电 子 科 技 大 学 实 验 报 告

课程名称： Python语言程序设计及其应用

实验地点： 科A229

指导教师： 张勇

评 分：

完成实验学生信息：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序** | **姓名** | **学号** | **选课**  **序号** | **贡献**  **百分比(%)** | **备注**  **（主要工作**） |
| 1 | 付文亮 | 2022080401004 | 69 | 33.3 | 1.搜集第三个算法并整理  2.分析实验改进措施 |
| 2 | 傅若山 | 2022080401023 | 71 | 33.3 | 1.搜集第二个算法并整理  2.参与选取异常数据分类算法的讨论 |
| 3 | 苏徐涛 | 2022080401011 | 70 | 33.3 | 1.搜集第一个算法并整理  2.参与选取异常数据分类算法的讨论 |
| 4 |  |  |  |  | 1.  2. |

1. 学生人数按照任课教师要求限定；
2. 对于“评价、改进、总结和体会”都要认真填写，和其他内容是评价实验成绩的重要参考。

**实验题目名称: 数据挖掘实验**

1. **实验内容**

数据挖掘算法

请搜集不少于2种异常数据分类算法，并用开源数据集（或者自行模拟数据）测试两种算法应用效果。要求给出算法的步骤、实验程序、实验程序、结果分析。

1. **实验目的**

搜集不少于2种异常数据分类算法，并用开源数据集（或者自行模拟数据）测试两种算法应用效果。

1. **实验过程**

（如果有多个问题，请对每个问题分别整理出下列内容）

目 录

[**1.** **实验1：基于距离的异常数据分类算法** 4](#_Toc134969045)

[1.1. 问题分析 4](#_Toc134969046)

[1.2. 系统设计与算法设计 4](#_Toc134969047)

[1.3. 编写程序 4](#_Toc134969048)

[1.4. 运行结果 5](#_Toc134969049)

[1.5. 实验结果分析 5](#_Toc134969050)

[1.6. 优缺点及改进方向 6](#_Toc134969051)

[1.7. 心得体会与总结 6](#_Toc134969052)

[**2.** **实验2：聚类分析方法** 6](#_Toc134969053)

[2.1 问题分析 6](#_Toc134969054)

[2.2 系统设计与算法设计 6](#_Toc134969055)

[2.3编写程序 6](#_Toc134969056)

[2.4 运行结果 8](#_Toc134969057)

[2.5 优缺点及改进方向 8](#_Toc134969058)

[2.6心得体会与总结 8](#_Toc134969059)

[**3.** **实验3：异常数据分类算法——孤立森林算法** 8](#_Toc134969060)

[3.1. 问题分析 8](#_Toc134969061)

[3.2. 系统设计与算法设计 9](#_Toc134969062)

[3.3. 编写程序 9](#_Toc134969063)

[3.4. 运行结果 10](#_Toc134969064)

[3.5. 实验结果分析 10](#_Toc134969065)

[3.6. 优缺点及改进方向 10](#_Toc134969066)

[3.7. 心得体会与总结 11](#_Toc134969067)

[**1.** **问题重述** 11](#_Toc134969068)

[**2.** **对本次实验的设计提出改进意见** 11](#_Toc134969069)

[**3.** **附件** 11](#_Toc134969070)

[附件1.基于距离的异常数据分类算法的程序 11](#_Toc134969071)

[附件2.聚类算法的程序 12](#_Toc134969072)

[附件3.孤立森林算法的程序 13](#_Toc134969073)

# **实验1：基于距离的异常数据分类算法**

## 问题分析

异常数据分类算法多种多样，其中基于距离的异常数据分类算法是一种简单但有效的异常数据检测方法。

基于距离的异常数据分类算法是一种基于距离度量的异常检测方法。该算法的基本思想是通过计算数据点与其最近邻之间的距离来检测异常值。具体来说，该算法将一个数据点的异常程度定义为其与最近邻之间的距离与平均距离之比，即异常系数。如果一个数据点的异常系数大于某个预设的阈值，则该数据点被认为是异常值。

