TITLE OF THE PROJECT BASED WORK:

Early Prediction of Heart Diseases using ML based model, their comparison

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE 300: MINI PROJECT

Submitted by

DARSHAN KUMAR
(Reg. No.: 122003059, CSE 4th YEAR)

January 2022

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "**Early Prediction of Heart Diseases, Using ML Based Models, Their Comparison**" submitted as a requirement for the course, **CSE300: MINI PROJECT** for B.Tech. is a Bonafide record of the work done by **Mr. DARSHAN KUMAR (Reg. No.122003059, B.TECH ,CSE-4th YEAR**) during the academic year 2021-22, in the School of Computing , under my supervision.

Signature of Project Supervisor:

Name with Affiliation : **Professor Ezhilarasie R.**

Date : **9 Jan 2022**

Project Based Work *Viva voce* held on : 18 Jan 2022

Examiner 1 **Examiner 2**

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ML	Machine Learning
KNN	K-nearest Neighbor
NBC	Naïve Bayes Classifier
AdaBoost	Adaptive Boosting
RF	Random Forest
LR	Logistic Regression
CVD	Cardio Vascular Diseases

ABSTRACT

CVD account for 30% of deaths, worldwide. The situation , is more worse in low income countries. In most cases, CVD is detected , only in its advanced stage, but the patient cannot be saved, at this stage. Thus Early diagnosis is needed. Ideally CVD is detected using CT scan, ultra sound, etc, which are not in reach of low income, rural population of India. The idea ,of this project is to generate useful life-saving information, from the medical records, which are otherwise useless. Machine Learning(ML) based algorithms, can be used to build, efficient, low-cost prediction systems, for early screening of CVD.(affordable Healthcare). 5 State-of-the-art ML algorithms (KNN ,Naïve Bayes, Logistic Regression, Adaboost, Random Forest) were applied, using Python libraries, to develop the prediction system. The best performing model, can be deployed in web, for easy reach of all people. ML based heart disease prediction systems, can be used as screening tools, to diagnose heart diseases in , primary health centers, in rural parts of India.

KEYWORDS: CVD, Affordable healthcare, Early diagnosis, ML , KNN , Adaboost , Random Forest

INDEX

SUMMARY OF BASE PAPER

Title: Machine Learning -based heart disease prediction system for Indian population: An exploratory study done in south India

Journal Name: Medical Journal Armed Forces India

Publisher: Elsevier

Year: 2020

Indexed in: Scopus(Elsevier)

Cardio Vascular Diseases , are the reason behind 17.8 million deaths(30% of total deaths ,worldwide). The situation is critical in middle-income Asian countries, like Bangladesh, India, where deaths due to CVD increased from 15.2% to 28.1%. CVD are mostly Detected in its advanced stages, among the under-privileged patients. But a patient cannot be saved, in the advanced stage. The current Base Paper, proposes to solve the above problem, using ML based models, as affordable, reliable, accurate Screening tools, thus making useful information, from the huge data generated/preserved, by the health care sectors. Tests like ultrasound, CT scan, MRI, etc. are used in western countries, to detect CVD, but it is out of reach , for rural population of India.

Most of the prediction systems, developed, so far, in western countries, do not take into account, the context of Indian lives. In India, the primary factors of heart diseases include: obesity ,hierarchical heart conditions, lack of physical activity, stress at workplace, smoking , alcohol consumption, history of hyper tension, diabetes, etc. These factors were not considered, in any of ML-based studies done so far.

The study has 2 objectives:

1) Development of accurate, cost effective prediction system ,for heart diseases, which is trained using routine clinical data, from primary health care centers, specifically suited for rural population.
2) Deployment of models, in websites, hosted in public cloud, for easy access , especially for rural areas.

Data Collection

The dataset used in base paper, is taken from medical records of tertiary hospitals ,from South India, and the patients included, both healthy patients, and those with heart ailments.

Exclusion criteria:

1)records of pregnant females

2)patients older than 100, younger than 20.

3)Patients with severe chronic diseases

4)Patients who reported to have been addicted to drugs, etc.

Risk Factor Attributes

To ensure cheaper cost, tests for triglycerides, insulin tests, etc were not considered, even though they are associated with heart diseases, in base paper

Among other attributes, following tests were proposed , to identify significant attributes:

1) T-test
2) Chi Square Test

P-value < 0.05 is chosen as the criteria.(randomness less than 5%, in data)

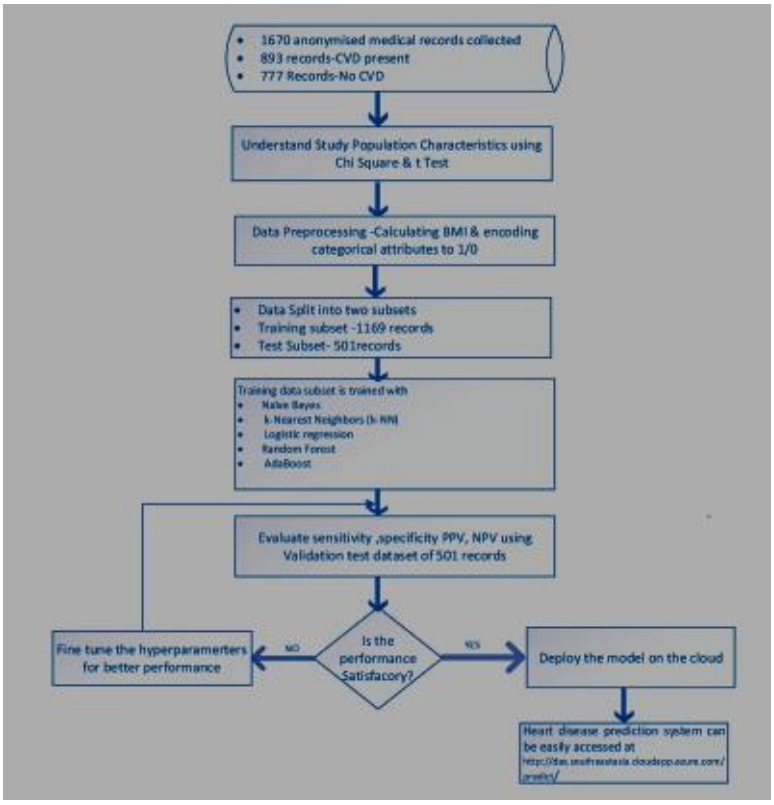


Fig 1: Workflow Proposed in base paper

Data Pre-Processing

1) non-numerical attributes, were label encoded, to get numerical data

2) for missing values, in Continuous valued Attributes, they were replaced, with mean of remaining values.

3)for missing values, in Categorical attributes, they were replaced with , the most frequently occurring value, for that particular attribute.

Table 1 – Description of attributes used in the study.

Attributes	Description	Categorical/ numeric
Age	Years	Numeric
Weight	Kilograms	Numeric
Height	Centimetres	Numeric
Total cholesterol levels	mg/dL	Numeric
Gender	Male/female	Categorical
Hypertension	Yes/no	Categorical
Diabetes	Yes/no	Categorical
Alcohol	Yes/no	Categorical
Smoking	Yes/no	Categorical
Exercise	Yes/no	Categorical
Stress	Yes/no	Categorical
Family history of cardiovascular disease (CVD)	Yes/no	Categorical
Healthy diet	Yes/no	Categorical
Risk of CVD	High/low	Categorical

Fig2: Description of attributes, used in study

Building the Model

Rows were randomly shuffled, and then 70% of the data was used for training , the models, and the rest 30% was used for testing the models.

In the data set used in the base paper, out of 1670 records, training subset had 1169 records.656 records of 1169, belonged to CVD class, rest were belonged to health people, with no CVD.

5 state of the art models were built:

1) K Nearest Neighbor (KNN)

2) Naïve Bayes Classifier (NBC)

3) Logistic Regression (LR)

4)Adaptive Boosting (AB)

5)Random Forest (RF)

Class	Training subset (70%)	Test subset (30%)	Total records
High-risk cardiovascular disease (CVD)	656	237	893
Low-risk CVD	513	264	777
Total records	1169	501	1670

Fig3: Details of Training and Testing subsets

Testing the Performance of the Models

following 4 components , of confusion matrix , were considered:

1) TP=True Positives

2) TN=True Negatives

3)FP=False Positives

4)FN=False Negatives

The following 5 parameters were calculated for every model, for comparison:

1)Classification Accuracy: (TN+TP) / [TN+TP+FP+FN]

2)Sensitivity: TP / [TP+FN]

3)Specificity: TN / [FP+TN]

4)PPV(positive predicted value): TP / [TP+FP]

5)NPV(negative predicted value): TN / [TN+FN]

Fine Tuning of Hyper Parameters

Base paper proposes to use : SK-Learn library's Grid Search CV class, to identify the best parameters, for each model.

Deployment

"Pickle" , "Flask", libraries were proposed to be used, to deploy the models, in backend of website, on Microsoft Azure Cloud.

Results

- KNN(N=12) showed sensitivity of 89%, specificity of 87.1%. Naïve Bayes showed sensitivity of 88.6%, specificity of 87.8%. Logistic regression showed accuracy of 90.8%. Models with ensemble techniques (Adaboost, RF) performed better than LR. They had higher accuracy, than LR.

- "Lack of exercise" and "stress" were found to be the most contributing factors for CVD.

Further Discussions:

- The models developed , can be trained with quality , voluminous data sets, and can be used as screening tool in health care centers, in rural India, and thus can detect CVD, in its initial stage itself.

- Other modern techniques, such as artificial neural networks(ANN), can be used, to further enhance the performance(accuracy) .

Limitations:

- The accuracy, overall performance depends solely on the data set used, and if the data set, does not have almost equal portions of genuine records, of both healthy, affected patients, then the models cannot be trained , to high accuracy.

- The aim of this study, was only to detect, robustness of the prediction models, based on ML.

Conclusions:

Issues of affordability, accessibility, reliability in healthcare sector, in rural India, can be addressed using state-of-the-art ML based prediction Models, which can easily accessed, via mobile based internet, in rural parts of India. Voluminous data, recorded in hospital databases ,will now, be made to generate useful information.

MERITS-DEMERITS OF BASE PAPER

MERITS

-starting from basic algorithms, like KNN, to hybrid algorithms, like Adaptive boosting, random forest, thus the base paper, proposes a complete solution.

- It takes into account, the Indian Context, thus focuses on factors like stress, less physical activity, etc attributes, which were not part of previous studies.

- Only basic, medical test results are considered, which are affordable in rural households, as people from rural setup, don't have access, to advanced tests.

-Records of medical test results, which were otherwise, of less use, are utilized, for this , just, extracting , useful information from them.

DE-MERITS

-The accuracy of the models, is dependent on the data set used. The data set used in base paper, is not open sourced.

-Methods like SVM, could have shown, better accuracy, but were not used.

-algorithms for the implementation of hybrid models, like Adaptive boosting, are not mentioned/detailed.

-effective pre-processing tests, like Correlation Test, are not used, in the base paper.

