南京郵電大學
Nanjing University of Posts and Telecommunications

Python程序设计（混合式）

for

张伯雷

南京邮电大学计算机学院，数据科学与工程系

https://bolei-zhang.github.io/course/python-ai.html

2024/12/6

目录

“近邻”与分类和聚类

- 人工智能能否根据事物的“**邻居（近邻）**”自动判断该事物所属的类别？

- K-近邻分类和K-均值聚类的主要区别在于近邻的类别是否已知。类别已知就叫“分类”，类别未知就叫“聚类”

- 分类：已知所有图片的类别，有一张新的动物的图片，需要预测其类别

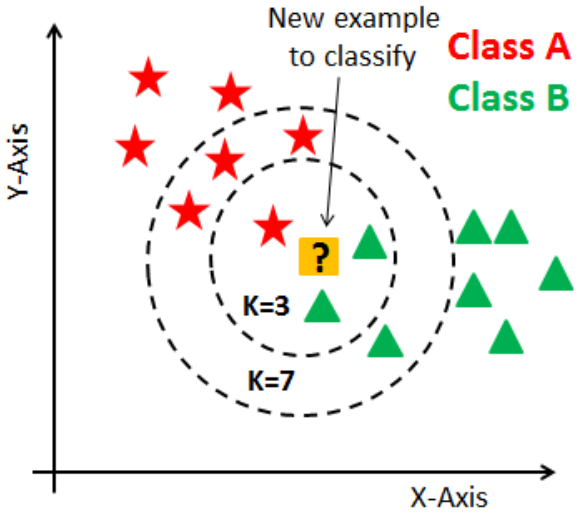
- 聚类：**并不了解图片的类别，需要将现有的图片划分为几类**，例如：爬行动物、哺乳动物、鸟类、鱼类...

K-近邻分类

- K-近邻分类（K Nearest Neighbors Classification，简称KNN）是一种分类方法

- 其基本思想是：已知一批数据集及其对应的类别标签（该数据集是训练数据集），为一批没有类别标签的测试数据预测其类别标签

- 具体做法是：将测试数据集中的样本与训练数据集中的所有样本**计算“距离”**，找到与测试样本距离**最近的前K个训练样本**，则该测试样本的预测类别就是这K个训练样本中**出现次数最多的那个类别**。



KNN算法的一般步骤

- 一般步骤
  - ✓ 第一步：计算测试样本到所有训练样本的距离
    - ⇢ 问题：样本之间的距离是怎么定义的?
  - ✓ 第二步：按距离对训练样本升序排列
  - ✓ 第三步：选取与测试样本距离最小的前K个训练样本
    - ⇢ 问题：如何选择合适的 K?
  - ✓ 第四步：计算各个类别在前K个训练样本中出现的频率
  - ✓ 第五步：返回出现频率最高的类别作为测试样本的预测类别
- 特点
  - ✓ KNN 算法不会形成一个模型函数（没有权重参数），这与线性回归、逻辑斯蒂分类不同
  - ✓ 线性回归和逻辑斯蒂分类的计算代价主要花费在训练阶段的模型拟合，模型拟合好后对测试数据进行预测的时间代价较小
  - ✓ KNN 算法不需要进行模型拟合，没有训练阶段（lazy learning），但是每次测试时，必须计算测试数据与所有训练样本的距离，因此测试阶段的计算代价非常大

自定义实现K-近邻分类

- 以房屋好卖预测问题为例
  - ✓ 由于两个特征属性“房屋单价”和“房屋面积”的取值范围差异较大，需进行归一化操作
  - ✓ 采用欧几里得距离计算样本之间的距离

$$Euc(\mathbf{x},\mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_{d-1} - y_{d-1})^2 + (x_d - y_d)^2}$$

  - ✓ 定义了一个K近邻分类函数 **knn_classifier()**
    - ⇢ 待预测的输入样例 **input**
    - ⇢ 训练数据 **trains**
    - ⇢ **trains** 所对应的类别标签 **classes**
    - ⇢ **k** 值大小
  - ✓ 对测试样例集中每个样例调用一次 knn_classifier()