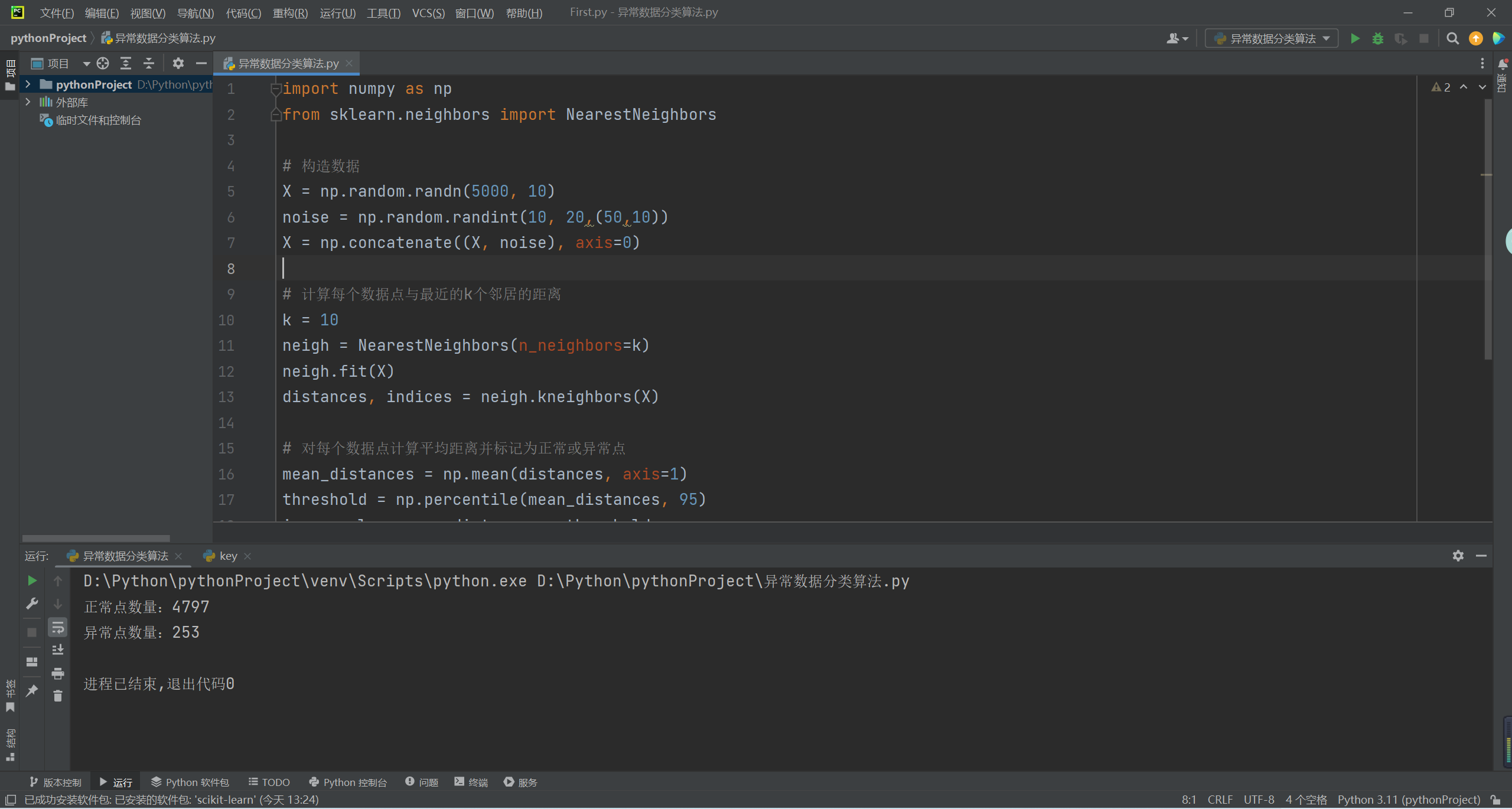
## 系统设计与算法设计

先用随机函数构造数据，计算每个数据点与最近的k个邻居的距离，然后对每个数据点计算平均距离并标记为正常点或异常点，最后输出结果。

## 编写程序

import numpy as np  
from sklearn.neighbors import NearestNeighbors  
  
# 构造数据  
X = np.random.randn(5000, 10)  
noise = np.random.randint(10, 20,(50,10))  
X = np.concatenate((X, noise), axis=0)  
  
# 计算每个数据点与最近的k个邻居的距离  
k = 10  
neigh = NearestNeighbors(n\_neighbors=k)  
neigh.fit(X)  
distances, indices = neigh.kneighbors(X)  
  
# 对每个数据点计算平均距离并标记为正常或异常点  
mean\_distances = np.mean(distances, axis=1)  
threshold = np.percentile(mean\_distances, 95)  
is\_anomaly = mean\_distances > threshold  
  
