Applying Ensemble Method on Cleveland Heart Disease Data Set*

1st Yihang Li
Mathematics
University of Central Florida
Orlando, America
lyh970828@Knights.ucf.edu

***Abstract*—Based on Synthpop in R, we expand our data set. By using Pipeline and GridSearchCV in Python, we build some baseline classifiers. After combing them in VotingClassifier, an ensemble method, we build our final model.**

***Index Terms*—Heart Disease, Synthpop, Baseline Models, Ensemble method**

I. Introduction

A. Background

Heart disease is the leading cause of death for both men and women in United States. More than 600,000 Americans die of heart disease each year. That's one in every four deaths in this country. [1]

The term "heart disease" refers to several types of heart conditions. The most common type is coronary heart disease(CHD), which can cause heart attack. Without early detection and clinical intervention, studies show that almost half of the patients diagnosed with CHD will eventually die of the disease. [2]

Worth mentioning, CHD is largely preventable, as many of its risk factors are modifiable. These include tobacco smoking, high blood pressure, high blood cholesterol, physical inactivity, poor nutrition, and obesity.

B. Literature Review

Prediction of heart disease has become a well studied topic. After reviewing several literature, the related data mining techniques can be summarised as Table I below:

TABLE I
Literature Review Summary

Authors/Time	Data Set	Methods	Accuracy
K. Srinivas et al. [3] 2010	Heart-statlog Heart-c/Heart-h	Naive Bayes ODANB	About 80%
Shouman et al. [4] 2012	Cleveland Heart Disease	Decision Tree with K-Means	About 83.9%
El-Bialy et al. [2] 2015	Coronary Artery Heart Disease	C4.5 fast Decision Trees	About 70%

C. Challenges in Medical Data Sets

As we know, applying data mining techniques often require large data sets. However, medical data sets are usually small due to the patient privacy laws and the maintaining and protection among most hospitals, clinics and healthcare centers.

D. Cleveland heart disease data set

This article is based on Cleveland heart disease data set [5], one of the data sets of Heart Disease Database available from UCI repository.

The Cleveland data set has been used most among the Heart Disease Database because of its completeness with only a few missing values. Moreover, there are two version of this data set, one is the raw data set contains 76 attributes, but most of them are useless due to missing or undefined data, another one is the processed data set, which is comma delimited and has been reduced to 14 attributes by El-Bialy et al. [2]. Here we choose the latter one, the processed version, with only 6 missing values coded with "?". Table II is the information about variables (5 of them are continuous and 9 of them are discrete).

E. Our Goal

Facing the Challenge above, our goal is mainly about three parts :

1) To expand the size of Cleveland data set;
2) To build some baseline data mining models with it;
3) To combine these models together by using the ensemble method .

II. Feature Engineering

In this part, we deal with missing data and do some data exploration. Most importantly, we then try to expand the data set in the R studio.

A. Data preprocess

After inspecting the data, we found the 6 missing value coded with "?" only exist in the variable named 'ca' and 'thal'. Since in the R language, the missing value is only expressed as NA, we first use the $revalue()$ function to substitute the question mark as NA, and then use the $na.omit()$ function to delete the whole rows containing the missing value. At last, there are 297 individuals with 14 variables.

TABLE II
THE INFORMATION OF CLEVELAND DATA SET VARIABLES

Attribute	Information
age	Age of patient(in years; continuous)
sex	Gender (1 = male, 0 = female; discrete)
cp	Chest pain type. (1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic; discrete)
rbp	Rest blood presure (in mmHg; continuous)
chol	Total cholesterol (mg/dl; continous)
fbs	Fast blood sugar(more than 120 mg/dl, where 1 = true, 0 = false; discrete)
restecg	Resting electrocardiographic results. (0: normal, 1: having ST-T wave abnormality, 2: left ventricular hypertrophy; discrete)
max_hra	Maximum heart rate achieved (continuous)
eia	Exercise induced angina (1 = yes, 0 = no; discrete)
oldpeak	ST depression relative to rest (continuous)
slope	The slope of the peak exercise ST segment (1: upsloping, 2: flat, 3: downsloping; discrete)
ca	Number of major vessels (0-3) (discrete)
thal	Thalassemia (3 = normal, 6 = fixed defect, 7 = reversible defect; discrete)
diag	Whether the individual is suffering from heart disease or not (0 = absence, 1, 2, 3, 4 = present.)

