第六讲 聚类与神经网络

2025/04/19

文件：机器学习实践课3.pdf、 PJ-cluster.zip
作业：机器学习-聚类算法实验

1. 聚类 (Clustering)

1.1 聚类与分类

- 分类 (classification)： **监督**学习任务，利用已知的样本标记训练学习器预测未知样本的类别
- 聚类 (clusteing)： **无监督**学习任务，不知道真实的样本标记，只把相似度高的样本聚合在一起

1.2 聚类的定义

- 利用样本在多维空间中的相对位置，将样本分成两个或多个集合的算法
- 按照事物的某些属性，把数据分组成为多个类，在同一个类内对象之间具有**较高的相似度**，不同类之间的对象差别较大。使**类间的相似性尽量小，类内相似性尽量大**
- **实施过程**：

2. K-means 聚类

2.1 算法实现

1. 随机选择 k 个对象，每个对象初始地代表一个类的平均值或中心
2. 对剩余的每个对象，根据其到类中心的距离，被划分到最近的类
3. 重新计算每个类的平均值
4. 不断重复 ii ~ iii 过程，直到每个类的均值和聚类结果不再改变为止

2.2 K 值影响

- K 值过小：本应为两类的被归为一类
- K 值过大：本应为一类的被强行拆开成多类

2.3 初始值影响

- 随机初始值不同可能导致聚类结果不同，可能聚类效果不佳

2.4 性能分析

- **优点**：
  - 简单直观，易于理解与实现
  - 复杂度相对较低，在 K 不是很大的情况下，$\mathrm{K\text{-}means}$ 的计算时间相对较短
  - 会产生密度比较高的簇
- **缺点**：
  - K 需要预先得知，很难预测到准确值
  - 对初始设置很敏感
  - 对噪声和离异点非常敏感

3. 神经网络基础

3.1 神经元的两种状态

- **抑制**：一般情况下大多数神经元处于抑制状态
- **兴奋**：一旦某个神经元收到**刺激**，导致它的电位超过一个阈值，那么这个神经元就会被激活，处于 “兴奋” 状态，进而向其他的神经元传播化学物质（信息）

3.2 感知机 (Perceptron)

- **1958年**，*弗兰克·罗森布拉特* 创造了**感知机**
  - 感知机是一种模仿脑神经元结构的线性二分类模型，结构简单包括**权重**（突触）、**偏置**（阈值）及**激活函数**（细胞体）
  - 有着目前神经网络的基本思想，比如**梯度下降**、**损失函数**等
- **1969年**，人工智能之父*马文·明斯基* 在其著作中，证明了感知机本质上是一个**线性模型**，其连最基本的**异或问题**都无法解决

- **1986年**，神经网络之父*辛顿*提出适用于**多层感知机** (Multiple Layer Perceptron) 的 Back Propagation（BP）算法
  - 传播过程中引入了**非线性函数** sigmoid，解决了非线性问题
- **1991年**，人们发现随着梯度下降的反向传播，BP 神经网络出现了梯度会越来越小（梯度消失）的现象，导致神经网路的层数不能多
- **90年代中期**，SVM（支持向量机）出现，它具有完备的数学理论和较强的可解释性，神经网络研究热度下降
- **20世纪10年代**，随着肾毒性学习研究增多，神经网路迎来第三次蓬勃发展，依旧是现在深度神经网络阶段

4. BP（Back Propagation）神经网络

4.1 神经元的人工模型

- **神经元的人工模型**：
  - 神经元及其突触是神经网络的基本器件
  - **人工神经元（节点）** 模拟生物神经元
- **从三个方面进行模拟**：
  - 节点本身的信息处理能力 **（数学模型）**
  - 节点与节点之间连接 **（拓扑结构）**
  - 相互连接的强度：权重 **（通过学习来调整）**
- **对神经元的每一个输入都有一个加权系数 w_{ij}，称为权重值**，其**正负**模拟了生物神经元中突触的兴奋和抑制，其**大小**则代表了突触的不同连接强度
- 对全部输入信号进行整合，相应于生物神经元的膜电位
- 只有当**其输入总和超过阈值时，神经元才被激活而发放脉冲**，否则神经元不会产生输出信号
- **人工神经元的输出**也同生物神经元一样**仅有一个**，输出用某种非线性函数来表示