-Methods used for dealing with missing values, is not mentioned in the base paper.

- Neural Networks, Perceptron, which could have shown better accuracy, for the problem statement, were not included.

EXISTING TECHNIQUES FOR SIMILAR PROBLEMS

-SVM is expected to be a effective solution, for similar problems as it assures to give global minima of the error, but its training will be resource intensive .

- Neural Networks(NN), with more than 1 hidden layers, is expected to give the highest accuracy . But training of NN, through Back Propagation, is resource intensive, and takes lots of time, unless done on a server

SOURCE CODE FOR DATA VISUALISATION AND PERFORMING T-TEST,CHI-SQUARE TEST,CORRELATION TEST

```
# Data Preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind
import seaborn as sns
from pylab import rcParams
df=pd.read_csv("framingham.csv")
df
#  to remove rows and columns with Null/NaN values.
df=df.dropna()
df
df.shape
# Exploratory Data analysis
df.groupby("POSSIBILITY OF CVD").mean()
df.groupby("male").mean()
df.groupby("currentSmoker").mean()
df.groupby("prevalentStroke").mean()
df.groupby("prevalentHyp").mean()
df.groupby("diabetes").mean()
### gender wise analysis of CVD:-
data=df.values #converting DataFrame object into np array object
print(data)
male_cvd=female_cvd=female_nocvd=male_nocvd=0
m=len(data) # m= 3656
for i in range(m):
    gen,cvd=data[i][0],data[i][-1]
```

```
    if(gen==1):
        if(cvd==1):
            male_cvd+=1
        else:
            male_nocvd+=1
    else:
        if(cvd==1):
            female_cvd+=1
        else:
            female_nocvd+=1
l=[[male_cvd,male_nocvd,male_cvd+male_nocvd],[female_cvd,female_nocvd,female_cvd+female_nocvd],[male_cvd+female_cvd,male_nocvd+female_nocvd]]
genDistribution=pd.DataFrame(data=l,columns=["have CVD","not having CVD","Total of each Gender"],index=["Male","Female","Total"])
display(genDistribution)

# Data Visualisation
plt.style.use("seaborn")
plt.figure(figsize=(19,10))
plt.scatter(df["heartRate"],df["POSSIBILITY OF CVD"])
plt.xlabel("heartRate")
plt.ylabel("cvd possibility")

plt.show()
plt.style.use("seaborn")
plt.figure(figsize=(19,10))
plt.scatter(df["diaBP"],df["POSSIBILITY OF CVD"])
plt.xlabel("diastolic BP")
plt.ylabel("cvd possibility")

plt.show()
```

```
plt.style.use("seaborn")
plt.figure(figsize=(19,10))
plt.scatter(df["sysBP"],df["POSSIBILITY OF CVD"])
plt.xlabel("systolic BP")
plt.ylabel("cvd possibility")

plt.show()
plt.style.use("seaborn")
plt.figure(figsize=(19,10))
plt.scatter(df["glucose"],df["POSSIBILITY OF CVD"])
plt.xlabel("glucose")
plt.ylabel("cvd possibility")

plt.show()
plt.style.use("seaborn")
plt.figure(figsize=(19,10))
plt.scatter(df["totChol"],df["POSSIBILITY OF CVD"])
plt.xlabel("average cholestrol ")
plt.ylabel("cvd possibility")

plt.show()

plt.style.use("seaborn")
plt.figure(figsize=(19,10))
plt.scatter(df["cigsPerDay"],df["POSSIBILITY OF CVD"])
plt.xlabel("average cigrates consumed per day ")
plt.ylabel("cvd possibility")

plt.show()
plt.style.use("seaborn")
plt.figure(figsize=(19,10))
```

```
plt.scatter(df["age"],df["POSSIBILITY OF CVD"])
plt.xlabel("age")
plt.ylabel("cvd possibility")

plt.show()
plt.style.use("seaborn")
plt.figure(figsize=(19,10))
plt.scatter(df["education"],df["POSSIBILITY OF CVD"])
plt.xlabel("education level ")
plt.ylabel("cvd possibility")

plt.show()
# Finding Means to predict the result of t-test

%matplotlib inline
rcParams['figure.figsize'] = 25,10
rcParams['font.size'] = 30
sns.set()
#np.random.seed(8)
def plot_distribution(inp):
    plt.figure()
    ax = sns.distplot(inp)
    plt.axvline(np.mean(inp), color="k", linestyle="dashed", linewidth=5)
    plt.axvline(np.min(inp), color="green", linestyle="dashed", linewidth=1)
    plt.axvline(np.max(inp), color="red", linestyle="dashed", linewidth=1)

    _, max_ = plt.ylim()
    plt.text(
        inp.mean(),
        max_ - max_ / 11,
        "Mean: {:.2f}".format(inp.mean()),
```

```
    )
    plt.text(
        inp.mean(),
        (max_ - max_ / 8),
        "std deviation: {:.2f}".format(inp.std()),
    )
    plt.text(
        inp.mean(),
        (max_ - max_ / 5),
        "variance: {:.2f}".format(inp.var()),
    )
    plt.text(
        inp.min(),
        max_ - max_ / 10,
        "Min: {:.2f}".format(inp.min()),
    )
    plt.text(
        inp.max(),
        max_ - max_ / 10,
        "Max: {:.2f}".format(inp.max()),
    )

    return plt.figure
plot_distribution(df["male"])
plot_distribution(df["age"])
plot_distribution(df["cigsPerDay"])
plot_distribution(df["totChol"])
plot_distribution(df["sysBP"])
plot_distribution(df["BMI"])
plot_distribution(df["diaBP"])
plot_distribution(df["heartRate"])
```

```
plot_distribution(df["glucose"])
# Performing T Test
# alpha =0.05 is given in base paper
def compare_2_groups(arr_1, arr_2, alpha=0.05):
    stat, p = ttest_ind(arr_1[50:70], arr_2[50:70],equal_var=False)
    print('T Statistics=%.3f, p value=%.3f' % (stat, p))
    if p < alpha:
        print('Statistically Significant')
    else:
        print('Statistically IN-Significant')
compare_2_groups(df["POSSIBILITY OF CVD"],df["male"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["age"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["education"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["currentSmoker"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["cigsPerDay"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["BPMeds"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["prevalentStroke"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["prevalentHyp"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["diabetes"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["totChol"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["sysBP"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["diaBP"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["BMI"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["heartRate"])
compare_2_groups(df["POSSIBILITY OF CVD"],df["glucose"])
# Finding "CORRELATION" of each attribute with, with Target attribute
predictionVar=df["POSSIBILITY OF CVD"]
df.corrwith(predictionVar,drop=True)
```

```
# puting above data in a DataFrame object
similarityMeasure=pd.DataFrame(df.corrwith(predictionVar,drop=True),columns=["correlation
Value"])
similarityMeasure
# important observation:    "education" has -ve correlation
# finding attributes , whose "correlation Value" >0.09
similarityMeasure[similarityMeasure["correlation Value"]>0.09]
# sorting above result in descending order
similarityMeasure[similarityMeasure["correlation          Value"]>0.09].sort_values("correlation
Value",ascending=False)
# Performing Chi-square Test
### (chi-square test is possible only for for categorical attributes)
### NULL Hypothesis: there is NO relation between the 2 attributes
### Alternative Hypothesis:  there is  relation between the 2 attributes
from scipy import stats
#### showing how it works , with "male" attribute as a sample
### step1: create contingency table
# creating contingency table (or) crosstab
c_table1=pd.crosstab(index=df['male'],columns=df['POSSIBILITY OF CVD'])
# displaying contingency table
c_table1
# this is how we access the 1st row, of above contingency table
c_table1.iloc[0].values
### step2: Perform chi-square test, with the target attribute
(chi1,p1,dof,_)=stats.chi2_contingency([c_table1.iloc[0].values,c_table1.iloc[1].values])
print("chi=",chi1,"\np value=",p1)
## performing chi-square test on all categorical attributes, and displaying result
l=[]
# for "male" attribute
l.append([p1,chi1,p1<0.05])
```

```
#for "currentSmoker" attribute
c_table1=pd.crosstab(index=df['currentSmoker'],columns=df['POSSIBILITY OF CVD'])
(chi1,p1,dof,_)=stats.chi2_contingency([c_table1.iloc[0].values,c_table1.iloc[1].values])
l.append([p1,chi1,p1<0.05])
#
c_table1=pd.crosstab(index=df['BPMeds'],columns=df['POSSIBILITY OF CVD'])
(chi1,p1,dof,_)=stats.chi2_contingency([c_table1.iloc[0].values,c_table1.iloc[1].values])
l.append([p1,chi1,p1<0.05])
#
c_table1=pd.crosstab(index=df['prevalentStroke'],columns=df['POSSIBILITY OF CVD'])
(chi1,p1,dof,_)=stats.chi2_contingency([c_table1.iloc[0].values,c_table1.iloc[1].values])
l.append([p1,chi1,p1<0.05])#
c_table1=pd.crosstab(index=df['prevalentHyp'],columns=df['POSSIBILITY OF CVD'])
(chi1,p1,dof,_)=stats.chi2_contingency([c_table1.iloc[0].values,c_table1.iloc[1].values])
l.append([p1,chi1,p1<0.05])#
c_table1=pd.crosstab(index=df['diabetes'],columns=df['POSSIBILITY OF CVD'])
(chi1,p1,dof,_)=stats.chi2_contingency([c_table1.iloc[0].values,c_table1.iloc[1].values])
l.append([p1,chi1,p1<0.05])
#print(l)

k1=pd.DataFrame(data=l,columns=["p                                    value","chi
value","p<0.05"],index=["male","currentSmoker","BPMeds","prevalentStroke","prevalentHyp",
"diabetes"])
display(k1)
print("\n\nrounding 'p value' to 5 decimal places, we get:")
k1.round(5)
### when p<0.05 we reject NULL hypothesis
```

SOURCE CODE FOR IMPLEMENTATION OF ML Models

KNN

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
pd.read_csv?
df=pd.read_csv("framingham.csv")
df
# "education" showed -ve correlation, with target label
# "male" was shown to be IN SIGNIFICANT, during T TEst(based on p-value)
# "diabetes" was shown to be IN SIGNIFICANT, during T TEst(based on p-value)
# "currentSmoker" fails in chi-square test
df=df.drop(axis=1,labels=["male","education","diabetes","currentSmoker"])
df
## Preprocessing data
#To understand , which attribute has how many ("NULL" or NaN) values
df.info()
# each column has 4238 entries.
# for attribute "glucose" , there are only 3850 "not-NULL" entries
#df.fillna?
# for categorical attributes, we replace Nan with most frequent attribute
#for continous attributes,we replace Nan with mean .

