数据处理

第二章 Python机器学习基础 2.1pdf.pdf
2.56MB

第二章 Python机器学习基础 2.2.pdf
1.99MB

第二章 python机器学习基础2.3.pdf
3.32MB

NumPy 函数

数组创建函数

numpy.array(序列对象, dtype=类型)：将序列对象转换为 NumPy 数组。

numpy.arange(start, stop, step)：创建指定范围内的等距数组。

numpy.linspace(start, stop, num=50)：创建指定范围内等距的浮点数数组。

numpy.ones(形状, dtype=None)：创建指定形状的全1数组。

numpy.zeros(形状, dtype=float)：创建指定形状的全0数组。

numpy.zeros_like(a, dtype=None)：创建与数组 a 形状和类型相同的全0数组。

numpy.random.rand(d0, d1, ..., dn)：创建指定形状的随机浮点数数组。

numpy.random.randint(low, high=None, size=None)：创建指定形状的随机整数数组。

numpy.random.randn(d0, d1, ..., dn)：创建指定形状的标准正态分布随机浮点数数组。

数组属性函数

<数组名>.size：查看数组元素的总个数。

<数组名>.shape：查看数组的形状，表示数组维度大小的元组。

<数组名>.ndim：查看数组有几个维度。

<数组名>.dtype：查看数组元素的类型。

type(<数组名>)：查看数组类型。

索引函数

🎨 **数组索引操作：**

一维数组：使用方括号 [] 和索引 i 获取元素，例如 a[i]。

多维数组：使用多个索引，例如 a[i, j, k] 获取元素。

支持负数索引和切片操作。

🏖️ **数组切片操作：**

使用冒号 : 表示切片，例如 a[start:stop:step]。

可以对多维数组进行切片，例如 a[start1:stop1:step1, start2:stop2:step2, ...]。

切片操作不会改变原数组的维数。

✍️ **索引与切片混用：**

将数组索引操作和切片操作混合使用，例如 a[i, :] 获取第 i 行的所有元素。

结果是一个比原数组维度低的子数组。

🎉 **花式索引：**

使用索引列表 [i1, i2, ...] 来访问元素，例如 a[[i1, i2, ...], j]。

可以灵活地选择数组中的行或列。

🎉 **布尔索引：**

使用布尔数组作为掩码，选择满足条件的元素，例如 a[a < 3]。

可以结合比较运算符和布尔运算符进行更复杂的条件选择。

数组运算函数

👍 **算术运算：**

+、-、*、/、//、%、** 等运算符可以进行元素级的运算。要求参与运算的数组形状相同，或者可以进行广播操作。

🌟 **广播：**

当数组形状不同时，可以进行广播操作，自动扩展数组形状，使其可以进行元素级的运算。例如 a + b，其中 a 是二维数组，b 是一维数组，b 会被扩展成与 a 相同形状的二维数组。

