In [ ]:
```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import scale
import statsmodels.api as sm
import matplotlib.pyplot as plt
from pandas.plotting import andrews_curves
from pandas.plotting import radviz
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

Load data

In [ ]:
```python
train=pd.read_csv("spam-train.txt",header=None)
test=pd.read_csv("spam-test.txt",header=None)
train.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52	5
0	0.00	0.0	0.00	0.00	0.00	0.00	0.0	0.00	0.00	0.00	...	0.000	0.610	0.000	0.203	0.000	0
1	0.00	0.0	0.59	0.11	0.00	0.00	0.0	0.00	0.11	0.23	...	0.227	0.322	0.113	0.056	0.075	0
2	0.06	0.0	0.40	0.00	0.13	0.13	0.0	0.13	0.00	0.00	...	0.028	0.085	0.000	0.000	0.000	0
3	0.00	0.0	0.00	0.00	0.00	0.00	0.0	0.00	0.00	0.00	...	0.000	0.000	0.000	0.000	0.000	0
4	0.00	0.0	0.00	0.00	0.00	0.44	0.0	0.00	0.00	0.00	...	0.000	0.150	0.000	0.000	0.000	0

5 rows × 58 columns

1) Standardize the columns so that they all have zero mean and unit variance

In [ ]:
```python
x_train1=scale(train.iloc[:,:-1])
y_train=train.iloc[:,-1]
x_test1=scale(test.iloc[:,:-1])
y_test=test.iloc[:,-1]
```

In [ ]:
```python
x_train1.std(axis=0) # check whether unit variance
```

Out[ ]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1.])
```

2) Transform the features using $log(x_{ij} + 1)$

In [ ]:
```python
x_train2=np.log(x_train1+1)
x_test2=np.log(x_test1+1)
```

3) Discretize each feature

In [ ]:

```
x_train3=(x_train2>0)*1
x_test3 =(x_test2>0)*1
```

a) visualize data

In [ ]:

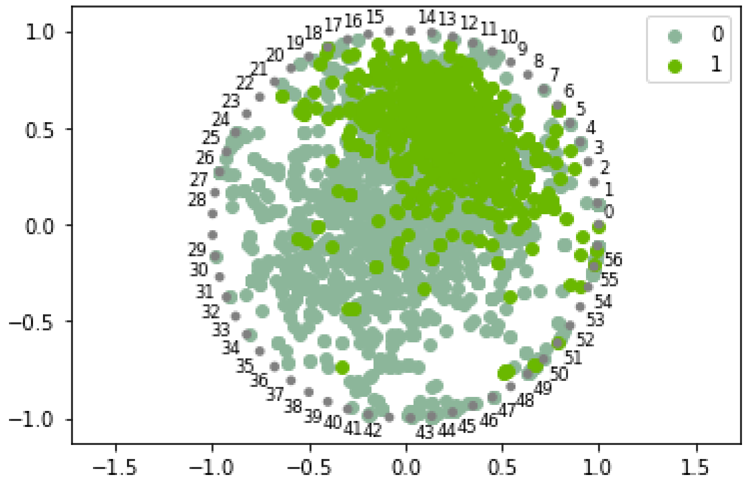
```
import warnings
warnings.filterwarnings("ignore")
```

Standardized data diagram

In [ ]:

```
data1=pd.DataFrame(x_train1)
data1["label"]=y_train
radviz(data1, 'label')
```

Out[ ]: <AxesSubplot:>

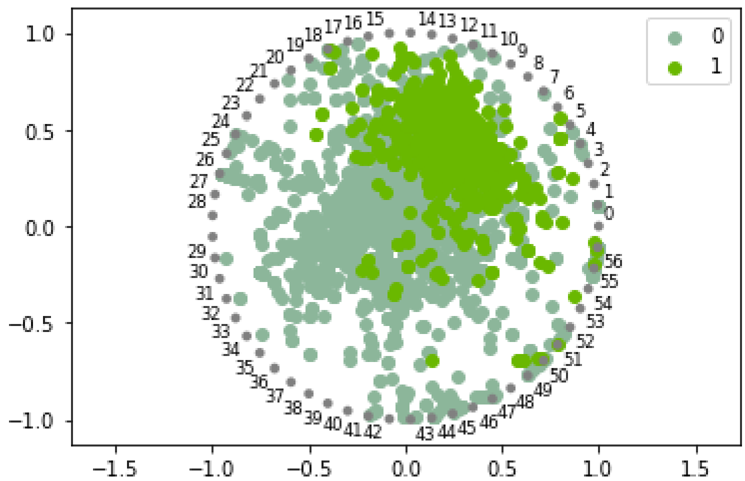


Log-transformed data diagram

In [ ]:

```
data2=pd.DataFrame(x_train2)
data2["label"]=y_train
radviz(data2, 'label')
```

Out[ ]: <AxesSubplot:>



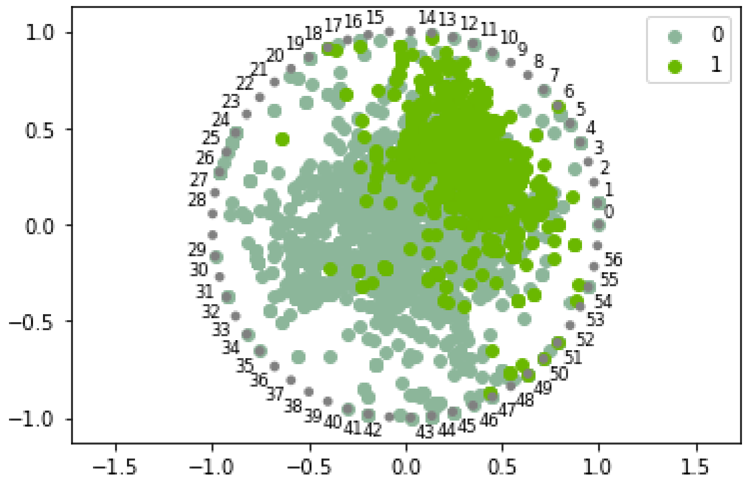
Discretized data diagram

In [ ]:

```
data3=pd.DataFrame(x_train3)
data3["label"]=y_train
```

```python
radviz(data3, 'label')
```

Out[ ]: <AxesSubplot:>



b) fit logistic model

i) fit on standardized data

In [ ]:
```python
# fit on standardized data
X=sm.add_constant(x_train1)
y=y_train
logit_data1=sm.Logit(y,X).fit()

# error rate on train set
logit_train_pred1=(logit_data1.predict(train)>0.5).astype(int)
logit_train_error1=sum(logit_train_pred1!=y_train)/len(y_train)
print(logit_train_error1)

# error rate on test set
logit_test_pred1=(logit_data1.predict(test)>0.5).astype(int)
logit_test_error1=sum(logit_test_pred1!=y_test)/len(y_test)
print(logit_test_error1)

# Significant features: p<0.05 is defined as significant
logit_data1.pvalues[logit_data1.pvalues<0.05].index[1:]
```

```
Optimization terminated successfully.
         Current function value: 0.188692
         Iterations 15
0.5500489077274209
0.5345501955671447
```

Out[ ]:
```
Index(['x4', 'x5', 'x7', 'x8', 'x9', 'x11', 'x16', 'x17', 'x19', 'x20', 'x21',
       'x23', 'x25', 'x27', 'x42', 'x44', 'x45', 'x46', 'x47', 'x49', 'x52',
       'x53', 'x55', 'x56', 'x57'],
      dtype='object')
```

ii) fit on log data

In [ ]:
```python
# fit on log data
X=sm.add_constant(x_train2)
y=y_train
logit_data2=sm.Logit(y,X).fit()

# error rate on train set
```

```
logit_train_pred2=(logit_data2.predict(train)>0.5).astype(int)
logit_train_error2=sum(logit_train_pred2!=y_train)/len(y_train)
print(logit_train_error2)

# error rate on test set
logit_test_pred2=(logit_data2.predict(test)>0.5).astype(int)
logit_test_error2=sum(logit_test_pred2!=y_test)/len(y_test)
print(logit_test_error2)

# Significant features: p<0.05 is defined as significant
logit_data1.pvalues[logit_data2.pvalues<0.05].index[1:]
```

```
Optimization terminated successfully.
         Current function value: 0.154490
         Iterations 14
0.527551353113792
0.5176010430247718
```

Out[ ]:

```
Index(['x5', 'x7', 'x8', 'x11', 'x13', 'x15', 'x16', 'x17', 'x18', 'x20',
       'x21', 'x23', 'x24', 'x25', 'x27', 'x28', 'x35', 'x37', 'x42', 'x43',
       'x44', 'x45', 'x46', 'x52', 'x53', 'x57'],
      dtype='object')
```

iii) fit on discretize data

In [ ]:

```
# fit on discretize data
X=sm.add_constant(x_train3)
y=y_train
logit_data3=sm.Logit(y,X).fit()

# error rate on train set
logit_train_pred3=(logit_data3.predict(train)>0.5).astype(int)
logit_train_error3=sum(logit_train_pred3!=y_train)/len(y_train)
print(logit_train_error3)

# error rate on test set
logit_test_pred3=(logit_data3.predict(test)>0.5).astype(int)
logit_test_error3=sum(logit_test_pred3!=y_test)/len(y_test)
print(logit_test_error3)

# Significant features: p<0.05 is defined as significant
logit_data1.pvalues[logit_data3.pvalues<0.05].index[1:]
```

```
Optimization terminated successfully.
         Current function value: 0.173849
         Iterations 12
0.46723182262797525
0.4485006518904824
```

Out[ ]:

```
Index(['x5', 'x7', 'x8', 'x10', 'x13', 'x15', 'x16', 'x17', 'x18', 'x20',
       'x21', 'x22', 'x23', 'x24', 'x25', 'x26', 'x27', 'x28', 'x31', 'x33',
       'x35', 'x37', 'x42', 'x44', 'x45', 'x46', 'x52', 'x53', 'x54', 'x55',
       'x57'],
      dtype='object')
```

c) fit LDA/QDA

LDA method

In [ ]:

```
# fit on standardized data
lda = LinearDiscriminantAnalysis()
lda.fit(x_train1, y)
```

```
# error rate on train set
lda_train_pred1=lda.predict(x_train1)
lda_train_error1=sum(lda_train_pred1!=y_train)/len(y_train)
print(lda_train_error1)

# error rate on test set
lda_test_pred1=lda.predict(x_test1)
lda_test_error1=sum(lda_test_pred1!=y_test)/len(y_test)
print(lda_test_error1)
```

```
0.10172807303553962
0.10299869621903521
```

In [ ]:

```
# fit on log data
lda = LinearDiscriminantAnalysis()
lda.fit(x_train2, y)

# error rate on train set
lda_train_pred2=lda.predict(x_train2)
lda_train_error2=sum(lda_train_pred2!=y_train)/len(y_train)
print(lda_train_error2)

# error rate on test set
lda_test_pred2=lda.predict(x_test2)
lda_test_error2=sum(lda_test_pred2!=y_test)/len(y_test)
print(lda_test_error2)
```

```
0.07042712748614281
0.07627118644067797
```

QDA method

In [ ]:

```
# fit on standardized data
qda = QuadraticDiscriminantAnalysis()
qda.fit(x_train1, y)

# error rate on train set
qda_train_pred1=qda.predict(x_train1)
qda_train_error1=sum(qda_train_pred1!=y_train)/len(y_train)
print(qda_train_error1)

# error rate on test set
qda_test_pred1=qda.predict(x_test1)
qda_test_error1=sum(qda_test_pred1!=y_test)/len(y_test)
print(qda_test_error1)
```

```
0.17867623084447343
0.17470664928292046
```

In [ ]:

```
# fit on log data
qda = QuadraticDiscriminantAnalysis()
qda.fit(x_train2, y)

# error rate on train set
qda_train_pred2=qda.predict(x_train2)
qda_train_error2=sum(qda_train_pred2!=y_train)/len(y_train)
print(qda_train_error2)

# error rate on test set
qda_test_pred2=qda.predict(x_test2)
qda_test_error2=sum(qda_test_pred2!=y_test)/len(y_test)
print(qda_test_error2)
```

```
0.15194000652103032
0.14993481095176012
```

d) fit svm

linear support vector machine

In [ ]:
```
# fit on standardized data
Svm =svm.SVC(kernel='linear')
Svm.fit(x_train1, y)