4.2 神经元的数学模型

- **神经元的激活函数**：
  - 神经元的不同数学模型的主要区别在于采用了不同的**激活函数**，从而使神经元具有不同的信息处理特性
  - 神经元的信息处理特性是决定人工智能神经网络整体性能的**三大要素之一**，反映了神经元输出预期激活状态之间的关系，最常用的激活函数有 4 种形式：
    a. **单极性阈值型激活函数**（单位阶跃函数，又称为硬限幅函数）：

$$f(x)=\begin{cases}1 & x\geq 0\\ 0 & x<0\end{cases}$$

    双极性阈值型激活函数（符号函数）：

$$f(x)=\begin{cases}1 & x>0\\ 0 & x=0\\ -1 & x<0\end{cases}$$

    b. **非线性激活函数**（函数本身及其倒数都是连续的，便于处理）
    Sigmoid:
    单极性：

$$f(x)=\frac{1}{1+e^{-x}}$$

    双极性：

$$f(x)=\frac{1-e^{-x}}{1+e^{-x}}$$

    c. **分段线性激活函数**（一定区间内满足线性关系，模拟了实际系统中的饱和特性）

$$f(x)=\begin{cases}0 & x\leq 0\\ cx & 0<x\leq x_c\\ 1 & x>x_c\end{cases}$$

    d. **概率型激活函数**（输入与输出间关系是不确定的，需用一个随机函数俩描述输出状态为 0 或 1 的概率）
    由于采用该激活函数的神经元输出状态分布与热力学中*玻尔兹曼 (Boltzmann) 分布* 相类似，因此这种神经元模型也成为**热力学模型**

4.3 BP 算法的推导

- 要求的参数与要满足一个目标，即**输出结果与标签误差最小**

- 问题转化成为**求关于这些权重的误差函数 E 的最小值**
- 根据之前学过的梯度下降法来求解问题，需要求出 E 关于权重的导数
- BP 算法可以利用梯度下降法求解神经网络参数，总共分为两个步骤：
  - **前向传播**：计算误差，前向传播公式记录误差和权重的关系
    - 假设选用均方误差：

$$E_{\text{total}} = \sum \frac{1}{2}(\text{target} - \text{output})^2$$

    - 先将 x_1 ，x_2 与偏置加权求和得到 h_1 的输入，经过激活函数后再输出
    - 同理可以计算 h_2 的输出
    - 再将 h_1 节点与 h_2 节点的输出与偏执加权求和，得到 out 节点输入，在经过激活函数得到输出
    - 使用输出值与标签可计算误差
  - **反向传播**：计算出误差关于权重的导数，并进行更新
    - 由复合函数求导法则，可以知道：

$$\frac{\Delta E_{\text{total}}}{\Delta w_5} = \frac{\Delta E_{\text{total}}}{\Delta \text{out}_{o1}} \times \frac{\Delta \text{out}_{o1}}{\Delta \text{net}_{o1}} \times \frac{\Delta \text{net}_{o1}}{\Delta w_5}$$

    - 第一步前向传播得到的攻势可以代入进上式，计算处误差关于权重 w_5 的导数
    - 由梯度下降法，为了求误差最小值，更新 w_5 的方式：

$$w_5^+ = w_5 = \alpha \times \frac{\Delta E_{\text{total}}}{\Delta w_5}$$

      其中，α 为学习率，控制更新的步长
  - 通过梯度下降法不断更新参数，知道误差满足我们需要的某些指标就可以停止了，此时得到的权重就是符合我们要求的权重

4.4 应用

- 函数逼近：用输入向量和相应的输出向量训练一个网络逼近一个函数
- 回归：用一个待定的输出向量将它与输入向量联系起来
- 分类：吧输入向量所定义的何时方始进行分类
- 数据压缩：减少输出向量维数以便于传输或存储

作业:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter('ignore')
from sklearn.preprocessing import StandardScaler # normalization
from sklearn.cluster import KMeans # k-means
from sklearn import metrics # rand metrics

# Load data
df = pd.read_csv(r"./wine-clustering.csv")
y = [0] * 59 + [1] * 71 + [2] * 48

# visualize data
print(df.head()) # first several rows of the data
print(df.sample(5)) # sample 5 rows of the data
continous = [i for i in df.columns if df[i].dtypes != 'int64']
# plt.figure(figsize = (10,5))
sns.pairplot(vars = continous,data = df)
plt.figure(figsize = (10,5))
sns.heatmap(df.corr(),annot = True)

# data preprocessing
print(df.duplicated().sum()) # check duplicate rows
print(df.isnull().sum()) # check null value

# 尝试数据归一化对聚类结果的影响
# 未归一化数据
X_unnormalized = df

# 归一化数据
sc = StandardScaler()
X_normalized = pd.DataFrame(sc.fit_transform(df),columns = df.columns)

# hypertuning
n_start, n_end = 1 , 11
wcss_normalized = []
wcss_unnormalized = []

for i in range(n_start, n_end):
        kmeans_normalized = KMeans(n_clusters = i, init = "k-means++", random_state = 0)
        kmeans_normalized.fit(X_normalized)
        wcss_normalized.append(kmeans_normalized.inertia_)

        kmeans_unnormalized = KMeans(n_clusters = i, init = "k-means++", random_state = 0)
        kmeans_unnormalized.fit(X_unnormalized)
        wcss_unnormalized.append(kmeans_unnormalized.inertia_)

