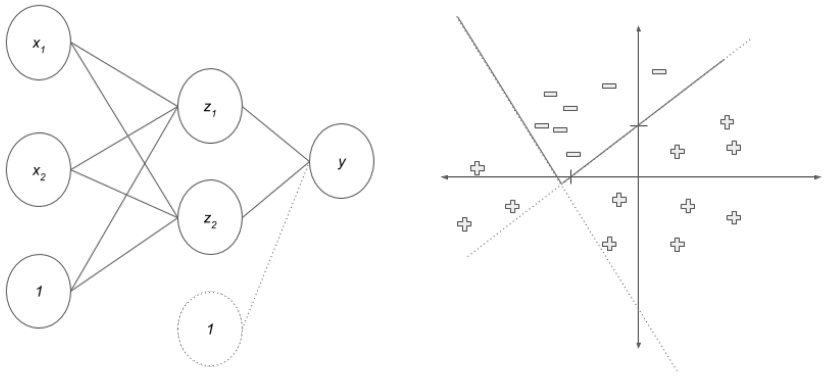

Figure 7.3: Two Perceptron hyperplanes

The superscript $^{(1)}$ denotes that these are weights for the first activation. Each activation is acted upon by a nonlinear differentiable activation function h. There are four activation functions which are widely used.

1. Sigmoid function, $h(x) = \dfrac{1}{1+e^{-x}}$;
2. ReLU (Rectified Linear Units) function, $h(x) = \max(x, 0)$;
3. Hyperbolic tangent function, $h(x) = \dfrac{1-e^{-2x}}{1+e^{-2x}}$;
4. A threshold function,

$$h(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ \\ 0 & \text{otherwise} \end{cases}.$$

The basic structure of the hidden layer is shown in fig. 7.4. The value $z_j = h(a_j)$ is referred to as hidden units.

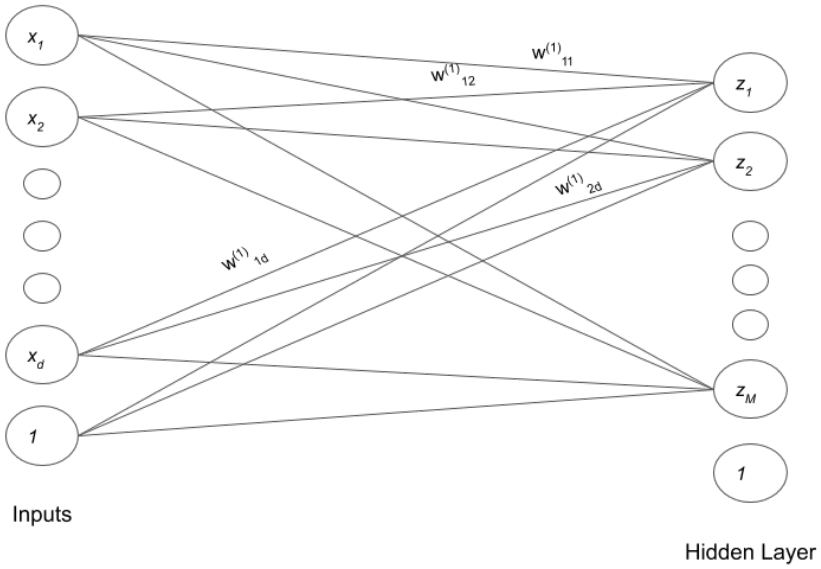

Figure 7.4: Hidden Layer

Suppose for simplicity that there are only two layers in this model: a hidden layer and an output layer, where there are K desired outputs. The output activations are

$$a_k^{(2)} = \sum_{i=1}^{M} w_{ki}^{(2)} z_i = w_k^{(2)T} z, \quad k = 1, 2, \ldots, K.$$

The output activations are acted on by another activation function,

$$y_k = h^{(2)}(a_k^{(2)}), \quad k = 1, 2, \ldots, K.$$

Notice that one can use different output activation function – the problem often dictates which output activation function is appropriate. For regression, $h^{(2)}$ is often chosen to be the identity, therefore $y_k = a_k^{(2)}$. For binary classification, one can choose $h^{(2)}$ as the Sigmoid or ReLU function; for multi-class classification, one can choose $h^{(2)}k$ as the soft-max function. This form of neural network is often refered to as the "Feed Forward Neural Network", see fig. 7.5. One can increase the number of layers, and vary the number of linear combinations in each hidden layer.

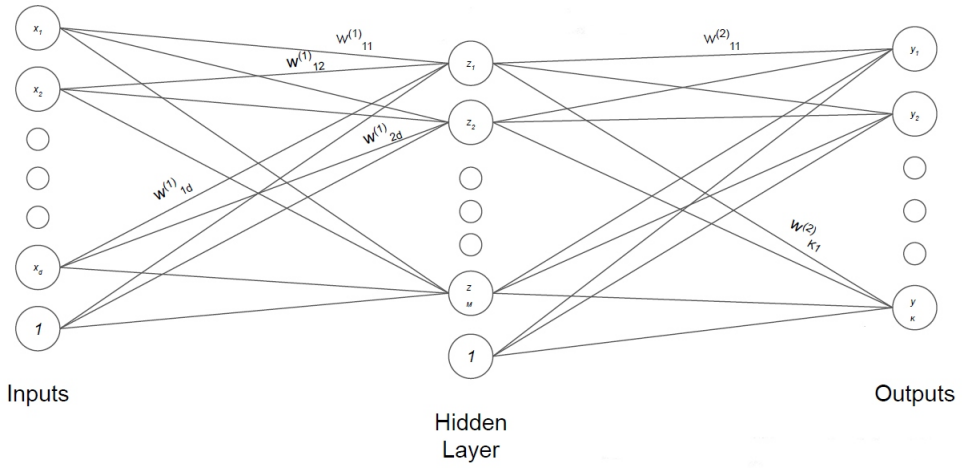

Figure 7.5: Feed Forward Neural Network

7.3 Backpropagation

Backpropagation is an algorithm to evaluate gradient descent which is used to minimize the errors. Let $E(w)$ be the Sum of Square Errors (SSE(w)). If we consider the sigmoid function $z = \sigma(a)$, where a is the perceptrons activation before applying the

logistic sigmoid: $a_i = w^T x_i + b$. Therefore,

$$E(w) = \text{SSE}(w) = \frac{1}{2}\sum_{i=1}^{N}(y_i - z_i)^2 = \frac{1}{2}\sum_{i=1}^{N}(y_i - \sigma(a_i))^2$$

