CS590 - DATA MINING PROJECT

Dept. of Computer Science

Bishop's University

Sherbrooke, Canada

Topic- Predict disease classes using genetic microarray data

Submitted By:

Ishan Gulati (002295885)

Aayushi Pachorkar(002287964)

Professor:

Dr. Layachi Bentabet

OBJECTIVE

The goal is to develop a method that uses genetic data for disease classification from samples in the datasets which represent patients , to learn the best model from training data and use it to predict the label (class) for each sample in test data where for each patient 7070 genes expressions (values) are measured in order to classify the patient's disease into one of the following cases: EPD, JPA, MED, MGL, RHB.

INTRODUCTION

The DNA microarray technology captures gene expressions of thousands of genes simultaneously which results in enormous high dimension data with redundant and irrelevant genes which makes the analysis challenging. Therefore gene selection techniques like Machine Learning, Data Mining algorithms such as decision trees, support vector machines, multilayer perceptron, Bayes classifiers, K-Nearest Neighbors Ensemble classifier techniques, and so on are used can be used for accuracy in prediction of diseases.

TECHNIQUES USED FOR PREDICTING DISEASE

- **Problem Investigation:**

  It's very crucial to understand the objective of problem and understanding of final result in order to achieve goal.

- **Data Cleaning :**

  Enormous amount of data is captured which consists of various irrelevant gene data as a large number of statistical tests for finding disease classes results in the occurrence of many false discoveries among genes called differentially expressed. This problem can further manifest itself in the irreproducibility of results of different studies, Therefore cleaning the data and preparing data in order to better expose the structure of the prediction problem is most important step for analysis.

- **Analyzing Data:**

  For diagnosing the data we use descriptive statistics(this technique summarize characteristics of data set) and visualization for the getting more accurate and better understanding of the data.

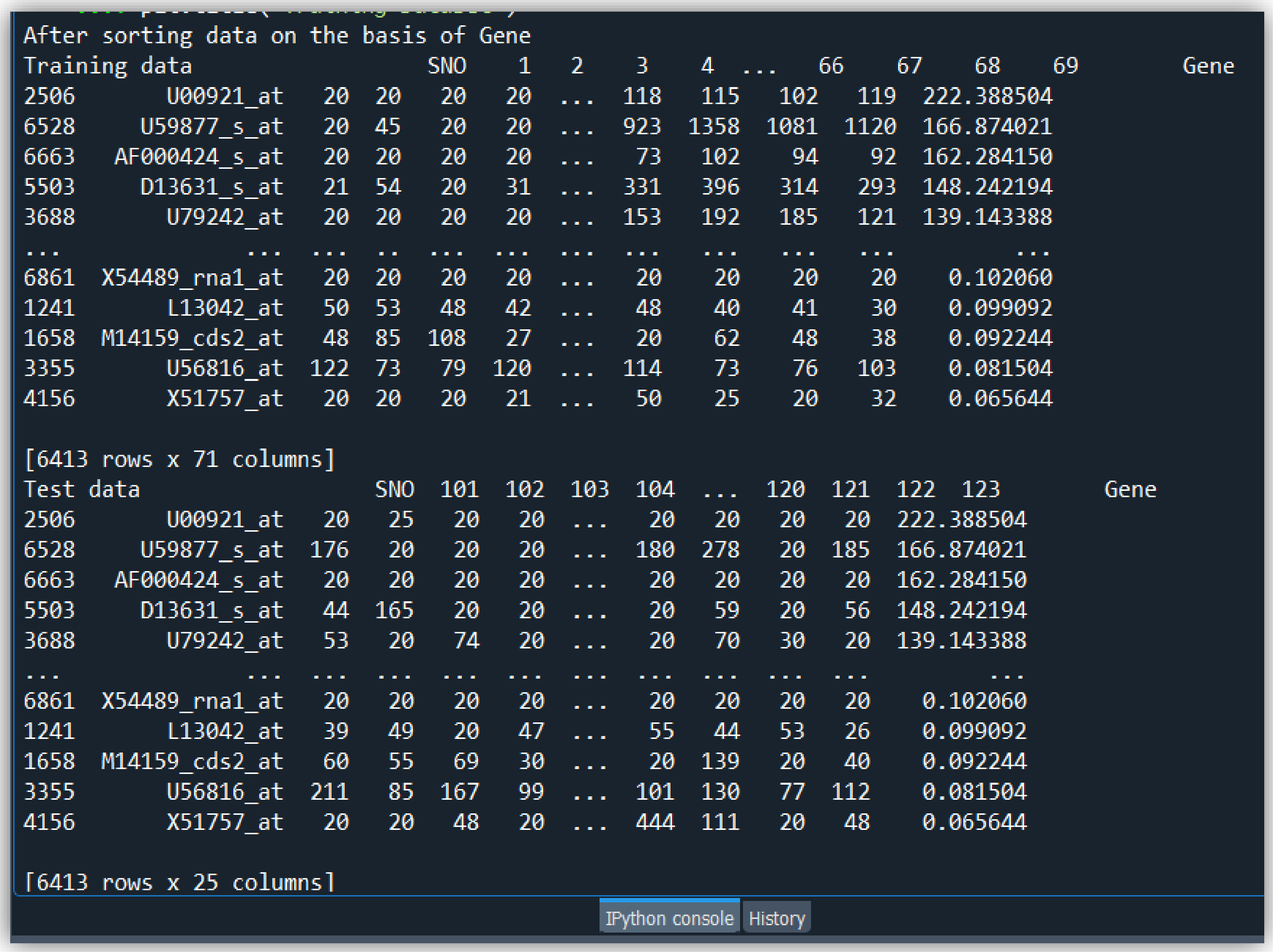
- **Determining Algorithm:**

  Testing different approaches of algorithms on selected data items and selecting the best few to examine further in order to get the most out of well-performing algorithms on the data.

APPROACH

STEP -1 DATA CLEANING

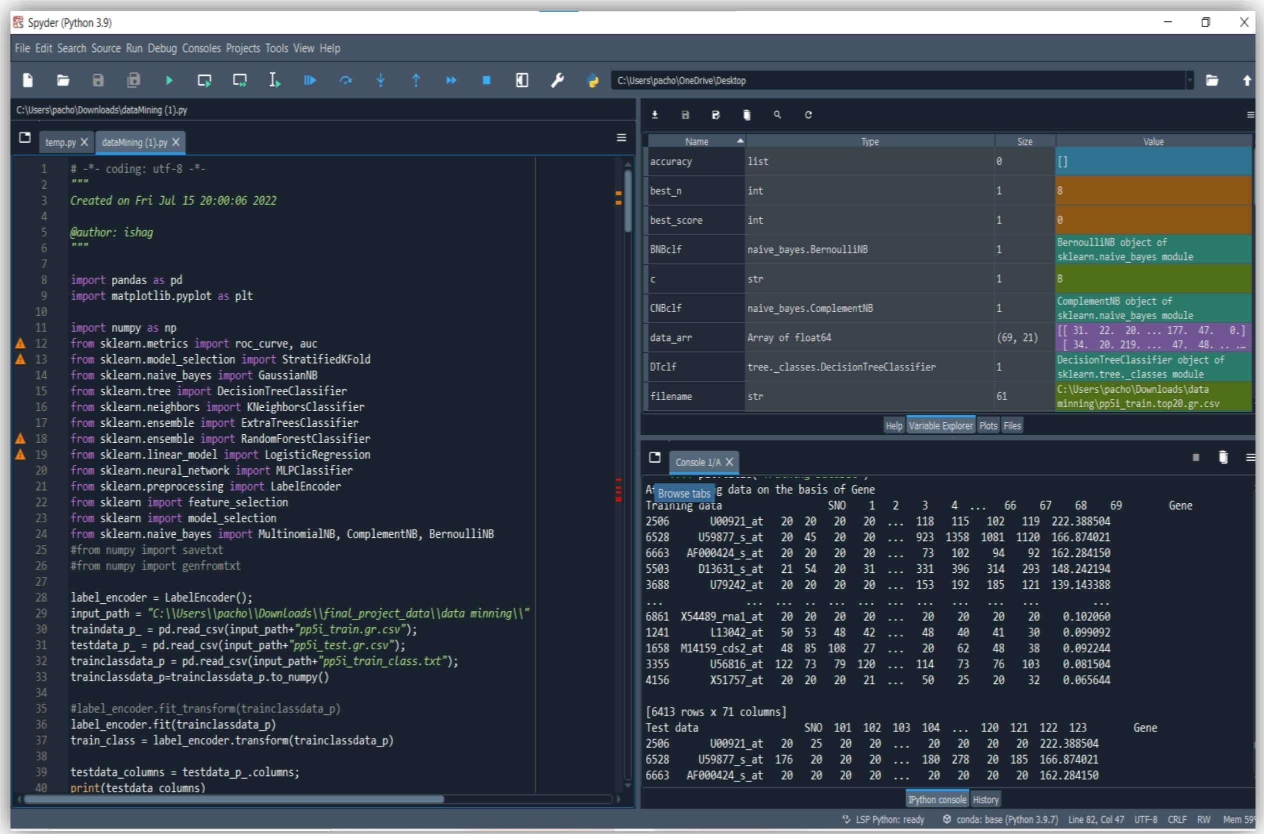
Firstly, we look at the data set carefully to remove any redundant or irrelevant information. In microarray experiments the number of analyzed samples is often much lower than the number of genes (probe sets) which leads to many false discoveries. Multiple testing correction methods control the number of false discoveries. Concerning this problem, filtering methods for improving the power of detection of differentially expressed genes.



```
After sorting data on the basis of Gene
Training data                SNO    1   2    3    4  ...   66    67    68    69        Gene
2506        U00921_at   20  20   20   20  ...  118   115   102   119  222.388504
6528      U59877_s_at   20  45   20   20  ...  923  1358  1081  1120  166.874021
6663    AF000424_s_at   20  20   20   20  ...   73   102    94    92  162.284150
5503      D13631_s_at   21  54   20   31  ...  331   396   314   293  148.242194
3688        U79242_at   20  20   20   20  ...  153   192   185   121  139.143388
...               ...  ...  ..  ...  ...  ...  ...   ...   ...   ...         ...
6861   X54489_rna1_at   20  20   20   20  ...   20    20    20    20    0.102060
1241        L13042_at   50  53   48   42  ...   48    40    41    30    0.099092
1658   M14159_cds2_at   48  85  108   27  ...   20    62    48    38    0.092244
3355        U56816_at  122  73   79  120  ...  114    73    76   103    0.081504
4156        X51757_at   20  20   20   21  ...   50    25    20    32    0.065644

[6413 rows x 71 columns]
Test data            SNO  101  102  103  104  ...  120  121  122  123        Gene
2506        U00921_at   20   25   20   20  ...   20   20   20   20  222.388504
6528      U59877_s_at  176   20   20   20  ...  180  278   20  185  166.874021
6663    AF000424_s_at   20   20   20   20  ...   20   20   20   20  162.284150
5503      D13631_s_at   44  165   20   20  ...   20   59   20   56  148.242194
3688        U79242_at   53   20   74   20  ...   20   70   30   20  139.143388
...               ...  ...  ...  ...  ...  ...  ...  ...  ...  ...         ...
6861   X54489_rna1_at   20   20   20   20  ...   20   20   20   20    0.102060
1241        L13042_at   39   49   20   47  ...   55   44   53   26    0.099092
1658   M14159_cds2_at   60   55   69   30  ...   20  139   20   40    0.092244
3355        U56816_at  211   85  167   99  ...  101  130   77  112    0.081504
4156        X51757_at   20   20   48   20  ...  444  111   20   48    0.065644

[6413 rows x 25 columns]
```

METHOD:

- We have set threshold for values where minimum value =20 and maximum value = 16000.
- To reduce margin error we have also rescale the data. The attributes are rescaled to a 0 to 1 scale.
- Log transformation is used on attributes with skewed distributions

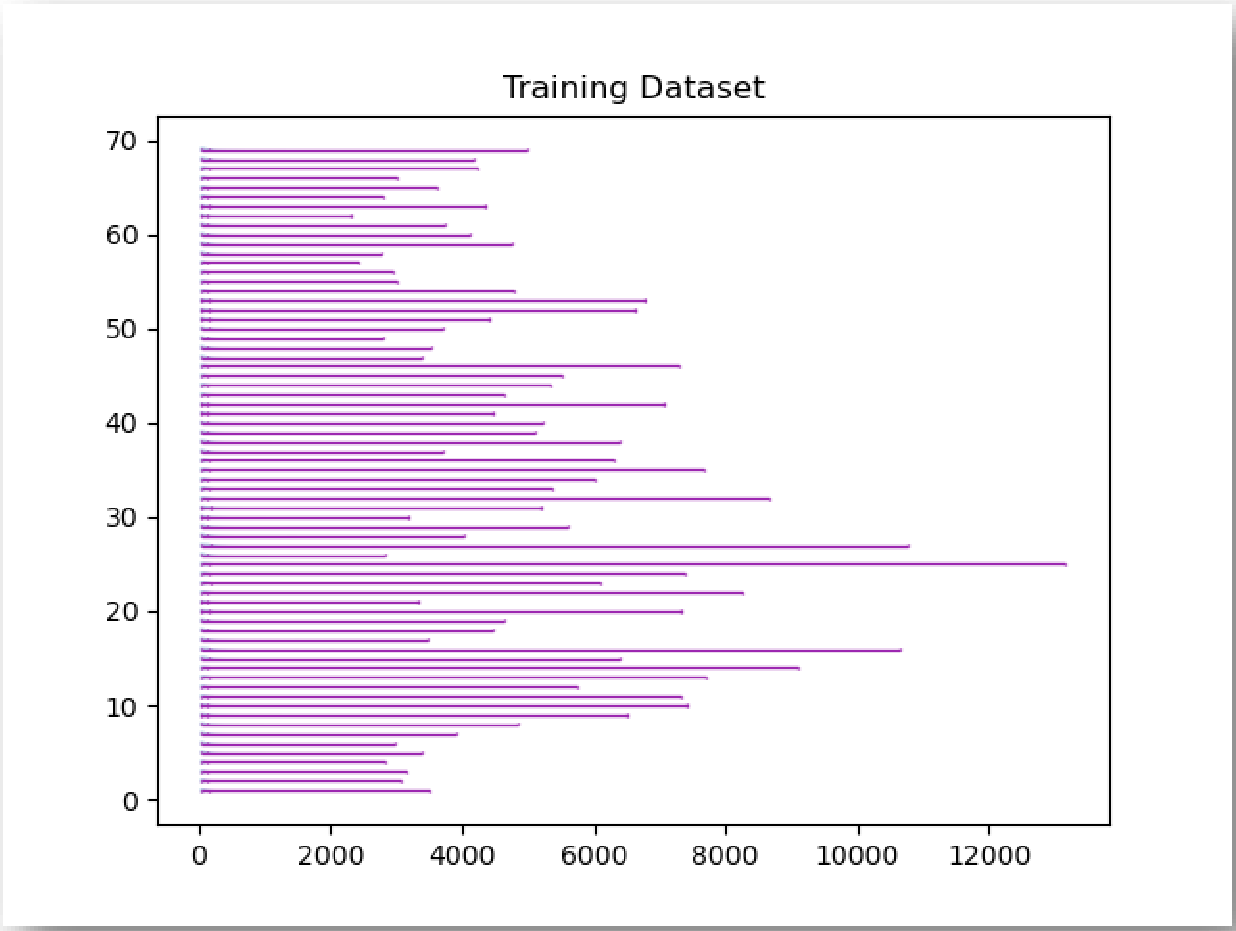


```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jul 15 20:00:06 2022

@author: ishag
"""

import pandas as pd
import matplotlib.pyplot as plt

import numpy as np
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn import feature_selection
from sklearn import model_selection
from sklearn.naive_bayes import MultinomialNB, ComplementNB, BernoulliNB
#from numpy import savetxt
#from numpy import genfromtxt

label_encoder = LabelEncoder();
input_path = "C:\\Users\\pacho\\Downloads\\final_project_data\\data minning\\"
traindata_p_ = pd.read_csv(input_path+"pp5i_train.gr.csv");
testdata_p_ = pd.read_csv(input_path+"pp5i_test.gr.csv");
trainclassdata_p = pd.read_csv(input_path+"pp5i_train_class.txt");
trainclassdata_p=trainclassdata_p.to_numpy()

#label_encoder.fit_transform(trainclassdata_p)
label_encoder.fit(trainclassdata_p)
train_class = label_encoder.transform(trainclassdata_p)

testdata_columns = testdata_p_.columns;
print(testdata columns)
```