# 输出结果  
print(f"正常点数量：{len(X[~is\_anomaly])}")  
print(f"异常点数量：{len(X[is\_anomaly])}")

## 运行结果

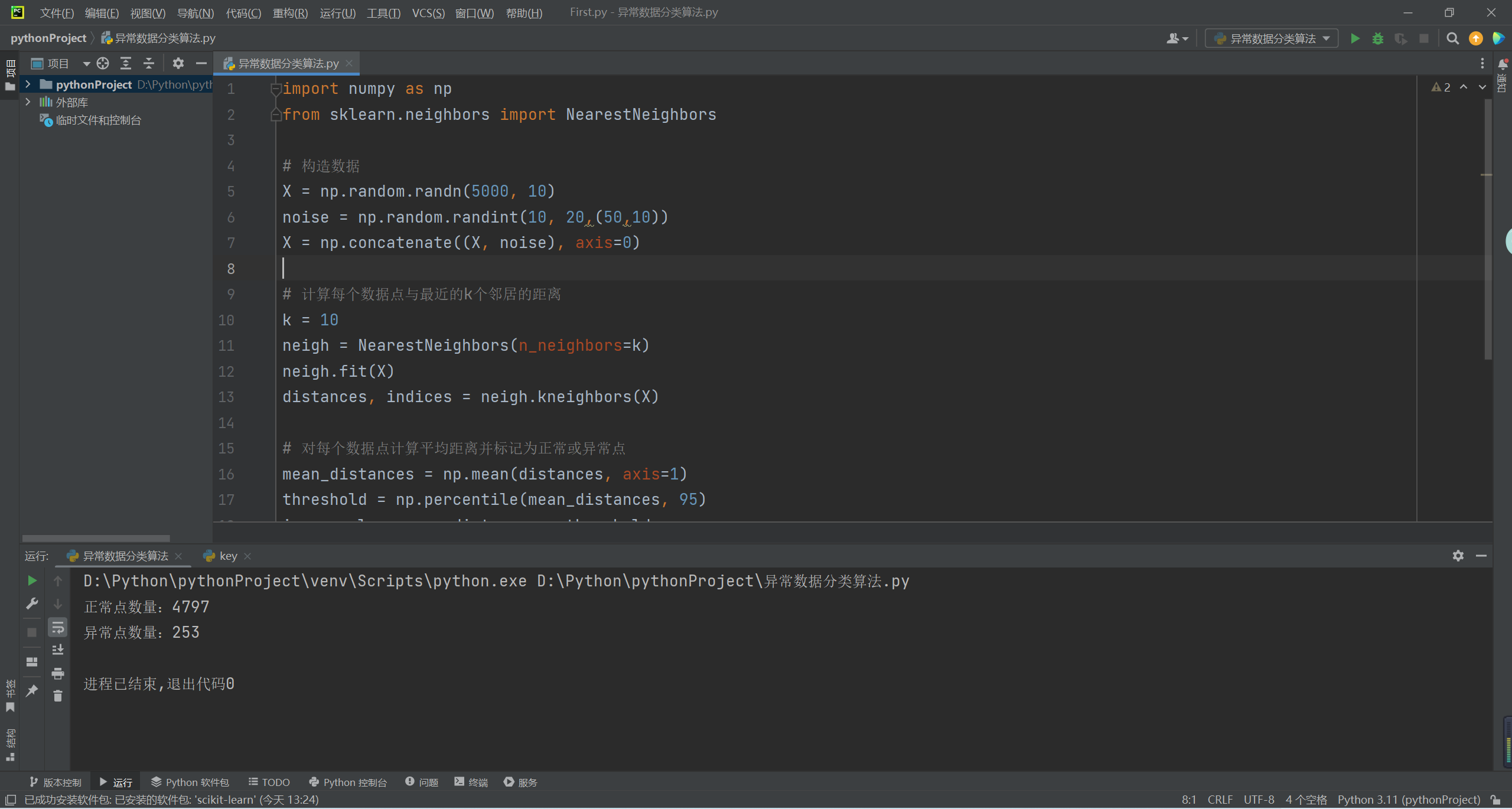


正常点数量：4797

异常点数量：253

## 实验结果分析

(展示运行结果,对结果进行分析,甚至检验,程序作为附件,附在本doc文档后面;另外邮件包含原始程序文件.)