Moreover, for the variable diag, as explained in the Table II, there are 5 levels of the individuals' heart disease condition. Among thoses, 0 indicates negtive result (health), and 1, 2, 3, 4 indicate positive result (the individual has heart disease). In order to simplify the classification, we use the $revalue()$ function to code 2, 3, 4 as 1. In the end, there are only two levels, 0 and 1.

B. Data Exploration

As mentioned above, there are 5 continuous variables and 9 discrete variables (include the target class : diag).

1) Continuous Part: Fig 1 is the pairplot of those 5 continuous variables separated by diag. Here diag = 0 with red color implies that the person is not suffering from heart disease and diag = 1 with blue color indiactes the person is suffering from. For more details, the position (1, 1) (Here, the first 1 is the first row, the second 1 is the first column, the followings are the same) of this pairplot is the distribution of age for each diag level, which indicates the older the age, the more possible getting heart disease. Also, we can see the the separate condition among these variables. For example, the position (3, 1) is separated better than the position (2, 1).

2) Discrete Part: For the variable sex, 0 = female and 1 = male. Fig 2 is the boxplot of the age based on both sex and diag. We see that for most of females who are suffering from heart disease are older than the males. Moreover, Fig 3 is the heatmap of these 9 discrete variables. Obviously, there is no single variable that has a very high correlation with the target result, diag.

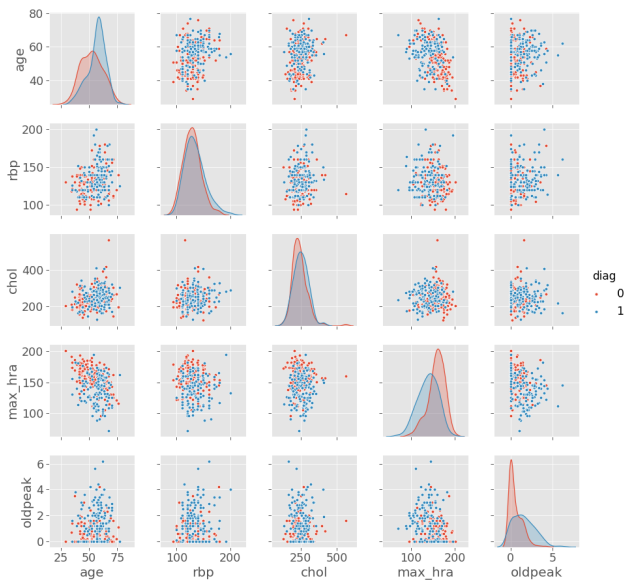

Fig. 1. Pairplot of 5 continuous variables

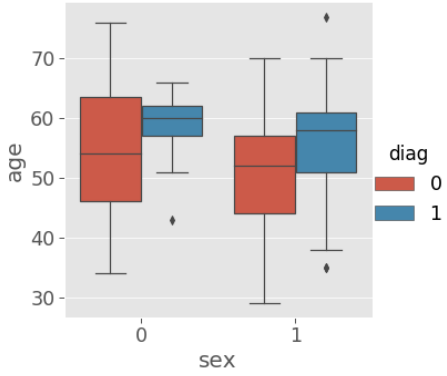

Fig. 2. The boxplot of age based on sex

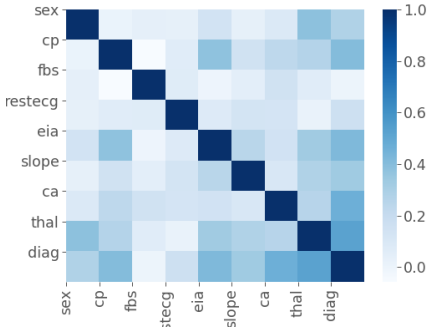

Fig. 3. Correlation Heatmap of 9 Discrete Variables

C. Expand Data Set

There are several ways to expand our limit data set such as using the bootstrap method, which is a resampling technique used to estimate statistics on a population by sampling a data set with replacement. Alternatively, a better choice is to use the synthetic data mentioned as surrogate data by (A.Sabay et al.(2018)) [6]. Also, this part mainly follows that article.

1) Synthetic Data: the basic idea of which is to replace some or all of the original observed data by sampling from appropriate probability distributions and preserve the essential statistical features [7]. Worth mentioning here, the use of synthetic data can anonymize sensitive data such as clinic data set.

