数据预处理实践课

数据归一化与标准化 — Python实践教程

授课教师：李阳源

2026年3月

✨ 讲义内容 + Python代码 + 运行结果 + 详细解说

实验一：环境准备与数据加载

教授：同学们，今天我们用Python来实践刚才学的归一化和标准化。首先，我们需要导入必要的库，然后创建一组模拟数据。

1.1 导入必要的 Python 库

说明：以下是本次实验需要用到的Python库。numpy用于数值计算，pandas用于数据处理，matplotlib用于可视化，scikit-learn提供经典的数据预处理工具。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler


# 设置中文显示（解决图表中文乱码问题）
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False


print('☑ 所有库导入成功！')
```

1.2 创建模拟数据集

教授：现在我们来创建一个具体的数据集。假设我们有一组“公寓信息”数据，包含面积、价格和建筑年份三个特征。注意这三个特征的尺度差异非常大。

```
# 设置随机种子，保证结果可复现
np.random.seed(42)


# 创建模拟数据
data = pd.DataFrame({
    '面积(m²)':    np.random.uniform(20, 200, 100),     # 20~200
    '价格(万元)':  np.random.uniform(100, 3000, 100),  # 100~3000
    '建筑年份':    np.random.randint(1980, 2024, 100)  # 1980~2023
})


print('—' * 50)
print('原始数据前5行：')
print(data.head())
print('—' * 50)
print('原始数据统计信息：')
print(data.describe().round(2))
```

“预期输出”：

```
——————————————————————————————————————————————
原始数据前5行：
    面积(m²)    价格(万元)    建筑年份
```

```
0   87.45      1538.22     2005
1  171.86      2287.64     1991
2  146.37       834.55     2018
...
——————————————————————————————————————

→ 注意观察：面积是20~200，价格是100~3000，年份是1980~2023
→ 尺度差异巨大！这就是为什么需要数据缩放。
```

教授：大家可以看到，面积的范围是20到20m²，价格是100到3000万元，建筑年份是1980劫2023年。这三个特征的尺度完全不同，如果直接用于机器学习，价格特征就会主导模型。

实验二：Min-Max 归一化

2.1 手动计算 Min-Max 归一化

教授：我们先手动计算一遍，这样能更深刻地理解公式。公式是：X_normalized = (X - X_min) / (X_max - X_min)。

```
# 手动计算 Min-Max 归一化
simple_data = np.array([5, 10, 15, 20, 25])
x_min = simple_data.min()   # 5
x_max = simple_data.max()   # 25


# 套用公式：(X - X_min) / (X_max - X_min)
normalized = (simple_data - x_min) / (x_max - x_min)


print(f'原始数据：    {simple_data}')
print(f'最小值：      {x_min}')
print(f'最大值：      {x_max}')
print(f'归一化结果：  {normalized}')
```

“预期输出”：

```
原始数据：    [ 5 10 15 20 25]
最小值：      5
最大值：      25
归一化结果：  [0.   0.25 0.5  0.75 1.  ]
```

教授：看到了吗？最小值5变成0，最大值25变成1，其他值均匀分布在中间。数据之间的相对关系完全保持不变。

2.2 用 scikit-learn 进行 Min-Max 归一化

教授：实际工作中，我们通常用scikit-learn库提供的MinMaxScaler。它内部就是用的这个公式，但更方便。

```
# 用 scikit-learn 的 MinMaxScaler
scaler_mm = MinMaxScaler()
```

```
# fit_transform: 先学习参数(min, max)，再进行变换
data_minmax = pd.DataFrame(
    scaler_mm.fit_transform(data),
    columns=data.columns
)

print('−' * 50)
print('Min-Max 归一化后的数据：')
print(data_minmax.head())
print('−' * 50)
print('统计信息：')
print(data_minmax.describe().round(4))
```

“预期输出”：

```
→ 所有特征的最小值变成00，最大值变成10
→ 三个特征现在都在 [0, 1] 范围内！
```

2.3 可视化对比：归一化前后

教授：让我们用箱线图来直观地看看归一化前后的变化。

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# 左图：原始数据
data.boxplot(ax=axes[0])
axes[0].set_title('原始数据', fontsize=14)
axes[0].set_ylabel('值')

# 右图：Min-Max归一化后
data_minmax.boxplot(ax=axes[1])
axes[1].set_title('Min-Max归一化后', fontsize=14)
axes[1].set_ylabel('值')

plt.tight_layout()
plt.savefig('01_minmax_comparison.png', dpi=150)
plt.show()
print('☑ 图表已保存：01_minmax_comparison.png')
```