#for continous attributes--->replace Nan with mean
df["cigsPerDay"]=df["cigsPerDay"].fillna(round(df["cigsPerDay"].mean()))
df["totChol"]=df["totChol"].fillna(round(df["totChol"].mean()))
df["BMI"]=df["BMI"].fillna(round(df["BMI"].mean(),2))
df["heartRate"]=df["heartRate"].fillna(round(df["heartRate"].mean()))
```

```
df["glucose"]=df["glucose"].fillna(round(df["glucose"].mean()))



#for categorical attributes-->replace Nan with most freq value
vall,freqq=np.unique(df["BPMeds"],return_counts=True)
#display(vall,freqq)
df["BPMeds"]=df["BPMeds"].fillna(vall[freqq.argmax()])

display(df)
df.info()
# KNN Implementation
### Step 0: preparing data
# random shuffling of data
#df=df.sample(frac=1)
#display(df)

#not doing this step, as data is already shuffled, as per dataset discription
# taking data , from the DataFrame object, to numpy object
data=df.values
print(data)
print(data.shape)
print(type(data))
#separating target label, from rest of the attributes
X=data[:,:-1]
Y=data[:,-1]  # target label
print(X)
print(X.shape)
print(Y)
print(Y.shape)
### step1 : splitting data for training and testing
#performing split , to train,test data
```

```
split=0.7*X.shape[0]
print(split)
split=int(split)
print("split length:",split)
print()

X_train= X[:split,:]
Y_train=Y[:split]
X_test=X[split:,:]
Y_test=Y[split:]

print("X_train:",X_train.shape)
print("X_test:",X_test.shape)
print("Y_train:",Y_train.shape)
print("Y_test:",Y_test.shape)
### Step 2: knn implementation (k=5)
#function to calculaate eucledian distance
def EucledianDist(x1,x2):
    return np.sqrt(sum((x1-x2)**2))
# A EXAMPLE
# defining 2 points,  in 4 dimension space
a=np.array([3,3,3,3])
b=np.array([1,1,1,1])
# calculating the eucledian distance , between these 2 points
EucledianDist(a,b)
# ie sqrt(4+4+4+4)=4
def knn(queryPoint,printIntermidiateResults=False,k=5):
    values=[]  #to store dist of query point, with each of training points,with label
    m=X_train.shape[0]  # ie m=2559

    for i in range(m):
```

```
        d=EucledianDist(queryPoint,X_train[i])
        values.append((d,Y_train[i]))

    #print(values)
    values.sort(key=lambda z:z[0])  #sort in ascending order of eucledian dist
    #print(values)
    values=values[:k]  # K nearest neighbour
    if(printIntermidiateResults):
        print("k nearest points, along with their label:\n",values)
    values=np.array(values) #converting to np array, to do slicing property

    #count the frequency of each of the unique labels
    new_values=np.unique(values[:,1],return_counts=True)
    if(printIntermidiateResults):
        print("\nunique labels, their frequency:\n",new_values)
    indexOf_MostFreqLabel=new_values[1].argmax()

    prediction=new_values[0][indexOf_MostFreqLabel]
    if(printIntermidiateResults):
        print("\npredicted label=",prediction)
    return prediction

#### understanding how it works, on 1 sample point

knn(X_test[15],True)
#compare above result of KNN, with the actual Label
print(Y_test[:20])
print("actual Label=",Y_test[15])
### Step 3: Testing with "Testing Set"
def TestKNN(k_parameter=5):
    m=X_test.shape[0] # m=1097
```

```
    print("number of test cases:",m)

    val=[]
    correctPrediction=0
    wrongPrediction=0
    for i in range(m):
        modelPrediction=knn(X_test[i],k=k_parameter)
        l=[modelPrediction,Y_test[i]]
        val.append(l)
        if(modelPrediction!=Y_test[i]):
            wrongPrediction+=1
        else:
            correctPrediction+=1
    df1=pd.DataFrame(data=val,columns=["Label predicted by KNN","Actual Label"])

    print("\n% of correct Prediction:",correctPrediction*100/m)
    print("% of wrong Prediction:",wrongPrediction*100/m)
    display(df1)
TestKNN(k_parameter=5)
TestKNN(k_parameter=15)
TestKNN(k_parameter=95)
# most of the Predicted Labels are 0,
#not at all helpful, to predict cvd
# Implementing KNN with sk-learn
from sklearn.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier(n_neighbors = 15)
knc.fit(X_train,Y_train)
knc.score(X_train,Y_train)
knc.score(X_test,Y_test)
# Evaluating Performance , by Evaluating Parameters
Y_pred=knc.predict(X_test)
```

```
def evaluateParameters(Y_pred,Y_test):
    tp,tn,fp,fn=0,0,0,0
    for i in range(len(Y_test)):
        actual=Y_test[i]
        if(actual==1):
            if(actual==Y_pred[i]):
                tp+=1
            else:
                fn+=1
        else:
            if(actual==Y_pred[i]):
                tn+=1
            else:
                fp+=1
    print("\naccuracy= ",(tp+tn)/(tp+tn+fn+fp))
    print("\nsensitivity= ",tp/(tp+fn))
    print("\nspecificity= ",tn/(fp+tn))
    print("\nPPv= ",tp/(tp+fp))
    print("\nNPV= ",tn/(tn+fn))

evaluateParameters(Y_pred,Y_test)
# save the trained model, for deploying on web

import pickle
with open('knn_model.pkl','wb') as f:
    pickle.dump(knc,f)
```

Naïve Bayes Classifier

```
import pandas as pd
import numpy as np
df=pd.read_csv("framingham.csv")
display(df)
## Preprocessing data
# "male" was shown to be IN SIGNIFICANT, during T TEst(based on p-value)
# "diabetes" was shown to be IN SIGNIFICANT, during T TEst(based on p-value)
# "currentSmoker" fails in chi-square test
df=df.drop(axis=1,labels=["male","education","diabetes","currentSmoker"])
df

#To understand , which attribute has how many ("NULL" or NaN) values
df.info()
# each column has 4238 entries.
# for attribute "glucose" , there are only 3850 "not-NULL" entries
#df.fillna?
# for categorical attributes, we replace Nan with most frequent attribute
#for continous attributes,we replace Nan with mean .

#for continous attributes--->replace Nan with mean
df["cigsPerDay"]=df["cigsPerDay"].fillna(round(df["cigsPerDay"].mean()))
df["totChol"]=df["totChol"].fillna(round(df["totChol"].mean()))
df["BMI"]=df["BMI"].fillna(round(df["BMI"].mean(),2))
df["heartRate"]=df["heartRate"].fillna(round(df["heartRate"].mean()))
df["glucose"]=df["glucose"].fillna(round(df["glucose"].mean()))


#for categorical attributes-->replace Nan with most freq value
vall,freqq=np.unique(df["BPMeds"],return_counts=True)
```

```
#display(vall,freqq)
df["BPMeds"]=df["BPMeds"].fillna(vall[freqq.argmax()])

display(df)
df.info()
# Naive Bayes Classifier Implementation
### Step 0: preparing data
# random shuffling of data
#df=df.sample(frac=1)
#display(df)

#not doing this step, as data is already shuffled, as per dataset discription
# taking data , from the DataFrame object, to numpy object
data=df.values
print(data)
print(data.shape)
print(type(data))
#separating target label, from rest of the attributes
X=data[:,:-1]
Y=data[:,-1]  # target label
print(X)
print(X.shape)
print(Y)
print(Y.shape)
### step1 : splitting data for training and testing
#performing split , to train,test data
split=0.7*X.shape[0]
print(split)
split=int(split)
print("split length:",split)
print()
```

```
X_train= X[:split,:]
Y_train=Y[:split]
X_test=X[split:,:]
Y_test=Y[split:]

print("X_train:",X_train.shape)
print("X_test:",X_test.shape)
print("Y_train:",Y_train.shape)
print("Y_test:",Y_test.shape)
### Step 2: NBC implementation
#with laplace correction
def prior_prob(label,y_train=Y_train):
    totalNumber_TrainingSamples=y_train.shape[0] #2559
    totalNumber_withLabel=np.sum(y_train==label)
    V=len(np.unique(y_train))
    #print(V)

    return (totalNumber_withLabel+1)/(totalNumber_TrainingSamples+V)

# using the above function
print(prior_prob(0),prior_prob(1))
#implementing with Laplace Correction
def cond_prob(column_index,column_value_inQuery,label,x_train=X_train,y_train=Y_train):
    x_filtered=x_train[y_train==label]
    V=len(np.unique(x_train[:,column_index]))
    numerator=np.sum(x_filtered[:,column_index]==column_value_inQuery)
    denominator=np.sum(y_train==label)

    return (numerator+1)/(denominator+V)
def predictLabel(queryRow,x_train=X_train,y_train=Y_train):
```

```
    classLabelsWeHave=np.unique(y_train) #[0,1]
    numberOfFeatures=x_train.shape[1] #12
    probalilities=[] #to store probability of all classes
    #calculating posterior probability for each label
    for label in classLabelsWeHave:
        likelihood=1.0
        for f in range(numberOfFeatures):
            conditionalProbability=cond_prob(f,queryRow[f],label)
            likelihood*=conditionalProbability
        prior=prior_prob(label)
        postProb=likelihood*prior
        probalilities.append(postProb)
    #print( probalilities)
    index=np.argmax(probalilities) #returns index of max value, in the list
    prediction=classLabelsWeHave[index]
    return prediction



#### understanding how it works, on 1 sample point
predictLabel(X_test[15])

print(Y_test[:20])
print(Y_test[15])



### Step 3: Testing with "Testing Set"
def findScore(x_test=X_test,y_test=Y_test):
    m=y_test.shape[0] #1097
    l=[]
```

```
    count_correctPrediction=0
    for i in range(m):
        predictedLabel=predictLabel(x_test[i])
        l.append((predictedLabel,y_test[i]))
        if(predictedLabel==y_test[i]):
            count_correctPrediction+=1
    print("accuracy %=",count_correctPrediction*100/m)
    df1=pd.DataFrame(data=l,columns=["Predicted Label","Actual Label"])
    display(df1)

findScore()
# implementing NBC using models of sk_learn
from sklearn.naive_bayes import GaussianNB,MultinomialNB
gnb=GaussianNB()
mnb=MultinomialNB()
gnb.fit(X_train,Y_train)
mnb.fit(X_train,Y_train)


#calculating score (or) accuracy
score=gnb.score(X_test,Y_test)
print("accuracy of gaussianNB=",score*100)
score=mnb.score(X_test,Y_test)
print("accuracy of multinomialNB=",score*100)
gnb.score?
# conclusion
### Since some of our important attributes are "continonus valued", thus, the "GAUSSIAN MODEL", gives better result
### Measuring Performance , by Evaluating Parameters
def evaluateParameters(Y_pred,Y_test):
    tp,tn,fp,fn=0,0,0,0
```

```
    for i in range(len(Y_test)):
        actual=Y_test[i]
        if(actual==1):
            if(actual==Y_pred[i]):
                tp+=1
            else:
                fn+=1
        else:
            if(actual==Y_pred[i]):
                tn+=1
            else:
                fp+=1
    print("\naccuracy= ",(tp+tn)/(tp+tn+fn+fp))
    print("\nsensitivity= ",tp/(tp+fn))
    print("\nspecificity= ",tn/(fp+tn))
    print("\nPPv= ",tp/(tp+fp))
    print("\nNPV= ",tn/(tn+fn))