2) Tool Used: In R language, Synthpop is a library that provides functions about data synthesis and comparison.

3) Generate Synthetic Data: Here we use syn() function to complete this step. The only thing we need to do is to set data = cle, m = 100, seed = 19970828. In the end, we generate 29700 individuals, which is 100 times compared to the original data set. Fig 4 is the comparison between synthetic data and original data by using compare() function.

Fig. 4. The Comparison between Original and Synthetic Data

D. Other Process

1) Data Standardization: There two reasons for us to do this step : On the one hand, these variables have different unit, on the other hand, most machine learning methods require us to scale data. Thus, we use $StandardScaler()$ from $python$ package $sklearn$ to standardize our data.

2) Train Test Split: In this step, we split the original data set into training set and testing set. The former one is used to build our baseline model while the latter one is used to test the model we build. Based on Python, from sklearn.model selection import $train\_test\_split()$ to complete this step, here we set the testsize = 0.3, and random state= 7 (random seed).

Be careful here, for one kind of models, we will parallelly train on the original training data and the expanded data. Finally, testing both of them on the original testing data.

III. Baseline Models

For this part, We use the $pipeline()$ from $sklearn$ [8], a $python$ package, to build classifiers as our baseline models. For some of them that have parameters to be tuned, we use $GridSearchCV()$ function to complete that. Finally, we summarize the results of them.

A. Pipeline and GridsearchCV

Pipeline can combine all feature prepossess and modelling into one object while gridsearchcv can test all the assumptions and parameters to find out which combination generates the best result.

B. Models to build

1) Logistic Regression: In this step, the function we used to build our first classifier is $LogisticRegression()$. Besides, there are two parameters we try to tune by using GridSearchCV, they are the regularization penalty method (L1 and L2) and penalty value C (Inverse of regularization strength, must be a positive float. Smaller values specify stronger regularization). After combing pipeline with and without GridSearchCV, we get:

- For original data set, the test accuracy with GridSearchCV is 0.856 while without is 0.844. In the case of GridSearchCV, the best penalty is L2 and the best C is 0.568987.
- For expanded data set, the test accuracy with GridSearchCV is 0.867 while without is 0.856. In the case of GridSearchCV, the best penalty is L1 and the best C is 0.009103.

2) Linear Discriminant Analysis: In this step, we use $LinearDiscriminantAnalysis()$ to build our classifier. Here we just build the classifiers parallelly on the original data set and expanded data set. And, we get:

- Test accuracy (original): 0.856
- Test accuracy (expanded): 0.867

3) K Nearest Neighbors: In this step, the function we used to build our another classifier is $KNeighborsClassifier()$. The same way, we need to tune the parameter K by using GridSearchCV. After combing pipeline with and without GridSearchCV, we get:

- For original data set, the test accuracy with GridSearchCV is 0.833 while without is 0.811. In the case of GridSearchCV, the best K is 8.
- For expanded data set, the test accuracy with GridSearchCV is 0.833 while without is 0.844. In the case of GridSearchCV, the best K is 9.
  Be careful here, by using the best K, our result is less acurracy instead of more. What happened? This is because we gridsearch only on the data that are used to build classifier. And please keep in mind, the testing data is unseen before we test the classifier. Beside, by default, this classifier choose $K = 5$, our gridsearch result is $K = 9$. Larger K means high bias and low variance, this can reasonably expain the result here.

4) Decision Tree: In this step, we build our forth classifier, $DecisionTreeClassifier()$. Here we try to tune two parameters by using GridSearchCV, they are the criterion (gini and entropy) and max_depth, the maximum depth of the tree. After combing pipeline with and without GridSearchCV, we get:

- For original data set, the test accuracy with GridSearchCV is 0.0.733 while without is 0.711. In the case of GridSearchCV, the best criterion is gini and the best max_depth is 5.
- For expanded data set, the test accuracy with GridSearchCV is 0.856 while without is 0.778. In the case of GridSearchCV, the best criterion and the best max_depth is the same as the original one.

5) Support Vector Machine: In this step, the function we used to build our last classifier is $SVC()$.There are two parameters we try to tune by using GridSearchCV, they are the kernel (linear, poly, rbf and sigmoid) and penalty value C. After combing pipeline with and without GridSearchCV, we get:

- For original data set, the test accuracy with GridSearchCV is 0.844 while without is 0.789. In the case of GridSearchCV, the best kernel is linear and the best C is 0.004292.
- For expanded data set, the test accuracy with GridSearchCV is 0.867 while without is 0.878. In the case of GridSearchCV, the best kernel is L1 and the best C is 0.009103.

