1．实验目的

（1）掌握sklearn中决策树方法的调用。

（2）掌握决策树方法基本调参技术。

（3）掌握基于决策树的组合方法，包括Adaboost和 Random Forest

（4）能够根据准确率比较、评价不同的分类模型。

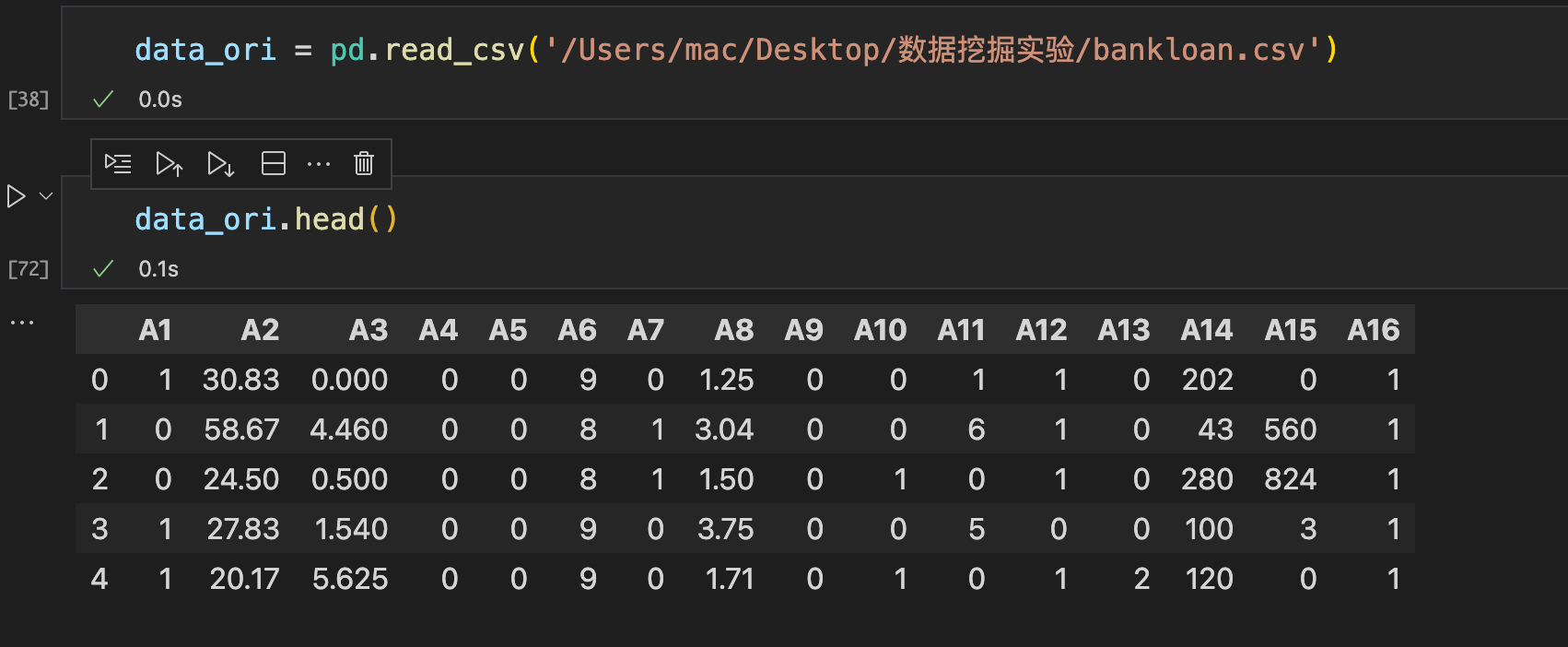
2．实验数据

数据文件bankloan.csv。银行贷款数据，一共包含690个样本，16个属性。基于隐私保护的考虑数据已经脱敏，属性名称用A1，A2，…，A16来表示，其中A16是类标号属性，有两个取值：“+”和“﹣”，“+”表示信贷申请被通过，“﹣”表示信贷申请被否决。需要注意的是，一些属性的值为“NA”，表示值缺失。

