1．实验目的

（1）掌握k-均值聚类的应用

（2）掌握基于密度的聚类dbscan方法

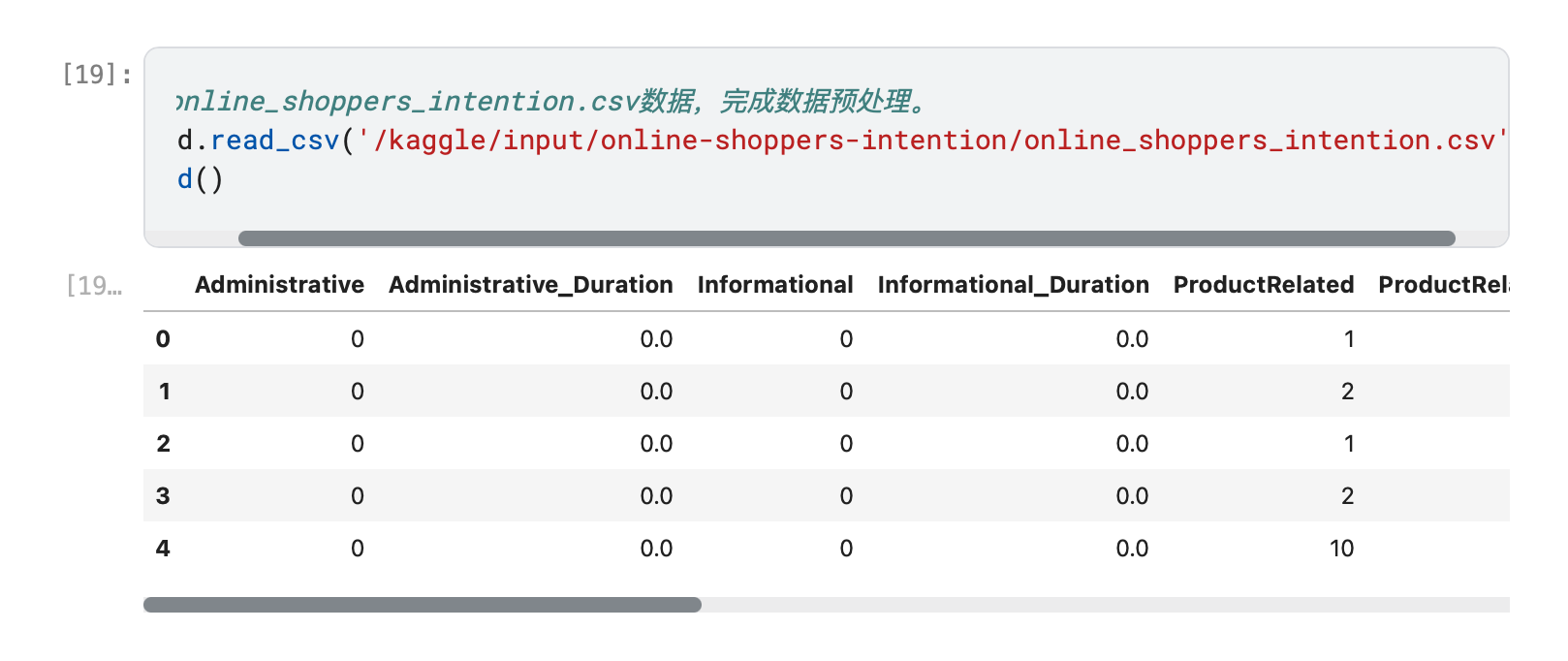
（3）掌握层次聚类

（4）能够对聚类结果进行适当的可视化