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial a}\frac{\partial a}{\partial w_j} = \sum_{i=1}^{N}(z_i - y_i)\{\sigma(a_i)[1 - \sigma(a_i)]\}(x_{ij})$$

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial a}\frac{\partial a}{\partial b} = \sum_{i=1}^{N}(z_i - y_i)\{\sigma(a_i)[1 - \sigma(a_i)]\}(1)$$

$$w^{(\tau+1)} = w^{(\tau)} - \eta\sum_{i=1}^{N}(z_i - y_i)\{\sigma(a_i)[1 - \sigma(a_i)]\}(x_{ij}) \tag{7.1}$$

$$b^{(\tau+1)} = b^{(\tau)} - \eta\sum_{i=1}^{N}(z_i - y_i)\{\sigma(a_i)[1 - \sigma(a_i)]\} \tag{7.2}$$

Now, define $\delta_j = \partial E_n/\partial a_j$, where E_n is the error function evaluated with the n^{th} training sample, and a_j is the j^{th} activation. Then for the weight connecting the i^{th} neurons output (in the previous layer) to the j^th neurons input (in the current layer),

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i \tag{7.3}$$

Once we have all δs, we can compute the LHS of the above equation 7.3 for any weight in the network. The δs for the output layer are straightforward. The δs for hidden layers are given via the chain rule.

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{7.4}$$

7.3.1 Procedure for Backpropagation

1. Apply an input vector x_n and forward propagate through the network, and compute all activations and outputs throughout the network. $a_j = \sum_i w_{ji} z_i$, and $z_j = h(a_j)$.

2. Determine δ_k for all output units, $\delta_j = \frac{\partial E_n}{\partial a_j}$.

3. Backpropagate the outputs δ_k to obtain δs for all hidden units

$$\delta_j \;=\; \frac{\partial\, E_n}{\partial\, a_j} \;=\; \sum_k \frac{\partial\, E_n}{\partial\, a_k}\frac{\partial\, a_k}{\partial\, a_j} \;=\; \sum_k \delta_k \frac{\partial\, a_k}{\partial\, a_j} \;=\; h'(a_j)\sum_k w_{kj}\,\delta_k$$

4. Compute the partial derivatives shown in eq. (7.3)

This can be extended to any number of hidden layers. The steps remain the same. Backpropagation is a method of efficiently calculating the error‘s partial derivative w.r.t. each network parameter. Combining this with numerical optimization methods (like gradient descent) gives a powerful learning algorithm for neural networks.

Let's understand backpropagation using a simple multi-layer example shown in fig. 7.6.

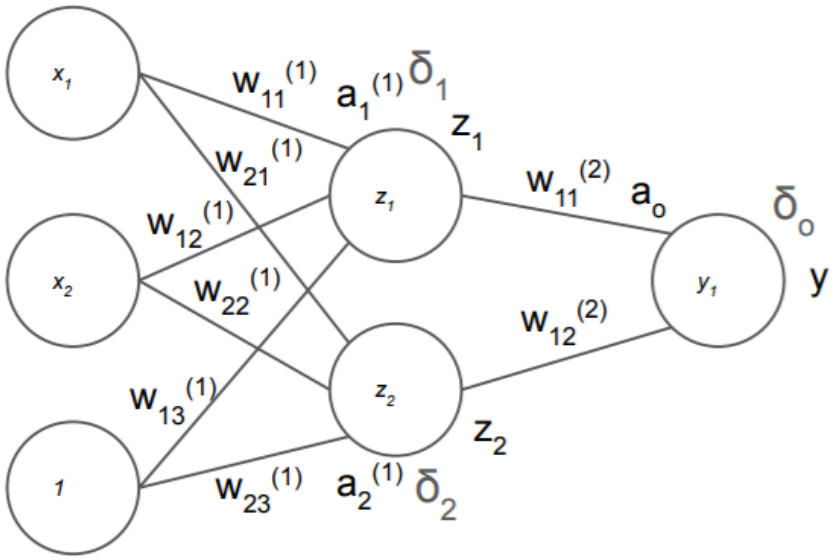

Figure 7.6: Two layer backpropagation

† First, we predict y using Feedforward.

$$a_1^{(1)} = w_{11}^{(1)} x_{n1} + w_{12}^{(1)} x_{n2} + w_{13}^{(1)}$$

$$z_1 = \sigma(a_1^{(1)})$$

$$a_2^{(1)} = w_{21}^{(1)} x_{n1} + w_{22}^{(1)} x_{n2} + w_{23}^{(1)}$$

$$z_2 = \sigma(a_1^{(2)})$$

$$a_0 = w_{11}^{(2)} Z_1 + w_{12}^{(2)} Z_2$$

$$y = \sigma(a_0)$$

† After that, we determine the output units δ

$$
\begin{aligned}
E_n &= \frac{1}{2}(y_n - \hat{y_n}) \\
\delta_0 &= \frac{\partial E_n}{\partial a_0} = \frac{\partial E_n}{\partial y}\frac{\partial y}{\partial a_0} \\
\delta_0 &= [y_n - \sigma(a_0)](\sigma(a_0)(1 - \sigma(a_0)))
\end{aligned}
$$

† Then, we will apply backpropagation to determine all δs.

$$
\begin{aligned}
\delta_j &= h'(a_j)\sum_k w_{kj}\,\delta_k \\
\delta_1 &= \sigma'(a_1)\,w_{11}^{(2)}\,\delta_0 = \delta(a_1)[1 - \delta(a_1)]\,w_{11}^{(2)}\,\delta_0 \\
\delta_2 &= \sigma'(a_2)\,w_{12}^{(2)}\,\delta_0 = \delta(a_2)[1 - \delta(a_2)]\,w_{12}^{(2)}\,\delta_0
\end{aligned}
$$

