1．实验目的

（1）掌握线性SVM和基于核的SVM的调用方法

（2）掌握不平衡数据集分类的代价敏感方法

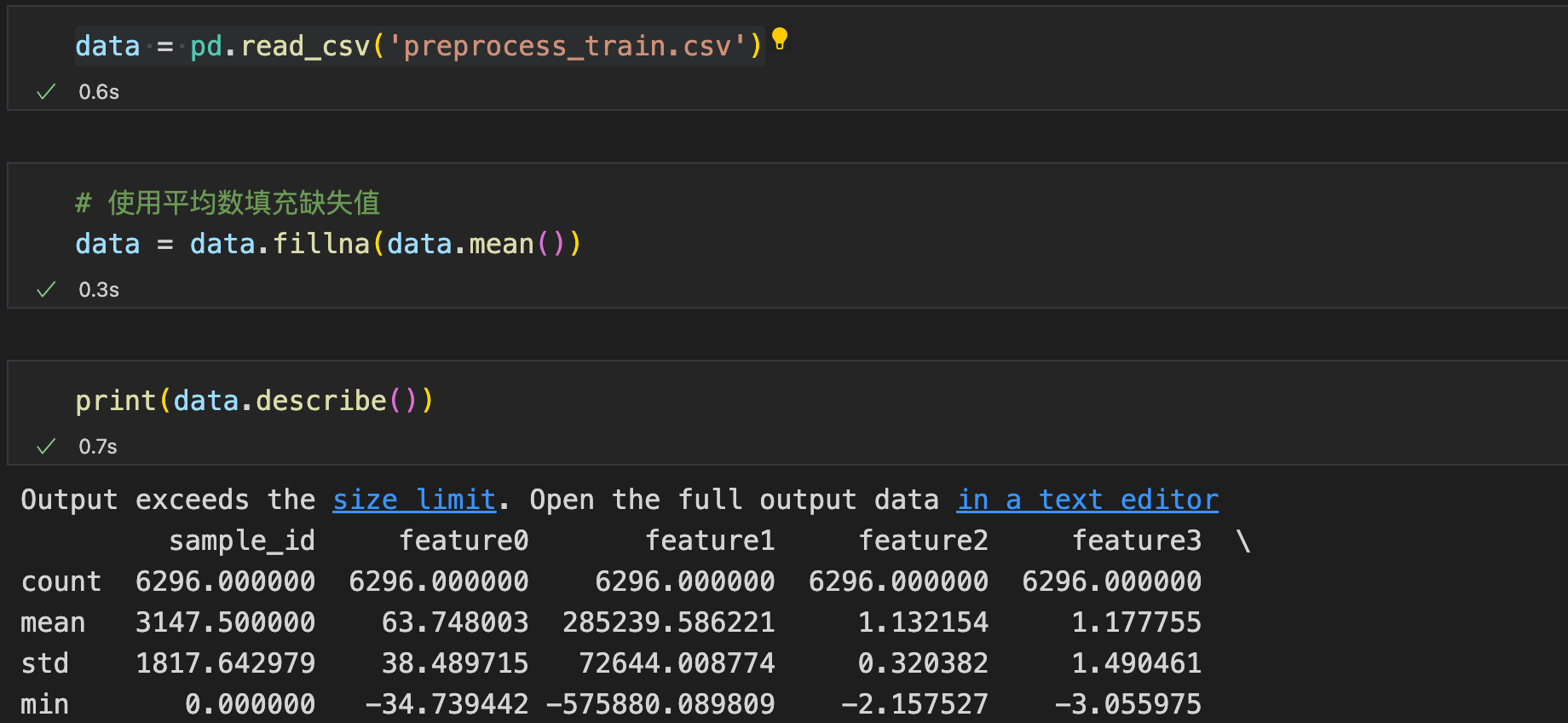
（3）能够恰当调节支持向量机的关键参数

