1．实验目的

（1）掌握多种常见分类模型的使用方法

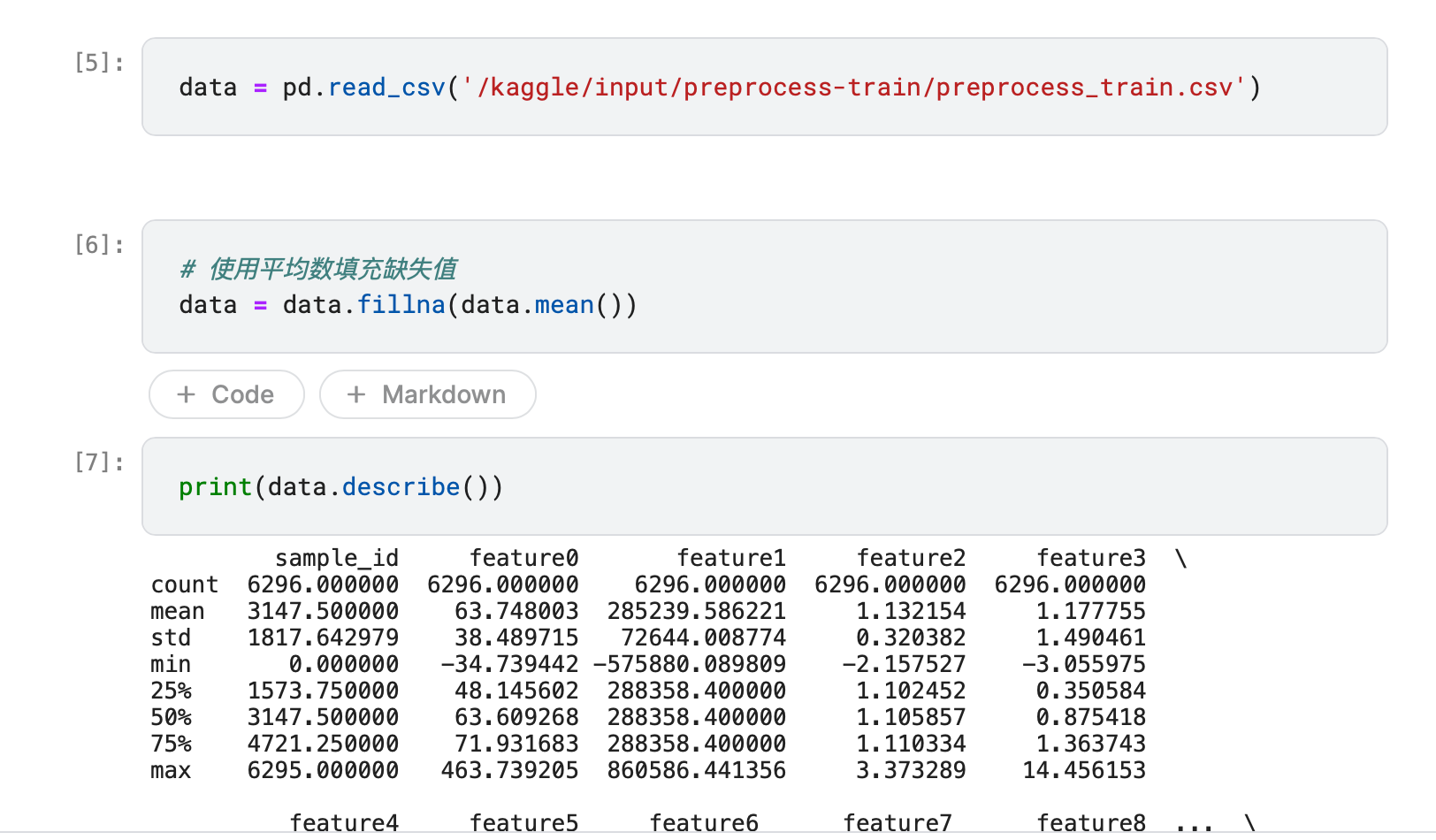
（2）能够用accuray，precision，recall，F1-score等多种评价指标比较分类模型

（3）掌握ROC曲线和AUC评价方法和可视化方法

（4）能够综合使用多种评价方法分析分类模型的优劣，筛选最佳模型

2．实验要求和步骤

（1）读取数据，完成数据预处理。



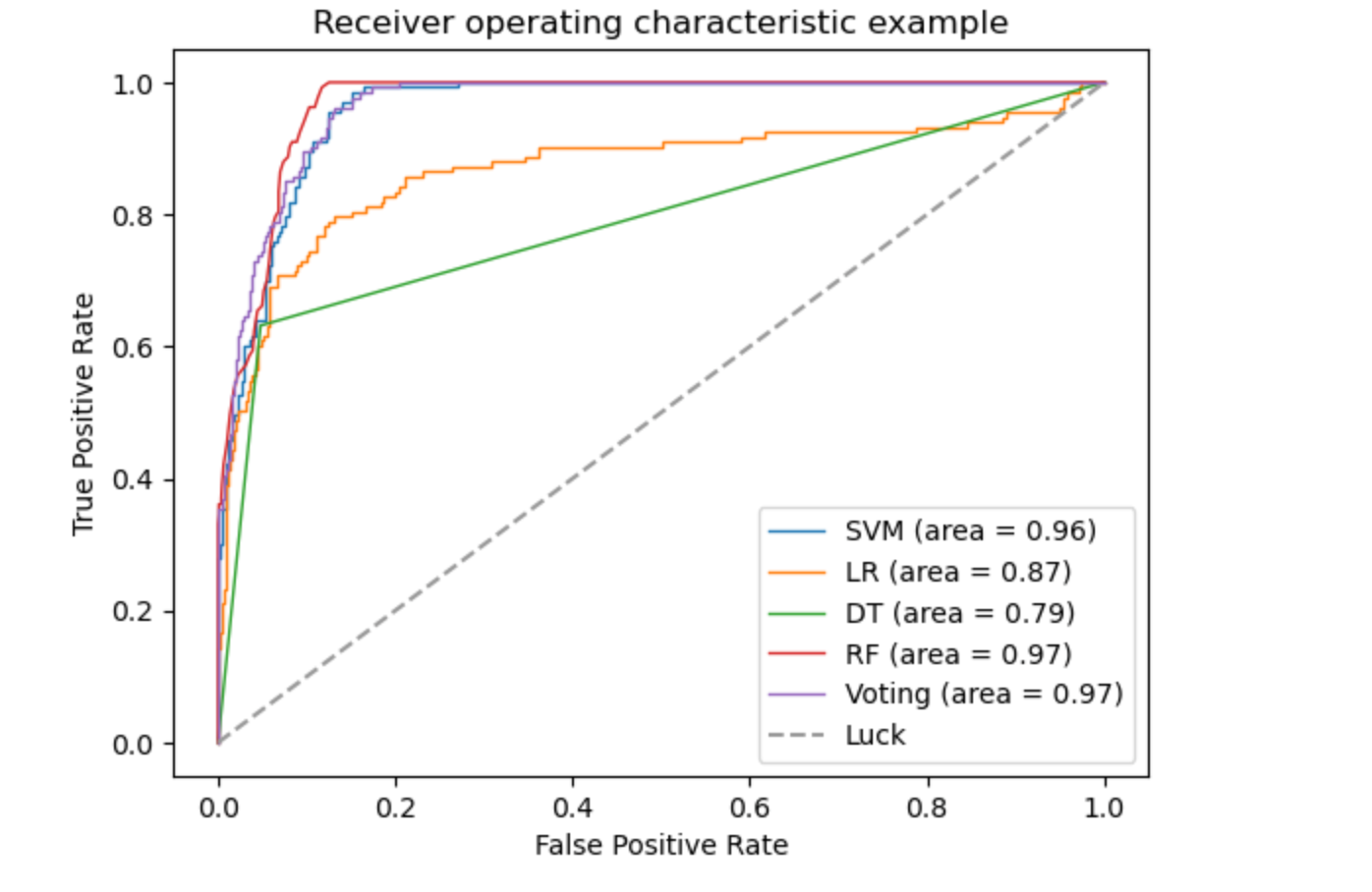
1. 选择合适的特征集合。



1. 生成多个分类模型，包括但不限于决策树，k-近邻，逻辑回归，SVM，Adaboost，Random Forest等，训练模型并优化各模型参数。
2. 对比各模型的accuray，precision，recall，F1-score。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | F1score | precesion | recall |
| K-means | 0.486508 | 0.137544 | 0.130731 | 0.176998 |
| 逻辑回归 | 0.800794 | 0.766212 | 0.798587 | 0.739730 |