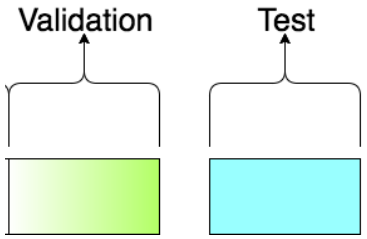
代码6.1

- 运行代码6.1，得到如下结果
  - ✓ 预测类别是：1 1 0 1 1　实际类别是：[1 0 1 1 1]

- 从输出结果可以看出，有两个样本预测错误，因此正确率是 0.6、精准率是 0.75、召回率是 0.75，要高于第四章逻辑斯蒂的分类结果。虽然 K 近邻分类方法直观简单，但是分类效果较好，在许多现实问题中，会首先尝试用 K 近邻分类方法作为一种基准方法。

- 思考：**不同的 K 值可能会导致不同的分类结果。**将以上代码中K值分别设置为 1 到 10，运行代码观察分类结果，说明哪个 K 值能达到最佳分类效果，为什么?

- 暴力法



KNN的三个基本要素

- 以上例子中，距离度量方法采用欧氏距离，K 值为 6，分类决策规则采用“多数表决法”。

- KNN的三基本要素
  - ✓ 距离度量
  - ✓ K值选择
  - ✓ 分类决策规则

距离度量

- L_p距离（L_p distance）或闵可夫斯基距离（Minkowski distance）

$$L_p(\mathbf{x},\mathbf{y}) = \sqrt[p]{|x_1 - y_1|^p + |x_2 - y_2|^p + \cdots + |x_{d-1} - y_{d-1}|^p + |x_d - y_d|^p}$$

$$L_p(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d} |x_1 - y_1|^p\right)^{\frac{1}{p}}$$

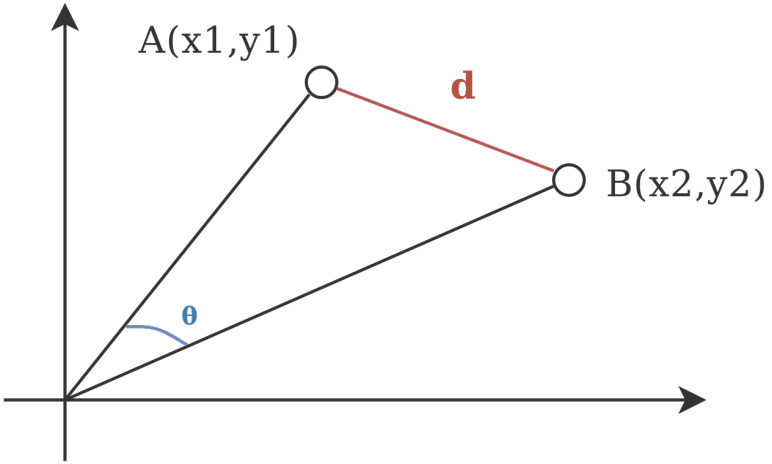
- 当p=2时，即是欧氏距离
- 当p=1时，是曼哈顿距离（Manhattan Distance）
- 当p=∞时，闵可夫斯基距离就变成各个坐标距离的最大值

距离度量

- 范数距离
  - ✓ 不同的距离度量方法所确定的最近邻居是不同的，x=(1, 1)，y=(5, 1)，z=(4, 4)
  - ✓ 当p=1或2时，y是x的最近邻居，当p=3时，z是x的最近邻居

- 余弦相似度（Cosine Distance）
  - ✓ 余弦相似度用**向量空间中两个向量夹角的余弦值**作为衡量两个样本间差异的大小

$$\cos ine(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{d} x_i \bullet y_i}{\sqrt{\sum_{i=1}^{d} x_i^{\,2}} \bullet \sqrt{\sum_{i=1}^{d} y_i^{\,2}}}$$



K值选择

- K值选择对K近邻分类会有重大影响
- 如果选择较小K值，只有与输入样本较近的样本才起作用，其它训练样例会因为距离远而没有“投票权”
  - ✓ 其缺点是预测结果对近邻的样例非常敏感，如果近邻恰巧是噪声，预测会出错，因此K值减小意味着容易发生过拟合
- 如果选择较大的K值，具有“投票权”的训练样例数量会增多，可能导致与输入样本不怎么相似的训练样例也会起作用，使得预测发生错误
  - ✓ 如果K=M（训练样本总数量），则无论输入样例是什么，都将简单地预测为训练样本集中样本数量最多的那个类别
- **K值一般取较小数值，可通过多次实验确定较好K值**

