**Title of the project:**

**"ANALYSIS OF VERTEBRAL DATASET USING NAÏVE BAYES CLASSIFICATION"**

**Abstract:**

• The project is aimed at the analysis of Vertebral Data Set to identify whether the individual’s orthopedic condition is under normal or abnormal state based on the physical vertebral details. The Vertebral Spinal back pain is extremely common, the symptoms and severity of Vertebral Spinal back pain vary greatly according to individual. A simple spinal back muscle strain might be excruciating enough to necessitate an emergency room visit, while a degenerating disc might cause only mild, intermittent discomfort. Vertebral Spinal pain can be caused by a variety of problems with any parts of the complex, interconnected network of spinal muscles, nerves, bones, discs or tendons in the lumbar spine.

• Typical sources of low back pain include: The large nerve roots in the low back that go to the legs may be irritated, The smaller nerves that supply the low back may be irritated, The large paired lower back muscles (erector spinae) may be strained, The bones, ligaments or joints may be damaged, An intervertebral disc may be degenerating.

**Introduction:**

The analysis helps to identify whether the individual’s orthopaedic condition is under normal or abnormal state based on the physical vertebral details. The Vertebral Spinal back pain is extremely common, the symptoms and severity of Vertebral Spinal back pain vary greatly according to individual. A simple spinal back muscle strain might be excruciating enough to necessitate an emergency room visit, while a degenerating disc might cause only mild, intermittent discomfort. Vertebral Spinal pain can be caused by a variety of problems with any parts of the complex, interconnected network of spinal muscles, nerves, bones, discs or tendons in the lumbar spine.

**Existing System:**

The existing system use the basic naive Bayes algorithm to perform the classification task. These systems are extensively affected by the improper choice of attribute selection for the classification task. The existing system can produce an accuracy of 70% by applying the basic naive Bayes algorithm.

**Proposed System:**

From the literature, we see that there has been a lot of effort to improve existing algorithms in different ways. In my project, I have implemented Naïve Bayes, K-means, Decision tree, random forest algorithm and SVM. Also in this project, efforts are made to improve efficiency of particular algorithms with a specific dataset. Datasets analyzed in this project include vertebral column disorders. I have tried optimizing multiple algorithms to achieve high accuracy in the datasets used. These algorithms are used to overcome the limitations of basic algorithms thereby producing a better machine learning model, which improves accuracy of classification.

**Dataset used:** Vertebral dataset

The dataset (From UCLA Repository) which we are analyzing is having 6 attributes related to the spinal information and also has more than 310 instances (rows of data). Each patient details is represented in the data set by six bio mechanical attributes derived from the shape and orientation of the pelvis and lumbar spine:

1. pelvic incidence

2. pelvic tilt

3. lumbar lordosis angle

4. sacralslope

5. pelvic radius

6. degree of spondylolisthesis.

**Literature review:**

• Zengchang Qin (2012), have proposed a hybrid method by combining the naïve Bayes and probability estimation trees to give good probability estimation[1], good performance without losing the transparency, and proposed two methodologies one uses the Naive Bayes and provides the probability estimation tree and the alternative method is use a group of small- sized PETs as Naïve Bayes estimators. The second methodology is similar to the probability estimation tree and naïve Bayes and transparency is greatly provided in naïve Bayes as it is pure black box model.

• Yuguang Huang, Lei Li(2011), introduces the Naïve Bayes classification algorithm based on Poisson distribution model[4], and the experimental results show that this method keeps high classification accuracy even in small sample set. The experimental results show that the method not only in the largescale data set has satisfactory classification results[4], but also shows a good classification performance in the small sample dataset. The obvious advantages of Bayesian method are efficient, fast, and SVM is regarded as a good classification method, but consuming too much time and space. Therefore, the future work will focus on the comparative analysis of the differences between them in effectiveness and efficiency.

• P. Radha Krishna , Supriya Kumar De (2005), the study shows that this approach is highly suitable for realworld applications, especially when databases contain uncertain information. Their analyzation is based on fuzzy proximity relations for the domain of each attribute. Though nothing can replace precise and complete probabilistic information, a useful classification system for data mining can be built even with imperfect data by introducing domain dependent constraints. Though there were many limitations, the naïve bayes technique has several advantages. It takes only one pass over the data and can work with partial knowledge about the data. Applying the techniques of mining and refining with statistical and fuzzy concepts allows goo decision support with known levels of confidence.