🏖️ 线性代数运算：

a.T：矩阵转置，行变列，列变行。

a.dot(b) 或 np.dot(a, b)：矩阵内积或点积。

np.linalg.inv(a)：计算矩阵 a 的逆矩阵。

聚合函数

np.sum(a, axis=None)：对数组 a 的所有元素或指定轴上的元素求和。

np.mean(a, axis=None)：对数组 a 的所有元素或指定轴上的元素求平均数。

np.std(a, axis=None)：对数组 a 的所有元素或指定轴上的元素求标准差。

np.var(a, axis=None)：对数组 a 的所有元素或指定轴上的元素求方差。

np.min(a, axis=None)：对数组 a 的所有元素或指定轴上的元素求最小值。

np.max(a, axis=None)：对数组 a 的所有元素或指定轴上的元素求最大值。

np.argmin(a, axis=None)：返回数组 a 中最小元素的索引。

np.argmax(a, axis=None)：返回数组 a 中最大元素的索引。

np.cumsum(a, axis=None)：对数组 a 的指定轴上的元素求累计和。

np.cumprod(a, axis=None)：对数组 a 的指定轴上的元素求累计积。

数组变形函数

x.reshape(t)：将数组 x 变形为参数 t 所指定的形状。

x.flatten()：将高维数组展平为一维向量。

x.ravel()：将高维数组展平为一维向量。

数组拼接函数

numpy.concatenate([数组a, 数组b, ...], axis=0)：按指定轴方向合并多个数组。

数组排序函数

numpy.sort(a, axis=-1)：返回排序后的新数组，原数组 a 不变。

a.sort(axis=-1)：用排好序的数组替代原数组 a。

numpy.argsort(a, axis=-1)：返回原始数组排好序的索引值数组。

Pandas 函数

DataFrame 创建函数

pandas.DataFrame(data, index=index, columns=columns)：创建 DataFrame。

DataFrame 属性函数

df.shape：查看 DataFrame 的形状。

df.dtypes：查看 DataFrame 中各列的数据类型。

DataFrame 数据访问函数

df.iloc[行索引, 列索引]：通过行索引和列索引访问 DataFrame 中的数据。

df.loc[行标签, 列标签]：通过行标签和列标签访问 DataFrame 中的数据。

DataFrame 数据操作函数

df.append(other, ignore_index=False)：将另一个 DataFrame 追加到当前 DataFrame。

df.drop(labels, axis=0, inplace=False)：删除指定行或列。

df.drop_duplicates(subset=None, keep='first', inplace=False)：删除重复行。

df.fillna(value)：用指定值填充 NaN 值。

df.replace(old, new)：替换 DataFrame 中的值。

df.sort_values(by, ascending=True)：按指定列排序。

df.describe()：生成描述性统计信息。

可视化函数

matplotlib.pyplot.scatter(x, y, c=color)：绘制散点图。

matplotlib.pyplot.plot(x, y)：绘制折线图。

matplotlib.pyplot.title(title)：设置图表标题。

matplotlib.pyplot.xlabel(label)：设置 x 轴标签。

matplotlib.pyplot.ylabel(label)：设置 y 轴标签。

matplotlib.pyplot.show()：显示图表。

数据预处理

模块一：数据清洗

数据清洗是提高数据质量的关键步骤，旨在清除与任务无关、格式非法或不在指定范围内的“脏数据”。主要方法包括：

- **重复值处理**: 使用 `duplicated()` 检查重复行， `drop_duplicates()` 移除重复行。

```
import pandas as pd

# 创建包含重复值的 DataFrame
data = pd.DataFrame({
    'name': ['Alice', 'Bob', 'Alice', 'Charlie', 'Bob'],
    'age': [25, 30, 25, 35, 30]
})

```

```
# 检查重复行
duplicates = data.duplicated()

# 移除重复行
data_cleaned = data.drop_duplicates()

print("重复值: ", duplicates)
print("去重后的数据: ")
print(data_cleaned)

```

- **缺失值处理**: 使用 `isnull()` 和 `notnull()` 检查缺失值，`fillna()` 填充缺失值，`dropna()` 删除缺失值。

```
import pandas as pd
import numpy as np

# 创建包含缺失值的 DataFrame
data = pd.DataFrame({
    'name': ['Alice', 'Bob', np.nan, 'Charlie', 'Dave'],
    'age': [25, 30, 35, np.nan, 40]
})

# 检查缺失值
missing_values = data.isnull()

# 填充缺失值
data_filled = data.fillna(value=0)  # 用 0 填充

# 删除缺失值
data_dropped = data.dropna()

print("缺失值: ", missing_values)
print("填充缺失值后的数据: ")
print(data_filled)
print("删除缺失值后的数据: ")
print(data_dropped)
```