Name	Type	Size	Value
accuracy	list	0	[]
best_n	int	1	8
best_score	int	1	0
BNBclf	naive_bayes.BernoulliNB	1	BernoulliNB object of sklearn.naive_bayes module
c	str	1	B
CNBclf	naive_bayes.ComplementNB	1	ComplementNB object of sklearn.naive_bayes module
data_arr	Array of float64	(69, 21)	[[31. 22. 20. ... 177. 47. 0.] [34. 20. 219. ... 47. 48. .. ...
DTclf	tree._classes.DecisionTreeClassifier	1	DecisionTreeClassifier object of sklearn.tree._classes module
filename	str	61	C:\Users\pacho\Downloads\data minning\pp5i_train.top20.gr.csv

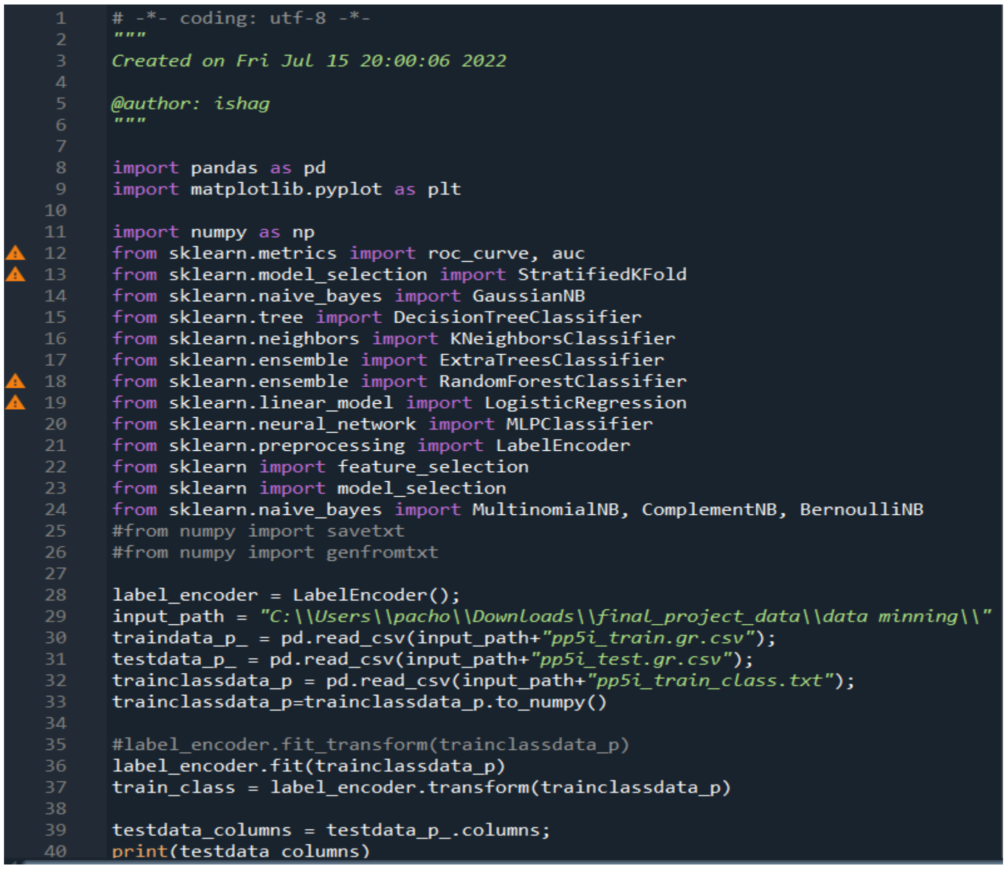
```
A[illegible]g data on the basis of Gene
Training data              SNO    1   2    3    4  ...   66    67    68    69        Gene
2506        U00921_at   20  20   20   20  ...  118   115   102   119  222.388504
6528      U59877_s_at   20  45   20   20  ...  923  1358  1081  1120  166.874021
6663     AF000424_s_at  20  20   20   20  ...   73   102    94    92  162.284150
5503      D13631_s_at   21  54   20   31  ...  331   396   314   293  148.242194
3688        U79242_at   20  20   20   20  ...  153   192   185   121  139.143388
...               ...  ...  ..  ...  ...  ...  ...   ...   ...   ...         ...
6861   X54489_rna1_at   20  20   20   20  ...   20    20    20    20    0.102060
1241        L13042_at   50  53   48   42  ...   48    40    41    30    0.099092
1658   M14159_cds2_at   48  85  108   27  ...   20    62    48    38    0.092244
3355        U56816_at  122  73   79  120  ...  114    73    76   103    0.081504
4156        X51757_at   20  20   20   21  ...   50    25    20    32    0.065644

[6413 rows x 71 columns]
Test data              SNO  101  102  103  104  ...  120  121  122  123        Gene
2506        U00921_at   20   25   20   20  ...   20   20   20   20  222.388504
6528      U59877_s_at  176   20   20   20  ...  180  278   20  185  166.874021
6663     AF000424_s_at  20   20   20   20  ...   20   20   20   20  162.284150
```

TRAINING THE DATASET

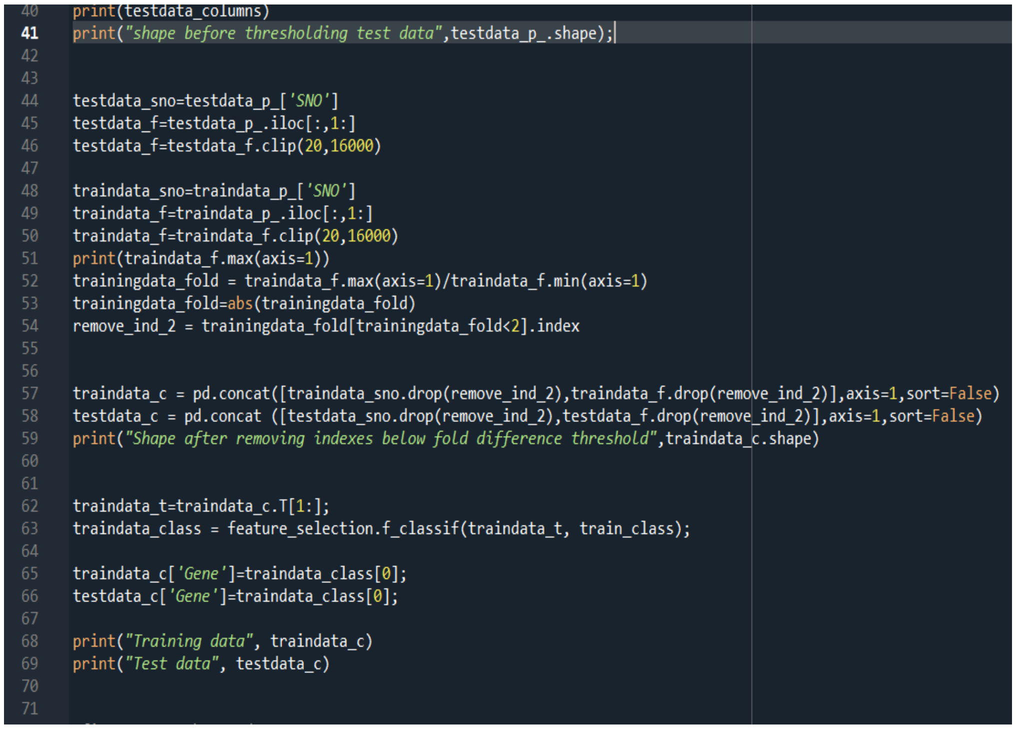


This is a plot of Training Dataset which gives a better visualization and understanding for the analyzation.

CODE:



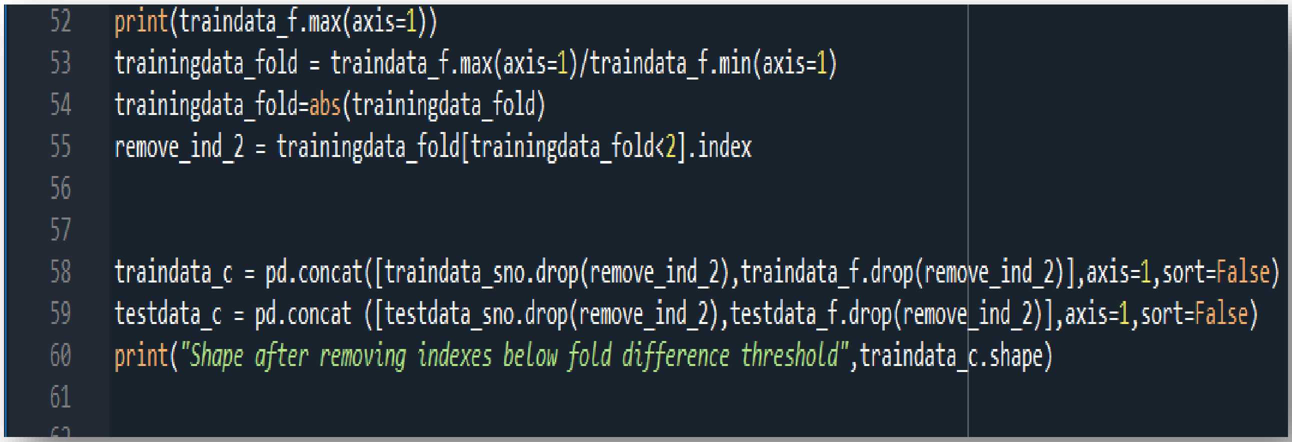
```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jul 15 20:00:06 2022

@author: ishag
"""

import pandas as pd
import matplotlib.pyplot as plt

import numpy as np
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn import feature_selection
from sklearn import model_selection
from sklearn.naive_bayes import MultinomialNB, ComplementNB, BernoulliNB
#from numpy import savetxt
#from numpy import genfromtxt

label_encoder = LabelEncoder();
input_path = "C:\\Users\\pacho\\Downloads\\final_project_data\\data minning\\"
traindata_p_ = pd.read_csv(input_path+"pp5i_train.gr.csv");
testdata_p_ = pd.read_csv(input_path+"pp5i_test.gr.csv");
trainclassdata_p = pd.read_csv(input_path+"pp5i_train_class.txt");
trainclassdata_p=trainclassdata_p.to_numpy()

#label_encoder.fit_transform(trainclassdata_p)
label_encoder.fit(trainclassdata_p)
train_class = label_encoder.transform(trainclassdata_p)

testdata_columns = testdata_p_.columns;
```



```python
print(testdata_columns)
print("shape before thresholding test data",testdata_p_.shape);


testdata_sno=testdata_p_['SNO']
testdata_f=testdata_p_.iloc[:,1:]
testdata_f=testdata_f.clip(20,16000)

traindata_sno=traindata_p_['SNO']
traindata_f=traindata_p_.iloc[:,1:]
traindata_f=traindata_f.clip(20,16000)
print(traindata_f.max(axis=1))
trainingdata_fold = traindata_f.max(axis=1)/traindata_f.min(axis=1)
trainingdata_fold=abs(trainingdata_fold)
remove_ind_2 = trainingdata_fold[trainingdata_fold<2].index


traindata_c = pd.concat([traindata_sno.drop(remove_ind_2),traindata_f.drop(remove_ind_2)],axis=1,sort=False)
testdata_c = pd.concat ([testdata_sno.drop(remove_ind_2),testdata_f.drop(remove_ind_2)],axis=1,sort=False)
print("Shape after removing indexes below fold difference threshold",traindata_c.shape)


traindata_t=traindata_c.T[1:];
traindata_class = feature_selection.f_classif(traindata_t, train_class);

traindata_c['Gene']=traindata_class[0];
testdata_c['Gene']=traindata_class[0];

print("Training data", traindata_c)
print("Test data", testdata_c)
```

STEP – 2 SELECTING TOP GENES BY CLASS

- When the data is composed of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale.
- We now have a better feeling for how different the attributes are. The min and max values as well as the means vary a lot. We are likely going to get better results by rescaling the data by removing fold difference i.e. ratio between maximum and minimum values on the training dataset.



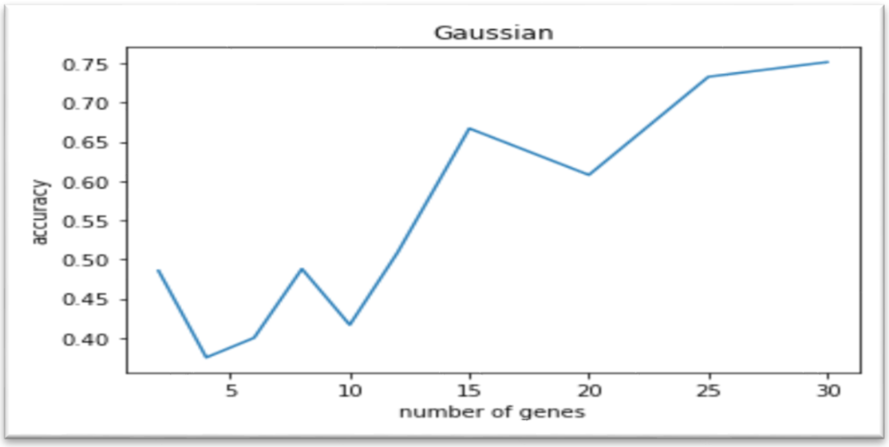
```
print(traindata_f.max(axis=1))
trainingdata_fold = traindata_f.max(axis=1)/traindata_f.min(axis=1)
trainingdata_fold=abs(trainingdata_fold)
remove_ind_2 = trainingdata_fold[trainingdata_fold<2].index


traindata_c = pd.concat([traindata_sno.drop(remove_ind_2),traindata_f.drop(remove_ind_2)],axis=1,sort=False)
testdata_c = pd.concat ([testdata_sno.drop(remove_ind_2),testdata_f.drop(remove_ind_2)],axis=1,sort=False)
print("Shape after removing indexes below fold difference threshold",traindata_c.shape)

```

STEP-3 FINDING BEST CLASSIFIER

By comparing error rates of different algorithms we are testing the accuracy with respect to number of genes of different classifiers like Naïve Bayes, K-NN, Decision tree, Neural network and AdaBoost classifier. By performing spot-checking we can find the best algorithms for our machine learning challenge. Further implementing this method to employ a combination of simple linear (LR and LDA) and nonlinear (KNN, CART, NB, and SVM) algorithms. We have found the accuracy and calculated all the values of n in all the classifiers. We have created a dictionary in which we have added all the classifiers and for each classifier we are iterating it for different values of N.