Y_pred=gnb.predict(X_test)
evaluateParameters(Y_pred,Y_test)
# saving trained model, for deploying on web
import pickle
with open('nbc_model.pkl','wb') as f:
    pickle.dump(gnb,f)
```

Logistic Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df=pd.read_csv("framingham.csv")
display(df)
# Preprocessing data
# "male" was shown to be IN SIGNIFICANT, during T TEst(based on p-value)
# "diabetes" was shown to be IN SIGNIFICANT, during T TEst(based on p-value)
# "currentSmoker" fails in chi-square test
df=df.drop(axis=1,labels=["male","education","diabetes","currentSmoker"])
df

#To understand , which attribute has how many ("NULL" or NaN) values
df.info()
# each column has 4238 entries.
# for attribute "glucose" , there are only 3850 "not-NULL" entries
#df.fillna?
# for categorical attributes, we replace Nan with most frequent attribute
#for continous attributes,we replace Nan with mean .

#for continous attributes--->replace Nan with mean
df["cigsPerDay"]=df["cigsPerDay"].fillna(round(df["cigsPerDay"].mean()))
df["totChol"]=df["totChol"].fillna(round(df["totChol"].mean()))
df["BMI"]=df["BMI"].fillna(round(df["BMI"].mean(),2))
df["heartRate"]=df["heartRate"].fillna(round(df["heartRate"].mean()))
df["glucose"]=df["glucose"].fillna(round(df["glucose"].mean()))
```

```
#for categorical attributes-->replace Nan with most freq value
vall,freqq=np.unique(df["BPMeds"],return_counts=True)
#display(vall,freqq)
df["BPMeds"]=df["BPMeds"].fillna(vall[freqq.argmax()])

display(df)
df.info()

# Logistic Regression Implementation
### step 0: Preparing Data
# random shuffling of data
#df=df.sample(frac=1)
#display(df)

#not doing this step, as data is already shuffled, as per dataset discription
# taking data , from the DataFrame object, to numpy object
data=df.values
print(data)
print(data.shape)
print(type(data))
#separating target label, from rest of the attributes
X=data[:,:-1]
Y=data[:,-1]  # target label
print(X)
print(X.shape)
print(Y)
print(Y.shape)
```

```
### performing Data Normalisation
 # data normalisation (mean=0,,,and std deviation=1 along each axis)
# squeezing the features to [-1,1] range
x_mean=X.mean(axis=0)
print(x_mean)
x_std=X.std(axis=0)
print(x_std)

#applying the transformation
X=(X-x_mean)/x_std
print(X.mean(axis=0))
# mean is v close to 0 now
print(X.std(axis=0))
#std dev is 1 along each axis
### step1: Spliting Data for Training and Testing
#performing split , to train,test data
split=0.7*X.shape[0]
print(split)
split=int(split)
print("split length:",split)
print()

X_train= X[:split,:]
Y_train=Y[:split]
X_test=X[split:,:]
Y_test=Y[split:]

print("X_train:",X_train.shape)
```

```
print("X_test:",X_test.shape)
print("Y_train:",Y_train.shape)
print("Y_test:",Y_test.shape)
X_train
### step2 : LR Implementation
 def sigmoid(z):
       return 1.0/(1.0 + np.exp(-z))

#sigmoid value for a large number is 1
sigmoid(100)
#sigmoid value for a smaller number is approx 0
sigmoid(-100)
def hypothesis(x,theta):
    """
    x= entire data array(with 0th column as 1, added mannually): m,n+1
    theta= n+1,1
    """
    #this gives gavues of the "prediction" we are making
    return sigmoid(np.dot(x,theta))
# to find optimal theta
# error (or) cost function
# cost function is a "convex curve",thus global minima can be obtained, using "gradient descent"
def error(x,y,theta):
    """
    x=m,n+1
    y=m,1
    theta=n+1,1
```

```
    return : scalar_value =loss
    """
    hi=hypothesis(x,theta)
    #finding error
    e=-1*np.mean((y*np.log(hi) +  (1-y)*np.log(1-hi)))
    return e
# initially theta has all values as 0
# thus our separating line, initially is the x -axis
def gradient(x,y,theta):
    """
    x=m,n+1
    y=m,1
    theta=n+1,1

    return : gradient_vector =(n+1,1)
    """
    hi=hypothesis(x,theta)
    # "hi" is our prediction , based on current "theta"
    grad=-np.dot(x.T,(y-hi))
    m=x.shape[0]
    return grad/m



def gradient_descent(x,y,lr=0.1,max_itr=200):
    """
    lr:learning rate
    max_itr: max iterations we will take, to get optimal theta
    x=m,n+1
```

```
    y=m,1
    """
    #no of attributes , for each row
    n=X.shape[1]
    theta=np.zeros((n+1,1))

    error_list=[]
    # we will "error_list" in graph

    for i in range(max_itr):
        err=error(x,y,theta)
        error_list.append(err)

        grad=gradient(x,y,theta)
        #update theta using "gradient descent"
        theta=theta-(lr*grad)
    return (theta,error_list)



# initial theta
# thus our separating line, initially is the x -axis
n=X.shape[1]
theta=np.zeros((n,1))
theta
# adding a column of "1" as the 1st column, to existing training set
ones=np.ones((X_train.shape[0],1))
X_new_train=np.hstack((ones,X_train))
```

```
#converting Y_train, from horizontal to vertical vector
Y_train_new=Y_train.reshape((-1,1))
print("after reshaping Y_train_new:",Y_train_new)
print("old shape",X_train.shape)
print("new shape",X_new_train.shape)
#printing 1st column
print("1st column=",X_new_train[:,0])
theta,error_list=gradient_descent(X_new_train,Y_train_new)
plt.plot(error_list)
# "error" decreases sharply, with higher "learning rate"
theta,error_list=gradient_descent(X_new_train,Y_train_new,lr=0.5)
plt.plot(error_list)
#printing optimal theta
theta



### step 3: Testing with "Testing Set"
def predict(x,theta):
    h=hypothesis(x,theta)
    output=np.zeros(h.shape)

    output[h>=0.5]=1
    output=output.astype('int')
    return output,h
# adding a column of "1" as the 1st column, to testing set
ones=np.ones((X_test.shape[0],1))
X_new_test=np.hstack((ones,X_test))
```

```
print("old shape",X_test.shape)
print("new shape",X_new_test.shape)
#printing 1st column
print("printing 1st 3 rows,columns of X_new_train:")
print(X_new_test[:3,:4])

#converting Y_test, from horizontal to vertical vector
Y_test_new=Y_test.reshape((-1,1))
print("Y_test_new:")
print(Y_test_new)
train_data_preds,h=predict(X_new_train,theta)
# hypothesis or prediction
print("h=",h)
print("train_data_preds=",train_data_preds)

#doing similarily for test data
test_data_preds,_=predict(X_new_test,theta)
def accuracy(actual,preds):
    actual=actual.astype('int')
    #ensure that "actua","preds" are vertical vectors, with same dimensions
    correctPred=np.sum(actual==preds)
    acc=correctPred/actual.shape[0]
    return acc*100


train_acc=accuracy(Y_train_new,train_data_preds)
```

```
print(train_acc)
test_acc=accuracy(Y_test_new,test_data_preds)
print(test_acc)

# Prediction using inbuilt SKLearn (LR model)
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
### training the model
model.fit(X_train,Y_train)
# for 1st value
theta_0=model.intercept_
#for rest of the values
theta_s=model.coef_
print(theta_0,theta_s)



# compare these values , with "optimal theta" that we got
print("\n\ncomapring it with the theta that we got, previously:\n")
print(theta)
### calculating Score
#score on training data
model.score(X_train,Y_train)*100
#score on testing data
model.score(X_test,Y_test)*100
### measuring Performane by evaluating Parameters
Y_pred=model.predict(X_test)
def evaluateParameters(Y_pred,Y_test):
```

```
    tp,tn,fp,fn=0,0,0,0
    for i in range(len(Y_test)):
        actual=Y_test[i]
        if(actual==1):
            if(actual==Y_pred[i]):
                tp+=1
            else:
                fn+=1
        else:
            if(actual==Y_pred[i]):
                tn+=1
            else:
                fp+=1
    print("\naccuracy= ",(tp+tn)/(tp+tn+fn+fp))
    print("\nsensitivity= ",tp/(tp+fn))
    print("\nspecificity= ",tn/(fp+tn))
    print("\nPPv= ",tp/(tp+fp))
    print("\nNPV= ",tn/(tn+fn))



evaluateParameters(Y_pred,Y_test)
# saving trained model, to deploy on web
import pickle
with open('lr_model.pkl','wb') as f:
    pickle.dump(model,f)
```