分类决策规则

- K近邻分类最常用的决策规则是“多数表决法”，即由K个最靠近的训练样例中的多数类决定输入样例的类别。此外，邻居的投票权重还可以根据距离进行加权，越近的邻居所起的作用越大，距离远的邻居的投票权重就小一些。

Scikit-learn实现KNN

- 类名：sklearn.neighbors.NearestNeighbors
- 第一步：创建NearestNeighbors对象
  - ✓ model = NearestNeighbors(n_neighbors, algorithm, metric)
  - ✓ n_neighbors是需要手动指定的K值
  - ✓ algorithm参数表示KNN的实现算法，有如下三种选择：
    - ⇢ "brute"：暴力法
    - ⇢ "kd_tree"：KD树
    - ⇢ "ball_tree"：在一系列嵌套的超球体上分割数据，用于改进KD树的二叉树结构
  - ✓ metric参数表示KNN的距离计算方法，选择如下：
    - ⇢ "minkowski"：闵可夫斯基距离（默认值）
    - ⇢ "euclidean"：KD树
    - ⇢ "manhattan"：曼哈顿距离
    - ⇢ "cosine"：余弦距离

Scikit-learn实现KNN

- 第二步：拟合模型，使用语句 model.fit(trainData)

- 第三步：对测试样例进行预测，distances, indices = model.kneighbors(testData)，其中distances是K个最近邻训练样本到输入样本的距离，indices是对应样本在训练集中的下标

代码6.2

```
#代码5.2 使用Scikit-learn类库对房屋好卖进行预测
import numpy as np
from sklearn.neighbors import NearestNeighbors
def normalize ( X, mean, std ) :#对数据进行归一化的函数
    return ( X-mean ) / std
xTrain = np.array([[3.32, 94], [3.05, 120], [3.70, 160], [3.52, 170], [3.60, 155],[3.36, 78], [2.70, 75],  [2.90, 80],
[3.12, 100], [2.80, 125]])
yTrain = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
mean = xTrain.mean ( axis = 0 ) #训练数据平均值
std = xTrain.std ( axis = 0 )  #训练数据方差
xTrain = normalize ( xTrain, mean, std ) #归一化训练数据
xTest = np.array ( [ [ 3.00, 100 ], [ 3.25, 93 ], [ 3.63, 163 ], [ 2.82, 120 ], [ 3.37, 89 ] ] )
xTest = normalize ( xTest, mean, std ) #归一化测试数据
yTest = np.array ( [ 1, 0, 1, 1, 1 ] )

#创建模型，使用kd-tree算法进行存储和查找
model = NearestNeighbors ( n_neighbors = 3, algorithm = "kd_tree", metric = "euclidean" )
model.fit ( xTrain )
distances, indices = model.kneighbors ( xTest )
print ( "最近邻的下标是：", indices )
print ( "对应的最近K个类别是：", yTrain [ indices ] )
```

最近邻的下标是：

[[8 1 7] [0 8 5] [2 4 3] [9 1 8] [0 5 8]]

对应的最近K个类别是：

[[1 0 1] [0 1 1] [0 0 0] [1 0 1] [0 1 1]]