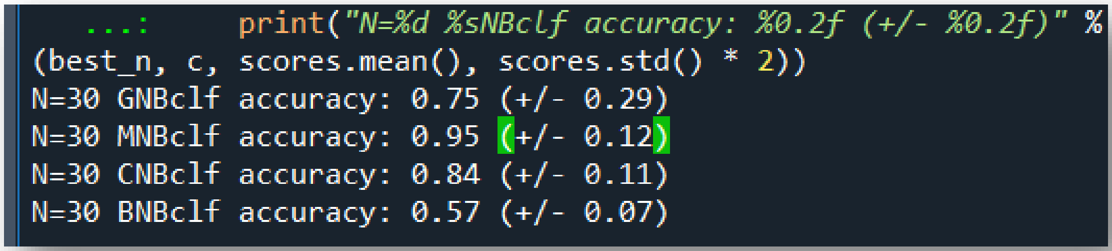
Gaussian -Naïve Bayes Classifier

Gaussian classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Gaussian form is used to represent real valued random variables whose distribution is not known. Reviews and conclusions resulting from gaussian analysis are intuitive which are easy to explain to audiences with basic knowledge of statistics.



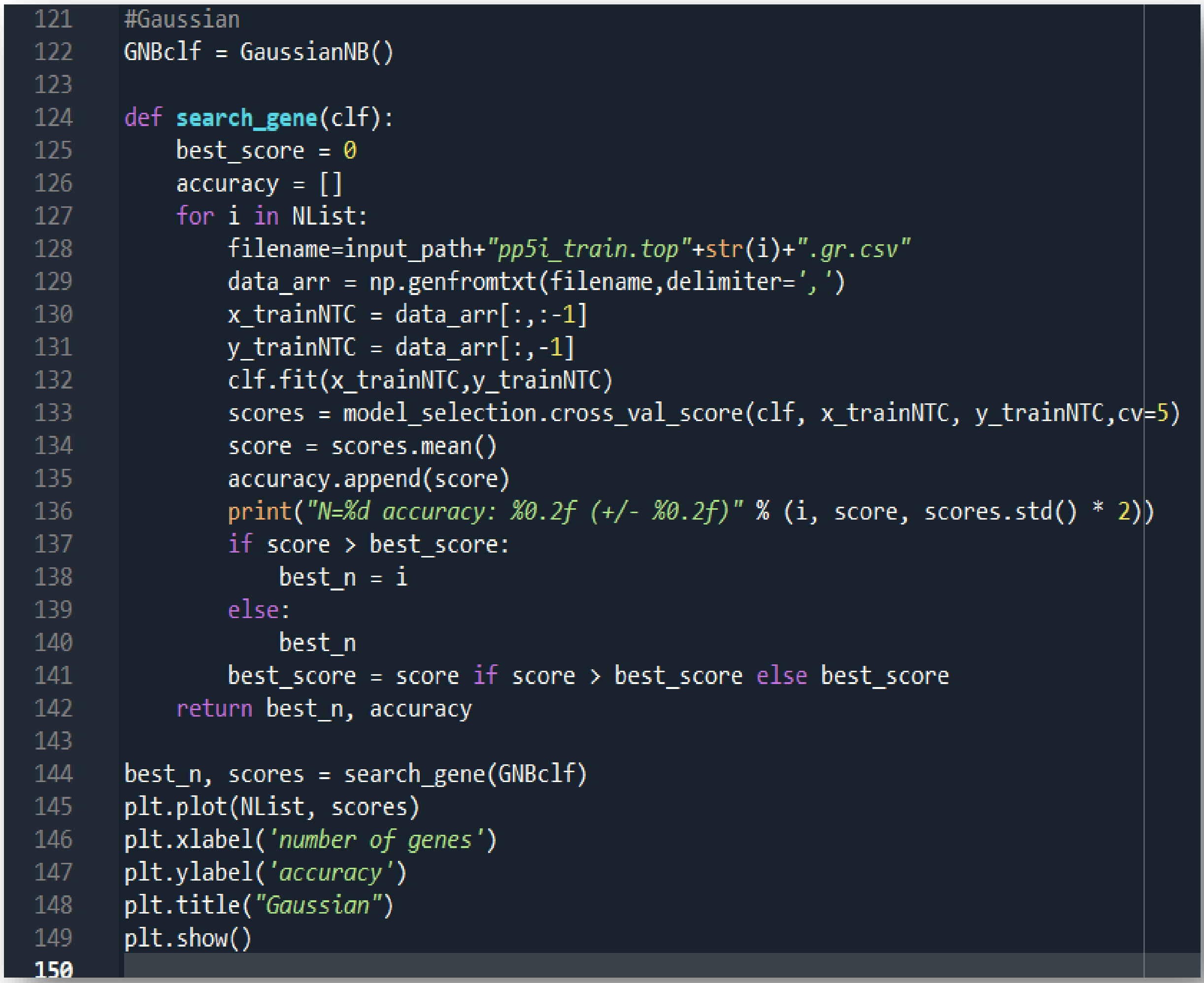
A Gaussian classifier plot for Disease Prediction finding accuracy w.r.t number of genes

Accuracy measure for Gaussian classifier:



```
   ...:     print("N=%d %sNBclf accuracy: %0.2f (+/- %0.2f)" %
(best_n, c, scores.mean(), scores.std() * 2))
N=30 GNBclf accuracy: 0.75 (+/- 0.29)
N=30 MNBclf accuracy: 0.95 (+/- 0.12)
N=30 CNBclf accuracy: 0.84 (+/- 0.11)
N=30 BNBclf accuracy: 0.57 (+/- 0.07)
```

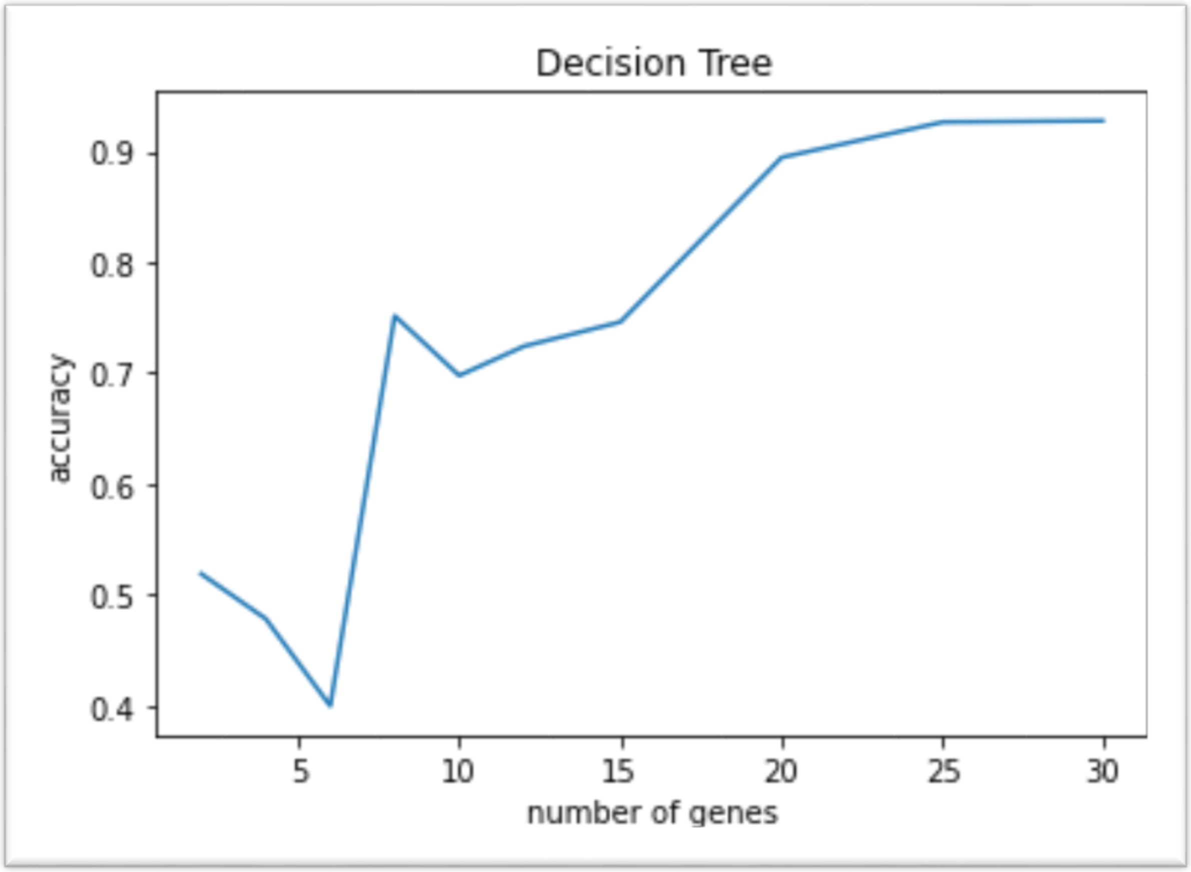
CODE:



```
#Gaussian
GNBclf = GaussianNB()

def search_gene(clf):
    best_score = 0
    accuracy = []
    for i in NList:
        filename=input_path+"pp5i_train.top"+str(i)+".gr.csv"
        data_arr = np.genfromtxt(filename,delimiter=',')
        x_trainNTC = data_arr[:,:-1]
        y_trainNTC = data_arr[:,-1]
        clf.fit(x_trainNTC,y_trainNTC)
        scores = model_selection.cross_val_score(clf, x_trainNTC, y_trainNTC,cv=5)
        score = scores.mean()
        accuracy.append(score)
        print("N=%d accuracy: %0.2f (+/- %0.2f)" % (i, score, scores.std() * 2))
        if score > best_score:
            best_n = i
        else:
            best_n
        best_score = score if score > best_score else best_score
    return best_n, accuracy

best_n, scores = search_gene(GNBclf)
plt.plot(NList, scores)
plt.xlabel('number of genes')
plt.ylabel('accuracy')
plt.title("Gaussian")
plt.show()

```

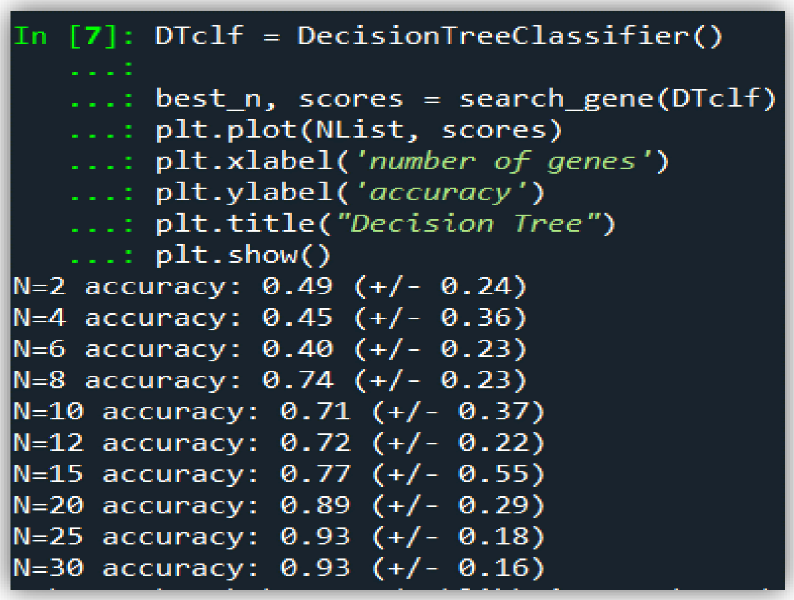
Decision Tree Classifier

Decision trees (DT) are well-suited for large real world tasks as they scale well and can represent complex concepts by constructing simple yet robust logic-based classifiers amenable to direct expert interpretation. It represents one of the most popular classification techniques having advantage as they are easy to understand by humans which makes them particularly useful when the aim of modelling is to understand the underlying processes of the environment. Decision trees are also applicable when the data does not satisfy rigorous assumptions .Decision trees may be of lower predictive quality then more complex classifiers.