# Elbow method for normalized data
plt.figure(figsize = (10,5))
plt.plot(range(n_start, n_end),wcss_normalized,marker = '*', label='Normalized')
plt.xticks(range(n_start, n_end))
plt.title('Elbow Method for Normalized Data')
plt.show()

# Elbow method for unnormalized data
plt.figure(figsize = (10,5))
plt.plot(range(n_start, n_end),wcss_unnormalized,marker = '*', label='Unnormalized')
plt.xticks(range(n_start, n_end))
plt.title('Elbow Method for Unnormalized Data')
plt.show()

# 尝试不同的聚类类别数
n_clusters_list = [2, 3, 4, 5]
index_list=["Alcohol","Malic_Acid","Ash","Ash_Alcanity","Magnesium","Total_Phenols","Flavanoids","Nonflavanoid_Phenols","Proanthocyanins","Co
```

```
ari_normalized_list = []
ari_unnormalized_list = []

for n_clusters in n_clusters_list:
        # 归一化数据建模
        kmeans_normalized = KMeans(n_clusters = n_clusters,init = "k-means++", random_state = 0)
        X_normalized['category'] = kmeans_normalized.fit_predict(X_normalized)
        ari_normalized = metrics.cluster.adjusted_rand_score(y, X_normalized['category'])
        ari_normalized_list.append(ari_normalized)

        # 未归一化数据建模
        kmeans_unnormalized = KMeans(n_clusters = n_clusters,init = "k-means++", random_state = 0)
        X_unnormalized['category'] = kmeans_unnormalized.fit_predict(X_unnormalized)
        ari_unnormalized = metrics.cluster.adjusted_rand_score(y, X_unnormalized['category'])
        ari_unnormalized_list.append(ari_unnormalized)

        # 对不同的维度进行聚类结果可视化（以Alcohol和Proanthocyanins为例）
        plt.figure(figsize = (10,5))
        plt.subplot(1, 2, 1)
        sns.scatterplot(x = 'Alcohol',y = 'Proline',hue = 'category',data = X_normalized,palette=['red','green','blue','yellow'][:n_clusters]
        plt.title(f'Normalized, n_clusters={n_clusters}')

        plt.subplot(1, 2, 2)
        sns.scatterplot(x = 'Alcohol',y = 'Proline',hue = 'category',data = X_unnormalized,palette=['red','green','blue','yellow'][:n_cluster
        plt.title(f'Unnormalized, n_clusters={n_clusters}')
        plt.show()
plt.figure(figsize = (100,100))
for xxx in range(0,13):
        for yyy in range(0,13):
                # 归一化数据建模
                if xxx==yyy:
                        continue
                xx=index_list[xxx]
                yy=index_list[yyy]
                kmeans_normalized = KMeans(n_clusters = 3, init = "k-means++", random_state = 0)
                X_normalized['category'] = kmeans_normalized.fit_predict(X_normalized)
                ari_normalized = metrics.cluster.adjusted_rand_score(y, X_normalized['category'])
                ari_normalized_list.append(ari_normalized)

                # 未归一化数据建模
                kmeans_unnormalized = KMeans(n_clusters = 3, init = "k-means++", random_state = 0)
                X_unnormalized['category'] = kmeans_unnormalized.fit_predict(X_unnormalized)
                ari_unnormalized = metrics.cluster.adjusted_rand_score(y, X_unnormalized['category'])
                ari_unnormalized_list.append(ari_unnormalized)

                # 对不同的维度进行聚类结果可视化（以Alcohol和Proanthocyanins为例）

                plt.subplot(13, 13, 13*xxx+yyy+1)
                sns.scatterplot(x=xx,y=yy,hue = 'category',data = X_normalized,palette=['red','green','blue'])
                plt.title(f'Normalized')

# plt.show()
plt.savefig("D:\\SJTU_AI\\讲义\\06第六讲\\Figure_9.png")

# # 打印不同聚类类别数下的Adjusted Rand Index
# for i, n_clusters in enumerate(n_clusters_list):
#       print(f'Normalized, n_clusters={n_clusters}, Adjusted Rand Index: {ari_normalized_list[i]}')
#       print(f'Unnormalized, n_clusters={n_clusters}, Adjusted Rand Index: {ari_unnormalized_list[i]}')

# # 根据KMeans 的主要参数对模型进行调整，并分析聚类结果
# # 以n_clusters=3为例，调整init参数为random
# kmeans_random = KMeans(n_clusters = 3, init = 'random', random_state = 0)
# X_normalized['category_random'] = kmeans_random.fit_predict(X_normalized)
# ari_random = metrics.cluster.adjusted_rand_score(y, X_normalized['category_random'])
# print('Adjusted Rand Index with init=random:', ari_random)

# # 可视化调整参数后的聚类结果
# plt.figure(figsize = (10,5))
# sns.scatterplot(x = 'Alcohol',y = 'proline',hue = 'category_random',data = X_normalized,palette=['red','green','blue'])
```

```
# plt.title('Clustering Result with init=random')
# plt.show()
```