| 决策树 | 0.801587 | 0.782645 | 0.786670 | 0.780691 |
| 随机森林 | 0.851587 | 0.822354 | 0.886115 | 0.774951 |
| 多种投票 | 0.843651 | 0.818433 | 0.785485 | 0.858552 |

1. 绘制各模型的ROC曲线，输出AUC。建议，尝试将多个模型的ROC绘制在一幅图中。



1. 根据上述指标，分析各模型的优、劣。

K-means的F1分数和准确度比其他算法都低，但精度和召回率都很低。K-means通常用于聚类问题，这里它被应用于分类问题，可能会导致表现不佳。

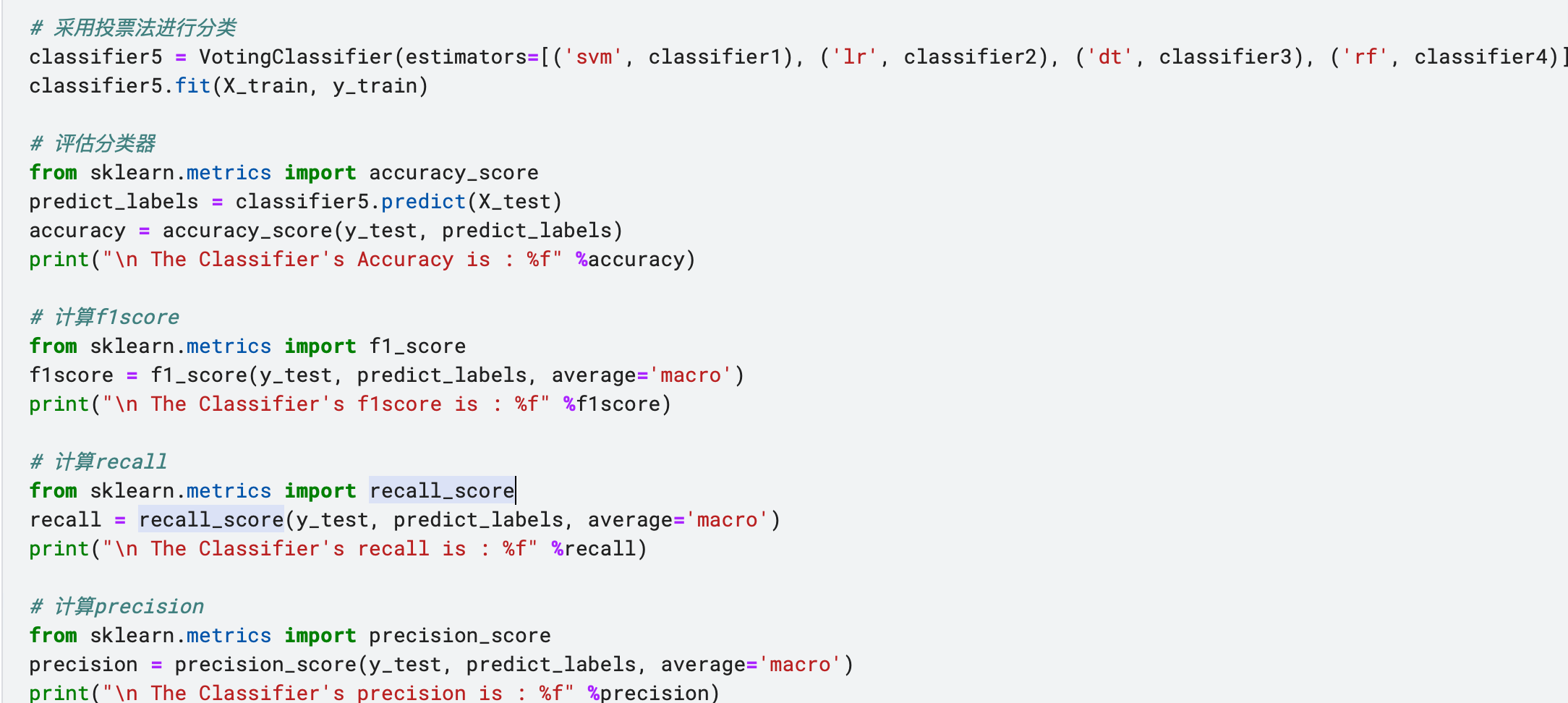
逻辑回归表现良好，准确度高达80.08％，同时F1得分较高，说明模型预测能力和误差较小。然而，召回率较低，这可能意味着模型在预测正类时漏掉了一些实际上是正类的数据。