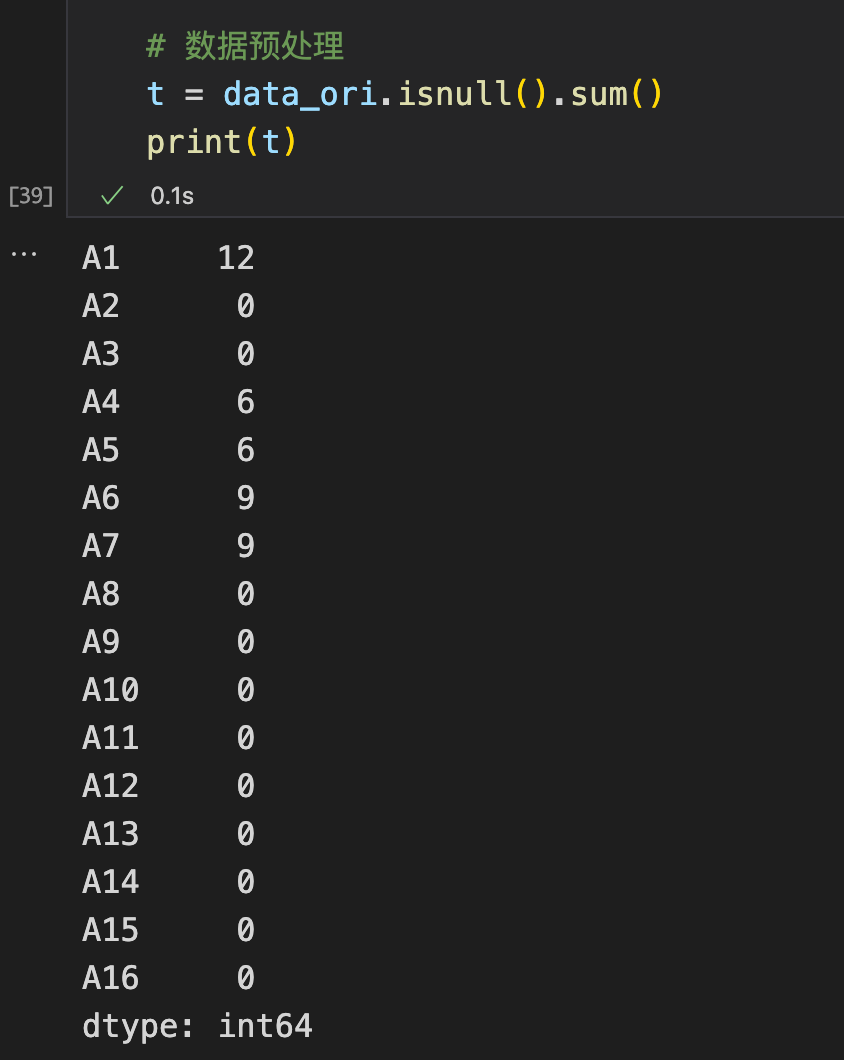
3．实验要求和步骤

（1）读取数据，完成数据预处理

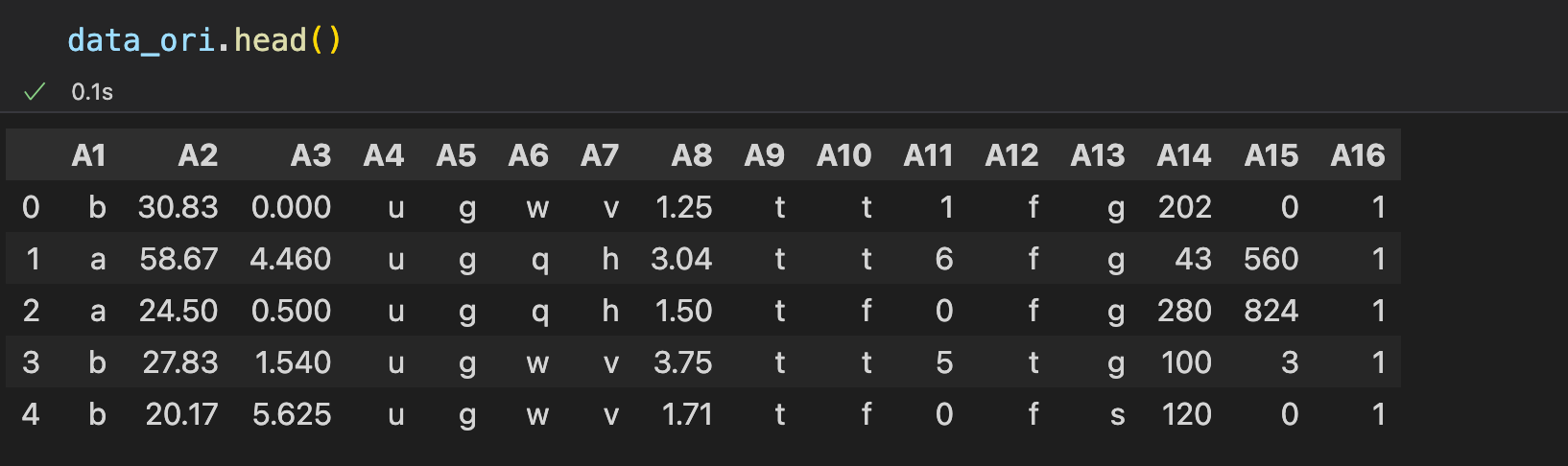
利用Panada库完成对于数据的读取：



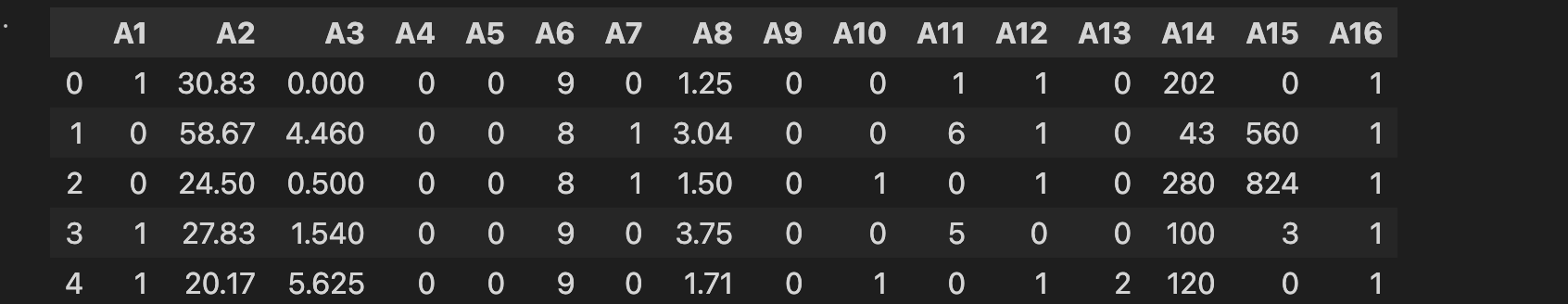
统计各个属性当中缺失值的个数并用众数对于离散数据中的缺失值进行补全:



完成缺失值处理之后得到的数据

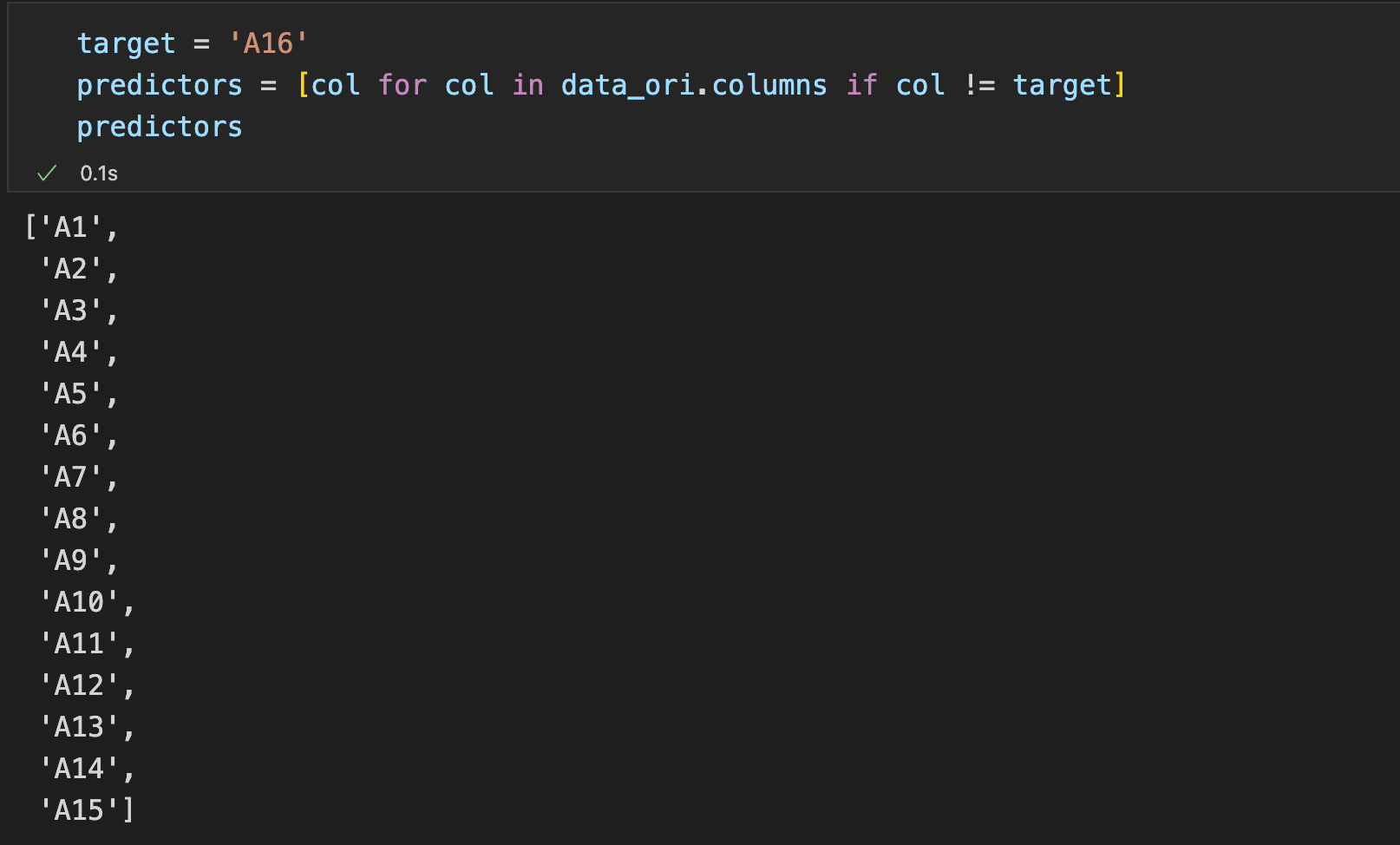


对于非数据化的数值进行数据化之后得到的数据

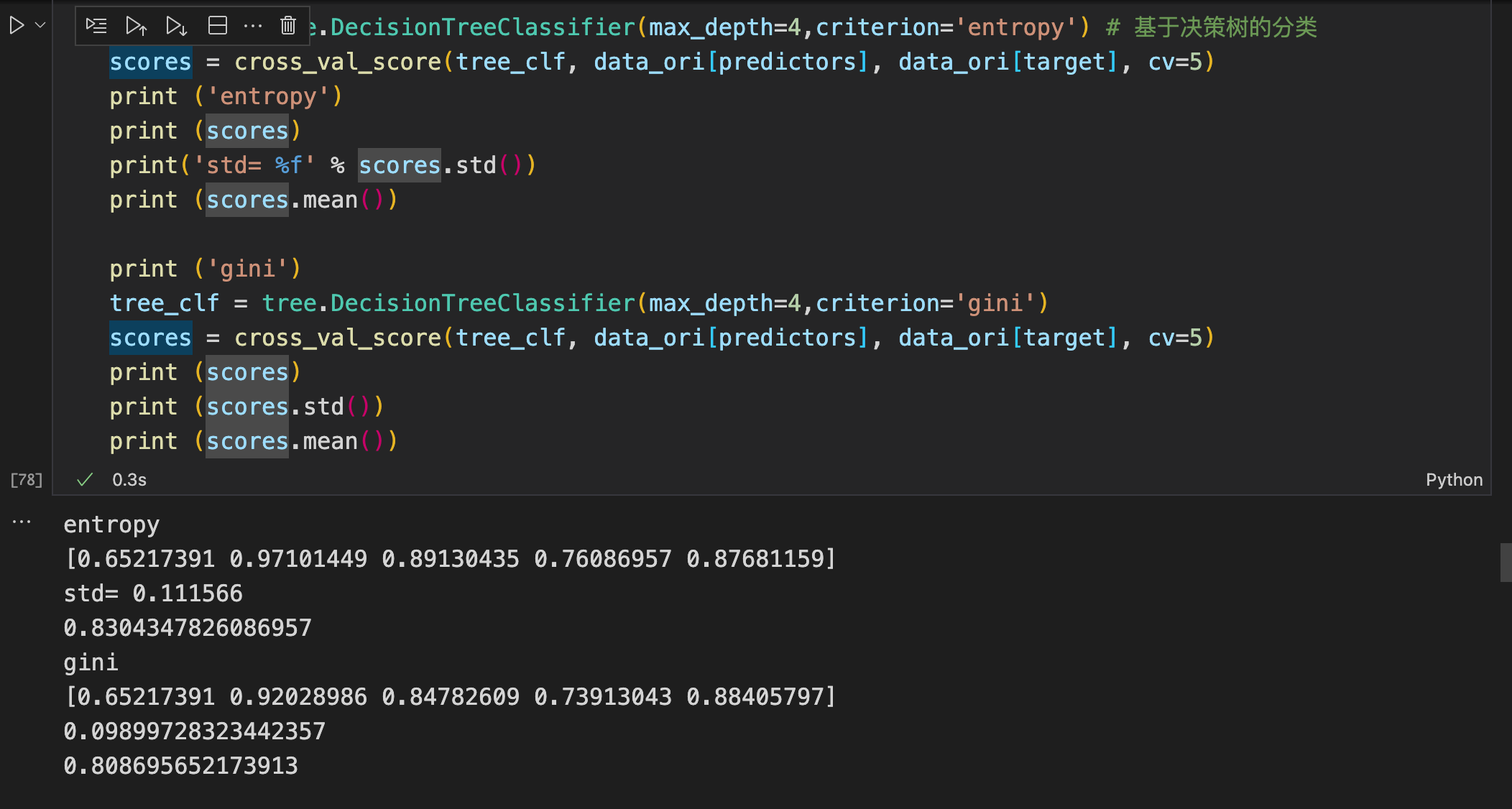


1. 选择合适的特征集合

暂时将其中除了target之外的所有数据都设置为predictors



1. 基于交叉验证方法，训练一个决策树模型，参数criterion分别选择gini和entropy，优化参数，比较分类准确率,可以发现采用entropy的准确率略高于gini

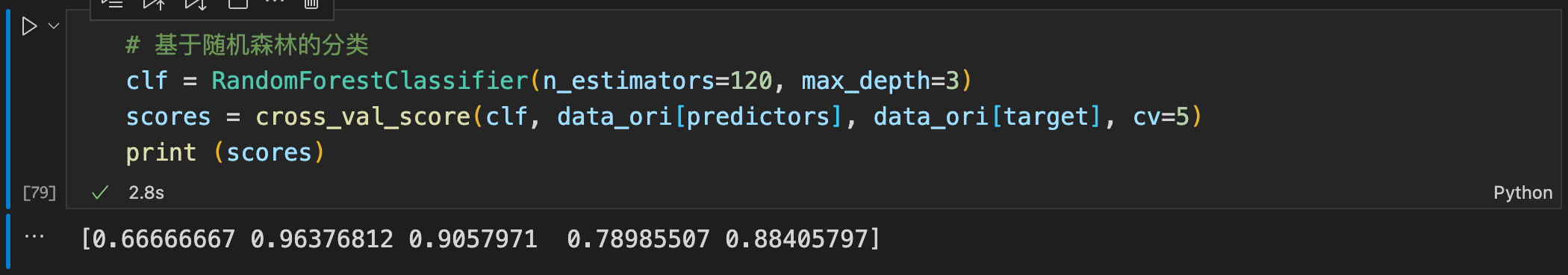


1. 基于交叉验证方法，分别训练Adaboost和 Random Forest模型，优化参数。建议，某个重要参数，例如n\_estimator，调参时，同时对比两个算法的性能，并绘制为图表，例如accuracy与基分类器数量的关系曲线图。

