Readme

Day01

Date:2021-07-20
Made By: 纸 石头 紫阳花

K近邻算法原理

给定一个已知标签的训练集，输入没有标签的新数据后，在训练数据集中找到与新数据最近的k个实例，如果这k个实例多数属于某个类别，则将新数据标注该类别的标签。

在判断距离时如果有多个特征属于多维空间的距离判断时，应用于公式欧几里得度量：

n维空间的公式

$$d(x,y) := \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2} = \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}.$$

k近邻算法的标准步骤

1. 计算已知数据集中的点与新数据的距离
2. 按照距离进行递增排序
3. 选取当前距离最小的k个点
4. 确定前k个点所在类别出现的频率
5. 返回前k个点中出现频率最高的类别作为当前预测类别

算法Python实现

构建数据集

```
# 构建原始数据集
# 先用字典形式然后再转成DataFrame
import pandas as pd

rowdata = {"电影名称": ["无问西东","后来的我们","前任3","红海行动","唐人街探案","战狼2"],
           "打斗镜头": [1, 5, 12, 108, 112, 115],
           "接吻镜头": [101, 89, 87, 5, 9, 8],
           "电影类型": ["爱情片", "爱情片", "爱情片", "动作片", "动作片", "动作片"]
           }
# pandas 的DataFrame方法可以利用字典进行初始化为DataFrame格式
movie_data = pd.DataFrame(rowdata)
print(movie_data)
```

结果显示：

```
    电影名称  打斗镜头  接吻镜头 电影类型
0   无问西东     1   101  爱情片
1  后来的我们     5    89  爱情片
2    前任3    12    87  爱情片
3   红海行动   108     5  动作片
4  唐人街探案   112     9  动作片
5    战狼2   115     8  动作片

进程已结束，退出代码为 0
```

计算距离

```
# 计算新数据点与训练集数据的距离
new_data = [24, 67]
dist = list((((movie_data.iloc[:6, 1:3] - new_data)**2).sum(1))**0.5)
```

DataFrame的illoc方法对DataFrame数据进行切片（参数：[行， 列]）直接将切片后数据减去new_data相当于将每行的两列数据减去new_data。

.sum()方法用于将两列数据进行求和。

- 参数为1：列于列之和

```
0      1685
1       845
2       544
3     10900
4     11108
5     11762
dtype: int64
```

- 参数为0：行与行之和

DataFrame的数据列表化后会消去列索引作为列表索引，只存储数据大小

```
[41.048751503547585, 29.068883707497267, 23.323807579381203, 104.4030650891055, 105.39449701004318, 108.45275469069469]

进程已结束，退出代码为 0
```

根据距离进行排序

```
# 对距离进行升序排序
# 设k=4
k = 4
dist_1 = pd.DataFrame({"dist":dist, "label":(movie_data.iloc[:6, 3])})
dr = dist_1.sort_values(by = "dist")[:k]
print(dr)
```

由得出的新数据距离列表与训练集通过字典构成新的DataFrame数据集

利用DataFrame的sort_values(by =)[:]方法通过by参数选取排序依据，并进行对得出的DataFrame数据进行切片处理。

结果显示：

```
C:\Users\86188\AppData\Local\Programs\Python\Python39
          dist label
2    23.323808   爱情片
1    29.068884   爱情片
0    41.048752   爱情片
3   104.403065   动作片

进程已结束，退出代码为 0
```

确定标签出现频率

```
re = dr.loc[:, 'label'].value_counts()
result = []
result.append(re.index[0])
print(result)
```

```
爱情片     3
动作片     1
Name: label, dtype: int64

进程已结束，退出代码为 0
```