（4）掌握gridsearch调参方法

2．实验要求和步骤

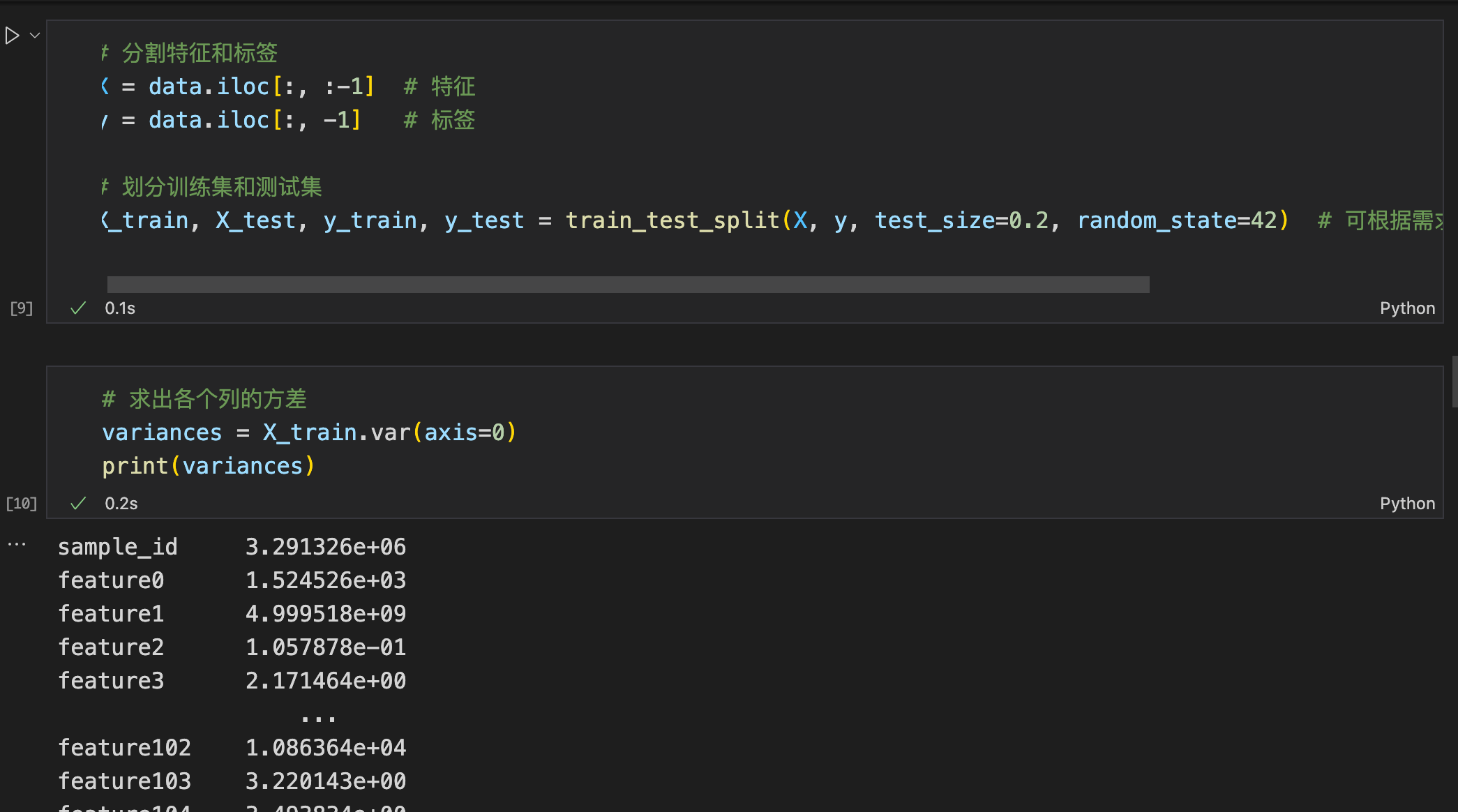
（1）读取数据，完成数据预处理。观察数据分布情况，适当可视化分析数据，并输出结果，插入实验报告。