该程序用大量的随机数据模拟数据集。输出结果展示了正常点和异常点的数量，比较直观地展示了该算法的分类效果。

## 优缺点及改进方向

优点：该代码作为基于距离的异常数据分类算法的演示代码，比较短小精悍且易于理解， 逻辑清晰，思路明确。

缺点：仅仅演示了算法中最简单的内容，解释了算法的运行机制，还不能在实际场景中 应用。

改进方向：编写类，把函数改写成方法，增加更多方法和数据，使之能满足现实的需要。

## 心得体会与总结

该实验让我第一次接触了与数据有关的算法，第一次接触了scikit-learn这个有关深度学习的库，拓展了自己的视野。通过从网上查阅和学习相关资料，增强了自学能力和代码理解能力。

# **实验2：聚类分析方法**

## 2.1 问题分析

此类方法利用聚类算法（如K-means、\_x0008\_DBSCAN等）对待检测数据进行聚类，通过聚类的结果来分辨正常与异常的数据，是一种典型的非监督式异常检测技术。

## 2.2 系统设计与算法设计

通常来讲，基于聚类的异常检测可基于三种假设来分辨异常数据：

1. 1. 不属于任何簇（Cluster）的数据即为异常；
2. 2. 距离簇中心很远的数据即为异常；
3. 3. 归属于数据点少或稀疏簇的数据即为异常。

## 2.3编写程序

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import mglearn

from sklearn.preprocessing import LabelEncoder

import warnings

warnings.filterwarnings('ignore')

import seaborn as sns

from sklearn.datasets import make\_blobs

from matplotlib import pyplot

plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus']=False #用来正常显示负号

# 生成测试聚类分析代码需要的随机数据

# 生成随机数

data,target=make\_blobs(n\_samples=100000,n\_features=10,centers=10)

# 给随机数定义列名

a = []

for i in range(10):

a.append('X'+str(i))

data = pd.DataFrame(data)

data.columns = a

# 在二维图中绘制样本，每个样本颜色不同

# 每次只能选取2个特征绘图（二维图像）

pyplot.scatter(data[:,1],data[:,0],c=target);

pyplot.show()

# 输出聚类中心

from sklearn.cluster import KMeans

k = 4 # 定义聚类的类别中心个数，即聚成4类

iteration = 500 # 计算聚类中心的最大循环次数

model = KMeans(n\_clusters = k,n\_jobs = 4,max\_iter = iteration)

model.fit(data)

r1 = pd.Series(model.labels\_).value\_counts()

r2 = pd.DataFrame(model.cluster\_centers\_)

r = pd.concat([r2,r1],axis=1)

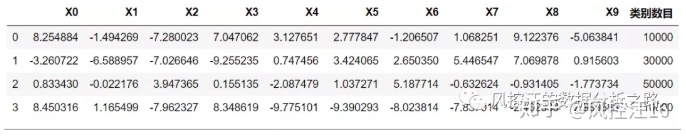
r.columns = list(data.columns)+[u'所属类别数目']

# 给每行数据标记所属类别

r = pd.concat([data,pd.Series(model.labels\_,index=data.index)],axis=1)

r.columns = list(data.columns)+[u'所属类别数目']

## 2.4 运行结果



## 2.5 优缺点及改进方向

优点： 算法简单，时间复杂度低。缺点： 无法处理非球形等不规则的数据、对初始值k的设置很敏感、对离群点敏感。

## 2.6心得体会与总结

体会了聚类算法在处理异常数据中的强大功能，拓展了自己的事业，增强了我对数据挖掘方面算法的兴趣。

# **实验3：异常数据分类算法——孤立森林算法**

## 问题分析

孤立森林算法是适用于连续数据异常检测的一种异常数据分类算法，在统计学中，数据空间里面，分布稀疏的区域表示数据发生在此区域的概率很低，因此可以认为落在这些区域里的数据是异常的。据此设计出的算法便形象地称为孤立森林算法。

## 系统设计与算法设计

先用随机超平面来切割数据空间，切一次可以生成两个子空间。然后继续用随机超平面来切割每个子空间，重复操作直到每个子空间内只有一个数据点为止。最终密度很高的簇会被切割很多次才停止切割，但密度很低的点切割几次便停止了，依此便能将那些很早就切割停止的点判断为异常数据。

## 编写程序

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import IsolationForest

rng = np.random.RandomState(42)

# Generate train data

X = 0.3 \* rng.randn(100, 2)

X\_train = np.r\_[X + 2, X - 2]

# Generate some regular novel observations

X = 0.3 \* rng.randn(20, 2)

X\_test = np.r\_[X + 2, X - 2]

# Generate some abnormal novel observations

X\_outliers = rng.uniform(low=-4, high=4, size=(20, 2))