Finally, let us summarize the result of them as Table III as follows:

TABLE III
TEST ACCURACY RESULTS OF BASELINE MODELS

Data Set	Logistic	LDA	KNN	Decision Tree	SVC
Original	0.844	0.856	0.811	0.711	0.789
(GridSearch)	0.856	None	0.833	0.733	0.844
Expanded	0.867	0.867	0.844	0.778	0.867
(GridSearch)	0.856	None	0.833	0.856	*

Note : *For SVC (expanded with gridsearch), the time to get the result is too much due to the combination of SVC and gridsearch on large data set.

IV. APPLYING ENSEMBLE METHODS

Based on our baseline models above, now we can try to merge them as our final model by using ensemble methods.

A. Ensemble methods

There are mainly two families of ensemble methods, one is averaging methods, the other is boosting methods. Here we mainly consider about the former one, averaging methods. In averaging methods, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced [8]. In our problem, the structure can be summarized as Fig 5.

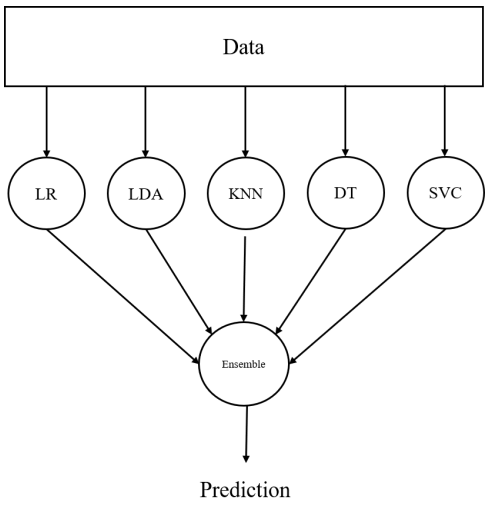

Fig. 5. Ensemble Structure

B. Implement

We use $VotingClassifier()$ in python to implement this. The idea behind the VotingClassifier is to combine conceptually different machine learning classifiers and use a majority vote or the average predicted probabilities (soft vote) to predict the class labels. Such a classifier can be useful for a set of equally well performing model in order to balance out their individual weaknesses [8].

Here we use the baseline models with gridsearch parameters in it as the parameter of $VotingClassifier()$. Then we train the model on the expanded data set and test it on the original testing data set.

C. Evaluate

In this part, we calculate the test accuracy of VotingClassifier is 0.867. More over, Fig 6 is the ROC curve of it and the AUC value is 0.919.

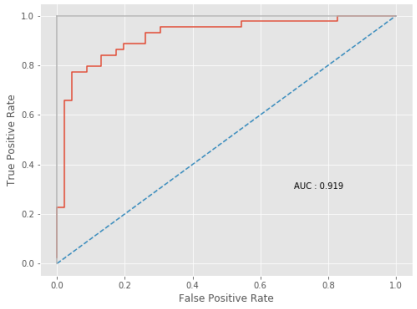

Fig. 6. Ensemble Structure

V. CONCLUSION

A. Summary

Based on Synthpop in R, we have expanded our data set. By using Pipeline and GridSearchCV in Python, we have

built some baseline classifiers and compared them on the original data set and expanded data set. After combing them in VotingClassifier, an ensemble method, we build our final model and give the test accuracy and ROC curve.

B. Some Advice

As we mentioned at begining, CHD is largely preventable, as many of its risk factors are modifiable. So, here are some advice :

- Eat a healthy, balanced diet.
- Be more physically active
- Keep to a healthy weight
- Give up smoking
- Reduce your alcohol consumption
- Keep your blood pressure under control
- Keep your diabetes under control

REFERENCES

[1] C. NCHS, "Underlying cause of death 1999-2013 on cdc wonder online database," *Underlying Cause Death*, vol. 2013, 1999.

[2] R. El-Bialy, M. A. Salamay, O. H. Karam, and M. E. Khalifa, "Feature analysis of coronary artery heart disease data sets," *Procedia Computer Science*, vol. 65, pp. 459–468, 2015.