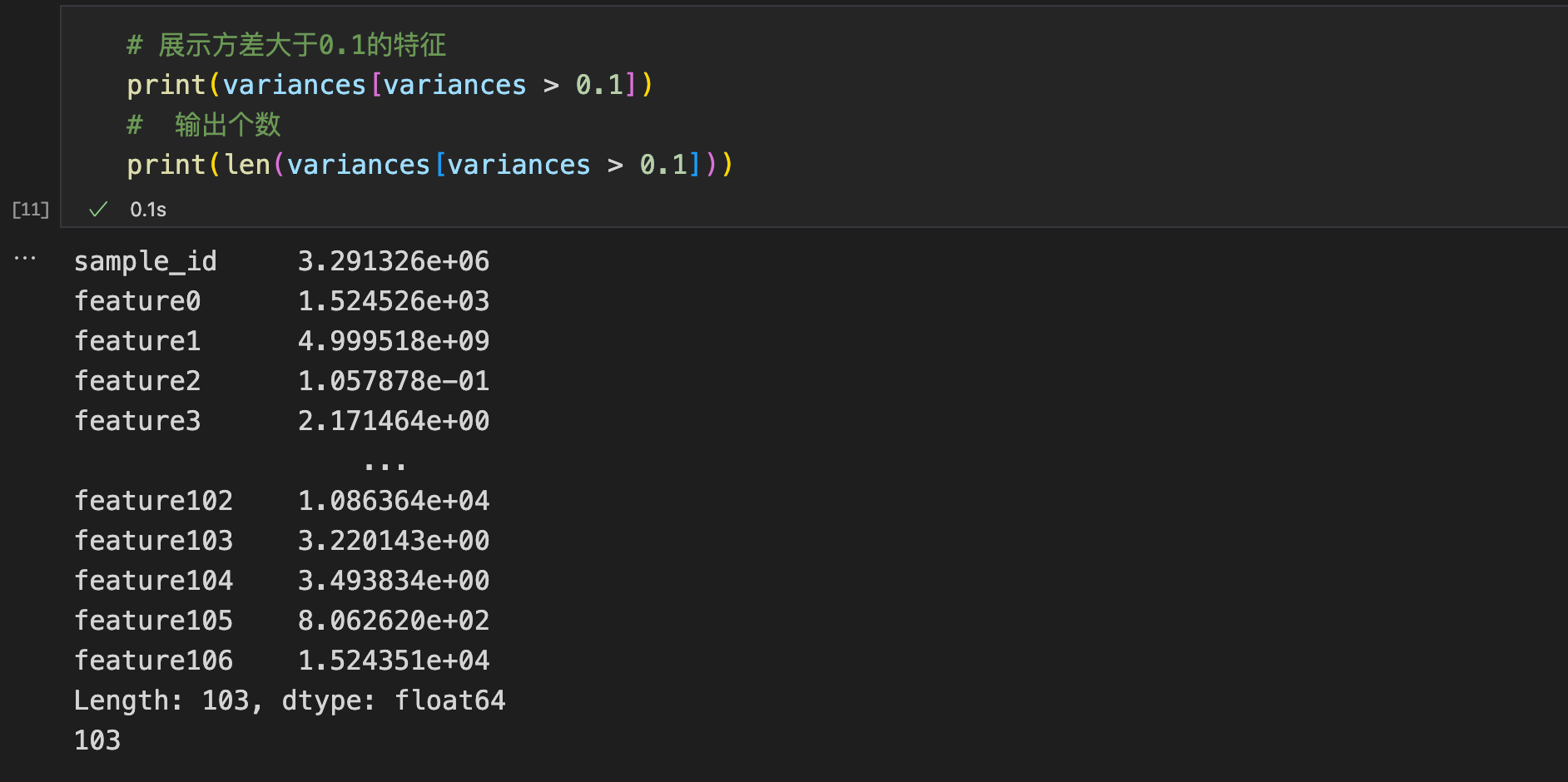
对于数据中缺失的部分采用平均值进行填补



1. 选择合适的特征集合。

为了选取区分度比较大的特征，我选择将方差小于0.1的特征全部抛弃，只是保留了方差大于0.1的特征，全部的108个特征当中抛弃了5个

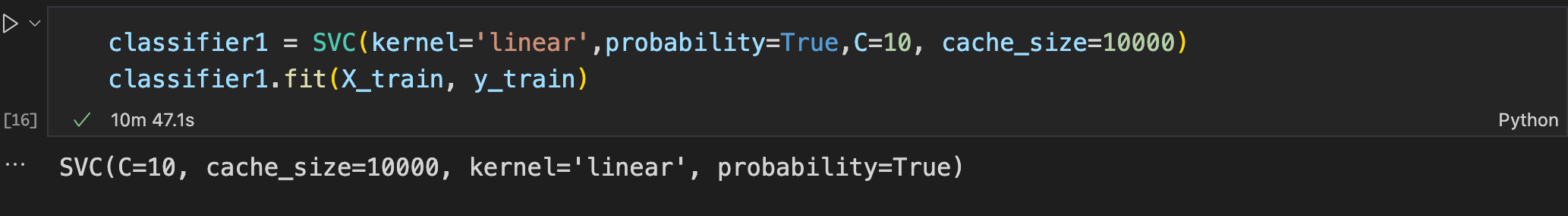