† Then, compute error derivatives w.r.t. weights using equation 7.3.

$$
\begin{aligned}
\frac{\partial E_n}{\partial w_{11}^{(2)}} &= \delta_0\,z_1 & \qquad \frac{\partial E_n}{\partial w_{12}^{(1)}} &= \delta_1\,x_2 \\
\frac{\partial E_n}{\partial w_{12}^{(2)}} &= \delta_0\,z_2 & \qquad \frac{\partial E_n}{\partial w_{22}^{(1)}} &= \delta_2\,x_2 \\
\frac{\partial E_n}{\partial w_{11}^{(1)}} &= \delta_1\,x_1 & \qquad \frac{\partial E_n}{\partial w_{13}^{(1)}} &= \delta_1\,(1) \\
\frac{\partial E_n}{\partial w_{21}^{(1)}} &= \delta_2\,x_1 & \qquad \frac{\partial E_n}{\partial w_{23}^{(1)}} &= \delta_2\,(1)
\end{aligned}
$$

† Finally, updates the weights using $w_{ab}^{(c)(\tau+1)} = w_{ab}^{(c)(\tau)} - \eta\,\frac{\partial E_n}{\partial w_{ab}^{(c)}}$

7.4 Results

Table 7.1 shows the accuracy after applying this model. In this model, three hidden layers are used. Using ReLU function in hidden layers and Sigmoid function in the output layer, the results will have high accuracy. Initially, I consider the batch size of 25 and 100 epochs. For tuning, the batch sizes are 25 and 50, and epochs are 100 and 200. After tuning the best parameters for Australian data are the batch size of 50 and 100 epochs, while it remains the same for the German data. An epoch is one cycle through the full training data set. It defines the number times that the learning algorithm will work through the entire training data set. Batch is the size of the training set.

Figures 7.9 and 7.10 are the dicision boundary plots. The data is separated by the multiple hyperplanes. Figures 7.7 and 7.8 are the confusion matrices. The false-positives for German and Australian data are 32 and 10 respectively.

Table 7.1
Accuracy of the ANN model. The ANN model has lesser accuracy compared to the Random Forest model.

Dataset	Mean Accuracy (%)	
	Before Tuning	After Tuning
Australian data	87	88
German data	75	75

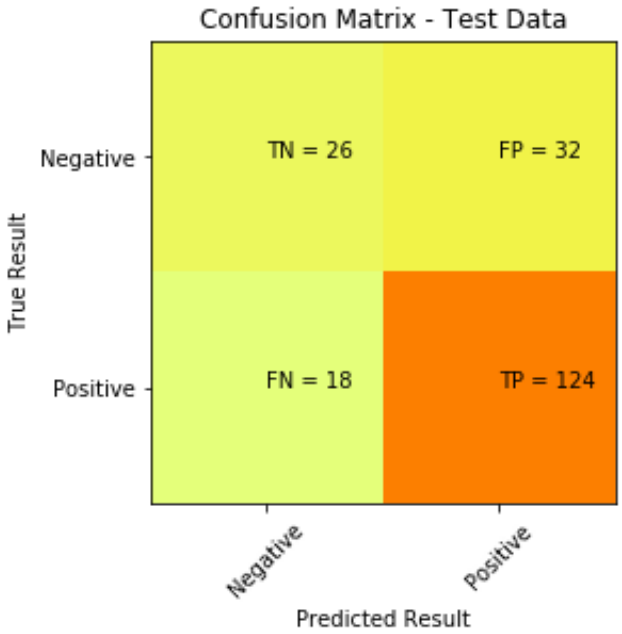

Figure 7.7: Confusion matrix of Artificial Neural Networks using German data set. There is an unacceptably large number of false positives.

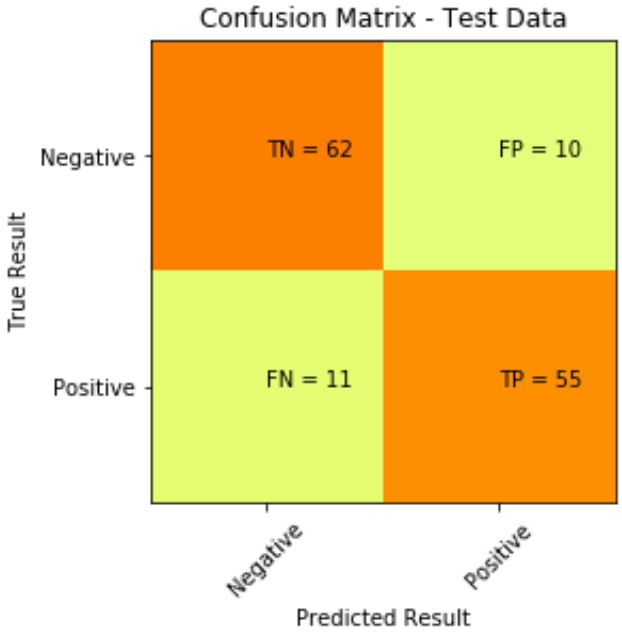

Figure 7.8: Confusion matrix of Artificial Neural Networks using Australian data set. There is a small number of false positives.

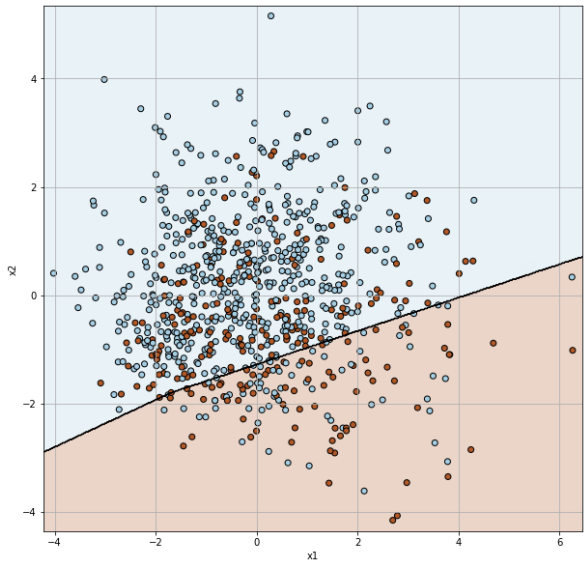

Figure 7.9: Decision boundary plot using Artificial Neural Networks of German data set. Observe that the decision boundaries are multiple hyperplanes for this model.