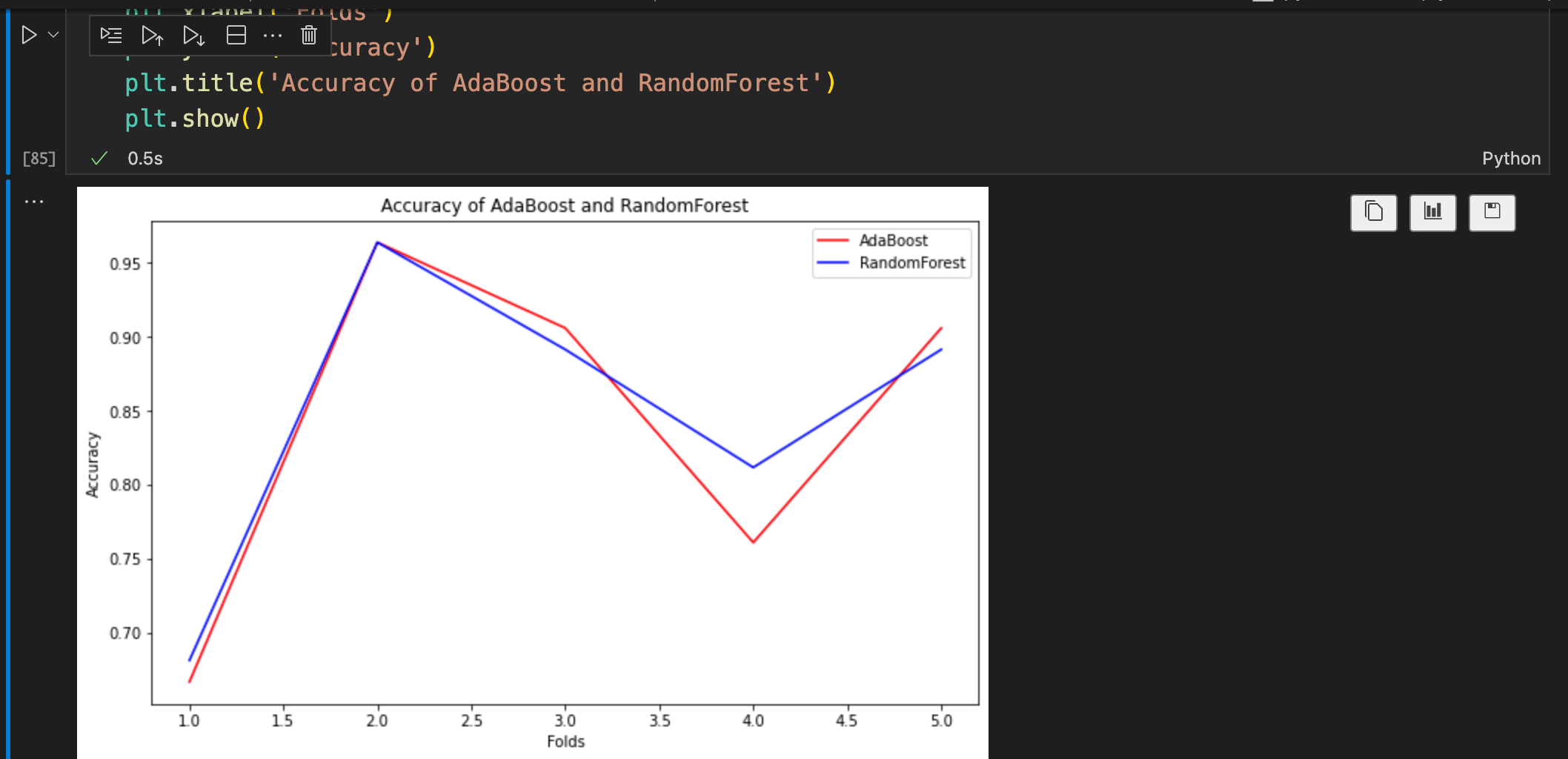
基于adaboost 进行分类得到的结果



基于随机森林分类得到的结果

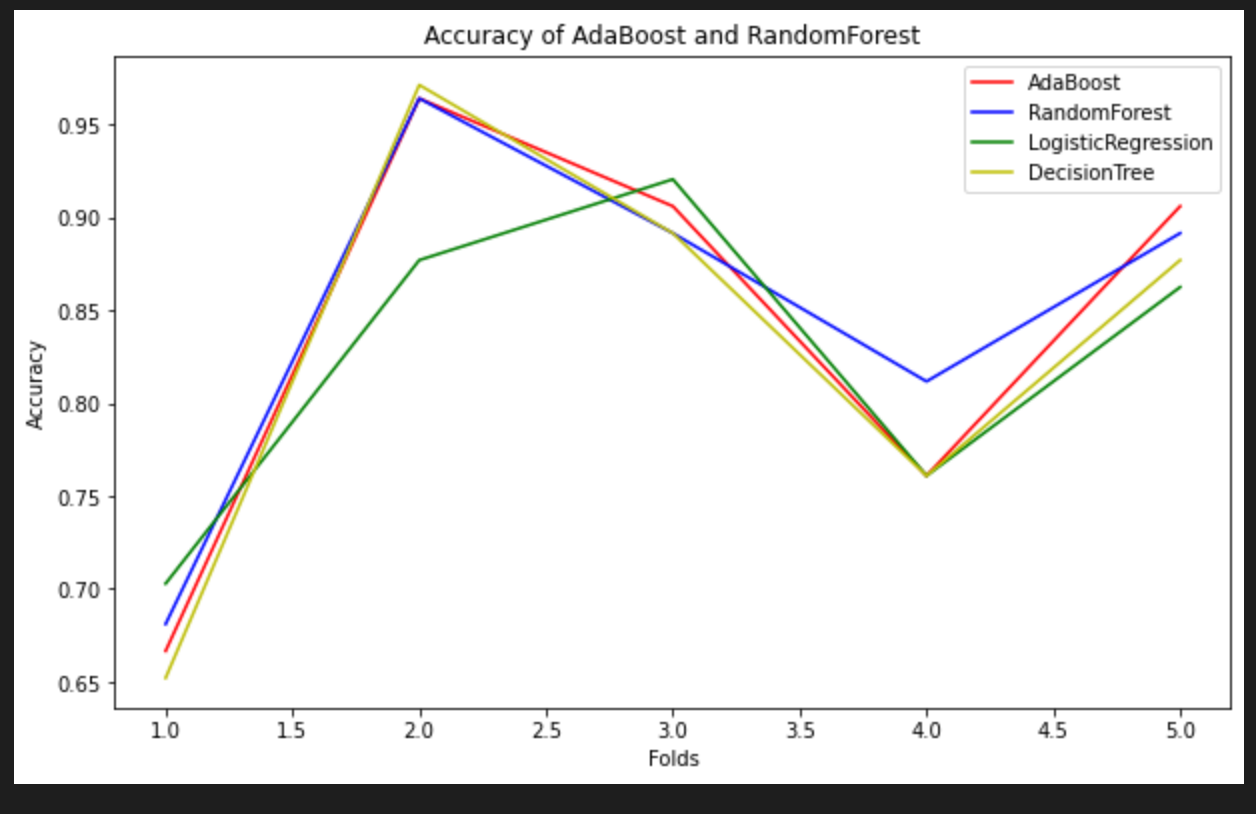


绘制两种模型的对比图之后可以得到



1. 综合比较多个模型的性能，尝试优化特征工程，改进各模型的效果

将上面用到的四种模型绘制于同一个对比图当中