# fit the model

clf = IsolationForest(max\_samples=100, random\_state=rng)

clf.fit(X\_train)

y\_pred\_train = clf.predict(X\_train)

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

y\_pred\_outliers = clf.predict(X\_outliers)

# plot the line, the samples, and the nearest vectors to the plane

xx, yy = np.meshgrid(np.linspace(-5, 5, 50), np.linspace(-5, 5, 50))

Z = clf.decision\_function(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.title("IsolationForest")

plt.contourf(xx, yy, Z, cmap=plt.cm.Blues\_r)

b1 = plt.scatter(X\_train[:, 0], X\_train[:, 1], c='white',

s=20, edgecolor='k')

b2 = plt.scatter(X\_test[:, 0], X\_test[:, 1], c='green',

s=20, edgecolor='k')

c = plt.scatter(X\_outliers[:, 0], X\_outliers[:, 1], c='red',

s=20, edgecolor='k')

plt.axis('tight')

plt.xlim((-5, 5))

plt.ylim((-5, 5))

plt.legend([b1, b2, c],

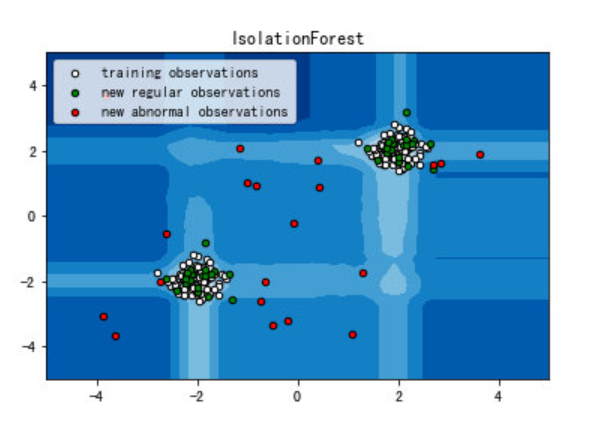
["training observations",

"new regular observations", "new abnormal observations"],

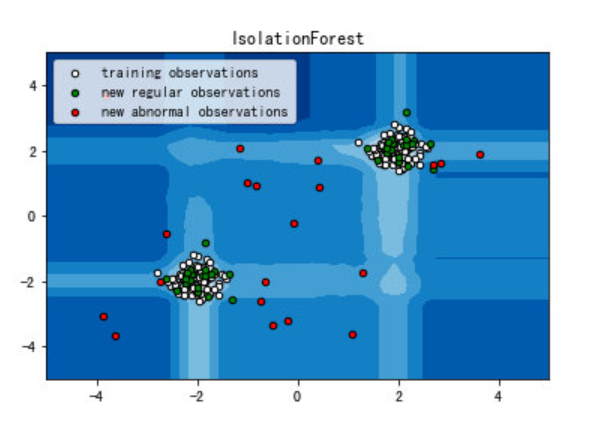
loc="upper left")

plt.show()

## 运行结果



## 实验结果分析



其中，红色的点为异常点，白色是训练集，绿色是测试数据。

## 优缺点及改进方向

优点:是一种高效的异常检测算法，可以有效地找到异常数据，并且错误率低。

缺点:需要引入sklearn库中的模块，代码进行超平面切割时每次选择划分属性和划分点都是随机的，不能根据信息确定，代码执行效率较低。

改进方向:将每次切割的超平面的信息与数据信息相关联，提高代码执行效率，同时简化代码复杂程度，减少对其他模块的依赖程度。

## 心得体会与总结

心得体会:了解了新的异常数据检测算法，知道了可以通过不断分割的方式判断数据的密集程度进而推断出数据是否异常，同时对统计学也有了更深一步的理解，学会将统计学知识与算法相结合来思考解决问题的办法。

总结:当思考一个问题时可以与其他学科相结合共同探索问题解决办法，对于异常数据分类算法问题中，孤立森林算法不乏是一个有效的算法。

# **问题重述**

请搜集不少于2种**异常数据**分类算法，并用开源数据集（或者自行模拟数据）测试两种算法应用效果。

要求给出算法的步骤、实验程序、实验程序、结果分析。

在实际问题中涉及大量数据的处理中学习异常数据分类算法可以提高数据的管理效率，使异常数据可以及时被发现，减少异常数据的使用带来最终结果的偏差。

# **对本次实验的设计提出改进意见**

(向教师反馈实验的设计，例如，修改实验任务使得更能体现能力、水平；增加实验任务等等；所有围绕实验的建议、意见。)