• Jia Wu, Zhihua Cai, and Xingquan Zhu (2015), we propose an Artificial Immune System (AIS) based self adaptive probability estimation method, namely AISENB, which uses AIS to automatically and self adaptively select the optimal terms and values for probability estimation. The unique immune system based evolutionary computation process; including clone, mutation, and crossover, ensure that AISENB can adjust itself to the data without explicit specification of functional or distributional forms forthunder lying model. From the two typical probability estimation methods[3], Laplace-estimate (LENB) and Mestimate (MENB),for Naive Bayes classification. Our analysis shows that MENB has better performance than LENB.

P. Radha Krishna , Supriya Kumar De (2005), the study shows that this approach is highly suitable for real-world applications[5], especially when databases contain uncertain information. Their analyzation is based on fuzzy proximity relations for the domain of each attribute. Though nothing can replace precise and complete probabilistic information, a useful classification system for data mining can be built even with imperfect data by introducing domain dependent constraints. Though there were many limitations, the naïve bayes technique has several advantages[5]. It takes only one pass over the data and can work with partial knowledge about the data. Applying the techniques of mining and refining with statistical and fuzzy concepts allows goo decision support with known levels of confidence.

**Algorithms used:**

**Classification with “K-Means”**

K-Means clustering intends to partition n objects into k clusters in which each object belongs to the cluster with the nearest mean. This method produces exactly k different clusters of greatest possible distinction. The objective of K-Means clustering is to minimize total intra-cluster variance, or, the squared error function.

Clusters the data into k groups where k is predefined. Select k points at random as cluster centers. Assign objects to their closest cluster center according to the Euclidean distance function. Calculate the centroid or mean of all objects in each cluster. Repeat steps 2, 3 and 4 until the same points are assigned to each cluster in consecutive rounds

**Classification with “Decision Tree”**

Decision tree technique provides a powerful technique for classification and prediction in Vertebral Dataset problem. 10-fold cross validation is used to prepare training and test data.

Compared to other algorithms decision trees requires less effort for data preparation during pre-processing

Simple to understand

Can handle both numerical and categorial data But Overfitting occurs when the algorithms capture noise in the data and High variance occurs where the model can get unstable due to small variations in data

**Classification with “Naïve Bayes”**

Statistical method for classification.

Supervised learning method.

Assumes an underlying probabilistic model, the Bayes theorem and can solve problems involving both categorical and continuous valued attributes.

It uses Bayesian Theorem

|  |
| --- |
| **P(H|X) = p(X|H) P(H) / p(X)** |

**Classification with “Support Vector Machine”**

A Support Vector Machine method for the classification of both linear and nonlinear data. In a brief, an SVM is an algorithm that works as follows.

• SVM transform the original training data into a higher dimension using nonlinear mapping.

• Within this new dimension, it searches for the linear optimum separating hyper-plane to differentiate the tuples among the sets.

• With an appropriate nonlinear mapping to an adequate high dimension, data from two sets can always be separated by a hyper-plane. The SVM finds this hyper-plane with the help of support vectors and margins. The target is to identify the best one which will have the minimum classification error on preceding unseen tuples.

**Classification with “Random Forest Algorithm”**

Random forests or random decision forests is method that operates by constructing multiple decision tress during training phase .The Decision of the majority of the tree is chosen by random forest as the final decision

● No Over fitting- Use of multiple trees reduce the risk of over fitting

● High Accuracy – runs efficiently on large database for large data it produces highly accurate predictions

● Estimates missing data - Random forest can maintain its accuracy when large proportion of data is missing. But complexity is high and the training period is longer.