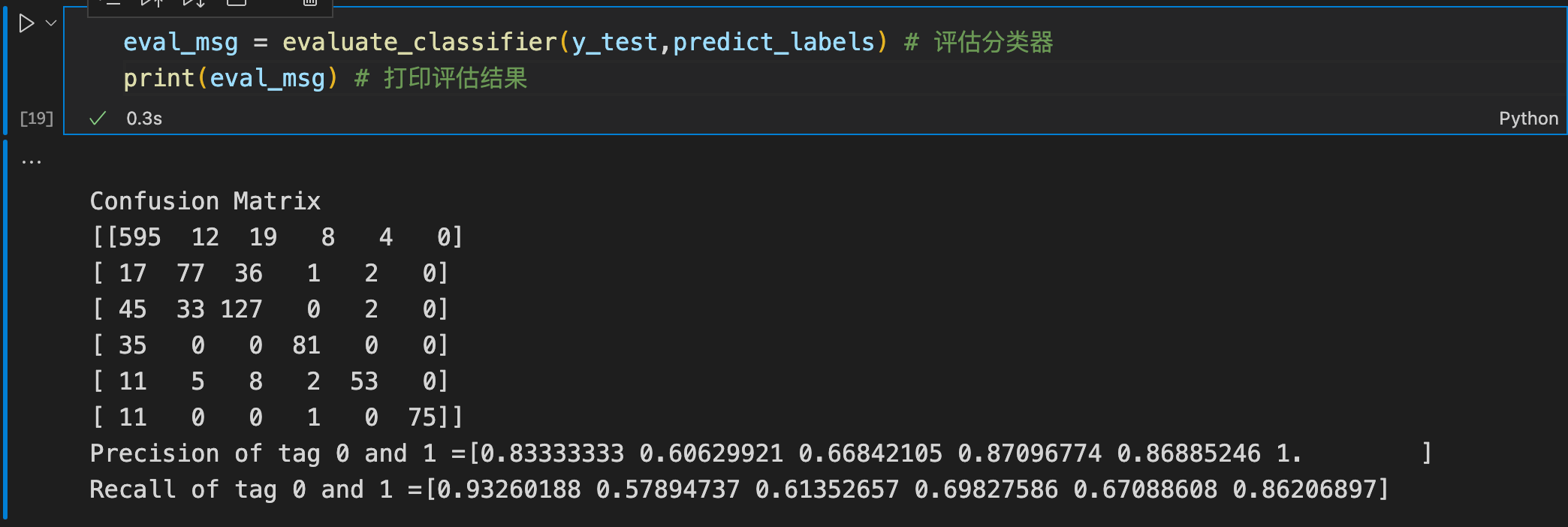


随后对于处理后的数据进行归一化处理



1. 生成SVM模型，设置线性核与非线性核，初步尝试参数优化，包括kernel，C，gamma，cache\_size，class\_weight等，尽量提高模型的f1-score。