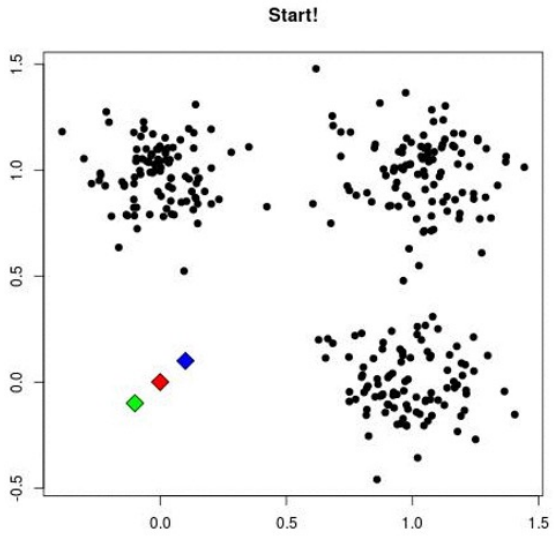
KNN的优缺点分析

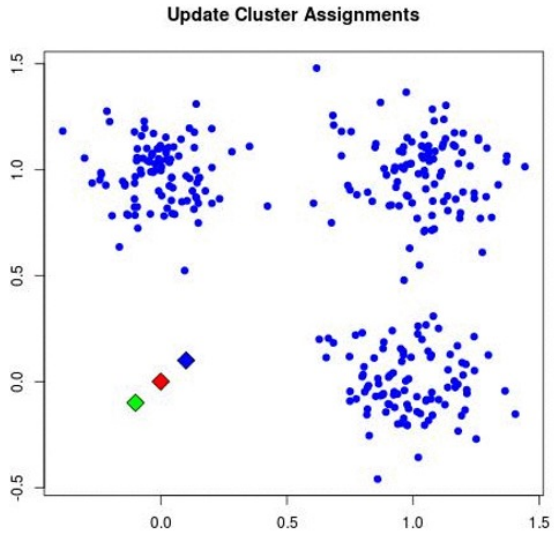
- K近邻法的优点如下：
  - ✓ 简单易实现；
  - ✓ 精度较高，对异常值不太敏感；
  - ✓ 适合于多分类的情况。
- K近邻法缺点是计算时间复杂度高，空间复杂度也高
  - ✓ Curse of dimensionality: 在高维空间中，数据稀疏导致距离度量失去作用。
- K-D树，用于以减少计算距离的次数
- 演示代码6.3，实现KNN算法手写数字识别
  - ✓ 预测错误数要低于第四章的逻辑斯蒂分类法
  - ✓ 使用K-D树比用brute花费了更多的时间
  - ✓ K-D树的使用条件是“样本数远大于特征树”

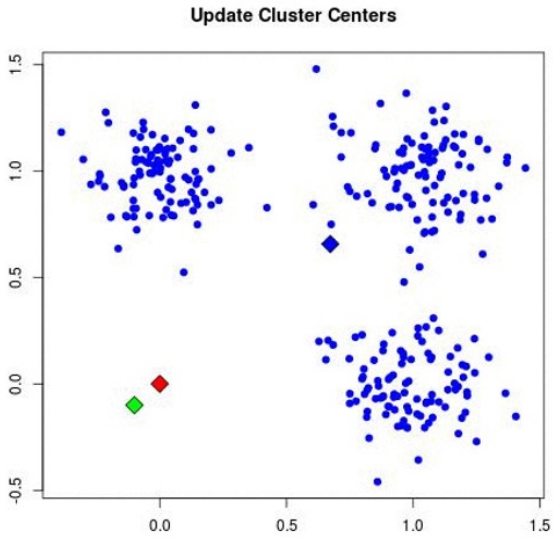
K-均值聚类

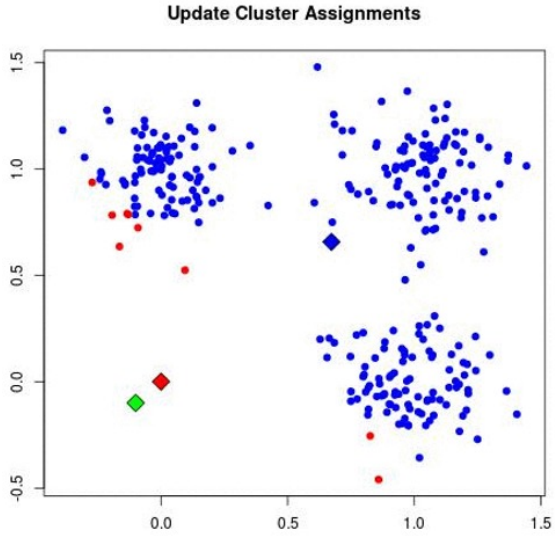
- 聚类：**将相似的对象组成相同的类，使得同类的对象之间距离尽量小，不同类对象之间的距离尽量大**
- K-均值聚类，也称K-means聚类，是“无监督”方法
- K均值聚类是基于划分样本集合的聚类算法
  - ✓ 没有人工标记的类别标签
  - ✓ 在计算过程中根据样本间的距离将样本集划分为K个子集
  - ✓ 每个子集代表一个类别，每个类别有一个“中心”，这个中心的“坐标”是所有属于该类别的样本的平均值；
  - ✓ 样本与类别的距离可以通过计算样本与类别中心的距离得到，每个样本所属的类别是与该样本最近的类别。
- **K均值聚类算法是一个动态迭代直至收敛的过程**
  - ✓ 类别“中心”的坐标是根据属于该类的样本坐标而获得，如果属于该类的样本发生变化则该类的中心也会相应变化
  - ✓ 样本所属类别又是根据其与类别中心距离而定，因此类别中心坐标变了后样本到类别中心的距离也会变化
  - ✓ 该动态过程最终会收敛达到一个稳定状态