（5）掌握聚类性能评价方法

2．实验要求和步骤

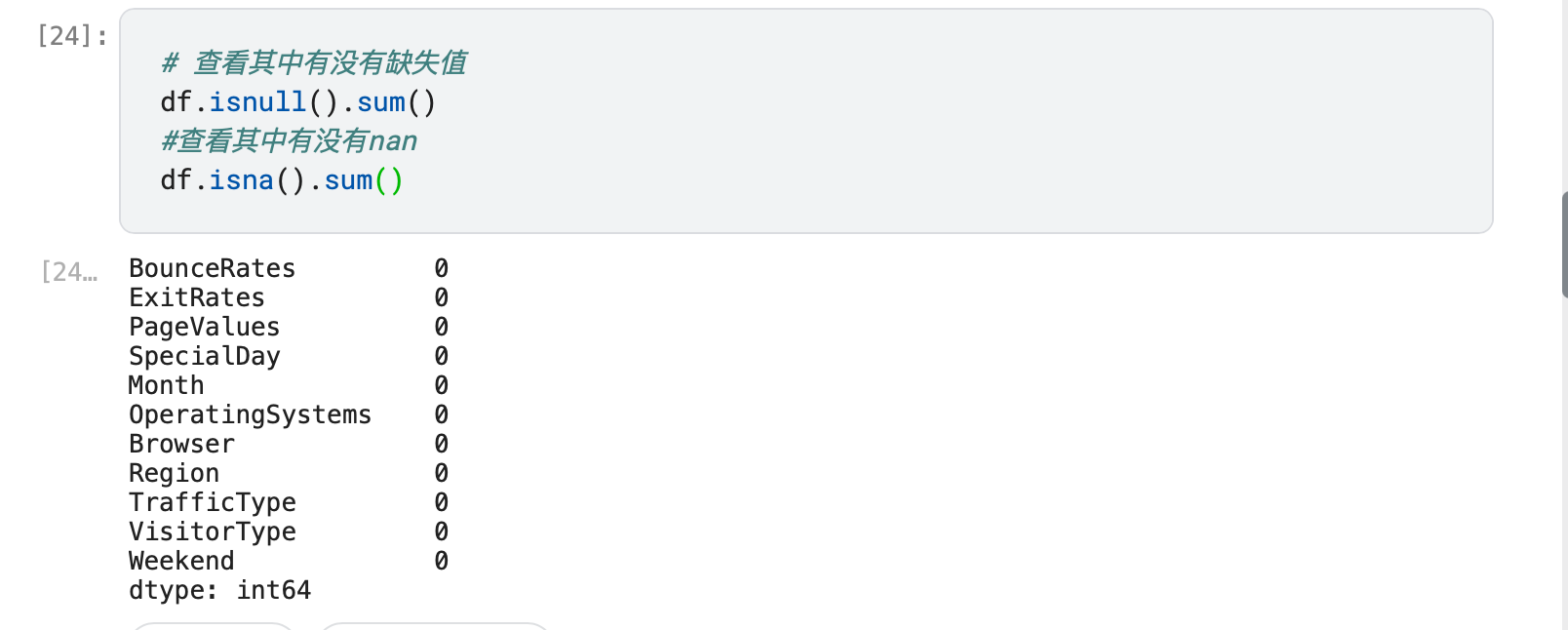
（1）读取online\_shoppers\_intention.csv数据，完成数据预处理。