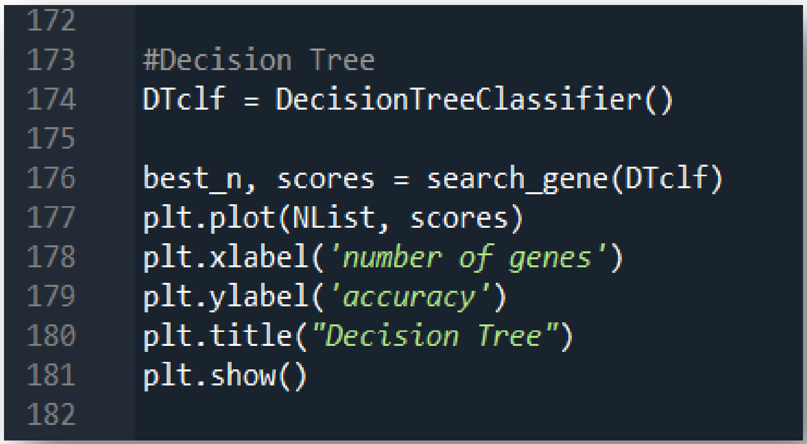
A Decision Tree classifier plot for Disease Prediction finding accuracy w.r.t number of genes

Accuracy measure for decision tree



```
In [7]: DTclf = DecisionTreeClassifier()
   ...:
   ...: best_n, scores = search_gene(DTclf)
   ...: plt.plot(NList, scores)
   ...: plt.xlabel('number of genes')
   ...: plt.ylabel('accuracy')
   ...: plt.title("Decision Tree")
   ...: plt.show()
N=2 accuracy: 0.49 (+/- 0.24)
N=4 accuracy: 0.45 (+/- 0.36)
N=6 accuracy: 0.40 (+/- 0.23)
N=8 accuracy: 0.74 (+/- 0.23)
N=10 accuracy: 0.71 (+/- 0.37)
N=12 accuracy: 0.72 (+/- 0.22)
N=15 accuracy: 0.77 (+/- 0.55)
N=20 accuracy: 0.89 (+/- 0.29)
N=25 accuracy: 0.93 (+/- 0.18)
N=30 accuracy: 0.93 (+/- 0.16)
```

CODE:

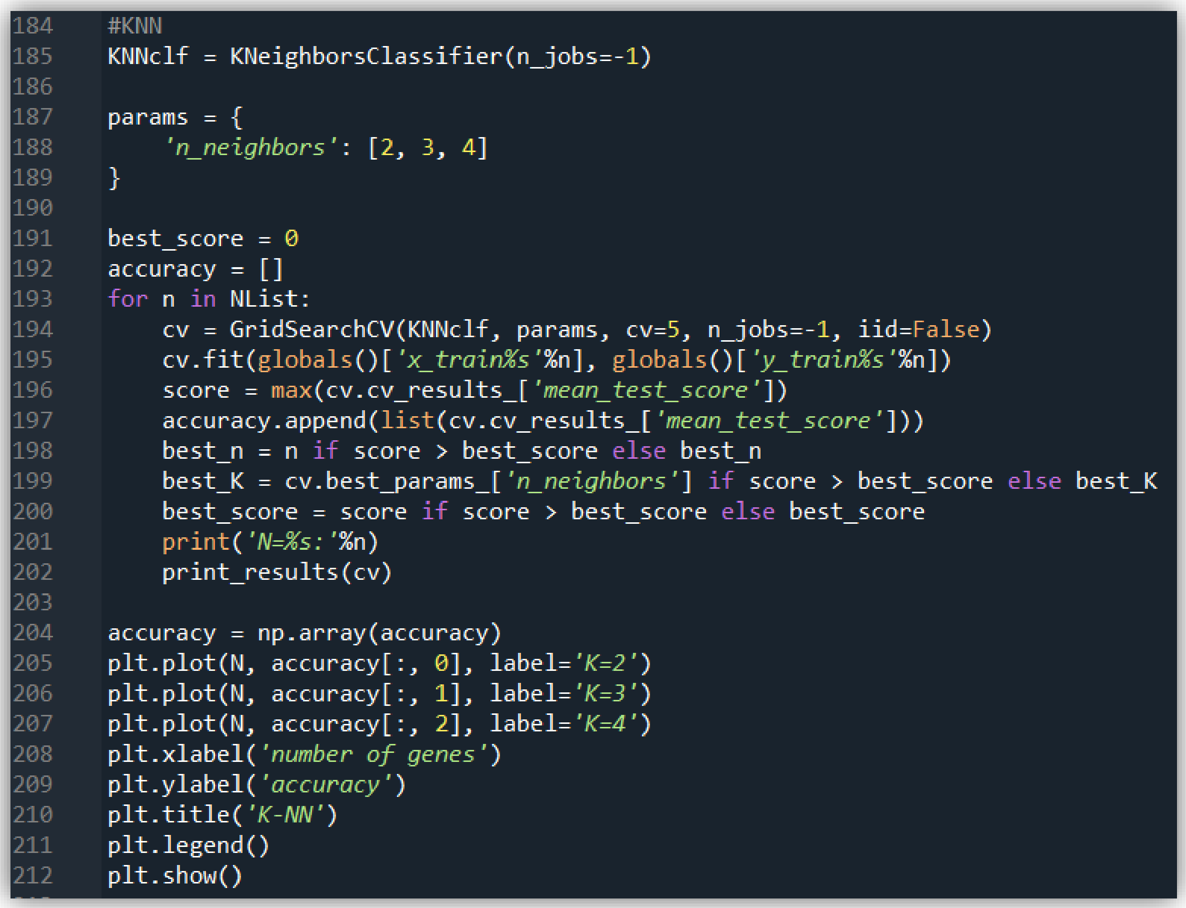


```

#Decision Tree
DTclf = DecisionTreeClassifier()

best_n, scores = search_gene(DTclf)
plt.plot(NList, scores)
plt.xlabel('number of genes')
plt.ylabel('accuracy')
plt.title("Decision Tree")
plt.show()

```

KNN Classifier

Classification technique has a vital role in microarray experiments, for purposes of classifying biological samples and prediction using microarray gene expression data. K-nearest neighbor classifier is one of the introductory supervised classifier, The simple version of the K-nearest neighbor classifier algorithms is to predict the target label by finding the nearest neighbor class. The closest class will be identified using the distance measures like Euclidean distance. This classifier provides accuracy based on the k value.

CODE:

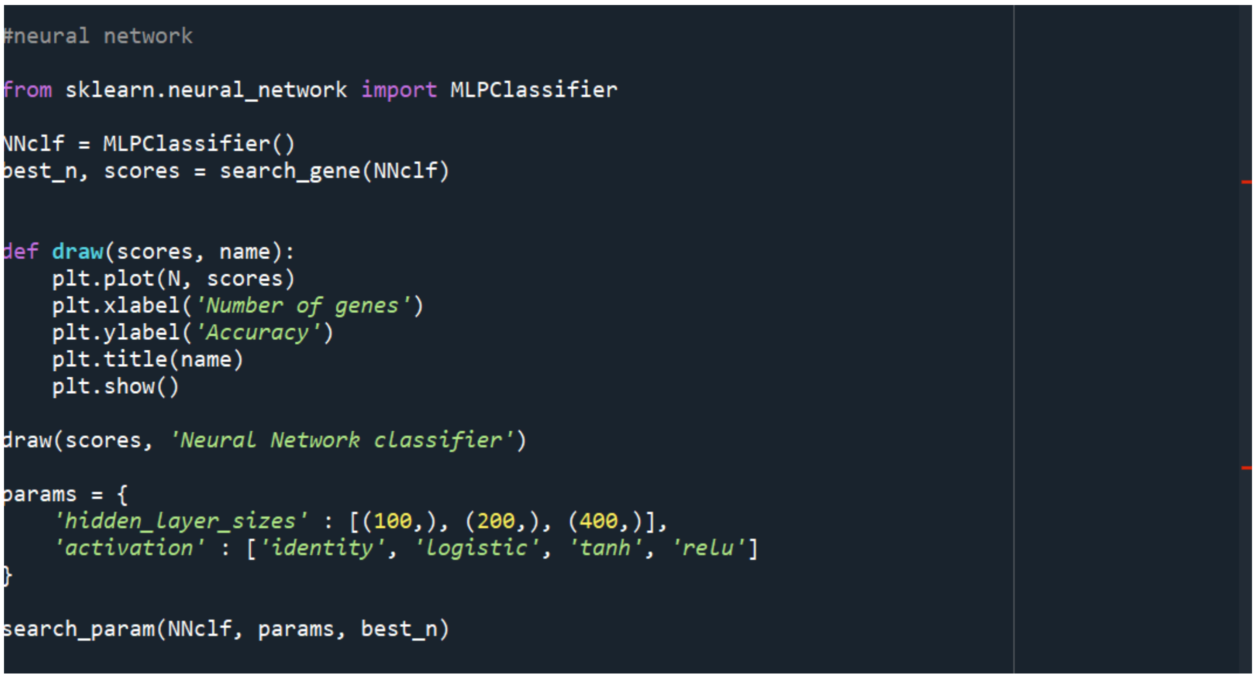


```python
#KNN
KNNclf = KNeighborsClassifier(n_jobs=-1)

params = {
    'n_neighbors': [2, 3, 4]
}

best_score = 0
accuracy = []
for n in NList:
    cv = GridSearchCV(KNNclf, params, cv=5, n_jobs=-1, iid=False)
    cv.fit(globals()['x_train%s'%n], globals()['y_train%s'%n])
    score = max(cv.cv_results_['mean_test_score'])
    accuracy.append(list(cv.cv_results_['mean_test_score']))
    best_n = n if score > best_score else best_n
    best_K = cv.best_params_['n_neighbors'] if score > best_score else best_K
    best_score = score if score > best_score else best_score
    print('N=%s:'%n)
    print_results(cv)

accuracy = np.array(accuracy)
plt.plot(N, accuracy[:, 0], label='K=2')
plt.plot(N, accuracy[:, 1], label='K=3')
plt.plot(N, accuracy[:, 2], label='K=4')
plt.xlabel('number of genes')
plt.ylabel('accuracy')
plt.title('K-NN')
plt.legend()
plt.show()
```

Neural Network Classifier

Neural network classifier consists of units (neurons), arranged in layers, which convert an input vector into some output. Each unit takes an input, applies a (often nonlinear) function to it and then passes the output on to the next layer. Generally the networks are defined to be feed-forward: a unit feeds its output to all the units on the next layer, but there is no feedback to the previous layer. Weightings are applied to the signals passing from one unit to another, and it is these weightings which are tuned in the training phase to adapt a neural network to the particular problem at hand. This is the learning phase. However, neural networks are more computationally expensive than any other traditional algorithm. Reducing the network to a specific value of the sampling error implies that the training is complete. This value does not provide us with the best results.

CODE

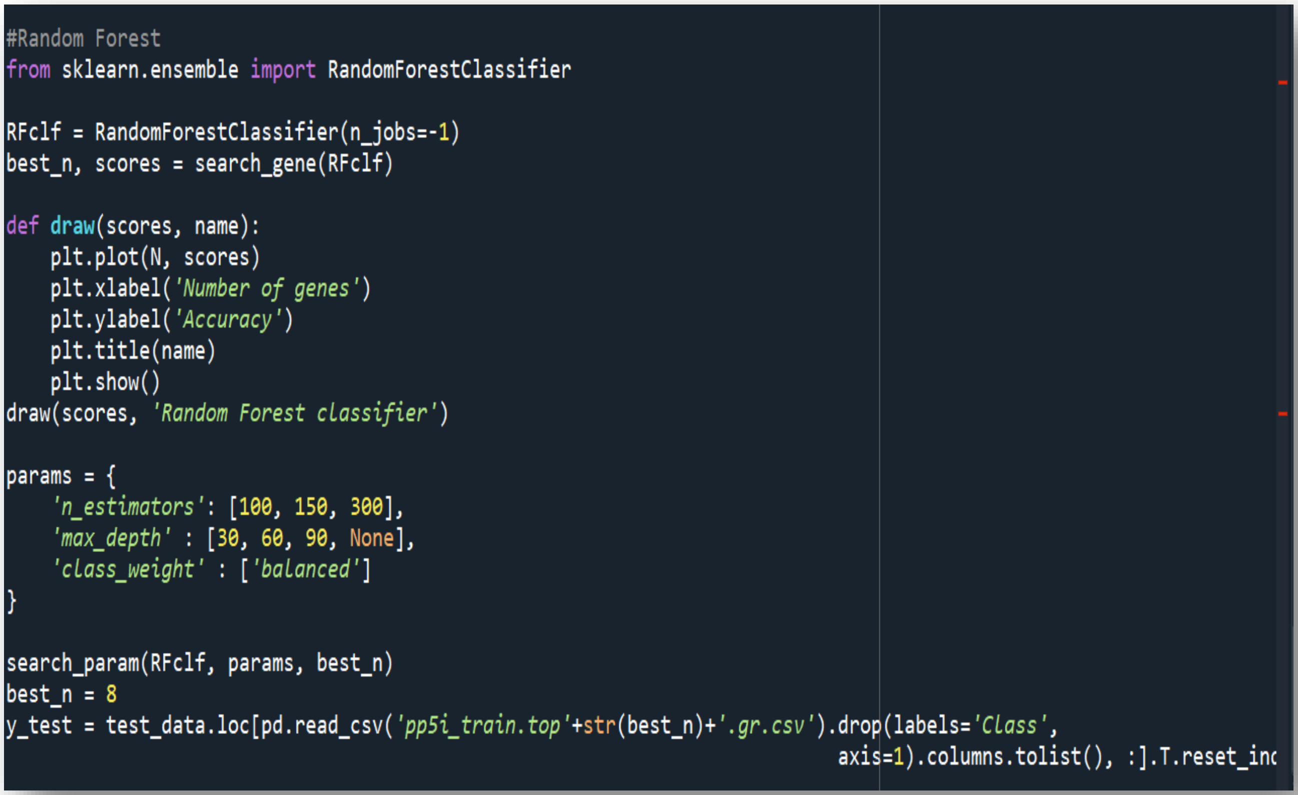


```
#neural network

from sklearn.neural_network import MLPClassifier

NNclf = MLPClassifier()
best_n, scores = search_gene(NNclf)


def draw(scores, name):
    plt.plot(N, scores)
    plt.xlabel('Number of genes')
    plt.ylabel('Accuracy')
    plt.title(name)
    plt.show()

draw(scores, 'Neural Network classifier')

params = {
    'hidden_layer_sizes' : [(100,), (200,), (400,)],
    'activation' : ['identity', 'logistic', 'tanh', 'relu']
}

search_param(NNclf, params, best_n)
```

Random Forest Classifier

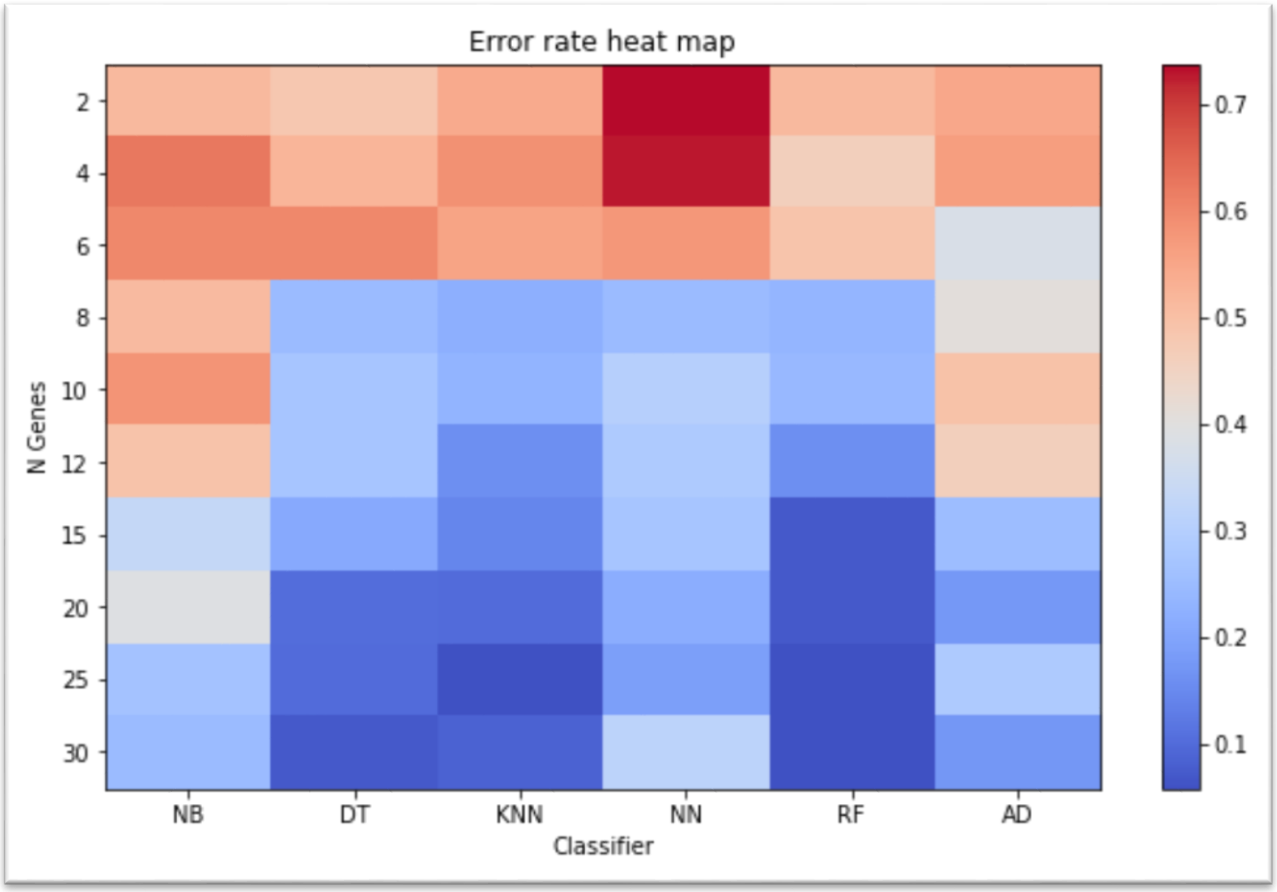
A large number of decision trees are built during the training phase of the random forests or random decision forests ensemble learning approach, which is used for classification, regression, and other tasks. The class that the majority of the trees chose is the output of the random forest for classification problems. The mean or average prediction of each individual tree is returned for regression tasks. The tendency of decision trees to overfit their training set is corrected by random decision forests. Although they frequently outperform decision trees, gradient enhanced trees are more accurate than random forests. However, their effectiveness may be impacted by data peculiarities.