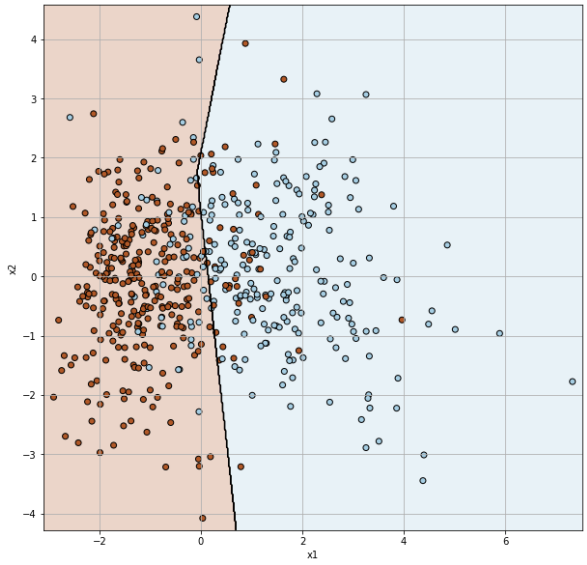

Figure 7.10: Decision boundary plot using K - Nearest Neighbors of Australian data set. Observe that the decision boundaries are multiple hyperplanes for this model.

Chapter 8

Results and Discussion

Table 8.1 shows the comparison of accuracy between our models and the models used in the paper [4]. Table 8.2 shows the comparison of false positives and false negatives after applying various classification models.

Table 8.1
Comparison Table. Comparison of accuracy between our models and the models used in the paper

Models	German data (%)		Australian data (%)	
	Our Results	From paper	Our Results	From paper
KNN	75	72	85	89
Logistic Regression	77	-	87	-
Naive Bayes	75	77	87	78
SVM	77	78	87	85
Decision Trees	69	85	85	90
Random Forest	75	-	88	-
ANN	75	77	77	82

From the table 8.1, we can see that the Random Forest classification model gives

Table 8.2
Comparison Table of False Positives and False Negatives

Models	German dataset		Australian dataset	
	False Positives	False Negatives	False Positives	False Negatives
KNN	28	17	8	12
Logistic Regression	30	25	14	8
Naive Bayes	27	23	8	10
SVM	33	22	19	4
Decision Tree	28	32	18	7
Random Forest	30	10	11	7
ANN	32	18	10	11

the best accuracy for the Australian credit dataset. However, SVM and Logistic Regression give best accuracy for the German credit dataset.

From the table 8.2, it seems that KNN and Naive Bayes models give least false-positives for the Australian dataset. For the German dataset, Naive Bayes model gives least false-positives. False-negatives is we predict a customer non-credible instead of credible. Large number of false-negative affects the profit. But, if we consider less risk then Naive Bayes model is best for both the datasets, and KNN model is also good model for both the datasets.

References

[1] UC Irvine Machine Learning Repository Australian Credit Data. http://archive.ics.uci.edu/ml/datasets/Statlog+%28Australian+Credit+Approval%29.

[2] UC Irvine Machine Learning Repository German Credit Data. https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data).

[3] Principal Component Analysis. https://en.wikipedia.org/wiki/Principal_component_analysis/.

[4] Suman Kumar Mohapatra Trilok Nath Pandey, Alok Kumar Jagadev and Satchidananda Dehuri. Credit Risk Analysis using Machine Learning Classifiers. *IEEE*, 2017.

[5] Udemy course: Machine Learning A-Z by Kirill Eremenko, Hadelin de Ponteves. https://www.udemy.com/machinelearning/.

[6] Artificial Intelligence - All in One by Andrew Ng.

https://www.youtube.com/playlist?list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN/.

[7] Kaushik Roy Prajesh P. Anchalia. The k-Nearest Neighbor Algorithm Using MapReduce Paradigm. *IEEE*, 2014.