Random Forest Classifier

```
import pandas as pd
import numpy as np
df=pd.read_csv("framingham.csv")
display(df)
# Preprocessing Data
# "education" showed -ve correlation, with target label
# "male" was shown to be IN SIGNIFICANT, during T TEst(based on p-value)
# "diabetes" was shown to be IN SIGNIFICANT, during T TEst(based on p-value)
# "currentSmoker" fails in chi-square test
df=df.drop(axis=1,labels=["male","education","diabetes","currentSmoker"])
df
#To understand , which attribute has how many ("NULL" or NaN) values
df.info()
# each column has 4238 entries.
# for attribute "glucose" , there are only 3850 "not-NULL" entries
#df.fillna?

# for categorical attributes, we replace Nan with most frequent attribute
#for continous attributes,we replace Nan with mean .

#for continous attributes--->replace Nan with mean
df["cigsPerDay"]=df["cigsPerDay"].fillna(round(df["cigsPerDay"].mean()))
df["totChol"]=df["totChol"].fillna(round(df["totChol"].mean()))
df["BMI"]=df["BMI"].fillna(round(df["BMI"].mean(),2))
df["heartRate"]=df["heartRate"].fillna(round(df["heartRate"].mean()))
df["glucose"]=df["glucose"].fillna(round(df["glucose"].mean()))
```

```
#for categorical attributes-->replace Nan with most freq value
vall,freqq=np.unique(df["BPMeds"],return_counts=True)
#display(vall,freqq)
df["BPMeds"]=df["BPMeds"].fillna(vall[freqq.argmax()])

display(df)



df.info()
#now all attributes, has all 4238 values present
#how to access a row, in a DataFrame object
#2nd row of df
df.loc[2]
df.columns

# Decison Tree Implementation
### step 0: preparing data
# random shuffling of data
#df=df.sample(frac=1)
#display(df)

#not doing this step, as data is already shuffled, as per dataset discription
# taking data , from the DataFrame object, to numpy object
data=df.values
print(data)
print(data.shape)
print(type(data))
```

```
#separating target label, from rest of the attributes
X=data[:,:-1]
Y=data[:,-1]  # target label
print(X)
print(X.shape)
print(Y)
print(Y.shape)
### step 1: spliting data for training and testing
#performing split , to train,test data
split=0.7*X.shape[0]
print(split)
split=int(split)
print("split length:",split)
print()

X_train= X[:split,:]
Y_train=Y[:split]
X_test=X[split:,:]
Y_test=Y[split:]

print("X_train:",X_train.shape)
print("X_test:",X_test.shape)
print("Y_train:",Y_train.shape)
print("Y_test:",Y_test.shape)
#separating target label, from rest of the attributes
# we will use only dataframe objects, in all our calculations, not np arrays
df_x_train=pd.DataFrame(data=X_train,columns=['age', 'cigsPerDay', 'BPMeds',
'prevalentStroke', 'prevalentHyp',
      'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose'])
```

```
df_y_train=pd.DataFrame(data=Y_train,columns=["POSSIBILITY OF CVD"])  # target label
#now creating a data frame for training, testing
df_xy_train=pd.DataFrame(df[:split])
df_xy_test=pd.DataFrame(df[split:])
#make the index of df_xy_test to start from 0
df_xy_test=df_xy_test.reset_index(drop=True)

display(df_x_train)
print(df_x_train.shape)
display(df_y_train)
print(df_y_train.shape)
print(type(df_x_train))
display(df_xy_train)
print(df_xy_train.shape)
display(df_xy_test)
print(df_xy_test.shape)

### step 2: Decision Tree implementation
#define entropy of a column
# entropy is a measure of "randomness"
def entropy(columnn):
    classes,freq=np.unique(columnn,return_counts=True)
    N=float(columnn.shape[0])

    ent=0.0
    for i in freq:
        p=i/N
        ent+=(-1.0*p*np.log2(p))
```

```
    return ent

# understanding how the func works with a example
eg=np.array([1,1,1,0,0,0])
print(eg.shape)
print(np.unique(eg,return_counts=True))
print("entropy =",entropy(eg))
print("entropy is max(entropy=1) when probability=0.5 for each class in a binary data")
def divide_data(x_data,fkey,fval):
    """
    x_data: undivided data
    fkey: attribute ,whose values are to be considered for division
    fval: all rows, whose fkey>fval, copy to right node, otherwise left node
    """
    #to store all rows, whose value at fkey<=fval
    x_left=pd.DataFrame(data=[],columns=x_data.columns)
    #to store all rows, whose value at fkey>fval
    x_right=pd.DataFrame(data=[],columns=x_data.columns)

    for i in range(x_data.shape[0]):
        val=x_data[fkey].loc[i]
        if(val>fval):
            x_right=x_right.append(x_data.loc[i])
        else:
            x_left=x_left.append(x_data.loc[i])
    return x_left,x_right
```

```
#understanding how the func works
x_left,x_right=divide_data(df_x_train[:5],'age',50)
#0th column is age in X_train
print("undivided data=\n")
display(df_x_train[:5])
print("x_left(age<=50)=\n")
display(x_left)
print("x_right(age>50)=\n")
display(x_right)


# aim is to select a feature for tree branching, ...which gives the highest info gain, ie..after branching entropy or randomness is the lowest
def information_gain(xy_data,fkey,fval):
    left,right=divide_data(xy_data,fkey,fval)


    # % of total samples that are on left and right
    l=float(left.shape[0]/xy_data.shape[0])
    r=float(right.shape[0])/xy_data.shape[0]


    #if all rows come to one side
    if(l==0 or r==0):
        return -10000 #min info gain ie -ve value
    i_gain=entropy(xy_data["POSSIBILITY OF CVD"])-(l*entropy(left["POSSIBILITY OF CVD"])  +  r*entropy(right["POSSIBILITY OF CVD"]))
    return i_gain



#testing the above function with df_xy_train[[[]]]
for attribute in df_x_train.columns:
```

```
    print(attribute)
    print(information_gain(df_xy_train,attribute,df_xy_train[attribute].mean()))
class DecisionTree:
    def __init__(self,current_depth=0,max_depth=5):
        self.left=None
        self.right=None
        self.fkey=None
        self.fval=None
        self.max_depth=max_depth
        self.current_depth=current_depth
        #at leaf node, we will do majority vote, to predict the class, thus we need this "target"
variable
        self.target=None
    def train(self,df_xy_train):
        features=['age', 'cigsPerDay', 'BPMeds', 'prevalentStroke', 'prevalentHyp',
       'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose']
        info_gains=[] #to store info_gain of all above attributes

        for i in features:
            i_gain=information_gain(df_xy_train,i,df_xy_train[i].mean())
            info_gains.append(i_gain)

        self.fkey=features[np.argmax(info_gains)] #the attribute with highest info_gain
        self.fval=df_xy_train[self.fkey].mean() #mean of above attribute, in current data
        print("dividing data based on attribute: ",self.fkey)

        #split data
        df_xy_left,df_xy_right=divide_data(df_xy_train,self.fkey,self.fval)
        df_xy_left= df_xy_left.reset_index(drop=True)
```

```
df_xy_right= df_xy_right.reset_index(drop=True)

#below we discuss 2 stoping conditions
#1) if a leaf node
if(df_xy_left.shape[0]==0 or df_xy_right.shape[0]==0):
    if(df_xy_train["POSSIBILITY OF CVD"].mean()>=0.5):
        self.target=1
    else:
        self.target=0
    return
#2)  if current_depth>max_depth
if(self.current_depth>=self.max_depth):
    if(df_xy_train["POSSIBILITY OF CVD"].mean()>=0.5):
        self.target=1
    else:
        self.target=0
    return

#recursive case
self.left=DecisionTree(current_depth=self.current_depth+1)
self.left.train(df_xy_left)

self.right=DecisionTree(current_depth=self.current_depth+1)
self.right.train(df_xy_right)

#setting the target value, for non-leaf nodes
#this will help us predict a class, for data at a non-leaf node
if(df_xy_train["POSSIBILITY OF CVD"].mean()>=0.5):
```

```
            self.target=1
        else:
            self.target=0
        return
    def predict(self,queryRow):
        if(queryRow[self.fkey]>self.fval):
            #go to right
            if(self.right is None):
                return self.target
            return self.right.predict(queryRow)
        else:
            #go to left
            if(self.left is None):
                return self.target
            return self.left.predict(queryRow)
#df_xy_train is my training data which contains all attributes including "POSSIBILITY OF CVD"
#df_xy_test is my testing data which contains all attributes including "POSSIBILITY OF CVD"
d=DecisionTree()


d.train(df_xy_train)
print("root node dividing attribute:",d.fkey)
print(d.fval)
print("depth 1 ,@left, dividing attribute:",d.left.fkey)
print("depth 1 ,@right, dividing attribute:",d.right.fkey)
### step 3: testing on ...testing data
y_pred=[]
for i in range(df_xy_test.shape[0]):
    y_pred.append(d.predict(df_xy_test.loc[i]))
```

```
y_pred=np.array(y_pred)
#print(y_pred)
# Y_test is the actual predtion of the testing set
#print(Y_test)
Y_test=Y_test.astype(int) # as Y_test has float elements
#print(Y_test)
print(type(Y_test))
print(type(y_pred))
noOfCorrectPred=np.sum((y_pred==Y_test))
acc=noOfCorrectPred/Y_test.shape[0]
print("accuracy on test data= ",acc*100," %")

# Decision Tree using sk-learn
from sklearn.tree import DecisionTreeClassifier
sk_tree=DecisionTreeClassifier?
sk_tree=DecisionTreeClassifier(criterion='entropy',max_depth=5)
# to avoid overfitting, we will keep "max depth=5"
sk_tree.fit(X_train,Y_train)
sk_tree.predict(X_test)
### accuracy when no 'overfitting'(as max_depth is restricted to 5)
acc1=sk_tree.score(X_train,Y_train)*100
print("accuracy on test data= ",acc1," %")
acc=sk_tree.score(X_test,Y_test)*100
print("accuracy on test data= ",acc," %")
from sklearn import tree
text_representation=tree.export_text(sk_tree)
print(text_representation)
```

```
import matplotlib.pyplot as plt

tree.plot_tree?

fig=plt.figure(figsize=(55,40))

_=tree.plot_tree(sk_tree,feature_names=['age', 'cigsPerDay', 'BPMeds', 'prevalentStroke',
'prevalentHyp',

      'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose'],class_names=["no CVD","have
CVD"],filled=True)


### accuracy when 'Over-fitting' exists(no max_depth restriction)

# no restriction on "max_depth"

sk_tree=DecisionTreeClassifier(criterion='entropy')


sk_tree.fit(X_train,Y_train)

acc1=sk_tree.score(X_train,Y_train)*100

print("accuracy on test data= ",acc1," %")

acc=sk_tree.score(X_test,Y_test)*100

print("accuracy on test data= ",acc," %")

## when over-fiting exists, testing accuracy is very small, compared to training accuracy



# Random Forest using SK-learn

from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(n_estimators=10,criterion='entropy',max_depth=5)

rf.fit(X_train,Y_train)

rf.score(X_train,Y_train)

rf.score(X_test,Y_test)

#testing accuracy is higher than training accuracy


# geting Accuracy,   using 'cross-validation'
```

```
from sklearn.model_selection import cross_val_score


#cross validation=cv=5

# dividing training set into 5 parts, using any 4 parts for training, then testing on the 5th part

#all combinations of 4 parts will be taken, and tested on the 5th

acc=cross_val_score(RandomForestClassifier(n_estimators=10,criterion='entropy',max_depth=5
),X_train,Y_train,cv=5  )

acc

#we take mean of these

acc=np.mean(acc)

print("cross validation accuracy\nOf random forest classifier = ",acc*100)

# Hyper Tuning the "no. of tress" used in Random Forest Classifier

### n_estimator=no of trees

acc_list=[]

for i in range(1,50):


acc=cross_val_score(RandomForestClassifier(n_estimators=i,criterion='entropy',max_depth=5),
X_train,Y_train,cv=5  ).mean()

    acc_list.append(acc)

print(acc_list)


plt.style.use("seaborn")

plt.plot(acc_list)


print("max accracy occurs at n_estimator=",np.argmax(acc_list))

print("max cross validation accuracy with\nrandom forest classifier on training
data=",acc_list[37]*100," %")

## finding score on test data , with n_estimator=37

rf=RandomForestClassifier(n_estimators=37,criterion='entropy',max_depth=5)

rf.fit(X_train,Y_train)
```

```
rf.score(X_test,Y_test)*100
Y_rf_pred=rf.predict?
Y_rf_pred=rf.predict(X_test)
###  measuring performance by evaluating parameters
def evaluateParameters(Y_pred,Y_test):
    tp,tn,fp,fn=0,0,0,0
    for i in range(len(Y_test)):
        actual=Y_test[i]
        if(actual==1):
            if(actual==Y_pred[i]):
                tp+=1
            else:
                fn+=1
        else:
            if(actual==Y_pred[i]):
                tn+=1
            else:
                fp+=1
    print("\naccuracy= ",(tp+tn)/(tp+tn+fn+fp))
    print("\nsensitivity= ",tp/(tp+fn))
    print("\nspecificity= ",tn/(fp+tn))
    print("\nPPv= ",tp/(tp+fp))
    print("\nNPV= ",tn/(tn+fn))
evaluateParameters(Y_rf_pred,Y_test)
# saving trained model, for deploying on web
import pickle
with open('rf_model.pkl','wb') as f:
    pickle.dump(rf,f)
```