CODE:



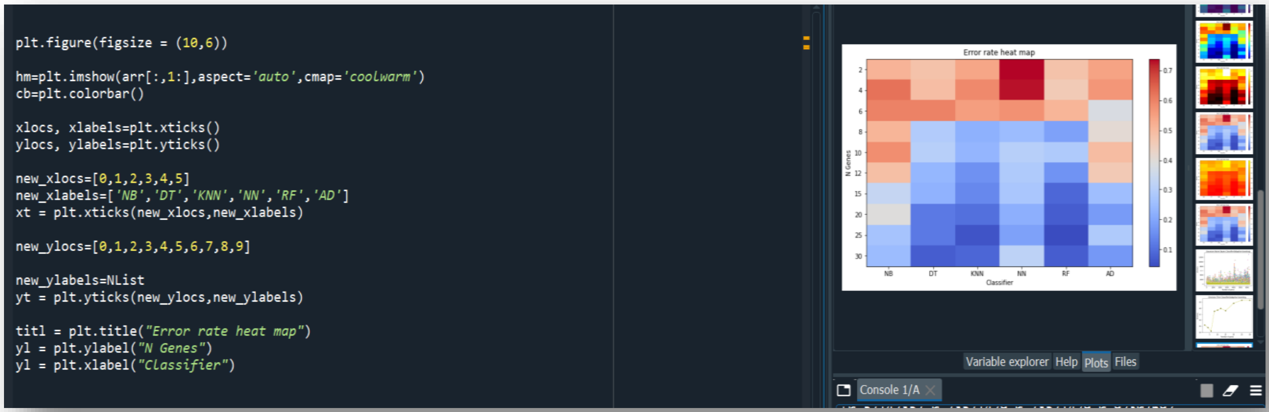
```python
#Random Forest
from sklearn.ensemble import RandomForestClassifier

RFclf = RandomForestClassifier(n_jobs=-1)
best_n, scores = search_gene(RFclf)

def draw(scores, name):
    plt.plot(N, scores)
    plt.xlabel('Number of genes')
    plt.ylabel('Accuracy')
    plt.title(name)
    plt.show()
draw(scores, 'Random Forest classifier')

params = {
    'n_estimators': [100, 150, 300],
    'max_depth' : [30, 60, 90, None],
    'class_weight' : ['balanced']
}

search_param(RFclf, params, best_n)
best_n = 8
y_test = test_data.loc[pd.read_csv('pp5i_train.top'+str(best_n)+'.gr.csv').drop(labels='Class',
                                                                         axis=1).columns.tolist(), :].T.reset_inc
```

REPRESENTATION OF ERROR RATE USING HEAT MAP

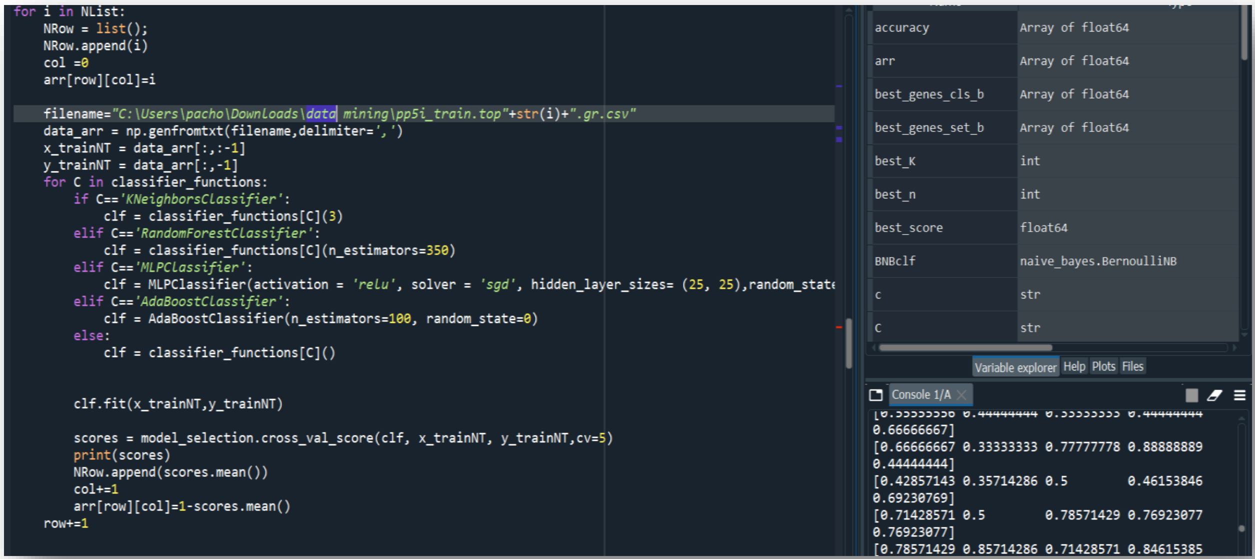


Plot visualizing the error rate of n genes w.r.t classifiers

CODE:



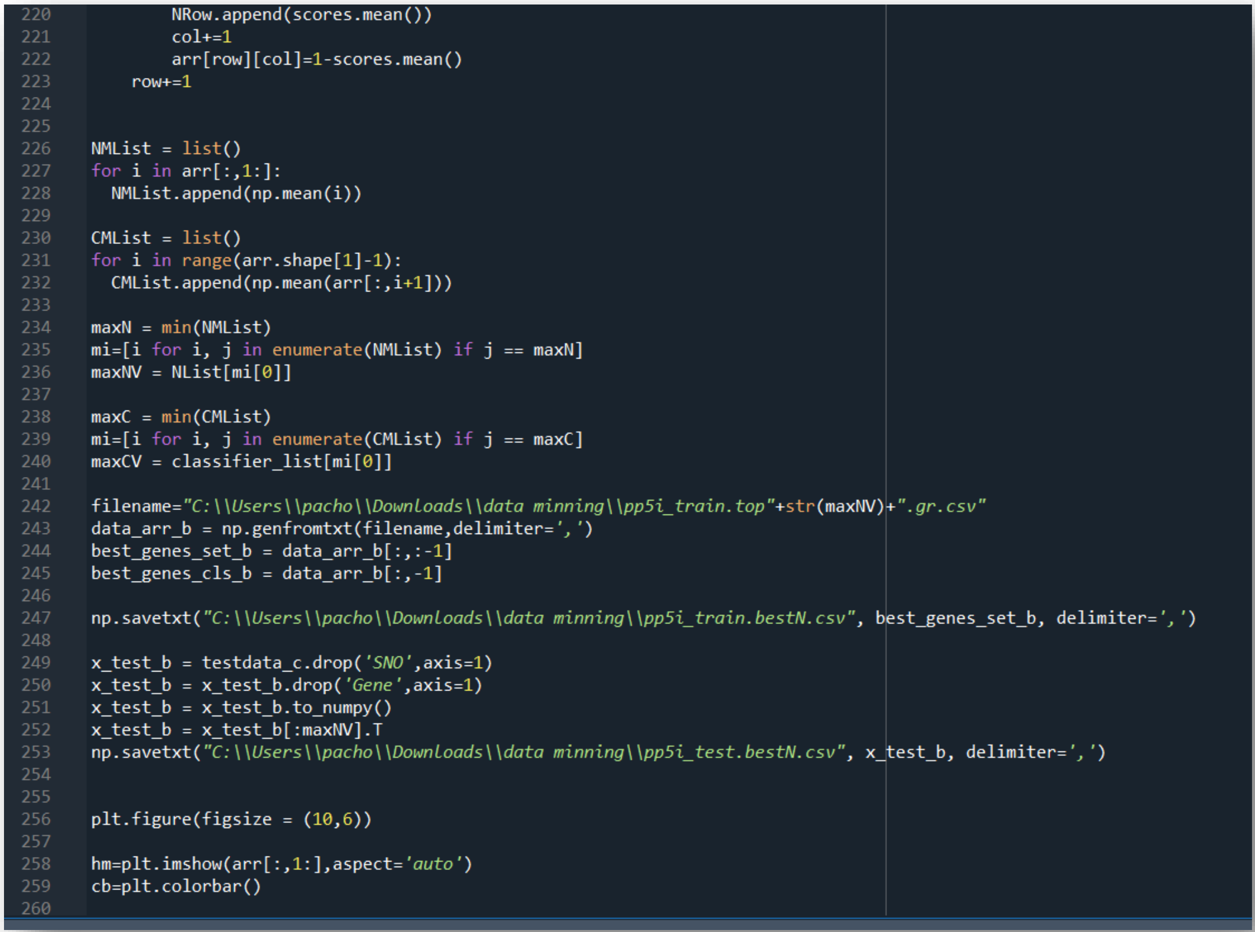
```
plt.figure(figsize = (10,6))

hm=plt.imshow(arr[:,1:],aspect='auto',cmap='coolwarm')
cb=plt.colorbar()

xlocs, xlabels=plt.xticks()
ylocs, ylabels=plt.yticks()

new_xlocs=[0,1,2,3,4,5]
new_xlabels=['NB','DT','KNN','NN','RF','AD']
xt = plt.xticks(new_xlocs,new_xlabels)

new_ylocs=[0,1,2,3,4,5,6,7,8,9]

new_ylabels=NList
yt = plt.yticks(new_ylocs,new_ylabels)

titl = plt.title("Error rate heat map")
yl = plt.ylabel("N Genes")
yl = plt.xlabel("Classifier")
```



```
for i in NList:
    NRow = list();
    NRow.append(i)
    col =0
    arr[row][col]=i

    filename="C:\Users\pacho\Downloads\data mining\pp5i_train.top"+str(i)+".gr.csv"
    data_arr = np.genfromtxt(filename,delimiter=',')
    x_trainNT = data_arr[:,:-1]
    y_trainNT = data_arr[:,-1]
    for C in classifier_functions:
        if C=='KNeighborsClassifier':
            clf = classifier_functions[C](3)
        elif C=='RandomForestClassifier':
            clf = classifier_functions[C](n_estimators=350)
        elif C=='MLPClassifier':
            clf = MLPClassifier(activation = 'relu', solver = 'sgd', hidden_layer_sizes= (25, 25),random_state
        elif C=='AdaBoostClassifier':
            clf = AdaBoostClassifier(n_estimators=100, random_state=0)
        else:
            clf = classifier_functions[C]()


        clf.fit(x_trainNT,y_trainNT)

        scores = model_selection.cross_val_score(clf, x_trainNT, y_trainNT,cv=5)
        print(scores)
        NRow.append(scores.mean())
        col+=1
        arr[row][col]=1-scores.mean()
    row+=1
```

accuracy	Array of float64
arr	Array of float64
best_genes_cls_b	Array of float64
best_genes_set_b	Array of float64
best_K	int
best_n	int
best_score	float64
BNBclf	naive_bayes.BernoulliNB
c	str
C	str

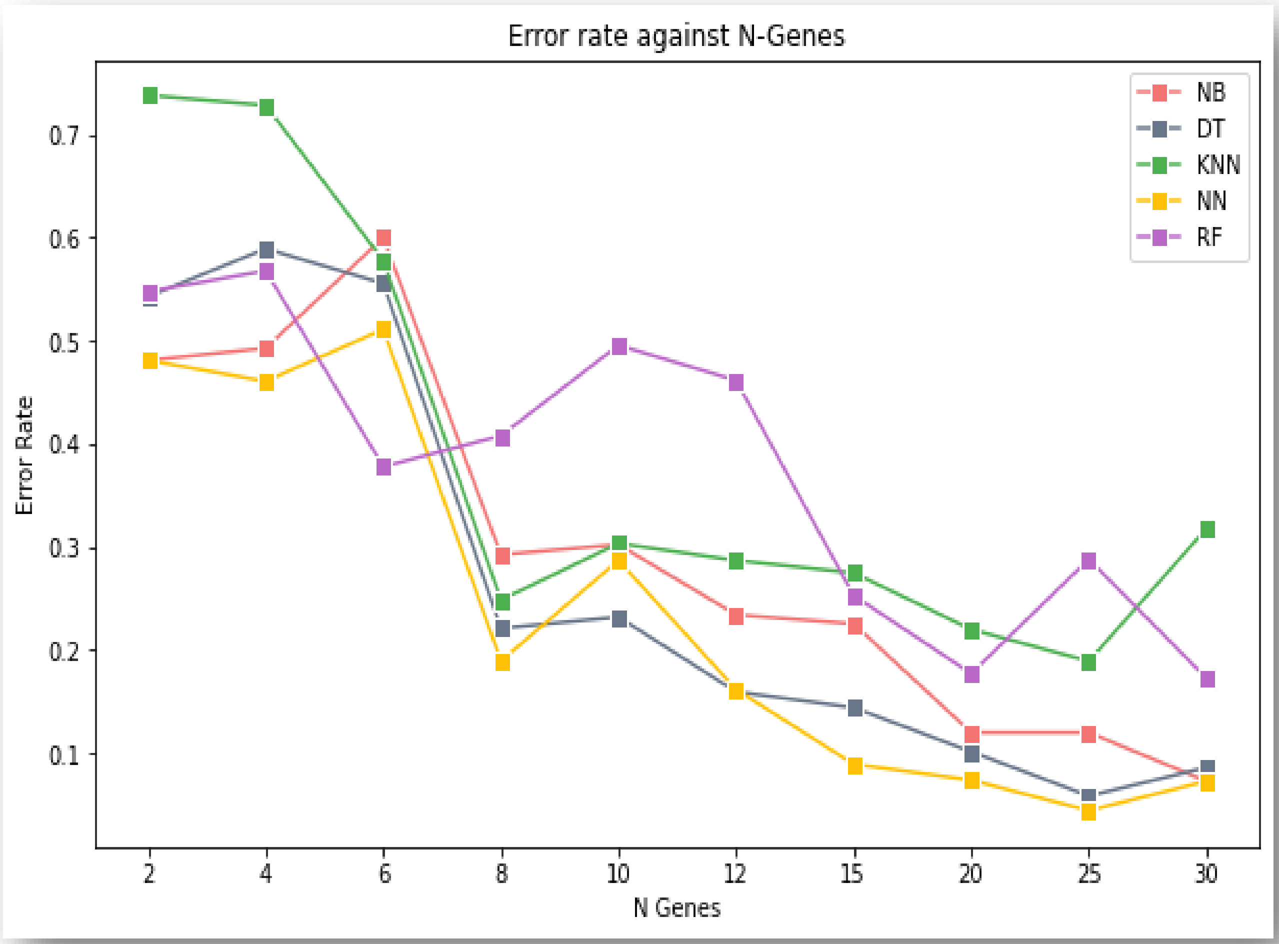
```
[0.55555556 0.44444444 0.33333333 0.44444444
0.66666667]
[0.66666667 0.33333333 0.77777778 0.88888889
0.44444444]
[0.42857143 0.35714286 0.5        0.46153846
0.69230769]
[0.71428571 0.5        0.78571429 0.76923077
0.76923077]
[0.78571429 0.85714286 0.71428571 0.84615385
```



