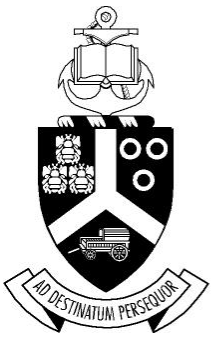


UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Electrical, Electronic and Computer Engineering

EAI 320 - Intelligent systems

# EAI 320 - Practical Assignment 7 Report

*Author:*
Mohamed Ameen Omar

*Student number:*
u16055323

May 4, 2018

# Contents

# List of Figures

# 1 Introduction

Statistical learning methods are used in the field of Artificial Intelligence in an attempt to classify and make predictions with regards to both discrete and continuous datasets.

Whereas in Practical 6, students investigated an attempt to mimic brain biology with Artificial Neural Networks, in Practical 7 students were required to investigate statistical methods for machine learning and pattern recognition.

The statistical methods that were investigated in this practical include the KNN algorithm, Linear regression as well Logistic regression techniques for classification as well as regression problem sets.

# 2 Problem Definition

This practical assignment was separated into three questions. The first dealt with an introduction into the k-Nearest-Neighbours algorithm for classification problems. The Second Question focused on regression problems, in particular the Linear regression model to solve regression problems. The third question introduced students to Logistic regression and it's ability to be used in classification problems.

The details for each questions are explained below.

## 2.1 Question 1: k-Nearest-Neighbours

The K-Nearest neighbours algorithms is an extremely simply but powerful and effective algorithm used in classification problems. In particular, it's effectiveness is seen in pattern recognition problem sets.

The KNN algorithm operates on a very simple principal - entities that are similar in one respect, will very likely be identical in another respect [1]. Using this assumption the KNN algorithm has the ability to solve a wide range of problems with consistent accuracy.

The algorithm operates as follows: given a set of known data points, the algorithm is able to estimate the result of an unknown input based off of the Euclidean distance from the unknown input to every known data point.

The following steps are followed for a classification problem:

1. Specify a value for $k$.
2. Given an unknown input.
3. For the input, calculate the Euclidean distance from that point to **every** known data point.
4. Choose the $k$ closet points to the unknown point.
5. Count the number of occurrences of the different classes that occur from the $k$ closest neighbours.
6. The class with the highest number of occurrences is the predicted output.

Students were tasked to implement the KNN algorithm for a classification problem based off of the Iris dataset given. This dataset included both test and training data in order for the implementation to be verified.

A plot of the first three attributes for every sample needed to coded in python as well as the relationship between the number of neighbours specified and the number of errors or incorrect predictions made by the KNN, needed to be evaluated.

## 2.2 Question 2: Linear Regression

Linear Regression is one of the many statistical methods used in the field of Artificial Intelligence. It attempts to model a given known data set as a linear function, thus allowing for extrapolation or predictions to be made about related unknown data.

Regression problems deal with continuous inputs that result in continuous outputs as opposed to classification problems that deal with continuous or discrete inputs that produce discrete outputs.

Least squares regression line is one of the many methods to model data as a linear function. It is more widely known as the *line of best fit*. The function would take in the input and based off the weight($m$) and the constant offset($c$) produce a predicted output.

The KNN algorithm can also be used for regression problems, by instead of classifying the input as the most common *class* within the K-nearest neighbours, it would return the average output value for the K-nearest neighbours.

Using both of the aforementioned techniques, students were required to implement these algorithms, plot the given data as well as the original data points, make predictions for the given inputs using both of their implementations and analyse the results obtained.

## 2.3 Question 3: Logistic Regression

Logistic regression is one of the most popular algorithms used in machine learning and the field of artificial intelligence, for binary(dual output) classification problems. Logistic regression makes use of the logistic function in order to make predictions for unknown data based off of known data.

Question 3 required students to evaluate the Logistic function and use it to make predictions for the given test cases based off of the given known dataset.

The co-coefficients for the Logistic function are changed through a process of training. Wherein known data is used to evaluate the accuracy of the function. During each epoch - one iteration through the entire dataset, these co-efficient or weights are updated based off of the output they predict for each sample.

The decision boundary is a surface that separates all possible input values into the binary classes to which they could belong. Students were required to after training their Logistic function to plot the original dataset as well as the decision boundary for their Logistic function.

# 3 Implementation and Methodology

## 3.1 Question 1: k-Nearest-Neighbours

The KNN algorithm operates on a very simple principal - entities that are similar in one respect, will very likely be identical in another respect [1]. The algorithm operates as follows: given a set of known data points, the algorithm is able to estimate the result of an unknown input based off of the Euclidean distance from the unknown input to every known data point.

The implementation has been made using the Python programming language in the source file called *knn.py*.

The class takes in the names of the training and testing datasets and stores the data in python dictionaries where the *key* is the label for a particular sample and the *value* to each dictionary *key* is all the samples that belong to that label.

The number of neighbours $k$ must also be specified in order for the program to run. The *matplolib* Python library is used in order to produce the required plots. The program predicts the output for each sample input as follows:

1. Specify a value for $k$ (Initially 1).
2. Take a single unknown sample.
3. For the sample, calculate the Euclidean distance from that point to **every** known data point.
4. Choose the $k$ closet points to the unknown point.
5. Count the number of occurrences of the different classes that occur from the $k$ closest neighbours.
6. The class with the highest number of occurrences is the predicted output.
7. Check if this prediction is correct.
8. Once all samples have been evaluated, check the total number of errors, if the error is not zero, increment k and repeat the aforementioned procedure until a zero error occurs and the value for $k$ is not even.

The reason for adding the condition that $k$ must not be even is that if $k$ is even, a tie may occur between two classes [1] and in that case, the result of classification is random left up-to the programmer or the programming language and data storage structures used in the implementation. The outputs can be found under the Question 1 subsection of the Results section in this report. The source code can be found in Appendix A as listing 1.

## 3.2 Question 2: Linear Regression

The KNN approach to the regression problem follows the same procedure as mentioned above differing in one respect. Instead of choosing the *class* that occurs the most from the closest neighbours, the average output value of those nearest neighbours is assigned to an unknown output.

The Linear regression logarithm on the other hand, uses the squared sum of the inputs, the square of the sum of the inputs and the outputs to estimate accurate co-efficient values for the linear model.

In order to plot the required figures the *matplotlib* Python library and associated functions was used. The estimated linear regression model can be seen in figure 7. The rest of the outputs can be found under the Question 2 subsection of the Results section in this report.The source code can be found in Appendix A as listing 2.

## 3.3 Question 3: Logistic Regression

The Logistic Regression algorithm makes use of the sigmoid function to ensure that all outputs or predictions are within the range 0 to 1. Since this is a classification problem involving probabilities, the Logistic regression approach seems feasible. The implementation has been made using the Python programming language in the source file called *logisticRegression.py*.

In order to plot the required figures the *matplotlib* Python library and associated functions was used.

This implementation stores the input data and the output data in a *numpy* array. The inputs are first normalized by dividing each sample by 100 since we are dealing with percentages. The outputs are probabilities that could be rounded to the nearest whole number (0 or 1) indicating the most likely classification. The class stores the logistic function co-efficients ($b_0$, $b_1$ and $b_2$) locally with initial values of zero.

During each training iteration (epoch) the outputs or predicted outcomes are evaluated and the logistic function co-efficients are updated using the equation: $b + alpha * (yprediction) * prediction * (1 - prediction) * x$. Where b is the co-efficient that is being updated, alpha is the specified learning rate and $x$ is the input associated with the co-efficient. $b_0$ is a bias and thus has a $x = 1$, $b_1$ and $b_2$ are associated with the first and second inputs respectively.

The process of training or iterating through the training data set until the number of epochs has been reached. The number of errors for each epoch for a given learning rate is also recorded and plotted. Examples of these plots can been seen in figures 19, 24 and 29. The rest of the outputs can be found under the Question 3 subsection of the Results section in this report. The source code can be found in Appendix A as listing 3.

# 4 Results

Below are the results obtained for Question 1, Question 2 and Question 3.

