CS 590 Challenge1: Data Mining Project

Predict disease classes using genetic microarray data

Author: Ye Cheng

Project Description

This project will compare the data of a disease on a gene with the data of other diseases on this gene to calculate the T value. Therefore, each gene will have five t-values, and then we will sort the t-values of each disease and select the genes related to disease comparison. Finally, this project will compare the relative advantages and disadvantages of classifiers, and give a chart to show the relationship between error rate and gene number. This project will use python, sklearn, and Jupiter notebook to complete the task.

Specific steps

Step 1: Data Cleaning

Use the dropna() statement to delete missing values.

Step 1. Data Cleaning

In [1]:
```
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.utils.multiclass import unique_labels
from statistics import mean
import statistics

datas = pd.read_csv('./pp5i_train.gr.csv')
datas = datas.dropna()
#datas.head()
```

In [2]:
```
for i in range(1,70):
    ii = str(i)
    #datas[ii].to_list()
    #datas[ii].values.astype(int)
    datas[ii] = np.clip(datas[ii],a_min = 20,a_max = 16000)
#datas = datas.replace([19,16000],np.nan)
#datas = datas.replace([19,16000],0)
datas = datas.dropna()
```

Step 2. Selecting top genes by class

First, we set the maximum and minimum values of the data (16000, 20). If it is not within the data range, we will set it to the extreme value (16000 or 20). Second, we divide the maximum and minimum values of each gene to get the folding differences. Then, we delete genes with folding difference less than 2 from the data. Finally, we have 6413 genes left.

Step 2. Selecting top genes by class

Remove from train data genes with fold differences

In [3]:
```
max_elements = datas.max(axis=1)
#print(max_elements)

min_elements = datas.min(axis=1)
#print(min_elements)
fold = max_elements/min_elements
#print(fold)
```

In [4]:
```
lowfold = []
for i,v in fold.items():
    if(v<2):
        #print(i,v)
        lowfold.append(i)
for i in lowfold:
    datas = datas.drop([i],axis=0)
print(datas.shape)
#datas = datas.loc[datas.max(axis=1) -datas.min(axis=1) > 2]

datas = datas.reset_index(drop=True)    #resetindex
```

```
(6413, 70)
```

Then we will find out the patient index corresponding to each disease.

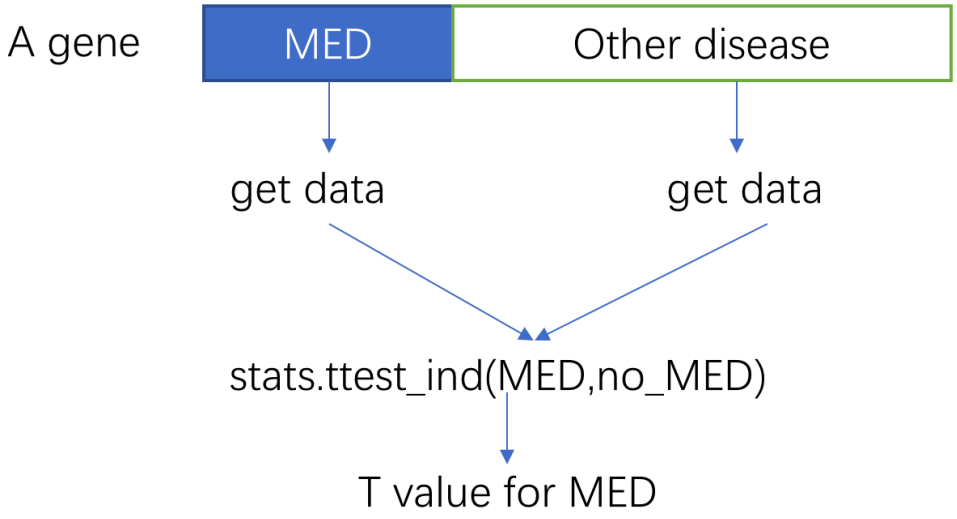
First, we extract all diseases (class_name = unique_labels (classes)).

generate subsets with top class

In [5]:
```
train_class = open('pp5i_train_class.txt')
#classes.read()
classes = []
for i in train_class:
    classes.append(i)

'''
with open('pp5i_train.gr.csv', 'a') as f:
    writer = csv.writer(f)
    writer.writerow(classes)
'''
class_name = unique_labels(classes)
EPD_index = []
JPA_index = []
MED_index = []
MGL_index = []
RHB_index = []


for i in range(1,len(classes)):

    if(classes[i] == 'EPD\n'):
        EPD_index.append(i)
    if(classes[i] == 'JPA\n'):
        JPA_index.append(i)
    if(classes[i] == 'MED\n'):
        MED_index.append(i)
    if(classes[i] == 'MGL\n'):
        MGL_index.append(i)
    if(classes[i] == 'RHB\n'):
        RHB_index.append(i)

print(datas)
```

Then we will compare the index of specific patients with other patients to calculate the T value (stats.ttest_ind). This will get the T value of the disease on this gene. The following figure shows an example of finding the T value of MED. In the same way, we can get the T value of other diseases (EPD, JPA, MGL, and RHB).