Code:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import warnings

from plotly.offline import init\_notebook\_mode, iplot

init\_notebook\_mode(connected = True)

import plotly.graph\_objs as go

warnings.filterwarnings("ignore", category=FutureWarning)

import os

print(os.listdir("C:\Program Files\R"))

#import data

data = pd.read\_csv("C:\Program Files\R\indu dmt.csv")

print(data.info())

#split data to x, y

x = data.drop(["class"], axis = 1)

y = data["class"].values

#normalized data

x = (x - np.min(x)) / (np.max(x) - np.min(x)).values

x.head()

#change y values abnormal/normal to 0/1

y = np.array( [1 if each == "Abnormal" else 0 for each in y] )

# SPLIT DATA TO train and test

from sklearn.model\_selection import train\_test\_split

#x = checkup, y = classes

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y, test\_size = 0.2, random\_state=42)

x\_train = x\_train.T

x\_test = x\_test.T

y\_train = y\_train.T

y\_test = y\_test.T

#%% PARAMETER INITIALIZE

def initialize\_weights\_and\_bias(dimension):

w = np.full((dimension,1), 0.01)

b = 0.0

return w,b

def sigmoid(z):

y\_head = 1/(1 + np.exp(-z))

return y\_head

def forward\_backward\_propagation(w, b, x\_train, y\_train):

#foward propagation

z = np.dot(w.T, x\_train) + b

y\_head = sigmoid(z)

loss = - y\_train \* np.log(y\_head) - (1-y\_train) \* np.log(1-y\_head)

cost = (np.sum(loss)) / x\_train.shape[1] #backward propagation

derivative\_weight = (np.dot(x\_train, ((y\_head-y\_train).T))) / x\_train.shape[1]

derivative\_bias = np.sum(y\_head - y\_train) / x\_train.shape[1]

gradients = {"derivative\_weight" : derivative\_weight, "derivative\_bias" : derivative\_bias}

return cost, gradients

def update(w, b, x\_train, y\_train, learning\_rate, number\_of\_iteration):

cost\_list = []

cost\_list2 = []

index = []

#updating parameters

for i in range(number\_of\_iteration):

#make forward and backward propagation and find cost and gradients

cost, gradients = forward\_backward\_propagation(w, b, x\_train, y\_train)

cost\_list.append(cost)

#Update et

w = w - learning\_rate \* gradients["derivative\_weight"]

b = b - learning\_rate \* gradients["derivative\_bias"]

if i % 100 == 0:

cost\_list2.append(cost)

index.append(i)

print("Cost after iteration %i : %f" %(i, cost))

parameters = {"weight" : w, "bias" : b}

plt.plot(index, cost\_list2)

plt.xticks(index, rotation='vertical')

plt.xlabel("Number of iteration")

plt.ylabel("Cost")

plt.show()

return parameters, gradients, cost\_list

#%% PREDICT, TEST

def predict(w, b, x\_test):

z = sigmoid(np.dot(w.T, x\_test) + b)

Y\_prediction = np.zeros((1, x\_test.shape[1]))

for i in range(z.shape[1]):

if z[0,i] <= 0.5:

Y\_prediction[0, i] = 0

else:

Y\_prediction[0, i] = 1

return Y\_prediction

def logistic\_regression(x\_train, y\_train, x\_test, y\_test, learning\_rate, num\_iterations):

#initialize

dimension = x\_train.shape[0] #that is 4096

w,b = initialize\_weights\_and\_bias(dimension)

parameters, gradients, cost\_list = update(w, b, x\_train, y\_train, learning\_rate, num\_iterations)

#y\_prediction\_train = predict(parameters["weight"], parameters["bias"], x\_train)

y\_prediction\_test = predict(parameters["weight"], parameters["bias"], x\_test)

print("My Test Accuracy : {} %" .format(100 - np.mean(np.abs(y\_prediction\_test - y\_test)) \* 100))

return y\_prediction\_test

my\_predict =logistic\_regression(x\_train, y\_train, x\_test, y\_test, learning\_rate = 5, num\_iterations = 1000).reshape(-1,1)

#CONFUSION MATRIX,

from sklearn.metrics import confusion\_matrix

my\_cm = confusion\_matrix(y\_test, my\_predict)

import seaborn as sns

import matplotlib.pyplot as plt

#MY LR CONFUSION MATRIX PLOT

plt.figure(figsize=(5,5))

sns.heatmap(my\_cm, annot = True, linewidth = 0.5, linecolor="red", fmt = ".0f")

plt.xlabel("Predict Values")

plt.ylabel("True Values")

plt.title("MY CONFUSION MATRIX PLOT")

plt.show()