1. 基于GridSearchCV，组合多种参数，尝试找出最佳参数组合，获得最高f1-score。以表格的形式，对比不同参数获得的不同性能。



1. 打印输出较优模型的分类评价报告。



（6）持久化保存获得的最优svm模型。

3．分析与讨论

（1）总结归纳本次调参经验。

1.为了减少计算量，对于一些对于模型最终的结果影响不大的可以适当进行抛弃

2.下次再搞这么大的数据集我选择直接上kaggle用服务器，用自己的电脑跑太坐牢了，当中的一个步骤直接算了一百多分钟。

（2）本实验任务，对数据做规范化和降维，是否能改善分类性能？

我的降维度并没有提高正确率，只是之前数据太多了电脑性能算不动了，迫不得已，如果再算力上有所保障可能能够得到更好一点的结果

4．附录

# %%

# system lib

from sklearn.metrics import accuracy\_score,confusion\_matrix,precision\_score,recall\_score

from sklearn.svm import SVC, LinearSVC, NuSVC

from sklearn import model\_selection

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier #随机森林

from sklearn import tree

#用于参数搜索

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import classification\_report

from sklearn.metrics import roc\_curve, auc #绘制ROC曲线

import pylab as pl

from time import time

import datetime

import numpy as np

# %%

import pickle

from sklearn.model\_selection import cross\_validate

import pandas as pd

# %%

def load\_data(filename):

"""根据数据格式，读取数据中的X和分类标签y

"""

return x\_data, ylabel

def evaluate\_classifier( real\_label\_list,predict\_label\_list):

"""

return Precision, Recall and ConfusionMatrix

Input : predict\_label\_list,real\_label\_list

"""

msg=''

Confusion\_matrix = confusion\_matrix( real\_label\_list,predict\_label\_list)

msg += '\n Confusion Matrix\n ' + str(Confusion\_matrix)

precision = precision\_score(real\_label\_list,predict\_label\_list, average=None)

recall = recall\_score(real\_label\_list,predict\_label\_list, average=None)

msg += '\n Precision of tag 0 and 1 =%s' %str(precision)

msg += '\n Recall of tag 0 and 1 =%s' %str(recall)

return msg

def test\_svm(train\_file, test\_file):

"""用SVM分类 """

# use SVM directly

train\_xdata, train\_ylabel = load\_data(train\_file)

test\_xdata, test\_ylabel = load\_data(test\_file)

print('\nuse SVM directly')

#classifier1 = SVC(kernel='linear')

#classifier1 = SVC(kernel='linear',probability=True, C=200, cache\_size=500)

classifier1 = SVC(kernel='linear',probability=True,C=10, cache\_size=500)

classifier1.fit(train\_xdata, train\_ylabel)

predict\_labels = classifier1.predict(test\_xdata)

accuracy = accuracy\_score(test\_ylabel, predict\_labels)

print("\n The Classifier's Accuracy is : %f" %accuracy)

#

eval\_msg = evaluate\_classifier(test\_ylabel,predict\_labels)

print(eval\_msg)

#

#GridSearchCV搜索最优参数示例

print("GridSearchCV搜索最优参数......")

t0 = time()

param\_grid = {

"C": [1e3, 5e3, 1e4, 5e4, 1e5],

"gamma": [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1],

}

classifier1 = GridSearchCV(SVC(kernel="rbf", class\_weight="balanced",probability=True), param\_grid)

classifier1 = classifier1.fit(train\_xdata, train\_ylabel)

print("done in %0.3fs" % (time() - t0))

print("Best estimator found by grid search:")

print(classifier1.best\_estimator\_)

#对于SVM来说，概率是通过交叉验证得到的，与其预测的结果未必一致，对小数据集来说，此概率没什么意义

probas\_ = classifier1.predict\_proba(test\_xdata)