## 4.1 Question 1: k-Nearest-Neighbours

### Question 1a

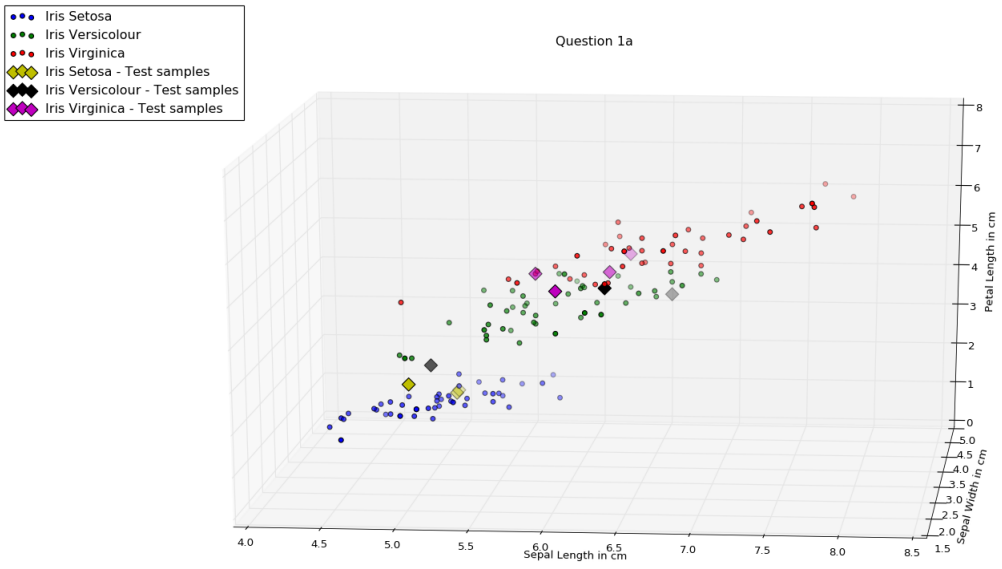


Figure 1: Python plot of the data set and samples given (1 of 2)

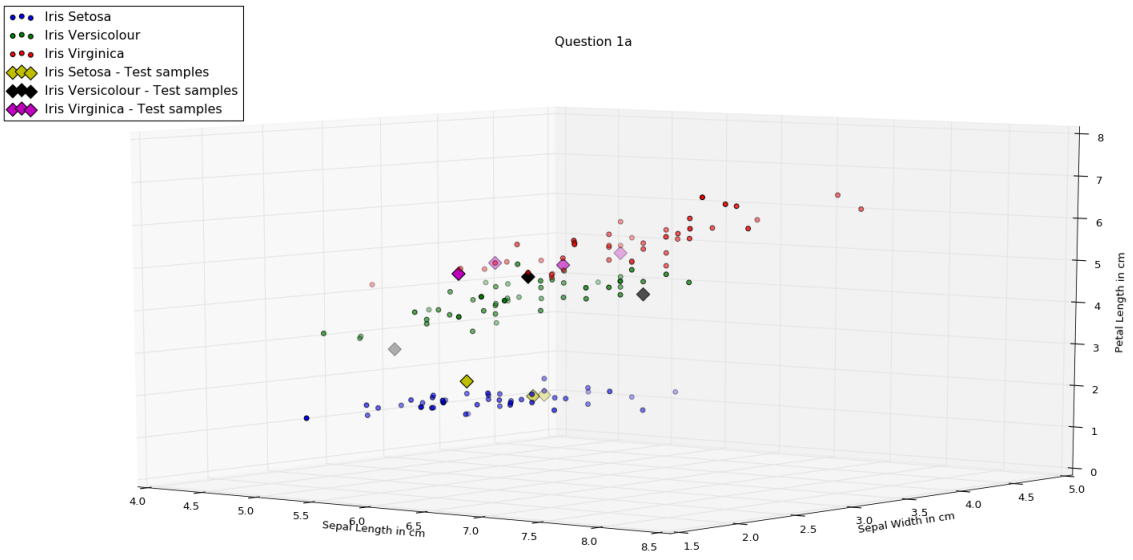


Figure 2: Python plot of the data set and samples given (2 of 2)

## Question 1b

```
Conducting KNN with number of closest Neighbours = 1
The number of errors are = 3
```

Figure 3: Output when applying KNN with K=1

## Question 1c

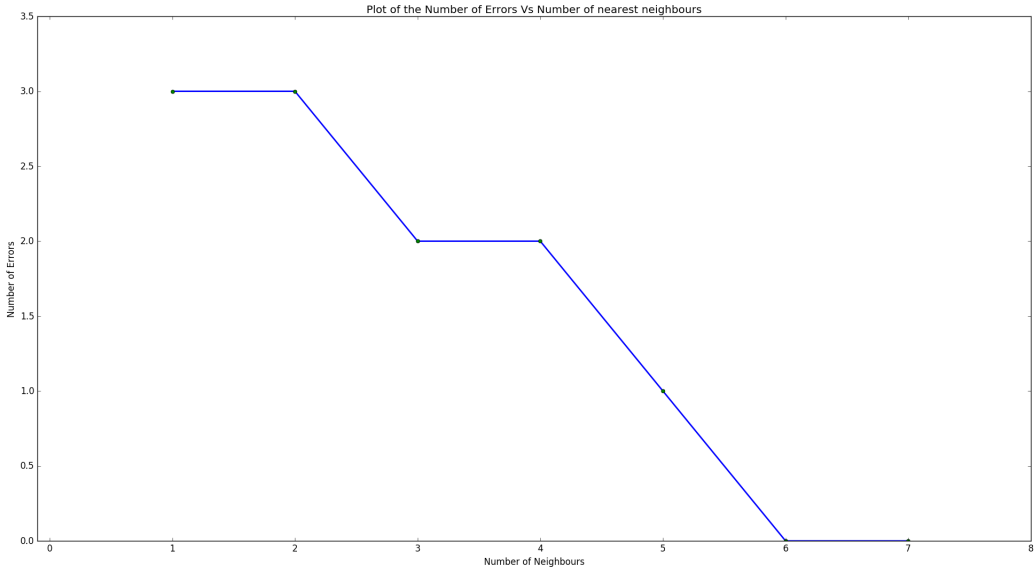


Figure 4: Plot of the Number of errors vs values of number of nearest neighbours

```
The Error for 1 nearest neighbours (k) is  3
The Error for 2 nearest neighbours (k) is  3
The Error for 3 nearest neighbours (k) is  2
The Error for 4 nearest neighbours (k) is  2
The Error for 5 nearest neighbours (k) is  1
The Error for 6 nearest neighbours (k) is  0
The Error for 7 nearest neighbours (k) is  0
Plotting Errors vs Number of neighbours
```

Figure 5: Output illustrating the errors when running KNN with values of K incrementing until a zero error is reached

## 4.2 Question 2: Linear Regression

### Linear Regression

```
Plotting the Linear Regression Line given by y = Bx+A
With A = 576.681937173 and B = -3.0068353694

The Euclidean Error for the true output value and the  predicted value is given below:
Input = 18.0, error is =  12.5589005236
Input = 20.0, error is =  73.4547702152
Input = 22.0, error is =  49.468440954
Input = 23.0, error is =  2.47527632344
Input = 23.0, error is =  47.5247236766
Input = 25.0, error is =  11.5110529378
Input = 27.0, error is =  64.502617801
Input = 28.0, error is =  17.5094531704
Input = 29.0, error is =  29.4837114602
Input = 32.0, error is =  70.4632053519
Input = 37.0, error is =  45.4290285049
Input = 41.0, error is =  6.59831297266
Input = 46.0, error is =  11.6324898197
Input = 49.0, error is =  49.3470040721
Input = 53.0, error is =  42.6803374055
Input = 55.0, error is =  8.69400814427
Input = 63.0, error is =  37.2513089005
Input = 65.0, error is =  38.7623618383
Input = 66.0, error is =  78.2308027923
Input = 67.0, error is =  34.7760325771
Input = 68.0, error is =  72.2171320535
Input = 70.0, error is =  23.7965386853
Input = 71.0, error is =  43.1966259453
Input = 72.0, error is =  9.81020942408
Input = 73.0, error is =  77.1829552065
Input = 74.0, error is =  65.8238801629
Input = 75.0, error is =  108.830715532
Input = 77.0, error is =  14.8443862711
Input = 79.0, error is =  29.1419429901
Input = 82.0, error is =  29.8785631181

The Average Error for the known data and the Linear Regression Line is: 40.235892961
```

Figure 6: Output illustrating the errors when running KNN with values of K incrementing until a zero error is reached