决策树和随机森林都比逻辑回归的召回率更高，但随机森林的表现优于决策树。这是因为随机森林由多个决策树组成，并且具有更好的泛化能力和更少的过拟合风险。然而，随机森林的精度比其他算法低一些，可能因为该算法倾向于将数据分类到数量更多的类别中。

多总体而言，逻辑回归、决策树和随机森林表现较好，多种投票算法表现略高。我们需要考虑实际问题和业务需求来选择最适合的算法。

（7）尝试用sklearn.ensemble 中的VotingClassifier，组合几个训练好的分类模型，观察用投票的方式，是否可以改善分类性能。

种投票算法的F1分数和准确度都比逻辑回归略高，召回率最高，但精度最低。这是因为该算法在预测时将多个模型的输出结合起来，而不是单个模型的输出，因此它的表现通常比单个模型更好。但是，它的精度可能受到多个模型的错误预测的影响。



3．分析与讨论

（1）总结、讨论组合方法是否可以改进分类性能。

根据上面给出的评价指标，我们可以考虑使用组合方法来提高模型的性能。

一种简单的组合方法是通过加权平均值或投票来结合多个分类器的预测结果。这种方法的好处在于可以利用不同分类器的优势，从而提高整体性能。

在这个例子中，多种投票算法已经通过将多个模型的输出结合在一起来组合了多个分类器的预测结果。由于多种投票算法的F1分数和准确度都比逻辑回归略高，召回率最高，但精度最低，这表明它确实改善了模型的性能。

我们还可以尝试使用其他组合方法，如堆叠或深度组合，这些方法将多个分类器的预测结果馈送到一个或多个超级分类器中。这些方法可能需要更多的计算资源和调整超参数，但也可能进一步提高模型性能。

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('/Users/mac/Desktop/数据挖掘实验/实验五/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

# %%

y\_test

# %%

y

# %%

# 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]

# %%

X\_train.head()

# %%

# 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)

# %%

y\_train .head()

# %%

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')

# %% [markdown]

# # 采用K-means进行分类

# %%

from sklearn.cluster import KMeans

from sklearn import metrics

# %%

# 选择最优的K值

# 评估不同K值的聚类效果

from matplotlib import pyplot as plt

K = range(2, 10)

meandistortions = []

for k in K:

kmeans = KMeans(n\_clusters=k)

kmeans.fit(X\_train)

meandistortions.append(sum(np.min(cdist(X\_train, kmeans.cluster\_centers\_, 'euclidean'), axis=1)) / X\_train.shape[0])

# 绘制K值与误差平方和的关系图

plt.plot(K, meandistortions, 'bx-')

plt.xlabel('k')

plt.ylabel('Average distortion')

plt.title('Selecting k with the Elbow Method')

plt.show()

# %%

# k=2 计算 accuracy precision recall f1score

kmeans = KMeans(n\_clusters=2)

kmeans.fit(X\_train)

y\_pred = kmeans.predict(X\_test)

print(y\_pred)

# 计算accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

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

# 计算f1score

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

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

# 计算precision

precision = precision\_score(y\_test, y\_pred, average='macro')

print("\n The Kmeans's precision is : %f" %precision)