（6）书写图文并茂的数据分析报告。性能比较，应该列一个表格。

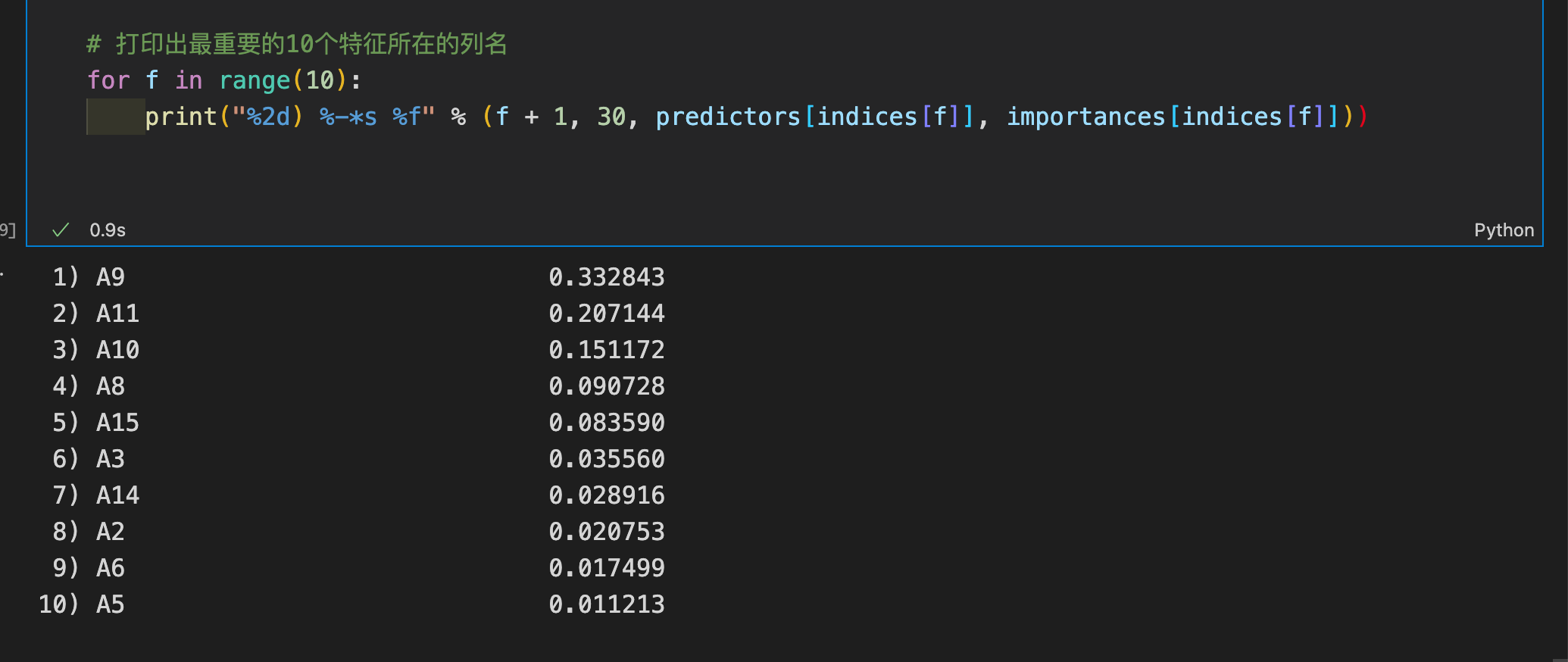
列表之后综合比较随机森林法拥有最高的准确率并且方差比较小，逻辑回归法准确率最低



4．分析与讨论

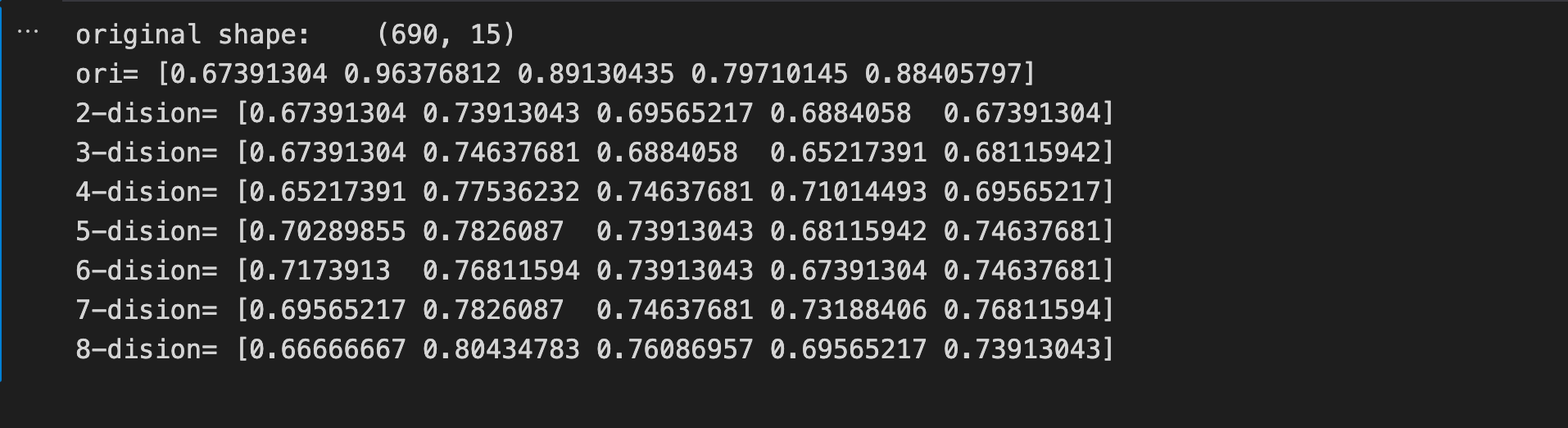
（1）实验数据中，如何评价各特征维度对分类结果的重要性？列出Top10重要 特征，给出必要 的说明。