#对于二分类问题，可为分类器绘制ROC曲线，计算AUC

# Compute ROC curve and area the curve

fpr, tpr, thresholds = roc\_curve(test\_ylabel, probas\_[:, 1])

roc\_auc = auc(fpr, tpr)

print("Area under the ROC curve : %f" % roc\_auc)

# Plot ROC curve

pl.clf()

pl.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc\_auc)

pl.plot([0, 1], [0, 1], 'k--')

pl.xlim([0.0, 1.0])

pl.ylim([0.0, 1.0])

pl.xlabel('False Positive Rate')

pl.ylabel('True Positive Rate')

pl.title('%s SVM ROC' %train\_file)

pl.legend(loc="lower right")

pl.show()

# %%

data = pd.read\_csv('preprocess\_train.csv')

# %%

# 使用平均数填充缺失值

data = data.fillna(data.mean())

# %%

print(data.describe())

# %%

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

from sklearn.preprocessing import StandardScaler

# %%

# 分割特征和标签

X = data.iloc[:, :-1] # 特征

y = data.iloc[:, -1] # 标签

# 划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) # 可根据需求设置测试集比例和随机种子

# %%

# 求出各个列的方差

variances = X\_train.var(axis=0)

print(variances)

# %%

# 展示方差大于0.1的特征

print(variances[variances > 0.1])

# 输出个数

print(len(variances[variances > 0.1]))

# %%

# 选择方差大于0.1的特征

X\_train = X\_train.loc[:, variances > 0.1]

# %%

# 对于test集选择相同的特征

X\_test = X\_test.loc[:, variances > 0.1]

# %%

# 特征归一化

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train) # 注意这里是fit\_transform

X\_test = scaler.transform(X\_test) # 注意这里是transform

print('X\_train.shape:', X\_train.shape)

print('X\_test.shape:', X\_test.shape)

# %%

# 方差选择法数据预处理

from sklearn.feature\_selection import VarianceThreshold

# 创建VarianceThreshold对象

selector = VarianceThreshold(threshold=0.01)

# 在训练集上拟合并应用特征选择

X\_train = selector.fit\_transform(X\_train)

# 在测试集上应用相同的特征选择

X\_test = selector.transform(X\_test)

print('X\_train.shape:', X\_train.shape)

print('X\_test.shape:', X\_test.shape)

# %%

classifier1 = SVC(kernel='linear',probability=True,C=10, cache\_size=10000)

classifier1.fit(X\_train, y\_train)

# %%

from sklearn.metrics import f1\_score

predict\_labels = classifier1.predict(X\_test)

accuracy = accuracy\_score(y\_test, predict\_labels)

print("\n The Classifier's Accuracy is : %f" %accuracy)

# 计算f1score

f1score = f1\_score(y\_test, predict\_labels, average='macro')

print("\n The Classifier's f1score is : %f" %f1score)

# %%

classifier1 = SVC(kernel='linear',probability=True,C=10, cache\_size=5000)

classifier1.fit(X\_train, y\_train)

# %%

eval\_msg = evaluate\_classifier(y\_test,predict\_labels) # 评估分类器

print(eval\_msg) # 打印评估结果

# %%

print("GridSearchCV搜索最优参数......")

t0 = time()

param\_grid = {

"C": [1e3, 5e3, 1e4, 5e4, 1e5],

"gamma": [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1],

}

classifier1 = GridSearchCV(SVC(kernel="rbf",probability=True), param\_grid) #balance不需要

classifier1 = classifier1.fit(X\_train, y\_train)

# %%

print("done in %0.3fs" % (time() - t0))

print("Best estimator found by grid search:") # 打印最优参数

print(classifier1.best\_estimator\_) # 打印最优参数

# %%

probas\_ = classifier1.predict\_proba(X\_test) # 对测试集进行预测

print(probas\_)# 打印预测结果

# %%

# 持久化保存获得的最优svm模型。

import joblib

joblib.dump(classifier1, 'svm\_model.pkl')