:
```
row_elements1 = []

for index, row in datas.iterrows():
    #print(index) # Output the index value of each row
    row_elements = []
    row_elements = datas.iloc[index]
    row_elements = row_elements.tolist()
    row_elements1.append(row_elements)

EPD_T = []
JPA_T = []
MED_T = []
MGL_T = []
RHB_T = []

#MED
for value in range(0,len(row_elements1)):
    MED = []
    no_MED = []
    for i in range(1,len(row_elements1[value])):

        if(row_elements1[value][i] !=16000):
            exist_count = MED_index.count(i)
            if exist_count > 0:
                MED.append(row_elements1[value][i])
            else:
                no_MED.append(row_elements1[value][i])

    t,p = stats.ttest_ind(MED,no_MED)
    MED_T.append(abs(t))
```



Then we rank the T values of each disease and select top 2, 4, 6, 8, 10, 12, 15, 20, 25, and 30 top genes.

TOP2

In [72]:
```
datas['EPD_T'] = EPD_T
datas['JPA_T'] = JPA_T
datas['MED_T'] = MED_T
datas['MGL_T'] = MGL_T
datas['RHB_T'] = RHB_T

EPD_top2 = datas.sort_values('EPD_T',ascending=False)
EPD_top2 = EPD_top2.head(2)
EPD_top2 = EPD_top2['SNO']
EPD_top2 = EPD_top2.tolist()
EPD_top2=pd.DataFrame(EPD_top2)

JPA_top2 = datas.sort_values('JPA_T',ascending=False)
JPA_top2 = JPA_top2.head(2)
JPA_top2 = JPA_top2['SNO']
JPA_top2 = JPA_top2.tolist()
JPA_top2=pd.DataFrame(JPA_top2)

MED_top2 = datas.sort_values('MED_T',ascending=False)
MED_top2 = MED_top2.head(2)
MED_top2 = MED_top2['SNO']
MED_top2 = MED_top2.tolist()
MED_top2=pd.DataFrame(MED_top2)

MGL_top2 = datas.sort_values('MGL_T',ascending=False)
MGL_top2 = MGL_top2.head(2)
MGL_top2 = MGL_top2['SNO']
MGL_top2 = MGL_top2.tolist()
MGL_top2=pd.DataFrame(MGL_top2)

RHB_top2= datas.sort_values('RHB_T',ascending=False)
RHB_top2 = RHB_top2.head(2)
RHB_top2 = RHB_top2['SNO']
RHB_top2 = RHB_top2.tolist()
RHB_top2=pd.DataFrame(RHB_top2)

dftop2 = pd.concat((EPD_top2,JPA_top2,MED_top2,MGL_top2,RHB_top2),axis=0,join='inner')
#print(dftop2)

top2 = []

for i in range(0,10):
    if i <2:
        top2.append('EPD')
    elif i <4:
        top2.append('JPA')
    elif i <6:
        top2.append('MED')
    elif i <8:
        top2.append('MGL')
    elif i <10:
        top2.append('RHB')

#print(top2)
dftop2['Class'] = top2
#print(dftop2)
dftop2 = dftop2.T
dftop2.to_csv('pp5i_train.top2.gr.csv',header=None)
```

we will examine the data to see if there is a duplicate top gene. After testing, we didn't find a duplicate gene (set_row30=set(row30)).