# error rate on train set
svm_train_pred1=Svm.predict(x_train1)
svm_train_error1=sum(svm_train_pred1!=y_train)/len(y_train)
print(svm_train_error1)

# error rate on test set
svm_test_pred1=Svm.predict(x_test1)
svm_test_error1=sum(svm_test_pred1!=y_test)/len(y_test)
print(svm_test_error1)
```

```
0.06488425171177047
0.07170795306388526
```

In [ ]:
```
# fit on standardized data
Svm =svm.SVC(kernel='linear')
Svm.fit(x_train2, y)

# error rate on train set
svm_train_pred2=Svm.predict(x_train2)
svm_train_error2=sum(svm_train_pred2!=y_train)/len(y_train)
print(svm_train_error2)

# error rate on test set
svm_test_pred2=Svm.predict(x_test2)
svm_test_error2=sum(svm_test_pred2!=y_test)/len(y_test)
print(svm_test_error2)
```

```
0.056080860776002606
0.06258148631029987
```

In [ ]:
```
# fit on discretize data
Svm =svm.SVC(kernel='linear')
Svm.fit(x_train3, y)

# error rate on train set
svm_train_pred3=Svm.predict(x_train3)
svm_train_error3=sum(svm_train_pred3!=y_train)/len(y_train)
print(svm_train_error3)

# error rate on test set
svm_test_pred3=Svm.predict(x_test3)
svm_test_error3=sum(svm_test_pred3!=y_test)/len(y_test)
print(svm_test_error3)
```

```
0.06423214867949135
0.07301173402868318
```

non-linear support vector machine

In [ ]:
```
# fit on standardized data
```

```
Svm =svm.SVC(kernel='rbf')
Svm.fit(x_train1, y)

# error rate on train set
n_svm_train_pred1=Svm.predict(x_train1)
n_svm_train_error1=sum(svm_train_pred1!=y_train)/len(y_train)
print(n_svm_train_error1)

# error rate on test set
n_svm_test_pred1=Svm.predict(x_test1)
n_svm_test_error1=sum(n_svm_test_pred1!=y_test)/len(y_test)
print(n_svm_test_error1)
```

```
0.06488425171177047
0.06453715775749674
```

In [ ]:

```
# fit on log data
Svm =svm.SVC(kernel='rbf')
Svm.fit(x_train2, y)

# error rate on train set
n_svm_train_pred2=Svm.predict(x_train2)
n_svm_train_error2=sum(svm_train_pred2!=y_train)/len(y_train)
print(n_svm_train_error2)

# error rate on test set
n_svm_test_pred2=Svm.predict(x_test2)
n_svm_test_error2=sum(n_svm_test_pred2!=y_test)/len(y_test)
print(n_svm_test_error2)
```

```
0.056080860776002606
0.04954367666232073
```

In [ ]:

```
# fit on discretize data
Svm =svm.SVC(kernel='rbf')
Svm.fit(x_train3, y)

# error rate on train set
n_svm_train_pred3=Svm.predict(x_train3)
n_svm_train_error3=sum(svm_train_pred3!=y_train)/len(y_train)
print(n_svm_train_error3)

# error rate on test set
n_svm_test_pred3=Svm.predict(x_test3)
n_svm_test_error3=sum(n_svm_test_pred3!=y_test)/len(y_test)
print(n_svm_test_error3)
```

```
0.06423214867949135
0.05345501955671447
```

e) decision tree classifier

In [ ]:

```
# fit on standardized data
dc = RandomForestClassifier(max_depth=6)
dc.fit(x_train1, y_train)

# error rate on train set
dc_train_pred1=dc.predict(x_train1)
dc_train_error1=sum(dc_train_pred1!=y_train)/len(y_train)
print(dc_train_error1)

# error rate on test set
dc_test_pred1=dc.predict(x_test1)
```

```
dc_test_error1=sum(dc_test_pred1!=y_test)/len(y_test)
print(dc_test_error1)
```

```
0.051516139550048905
0.05410691003911343
```

In [ ]:

```
# fit on log data
dc = RandomForestClassifier(max_depth=6)
dc.fit(x_train2, y_train)