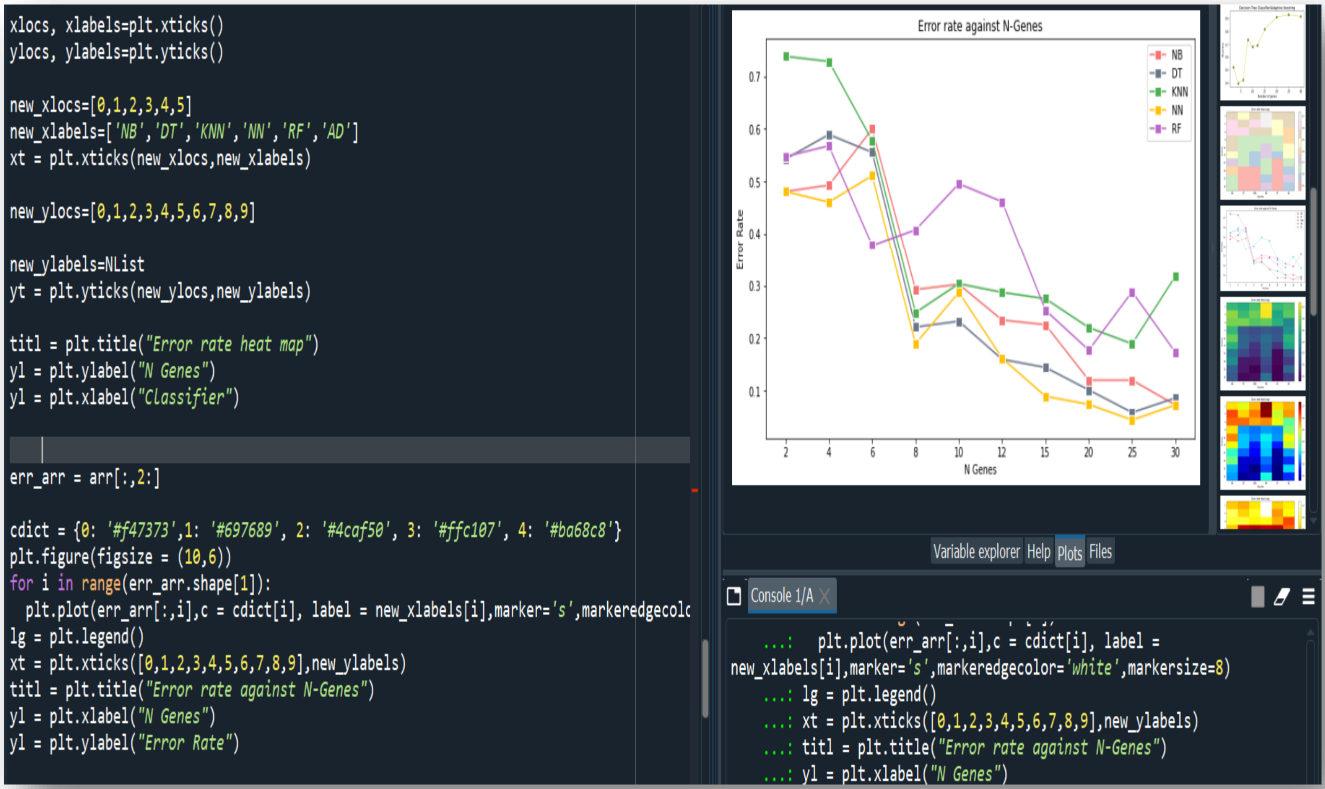
```
            NRow.append(scores.mean())
            col+=1
            arr[row][col]=1-scores.mean()
        row+=1


NMList = list()
for i in arr[:,1:]:
  NMList.append(np.mean(i))

CMList = list()
for i in range(arr.shape[1]-1):
  CMList.append(np.mean(arr[:,i+1]))

maxN = min(NMList)
mi=[i for i, j in enumerate(NMList) if j == maxN]
maxNV = NList[mi[0]]

maxC = min(CMList)
mi=[i for i, j in enumerate(CMList) if j == maxC]
maxCV = classifier_list[mi[0]]

filename="C:\\Users\\pacho\\Downloads\\data minning\\pp5i_train.top"+str(maxNV)+".gr.csv"
data_arr_b = np.genfromtxt(filename,delimiter=',')
best_genes_set_b = data_arr_b[:,:-1]
best_genes_cls_b = data_arr_b[:,-1]

np.savetxt("C:\\Users\\pacho\\Downloads\\data minning\\pp5i_train.bestN.csv", best_genes_set_b, delimiter=',')

x_test_b = testdata_c.drop('SNO',axis=1)
x_test_b = x_test_b.drop('Gene',axis=1)
x_test_b = x_test_b.to_numpy()
x_test_b = x_test_b[:maxNV].T
np.savetxt("C:\\Users\\pacho\\Downloads\\data minning\\pp5i_test.bestN.csv", x_test_b, delimiter=',')


plt.figure(figsize = (10,6))

hm=plt.imshow(arr[:,1:],aspect='auto')
cb=plt.colorbar()
```

COMPARISION OF ERROR RATE AGAINST N-GENS OF CLASSIFIER

Comparison of the effectiveness and accuracy of classifiers is demonstrated through the plot. The results show that our gene selection method is capable of achieving better accuracies in Extra Tree Classifier as compared to other classifiers with minimum error rate.



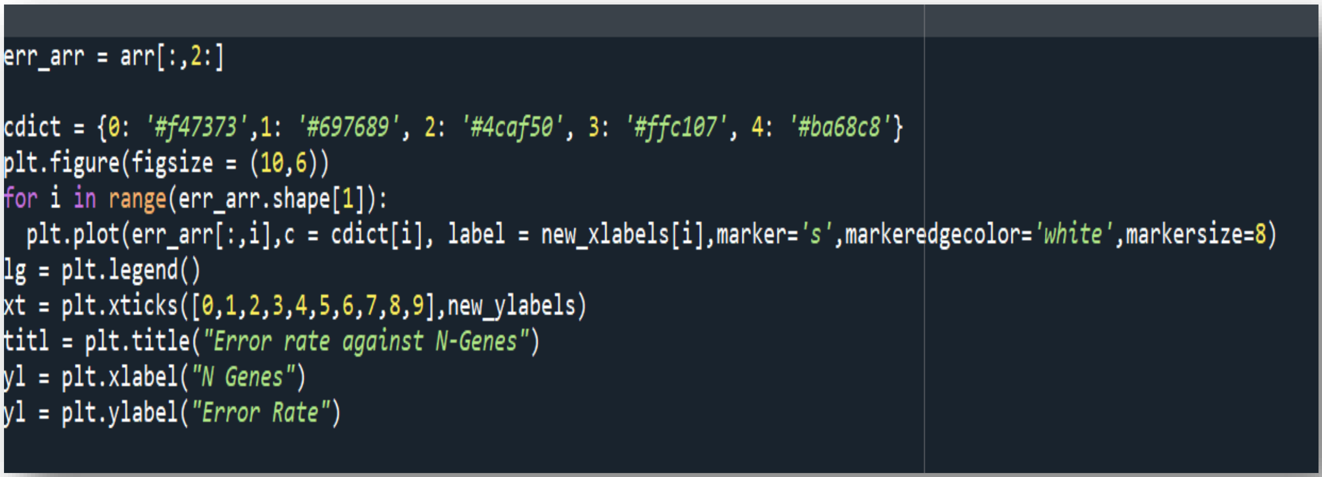
plot showing error against n- genes for different classifiers



```
xlocs, xlabels=plt.xticks()
ylocs, ylabels=plt.yticks()

new_xlocs=[0,1,2,3,4,5]
new_xlabels=['NB','DT','KNN','NN','RF','AD']
xt = plt.xticks(new_xlocs,new_xlabels)

new_ylocs=[0,1,2,3,4,5,6,7,8,9]

new_ylabels=NList
yt = plt.yticks(new_ylocs,new_ylabels)

titl = plt.title("Error rate heat map")
yl = plt.ylabel("N Genes")
yl = plt.xlabel("Classifier")


err_arr = arr[:,2:]

cdict = {0: '#f47373',1: '#697689', 2: '#4caf50', 3: '#ffc107', 4: '#ba68c8'}
plt.figure(figsize = (10,6))
for i in range(err_arr.shape[1]):
  plt.plot(err_arr[:,i],c = cdict[i], label = new_xlabels[i],marker='s',markeredgecolc
lg = plt.legend()
xt = plt.xticks([0,1,2,3,4,5,6,7,8,9],new_ylabels)
titl = plt.title("Error rate against N-Genes")
yl = plt.xlabel("N Genes")
yl = plt.ylabel("Error Rate")
```

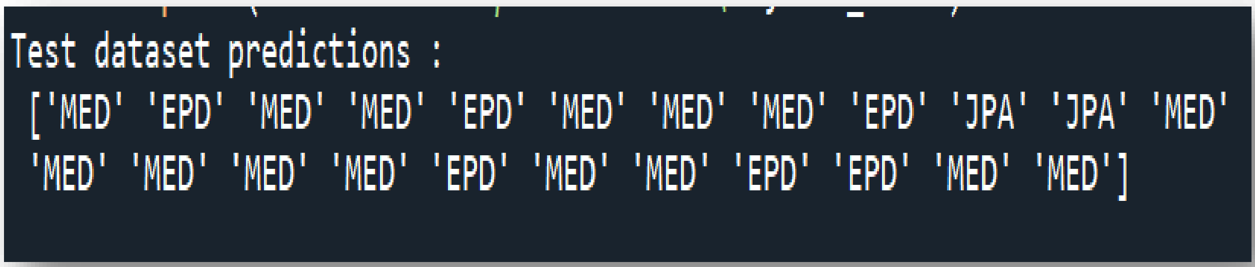
```
   ...:   plt.plot(err_arr[:,i],c = cdict[i], label =
new_xlabels[i],marker='s',markeredgecolor='white',markersize=8)
   ...: lg = plt.legend()
   ...: xt = plt.xticks([0,1,2,3,4,5,6,7,8,9],new_ylabels)
   ...: titl = plt.title("Error rate against N-Genes")
   ...: yl = plt.xlabel("N Genes")
```

<u>CODE</u>:



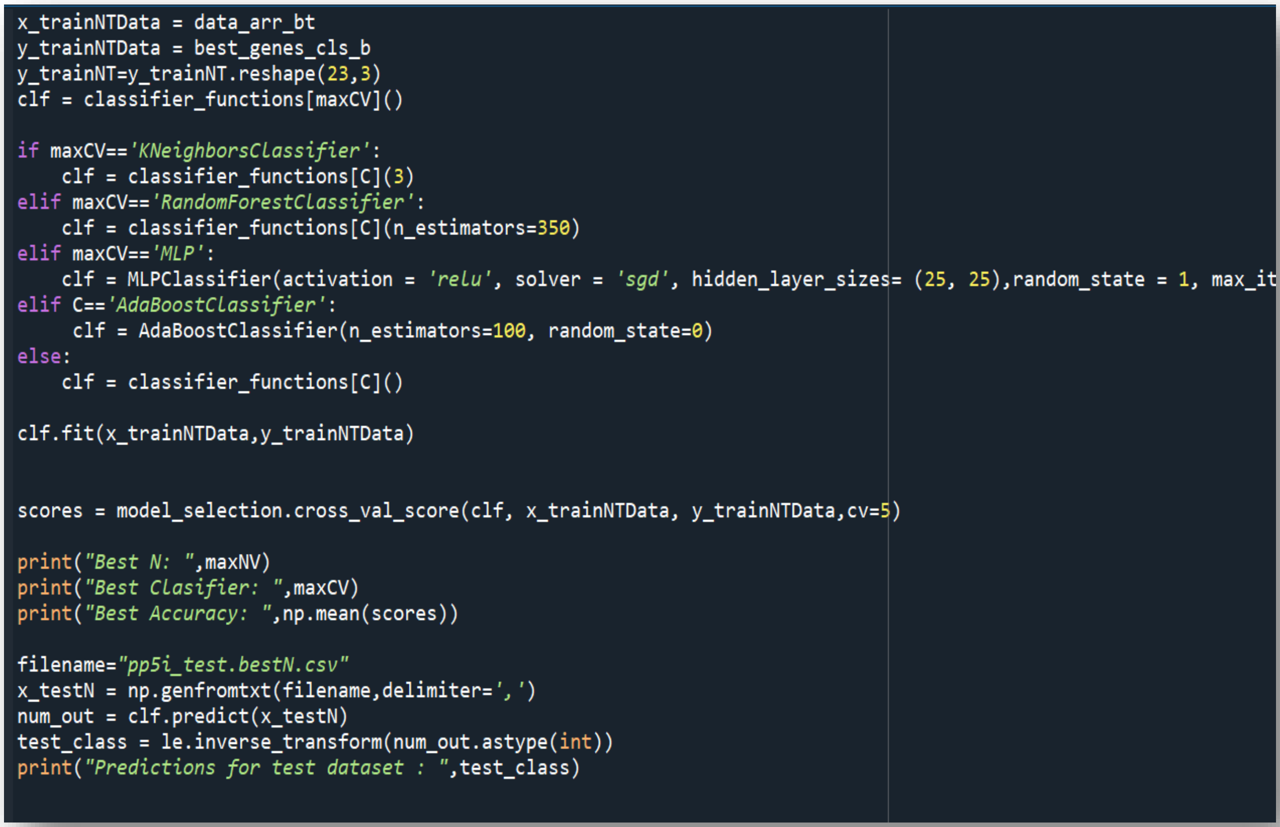
```
err_arr = arr[:,2:]

cdict = {0: '#f47373',1: '#697689', 2: '#4caf50', 3: '#ffc107', 4: '#ba68c8'}
plt.figure(figsize = (10,6))
for i in range(err_arr.shape[1]):
  plt.plot(err_arr[:,i],c = cdict[i], label = new_xlabels[i],marker='s',markeredgecolor='white',markersize=8)
lg = plt.legend()
xt = plt.xticks([0,1,2,3,4,5,6,7,8,9],new_ylabels)
titl = plt.title("Error rate against N-Genes")
yl = plt.xlabel("N Genes")
yl = plt.ylabel("Error Rate")
```