:
```
T30 = pd.read_csv('./pp5i_train.top30.gr.csv')
row30 = T30.columns #
print(row30)

set_row30=set(row30)
#Set generates an iteratable object whose elements are unordered and not repeated
if len(set_row30)==len(row30):
    print('No Duplicate elements in the list')
else:
    print('Duplicate elements in the list')
```

```
Index(['SNO', 'X03363_s_at', 'X05908_at', 'U15131_at', 'L49054_at',
       'M31516_s_at', 'U53204_at', 'L25878_s_at', 'U90913_at', 'X72964_at',
       ...
       'X13839_at', 'L14565_at', 'L11353_at', 'X54304_at', 'M23114_at',
       'U21858_at', 'X95325_s_at', 'U72263_s_at', 'U96131_at', 'M24069_at'],
      dtype='object', length=151)
No Duplicate elements in the list
```

At the end of the second step, we will get pp5i_ train. topN. Gr.csv file.

pp5i_train.top2.gr

0	X03363_s_at	X05908_at	U00921_at	AF000424_s_at	S82470_at	U16954_at	U52155_at	M61156_at	D83174_s_at	U47621_at
Class	EPD	EPD	JPA	JPA	MED	MED	MGL	MGL	RHB	RHB

Step 3. Find the best classifier/best gene set combination

First, we extract the topN gene from the test set as required.

Extraction of top gene from test set

In [123]:
```
datas = pd.read_csv('./pp5i_train.top2.gr.csv')
tests = pd.read_csv('./pp5i_test.gr.csv')

xx = datas.columns
xx = np.array(xx)
xx = xx.tolist()
del(xx[0])

allgenename = []
allgenename = tests['SNO']
#print(allgenename)
tests = tests.T
tests.columns = allgenename
tests.drop(index=tests.index[0],axis=0,inplace=True)

#i, j = np.where(tests.values == 'Z78285_f_at')
#print(tests)
```

In [124]:
```
#add top gene to test1

test1 = pd.DataFrame()

for i in xx:
    test1[i] = tests[i]
#print(test1)
test1.shape
```

Out[124]: (23, 10)

Then divide the data into two parts: train and test. x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2). This will be used for cross validation later.

]:
```
x=datas.head(69)
x = x.drop('SNO',axis=1)
#print(x)

train_class = open('pp5i_train_class.txt')
#classes.read()
classes = []
for i in train_class:
    classes.append(i.replace("\n", ""))
classes = np.array(classes)
classes = classes.tolist()
del(classes[0])
#print(classes)
y = classes

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
best_accuracy = []
average_error_rate = []
test_predict = []
```

Five classifiers will be used in this project (Naïve Bayes, Decision tree, K–NN, Neural network, and Linear classifier). We will use sklearn to

implement (GaussianNB(), tree.DecisionTreeClassifier(), KNeighborsClassifier(n_neighbors=2,3,4), MLPClassifier(), SGDClassifier()).

Naïve Bayes

In [46]:
```
clf=GaussianNB()

clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print("y_predict:",y_predict)
print("")
print("actual y_test:",y_test)
print("")
accuracy = ((y_predict!=y_test)==0).sum()/len(y_test)
print('accuracy: ',accuracy)
print("")
error_rate = 1-accuracy
print('error_rate: ',error_rate)

best_accuracy.append(accuracy)
average_error_rate.append(error_rate)
print("")

clf=GaussianNB()
clf.fit(x, y)
y_predict = clf.predict(test1)
print("y_predict1 for test set:",y_predict)
test_predict.append(y_predict)
```

```
y_predict: ['RHB' 'RHB' 'MED' 'MGL' 'MED' 'RHB' 'MGL' 'EPD' 'MED' 'MED' 'RHB' 'RHB'
 'MED' 'MED']

actual y_test: ['EPD', 'MED', 'MED', 'MGL', 'MED', 'EPD', 'MGL', 'EPD', 'MED', 'MED', 'RHB', 'RHB', 'MED', 'MED']

accuracy:  0.7857142857142857

error_rate:  0.2142857142857143

y_predict1 for test set: ['EPD' 'EPD' 'JPA' 'EPD' 'EPD' 'EPD' 'JPA' 'EPD' 'EPD' 'JPA' 'JPA' 'EPD'
 'EPD' 'EPD' 'EPD' 'EPD' 'JPA' 'EPD' 'JPA' 'EPD' 'EPD' 'JPA' 'EPD']
```