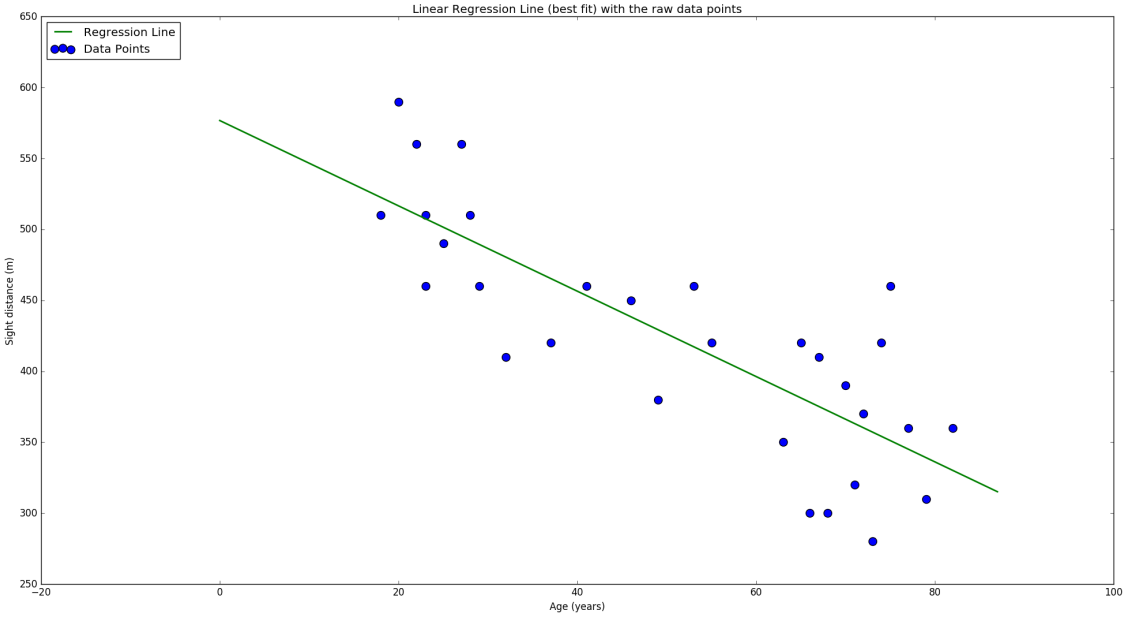


Figure 7: Plot of the calculated regression line and original data points

```
Predicted sight distance for 16 year old is (LR):
528.572571262

Predicted sight distance for 85 year old is (LR):
321.180930774
```

Figure 8: Output showing the predicted outcomes for a 16 and 85 year old using Linear regression

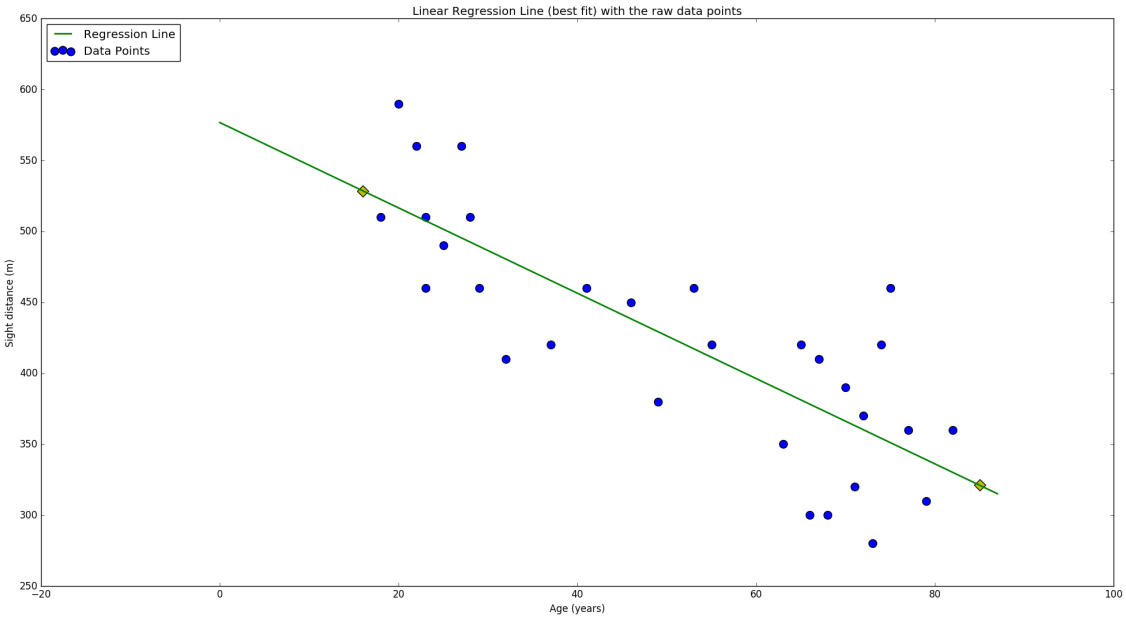


Figure 9: Plot of the calculated regression line and original data points as well as predicted outcomes for a 16 and 85 year old.

## KNN regression

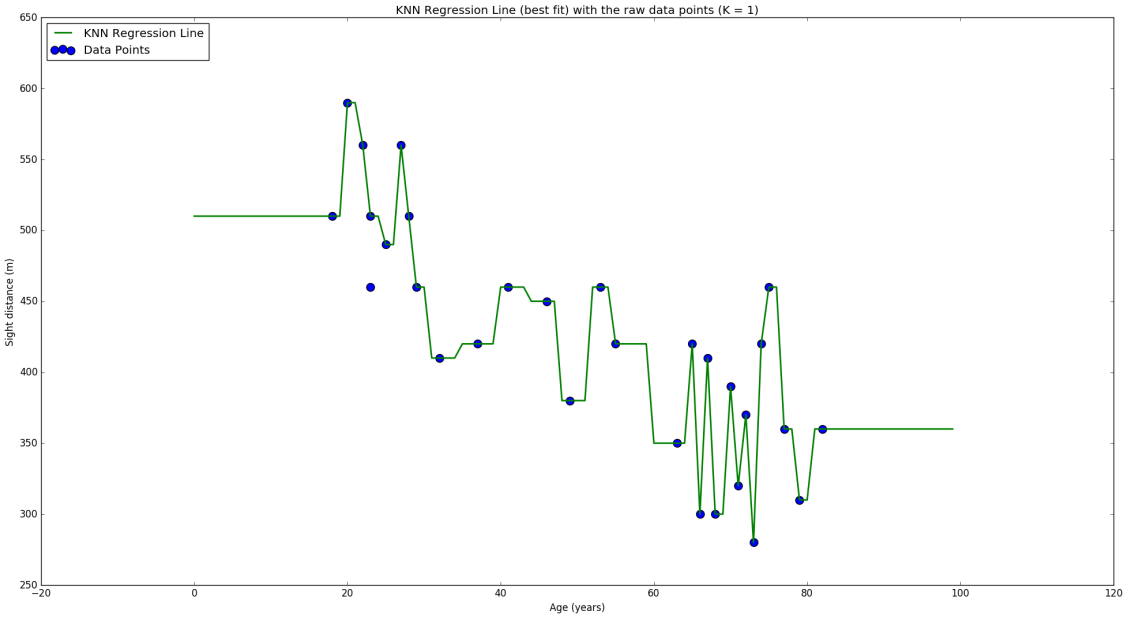


Figure 10: Plot of the output of the KNN regression with values from 0 to 100 with K = 1

```
Predicted sight distance for 16 year old is (KNNR, K = 1):
510.0

Predicted sight distance for 85 year old is (KNNR, K = 1):
360.0
```

Figure 11: Output showing the predicted outcomes for a 16 and 85 year old using KNN regression and K = 1

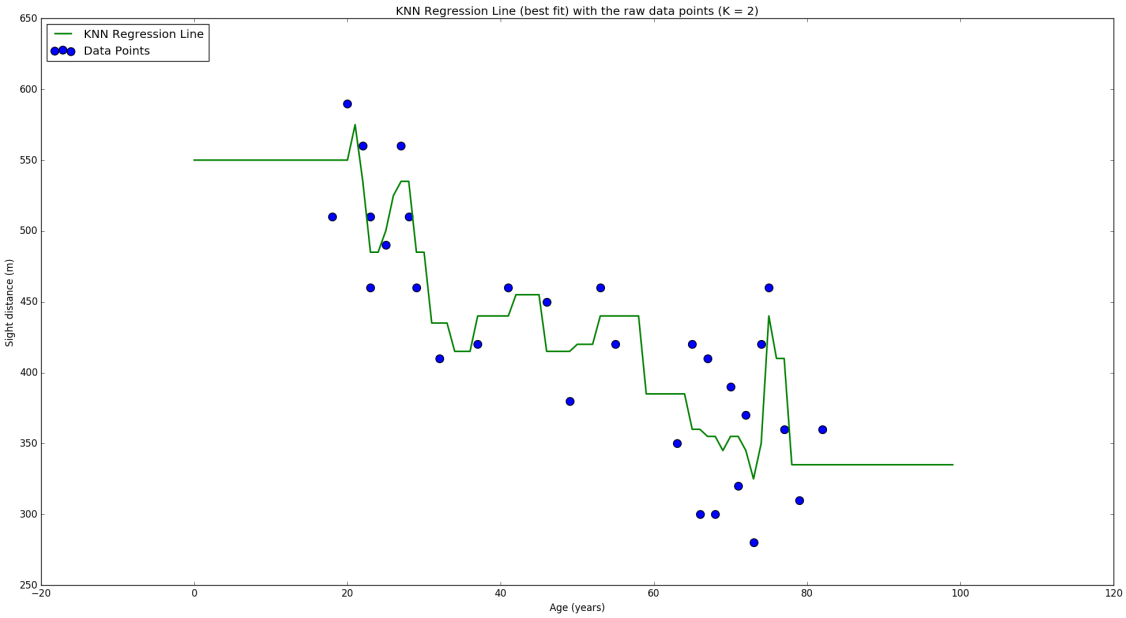


Figure 12: Plot of the output of the KNN regression with values from 0 to 100 with K = 2

```
Predicted sight distance for 16 year old is (KNNR, K = 2):
550.0

Predicted sight distance for 85 year old is (KNNR, K = 2):
335.0
```