- **异常值处理**: 使用 3σ 原则、箱线图法、Z-score 法等方法检测异常值，并进行替换或删除。

```
import pandas as pd

# 创建包含异常值的 DataFrame
```

```
data = pd.DataFrame({
    'score': [80, 90, 95, 110, 85, 70]
})

# 3σ 原则检测异常值
mean = data['score'].mean()
std = data['score'].std()
threshold = 3 * std

outliers = data[(data['score'] < mean - threshold) | (data['score'] > mean + threshold)]

# 处理异常值（示例：用均值替换）
data_cleaned = data.replace(to_replace=outliers, value=mean)

print("异常值：")
print(outliers)
print("处理异常值后的数据：")
print(data_cleaned)
```

- **统一数据格式**: 将数据转换为统一的格式，例如将字符串转换为数值类型。

```
data['name'] = data['name'].astype('category')
```

```
data['age'] = data['age'].astype('float')
```

模块二：数据合并

Pandas 提供了多种合并数据集的方法，包括：

- **concat()**: 沿着一个轴将多个对象进行堆叠。

```
import pandas as pd

# 创建两个 DataFrame
df1 = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
df2 = pd.DataFrame({'A': [7, 8, 9], 'B': [10, 11, 12]})

# 沿着 0 轴（默认）堆叠数据
```

```
df_concat = pd.concat([df1, df2], ignore_index=True)

print(df_concat)
```

输出：

```
   A   B
1  4
2  5
3  6
7  10
8  11
9  12
```

- **merge()**: 根据主键合并数据。

```
import pandas as pd
# 创建两个 DataFrame
df1 = pd.DataFrame({'key': ['A', 'B', 'C'], 'value': [1, 2, 3]})
df2 = pd.DataFrame({'key': ['B', 'D', 'D'], 'value': [4, 5, 6]})

# 根据 'key' 列合并数据
df_merged = pd.merge(df1, df2, on='key', how='inner')  # 内连接

print(df_merged)
```

- **join() - 通过行索引合并数据。**

```
import pandas as pd
# 创建两个 DataFrame
df1 = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]}, index=['a', 'b', 'c'])
df2 = pd.DataFrame({'C': [7, 8, 9]}, index=['b', 'c', 'd'])

# 根据 index 列合并数据
df_joined = df1.join(df2)

print(df_joined)
```

输出：

```
   A  B  C
a  1  4 NaN
b  2  5  7
c  3  6  8
d NaN NaN  9
```

- `concat()` 用于将多个 DataFrame 堆叠在一起，沿着指定的轴 (0 或 1) 进行合并。
- `merge()` 用于根据一个或多个键将两个 DataFrame 合并在一起，可以指定连接方式 (内连接、外连接等)。
- `join()` 用于根据索引将两个 DataFrame 合并在一起，类似于 SQL 中的 JOIN 操作。

模块三：数据转换

数据转换是将数据转换为适合模型训练的格式，主要方法包括：

- **分类变量**: 使用 one-hot 编码将类别特征转换为哑变量。

分类变量是指其取值为离散类别，例如性别、季节、颜色等。直接使用分类变量进行模型训练通常效果不佳，因为模型无法理解类别之间的关系。One-Hot 编码将每个类别转换为一个新的二进制特征，值为 0 或 1，从而将分类变量转换为数值特征，方便模型训练。（二进制组合）

```python
import pandas as pd

# 创建包含分类变量的 DataFrame
df = pd.DataFrame({'color': ['red', 'green', 'blue', 'red'],'shape':
['circle', 'square', 'triangle', 'circle']
})

#使用 get_dummies() 进行 One-Hot 编码
df_encoded = pd.get_dummies(df)

print(df_encoded)
```

输出:

```
   color_red  color_green  color_blue  shape_circle  shape_square
shape_triangle
0          1            0           0             1             0
   0
1          0            1           0             0             1
   0
```

```
4  2           0           0           1           0           0
   1
5  3           1           0           0           1           0
   0
```