STEP -4 GENRATE PREDICTIONS FOR THE TEST SET



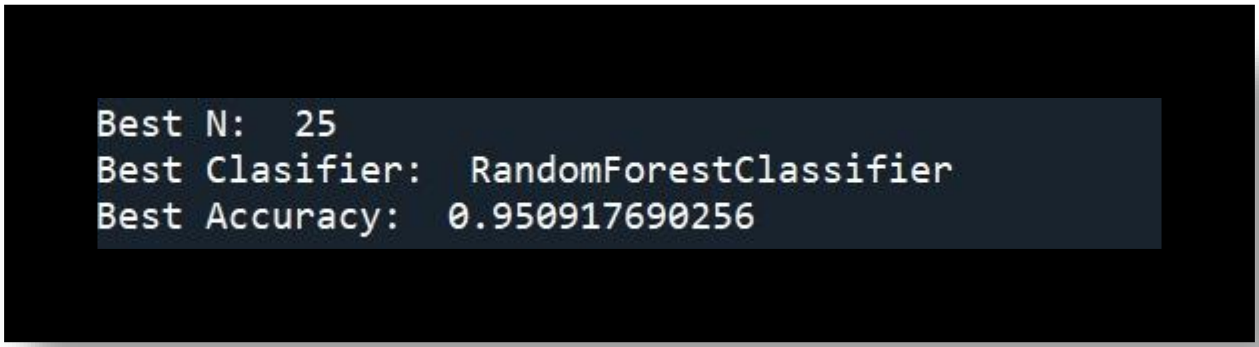
```
Test dataset predictions :
 ['MED' 'EPD' 'MED' 'MED' 'EPD' 'MED' 'MED' 'MED' 'EPD' 'JPA' 'JPA' 'MED'
 'MED' 'MED' 'MED' 'MED' 'EPD' 'MED' 'MED' 'EPD' 'EPD' 'MED' 'MED']
```

CODE:



```
x_trainNTData = data_arr_bt
y_trainNTData = best_genes_cls_b
y_trainNT=y_trainNT.reshape(23,3)
clf = classifier_functions[maxCV]()

if maxCV=='KNeighborsClassifier':
    clf = classifier_functions[C](3)
elif maxCV=='RandomForestClassifier':
    clf = classifier_functions[C](n_estimators=350)
elif maxCV=='MLP':
    clf = MLPClassifier(activation = 'relu', solver = 'sgd', hidden_layer_sizes= (25, 25),random_state = 1, max_it
elif C=='AdaBoostClassifier':
     clf = AdaBoostClassifier(n_estimators=100, random_state=0)
else:
    clf = classifier_functions[C]()

clf.fit(x_trainNTData,y_trainNTData)


scores = model_selection.cross_val_score(clf, x_trainNTData, y_trainNTData,cv=5)

print("Best N: ",maxNV)
print("Best Clasifier: ",maxCV)
print("Best Accuracy: ",np.mean(scores))

filename="pp5i_test.bestN.csv"
x_testN = np.genfromtxt(filename,delimiter=',')
num_out = clf.predict(x_testN)
test_class = le.inverse_transform(num_out.astype(int))
print("Predictions for test dataset : ",test_class)
```

CONCLUSION



```
Best N:  25
Best Clasifier:  RandomForestClassifier
Best Accuracy:  0.950917690256
```

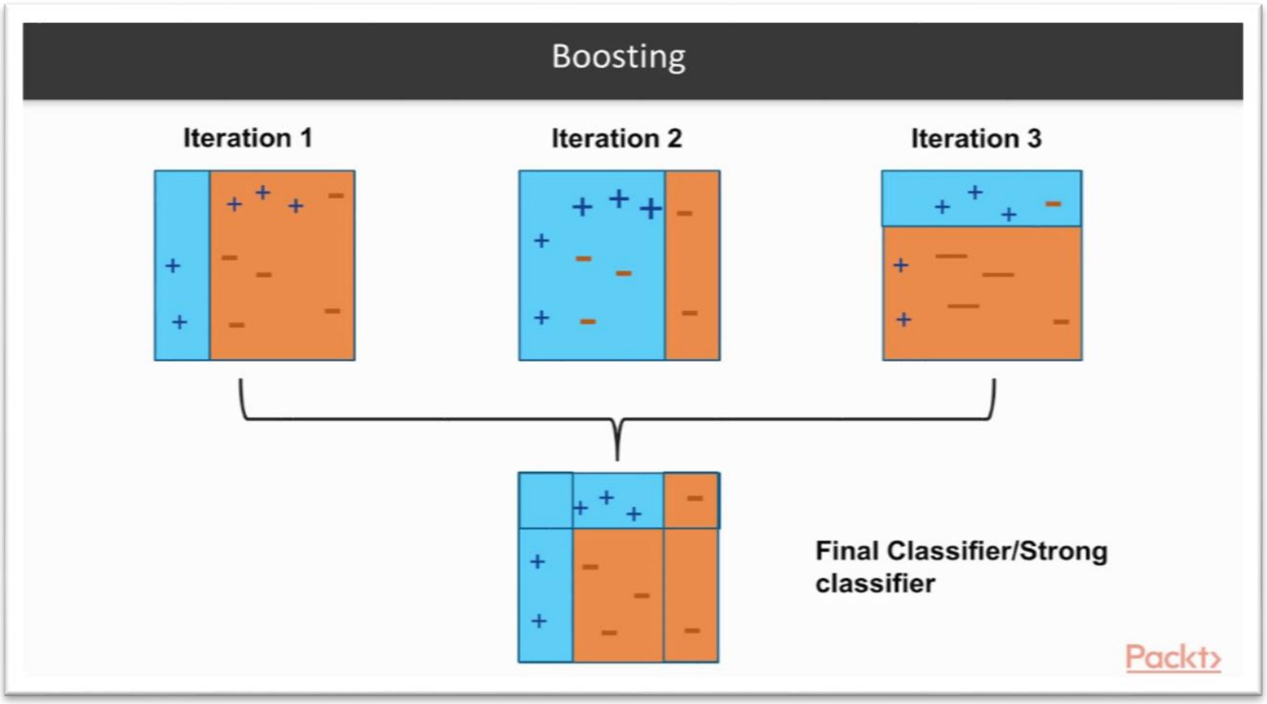
The best classifier for disease prediction is Random Forest Classifier with an accuracy rate of 95.09%

- By performing analysis of several publicly available datasets and simulated datasets we demonstrate that the proposed that Random Forest Classifier method can effectively identify a compact set of genes with high classification accuracy it’s a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a “forest” to output it’s classification result.

STEP -5 Generate a prediction using Adaptive Boosting

AdaBoost Classifier

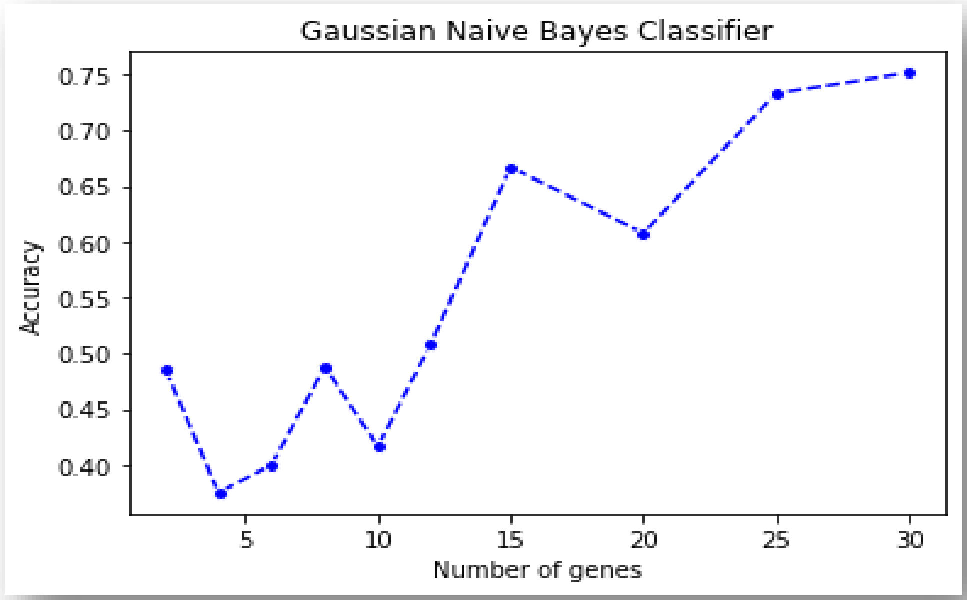
An AdaBoost classifier is a postmodern with the concept of setting the weights of classifiers and training the data sample in each iteration to ensure accurate predictions of unusual observations by fitting a classifier on the original dataset and then fitting additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted so that subsequent classifiers focus more on difficult cases by fitting a classifier on the original dataset and then fitting additional copies of the classifier on the same dataset but The AdaBoost classifier combines multiple low-performing classifiers to produce a high-accuracy strong classifier. AdaBoost does not exhibit overfitting. AdaBoost is vulnerable to data noise. Because it tries to fit each point perfectly, outliers have a significant impact on it.

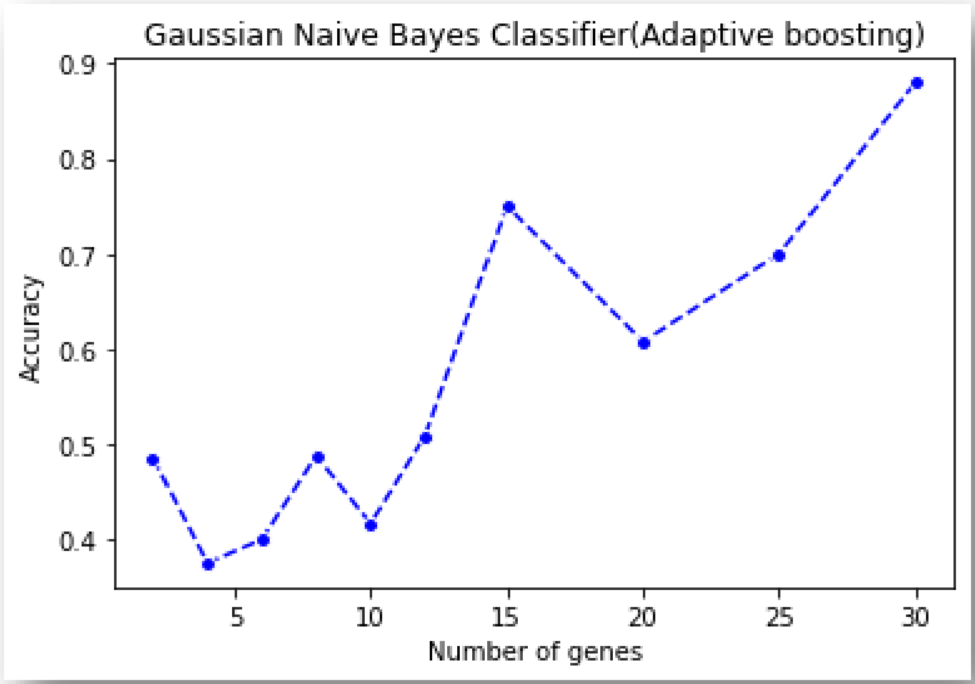


ALGORITHM:

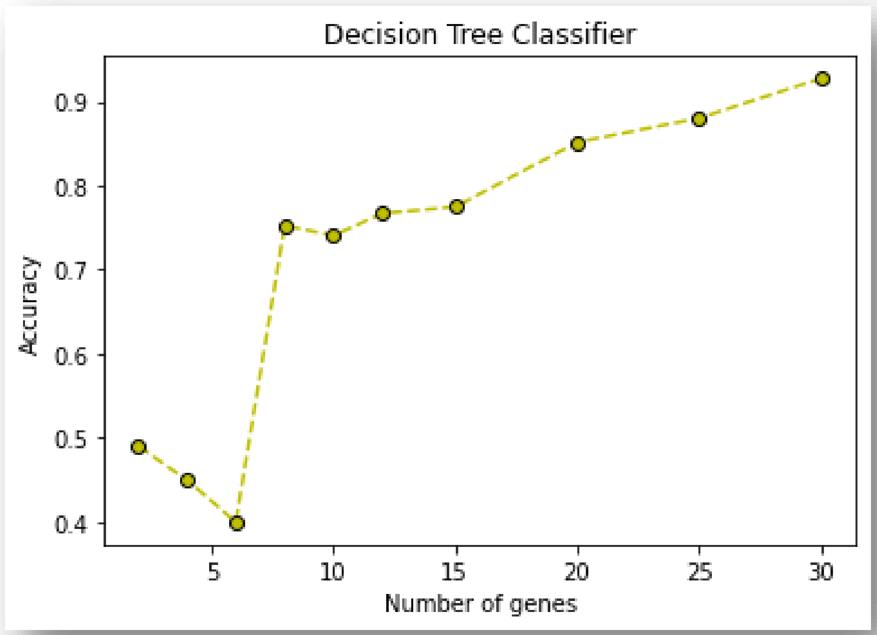
- Based on the weighted samples, a weak classifier (e.g., a decision stump) is built on top of the training data. The weights of each sample indicate how critical it is to be correctly classified in this case. For the first stump, we assign equal weights to all samples.
- We create a decision stump for each variable and evaluate how well each stump classifies samples into their respective target classes. For example, in the diagram below, we check for Age, Junk Food Consumption, and Physical Activity. We'd look at how many samples were classified correctly or incorrectly as Fit or Unfit for each individual stump.
- More weight is assigned to the incorrectly classified samples in order for them to be correctly classified in the next decision stump. Weight is also assigned to each classifier based on its accuracy, so high accuracy equals high weight.
- Repeat Step 2 until all of the data points have been correctly classified or the maximum iteration level is reached.