Decision tree

In [6]:
```
Dec = tree.DecisionTreeClassifier()

Dec.fit(x_train, y_train)
y_predict = Dec.predict(x_test)
print("y_predict:",y_predict)
print("")
print("actual y_test:",y_test)
print("")
accuracy1 = ((y_predict!=y_test)==0).sum()/len(y_test)
print('accuracy: ',accuracy1)
print("")
error_rate1 = 1-accuracy1
print('error_rate: ',error_rate1)

best_accuracy.append(accuracy1)
average_error_rate.append(error_rate1)
print("")

Dec = tree.DecisionTreeClassifier()
Dec.fit(x, y)
y_predict1 = Dec.predict(test1)
print("y_predict1 for test set:",y_predict1)
test_predict.append(y_predict1)
```

```
y_predict: ['JPA' 'RHB' 'JPA' 'MED' 'MED' 'JPA' 'MED' 'MED' 'RHB' 'RHB' 'MED' 'JPA'
 'MED' 'MED']

actual y_test: ['JPA', 'RHB', 'JPA', 'MED', 'MED', 'JPA', 'RHB', 'MED', 'MED', 'RHB', 'MED', 'JPA', 'MED', 'MED']

accuracy:  0.8571428571428571

error_rate:  0.1428571428571429

y_predict1 for test set: ['MGL' 'JPA' 'MED' 'MED' 'EPD' 'MED' 'MED' 'MED' 'EPD' 'JPA' 'JPA' 'MED'
 'MED' 'MED' 'MED' 'MED' 'MGL' 'MED' 'MED' 'RHB' 'RHB' 'MED' 'MGL']
```

K-nn

In [48]:
```
knn = KNeighborsClassifier(n_neighbors=2)
#knn = KNeighborsClassifier(n_neighbors=2,algorithm='brute')

knn.fit(x_train, y_train)
y_predict = knn.predict(x_test)
print("y_predict:",y_predict)
print("")
print("actual y_test:",y_test)
print("")
accuracy2 = ((y_predict!=y_test)==0).sum()/len(y_test)
print('accuracy: ',accuracy2)
print("")
error_rate2 = 1-accuracy2
print('error_rate: ',error_rate2)

best_accuracy.append(accuracy2)
average_error_rate.append(error_rate2)
print("")

knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(x, y)
y_predict2 = knn.predict(test1)
print("y_predict2 for test set:",y_predict2)
test_predict.append(y_predict2)
```

```
y_predict: ['MED' 'MED' 'MED' 'MGL' 'EPD' 'EPD' 'MGL' 'EPD' 'MED' 'MED' 'MED' 'RHB'
 'MED' 'MED']

actual y_test: ['EPD', 'MED', 'MED', 'MGL', 'MED', 'EPD', 'MGL', 'EPD', 'MED', 'MED', 'RHB', 'RHB', 'MED', 'MED']

accuracy:  0.7857142857142857

error_rate:  0.2142857142857143

y_predict2 for test set: ['MED' 'EPD' 'MED' 'MED' 'EPD' 'MED' 'MED' 'MED' 'EPD' 'JPA' 'JPA' 'MED'
 'MED' 'MED' 'MED' 'MED' 'JPA' 'MED' 'MED' 'RHB' 'EPD' 'MED' 'EPD']
```

neural network

In [51]:
```
neu = MLPClassifier()
#neu = MLPClassifier(solver='adam')

neu.fit(x_train, y_train)
y_predict = neu.predict(x_test)
print("y_predict:",y_predict)
print("")
print("actual y_test:",y_test)
print("")
accuracy5 = ((y_predict!=y_test)==0).sum()/len(y_test)
print('accuracy: ',accuracy5)
print("")
error_rate5 = 1-accuracy5
print('error_rate: ',error_rate5)

best_accuracy.append(accuracy5)
average_error_rate.append(error_rate5)
print("")

neu = MLPClassifier()
neu.fit(x, y)
y_predict5 = neu.predict(test1)
print("y_predict5 for test set:",y_predict5)
test_predict.append(y_predict5)
```

```
y_predict: ['RHB' 'MED' 'MED' 'MGL' 'MED' 'EPD' 'MGL' 'EPD' 'MED' 'MED' 'RHB' 'RHB'
 'MED' 'MED']

actual y_test: ['EPD', 'MED', 'MED', 'MGL', 'MED', 'EPD', 'MGL', 'EPD', 'MED', 'MED', 'RHB', 'RHB', 'MED', 'MED']

accuracy:  0.9285714285714286

error_rate:  0.0714285714285714

y_predict5 for test set: ['MED' 'EPD' 'MED' 'MED' 'EPD' 'MED' 'MED' 'MED' 'EPD' 'JPA' 'JPA' 'MED'
 'MED' 'MED' 'MED' 'MED' 'EPD' 'MED' 'MED' 'EPD' 'EPD' 'MED' 'EPD']
```