# 计算recall

recall = recall\_score(y\_test, y\_pred, average='macro')

print("\n The Kmeans's recall is : %f" %recall)

# %%

# 采用逻辑回归进行分类

from sklearn.linear\_model import LogisticRegression

# 采用逻辑回归进行分类

classifier2 = LogisticRegression()

classifier2.fit(X\_train, y\_train)

# 评估分类器

from sklearn.metrics import accuracy\_score

predict\_labels = classifier2.predict(X\_test)

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

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

# 计算f1score

from sklearn.metrics import f1\_score

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

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

# 计算precision

from sklearn.metrics import precision\_score

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

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

# 计算recall

from sklearn.metrics import recall\_score

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

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

# %%

# 采用决策树进行分类

from sklearn.tree import DecisionTreeClassifier

# 采用决策树进行分类

classifier3 = DecisionTreeClassifier()

classifier3.fit(X\_train, y\_train)

# 评估分类器

from sklearn.metrics import accuracy\_score

predict\_labels = classifier3.predict(X\_test)

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

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

# 计算f1score

from sklearn.metrics import f1\_score

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

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

# 计算precision

from sklearn.metrics import precision\_score

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

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

# 计算recall

from sklearn.metrics import recall\_score

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

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

# %%

# 采用随机森林进行分类

from sklearn.ensemble import RandomForestClassifier

# 采用随机森林进行分类

classifier4 = RandomForestClassifier()

classifier4.fit(X\_train, y\_train)

# 评估分类器

from sklearn.metrics import accuracy\_score

predict\_labels = classifier4.predict(X\_test)

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

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

# 计算f1score

from sklearn.metrics import f1\_score

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

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

# 计算precision

from sklearn.metrics import precision\_score

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

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

# 计算recall

from sklearn.metrics import recall\_score

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

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

# %%

# 多种分类器进行投票得到最终结果

from sklearn.ensemble import VotingClassifier

# 采用投票法进行分类

classifier5 = VotingClassifier(estimators=[('svm', classifier1), ('lr', classifier2), ('dt', classifier3), ('rf', classifier4)], voting='soft')

classifier5.fit(X\_train, y\_train)

# 评估分类器

from sklearn.metrics import accuracy\_score

predict\_labels = classifier5.predict(X\_test)

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

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

# 计算f1score

from sklearn.metrics import f1\_score

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

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

# 计算recall

from sklearn.metrics import recall\_score

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

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

# 计算precision

from sklearn.metrics import precision\_score

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

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

# %% [markdown]

# 绘制各模型的ROC曲线，输出AUC。建议，尝试将多个模型的ROC绘制在一幅图中。

# %%

# %%

y\_test\_new=y\_test

# %%

#绘制各模型的ROC曲线，输出AUC。建议，尝试将多个模型的ROC绘制在一幅图中。

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

# 绘制ROC曲线

fpr, tpr, thresholds = roc\_curve(y\_test, classifier1.predict\_proba(X\_test)[:,1], pos\_label=1)

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, lw=1, label='SVM (area = %0.2f)' % (roc\_auc))

fpr, tpr, thresholds = roc\_curve(y\_test, classifier2.predict\_proba(X\_test)[:,1], pos\_label=1)

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, lw=1, label='LR (area = %0.2f)' % (roc\_auc))

fpr, tpr, thresholds = roc\_curve(y\_test, classifier3.predict\_proba(X\_test)[:,1], pos\_label=1)

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, lw=1, label='DT (area = %0.2f)' % (roc\_auc))

fpr, tpr, thresholds = roc\_curve(y\_test, classifier4.predict\_proba(X\_test)[:,1], pos\_label=1)

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, lw=1, label='RF (area = %0.2f)' % (roc\_auc))

fpr, tpr, thresholds = roc\_curve(y\_test, classifier5.predict\_proba(X\_test)[:,1], pos\_label=1)

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, lw=1, label='Voting (area = %0.2f)' % (roc\_auc))

plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')

plt.xlim([-0.05, 1.05])

plt.ylim([-0.05, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic example')

plt.legend(loc="lower right")

plt.show()