DataFrame的loc方法通过参数选取特定标签（即DataFrame的列对象）进行切片，其他列都舍去

DaraFrame的value_counts方法计算列对象对应的数量相当于mysql中的group by

由于re的主键（索引）时第一列，故用index对出现频率最高的标签进行选取

re.index[0]取出现频率最高的标签作为新数据标签

将过程封装为函数

```
"""
函数功能：knn分类器
参数说明：
    inX为新数据坐标
    dataset为旧数据集
    k为超参数
返回：
    result为结果集
"""

def classify01(inX, dataset, k):
    dist = list((((dataset.iloc[:6, 1:3] - inX) ** 2).sum(1)) ** 0.5)
    dist_1 = pd.DataFrame({"dist": dist, "label": (dataset.iloc[:6,
3])})
    dr = dist_1.sort_values(by="dist")[:k]
```

```
    re = dr.loc[:, 'label'].value_counts()
    result = []
    result.append(re.index[0])
    print(result)

    return result

classify01([31, 24], movie_data, 4)
```

knn算法不具有显示的学习过程，没有数据的训练，直接使用位置的数据与已知的数据进行比较，得到结果。

KNN的sklearn实现

```
# 分割数据集
X_train, X_test, y_train,  y_test = train_test_split(X, y, random_state = 0)
# 建立模型
KNNmodel = neighbors.KNeighborsClassifier(n_neighbors=6,      # n_neighbors指定超参数K值
                                          weights='uniform', # uniform指定所有邻居的权重相等， distance指定权重与距离成反比
                                          algorithm='auto',  # 限定半径最近邻法使用的算法
    # 'ball_tree': BallTree算法
    # 'kd_tree': kd树算法
    # 'brute': 暴力搜索算法
    # 'auto': 自动决定适合的算法
                                          leaf_size=30, # 指定ball_tree或kd_tree的叶节点规模。他影响树的构建和查询速度
                                          p=2,  # p=1:曼哈顿距离; p=2:欧式距离
                                          metric='minkowski', # 指定距离度量，默认为'minkowski'距离
                                          metric_params=None,
                                          n_jobs=1           # 并行的CPU数
                                         )


# 训练模型
KNNmodel.fit(X_train, y_train)
# 模型预测
y_predict = KNNmodel.predict(X_test)
# 泛化精度
KNNmodel.score(X_test, y_test)# 可得模型准确度
```

```
# 训练集精度
KNNmodel.score(trainX, trainY)
```

KNN算法实例

约会网站配对效果判定

导入数据集

```
# 约会网站判定
import pandas as pd
import matplotlib.pyplot as plt

# 导入数据集
datingTest = pd.read_table("datingTestSet.txt", header=None)
# print(datingTest.head())
# print(f"导入数据规格{datingTest.shape}")
# print(f"导入数据信息{datingTest.info}")
```

pandas的文件读取方式：

读取csv文件

```
pd.read_csv(Filepath, header = None)     #header避免第一列作为表头（自动添加列索引），否则默认第一列为表头
#  index_col=[“列名”]可将某一列或几列作为行索引（相当于数据库中的主键）（前提是表头为列名）
#  usecols参数可以单独取出某几列
#  parse_dates将某一列的时间字符标准化
#  nrows确定读入的行数
```

读取txt文件

```
data = pd.read_table(Filepath)           #读取txt形式的表格
# 若要用read_csv来读取txt文件数据则需要进行分隔符设置，如
data = pd.read_csv(Filepath, sep="\t")
```

读excel文件

```
data = pd.read_excel(Filepath)
```

pandas文件写入

```
data.to_csv(Filepath, index=False)       # Flase表示去除索引、
# 写txt文件也使用to_csv但要多加sep关键字来设置分割方法
```

分析数据并绘图

```
# 分析数据
# 把不同的数据用不同的颜色进行区分
Colors = []
for i in range(datingTest.shape[0]):
    m = datingTest.iloc[i, -1]      # 通过切片取标签
    # print(f"m={m}")
    if m == "didntLike":
        Colors.append("black")
    elif m == "smallDoses":
        Colors.append("orange")
    else:
        Colors.append("red")

# 绘制两两特征上的散点图
plt.rcParams["font.sans-serif"] = ['Simhei']    # 将字体设置为黑体

# 设置画布
pl = plt.figure(figsize=(12, 8))

# 添加子画布
fig1 = pl.add_subplot(221)  # 221表示将画布分为两行两列，且当前子画布为第一个
plt.scatter(datingTest.iloc[:, 1], datingTest.iloc[:, 2], marker=".", c
= Colors)
plt.xlabel("玩游戏所占时间比")
plt.ylabel("每周消费冰激凌公升数")

fig2 = pl.add_subplot(222)
plt.scatter(datingTest.iloc[:, 0], datingTest.iloc[:, 1], marker=".", c
= Colors)
plt.xlabel("每年飞行里程")
plt.ylabel("玩游戏所占时间比")

fig3 = pl.add_subplot(223)
plt.scatter(datingTest.iloc[:, 0], datingTest.iloc[:, 2], marker=".", c
= Colors)
plt.xlabel("每年飞行里程")
plt.ylabel("每周消费冰激凌公升数")

plt.show()
```