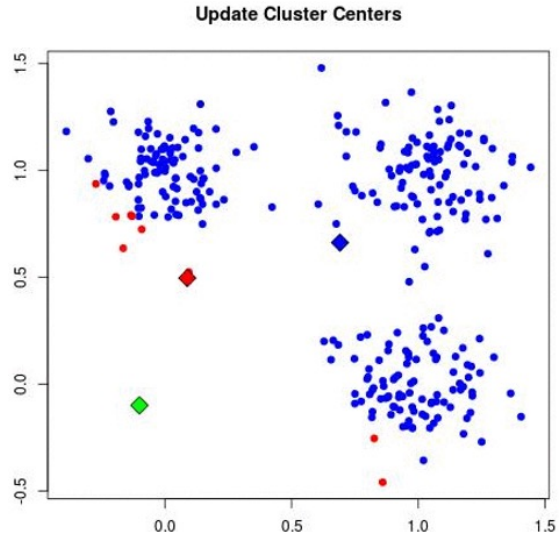
K-均值聚类例子

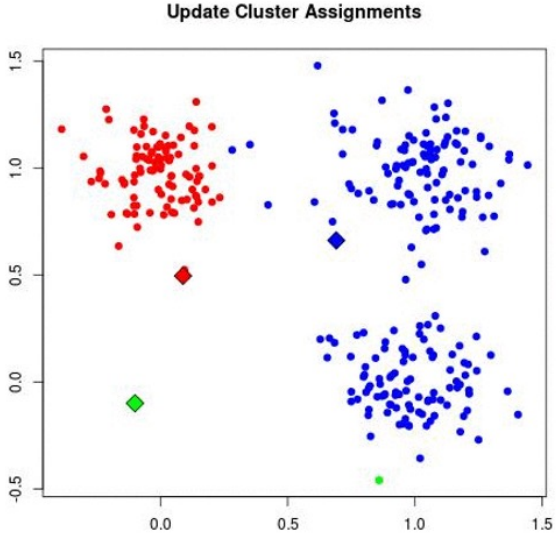

Start!


Update Cluster Assignments


Update Cluster Centers

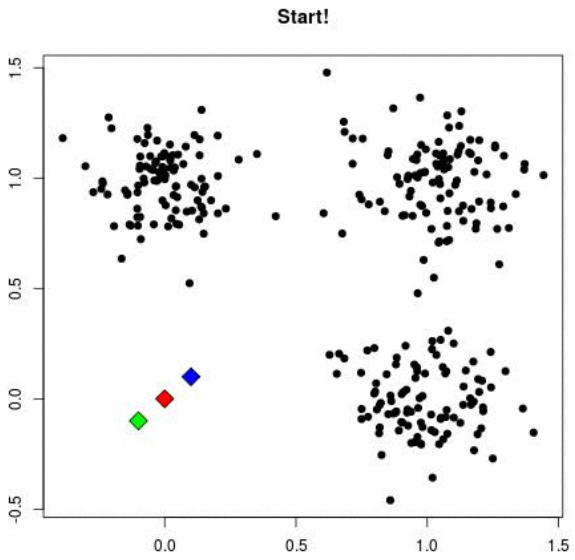

Update Cluster Assignments


Update Cluster Centers


Update Cluster Assignments

K-均值聚类例子

- 图演示了K-均值聚类的一般步骤，说明如下：
- 给定没有类别标记的数据集，每个圆点表示数据集中的一个样本；
- 设定K值为3，随机初始化类别的“中心”坐标，以菱形表示类别中心；
- 不断循环以下两步：
  - ✓ re_classify_examples：计算每个样本到这三个类别中心点的距离，将各个样本归类为与之最近的类别，不同颜色分别表示不同类别的样本点；
  - ✓ cal_centers：重新计算三个类别的中心点，即各类别中所有样本点的平均值；
- 达到稳定态后聚类结束。