Linear classifier

In [52]:
```
lin = SGDClassifier()

lin.fit(x_train, y_train)
y_predict = lin.predict(x_test)
print("y_predict:",y_predict)
print("")
print("actual y_test:",y_test)
print("")
accuracy6 = ((y_predict!=y_test)==0).sum()/len(y_test)
print('accuracy: ',accuracy6)
print("")
error_rate6 = 1-accuracy6
print('error_rate: ',error_rate6)

best_accuracy.append(accuracy6)
average_error_rate.append(error_rate6)
print("")

lin = SGDClassifier()
lin.fit(x, y)
y_predict6 = lin.predict(test1)
print("y_predict6 for test set:",y_predict6)
test_predict.append(y_predict6)
```

```
y_predict: ['RHB' 'MED' 'MED' 'MGL' 'RHB' 'RHB' 'MGL' 'EPD' 'MED' 'MED' 'RHB' 'RHB'
 'MED' 'MED']

actual y_test: ['EPD', 'MED', 'MED', 'MGL', 'MED', 'EPD', 'MGL', 'EPD', 'MED', 'MED', 'RHB', 'RHB', 'MED', 'MED']

accuracy:  0.7857142857142857

error_rate:  0.2142857142857143

y_predict6 for test set: ['MED' 'EPD' 'MED' 'MED' 'EPD' 'MED' 'MED' 'MED' 'EPD' 'JPA' 'JPA' 'MED'
 'MED' 'MED' 'MED' 'MED' 'MGL' 'MED' 'MED' 'EPD' 'EPD' 'MED' 'EPD']
```

We also modified the classifier with 1–2 additional relevant parameters.

The error rate has been reduced to some extent, but significantly. The

average error rate is almost constant (less than 0.1 change)

Gene 8

Before Change

```
[0.19000000000000003, 0.19714285714285715, 0.05071428571428569, 0.02928571428571427, 0.03928571428571428, 0.09, 0.071
42857142857142]

min_index 3
```

After Change

```
[0.1792857142857143, 0.20857142857142857, 0.06, 0.03142857142857142, 0.03857142857142856, 0.09928571428571427, 0.06]

min_index 3
```

```
#Dec = tree.DecisionTreeClassifier(min_samples_split=0.1)
Dec = tree.DecisionTreeClassifier()
```

```
knn = KNeighborsClassifier(n_neighbors=2)
#knn = KNeighborsClassifier(n_neighbors=2,algorithm='brute')
```

```
neu = MLPClassifier()
#neu = MLPClassifier(solver='adam')
```

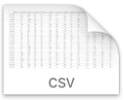
Step 4. Generate predictions for the test set

Now we have the best train files and test files. The following figures are examples of these documents


pp5i_train.best2.csv


pp5i_train.best4.csv


pp5i_train.best6.csv

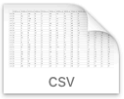

pp5i_train.best8.csv


pp5i_train.best10.csv


pp5i_train.best12.csv


pp5i_train.best15.csv


pp5i_train.best20.csv


pp5i_train.best25.csv


pp5i_train.best30.csv

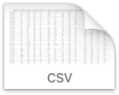

pp5i_test.best2.csv


pp5i_test.best4.csv


pp5i_test.best6.csv

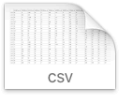

pp5i_test.best8.csv


pp5i_test.best10.csv


pp5i_test.best12.csv


pp5i_test.best15.csv


pp5i_test.best20.csv


pp5i_test.best25.csv


pp5i_test.best30.csv

pp5i_train.best2

SNO	X03363_s_at	X05908_at	U00921_at	AF000424_s_at	S82470_at	U16954_at	U52155_at	M61156_at	D83174_s_at	U47621_at
1	20	77	20	20	66	931	20	20	189	68
2	29	43	20	20	69	845	20	20	152	33
3	28	20	20	20	71	1107	20	20	95	20
4	22	20	20	20	74	1187	24	20	208	46
5	24	20	20	20	93	969	41	20	124	20
6	33	20	20	20	99	1172	34	20	96	20