**#%% CLASSIFICATION WITH KNN**

from sklearn.neighbors import KNeighborsClassifier

knn\_score = []

for i in range(1, 40):

knn = KNeighborsClassifier(n\_neighbors=i)

knn.fit(x\_train.T, y\_train.T)

knn\_score.append( knn.score(x\_test.T, y\_test.T) )

df = pd.DataFrame(knn\_score)

plt.figure(figsize=(7,5))

plt.plot(df.index+1, df.values, color="blue")

plt.title("K Accuracy")

plt.xlabel("K value")

plt.ylabel("Accuracy")

plt.show()

knn = KNeighborsClassifier(n\_neighbors=15)

knn.fit(x\_train.T, y\_train.T)

y\_knn\_predict = knn.predict(x\_test.T)

#ACCURACY

knn\_score = knn.score(x\_test.T, y\_test.T) \* 100

print("Test Accuracy According To KNN(K=15): {}".format(knn\_score))

##CONFUSION MATRIX

knn\_cm = confusion\_matrix(y\_test, y\_knn\_predict)

#KNN CONFUSİON MATRİX PLOT

plt.figure(figsize=(6,5))

sns.heatmap(knn\_cm, annot = True, linewidth = 0.5, linecolor="red", fmt = ".0f")

plt.xlabel("Predict Values")

plt.ylabel("True Values")

plt.title("K=15 CONFUSİON MATRİX PLOT")

plt.show()

**#%% CLASSIFICATION WITH SVM (SUPPORT VECTOR MACHINE)**

from sklearn.svm import SVC

svm = SVC(random\_state = 42)

svm.fit(x\_train.T, y\_train.T)

#ACCURACY

svm\_score = svm.score(x\_test.T, y\_test.T) \* 100

print("Test Accuracy According To SVM : {}".format(svm\_score))

#PREDICT WITH SVM

svm\_predict = svm.predict(x\_test.T)

#CONFUSION MATRIX

svm\_cm = confusion\_matrix(y\_test, svm\_predict)

#SVM CONFUSİON MATRIX PLOT

plt.figure(figsize=(5,5))

sns.heatmap(svm\_cm, annot = True, linewidth = 0.5, linecolor="red", fmt = ".0f")

plt.xlabel("Predict Values")

plt.ylabel("True Values")

plt.title("SK CONFUSİON MATRİX PLOT")

plt.show()

**# CLASSIFICATION WITH NAIVE BAYES**

from sklearn.naive\_bayes import GaussianNB

nb = GaussianNB()

nb.fit(x\_train.T, y\_train.T)

#ACCURACY

nb\_score = nb.score(x\_test.T, y\_test.T) \* 100

print("Test Accuracy According To Naive Bayes : {}".format(nb\_score))

#PREDICT WITH NAIVE BAYES

nb\_predict = nb.predict(x\_test.T)

#CONFUSION MATRIX

nb\_cm = confusion\_matrix(y\_test, nb\_predict)

#NAIVE BAYES CONFUSİON MATRIX PLOT

plt.figure(figsize=(5,5))

sns.heatmap(nb\_cm, annot = True, linewidth = 0.5, linecolor="red", fmt = ".0f")

plt.xlabel("Predict Values")

plt.ylabel("True Values")

plt.title("NAIVE BAYES CONFUSİON MATRİX PLOT")

plt.show()

**#%% CLASSIFICATION WITH DESCION TREE**

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

dt.fit(x\_train.T, y\_train.T)

#ACCURACY

dt\_score = dt.score(x\_test.T, y\_test.T) \* 100

print("Test Accuracy According To Decision Tree : {}".format(dt\_score))

#PREDICT WITH decision tree

dt\_predict = dt.predict(x\_test.T)

#CONFUSION MATRIX

dt\_cm = confusion\_matrix(y\_test, nb\_predict)

#DESCION TREE CONFUSİON MATRIX PLOT

plt.figure(figsize=(5,5))

sns.heatmap(dt\_cm, annot = True, linewidth = 0.5, linecolor="red", fmt = ".0f")

plt.xlabel("Predict Values")

plt.ylabel("True Values")

plt.title("DECISION TREE CONFUSİON MATRİX")

plt.show()

**#%% CLASSIFICATION WITH RANDOM FOREST**

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n\_estimators=300, random\_state=1)

rf.fit(x\_train.T, y\_train.T)