利用sklearn库当中的feature\_importances\_函数给出对于各个特征的重要性进行评估之后可以得出最重要的10个特征如下



（2）本实验任务，降维方法是否可改善分类性能？

由于在之前的实验过程当中随机森林算法表现出了最好的性能，因此在这里采用随机森林算法对于两者进行对比，发现降低成2维到8维之后采用随机森林算法得到的结果明显比之前的要差，正确率远低于没有降维之间处理的数据，并且随之维度的升高其正确率是逐步上升到，因此我认为不能采用降低维度的方法来提高正确率



5．附录

# %%

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

from sklearn import tree

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.naive\_bayes import GaussianNB

%matplotlib inline

# %%

data\_ori = pd.read\_csv('/Users/mac/Desktop/数据挖掘实验/bankloan.csv')

# %%

data\_ori.head()

# %%

# 数据预处理

t = data\_ori.isnull().sum()

print(t)

# %%

print(t[t>0])

# %%

# 无缺失值

data\_ori['A1'].value\_counts()

# %%

data\_ori['A1'].fillna('b',inplace=True) # 用众数填充

# %%

data\_ori['A4'].value\_counts()

# %%

data\_ori['A4'].fillna('u',inplace=True) # 用众数填充

# %%

data\_ori['A5'].value\_counts()

# %%

data\_ori['A5'].fillna('g',inplace=True) # 用众数填充

# %%

data\_ori['A6'].value\_counts()

# %%

data\_ori['A6'].fillna('c',inplace=True) # 用众数填充

# %%

data\_ori['A7'].value\_counts()

# %%

data\_ori['A7'].fillna('v',inplace=True) # 用众数填充

# %%

data\_ori['A9'].value\_counts()

# %%

data\_ori['A9'].fillna('t',inplace=True) # 用众数填充

# %%

data\_ori['A10'].value\_counts()

# %%

data\_ori['A10'].fillna('f',inplace=True) # 用众数填充

# %%

data\_ori['A12'].value\_counts()

# %%

data\_ori['A12'].fillna('f',inplace=True) # 用众数填充

# %%

data\_ori['A13'].value\_counts()

# %%

data\_ori['A13'].fillna('g',inplace=True) # 用众数填充

# %%

cols = ['A1','A4','A5','A6','A7','A9','A10','A12','A13']

# %%

for col in cols:

print( col,'->' ,data\_ori[col].unique())

# %%

data\_ori.shape

# %%

data\_ori.head()

# %%

# 将数值进行数据化

data\_ori['A1'] = data\_ori['A1'].map({'a':0,'b':1})

data\_ori['A4'] = data\_ori['A4'].map({'u':0,'y':1,'l':2})

data\_ori['A5'] = data\_ori['A5'].map({'g':0,'p':1,'gg':2})

data\_ori['A6'] = data\_ori['A6'].map({'c':0,'d':1,'cc':2,'i':3,'j':4,'k':5,'m':6,'r':7,'q':8,'w':9,'x':10,'e':11,'aa':12,'ff':13})

data\_ori['A7'] = data\_ori['A7'].map({'v':0,'h':1,'bb':2,'j':3,'n':4,'z':5,'dd':6,'ff':7,'o':8})

data\_ori['A9'] = data\_ori['A9'].map({'t':0,'f':1})

data\_ori['A10'] = data\_ori['A10'].map({'t':0,'f':1})

data\_ori['A12'] = data\_ori['A12'].map({'t':0,'f':1})

data\_ori['A13'] = data\_ori['A13'].map({'g':0,'p':1,'s':2})

# %%

data\_ori.head()

# %%

# 查找其中是否还存在缺失值

data\_ori.isnull().sum()

# %%

target = 'A16'

predictors = [col for col in data\_ori.columns if col != target]

predictors

# %%

# 交叉验证

from sklearn.model\_selection import cross\_val\_score

from sklearn.linear\_model import LogisticRegression

# Initialize our algorithm

alg = LogisticRegression(random\_state=1) # 逻辑回归

# Compute the accuracy score for all the cross validation folds. (much simpler than what we did before!)

scores\_lr = cross\_val\_score(alg, data\_ori[predictors], data\_ori[target], cv=5)

print (scores\_lr)

print (scores\_lr.std())

# Take the mean of the scores (because we have one for each fold)

print(scores\_lr.mean())