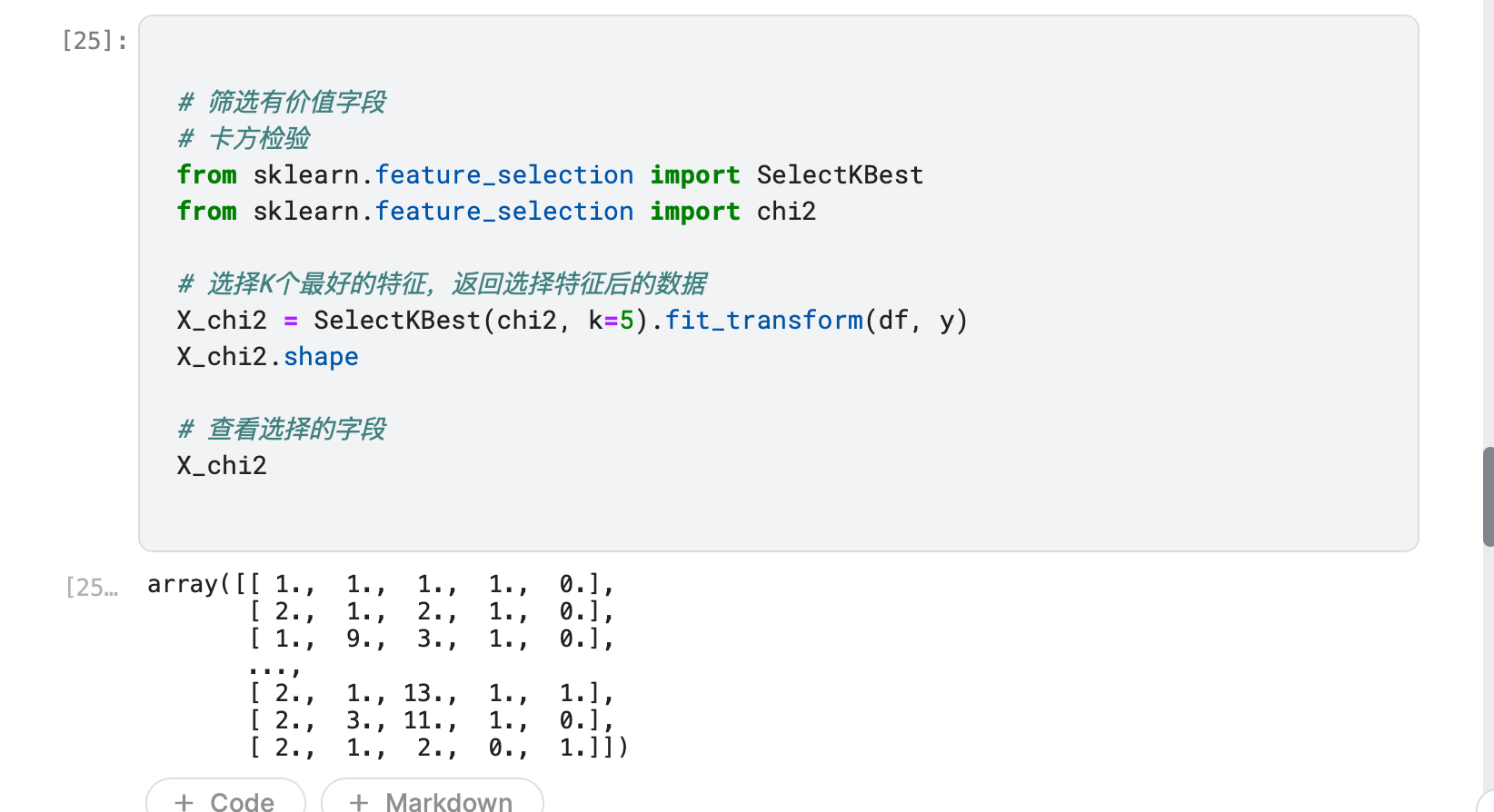


1. 去除字段Revenue（该字段适合做分类标签）。
2. 筛选有价值的字段，采用至少两种聚类算法，对该数据集聚类。

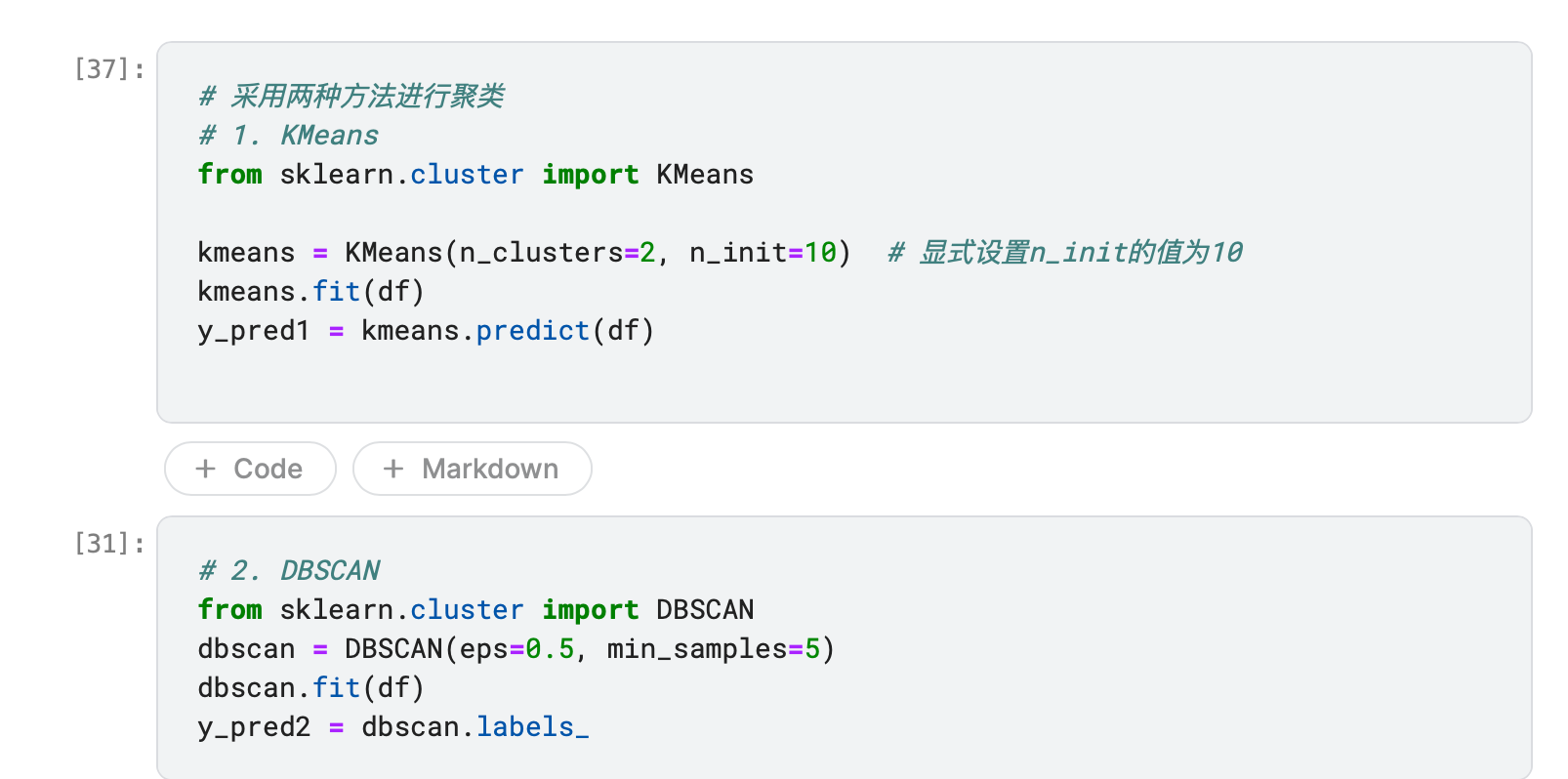




1. 注意合理选择聚类数目。



（5）利用聚类评价方法，评价聚类结果。



对于两种聚类的效果进行评估：



3．分析与讨论

（1）讨论聚类与分类方法的优缺点和应用范围。

1.K均值聚类（K-means Clustering）：

K均值是一种迭代聚类算法，它将数据点分为K个不同的簇，每个簇代表一个聚类中心。以下是K均值聚类的主要特点：

算法步骤：

(1)选择初始聚类中心，将每个数据点分配到最接近的聚类中心，然后重新计算聚类中心，并重复这个过程直到收敛。

(2)需要指定簇的数量K。

(3)假设簇是球形的，每个数据点都被分配到一个簇中。

(4)对于大型数据集，K均值计算效率高，但对异常值敏感。

(5)适合于数据集具有明显分离的簇的情况。

2.DBSCAN（Density-Based Spatial Clustering of Applications with Noise）：

DBSCAN是一种基于密度的聚类算法，它通过识别具有足够密度的区域并将其合并为簇来进行聚类。以下是DBSCAN的主要特点：

算法步骤：

(1)通过选择一个数据点并检查其邻域中的点来构建簇。如果邻域内的点密度足够高，将形成一个簇，否则将被标记为噪声或者边界点。

(2)不需要预先指定簇的数量，能够自动检测簇的数量。

(3)对于任意形状和大小的簇都有效，不依赖于数据点之间的距离或簇的形状。

(4)能够处理噪声和异常值。

(5)对于具有不同密度的簇和非球形簇的数据集效果较好。

总体而言，K均值适用于数据点间距离明显的球形簇，需要预先指定簇的数量；而DBSCAN适用于不规则形状和不同密度的簇，能够自动检测簇的数量。对于本数据集而言KMeans的效果更加优秀。

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('/kaggle/input/preprocess-train/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')

# %%

# 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

from scipy.spatial.distance import cdist

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()

# %%

# 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recall

from sklearn.metrics import recall\_score

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

# %%

# 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recall

from sklearn.metrics import recall\_score

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

# %%

# 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recall

from sklearn.metrics import recall\_score

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

# %%

# 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)

# %%

# 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。

# 绘制ROC曲线

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

# 绘制ROC曲线

def plot\_roc\_curve(fpr, tpr, label=None):

plt.plot(fpr, tpr, linewidth=2, label=label)

plt.plot([0, 1], [0, 1], 'k--') # Dashed diagonal

plt.axis([0, 1, 0, 1]) # 范围

plt.xlabel('False Positive Rate') # x轴标签

plt.ylabel('True Positive Rate') # y轴标签

plt.legend(loc="lower right") # 图例位置

# 将多个模型的ROC绘制在一幅图中

plt.figure(figsize=(8, 6)) # 设置画布大小

for clf in (classifier1, classifier2, classifier3, classifier4, classifier5):

y\_scores = clf.predict\_proba(X\_test)

fpr, tpr, thresholds = roc\_curve(y\_test, y\_scores[:, 1])

plot\_roc\_curve(fpr, tpr, clf.\_\_class\_\_.\_\_name\_\_)

plt.show()