Adaptive Boosting

```
a import pandas as pd
import numpy as np

df=pd.read_csv("framingham.csv")
display(df)
# Preprocessing Data
# "education" showed -ve correlation, with target label
# "male" was shown to be IN SIGNIFICANT, during T TEst(based on p-value)
# "diabetes" was shown to be IN SIGNIFICANT, during T TEst(based on p-value)
# "currentSmoker" fails in chi-square test
df=df.drop(axis=1,labels=["male","education","diabetes","currentSmoker"])
df
#To understand , which attribute has how many ("NULL" or NaN) values
df.info()
# each column has 4238 entries.
# for attribute "glucose" , there are only 3850 "not-NULL" entries


# for categorical attributes, we replace Nan with most frequent attribute
#for continous attributes,we replace Nan with mean .

#for continous attributes--->replace Nan with mean
df["cigsPerDay"]=df["cigsPerDay"].fillna(round(df["cigsPerDay"].mean()))
df["totChol"]=df["totChol"].fillna(round(df["totChol"].mean()))
df["BMI"]=df["BMI"].fillna(round(df["BMI"].mean(),2))
df["heartRate"]=df["heartRate"].fillna(round(df["heartRate"].mean()))
df["glucose"]=df["glucose"].fillna(round(df["glucose"].mean()))
```

```
#for categorical attributes-->replace Nan with most freq value
vall,freqq=np.unique(df["BPMeds"],return_counts=True)
#display(vall,freqq)
df["BPMeds"]=df["BPMeds"].fillna(vall[freqq.argmax()])

display(df)



df.info()
#now all attributes, has all 4238 values present
### step 0: Preparing Data
# random shuffling of data
#df=df.sample(frac=1)
#display(df)

#not doing this step, as data is already shuffled, as per dataset discription
# taking data , from the DataFrame object, to numpy object
data=df.values
print(data)
print(data.shape)
print(type(data))
#separating target label, from rest of the attributes
X=data[:,:-1]
Y=data[:,-1]  # target label
print(X)
print(X.shape)
```

```
print(Y)
print(Y.shape)
### step 1: spliting training and testng data
#performing split , to train,test data
split=0.7*X.shape[0]
print(split)
split=int(split)
print("split length:",split)
print()

X_train= X[:split,:]
Y_train=Y[:split]
X_test=X[split:,:]
Y_test=Y[split:]
print("X_train:",X_train.shape)
print("X_test:",X_test.shape)
print("Y_train:",Y_train.shape)
print("Y_test:",Y_test.shape)
# step 2: Implementation of Adaboost
## weak classifier= stums(Decision Tree of depth=1)
## bosting technique=adaptive boosting
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
abc=AdaBoostClassifier(
    base_estimator=DecisionTreeClassifier(criterion='entropy',max_depth=1),
    n_estimators=96,
    learning_rate=1.2,
    random_state=96)
```

```
abc
abc.fit(X_train,Y_train)
abc.score(X_train,Y_train)
### step 3 :testing on test set
abc.score(X_test,Y_test)
### step 4: Hyper Parameter Tuning
testing_acc=[]
for i in range(60,500):
    abc1=AdaBoostClassifier(
    base_estimator=DecisionTreeClassifier(criterion='entropy',max_depth=1),
    n_estimators=i,
    learning_rate=1.2,
    random_state=96)

    abc1.fit(X_train,Y_train)

    testing_acc.append(abc1.score(X_test,Y_test))

import matplotlib.pyplot as plt
plt.plot(testing_acc)

testing_acc=np.array(testing_acc)
index=testing_acc.argmax()
index
print("max possible accuracy=",testing_acc[160]*100," %")
### step 5: evaluate performance using various parameters
Y_pred=abc.predict(X_test)
```

```
def evaluateParameters(Y_pred,Y_test):
    tp,tn,fp,fn=0,0,0,0
    for i in range(len(Y_test)):
        actual=Y_test[i]
        if(actual==1):
            if(actual==Y_pred[i]):
                tp+=1
            else:
                fn+=1
        else:
            if(actual==Y_pred[i]):
                tn+=1
            else:
                fp+=1
    print("\naccuracy= ",(tp+tn)/(tp+tn+fn+fp))
    print("\nsensitivity= ",tp/(tp+fn))
    print("\nspecificity= ",tn/(fp+tn))
    print("\nPPv= ",tp/(tp+fp))
    print("\nNPV= ",tn/(tn+fn))
evaluateParameters(Y_pred,Y_test)
# saving trained model, for deploying on web
import pickle
with open('ab_model.pkl','wb') as f:
    pickle.dump(abc,f)
```

CODE FOR DEPLOYMENT(USING FLASK FRAMEWORK):

Index.html

```
<!DOCTYPE html>
<html>
    <body bgcolor=#d4a3af>
        <center>
            <h1>CVD Detection Portal</h1>
            <br>

            <form action="{{url_for('predict')}}",method="POST">
                <b>


                    age                      :  <input type="number",
name='b',value='55' placeholder="example 55"  min="32"  required> <br><br>


                    cigs per day             :  <input type="number",
name='e',value='26', placeholder="example 26"  min='0' required> <br><br>
                    taking BP medication     :  <input type="number",
name='f',value='1', placeholder="1 or 0"  min='0' max='1' style="width: 11em"
required> <br><br>
                    history of Stroke        :  <input type="number",
name='g',value='1', placeholder="1 or 0"  min='0' max='1' style="width:
11em"  required> <br><br>
                    history of hyper tension : <input type="number", name='h',
value='1', placeholder="1 or 0"  min='0' max='1' style="width: 11em" required>
<br><br>



                    total cholestrol(avg)    :  <input type="number",
name='j',value='250', placeholder="example 250"  min='113'  required> <br><br>
                    avg systolic BP          :  <input type="number",
name='k',value='121', placeholder="example 121"  min='65'  required> <br><br>
                    avg diastolic BP         :  <input type="number",
name='l',value='81', placeholder="example 81"  min='40' required> <br><br>
                    BMI                      : <input type="number",
name='m',value='22.91', placeholder="example 22.91" min='15'  required> <br><br>
                    avg heartRate            :  <input type="number", name='n',
value='88',placeholder="example 88"  min='44'  required> <br><br>
                    avg glucose count        :  <input type="number",
name='o',value='86', placeholder="40 to 400"  min='40'  required> <br><br>
```

```
                </b>
                <input type="submit" , class="btn btn-primary btn-block btn-
large", value=" predict!! ">
                <br>

            </form>
            <br>
            <br>

            <b><h3>created by:</h3><br>
            DARSHAN KUMAR<br>
            B Tech CSE-2022 batch<br>
            as a part for 7th Semester : MINI-PROJECT(2021)<br>
            for :- sastra deemed to be university<br></b>


        </center>
    </body>
</html>
```

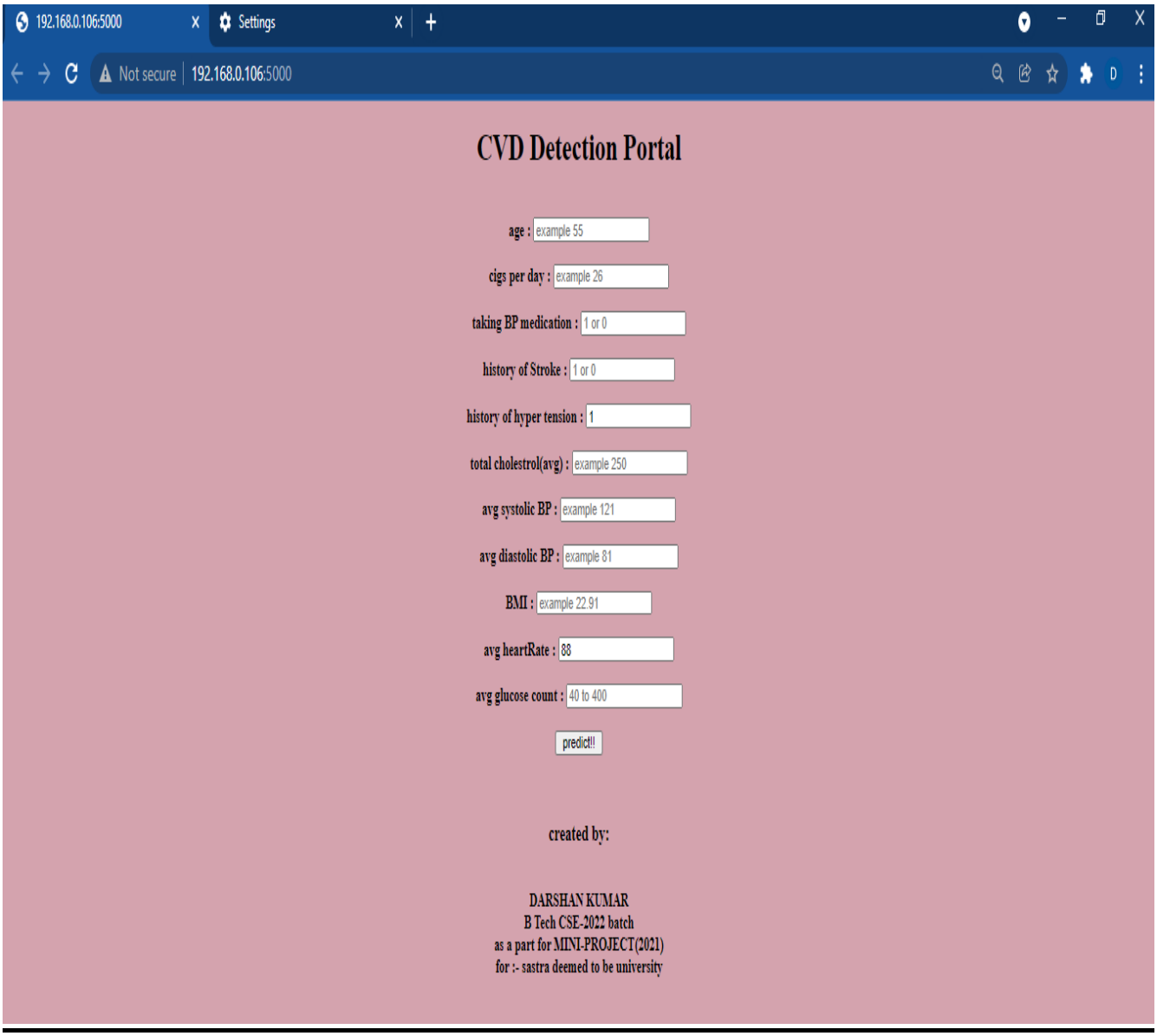
app.py

```
from flask import Flask,render_template,request
import joblib
import numpy as np
import os

port = int(os.environ.get('PORT', 5000))

model_knn=joblib.load('knn_model.pkl')
model_nbc=joblib.load('nbc_model.pkl')
model_lr=joblib.load('lr_model.pkl')
model_rf=joblib.load('rf_model.pkl')
model_ab=joblib.load('ab_model.pkl')

x_mean=np.array([4.95849457e+01, 9.00306748e+00, 2.92590845e-02 ,5.89900897e-03,
3.10523832e-01 ,2.36724870e+02 ,1.32352407e+02 ,8.28934639e+01, 2.58019986e+01
,7.58789523e+01 ,8.19697971e+01])
x_std=np.array([8.57114852, 11.87782862 , 0.16853187  ,0.07657813 , 0.4627081 ,
44.32123288, 22.03549643, 11.90944427 , 4.07047223, 12.02375836 ,22.83391071])



app=Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict',methods=["POST","GET"])
def predict():
    #d1=request.values.get("b")
    names=['b','e','f','g','h','j','k','l','m','n','o']
    #print(d1,flush=True)
    int_features=[int(request.values.get(x)) for x in names]

    final_features=[np.array(int_features)]
    #print(final_features,flush=True)

    pred_knn=model_knn.predict(final_features)
    pred_nbc=model_nbc.predict(final_features)
    pred_rf=model_rf.predict(final_features)
    pred_ab=model_ab.predict(final_features)

    #normalising input, before passing it to lr_model
```