Appendix A

Sample Code

A.1 Machine_Learning.py

```
# Problem Description
# Machine_Learning.py
# Description:
# Predict whether the customer is creditable or not.

# Two datasets: 1) German Credit dataset 2) Australian ↩
   Credit dataset
# Six Machine Learning Classification Methods:
# 1) Logistic Regression - lr
# 2) K - Nearest Neighbors Classification - knn
# 3) Support Vector Machine Classification - svc
# 4) Decision Tree Classification - dt
```

```
# 5) Random Forest Classification - rf
# 6) Naive Bayes Classification - nb

# Special requirements or dependencies:
# None; Tested in Mac OS X with Python 2.7
# Compilation and execution:
# Compilation not necessary
# Execution takes approx 100-120 seconds on most modern ↩
   hardware.

# For the execution in terminal
# python Machine_Learning.py



# Import the library
import pandas as pd

# Import the dataset

#dataset = pd.read_excel('German_Credit_Data.xlsx')
#dataset.drop(["Acc","Telephone","Gender"],axis = 1, ↩
   inplace = True)
dataset = pd.read_excel('Australian_Credit_Data.xlsx')

# Data Pre-Processing(Splitting data, Categorical data, ↩
   Feature Scaling)
from Functions import data_preprocessing
X_train,X_test,y_train,y_test,X,y = data_preprocessing(↩
   dataset)

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier_lr = LogisticRegression()
```

```
classifier_lr.fit(X_train, y_train)

# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier_knn = KNeighborsClassifier(n_neighbors = 5, ↩
   metric = 'euclidean')
classifier_knn.fit(X_train, y_train)

# Fitting Kernel SVM to the Training set
from sklearn.svm import SVC
classifier_svc = SVC(kernel = 'poly', degree = 2)
classifier_svc.fit(X_train, y_train)

# Fitting Decision Tree Classification to the Training ↩
   set
from sklearn.tree import DecisionTreeClassifier
classifier_dt = DecisionTreeClassifier(criterion = 'gini↩
   ', min_samples_split = 10)
classifier_dt.fit(X_train, y_train)

# Fitting Random Forest Classification to the Training ↩
   set
from sklearn.ensemble import RandomForestClassifier
classifier_rf = RandomForestClassifier(n_estimators = ↩
   10, criterion = 'entropy', min_samples_split = 2)
classifier_rf.fit(X_train, y_train)

# Fitting Naive Bayes Classification to the Training set
from sklearn.naive_bayes import BernoulliNB
classifier_nb = BernoulliNB(alpha = 1, binarize = 0.0 )
classifier_nb.fit(X_train, y_train)

# Predicting the Test set results
y_pred_lr  = classifier_lr.predict(X_test)
```

```
y_pred_knn = classifier_knn.predict(X_test)
y_pred_svc = classifier_svc.predict(X_test)
y_pred_dt  = classifier_dt.predict(X_test)
y_pred_rf  = classifier_rf.predict(X_test)
y_pred_nb  = classifier_nb.predict(X_test)

# Plot Confusion Matrix before tuning
from Functions import  con_mat_plot
print('Confusion Matrix for Logistic Regression')
con_mat_plot(y_test,y_pred_lr)
print('Confusion Matrix for KNN')
con_mat_plot(y_test,y_pred_knn)
print('Confusion Matrix for SVM')
con_mat_plot(y_test,y_pred_svc)
print('Confusion Matrix for Decision Tree')
con_mat_plot(y_test,y_pred_dt)
print('Confusion Matrix for Random Forest')
con_mat_plot(y_test,y_pred_rf)
print('Confusion Matrix for Naive Bayes')
con_mat_plot(y_test,y_pred_nb)


# Applying K-Fold Cross Validation
from Functions import  k_fold_cross
accuracies_lr  = k_fold_cross(classifier_lr, X_train, ↩
   y_train)
accuracies_knn = k_fold_cross(classifier_knn, X_train, ↩
   y_train)
accuracies_svc = k_fold_cross(classifier_svc, X_train, ↩
   y_train)
accuracies_dt  = k_fold_cross(classifier_dt, X_train, ↩
   y_train)
accuracies_rf  = k_fold_cross(classifier_rf, X_train, ↩
   y_train)
```

```
accuracies_nb  = k_fold_cross(classifier_nb, X_train, ↩
   y_train)

print('LR: Mean Accuracy before tuning',accuracies_lr.↩
   mean()*100.)
print('KNN :Mean Accuracy before tuning',accuracies_knn.↩
   mean()*100.)
print('SVC :Mean Accuracy before tuning',accuracies_svc.↩
   mean()*100.)
print('Decision Tree :Mean Accuracy before tuning',↩
   accuracies_dt.mean()*100.)
print('Random Forest :Mean Accuracy before tuning',↩
   accuracies_rf.mean()*100.)
print('Naive Bayes :Mean Accuracy before tuning',↩
   accuracies_nb.mean()*100.)

# Applying Grid Search to find the best model and the ↩
   best parameters
from Functions import  choosing_parameters


# Best parameters for Logistic Regression

parameters_lr = [{'C': [1, 5, 10], 'tol': [1e-4,1e-5,1e↩
   -6,1e-10]}]
best_para_lr  = choosing_parameters(parameters_lr,↩
   classifier_lr,X_train,y_train)
C_lr          = best_para_lr[0]
tole_lr       = best_para_lr[1]

# Best parameters for KNN
parameters_knn = [{'n_neighbors': [3, 5, 7, 3, 9, 11, ↩
   13], 'metric': ['minkowski'],
                   'p': [1,2,3,4,5,6,7]}]
```

```
best_para_knn  = choosing_parameters(parameters_knn,↩
   classifier_knn,X_train,y_train)
N_knn          = best_para_knn[0]
metric_knn     = best_para_knn[1]
p_knn          = best_para_knn[2]

# Best parameters for SVM
parameters_svc = [{'kernel': ['rbf','poly'], 'degree': ↩
   [1, 2, 3, 4, 5, 6, 7]}]
best_para_svc  = choosing_parameters(parameters_svc,↩
   classifier_svc,X_train,y_train)
ker_svc        = best_para_svc[0]
deg_svc        = best_para_svc[1]

# Best parameters for Decision Tree
parameters_dt = [{'criterion': ['gini','entropy'], '↩
   min_samples_split': [2,4,6,8,10,15]}]
best_para_dt  = choosing_parameters(parameters_dt,↩
   classifier_dt,X_train,y_train)
criteria_dt   = best_para_dt[1]
min_split_dt  = best_para_dt[0]


# Best parameters for Random Forest
parameters_rf   = [{'criterion': ['gini','entropy'],'↩
   min_samples_split': [2,5, 10, 15, 20],
                    'n_estimators': [10, 20, 30, 40, ↩
                       50,100,150]}]
best_para_rf    = choosing_parameters(parameters_rf,↩
   classifier_rf,X_train,y_train)
min_samp_spl_rf = best_para_rf[0]
n_est_rf        = best_para_rf[1]
criterion_rf    = best_para_rf[2]
```

```
# Best parameters for Naive bayes
parameters_nb = [{'alpha': [1, 5, 7, 10, 20, 50, 60, 100],
                 'binarize': [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]}]
best_para_nb  = choosing_parameters(parameters_nb,classifier_nb,X_train,y_train)
alfa_nb       = best_para_nb[1]
bi_nb         = best_para_nb[0]

#Selecting the best parameters and apply the classification methods

# Logistic Regression
classifier_lr = LogisticRegression(C = int(C_lr), tol = float(tole_lr) )
classifier_lr.fit(X_train, y_train)

# KNN
classifier_knn = KNeighborsClassifier(n_neighbors = int(N_knn), metric = str(metric_knn), p = int(p_knn))
classifier_knn.fit(X_train, y_train)

# SVM
classifier_svc = SVC(kernel = str(ker_svc), degree = int(deg_svc))
classifier_svc.fit(X_train, y_train)

# Decision Tree Classification
classifier_dt = DecisionTreeClassifier(criterion = str(criteria_dt), min_samples_split = int(min_split_dt))
classifier_dt.fit(X_train, y_train)

# Random Forest Classification
```

```
classifier_rf = RandomForestClassifier(n_estimators = ↩
   int(n_est_rf), criterion = str(criterion_rf),
                                       min_samples_split = ↩
                                          int(↩
                                          min_samp_spl_rf))
classifier_rf.fit(X_train, y_train)

# Naive Bayes Classification
classifier_nb = BernoulliNB(alpha = int(alfa_nb), ↩
   binarize = float(bi_nb))
classifier_nb.fit(X_train, y_train)


# Predict the Test set results
y_pred_lr = classifier_lr.predict(X_test)
y_pred_knn = classifier_knn.predict(X_test)
y_pred_svc = classifier_svc.predict(X_test)
y_pred_dt = classifier_dt.predict(X_test)
y_pred_rf = classifier_rf.predict(X_test)
y_pred_nb = classifier_nb.predict(X_test)

# Create Confusion Matrix
from Functions import  con_mat_plot
print('Confusion Matrix for Logistic Regression')
con_mat_plot(y_test,y_pred_lr)
print('Confusion Matrix for KNN')
con_mat_plot(y_test,y_pred_knn)
print('Confusion Matrix for SVM')
con_mat_plot(y_test,y_pred_svc)
print('Confusion Matrix for Decision Tree')
con_mat_plot(y_test,y_pred_dt)
print('Confusion Matrix for Random Forest')
con_mat_plot(y_test,y_pred_rf)
print('Confusion Matrix for Naive Bayes')
```

```
con_mat_plot(y_test,y_pred_nb)

# Apply K-Fold Cross Validation
from Functions import  k_fold_cross
accuracies_lr  = k_fold_cross(classifier_lr, X_train, ↩
   y_train)
accuracies_knn = k_fold_cross(classifier_knn, X_train, ↩
   y_train)
accuracies_svc = k_fold_cross(classifier_svc, X_train, ↩
   y_train)
accuracies_dt  = k_fold_cross(classifier_dt, X_train, ↩
   y_train)
accuracies_rf  = k_fold_cross(classifier_rf, X_train, ↩
   y_train)
accuracies_nb  = k_fold_cross(classifier_nb, X_train, ↩
   y_train)
print('LR: Mean Accuracy after tuning',accuracies_lr.↩
   mean()*100.)
print('KNN :Mean Accuracy after tuning',accuracies_knn.↩
   mean()*100.)
print('SVM :Mean Accuracy after tuning',accuracies_svc.↩
   mean()*100.)
print('Decision Tree :Mean Accuracy after tuning',↩
   accuracies_dt.mean()*100.)
print('Random Forest :Mean Accuracy after tuning',↩
   accuracies_rf.mean()*100.)
print('Naive Bayes :Mean Accuracy after tuning',↩
   accuracies_nb.mean()*100.)

from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