# error rate on train set
dc_train_pred2=dc.predict(x_train2)
dc_train_error2=sum(dc_train_pred2!=y_train)/len(y_train)
print(dc_train_error2)

# error rate on test set
dc_test_pred2=dc.predict(x_test2)
dc_test_error2=sum(dc_test_pred2!=y_test)/len(y_test)
print(dc_test_error2)
```

```
0.05086403651776981
0.052803129074315516
```

In [ ]:

```
# fit on log data
dc = RandomForestClassifier(max_depth=6)
dc.fit(x_train3, y_train)

# error rate on train set
dc_train_pred3=dc.predict(x_train3)
dc_train_error3=sum(dc_train_pred3!=y_train)/len(y_train)
print(dc_train_error3)

# error rate on test set
dc_test_pred3=dc.predict(x_test3)
dc_test_error3=sum(dc_test_pred3!=y_test)/len(y_test)
print(dc_test_error3)
```

```
0.07042712748614281
0.07496740547588006
```

Report classification errors

In [ ]:

```
Methods=["Logit","LDA","QDA","SVM-linear","SVM-nonlinear","Decision Tree"]
Methods = [val for val in Methods for _ in range(3)]
Data=["standardized","log-transformed","Discretized"]*6
Train_error_rate=[logit_train_error1,logit_train_error2,logit_train_error3,
                    lda_train_error1,  lda_train_error2,             "---",
                    qda_train_error1,  qda_train_error2,             "---",
                    svm_train_error1,  svm_train_error2,  svm_train_error3,
                  n_svm_train_error1,n_svm_train_error2,n_svm_train_error3,
                     dc_train_error1,   dc_train_error2,   dc_train_error3]
Test_error_rate=[logit_test_error1,logit_test_error2,logit_test_error3,
                   lda_test_error1,  lda_test_error2,            "---",
                   qda_test_error1,  qda_test_error2,            "---",
                   svm_test_error1,  svm_test_error2,  svm_test_error3,
                 n_svm_test_error1,n_svm_test_error2,n_svm_test_error3,
                    dc_test_error1,   dc_test_error2,   dc_test_error3]
result=pd.DataFrame({"Methods":Methods,
                    "Data":Data,
                    "Train_error_rate":Train_error_rate,
                    "Test_error_rate":Test_error_rate})
result
```

Out[ ]:

	Methods	Data	Train_error_rate	Test_error_rate
0	Logit	standardized	0.550049	0.53455
1	Logit	log-transformed	0.527551	0.517601
2	Logit	Discretized	0.467232	0.448501
3	LDA	standardized	0.101728	0.102999
4	LDA	log-transformed	0.070427	0.076271
5	LDA	Discretized	---	---
6	QDA	standardized	0.178676	0.174707
7	QDA	log-transformed	0.15194	0.149935
8	QDA	Discretized	---	---
9	SVM-linear	standardized	0.064884	0.071708
10	SVM-linear	log-transformed	0.056081	0.062581
11	SVM-linear	Discretized	0.064232	0.073012
12	SVM-nonlinear	standardized	0.064884	0.064537
13	SVM-nonlinear	log-transformed	0.056081	0.049544
14	SVM-nonlinear	Discretized	0.064232	0.053455
15	Decision Tree	standardized	0.051516	0.054107
16	Decision Tree	log-transformed	0.050864	0.052803
17	Decision Tree	Discretized	0.070427	0.074967

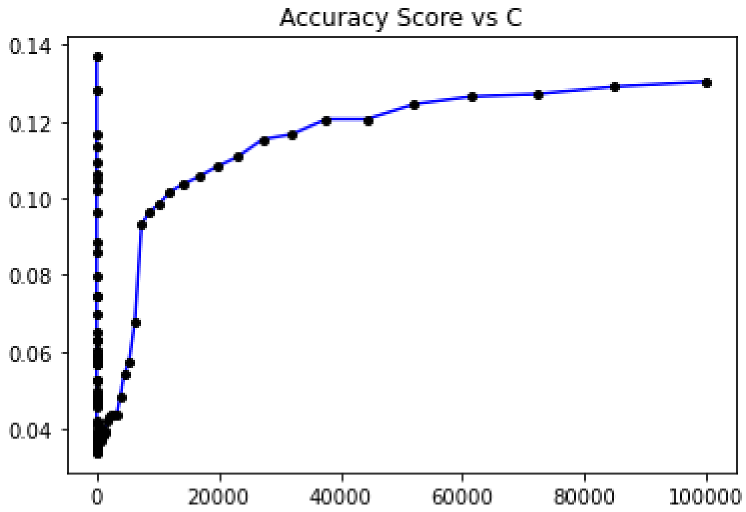
In the table above, we see the nonlinear support vector machine on log-transformed data performs best, which has the lowest test error rate and also fits well on training data. The worst model is logit model, the test error rate is approximately close to 0.5.

Finally, we choose nonlinear support vector machine on log-transformed data as our best model. Then, we try different soft margin param C, and find the optimal option to make the test error rate as small as possible.

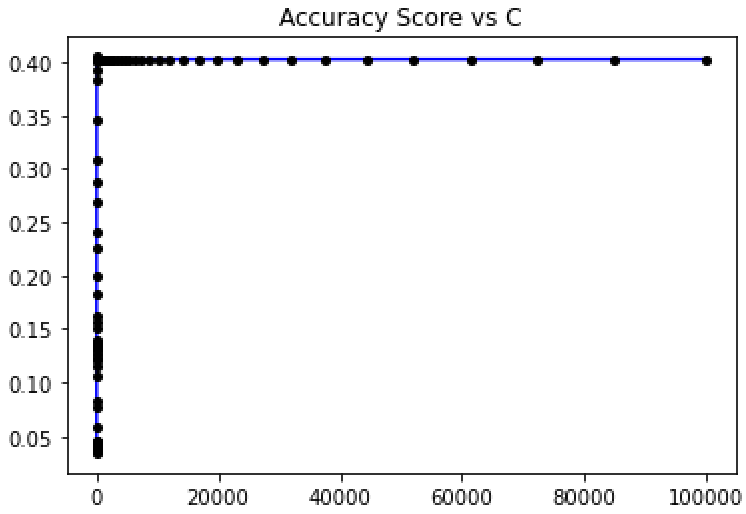
In [ ]:

```
#Find best C
scores_list = []
C_list = 10**np.linspace(-2,5,100)
for C_val in C_list:
    model = svm.SVC(kernel='rbf', C=C_val)
    model.fit(x_train2,y_train)
    pred=model.predict(x_test2)
    test_error=sum(pred!=y_test)/len(y_test)
    scores_list.append(test_error)

plt.plot(C_list, scores_list,  color = 'blue', marker = '.', markersize = 8,
         markeredgecolor = 'black', markerfacecolor = 'black',label = 'Score')
plt.title('Accuracy Score vs C')
plt.show()
```

```
index = np.argmin(np.array(scores_list))
C_best = C_list[index]
print('The best C is ', C_best)

#Find best gamma
scores_list = []
gamma_list = 10**np.linspace(-2, 5, 100)
for g_val in gamma_list:
    model = svm.SVC(kernel='rbf', C=C_best, gamma=g_val)
    model.fit(x_train2, y_train)
    pred = model.predict(x_test2)
    test_error = sum(pred != y_test)/len(y_test)
    scores_list.append(test_error)

plt.plot(gamma_list, scores_list,  color='blue', marker='.', markersize=8,
         markeredgecolor='black', markerfacecolor='black', label='Score')
plt.title('Accuracy Score vs C')
plt.show()

index = np.argmin(np.array(scores_list))
g_best = gamma_list[index]
print('The best gamma is ', g_best)
```




```
The best C is  77.4263682681127
```




```
The best gamma is  0.05994842503189409
```

The best hyperparameter C for SVM model is 77.42. And best gamma is 0.05995.

In [ ]:
```
model = svm.SVC(kernel='rbf', C=C_best,gamma=g_best)
model.fit(x_train2,y_train)
pred=model.predict(x_test2)
```

```python
test_error=sum(pred!=y_test)/len(y_test)
test_error
```

Out[ ]:

```
0.03455019556714472
```

Now the test error rate has been decreased to 3.46%.It does a good job!

Everyone contributed equally.