[3] K. Srinivas, B. K. Rani, and A. Govrdhan, "Applications of data mining techniques in healthcare and prediction of heart attacks," *International Journal on Computer Science and Engineering (IJCSE)*, vol. 2, no. 02, pp. 250–255, 2010.

[4] M. Shouman, T. Turner, and R. Stocker, "Integrating decision tree and k-means clustering with different initial centroid selection methods in the diagnosis of heart disease patients," in *Proceedings of the International Conference on Data Mining (DMIN)*. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2012, p. 1.

[5] R. Detrano., "UCI cleveland heart disease data set," 1988. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Heart+Disease

[6] A. Sabay, L. Harris, V. Bejugama, and K. Jaceldo-Siegl, "Overcoming small data limitations in heart disease prediction by using surrogate data," *SMU Data Science Review*, vol. 1, no. 3, p. 12, 2018.

[7] B. Nowok, G. M. Raab, C. Dibben *et al.*, "synthpop: Bespoke creation of synthetic data in r," *J Stat Softw*, vol. 74, no. 11, pp. 1–26, 2016.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

APPENDIX A
R CODE

```
cle = read.csv("cleveland14.csv", head =
    FALSE)
colnames(cle) <-
    c("age","sex","cp","rbp","chol","fbs",
"restecg","max_hra","eia","oldpeak","slope",
    "ca", "thal", "diag")
library(plyr)
cle$ca <- revalue(cle$ca, c("?"= NA))
cle$thal <- revalue(cle$thal, c("?"= NA))
cle = na.omit(cle)
cle$diag <- revalue(as.factor(cle$diag),
    c('2' = '1', '3' = '1', '4' = '1'))
write.csv(cle, "cle_ori_pro.csv")
library('lattice')
library('MASS')
library('nnet')
library('ggplot2')
library('synthpop')
my.seed = 19970828
synth.obj = syn(cle, m = 100, seed = my.seed)
mycols = c("darkmagenta", "turquoise")
compare(synth.obj, cle,nrow = 3, ncol = 5,
    cols = mycols)$plot
dd <- do.call(rbind, synth.obj$syn)
write.csv(dd, file = 'synth_cleveland.csv')
```

APPENDIX B
PYTHON CODE

A. Package Used

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use("ggplot")
import seaborn as sns
from sklearn.model_selection import
    train_test_split
from sklearn.preprocessing import
    StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import
    GridSearchCV
from sklearn.linear_model import
    LogisticRegression
from sklearn.discriminant_analysis import
    LinearDiscriminantAnalysis
from sklearn.neighbors import
    KNeighborsClassifier
from sklearn.tree import
    DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import
    classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import roc_curve,
    roc_auc_score, auc
```

B. Part of Data Processing

```
Cle =
    pd.read_csv("synth_cleveland.csv").iloc[:,
    1:]
cle_original =
    pd.read_csv("cle_ori_pro.csv").iloc[:, 1:]
cle_original.describe()
sns.catplot(kind = 'count', data =
    cle_original,
x = 'age', hue = 'diag', order =
    cle_original['age'].sort_values().unique())
.fig.set_size_inches(18,8)
sns.pairplot(cle_original[['age', 'rbp',
    'chol', 'max_hra', 'oldpeak', 'diag']],
    hue='diag', vars=['age', 'rbp', 'chol',
    'max_hra', 'oldpeak'])
plt.savefig("Picture/continuous.png")
sns.factorplot("sex", "age", "diag", data =
    cle_original, kind="violin")
plt.savefig("Picture/violinplot.png")
#set kind = 'box' -> boxplot
#A violin plot is a method of plotting
    numeric data.
#It is similar to a box plot,
#with the addition of a rotated kernel
    density plot on each side.
data = cle_original[['sex', 'cp', 'fbs',
    'restecg', 'eia', 'slope', 'ca', 'thal',
    'diag']]
plt.rcParams['figure.figsize'] = [8, 6]
sns.heatmap(data.corr(), cmap="Blues")
plt.yticks(np.arange(data.shape[1]),
    data.columns)
plt.xticks(np.arange(data.shape[1]),
    data.columns)
plt.savefig("Picture/discrete.png")

Xtrain_ori, Xtest_ori, ytrain_ori, ytest_ori
    = train_test_split(cle_original.iloc[:,
    :-1], cle_original.iloc[:, -1], test_size
    = 0.3, random_state = 7)
Xtrain_pro, ytrain_pro = Cle.iloc[:, :-1],
    Cle.iloc[:, -1]
```

C. Baseline Models