```
# Logistic Regression
classifier_lr = LogisticRegression(C = int(C_lr), tol = ↩
   float(tole_lr) )
classifier_lr.fit(X_train, y_train)

# KNN
classifier_knn = KNeighborsClassifier(n_neighbors = int(↩
   N_knn), metric = str(metric_knn), p = int(p_knn))
classifier_knn.fit(X_train, y_train)

# SVM
classifier_svc = SVC(kernel = str(ker_svc), degree = int↩
   (deg_svc))
classifier_svc.fit(X_train, y_train)

# Decision Tree Classification
classifier_dt = DecisionTreeClassifier(criterion = str(↩
   criteria_dt), min_samples_split = int(min_split_dt))
classifier_dt.fit(X_train, y_train)

# Random Forest Classification
classifier_rf = RandomForestClassifier(n_estimators = ↩
   int(n_est_rf), criterion = str(criterion_rf),
                                       min_samples_split = ↩
                                          int(↩
                                          min_samp_spl_rf))
classifier_rf.fit(X_train, y_train)

# Naive Bayes Classification
classifier_nb1 = BernoulliNB(alpha = int(alfa_nb), ↩
   binarize = float(bi_nb))
classifier_nb1.fit(X_train, y_train)

from sklearn.naive_bayes import GaussianNB
```

```
classifier_nb = GaussianNB()
classifier_nb.fit(X_train, y_train)

# Decision boundary plot

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_lr ,X_train, y_train, ↩
   X_test, cmap='Paired_r')

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_knn ,X_train, y_train,↩
    X_test, cmap='Paired_r')

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_svc ,X_train, y_train,↩
    X_test, cmap='Paired_r')

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_dt ,X_train, y_train, ↩
   X_test, cmap='Paired_r')

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_rf ,X_train, y_train, ↩
   X_test, cmap='Paired_r')

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_nb ,X_train, y_train, ↩
   X_test, cmap='Paired_r')

# Learning curve

from sklearn.model_selection import ShuffleSplit
cv = ShuffleSplit(test_size=0.2, random_state = 0)
```

```
from Functions import plot_learning_curve
title = "Learning Curves (Logistic Regression)"
estimator = classifier_lr
plot_learning_curve(estimator, title, X, y, ylim=(0.25, 1.25), cv=cv, n_jobs=4)

title = "Learning Curves (KNN)"
estimator = classifier_knn
plot_learning_curve(estimator, title, X, y, (0.25, 1.25), cv=cv, n_jobs=4)

title = "Learning Curves (SVM)"
estimator = classifier_svc
plot_learning_curve(estimator, title, X, y, (0.25, 1.25), cv=cv, n_jobs=4)

title = "Learning Curves (Decision Tree)"
estimator = classifier_dt
plot_learning_curve(estimator, title, X, y, (0.25, 1.25), cv=cv, n_jobs=4)

title = "Learning Curves (Random Forest)"
estimator = classifier_rf
plot_learning_curve(estimator, title, X, y, (0.25, 1.25), cv=cv, n_jobs=4)

title = "Learning Curves (Naive Bayes)"
estimator = classifier_nb1
plot_learning_curve(estimator, title, X, y, (0.25, 1.25), cv=cv, n_jobs=4)
```

A.2 Functions.py