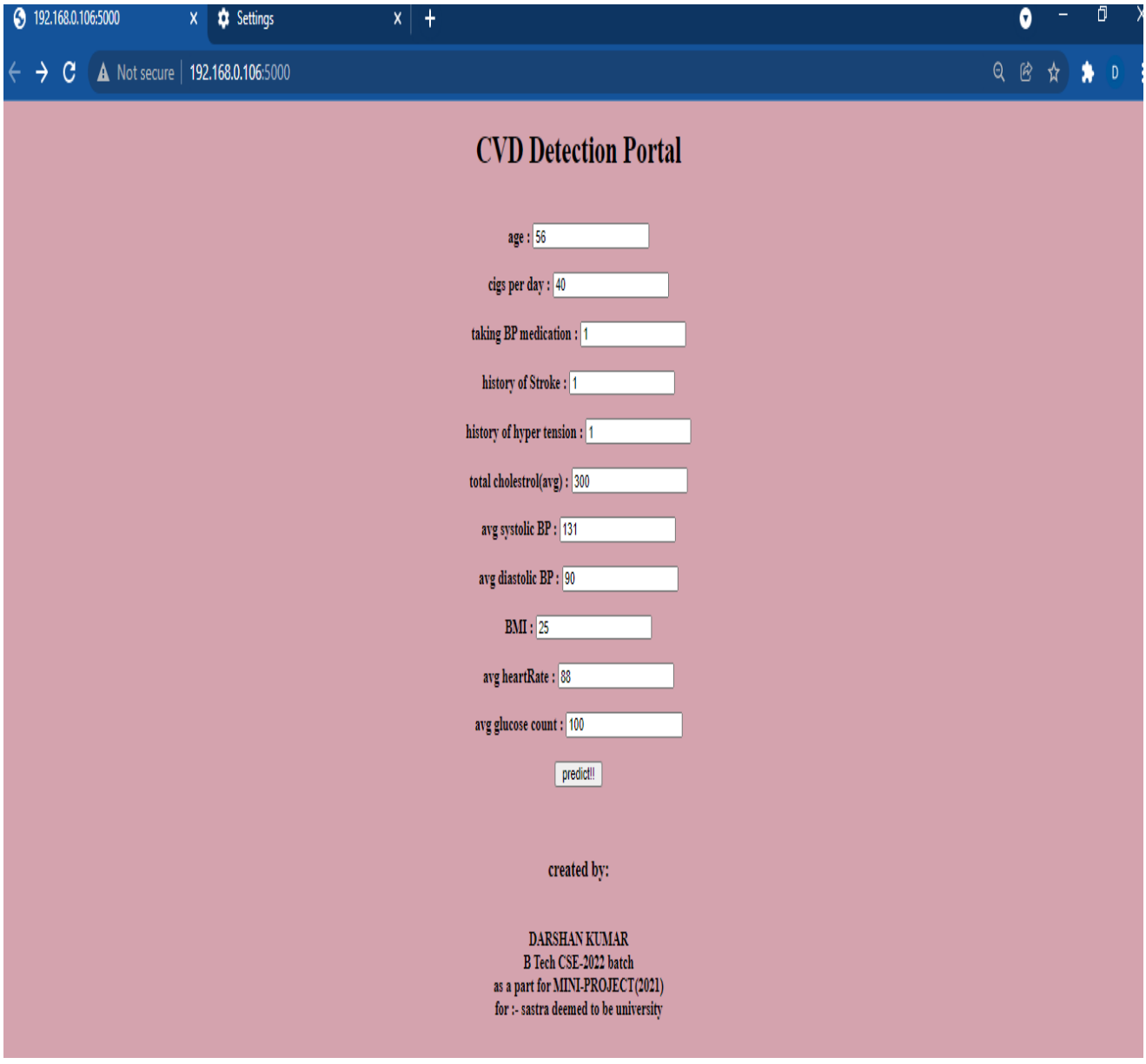
```
    np_final_features=np.array(final_features)
    normalised_np_final_features=(np_final_features-x_mean)/x_std
    pred_lr=model_lr.predict(normalised_np_final_features)

    #print(pred_knn,"prediction@@@@@@",flush=True)


    return  ('<h1> Prediction Results:</h1><br><h3>0:no sign of CVD<br>
1:significant possibility of CVD</h3><br>knn model: '+ str(pred_knn[0])+'<br> nbc
model: '+str(pred_nbc[0])+'<br> lr model: '+str(pred_lr[0])+'<br> rf model:
'+str(pred_rf[0])+'<br> ab model: '+str(pred_ab[0])+"<br><br>final result can be
obatained from <br> 1)majority voting of above,or <br>2)weighted avg-giving
higher weightage to models with higher accuracy")

if(__name__=="__main__"):
    app.run(host='0.0.0.0', port=port)
```

Screenshots of Output

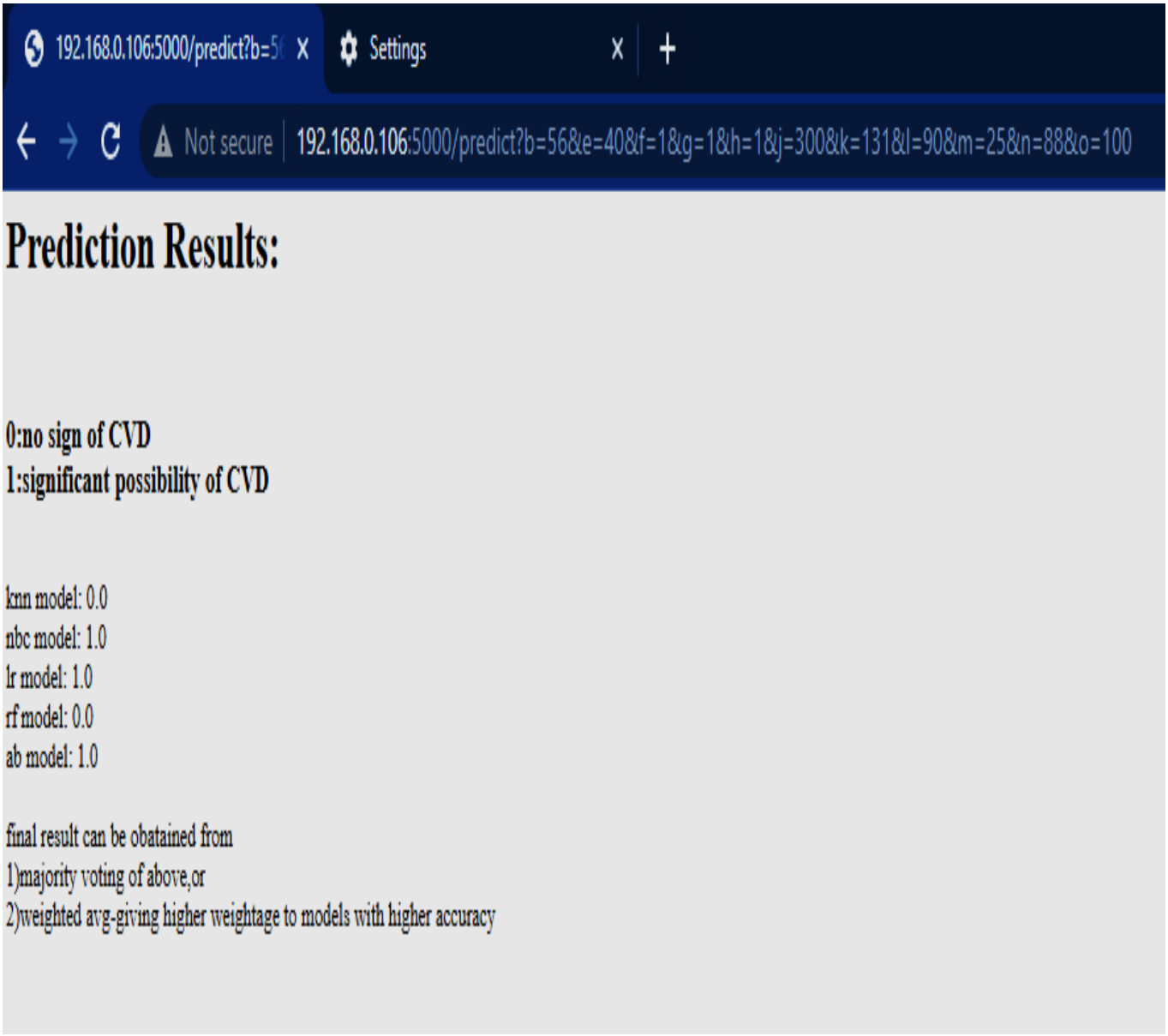
Step 1: Opening the Web Page



Step 2: Entering Data(max , min limit of each value, is set in html code)



Step 3: After Entering Correct, values, click "Predict" , to see the Prediction, of Each Model



CONCLUSIONS AND FUTURE PLANS

TABLE NO 1:

Explanation Of All the algorithms used

ALGORITHM	Explanation
K nearest neighbor	No training is involved, no parameters are learnt, final result is based on "**majority vote**".
Naïve Bayes Classifier	**Extended bayes theorem** is used, to find conditional probability wrt each class. The one, with higher probability wins. **Laplace smoothing** is used, to deal with 0 probability cases.
Logistic Regression	**Gradient descent is** used to learn the parameters. **Sigmoid function** is used to reduce the values to 0,1.
Random Forest (base learner: decision trees)	each learners , become specialized in the portion of the data set they receive(for training).final result is based on "**majority vote**" . It is the best choice, to avoid over-fitting.
Adaptive Boosting (weak learner: decision trees)	Sequential learners are used Only those records , which are wrongly classified by 1st learner, are used to train, 2nd learner.