$$V = \sum_{i=1}^{k} \sum_{x \in S_i} (x - \mu_i)^2$$

自定义编码实现KMeans

- 演示代码6.4，自定义实现K均值聚类算法
- 说明如下：
  - ✓ 定义了计算两个样本点欧氏距离的函数distance()；
  - ✓ 定义了一个为单个样本点选择最近类别的函数classify_one_example()；
  - ✓ 定义了一个为所有样本点重新分配类别的函数re_classify_examples()，该函数有三个参数X、example_bags、centers；
  - ✓ 定义了一个重新计算类别中心的函数cal_centers()，根据上一个函数重新选择类别后的样本，重新计算各个类别的中心，其参数分别是example_bags和centers；
  - ✓ 设置K值为2，加载数据并设置各类初始中心点坐标
  - ✓ K均值聚类的时间复杂度是

$$O(iter\_count \times m \times k \times d)$$

Scikit-learn实现KMeans

- Scikit-learn库包括两个K-均值算法
  - ✓ 传统的K-均值算法，对应的类是K-means
  - ✓ 基于采样的Mini Batch K-均值算法，MiniBatchKMeans（更适应于大数据）
- KMeans类构造方法说明
  - ✓ from sklearn.cluster import KMeans
  - ✓ model = KMeans(n_clusters, max_iter = 300, n_init = 10, init = "k-means++", algorithm = "auto")
    - ⇢ n_clusters：K值，一般需要多试一些值以获得较好的聚类效果
    - ⇢ max_iter：最大迭代次数，默认值是300
    - ⇢ n_init：用不同初始化聚簇“中心”运行算法的次数，默认值是10
    - ⇢ init： 即初始值选
- KMeans类主要属性方法说明
  - ✓ fit_predict(x)：用训练数据x拟合分类器模型并进行预测
  - ✓ cluster_centers_：质心坐标

K-Means类实现聚类算法

```
#代码6.5 使用Scikit-learn库K-Means类实现K均值聚类算法
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import Kmeans

X = np.loadtxt ( "cluster.txt", delimiter = "\t" )
model = KMeans ( n_clusters = 2 )
y_pred = model.fit_predict ( X )
plt.plot ( X [ y_pred == 0, 0 ], X [ y_pred == 0, 1 ], 'k^' ) #绘制分类为0类别的样本
plt.plot ( X [y_pred == 1, 0 ], X [ y_pred == 1, 1 ], 'kd' ) #绘制分类为1类别的样本
plt.show ( )
```

首先创建K-Means对象model，需要指定n_clusters参数，也就是K值；然后调用该对象的fit_predict()函数，对样本点X进行聚类，返回各个样本对应的类别，保存在y_pred变量中，该变量是一个NumPy向量；最后绘制聚类后的样本点。代码中，X[y_pred==0, 0]选择所有类别为0的样本点的第0个属性值，X[y_pred==1, 1] 选择所有类别为1的样本点的第1个属性值。

超参数选择，评价指标（课外）

- 超参数选择
  - ✓ 选择K
  - ✓ 选择初始点位置
  - ✓ 选择距离度量

- 评价指标
  - ✓ 没有真实标签：轮廓系数
  - ✓ 有标签：NMI、ARI

思考

- K均值聚类一定会收敛（达到稳定状态）吗？

总结

- K近邻分类的定义与实现
  - ✓ 自定义编码实现
  - ✓ Scikit-Learn库函数求解法

- K均值聚类的定义与实现
  - ✓ 自定义编码实现
  - ✓ Scikit-Learn库函数求解法

- 作业：Spoc第六章（DDL：12月9日00:00）