```
# importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit

# Function for Plotting:

def plot_classification(X_train,X_test,y_train,y_test,↩
   y_pred):
    from sklearn.decomposition import PCA
    pca = PCA(n_components = 2)
    X_train = pca.fit_transform(X_train)
    X_test = pca.transform(X_test)
    explained_variance = pca.explained_variance_ratio_


    comparision = pd.DataFrame.join(pd.DataFrame(y_test,↩
       columns=['y_test']),pd.DataFrame(y_pred,dtype=int,↩
       columns=['y_pred']))
    comparision['Comparision'] = comparision.apply(↩
       lambda x: 0 if x[0] == x[1] else 1, axis=1)

    plotdata = pd.DataFrame.join(pd.DataFrame(X_test,↩
       columns=['0','1']),comparision['Comparision'])


    # Plot misclassification
```

```
    fig3, ax3 = plt.subplots(figsize = (14,7))

    ax3.scatter(x = plotdata[plotdata.Comparision ==0]['↩
       0'],y = plotdata[plotdata.Comparision == 0]['1'], ↩
       marker = 'o',color = 'red')
    ax3.scatter(x = plotdata[plotdata.Comparision ==1]['↩
       0'],y = plotdata[plotdata.Comparision == 1]['1'], ↩
       marker = 'o',color = 'blue')

    ax3.legend(['Classified','Misclassified'])
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.title('Classification')
    plt.show()

# Function for Confusion Matrix:

def con_mat_plot(y_test,y_pred):
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.clf()
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.↩
       Wistia)
    classNames = ['Negative','Positive']
    plt.title('Confusion Matrix - Test Data')
    plt.ylabel('True Result')
    plt.xlabel('Predicted Result')
    tick_marks = np.arange(len(classNames))
    plt.xticks(tick_marks, classNames, rotation=45)
    plt.yticks(tick_marks, classNames)
    s = [['TN','FP'], ['FN', 'TP']]

    for i in range(2):
        for j in range(2):
```

```
            plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j↩
               ]))
    plt.show()

# Function k-fold cross validation
def k_fold_cross(classifier, X_train, y_train):
    from sklearn.model_selection import cross_val_score
    accuracies = cross_val_score(estimator = classifier,↩
        X = X_train, y = y_train, cv = 10)
    return accuracies.mean()

# Data Preprocessing
def data_preprocessing(dataset):
    listt = dataset.select_dtypes(include=['category', ↩
       object]).columns
    X = dataset.iloc[:, :-1].values
    y = dataset.iloc[:, -1].values

    for i in range(0,len(y)):
        if y[i] == 2:
            y[i] = 0


    listn = np.empty((len(listt),1))
    for i in range(0,len(listt)):
        listn[i,0] = dataset.columns.get_loc(listt[i])

    # Categorial data
    from sklearn.preprocessing import LabelEncoder, ↩
       OneHotEncoder
    labelencoder_X = LabelEncoder()

    for i in range(0,len(listn)):
```

```
        X[:,int(listn[i,0])] = labelencoder_X.↩
           fit_transform(X[:,int(listn[i,0])])
        onehotencoder        = OneHotEncoder(↩
           categorical_features = [int(listn[i,0])])

    # Splitting the dataset into the Training set and ↩
       Test set
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(↩
       X, y, test_size = 0.2)


    # Feature Scaling
    from sklearn.preprocessing import StandardScaler
    sc      = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test  = sc.transform(X_test)
    return X_train,X_test,y_train,y_test,X,y

# Function for selecting the best parameters:

def choosing_parameters(parameters,classifier,X_train,↩
   y_train):
    from sklearn.model_selection import GridSearchCV
    grid_search = GridSearchCV(estimator = classifier,
                               param_grid = parameters,
                               scoring = 'accuracy',
                               cv = 10,
                               n_jobs = -1)
    grid_search = grid_search.fit(X_train, y_train)
    best_accuracy = grid_search.best_score_
    best_parameters = grid_search.best_params_
    best_para = np.array(best_parameters.values())
    return best_para
```

```
# Function for decision boundary plot


def plot_decision_boundary(clf, X, Y, X_test, cmap='↩
   Paired_r'):
    clf.predict(X_test)
    h = 0.02
    x_min, x_max = X[:,0].min() - 10*h, X[:,0].max() + ↩
       10*h
    y_min, y_max = X[:,1].min() - 10*h, X[:,1].max() + ↩
       10*h
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(10,10))
    plt.contourf(xx, yy, Z, cmap=cmap, alpha=0.25)
    plt.contour(xx, yy, Z, colors='k', linewidths=0.7)
    plt.ylabel('x2')
    plt.xlabel('x1')
    plt.grid()
    plt.scatter(X[:,0], X[:,1], c=Y, cmap=cmap, ↩
       edgecolors='k');

# Function for learning curve

def plot_learning_curve(estimator, title, X, y, ylim=↩
   None, cv=None,
                        n_jobs=None, train_sizes=np.↩
                           linspace(.1, 1.0, 5)):

    plt.figure()
```

```
plt.title(title)
if ylim is not None:
    plt.ylim(*ylim)
plt.xlabel("Training examples")
plt.ylabel("Score")
train_sizes, train_scores, test_scores = ↩
   learning_curve(
    estimator, X, y, cv=cv, n_jobs=n_jobs, ↩
       train_sizes=train_sizes)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - ↩
   train_scores_std,
                 train_scores_mean + ↩
                    train_scores_std, alpha=0.1,
                 color="r")
plt.fill_between(train_sizes, test_scores_mean - ↩
   test_scores_std,
                 test_scores_mean + test_scores_std,↩
                     alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color↩
   ="r",
         label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color=↩
   "g",
         label="Cross-validation score")

plt.legend(loc="best")
return plt
```

A.3 ANN.py

```
# Problem Description
# ANN.py
# Description:
# Predict whether the customer is creditable or not.