# %%

# 基于随机森林的分类

clf = RandomForestClassifier(n\_estimators=120, max\_depth=3)

scores\_rf = cross\_val\_score(clf, data\_ori[predictors], data\_ori[target], cv=5)

print (scores\_rf)

# %%

tree\_clf = tree.DecisionTreeClassifier(max\_depth=4,criterion='entropy') # 基于决策树的分类

scores\_dt = cross\_val\_score(tree\_clf, data\_ori[predictors], data\_ori[target], cv=5)

print ('entropy')

print (scores\_dt)

print('std= %f' % scores\_dt.std())

print (scores\_dt.mean())

print ('gini')

tree\_clf = tree.DecisionTreeClassifier(max\_depth=4,criterion='gini')

scores = cross\_val\_score(tree\_clf, data\_ori[predictors], data\_ori[target], cv=5)

print (scores)

print (scores.std())

print (scores.mean())

# %%

# 基于AdaBoost的分类

clf = AdaBoostClassifier(n\_estimators=120, algorithm='SAMME')

scores\_ada = cross\_val\_score(clf, data\_ori[predictors], data\_ori[target], cv=5)

print (scores\_ada)

print (scores\_ada.std())

print (scores\_ada.mean())

# %%

# 绘制对比图

import matplotlib.pyplot as plt

%matplotlib inline

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

plt.plot([1,2,3,4,5],scores\_ada,'r',label='AdaBoost')

plt.plot([1,2,3,4,5],scores\_rf,'b',label='RandomForest')

plt.plot([1,2,3,4,5],scores\_lr,'g',label='LogisticRegression')

plt.plot([1,2,3,4,5],scores\_dt,'y',label='DecisionTree')

plt.legend(loc='upper right')

plt.xlabel('Folds')

plt.ylabel('Accuracy')

plt.title('Accuracy of AdaBoost and RandomForest')

plt.show()

# %%

# 对于四种分析方法的准确率、方差列成表格

data = {'AdaBoost':[scores\_ada.mean(),scores\_ada.std()],'RandomForest':[scores\_rf.mean(),scores\_rf.std()],'LogisticRegression':[scores\_lr.mean(),scores\_lr.std()],'DecisionTree':[scores\_dt.mean(),scores\_dt.std()]}

df = pd.DataFrame(data,index=['Accuracy','Variance'])

df

# %%

# 评价各个维度数据的重要性

clf = RandomForestClassifier(n\_estimators=120, max\_depth=3)

clf.fit(data\_ori[predictors], data\_ori[target])

importances = clf.feature\_importances\_ # 数值越大，说明该特征越重要

# print(importances)

# 重要性和特征名称对应

indices = np.argsort(importances)[::-1]

indices

# 打印出最重要的10个特征所在的列名

for f in range(10):

print("%2d) %-\*s %f" % (f + 1, 30, predictors[indices[f]], importances[indices[f]]))

# %%

# 对于数据进行降维度处理

from sklearn.decomposition import PCA

# 降维度到2维

pca = PCA(n\_components=2)

pca.fit(data\_ori[predictors])

X\_pca = pca.transform(data\_ori[predictors])

print("original shape: ", data\_ori[predictors].shape)

# 比较降维前后的数据得到的准确率和之前的对比

clf = RandomForestClassifier(n\_estimators=120, max\_depth=3)

scores\_rf = cross\_val\_score(clf, data\_ori[predictors], data\_ori[target], cv=5)

print ('ori=',scores\_rf)

clf = RandomForestClassifier(n\_estimators=120, max\_depth=3)

scores\_rf\_new = cross\_val\_score(clf, X\_pca, data\_ori[target], cv=5)

print ('2-dision=',scores\_rf\_new)

# 降维度到3维

pca = PCA(n\_components=3)

pca.fit(data\_ori[predictors])

X\_pca = pca.transform(data\_ori[predictors])

score\_rf\_new\_3 = cross\_val\_score(clf, X\_pca, data\_ori[target], cv=5)

print ('3-dision=',score\_rf\_new\_3)

# 降维度到4维

pca = PCA(n\_components=4)

pca.fit(data\_ori[predictors])

X\_pca = pca.transform(data\_ori[predictors])

score\_rf\_new\_4 = cross\_val\_score(clf, X\_pca, data\_ori[target], cv=5)

print ('4-dision=',score\_rf\_new\_4)

# 降维度到5维

pca = PCA(n\_components=5)

pca.fit(data\_ori[predictors])

X\_pca = pca.transform(data\_ori[predictors])

score\_rf\_new\_5 = cross\_val\_score(clf, X\_pca, data\_ori[target], cv=5)

print ('5-dision=',score\_rf\_new\_5)

# 降维度到6维

pca = PCA(n\_components=6)

pca.fit(data\_ori[predictors])

X\_pca = pca.transform(data\_ori[predictors])

score\_rf\_new\_6 = cross\_val\_score(clf, X\_pca, data\_ori[target], cv=5)

print ('6-dision=',score\_rf\_new\_6)

# 降维度到7维

pca = PCA(n\_components=7)

pca.fit(data\_ori[predictors])

X\_pca = pca.transform(data\_ori[predictors])

score\_rf\_new\_7 = cross\_val\_score(clf, X\_pca, data\_ori[target], cv=5)

print ('7-dision=',score\_rf\_new\_7)

# 降维度到8维

pca = PCA(n\_components=8)

pca.fit(data\_ori[predictors])

X\_pca = pca.transform(data\_ori[predictors])

score\_rf\_new\_8 = cross\_val\_score(clf, X\_pca, data\_ori[target], cv=5)

print ('8-dision=',score\_rf\_new\_8)