Figure 13: Output showing the predicted outcomes for a 16 and 85 year old using KNN regression and K = 2

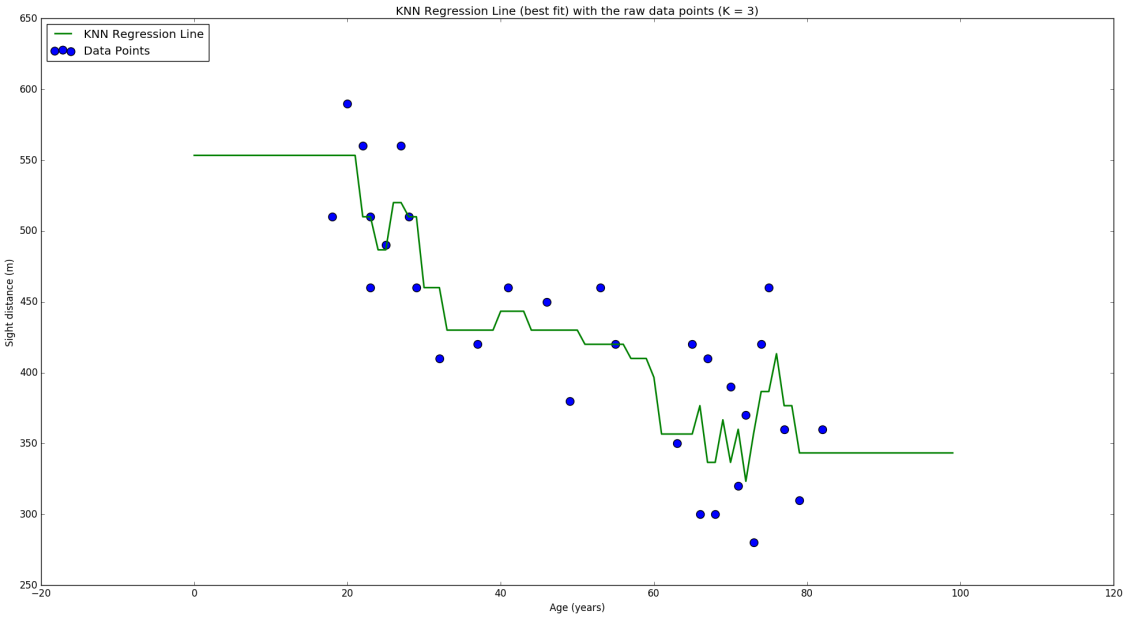


Figure 14: Plot of the output of the KNN regression with values from 0 to 100 with K = 3

```
Predicted sight distance for 16 year old is (KNNR, K = 3):
553.333333333

Predicted sight distance for 85 year old is (KNNR, K = 3):
343.333333333
```

Figure 15: Output showing the predicted outcomes for a 16 and 85 year old using KNN regression and K = 3

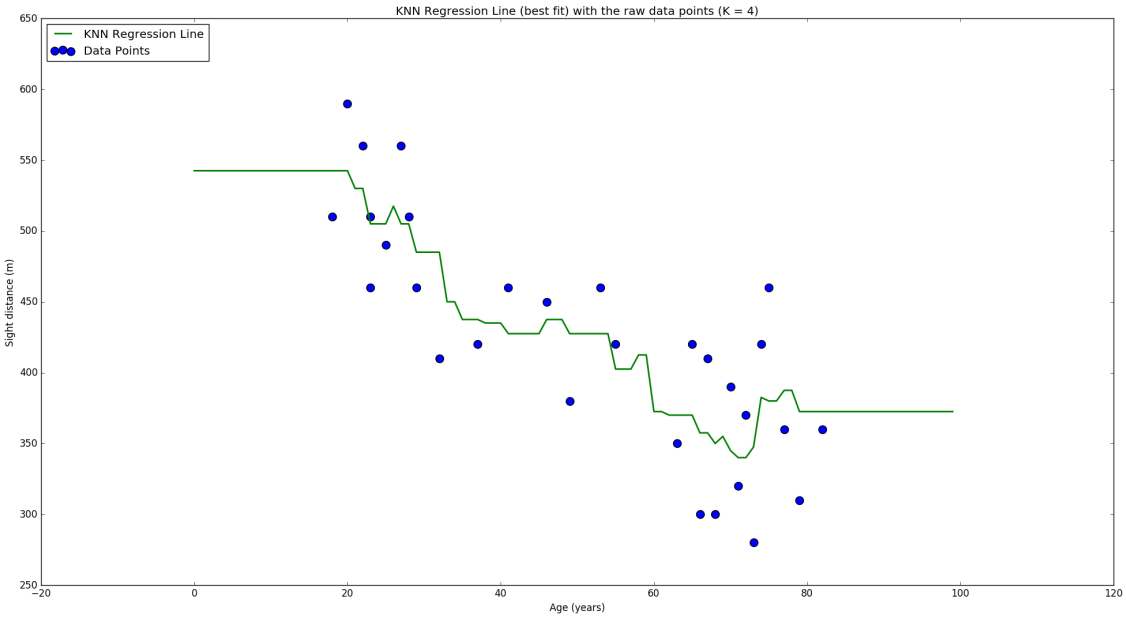


Figure 16: Plot of the output of the KNN regression with values from 0 to 100 with K = 4

```
Predicted sight distance for 16 year old is (KNNR, K = 4):
542.5

Predicted sight distance for 85 year old is (KNNR, K = 4):
372.5
```

Figure 17: Output showing the predicted outcomes for a 16 and 85 year old using KNN regression and K = 4

## 4.3 Question 3: Logistic Regression

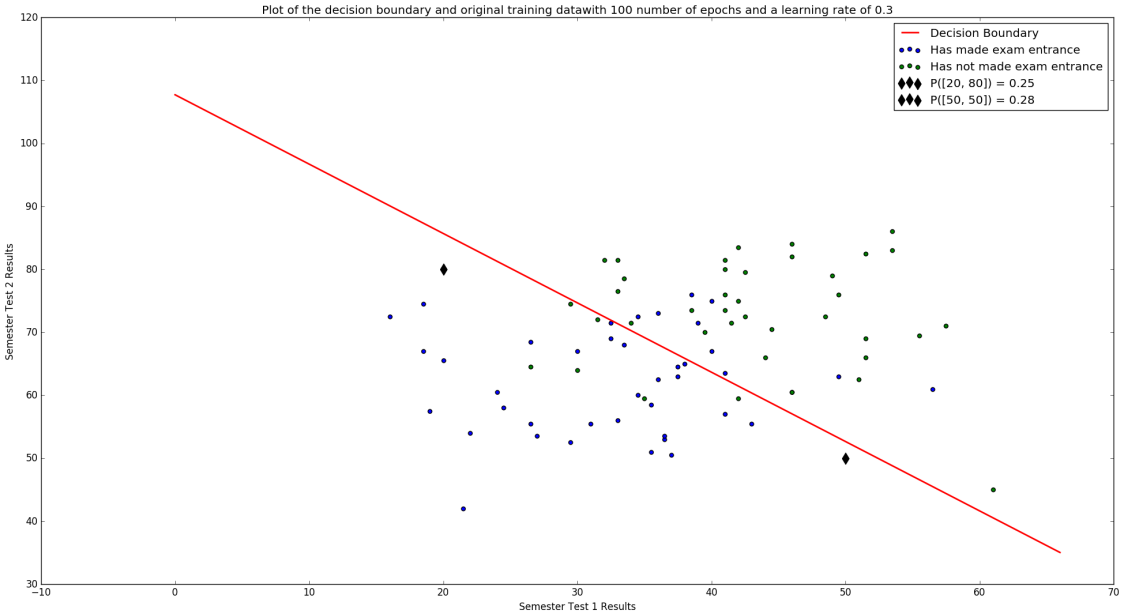


Figure 18: Plot illustrating the decision boundary and the original dataset. The predicted values are also shown. Number of epochs = 100 and a learning rate = 0.3.