```
model1_lr = LogisticRegression(random_state =
    7)

pipe_lr = Pipeline([('sc', StandardScaler()),
                ('clf', model1_lr)
                ])

# Create a list of options for the
    regularization penalty
penalty = ['l1', 'l2']
# Create a list of values of the
    regularization parameter
C = np.logspace(-4, 4, 50)
# Create a dictionary of all the parameter
    options
```

```
parameters = dict(clf__C=C,
             clf__penalty=penalty)

#The original one, train and test
pipe_lr.fit(Xtrain_ori, ytrain_ori)
print('Test accuracy (original without
    gridsearchcv): %.3f' %
    pipe_lr.score(Xtest_ori, ytest_ori ))

# Create a grid search object
grid = GridSearchCV(pipe_lr, parameters)

# Fit the grid search
grid.fit(Xtrain_ori, ytrain_ori)
# View The Best Parameters
print('Best Penalty:',
    grid.best_estimator_.get_params()['clf__penalty'])
print('Best C:',
grid.best_estimator_.get_params()['clf__C'])

print('Test accuracy
(the original with grid searchcv): %.3f' %
    grid.score(Xtest_ori, ytest_ori))

#all expanded train, the original one's test
pipe_lr.fit(Xtrain_pro, ytrain_pro)
print('Test accuracy (expanded
without gridsearchcv): %.3f' %
    pipe_lr.score(Xtest_ori, ytest_ori ))

# Create a grid search object
grid = GridSearchCV(pipe_lr, parameters)

# Fit the grid search
grid.fit(Xtrain_pro, ytrain_pro)
# View The Best Parameters
print('Best Penalty:',grid.best_estimator_
.get_params()['clf__penalty'])
print('Best C:', grid.best_estimator_.
get_params()['clf__C'])

print('Test accuracy
(the expanded with grid searchcv): %.3f' %
    grid.score(Xtest_ori, ytest_ori))

model2_lda = LinearDiscriminantAnalysis()
pipe_lda = Pipeline([('sc', StandardScaler()),
                ('clf', model2_lda)
                ])

#The original one, train and test
pipe_lda.fit(Xtrain_ori, ytrain_ori)
print('Test accuracy (original): %.3f' %
    pipe_lda.score(Xtest_ori, ytest_ori ))
#all expanded train, the original one's test
pipe_lda.fit(Xtrain_pro, ytrain_pro)
print('Test accuracy (expanded): %.3f' %
    pipe_lda.score(Xtest_ori, ytest_ori ))

#Others the same
```

D. Ensemble Part

```
estimators =
    [('lr',LogisticRegression(random_state
= 7, penalty = 'l2', C = 0.5689866029018293)),
           ('lda', model2_lda), ('knn',
           KNeighborsClassifier(n_neighbors =
               9)),
           ('dt', DecisionTreeClassifier(
           random_state = 7,
           criterion = 'gini', max_depth = 5)),
           ('svc', SVC(C = 0.004292, kernel =
               'linear',
           random_state = 7, probability =
               True))]#
ensemble = VotingClassifier(estimators,
    voting = 'soft')
pipe_line = Pipeline([('sc',
                StandardScaler()),
                ('clf', ensemble)
                ])
pipe_line.fit(Xtrain_pro, ytrain_pro)
print('Test accuracy: %.3f'
% pipe_line.score(Xtest_ori, ytest_ori ))
# Accuracy score is the simplest way to
    evaluate
print(accuracy_score(pipe_line
.predict(Xtest_ori), ytest_ori))
# But Confusion Matrix and Classification
    Report give more details about performance
print(confusion_matrix
(pipe_line.predict(Xtest_ori), ytest_ori))
print(classification_report
(pipe_line.predict(Xtest_ori), ytest_ori))
y_score =
    pipe_line.predict_proba(Xtest_ori)[:, 1]
# Create true and false positive rates
false_positive_rate, true_positive_rate,
threshold = roc_curve(ytest_ori, y_score)
roc_auc = auc(false_positive_rate,
    true_positive_rate)
# Plot ROC curve
plt.plot(false_positive_rate,
    true_positive_rate)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"),
    plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.text(0.7, 0.3, "AUC :
    {:.3f}".format(roc_auc))
plt.savefig("Picture/roc_auc.png")
```