#ACCURACY

rf\_score = rf.score(x\_test.T, y\_test.T) \* 100

print("Test Accuracy According To Random Forest Algorithm : {}".format(rf\_score))

#PREDICT WITH RANDOM FOREST

rf\_predict = rf.predict(x\_test.T)

#CONFUSION MATRIX

rf\_cm = confusion\_matrix(y\_test, rf\_predict)

#RANDOM FOREST CONFUSİON MATRIX PLOT

plt.figure(figsize=(5,5))

sns.heatmap(rf\_cm, annot = True, linewidth = 0.5, linecolor="red", fmt = ".0f")

plt.xlabel("Predict Values")

plt.ylabel("True Values")

plt.title(" RANDOM FOREST ALGORITHM CONFUSİON MATRİX")

plt.show()

**#ACCURACY COMPARISON OF ALL ALGORITHMS**

trace = go.Bar( x=['Logistic Regression', 'KNN', 'SVM', 'Naive Bayes', 'Decision Tree', 'Random Forest'],

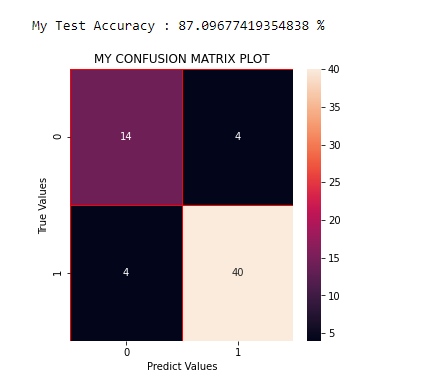
y=[lr\_score, knn\_score, svm\_score, nb\_score, dt\_score, rf\_score], marker=dict(color=['#008BF8', '#0FFF95', '#EE6C4D', '#A30000', '#2081C3', '#FF7700']),)

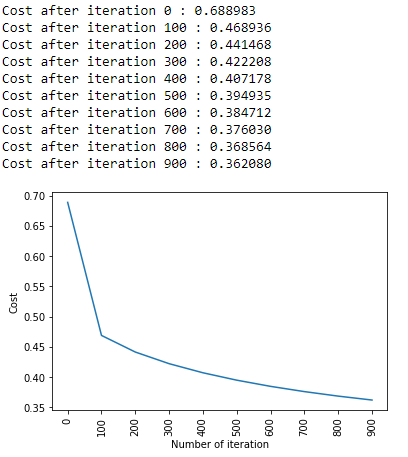
layout = go.Layout(title='Accuracy Comparison The All Algorithms',)

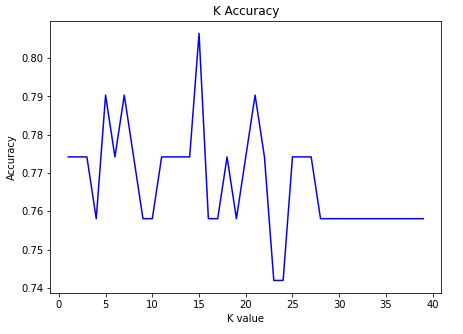
fig = go.Figure(data=[trace], layout=layout)

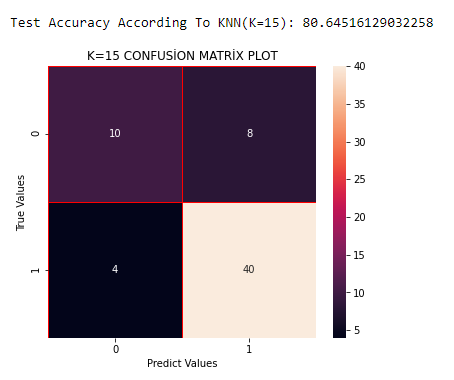
iplot(fig)