# Two datasets: 1) German Credit dataset 2) Australian Credit dataset
# Artificial Neural Network


# Special requirements or dependencies:
# None; Tested in Mac OS X with Python 2.7
# Compilation and execution:
# Compilation not necessary
# Execution takes approx 8-10 minutes on most modern hardware.

# For the execution in terminal
# python ANN.py

# Import the library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Import the dataset

dataset = pd.read_excel('German_Credit_Data.xlsx')
#dataset = pd.read_excel('Australian_Credit_Data.xlsx')
```

```
# Data Pre-Processing(Splitting data, Categorical data, ↩
   Feature Scaling)
from Functions import data_preprocessing
X_train,X_test,y_train,y_test,X,y = data_preprocessing(↩
   dataset)


# Make the ANN!

# Importing the Keras libraries and packages

import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier

def ann_model():

    # Initialising the ANN
    classifier = Sequential()

    # Adding the input layer and the first hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu', input_dim = np.↩
       shape(X_test)[1]))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu'))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu'))
```

```
    # Adding the output layer
    classifier.add(Dense(1, kernel_initializer = '↩
       uniform', activation = 'sigmoid'))

    # Compiling the ANN
    classifier.compile(optimizer = 'adam', loss = '↩
       binary_crossentropy', metrics = ['accuracy'])

    return classifier

# Fitting the ANN to the Training set
classifier = KerasClassifier(build_fn = ann_model, ↩
   batch_size = 25, epochs = 100)

# Cross validation
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score

kfold = StratifiedKFold(n_splits=10, shuffle=True, ↩
   random_state = 2)
results = cross_val_score(classifier, X_test, y_test, cv↩
   =kfold)
print(results.mean())


# Making predictions and evaluating the model
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

from Functions import  con_mat_plot
```

```
con_mat_plot(y_test,y_pred)



# Tuning the ANN
import keras
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
def build_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(units = 10, kernel_initializer ↩
      = 'uniform', activation = 'relu', input_dim = np.↩
      shape(X_test)[1]))
    classifier.add(Dropout(p = 0.1))
    classifier.add(Dense(units = 10, kernel_initializer ↩
      = 'uniform', activation = 'relu'))
    classifier.add(Dropout(p = 0.1))
    classifier.add(Dense(units = 1, kernel_initializer =↩
       'uniform', activation = 'sigmoid'))
    classifier.compile(optimizer = optimizer, loss = '↩
      binary_crossentropy', metrics = ['accuracy'])
    return classifier


classifier = KerasClassifier(build_fn = build_classifier↩
  )
parameters = {'batch_size' : [25, 50],
              'epochs' : [100, 200],
              'optimizer' : ['adam' , 'rmsprop']}

grid_search = GridSearchCV(estimator = classifier,
```

```
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10)
grid_search = grid_search.fit(X_train, y_train)
best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_
print('Best Accuracy = ',best_accuracy)
best_parameters = np.array(best_parameters.values())
epo             = int(best_parameters[0])
opt             = str(best_parameters[1])
bs              = int(best_parameters[2])

# Defining model with tuned parameteres

def ann_model():

    # Initialising the ANN
    classifier = Sequential()

    # Adding the input layer and the first hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu', input_dim = np.↩
       shape(X_test)[1]))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu'))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu'))

    # Adding the output layer
```

```
    classifier.add(Dense(1, kernel_initializer = '↩
        uniform', activation = 'sigmoid'))

    # Compiling the ANN
    classifier.compile(optimizer = opt, loss = '↩
        binary_crossentropy', metrics = ['accuracy'])

    return classifier

# Fitting the ANN to the Training set
classifier = KerasClassifier(build_fn = ann_model, ↩
    batch_size = bs, epochs = epo)
kfold = StratifiedKFold(n_splits=10, shuffle=True, ↩
    random_state = 2)
results = cross_val_score(classifier, X_test, y_test, cv↩
    =kfold)
print(results.mean())


# Making predictions and evaluating the model
classifier.fit(X_train, y_train)

# Predicting the Test set results

y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

# Confusion matrix

from Functions import  con_mat_plot
con_mat_plot(y_test,y_pred)

# Applying PCA
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

def ann_model():

    # Initialising the ANN
    classifier = Sequential()

    # Adding the input layer and the first hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu', input_dim = np.↩
       shape(X_test)[1]))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu'))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu'))

    # Adding the output layer
    classifier.add(Dense(1, kernel_initializer = '↩
       uniform', activation = 'sigmoid'))

    # Compiling the ANN
    classifier.compile(optimizer = 'adam', loss = '↩
       binary_crossentropy', metrics = ['accuracy'])

    return classifier

# Fitting the ANN to the Training set
```

```
classifier = KerasClassifier(build_fn = ann_model, ↩
   batch_size = 25, epochs = 100)
classifier.fit(X_train, y_train)

# Decision boundary

def plot_decision_boundary(clf, X, Y, X_test, cmap='↩
   Paired_r'):
    clf.predict(X_test)
    h = 0.02
    x_min, x_max = X[:,0].min() - 10*h, X[:,0].max() + ↩
       10*h
    y_min, y_max = X[:,1].min() - 10*h, X[:,1].max() + ↩
       10*h
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(10,10))
    plt.contourf(xx, yy, Z, cmap=cmap, alpha=0.25)
    plt.contour(xx, yy, Z, colors='k', linewidths=0.7)
    plt.ylabel('x2')
    plt.xlabel('x1')
    plt.grid()
    plt.scatter(X[:,0], X[:,1], c=Y, cmap=cmap, ↩
       edgecolors='k');

plot_decision_boundary(classifier ,X_train, y_train, ↩
   X_test, cmap='Paired_r')
```