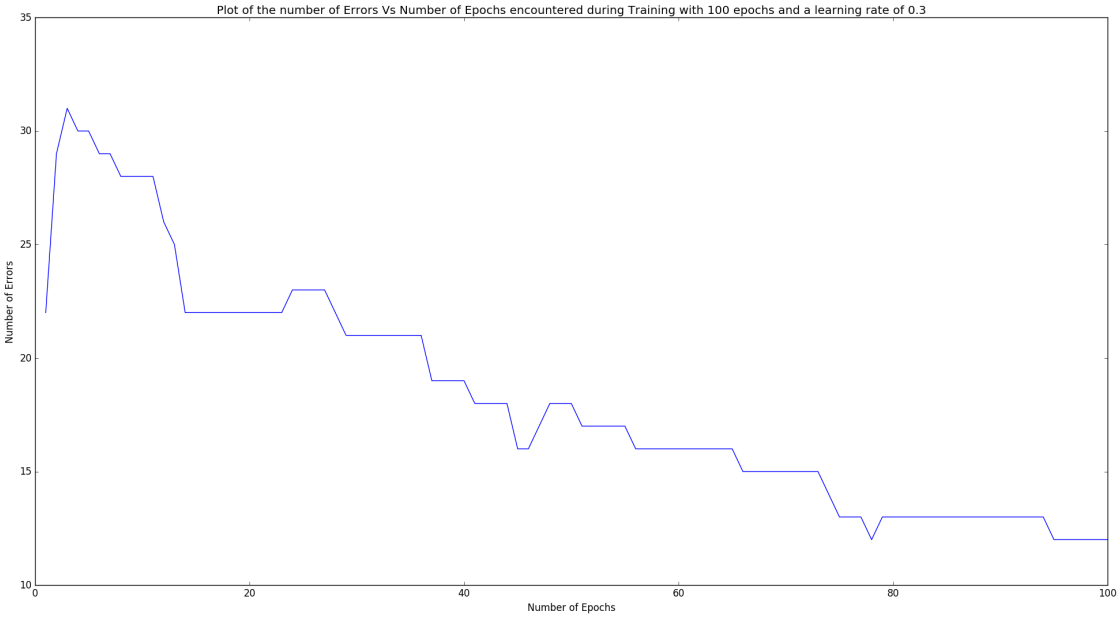


Figure 19: Plot illustrating the Error vs number of epochs with a learning rate of 0.3

```
Training with 100 epochs and a learning rate of 0.3
Training complete
Number of Errors during training were : [22, 29, 31, 30, 30, 29, 29, 28, 28, 28, 28, 26, 25, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 22, 21, 21, 21
, 17, 17, 17, 17, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 15, 15, 15, 15, 15, 15, 15, 15, 14, 13, 13, 13, 12, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 1
```

Figure 20: Output of the Logistic Regression with 100 epochs and a learning rate of 0.3

```
The probability of getting into the exam with results:
Semester Test 1 = 20%
Semester Test 2 = 80%
Probability = 0.247510444607

The probability of getting into the exam with results:
Semester Test 1 = 50%
Semester Test 2 = 50%
Probability = 0.280261554558
```

Figure 21: Output showing the predictions made with 100 epochs and a learning rate of 0.3

```
Final co-efficent values:
b0 = -6.78243548226
b1 = 6.1205538135
b2 = 5.55798798057
```

Figure 22: Output showing the final co-efficient values after 100 epochs and a learning rate of 0.3

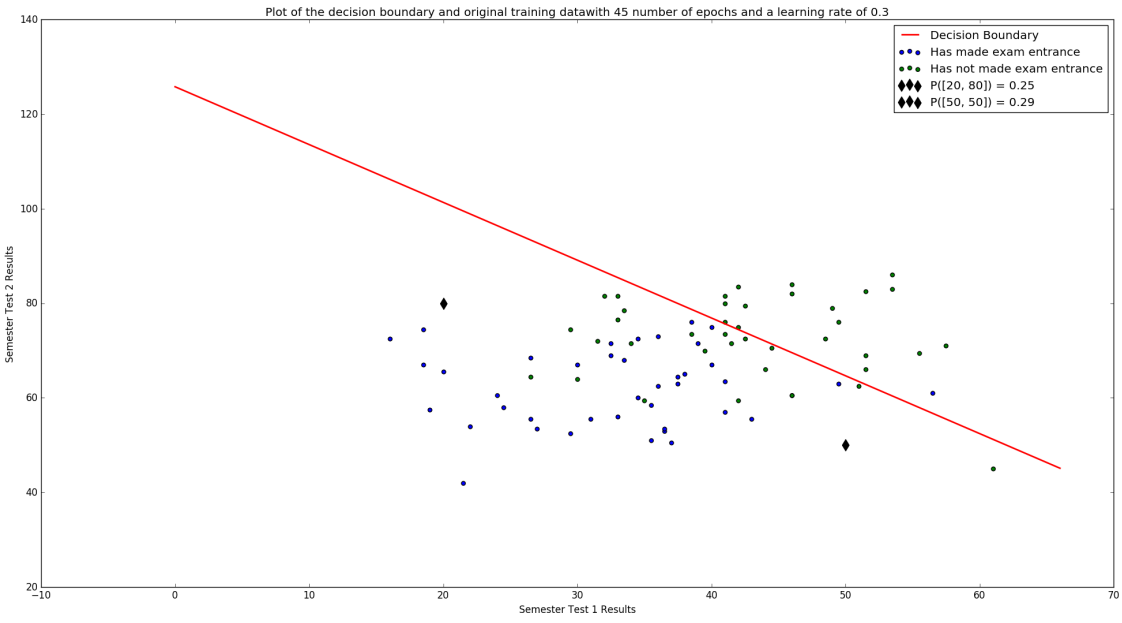


Figure 23: Plot illustrating the decision boundary and the original dataset. The predicted values are also shown. Number of epochs = 45 and a learning rate = 0.3.

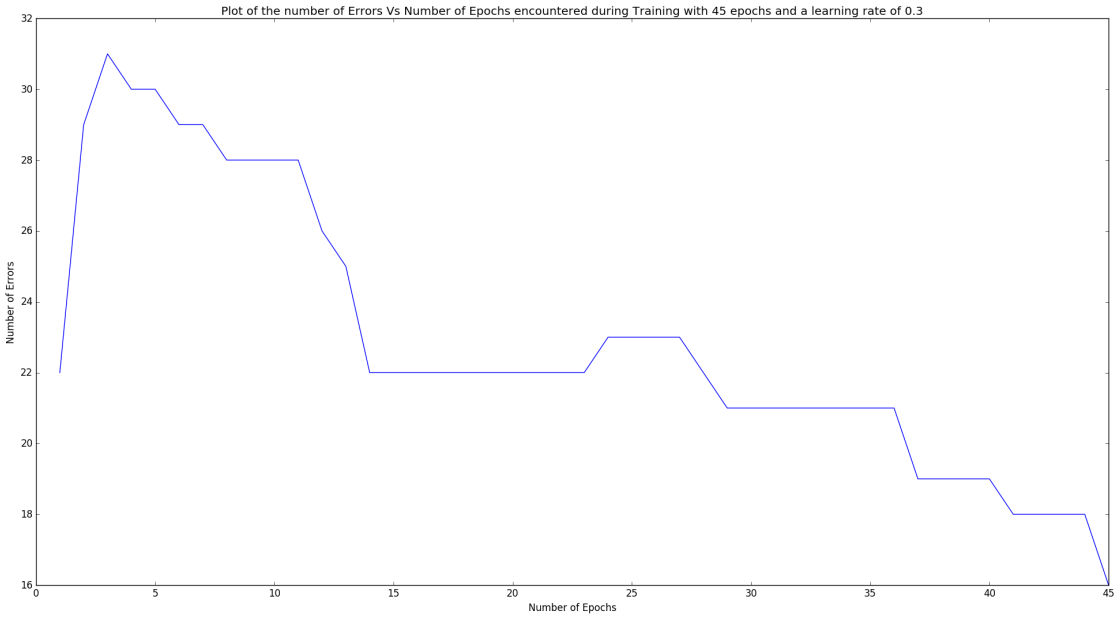


Figure 24: Plot illustrating the Error vs number of epochs with a learning rate of 0.3

```
Training with 45 epochs and a learning rate of 0.3
Training complete
Number of Errors during training were : [22, 29, 31, 30, 30, 29, 29, 28, 28, 28, 28, 26, 25, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 23, 23, 23, 23, 22, 21, 21, 21
```