- **非数值特征**: 使用 LabelEncoder 将非数值特征转换为数值。

LabelEncoder 将每个类别的唯一标识符转换为数值，例如将 “red” 转换为 0，“green” 转换为 1，“blue” 转换为 2。这种方法简单易懂，但无法保留类别之间的顺序关系。

```python
from sklearn.preprocessing import LabelEncoder

# 创建包含非数值特征的 DataFrame
df = pd.DataFrame({'color': ['red', 'green', 'blue', 'red'],'shape':
['circle', 'square', 'triangle', 'circle']
})

# 创建 LabelEncoder 对象
label_encoder = LabelEncoder()

# 对每个类别特征进行编码
df['color_encoded'] = label_encoder.fit_transform(df['color'])
df['shape_encoded'] = label_encoder.fit_transform(df['shape'])

print(df)
```

输出:

```
   color   shape  color_encoded  shape_encoded
0  red     circle             0              0
1  green   square             1              1
2  blue    triangle           2              2
3  red     circle             0              0
```

- **离散化连续数据**: 使用 cut() 将连续数据划分为离散区间。

cut() 将连续数据划分为离散区间，每个区间对应一个类别。例如，将年龄划分为 [0, 18), [18, 35), [35, 60), [60, 100) 四个区间，分别对应 “child”，“young adult”，“adult”，“senior” 四个类别。

```python
import pandas as pd

# 创建包含连续数据的 DataFrame
df = pd.DataFrame({'age': [5, 20, 30, 40, 70]})

# 定义区间
bins = [0, 18, 35, 60, 100]

# 定义标签
labels = ['child', 'young adult', 'adult', 'senior']

# 使用 cut() 进行离散化
df['age_category'] = pd.cut(df['age'], bins=bins, labels=labels, right=False)

print(df)
```

输出:

```
   age  age_category
0    5         child
1   20   young adult
2   30         adult
3   40         adult
4   70        senior
```

总结:

- One-Hot 编码适用于类别变量，将每个类别转换为一个新的二进制特征。
- LabelEncoder 适用于将类别变量的唯一标识符转换为数值。
- cut() 适用于将连续数据划分为离散区间，每个区间对应一个类别。

- **Min-Max 归一化**
- **原理**： 将数据特征的取值范围缩放到 [0, 1] 区间。**公式为：**

```
X_scaled = (X - X_min) / (X_max - X_min)
```

- 其中，X 是原始数据，X_min 是该特征的最小值，X_max 是该特征的最大值，X_scaled 是归一化后的数据。
- **应用**： 常用于**特征值相差较大**的情况，例如将年龄和收入进行归一化，使它们在相同的尺度上进行比较。
- **代码模块**： scikit-learn 的 `MinMaxScaler` 模块可以实现 Min-Max 归一化。

- **Z-Score 标准化**
- **原理**： 将数据特征的取值转换成均值为 0，标准差为 1 的标准正态分布。**公式为：**

```
X_scaled = (X - X_mean) / X_std
```

- 其中，X 是原始数据，X_mean 是该特征的均值，X_std 是该特征的标准差，X_scaled 是标准化后的数据。
- **应用**： 常用于**特征值分布不均匀**的情况，例如将身高和体重进行标准化，使它们在相同的分布上进行比较。
- **代码模块**： scikit-learn 的 `StandardScaler` 模块可以实现 Z-Score 标准化。

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Min-Max 归一化
scaler = MinMaxScaler()
X_minmax = scaler.fit_transform(X)

# Z-Score 标准化
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

模块四：数据抽取

数据抽取是从数据集中抽取特定字段或记录，主要方法包括：

- **字段抽取**: 抽取某列上指定位置的数据构成新列。

案例： 从身份证号码中抽取出生年份。

```python
import pandas as pd

# 创建包含身份证号码的 DataFrame
df = pd.DataFrame({'id': ['110101199901012345', '120102198802022345',
'130103197703032345']})

# 使用 str.slice() 抽取身份证号码中的出生年份
df['birth_year'] = df['id'].str.slice(start=6, end=10)

print(df)
```