67	39	147	115	102	199	313	32	23	166	37
68	41	147	102	94	190	287	30	20	149	20
69	45	160	119	92	203	269	34	20	121	20
EPD_	12.711130572	11.8037165	0.70523016	0.813696392421	4.08846955	4.04465948	1.58789598	1.14683483	0.3427708492	1.88745755
JPA_1	0.2197404502	0.62374656	30.1304510	25.98589706107	3.55588852	1.90931080	0.12614366	0.55201821	1.3337073642	0.95769654
MED_	5.3269907595	5.17112426	3.17460863	3.051065013044	10.9769001	10.7910749	2.61640752	2.44185885	3.2347023197	4.43185691
MGL_	1.1864824566	0.95012842	0.94414269	0.847023343620	1.55118053	2.35291276	15.1081523	10.90812900	1.1159189285	1.37476020
RHB_	0.6883506346	0.30136297	0.75822124	0.809377081907	1.99856169	2.60594179	1.18192094	0.87174455	12.663023642	10.6857780
Class	EPD	EPD	JPA	JPA	MED	MED	MGL	MGL	RHB	RHB

pp5i_test.best2

	X03363_s_at	X05908_at	U00921_at	AF000424_s_at	S82470_at	U16954_at	U52155_at	M61156_at	D83174_s_at	U47621_at	Class
101	18	-4	13	-14	89	566	79	21	202	-54	MED
102	90	383	25	-4	142	296	35	17	214	25	EPD
103	24	17	-7	-69	118	989	7	-3	219	-146	MED
104	8	-5	12	-26	59	1151	43	13	387	-17	MED
105	217	736	16	-11	120	35	33	10	316	4	EPD
106	-8	12	20	-31	-21	512	58	20	187	-106	MED
107	22	11	3	3	62	707	25	32	322	19	MED
108	27	2	8	-7	111	1080	36	12	91	6	MED
109	107	640	42	5	235	128	30	16	342	40	EPD
110	32	51	78	61	180	369	40	21	120	11	JPA
111	48	157	131	75	211	334	42	20	126	25	JPA
112	41	2	6	-9	68	1190	47	11	132	-13	MED
113	23	31	7	-20	66	959	10	16	119	39	MED
114	5	-12	10	-18	22	912	38	12	137	-43	MED
115	11	40	4	-16	-660	1182	26	10	176	-51	MED
116	-7	4	1	-39	26	529	21	13	556	-42	MED
117	10	153	17	-21	192	429	51	15	223	-12	MGL
118	21	148	10	-8	83	992	9	19	215	62	MED
119	16	28	3	-19	87	1348	1	-15	331	74	MED
120	47	1291	18	-16	196	569	18	29	2125	115	EPD
121	75	747	8	-13	225	167	3	21	906	54	EPD
122	15	11	0	-43	37	651	56	10	146	-44	MED
123	28	208	19	0	241	99	59	28	630	126	EPD

Step 5. Generate a prediction using Adaptive Boosting

Boosting means promotion Its function is to improve the last training every time. In the process of training, there are dependencies between the K classifiers. When the k–th classifier is introduced, it is actually the optimization of the first k–1 classifier. The AdaBoost algorithm is a way of relearning It was proposed by Freund and others in 1995. It is an implementation of boosting algorithm. This kind of algorithm trains

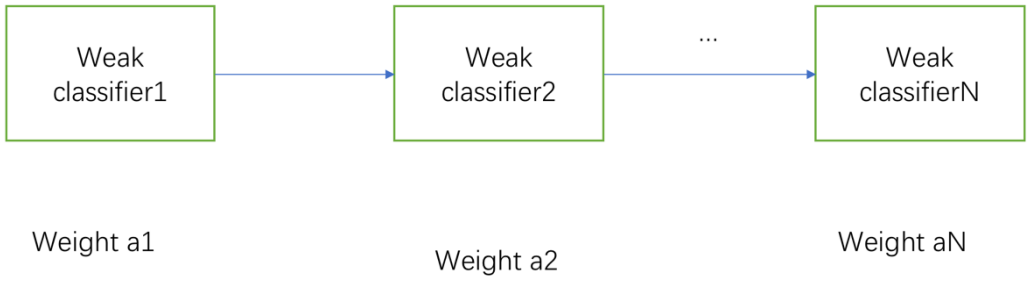
multiple weak classifiers and combines them into a strong classifier. Weak classifiers are easy to train, but strong classifiers are not easy to train.

The algorithm is actually a simple weak classification algorithm promotion process, which can improve the classification ability of data through continuous training. The whole process is as follows:

1. The first weak classifier is obtained by learning n training samples;
2. The wrong prediction samples and other new data together form a new N training samples, and the second weak classifier is obtained by learning this sample;
3. Add the wrong prediction samples of 1 and 2 and other new samples to form another new N training sample, and obtain the third weak classifier through the learning of this sample;
4. Finally, the improved strong classifier. That is, the classification of a data is determined by the weight of each classifier.