Figure 25: Output of the Logistic Regression with 45 epochs and a learning rate of 0.3

```
The probability of getting into the exam with results:
Semester Test 1 = 20%
Semester Test 2 = 80%
Probability = 0.250995729703

The probability of getting into the exam with results:
Semester Test 1 = 50%
Semester Test 2 = 50%
Probability = 0.289532106231
```

Figure 26: Output showing the predictions made with 45 epochs and a learning rate of 0.3

```
Final co-efficent values:
b0 = -4.16056600515
b1 = 3.58899376324
b2 = 2.93682311691
```

Figure 27: Output showing the final co-efficient values after 45 epochs and a learning rate of 0.3

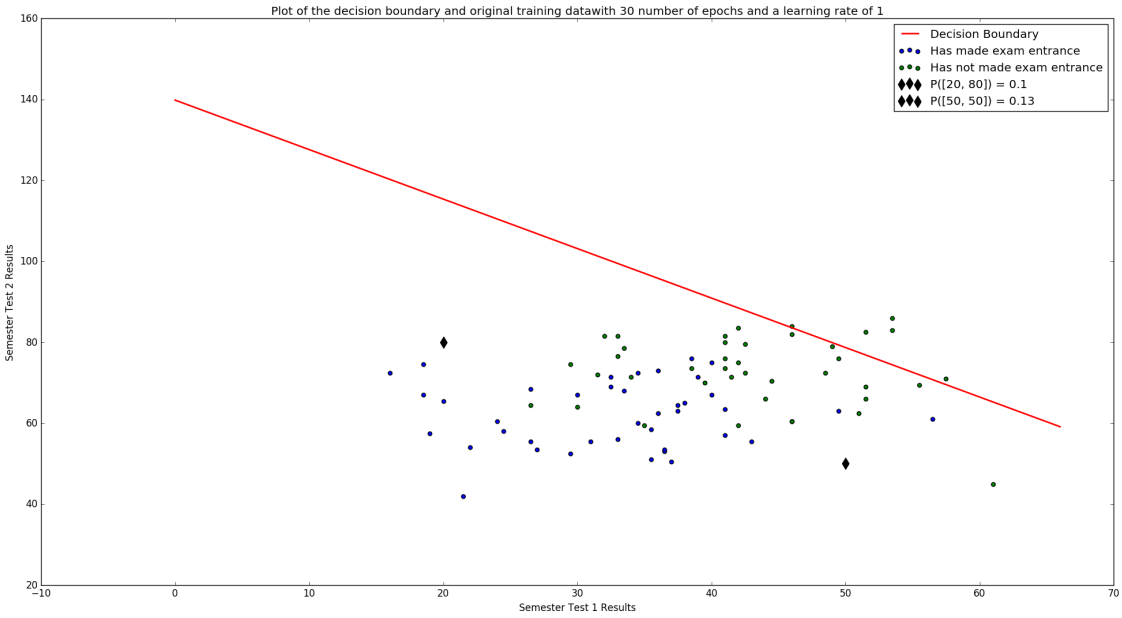


Figure 28: Plot illustrating the decision boundary and the original dataset. The predicted values are also shown. Number of epochs = 30 and a learning rate = 1.

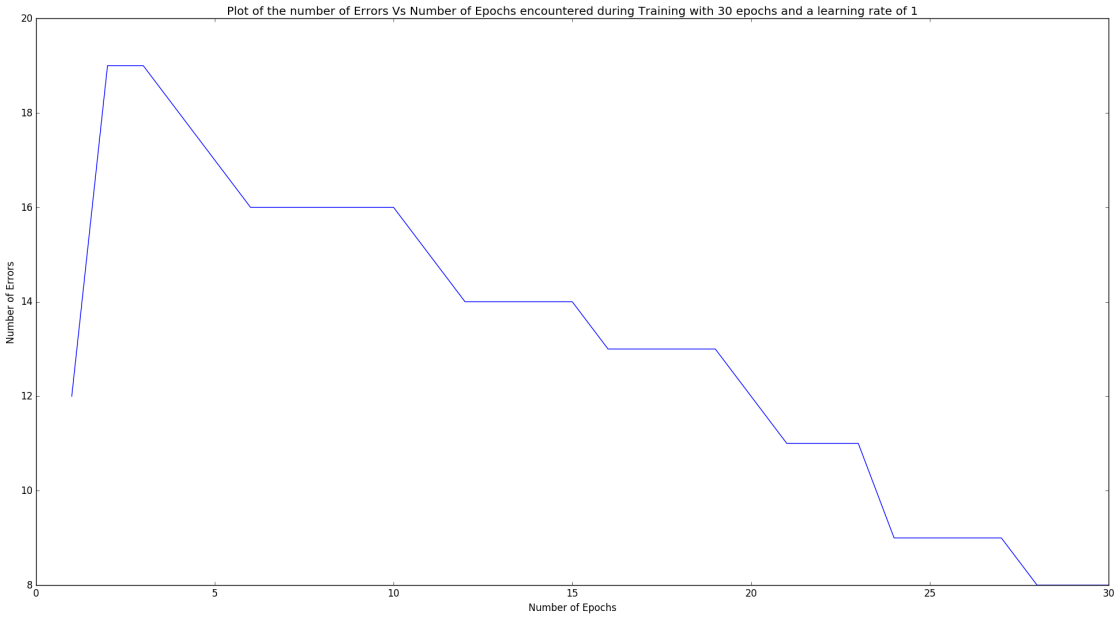


Figure 29: Plot illustrating the Error vs number of epochs with a learning rate of 1

```
Training with 30 epochs and a learning rate of 1
Training complete
Number of Errors during training were : [12, 19, 19, 18, 17, 16, 16, 16, 16, 16, 15, 14, 14, 14, 14, 13, 13, 13, 13, 12, 11, 11, 11, 9, 9, 9, 9, 8, 8, 8]
```

Figure 30: Output of the Logistic Regression with 30 epochs and a learning rate of 1

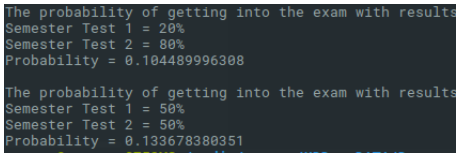


Figure 31: Output showing the predictions made with 30 epochs and a learning rate of 1

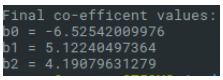


Figure 32: Output showing the final co-efficient values after 30 epochs and a learning rate of 1

# 5 Discussion

Minimization of the error experienced in both classification and regression problems are of utmost importance. Gradient descent is an excellent, popular strategy used for error minimization. A low learning rate results in slower learning and thus more time needed for the algorithm to learn a specific pattern. However, the learning rate cannot be too high since that could result in over-fitting and drastically more inaccurate results.

A brute force approach was used for the KNN algorithm. For a small dataset as provided this approach is adequate, however, intuitively it can be seen that the time complexity of the algorithm increases dramatically with larger, more realistic training and testing datasets. Therefore, more powerful searching approaches such as the k-d tree approach should be employed to search and gather the closest neighbours to a test sample and in extension classify the unknown data point.

From the results obtained we can see that the chosen algorithms are extremely powerful at both classification and regression problems, however they are not perfect. Errors do occur and will never truly reach zero due to randomness, outliers, insufficient data or incomplete data.

Different learning rates and different epoch values correspond to slight deviations in the predictions made as can be seen in the results section.

# 6 Conclusion

Statistical methods are extremely effective in pattern recognition. They can be used to both classify classes to unknown data as well as predict or extrapolate the continuous outcome for unknown data.

The KNN and Linear regression models are effective in extrapolation for inputs similar to known data, however for outliers and a range of inputs *"far away"* from the known set, these algorithms begin to fail. In order for these algorithms to be used, either the training data needs to be extremely vast or the unknown data that requires extrapolation, needs to be similar or within the range of known or training dataset.

The Logistic function gains it's power from the fact that it normalizes the data using the sigmoid function, thus ensuring that all inputs ever received are within a known range. This makes the function more effective and to a large extent more accurate in classifying known inputs.

Careful care needs to be taken in when choosing an algorithm to extrapolate or classify unknown data. The range and type of known data as well as the range and type of unknown data play a massive role in the effectiveness of the chosen pattern recognition technique.