输出:

```
        id birth_year
0  110101199901012345      1999
1  120102198802022345      1988
2  130103197703032345      1977
```

- **记录抽取**: 抽取满足给定条件的行。

案例： 抽取年龄大于 30 岁的记录。

```python
import pandas as pd

# 创建包含年龄信息的 DataFrame
df = pd.DataFrame({'name': ['Alice', 'Bob', 'Charlie'],'age': [25, 35, 45]})

# 使用条件索引抽取年龄大于 30 岁的记录
df_filtered = df[df['age'] > 30]

print(df_filtered)
```

输出:

```
      name  age
1      Bob   35
2  Charlie   45
```

- **随机抽取**: 随机从数据中抽取行数据。

案例： 随机抽取 3 条记录。

```
import pandas as pd
import numpy as np

# 创建 DataFrame
df = pd.DataFrame({'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],'age': [25, 30, 35, 40, 45]
})

# 使用 numpy.random.randint() 随机生成索引
indices = np.random.randint(0, df.shape[0], size=3)

# 使用 iloc 抽取记录
df_sampled = df.iloc[indices]

print(df_sampled)
```

输出:

```
      name  age
1      Bob   30
3    David   40
4      Eve   45
```

总结:

- 字段抽取可以使用 str.slice() 等字符串操作方法，从字符串中抽取特定位置的字符或子字符串。
- 记录抽取可以使用条件索引，根据条件表达式筛选满足条件的记录。
- 随机抽取可以使用 numpy.random.randint() 生成随机索引，并使用 iloc 抽取记录。

案例1：KNN

```
#一段KNN代码样例
import numpy as np
import numpy.random as rand
import matplotlib.pyplot as plt
rand.seed(1)
```

```
6
7  # 创建一个形状为 (100, 2) 的二维数组，每个元素都是从 [0, 1) 范围内均匀分布随机抽取的。然后将每个元素乘以 10 并减去 5，将范围转换为 [-5, 5)。
8
9  X = rand.rand(100,2)*10-5
10
11 # 从 X 的 100 行中随机选择一行作为索引 i，replace=False 表示不放回抽样，确保每次运行代码时选择的点不同。
12
13 i = rand.choice(X.shape[0],1,replace=False)
14
15 # 通过索引 i 获取 X 中对应行的数据，即选定点的坐标
16 selected = X[i]
17
18 # 计算 X 中每个点到选定点的欧氏距离的平方。(X - selected) 得到一个形状为 (100, 2) 的二维数组，每个元素代表一个点到选定点的距离向量。(X - selected)**2 计算距离向量的平方。np.sum((X - selected)**2, axis=1) 沿着 axis=1 方向（即列方向）求和，得到一个长度为 100 的一维数组，每个元素代表一个点到选定点的距离的平方。
19 dist =np.sum((X - selected)**2,axis=1) # 3.计算该点到各点的平方距离
20
21 #对距离数组 dist 进行排序，返回排序后的索引值数组 idx。
22 idx = np.argsort(dist) # 4.距离数组排序，返回索引值数组
23
24 neighbor = X[idx[1:k+1]] 通过索引 idx 中的前 k 个元素（去掉选定点本身），获取 X 中对应行的数据，即 k 个最近邻的坐标。
25 k=3
26 neighbor = X[idx[1:k+1]] # 5.取k个最近邻的坐标(去掉自己)
27
28 # 画图
29 plt.scatter(X[:,0],X[:,1],alpha=0.3)
30 plt.plot((selected[0,0]),(selected[0,1]),'rD')
31 plt.scatter(neighbor[:,0],neighbor[:,1],facecolor='none',47edgecolor='b',s=300)
```