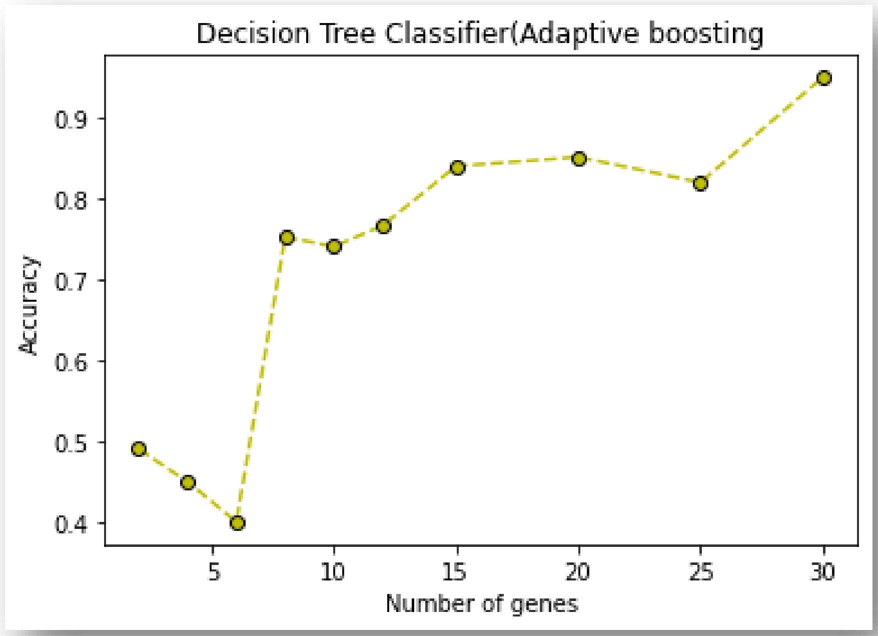
NAÏVE BAYES' (ADAPTIVE BOOSTING)



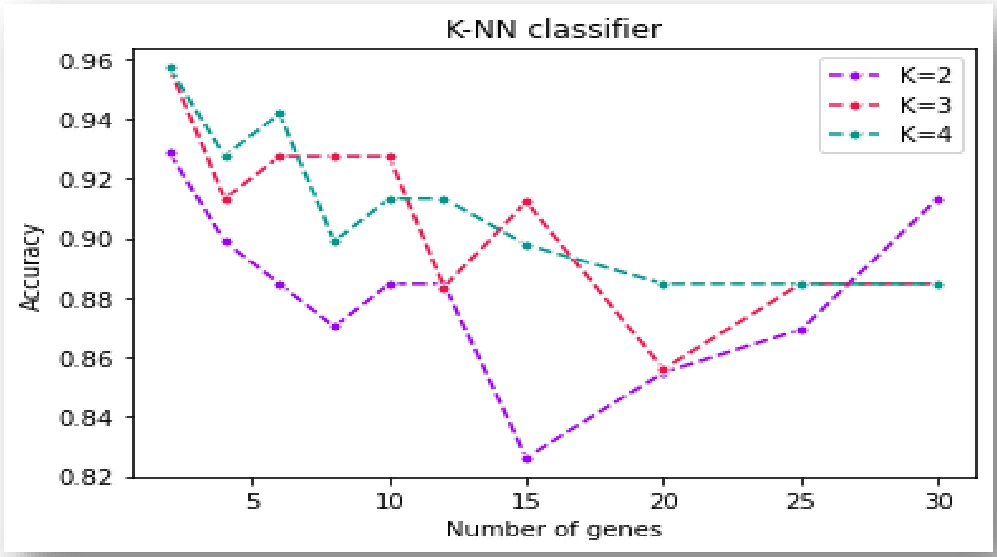


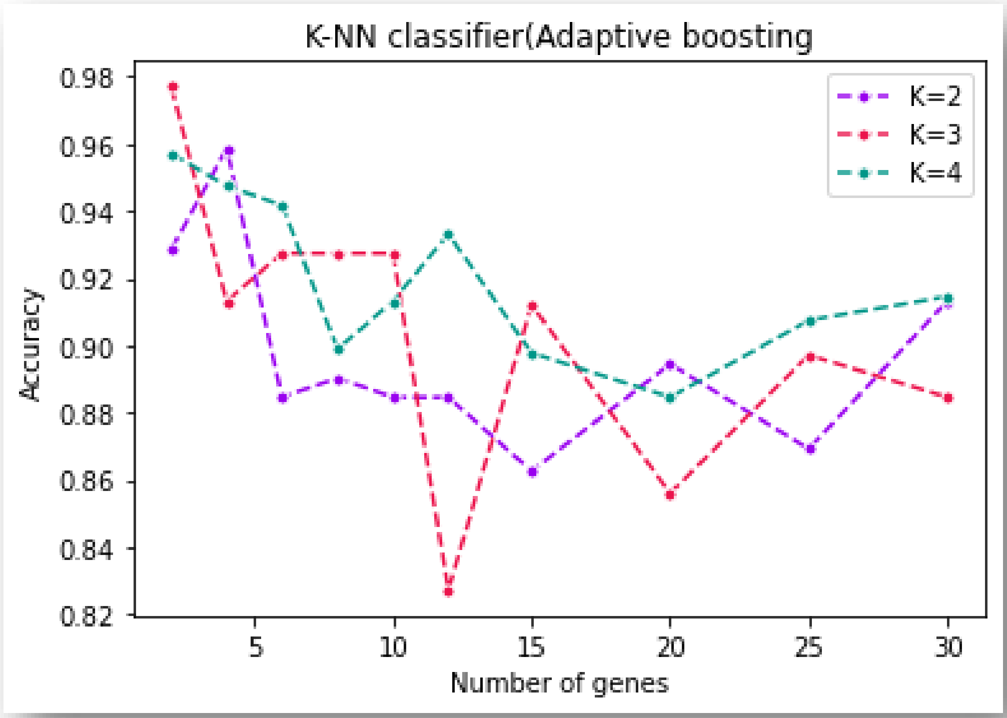
DECISION TREE (ADAPTIVE BOOSTING)



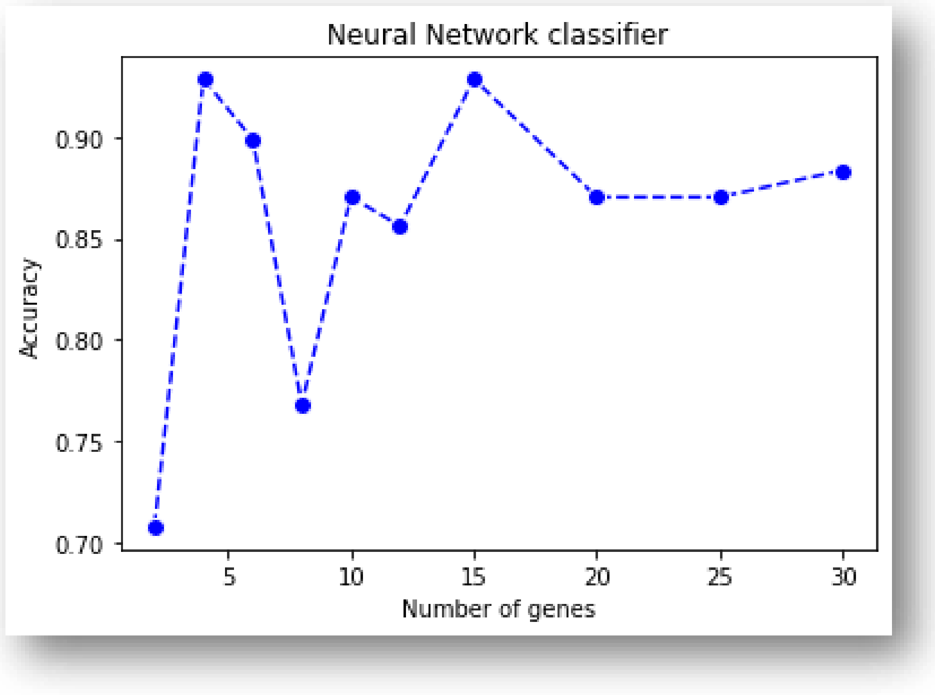


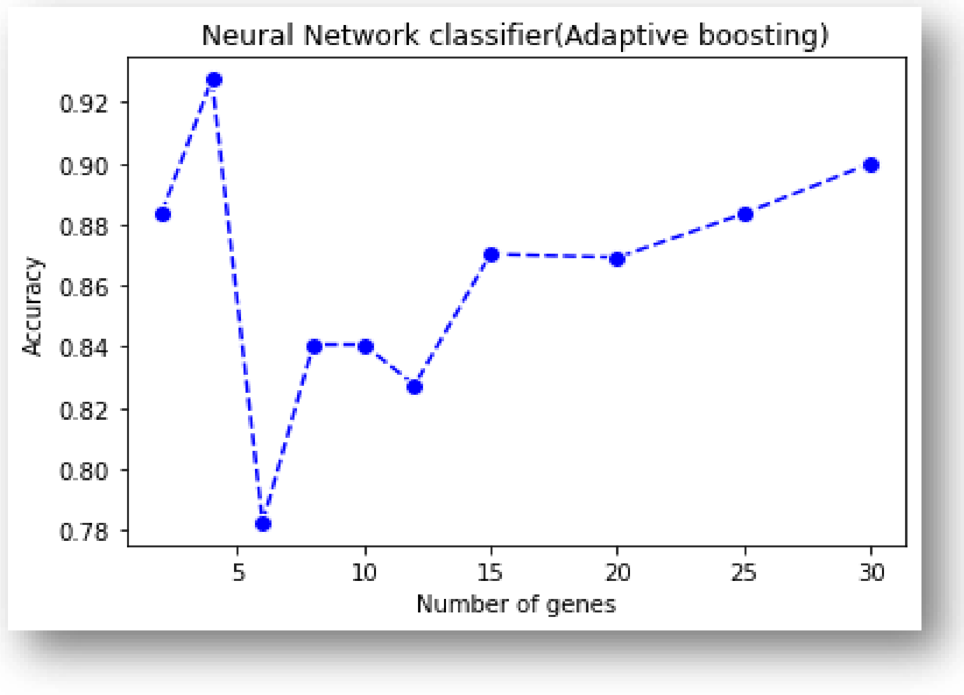
KNN CLASSIFIER(ADAPTIVE BOOSTING)



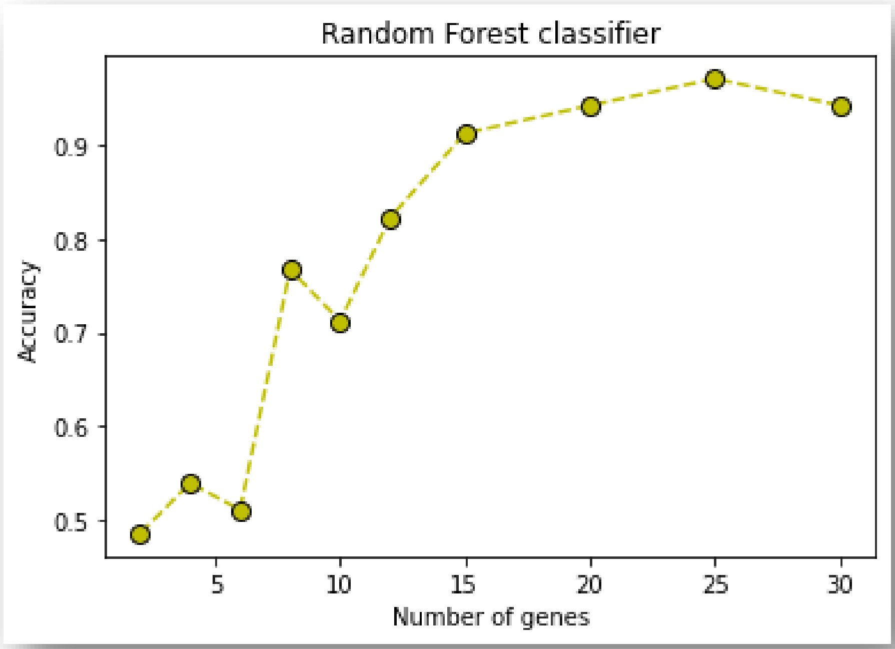


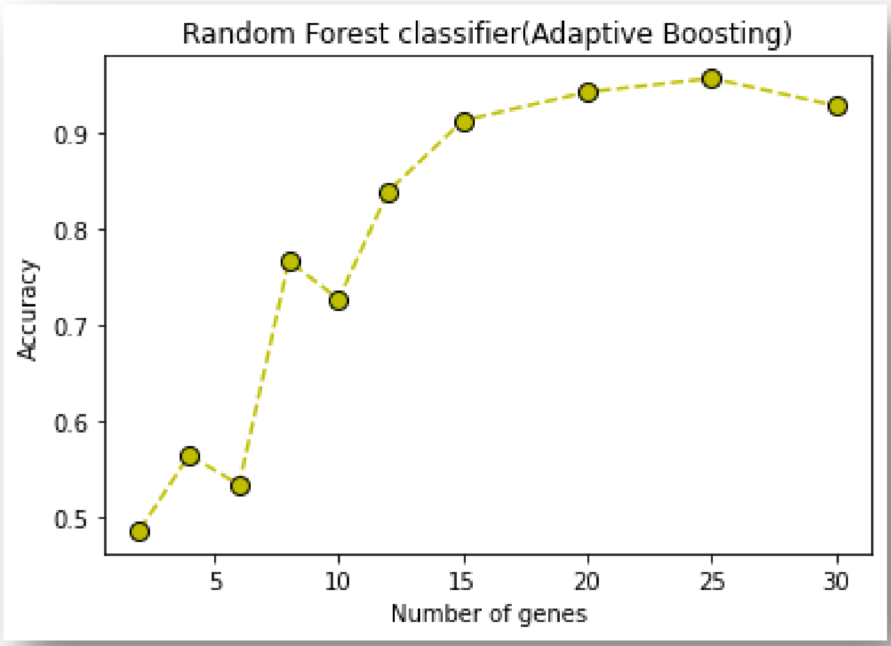
NEURAL NETWORK(ADAPTIVE BOOSTING)





RANDOM FOREST CLASSIFIER(ADAPTIVE BOOSTING)





REFRENCES

- **3.1. Cross-validation: evaluating estimator performance — scikit-learn 1.1.1 documentation**
- **Ojala and Garriga. Permutation Tests for Studying Classifier Performance. J. Mach. Learn. Res. 2010.**
- **https://www.geeksforgeeks.org/ml-extra-tree-classifier-for-feature-selection/#:~:text=Extremely%20Randomized%20Trees%20Classifier(Extra,to%20output%20it's%20classification%20result.**
- **Abdulla, M., and Khasawneh, M. T. (2020). G-Forest: an ensemble method for cost-sensitive feature selection in gene expression microarrays. *Artif. Intell. Med.* 108:101941. doi: 10.1016/j.artmed.2020.101941**
- **Aldamassi, M., Chen, Z., Merriman, B., Gussin, D., Nelson, S.: A Practical Guide to Microarray Analysis of Gene Expression. UCLA Microarray Core & Nelson Lab, UCLA Department of Human Genetics (2001)**
- **Freund, Y., Mason, L.: The alternating decision tree learning algorithm. In: Sixteenth International Conference on Machine Learning, Bled, Slovenia, pp. 124–133 (1999)**