# 7 Appendix A: Python Code

```
import copy
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import csv
import numpy
#Mohamed Ameen Omar
#16055323
#EAI 320 - Practical 7 2018
#Question 1
class KNN:
    def __init__(self, trainDataFile, trainLabelFile,testDataFile, testLabelFile):
        self.trainColors = ["b", "g", "r"]
        self.testColors = ["y", "k", "m"]
        self.labels = ["Iris Setosa", "Iris Versicolour", "Iris Virginica"]
        self.trainData = numpy.genfromtxt(trainDataFile,delimiter=',')
        self.trainLabels = numpy.genfromtxt(trainLabelFile,delimiter=',')
        self.testData = numpy.genfromtxt(testDataFile,delimiter=',')
        self.testLabels = numpy.genfromtxt(testLabelFile,delimiter=',')
        self.trainDict = {1:[], 2:[], 3:[]}
        self.testDict = {1:[], 2:[], 3:[]}
        self.storeTestData() #keys are the labels(1,2,3) and values are the attributes
        self.storeTrainData()

    def storeTrainData(self):
        for x in range(0,len(self.trainLabels)):
            self.trainDict[self.trainLabels[x]].append(self.trainData[x])

    def storeTestData(self):
        for x in range(0,len(self.testLabels)):
            self.testDict[self.testLabels[x]].append(self.testData[x])

    def plotData(self):
        fig = plt.figure()
        ax = fig.add_subplot(111, projection = '3d')
        for key in self.trainDict.keys():
            toPlot = numpy.transpose(self.trainDict[key]) #toPlot is a matrix of the co ordinates for each variable
            # ie - toPlot[0] is all the x values
            color = self.trainColors[key-1]
            name = self.labels[key-1]
            ax.scatter(toPlot[0], toPlot[1], toPlot[2], s = 110, c = color, marker = '.',
                       label = name)
        for key in self.testDict.keys():
            toPlot = numpy.transpose(self.testDict[key]) #toPlot is a matrix of the co ordinates for each variable
            # ie - toPlot[0] is all the x values
            color = self.testColors[key-1]
            name = (self.labels[key-1] + " - Test samples")
            ax.scatter(toPlot[0], toPlot[1], toPlot[2], s = 110, c = color, marker = 'D',
                       label = name)
```

```
        ax.set_xlabel('Sepal Length in cm')
        ax.set_ylabel('Sepal Width in cm')
        ax.set_zlabel('Petal Length in cm')
        plt.legend(loc = 'upper left')
        plt.title("Question 1a")
        plt.show()

    #runs the knn with the number of neighbors passed in
    #for every test sample classify it
    #retuns number of errors
    def _knn(self, k = 1):
        numErrors = 0
        for key in self.testDict.keys():
            for sample in range(0, len(self.testDict[key])):
                isError = self.knnClassify(key,self.testDict[key][sample],
k)
                if(isError is True):
                    numErrors = numErrors+1
        return numErrors

    #key is what the testSample actually is
    #testSample is an array of the attributes
    #k is the number of neighbours
    #classifies testSample, returns a boolean if it is a error or not
    def knnClassify(self,key,testSample,k):
        neighborsDist = []
        neighborsLabel = []
        ###get Euclidean Distance for every train to the testSample
        for xkey in self.trainDict.keys():
            for sample in range(0, len(self.trainDict[xkey])):
                neighborsDist.append(self.getEuclideanDistance(testSample,
self.trainDict[xkey][sample]))
                neighborsLabel.append(xkey)

        #sort arrays in ascending order
        swap = True
        while(swap is True):
            swap = False
            for x in range(0,len(neighborsDist)-1):
                if(neighborsDist[x] > neighborsDist[x+1]):
                    swap = True
                    temp = neighborsDist[x]
                    neighborsDist[x] = neighborsDist[x+1]
                    neighborsDist[x+1] = temp
                    temp = neighborsLabel[x]
                    neighborsLabel[x] = neighborsLabel[x+1]
                    neighborsLabel[x+1] = temp
        tempLabel = self.classify(neighborsLabel,k)
        #print("classified label:" ,tempLabel)
        #print("THIS KEY IS", key)
        return(tempLabel != key)

    #classifies based off of number of nearest neighbours
    #kWinner is a running count of the number of each class occurenes
    def classify(self, labels, k):
        if(k == 1):
            return labels[0]
```

```
        kWinner = [0,0,0] #0 - label 1
        for x in range(0,k):
            label = labels[x]
            kWinner[label-1] = kWinner[label-1] +1 #increment that label
        max_value = max(kWinner) #maximum value
        return (kWinner.index(max_value) +1) #return the index at which
    the max is found+1 (the classified label)

    #returns Euclidean Distance
    def getEuclideanDistance(self,testSample,trainSample):
        dist = 0
        for x in range(0,4):
            dist += numpy.square(testSample[x]-trainSample[x])
        return numpy.sqrt(dist)

    def knnVariableK(self):
        tempError = 1
        errors = []
        k = 1
        while(tempError != 0 or k%2 != 0):
            tempError = self._knn(k)
            errors.append(tempError)
            print("The Error for ", end = "")
            print(k, end = "")
            print(" nearest neighbours (k) is ", tempError)
            k=k+1
        #plot
        x = []
        for t in range(0,len(errors)):
            x.append(t+1)
        print("Plotting Errors vs Number of neighbours")
        plt.plot(x,errors, linewidth = 2)
        plt.plot(x,errors, 'bo', c = 'g', markersize = 5)
        plt.title('Plot of the Number of Errors Vs Number of nearest
    neighbours')
        plt.ylabel('Number of Errors')
        plt.xlabel('Number of Neighbours')
        plt.xlim(-0.1,8) #limits set to make the plot look more pretty
        plt.ylim(0,3.5)
        plt.show()

x = KNN("trainData.data", "trainLabels.data", "testData.data", "testLabels
    .data")
#x.plotData() #Question 1a
#print("Conducting KNN with number of closest Neighbours = 1")
#print("The number of errors are =", x._knn(1)) #Question 1b
x.knnVariableK()#Question 1c
```

Listing 1: Source code for question 1.

```
#http://www.statisticshowto.com/probability-and-statistics/regression-
    analysis/find-a-linear-regression-equation/
#Mohamed Ameen Omar
#16055323
#EAI 320 - Practical 7 2018
#Question 2
import copy
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import csv
import numpy

class regression:
    def __init__(self, filename):
        self.data = numpy.genfromtxt(filename,delimiter=',') #2d array
        self.A = 0
        self.B = 0

    def LinearRegression(self):
        co_efficients = self.LRgetCoEff()
        self.A = co_efficients[0]
        self.B = co_efficients[1]
        errors = []
        for x in range(0,len(self.data)):
            temp = self.regressionLine(self.data[x][0])
            error =  numpy.abs(temp-self.data[x][1]) #since x's are the
   same
            errors.append(error)
        print("Plotting the Linear Regression Line given by y = Bx+A")
        print("With A =", self.A, end = "")
        print(" and B =", self.B)
        print()
        self.plotRegressionLine(copy.deepcopy(self.data))
        print("The Euclidean Error for the true output value and the ",
   end  = "")
        print(" predicted value is given below:")
        for x in range(0, len(self.data)):
            print("Input =", self.data[x][0], end = "")
            print(", error is = ", errors[x])
        print()
        print("The Average Error for the known data and the Linear
   Regression Line is:", (sum(errors)/len(errors)))

    def plotRegressionLine(self,data):
        x = numpy.arange(0,88)
        plt.plot(self.regressionLine(x), c = 'g', linewidth = 2, label = '
   Regression Line')
        toPlot = numpy.transpose(data)
        plt.scatter(toPlot[0],toPlot[1], s = 100, c = 'b', label = 'Data
   Points')
        plt.legend(loc = 'upper left')
        plt.xlabel('Age (years)')
        plt.ylabel('Sight distance (m)')
        plt.title('Linear Regression Line (best fit) with the raw data
   points', loc ='center')
        plt.scatter(16,self.regressionLine(16), s = 100, c = 'y', marker =
    'D', label = 'Predicted sight distance for 16 year old') #16
        plt.scatter(85,self.regressionLine(85), s = 100, c = 'y', marker =
    'D', label = 'Predicted sight distance for 85 year old') #85
        plt.show()

    #returns an array of coefficents for least squares regression line
    def LRgetCoEff(self):
        sumY = 0
```

```
        sumX = 0
        sumXY = 0
        sumXsqrd = 0
        for x in range(0, len(self.data)):
            sumX = sumX + self.data[x][0]
            sumY = sumY+self.data[x][1]
            sumXY = sumXY + (self.data[x][0] * self.data[x][1])
            sumXsqrd = sumXsqrd + numpy.square(self.data[x][0])
        a = 0
        b = 0
        #compute a
        a = sumY * sumXsqrd
        a = a-(sumX*sumXY)
        temp = len(self.data) * sumXsqrd
        temp = temp - numpy.square(sumX)
        a = a/temp
        b = len(self.data) * sumXY
        b = b-(sumX*sumY)
        temp = len(self.data) * sumXsqrd
        temp = temp - numpy.square(sumX)
        b = b/temp
        return([a,b])

    def regressionLine(self,x):
        return ((self.B*x)+self.A)

##########################KNN#####################################################
    #returns the error if the input data is used
    def regressionKNNError(self,k = 1):
        temp = numpy.transpose(copy.deepcopy(self.data))
        inputs = temp[0]
        outputs = temp[1]
        numErrors = 0
        for x in range(0,len(inputs)):
            error = numpy.abs(self.regressionKNNoutput(k,inputs[x]) -
    outputs[x])
            if(error != 0):
                numErrors = numErrors+1
        return numErrors

    #returns the output based off of the KNN regression algo
    def regressionKNNoutput(self,k, input):
        temp = numpy.transpose(copy.deepcopy(self.data))
        inputs = temp[0]
        outputs = temp[1]
        distance = []
        estimatedOutput = []
        for x in range(0,len(inputs)):
            distance.append(numpy.abs(input-inputs[x]))
            estimatedOutput.append(outputs[x])
        #sort arrays in ascending order
        swap = True
        while(swap is True):
            swap = False
            for x in range(0,len(distance)-1):
                if(distance[x] > distance[x+1]):
                    swap = True
```

```
                    temp = distance[x]
                    distance[x] = distance[x+1]
                    distance[x+1] = temp
                    temp = estimatedOutput[x]
                    estimatedOutput[x] = estimatedOutput[x+1]
                    estimatedOutput[x+1] = temp
        knnOut = 0
        for x in range(0, k):
            knnOut = knnOut + estimatedOutput[x]
        knnOut = knnOut/k
        return knnOut #return the average

    def plotKNNregression(self,k):
        x = []
        for y in range(0,100):
            x.append(self.regressionKNNoutput(k,y))
        toPlot = numpy.transpose(self.data)
        #for y in toPlot[0]:
        #    x.append(self.regressionKNNoutput(k,y))
        #plt.plot(toPlot[0], x,c = 'g', linewidth = 2, label = 'KNN
    Regression Line')
        plt.plot( x,c = 'g', linewidth = 2, label = 'KNN Regression Line')
        plt.scatter(toPlot[0],toPlot[1], s = 100, c = 'b', label = 'Data
    Points')
        plt.legend(loc = 'upper left')
        plt.xlabel('Age (years)')
        plt.ylabel('Sight distance (m)')
        title = 'KNN Regression Line (best fit) with the raw data points (
    K = ' + str(k) + ')'
        plt.title(title, loc ='center')
        plt.show()

temp = regression("signdist.data")
temp.LinearRegression() #Question 2a/b
print()
print("Predicted sight distance for 16 year old is (LR):")
print(temp.regressionLine(16)) #2c
print()
print("Predicted sight distance for 85 year old is (LR):") #2d
print(temp.regressionLine(85))


#KNN regression below

k = 3
print()
print("Plot for KNNN regression with inputs from 0 to 100 and K =", k, end
    = " ")
print(":")
temp.plotKNNregression(k)
print()
print("Predicted sight distance for 16 year old is (KNNR, K =", k, end = "
    " )
print("):")
print(temp.regressionKNNoutput(k,16))
print()
print("Predicted sight distance for 85 year old is (KNNR, K =", k, end = "
```

```
    " )
print("):")
print(temp.regressionKNNoutput(k,85))
```