简化后的代码

```
1  import numpy as np
2  import numpy.random as rand
3  import matplotlib.pyplot as plt
4  from sklearn.neighbors import KNeighborsClassifier
5  rand.seed(1)
6  points = rand.rand(100,2)*10-5 # 创建200个[-5,5)分布随机点的坐标
7  knn = KNeighborsClassifier(n_neighbors=3) # 创建KNN分类器，k=3
8  knn.fit(points, np.arange(100)) # 用points训练KNN分类器，将每个点的索引作为标签
9  dist, idx = knn.kneighbors(points) # 计算points中每个点到其他点的距离和索引
10 plt.scatter(points[:,0], points[:,1], alpha=0.3) # 绘制点云
```

```
plt.plot(points[idx[0,1],0], points[idx[0,1],1], 'rD') # 标记选定点
plt.scatter(points[idx[0,1:4],0], points[idx[0,1:4],1], facecolor='none', edgecolor='b', s=300) # 标记最近邻
plt.show()
```

案例 2：有害海藻数据预处理

1. **步骤**:
   - 读取数据，并指定列名。
   - 剔除 ‘a7’ 为 NaN 的行。
   - 将 “XXXXXXX” 替换为 NaN。
   - 删除非 NaN 项数小于 13 的行。
   - 用均值填充剩余的 NaN 值。
   - 对类别特征进行 one-hot 编码。
   - 将预处理后的数据保存到文件。

```
import pandas as pd
import numpy as np

# 读取数据，并指定列名
data = pd.read_csv('analysis.data', header=None)
data.columns = ['season', 'size', 'speed', 'mxPH', 'mnO2', 'Cl', 'NO3', 'NH4', 'oPO4', 'PO4', 'Chla', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']

# 剔除 'a7' 为 NaN 的行
data = data.dropna(subset=['a7'])

# 将 "XXXXXXX" 替换为 NaN
data.replace(to_replace='XXXXXXX', value=np.nan, inplace=True)

# 删除非 NaN 项数小于 13 的行
data = data.dropna(thresh=13)

# 用均值填充剩余的 NaN 值
data.fillna(data.mean(), inplace=True)

# 对类别特征进行 one-hot 编码
data_encoded = pd.get_dummies(data, columns=['season', 'size', 'speed'])

# 将预处理后的数据保存到文件
data_encoded.to_csv('train.csv', index=False)
```

案例 3：泰坦尼克号幸存者数据预处理

1. **步骤**:
   - 读取数据，并以 PassengerId 为行索引。
   - 提取 Survived 列作为目标向量 y，其余列为特征矩阵 X。
   - 删除无用特征 ‘Name’，‘Ticket’，‘Cabin’。
   - 用 0 填充缺失数据。
   - 将性别数据 ‘male’ 替换为 0，‘female’ 替换为 1。
   - 对 ‘Embarked’ 进行 one-hot 编码。

```python
import pandas as pd

# 读取数据，并以 PassengerId 为行索引
data = pd.read_csv('train.csv', index_col='PassengerId')

# 提取 Survived 列作为目标向量 y，其余列为特征矩阵 X
y = data['Survived']
X = data.drop('Survived', axis=1)

# 删除无用特征 'Name', 'Ticket', 'Cabin'
X = X.drop(['Name', 'Ticket', 'Cabin'], axis=1)

# 用 0 填充缺失数据
X.fillna(0, inplace=True)

# 将性别数据 'male' 替换为 0, 'female' 替换为 1
X['Sex'] = X['Sex'].map({'male': 0, 'female': 1})

# 对 'Embarked' 进行 one-hot 编码
X = pd.get_dummies(X, columns=['Embarked'])

# （可选）数据标准化
X = (X - X.mean()) / X.std()

print(X.head())
```

输出:

```
    Pclass  Sex  Age  SibSp  Parch  Fare  Embarked_C  Embarked_Q  Embarked_S
0        3    0  22.0      1      0   7.250       0.0         0.0         1.0
1        1    1  38.0      1      0  71.283       0.0         1.0         0.0
2        3    1  26.0      0      0   7.925       0.0         0.0         1.0
3        1    0  35.0      1      0  53.100       0.0         0.0         1.0
4        3    0  35.0      0      0   8.050       0.0         0.0         1.0
```