We can use multiple weak classifiers to combine a strong classifier, so how to combine them? These weak classifiers are combined according to different weights. If the classification effect of weak classifier is good, the weight should be relatively small. If the classification effect of weak classifier is general, the weight should be increased. In this way, the wrong prediction data will have higher weight, so when training the next

classifier, focus on the wrong prediction samples of the previous predictor. Suppose we only carry out three rounds of training, select three weak classifiers and combine them into a strong classifier, then the final strong classifier.



This project uses AdaBoost classifier to apply adaptive boosting.

AdaboostM1

In [39]:
```
ada = AdaBoostClassifier(base_estimator= Dec,
                         n_estimators=500,
                         learning_rate=0.1,
                         random_state=0)
ada.fit(x_train, y_train)
y_predict = ada.predict(x_test)
print("y_predict:",y_predict)
print("")
print("actual y_test:",y_test)
print("")
accuracy7 = ((y_predict!=y_test)==0).sum()/len(y_test)
print('accuracy: ',accuracy7)
print("")
error_rate7 = 1-accuracy7
print('error_rate: ',error_rate7)

#best_accuracy.append(accuracy7)
#average_error_rate.append(error_rate7)
print("")

ada = AdaBoostClassifier(base_estimator= Dec,
                         n_estimators=500,
                         learning_rate=0.1,
                         random_state=0)
ada.fit(x, y)
y_predict7 = ada.predict(test1)
print("y_predict7 for test set:",y_predict7)
test_predict.append(y_predict7)
```

```
y_predict: ['EPD' 'RHB' 'MGL' 'MED' 'MED' 'JPA' 'MED' 'MED' 'MED' 'EPD' 'MED' 'MED'
 'MED' 'MED']

actual y_test: ['EPD', 'RHB', 'MGL', 'MED', 'MED', 'JPA', 'MED', 'MED', 'MED', 'MED', 'MED', 'MED', 'MED', 'MED']

accuracy:  0.9285714285714286

error_rate:  0.0714285714285714

y_predict7 for test set: ['MED' 'EPD' 'MED' 'MED' 'EPD' 'MED' 'MED' 'MED' 'EPD' 'MGL' 'JPA' 'MED'
 'MED' 'MED' 'MED' 'MED' 'MGL' 'MED' 'EPD' 'EPD' 'EPD' 'MED' 'MGL']
```

Strengths and weaknesses

Naïve Bayes:

Naive Bayes algorithm assumes that the attributes of data sets are independent of each other, so the logic of the algorithm is very simple, and the algorithm is relatively stable. When the relationship between the attributes of the data set is relatively independent, the naive Bayesian classification algorithm will have a better effect.

The condition of attribute independence is also the deficiency of naive Bayesian classifier. In many cases, the independence of dataset attributes is difficult to meet, because the attributes of dataset are often interrelated. If this problem occurs in the classification process, the classification effect will be greatly reduced.

Decision tree:

Decision tree is simple and intuitive, and the generated decision tree is very intuitive. There is basically no need for preprocessing and normalization in advance to deal with missing values It has good fault tolerance and high robustness for outliers.

The decision tree algorithm is very easy to over fit, resulting in weak generalization ability. It can be improved by setting the minimum sample number of nodes and limiting the depth of decision tree. For some complex relationships, the decision tree is difficult to learn, such as XOR.

K-NN:

The decision tree is simple and intuitive, and the generated decision tree is very intuitive. There is basically no need for preprocessing and normalization in advance to deal with missing values It has good fault tolerance and high robustness for outliers.

The decision tree algorithm is very easy to overfit, resulting in weak generalization ability. It can be improved by setting the minimum sample number of nodes and limiting the depth of the decision tree. For some complex relationships, the decision tree is difficult to learn, such as XOR.

Neural network:

The main advantage of the neural network is that its performance is better than most other machine learning algorithms

The most well-known disadvantage of neural networks is the "black box" nature, which means you don't know how and why neural networks produce a certain output. At the same time, neural networks usually need more data.

Linear classifier:

The algorithm is simple. The linear classifier is fast and easy to program, but the fitting effect may not be very good. The prediction is not accurate enough.

Project results

We use sklearn to implement (GaussianNB(), tree.DecisionTreeClassifier(), KNeighborsClassifier(n_neighbors=n), MLPClassifier(), SGDClassifier()) this project.

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import AdaBoostClassifier
```

One training and test cannot explain the error rate of the model, so we go through 100 training data and conduct cross-validation.