将题目具体化，即给出数据集和测试结果，让学生可以判断代码执行是否正确。给出一种异常数据分类算法的应用，加深学生对异常数据界定的理解从而加强学生的学习效率。

# **附件**

## 附件1.基于距离的异常数据分类算法的程序

import numpy as np  
from sklearn.neighbors import NearestNeighbors  
  
# 构造数据  
X = np.random.randn(5000, 10)  
noise = np.random.randint(10, 20,(50,10))  
X = np.concatenate((X, noise), axis=0)  
  
# 计算每个数据点与最近的k个邻居的距离  
k = 10  
neigh = NearestNeighbors(n\_neighbors=k)  
neigh.fit(X)  
distances, indices = neigh.kneighbors(X)  
  
# 对每个数据点计算平均距离并标记为正常或异常点  
mean\_distances = np.mean(distances, axis=1)  
threshold = np.percentile(mean\_distances, 95)  
is\_anomaly = mean\_distances > threshold  
  
# 输出结果  
print(f"正常点数量：{len(X[~is\_anomaly])}")  
print(f"异常点数量：{len(X[is\_anomaly])}")

## 附件2.聚类算法的程序

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import mglearn

from sklearn.preprocessing import LabelEncoder

import warnings

warnings.filterwarnings('ignore')

import seaborn as sns

from sklearn.datasets import make\_blobs

from matplotlib import pyplot

plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签

plt.rcParams['axes.unicode\_minus']=False #用来正常显示负号

# 生成测试聚类分析代码需要的随机数据

# 生成随机数

data,target=make\_blobs(n\_samples=100000,n\_features=10,centers=10)

# 给随机数定义列名

a = []

for i in range(10):

a.append('X'+str(i))

data = pd.DataFrame(data)

data.columns = a

# 在二维图中绘制样本，每个样本颜色不同

# 每次只能选取2个特征绘图（二维图像）

pyplot.scatter(data[:,1],data[:,0],c=target);

pyplot.show()

# 输出聚类中心

from sklearn.cluster import KMeans

k = 4 # 定义聚类的类别中心个数，即聚成4类

iteration = 500 # 计算聚类中心的最大循环次数

model = KMeans(n\_clusters = k,n\_jobs = 4,max\_iter = iteration)

model.fit(data)

r1 = pd.Series(model.labels\_).value\_counts()

r2 = pd.DataFrame(model.cluster\_centers\_)

r = pd.concat([r2,r1],axis=1)

r.columns = list(data.columns)+[u'所属类别数目']

# 给每行数据标记所属类别

r = pd.concat([data,pd.Series(model.labels\_,index=data.index)],axis=1)

r.columns = list(data.columns)+[u'所属类别数目']

## 附件3.孤立森林算法的程序

import numpy as np

import matplotlib.pyplot as plt

from sklearn.ensemble import IsolationForest

rng = np.random.RandomState(42)

# Generate train data

X = 0.3 \* rng.randn(100, 2)

X\_train = np.r\_[X + 2, X - 2]

# Generate some regular novel observations

X = 0.3 \* rng.randn(20, 2)

X\_test = np.r\_[X + 2, X - 2]

# Generate some abnormal novel observations

X\_outliers = rng.uniform(low=-4, high=4, size=(20, 2))

# fit the model

clf = IsolationForest(max\_samples=100, random\_state=rng)

clf.fit(X\_train)

y\_pred\_train = clf.predict(X\_train)

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

y\_pred\_outliers = clf.predict(X\_outliers)

# plot the line, the samples, and the nearest vectors to the plane

xx, yy = np.meshgrid(np.linspace(-5, 5, 50), np.linspace(-5, 5, 50))

Z = clf.decision\_function(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.title("IsolationForest")

plt.contourf(xx, yy, Z, cmap=plt.cm.Blues\_r)

b1 = plt.scatter(X\_train[:, 0], X\_train[:, 1], c='white',

s=20, edgecolor='k')

b2 = plt.scatter(X\_test[:, 0], X\_test[:, 1], c='green',

s=20, edgecolor='k')

c = plt.scatter(X\_outliers[:, 0], X\_outliers[:, 1], c='red',

s=20, edgecolor='k')

plt.axis('tight')

plt.xlim((-5, 5))

plt.ylim((-5, 5))

plt.legend([b1, b2, c],

["training observations",

"new regular observations", "new abnormal observations"],

loc="upper left")

plt.show()