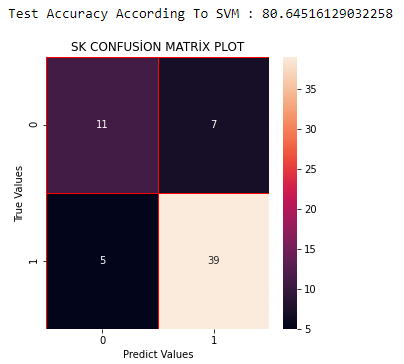
**Outputs:**

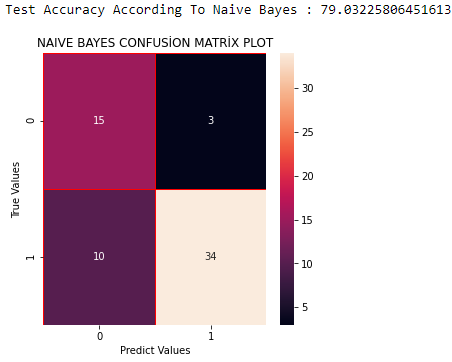


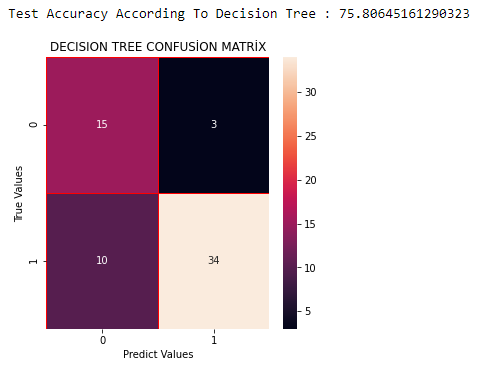


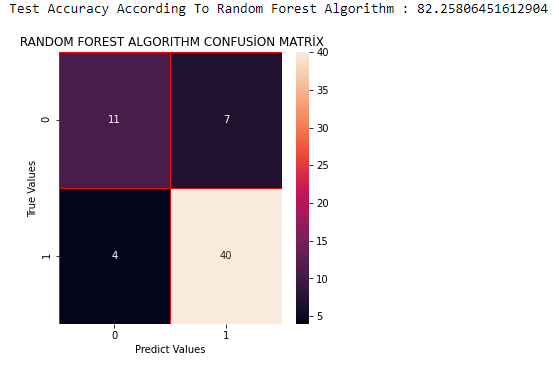


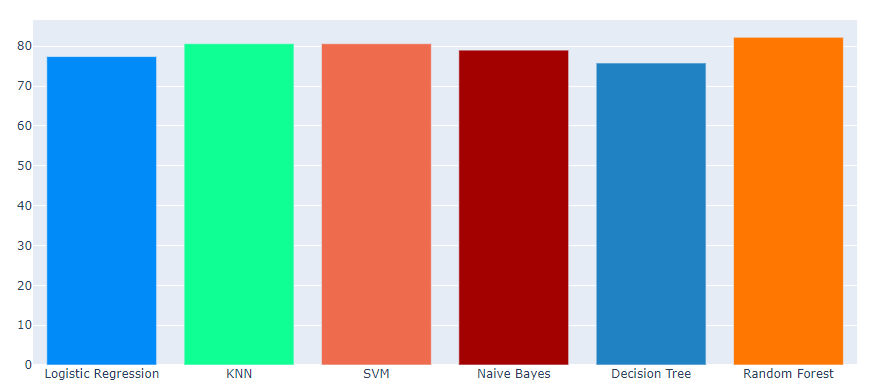












**Result:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Algorithm** | **Accuracy** |
| 1 | KNN | 80.6 |
| 2 | Support Vector Machine (SVM) | 80.64156 |
| 3 | Naïve Bayes | 79.0322 |
| 4 | Decision Tree | 75.806 |
| 5 | Random Forest | 82.258 |

**CONCLUSION:**

In the proposed system, we can see that Random forest Algorithm gives more Accuracy compared with naive Bayes, Decision Tree, K-means and Support vector Machine. It is very important to analysis the data to answer our question through data. The accuracy rate depends upon model which we use for a particle dataset.

In future work, we will explore even accurate analysis methods for the vertebral dataset, for example, support vector machines and various other classification modelling. Finally, deep learning continues to be popular and employed by the state of the art approaches. We expect this trend to continue in vertebral data analysis research, but also look forward to new innovative ideas that are discovered. Machine learning and data mining being an evolving field can generate new algorithms which can help in developing better models for mining. When there are better models, the accuracy of the diagnosis would be much higher thereby making the tool more reliable. Also hybrid algorithms like Naïve Bayes and Decision tree, Naïve Bayes and Random Forest seems to be producing higher accuracy and are reliable. We look forward for producing it.