TABLE 2:

Comparing accuracy of the algorithms used

Model	Accuracy on Test Data
K-Nearest Neighbor	0.8490566037735849
Naïve Bayes Classifier	0.8309748427672956
Logistic Regression	0.8545597484276729
Random Forest Classifier	0.8522012578616353
Adaptive Boosting Technique	0.8364779874213837

SUMMARY OF PERFORMANCE OF EACH ALGORITHM

KNN

- **accuracy= 0.8490566037735849**
- sensitivity= 0.047619047619047616
- specificity= 0.9889196675900277
- PPV= 0.42857142857142855
- NPV= 0.8561151079136691

Naïve Bayes

- **accuracy= 0.8309748427672956**
- sensitivity= 0.23809523809523808
- specificity= 0.9344413665743305
- PPV= 0.3879310344827586
- NPV= 0.8754325259515571

Logistic Regression

- **accuracy= 0.8545597484276729**
- sensitivity= 0.06349206349206349
- specificity= 0.9926131117266851
- PPV= 0.6
- NPV= 0.8586261980830671

Random Forest

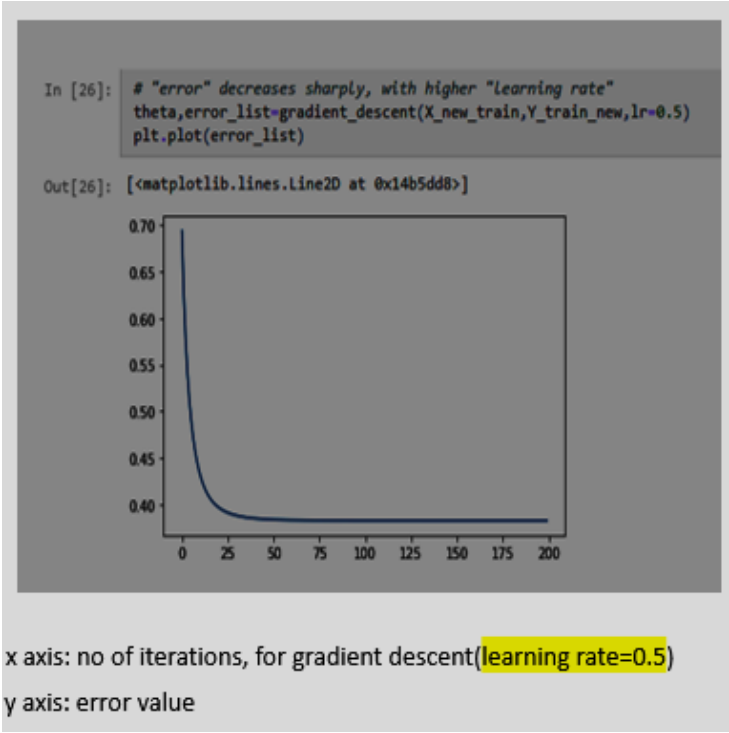
- **accuracy= 0.8522012578616353**
- sensitivity= 0.015873015873015872
- specificity= 0.9981532779316713
- PPV= 0.6
- NPV= 0.8531965272296764

Adaptive Boosting

- **accuracy= 0.8364779874213837**
- sensitivity= 0.1111111111111111
- specificity= 0.9630655586334257
- PPV= 0.3442622950819672
- NPV= 0.861271676300578

Graphs involved, during Training, Hyper tuning:

Logistic Regression(hyper Parameters: Learning Rate**):**



x axis: no of iterations, for gradient descent(learning rate=0.5)

y axis: error value

Random Forest(hyper Parameters: no of decision trees, max depth, entropy/gini index**):**

(“max_depth”=5, is used in all implementations, due to intensive, calculations , involved, with larger depth)

x axis: no of decision trees used

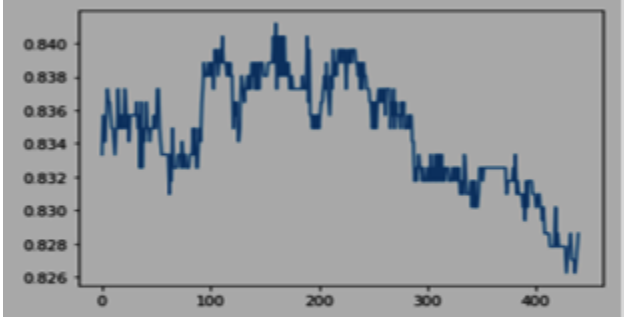
y axis: accuracy of model

Adaptive Boosting(hyper Parameters: no of estimators, learning rate)**:**

```
import matplotlib.pyplot as plt
```

```
plt.plot(testing_acc)
```

```
[<matplotlib.lines.Line2D at 0x1c5b7f28>]
```




```
testing_acc=np.array(testing_acc)
index=testing_acc.argmax()
index
```

```
160
```

x axis: no of weak learners(decision trees, with depth=1) used

y axis: accuracy of model

Conclusions:

- "Old Age", history of "Hyper Tension", presence of "high BP" , were seen as the **most** contributing factors for CVD.
- Un-controlled blood "glucose" levels, is also found to be a risk factor, for CVD.
- "Males" were found to be more prevalent to face CVD.
- Raised "cholesterol" , was also found to be a warning, for CVD.

FUTURE PLANS/IMPROVEMENTS POSSIBLE:

- The accuracy of the models, depends on the quality of the data set used. If a dedicated , study could be held in the city hospitals, and thus, correctly recording the patient medical records, than high accuracy , for the models could be guaranteed.
- Neural Network(example RNN)(with more than 1 hidden layer),could be used, to achieve even higher accuracy. But their Training through, back Propagation, requires, more computational resources(depending on number of Hidden Layers, in network).
- Additional gates could be used in RNN, to forget, unnecessary attributes, and pass on, only selected attributes, which it feels, contributes more to CVD.(Like in LSTM).
- ANN(artificial Neural Network) are also expected to show, better performance.

.

REFERENCES

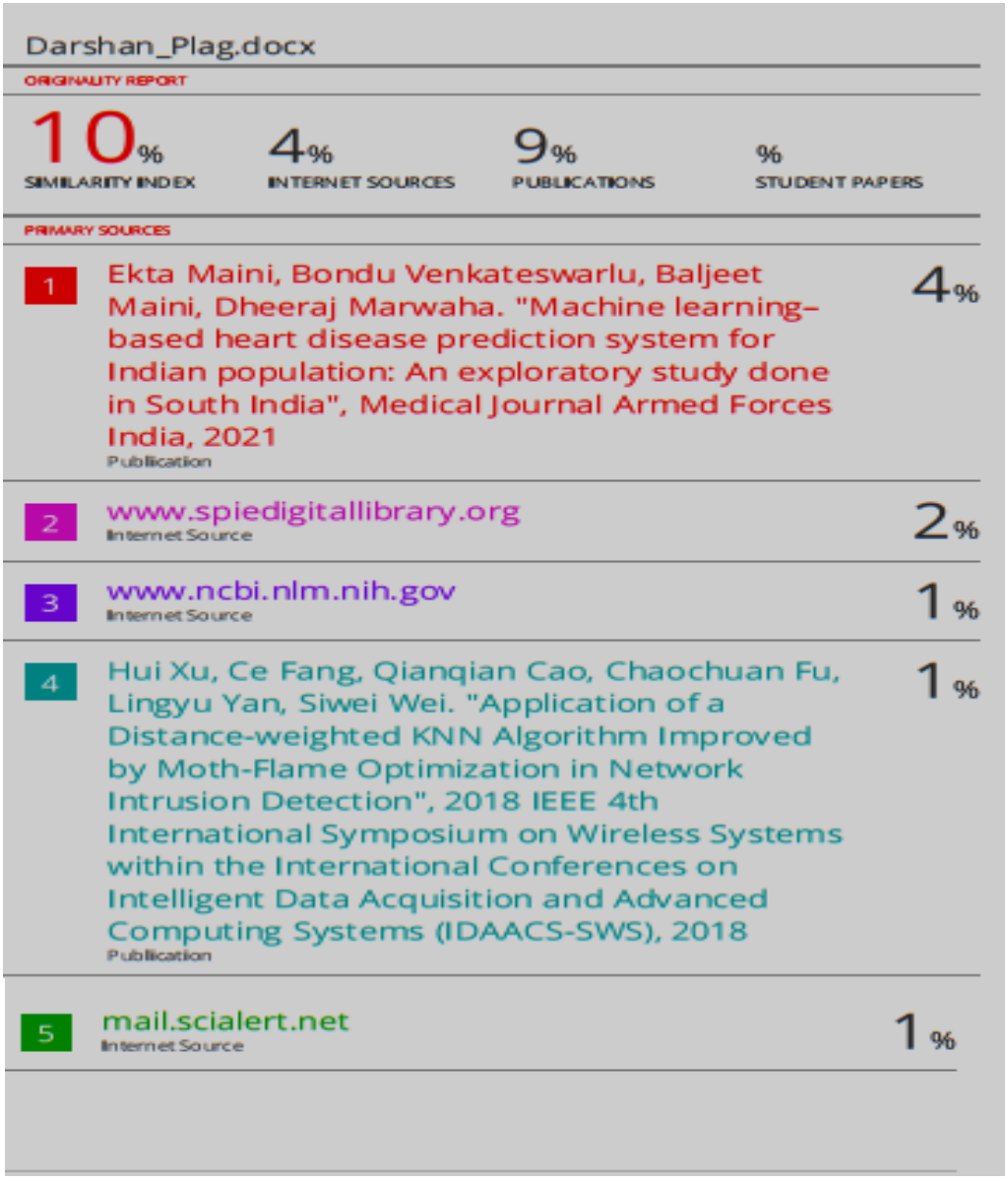
- official documentation of **SK-learn**, to understand, how the implementation **of t-test**, **chi square test** etc.
- official documentation of **Numpy** library, (to vectorize my code .(matrix multiplication ,element to element dot product)
- Tutorials of experts , in **YouTube**(to understand how, **models** can be efficiently implemented)

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer: 2006 (prescribed textbooks of MLT subject-for B Tech)

APPENDIX: BASE PAPER

Link: https://doi.org/10.1016/j.mjafi.2020.10.013

APPENDIX: PLAGIARISM REPORT



Darshan_Plag.docx

ORIGINALITY REPORT

10%	4%	9%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Ekta Maini, Bondu Venkateswarlu, Baljeet Maini, Dheeraj Marwaha. "Machine learning–based heart disease prediction system for Indian population: An exploratory study done in South India", Medical Journal Armed Forces India, 2021 Publication	4%
2	www.spiedigitallibrary.org Internet Source	2%
3	www.ncbi.nlm.nih.gov Internet Source	1%
4	Hui Xu, Ce Fang, Qianqian Cao, Chaochuan Fu, Lingyu Yan, Siwei Wei. "Application of a Distance-weighted KNN Algorithm Improved by Moth-Flame Optimization in Network Intrusion Detection", 2018 IEEE 4th International Symposium on Wireless Systems within the International Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS), 2018 Publication	1%
5	mail.scialert.net Internet Source	1%