Listing 2: Source code for question 2.

```
import copy
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import csv
import numpy
#https://machinelearningmastery.com/logistic-regression-tutorial-for-
    machine-learning/
#Mohamed Ameen Omar
#16055323
#EAI 320 - Practical 7 2018
#Question 3
class logisticRegression:
    def __init__(self, inputsFileName, outputsFileName, epochs,
    learningRate):
        self.inputs = self.normalize(numpy.genfromtxt(inputsFileName,
    delimiter=','))
        self.outputs = numpy.genfromtxt(outputsFileName,delimiter=',')
        self.learningRate = learningRate
        self.epochs = epochs
        self.b0 = 0
        self.b1 = 0
        self.b2 = 0
    #to get probablility based off of the output
    def logisticFunction(self,val):
        return ( 1 / ( 1+numpy.exp(-1*val) ) )

    #to get the output
    def logisticRegressionFunction(self,input1, input2):
        return(self.b0 + (self.b1*input1) + (self.b2*input2))

    #returns the probability
    def getProbability(self,input1,input2):
        return(self.logisticFunction(self.logisticRegressionFunction(
    input1,input2)))

    def normalize(self,data):
        return (data/100)

    def train(self):
        print("Training with", self.epochs, end = "")
        print(" epochs and a learning rate of", self.learningRate)
        epochError = []
        for epoch in range(0, self.epochs):
            # for every sample
            numErrors = 0
            for x in range(0, len(self.inputs)):
                input1 = self.inputs[x][0]
                input2 = self.inputs[x][1]
                trueOutput = self.outputs[x]
                prediction = self.getProbability(input1,input2)
```

```
                if(trueOutput != numpy.round(prediction)):
                    numErrors = numErrors+1
                self.b0 = self.updateB0(trueOutput,prediction)
                self.b1 = self.updateB1(trueOutput,prediction,input1)
                self.b2 = self.updateB2(trueOutput,prediction,input2)
            epochError.append(numErrors)
            #get prediciton
            #get error
            #update
        print("Training complete")
        print("Number of Errors during training were :", epochError)
        self.plotErrors(epochError)
        print()

    #b = b + alpha * (y     prediction) * prediction * (1     prediction)
    * x
    def updateB0(self,trueOutput, prediction):
        return(self.b0 + (self.learningRate*(trueOutput-prediction)*
    prediction*(1-prediction)*1))
    #to update b1
    def updateB1(self,trueOutput, prediction,input1):
        return(self.b1 + (self.learningRate*(trueOutput-prediction)*
    prediction*(1-prediction)*input1))
    #to update b2
    def updateB2(self,trueOutput, prediction,input2):
        return(self.b2 + (self.learningRate*(trueOutput-prediction)*
    prediction*(1-prediction)*input2))

    def plotErrors(self, errors):
        print("Plot of the number of Errors Vs Number of Epochs
    encountered during Training")
        xs = []
        for x in range(0, len(errors)):
            xs.append(x+1)
        plt.plot(xs,errors)
        title = "Plot of the number of Errors Vs Number of Epochs
    encountered during Training with "
        title = title + str(self.epochs) + " epochs and a learning rate of
     " + str(self.learningRate)
        plt.xlabel("Number of Epochs")
        plt.ylabel("Number of Errors")
        plt.title(title)
        plt.show()
    #for decision boundary
    def decBoundary(self,val):
        return (self.b0 + (self.b1*val)) / (-self.b2)
    #plot decision boundary and data points
    def plot(self):
        fig = plt.figure()
        tempRange = numpy.arange(13,80)
        toPlot = numpy.transpose(self.inputs)*100
        plt.scatter(toPlot[0][40:], toPlot[1][40:], s=25, c='b', marker='o
    ', label='Has made exam entrance')
        plt.scatter(toPlot[0][:40], toPlot[1][:40], s=25, c='g', marker='o
    ', label='Has not made exam entrance')
        plt.plot(self.decBoundary(tempRange/100)*100, c = 'r', linewidth =
     2, label='Decision Boundary')
```

```
        plt.scatter(20, 80, s=100, marker = 'd', c='k', label='P([20, 80])
    = ' + str(numpy.round(self.getProbability(20/100, 80/100), 2))) #
    Probability of 20, 80 getting exam entrance #
        plt.scatter(50, 50, s=100, marker = 'd', c='k', label='P([50, 50])
    = ' + str(numpy.round(self.getProbability(50/100, 50/100), 2))) #
    Probability of 50, 50 getting exam entrance #
        title = "Plot of the decision boundary and original training data"
        title = title + "with " +str(self.epochs) + " number of epochs and
     a learning rate of " +str(self.learningRate)
        plt.title(title)
        plt.xlabel('Semester Test 1 Results')
        plt.ylabel('Semester Test 2 Results')
        plt.legend(loc='best')
        plt.show()


epochs = 100 #change for epochs
learningRate = 0.3 #change for learning rate
temp = logisticRegression("examX.data", "examY.data",epochs,learningRate)
temp.train()
print("The probability of getting into the exam with results:")
print("Semester Test 1 = 20%")
print("Semester Test 2 = 80%")
print("Probability =", temp.getProbability(0.20,0.80) ) #Question 3c
print()
print("The probability of getting into the exam with results:")
print("Semester Test 1 = 50%")
print("Semester Test 2 = 50%")
print("Probability =",temp.getProbability(0.50,0.50) ) #Question 3d
print()
print()
print("Final co-efficent values:")
print("b0 =", temp.b0)
print("b1 =", temp.b1)
print("b2 =", temp.b2)
temp.plot()
```

Listing 3: Source code for question 3.

# 8 Bibliography

[1] S. Russel and P. Norvig, *Artificial intelligence : A modern approach*, Third. Pearson, 2010.