```
error_rate = []

for i in range(100):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
    clf=GaussianNB()

    clf.fit(x_train, y_train)
    y_predict = clf.predict(x_test)
    accuracy = ((y_predict!=y_test)==0).sum()/len(y_test)
    error_rate.append(1-accuracy)



average_error_rate.append(np.mean(error_rate))
```

We will select the lowest error rate of each gene data and the model it represents. We write the data of each gene N into a CSV file. The effect of using the k-NN (k = 3) model in this project is better than other models. As shown in the figure. The model name refers to the model with the lowest error rate when the gene is N. Ave err rate is the average error rate of this model.

summary

```
]: print(average_error_rate)
   print("")


   min_index = average_error_rate.index(min(average_error_rate))
   print("min_index",min_index)
   print("")
```

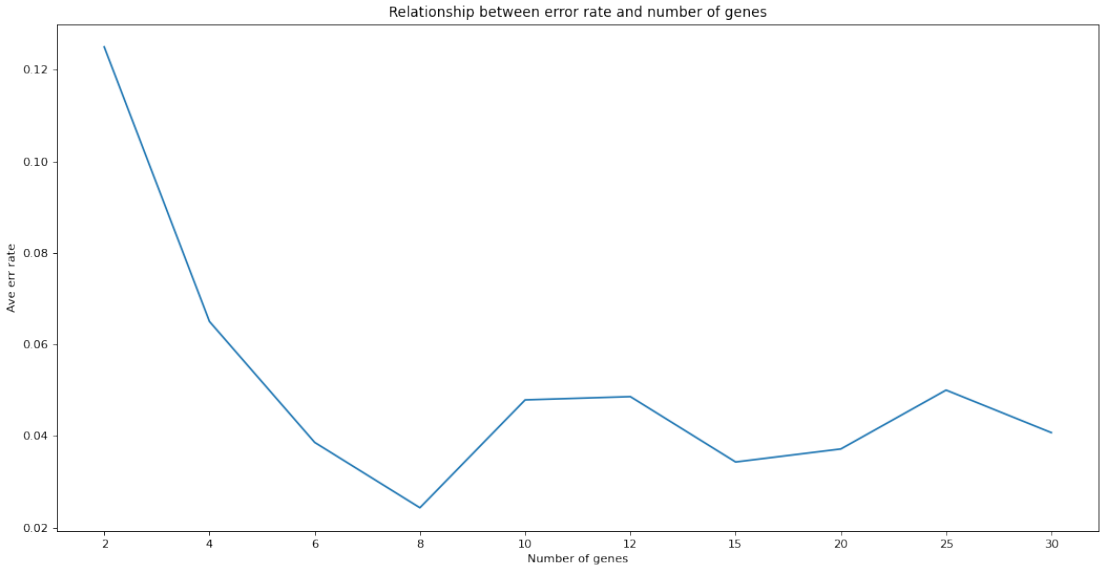
```
[0.19928571428571432, 0.14785714285714285, 0.12499999999999999, 0.06499999999999999, 0.07285714285714284, 0.117142857
14285712, 0.0914285714285714]

min_index 3
```

```
]: summary = pd.read_csv('./summary.csv')
   summary['4'] = round(np.min(average_error_rate), 5)

   name = ['Naïve Bayes','Decision tree','K-nn2','K-nn3','K-nn4','Neural network','Linear classifier']

   summary.loc[1:,'4'] = name[min_index]

   summary.to_csv("./summary.csv", index=False, sep=',')
```

summary

Unnamed: 0	2	4	6	8	10	12	15	20	25	30
ave err rate	0.125	0.065	0.03857	0.02429	0.04786	0.04857	0.03429	0.03714	0.05	0.04071
model name	Neural network	K-nn3	K-nn3	K-nn3	K-nn2	K-nn2	K-nn4	K-nn3	K-nn3	K-nn3

The following figure shows the relationship between the error rate and the number of genes.



We tend to select eight genes for each disease. If there are not enough genes, the prediction is inaccurate. If there are too many genes, irrelevant data will also interfere with the prediction results.

Reference

HAYES, Adam. (2022, March 12). *T-Test*. Investopedia. Retrieved May 21, 2022, from https://www.investopedia.com/terms/t/t-test.asp

AdaBoost. (n.d.). Wikipedia. Retrieved May 21, 2022, from https://en.wikipedia.org/wiki/AdaBoost

Supervised learning. (n.d.). Scikit-Learn. Retrieved May 21, 2022, from https://scikit-learn.org/stable/supervised_learning.html#supervised-learning