数据归一化处理，统一量纲

```
# 数据归一化
# 使用欧几里得度量
```

```
# 由于发现某一项特征的重要性不够，可以使用数据归一化处理
# 此处使用0-1标准化
def minmax(dataset):
    minDf = dataset.min()
    maxDf = dataset.max()
    normSet = (dataset - minDf)/(maxDf-minDf)
    return normSet

# 重新向DataFrame中加上标签
datingT = pd.concat([minmax(datingTest.iloc[:, : 3]),
datingTest.iloc[:,3]], axis=1)
# print(datingT.head(10))
```

切分数据集和测试机，并定义分类器

```
"""
函数功能：切分训练集和测试集
参数说明：
    dataset：原始数据集
    rate：训练集所占比例
返回：
    切分好的训练集和数据集
"""
def randSplit(dataset, rate=0.9):
    n = dataset.shape[0]          # 取数据集行数
    m = int(n*rate)      # 必须保证随机性
    train = dataset.iloc[:m, :]
    test = dataset.iloc[m:, :]
    test.index = range(test.shape[0])
    train.index = range(train.shape[0])
    return train, test

train, test = randSplit(datingT)
# print(test)

# 定义分类器
def datingClass(train, test, k):
    n = train.shape[1] - 1   # 除标签外的列数
    m = test.shape[0]
    result = []
    for i in range(m):
        # 欧几里得度量
        dist = list((((train.iloc[:, :n] - test.iloc[i, :n]) **
2).sum(1)) ** 0.5)
        # 将测试集当前数据与训练集数据的距离与训练集标签相连接为DataFrame格式
        dist_1 = pd.DataFrame({"dist":dist,
                               "labels":(train.iloc[:, n])})
```

```
        # 取最近的k个数据
        dr = dist_1.sort_values(by="dist")[ : k]
        re = dr.loc[:, "labels"].value_counts()
        # loc[:, "labels"]列只保留labels列，行全部取，再通过value_counts确定不同行的数量
        # re.index[0]即为出现次数最多的标签(由于只有标签名和数量两列，标签名为索引，可通过index取)
        result.append(re.index[0])
        # 通过pd.Series函数将预测结果转化成可加入test数据集的格式，并设置列名为predict
    result = pd.Series(result)
    # Series为一维标签数组能保存任何类型的值
    test.loc[:, "predict"] = result
    print(test)
    acc = (test.iloc[:, -1] == test.iloc[:, -2]).mean()
    print(f"模型预测的准确度为{acc}")

    return test

datingClass(train, test, 5)
```

KNN算法总结

算法功能

分类、回归

算法类型

有监督学习-惰性学习、距离类模型

数据输入

包含数据标签以及至少k个训练样本

特征空间中各个特征的量纲应统一，否则应进行归一化处理

自定义超参数k

数据输出

分类：输出预测的类别

回归：试过一次关于对象的属性值（最近的k个训练样本的平均值）

数据归一化处理

什么是数据归一化

将所有的数据映射到同一尺度。

特征数值化以后由于取值大小的不同造成了特征空间中样本点的距离会被个别特征值所主导，而受其他特征影响较小。如特征一的取值在1到200之间特征二的取值在1到20之间，则特征一在不做处理的情况下会主导特征的权重。

故应当做归一化处理，将不同特征按照某种方式映射到统一的尺度

数据归一化处理方法

数据归一化的方法 数据归一化的方法主要有两种：最值归一化（也称0-1归一化）和均值方差归一化。

最值归一化

可以将所有数据都映射到0-1之间，它适用于数据分布有明显边界的情况，容易受到异常值或极端数据影响，异常值会造成数据的整体偏斜。

公式：$$x_{scale} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

均值方差归一化

将所有数据归一化到均值为0方差为1的分布中，适用于数据没有边界且可能存在极端数据的情况。

公式：$$Xscale = \frac{Xi - Xmean}{Xstd}$$