教授：左图中，价格特征的箱线图远远高于其他两个特征。右图中，三个特征都被压缩到了0到1之间，现在它们的尺度终于一致了。

2.4 观察离群值对 Min-Max 的影响

教授：我们在讲义中提到过，Min-Max归一化对离群值非常敏感。现在我们来验证这一点。

```
# 创建含有离群值的数据
data_outlier = np.array([5, 10, 15, 20, 25, 1000])  # 1000是离群值

# Min-Max 归一化
normalized_outlier = (data_outlier - data_outlier.min()) / (data_outlier.max() -
data_outlier.min())
```

```
print('原始数据：        ', data_outlier)
print('归一化结果：      ', normalized_outlier.round(4))
print()
print('⚠ 问题：5~25的数据全部被压缩到了 0~0.02的很小范围内！')
print('    细节信息大量丢失，这就是Min-Max对离群值敏感的原因。')
```

“预期输出”：

```
原始数据：        [   5   10   15   20   25 1000]
归一化结果：      [0.     0.005  0.01   0.0151 0.0201 1.    ]


⚠ 问题：5~25的数据全部被压缩到了 0~0.02的很小范围内！
```

实验三：Z-score 标准化

3.1 手动计算 Z-score 标准化

教授：接下来我们来学习Z-score标准化。公式是：X_standardized = (X - μ) / σ，其中μ是均值，σ是标准差。

```
# 手动计算 Z-score 标准化
simple_data = np.array([5, 10, 15, 20, 25])


mu = simple_data.mean()       # 均值 = 15
sigma = simple_data.std()     # 标准差 = 7.07


# 套用公式：(X - μ) / σ
standardized = (simple_data - mu) / sigma


print(f'原始数据：    {simple_data}')
print(f'均值(μ)：     {mu}')
print(f'标准差(σ)：  {sigma:.2f}')
print(f'标准化结果：  {np.round(standardized, 2)}')
```

“预期输出”：

```
原始数据：    [ 5 10 15 20 25]
均值(μ)：     15.0
标准差(σ)：  7.07
标准化结果：  [-1.41 -0.71  0.    0.71  1.41]
```

教授：15是均值，所以Z值为0。比15小的数据变为负数，比15大的变为正数。和归一化不同，标准化的结果没有固定的范围限制。

3.2 用 scikit-learn 进行 Z-score 标准化

```
# 用 scikit-learn 的 StandardScaler
```

```
scaler_ss = StandardScaler()


data_zscore = pd.DataFrame(
    scaler_ss.fit_transform(data),
    columns=data.columns
)


print('—' * 50)
print('Z-score 标准化后的数据：')
print(data_zscore.head())
print('—' * 50)
print('统计信息（均值≈ 0，标准差≈ 1）：')
print(data_zscore.describe().round(4))
```

“预期输出”：

```
→ 所有特征的均值接近0，标准差接近1
→ Z值通常在 [-3, +3] 范围内
```

3.3 可视化对比：标准化前后

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))


# 左图：原始数据
data.boxplot(ax=axes[0])
axes[0].set_title('原始数据', fontsize=14)


# 右图：Z-score标准化后
data_zscore.boxplot(ax=axes[1])
axes[1].set_title('Z-score标准化后', fontsize=14)


plt.tight_layout()
plt.savefig('02_zscore_comparison.png', dpi=150)
plt.show()
print('☑ 图表已保存：02_zscore_comparison.png')
```

教授：标准化后，三个特征的分布都以0为中心，而且扩展程度（标准差）都是1。这样不同特征就可以公平地进行比较了。

3.4 离群值对比：Min-Max vs Z-score

教授：现在我们来对比一下，面对相同的离群值数据，两种方法的表现有什么不同。

```
# 对比 Min-Max 和 Z-score 对离群值的处理
data_outlier = np.array([5, 10, 15, 20, 25, 1000]).reshape(-1, 1)


mm_result = MinMaxScaler().fit_transform(data_outlier).flatten()
zs_result = StandardScaler().fit_transform(data_outlier).flatten()


comparison = pd.DataFrame({
    '原始数据': data_outlier.flatten(),
    'Min-Max': mm_result.round(4),
```

```
    'Z-score': zs_result.round(4)
})
print(comparison)
print()
print('→ Min-Max: 5~25的数据被压缩到 0~0.02—细节丢失！')
print('→ Z-score: 5~25的数据仍然有明显区分—更鲁棒！')
```

“预期输出”：

```
   原始数据  Min-Max  Z-score
0       5   0.0000  -0.4565
1      10   0.0050  -0.4428
2      15   0.0100  -0.4291
3      20   0.0151  -0.4154
4      25   0.0201  -0.4017
5    1000   1.0000   2.1455


→ Min-Max: 5~25的数据被压缩到 0~0.02
→ Z-score: 5~25的数据仍然可以区分
```

实验四：对数变换

4.1 基本对数变换

教授：对数变换是一种非线性变换，能把偏斜分布变得更接近正态分布。计算公式很简单：X_log = log(X)。

```
# 基本对数变换
data_log = np.array([1, 10, 100, 1000, 10000])


# 底数10的对数
log_result = np.log10(data_log)


print('原始数据：     ', data_log)
print('对数变换后：   ', log_result)
print()
print('→ 原来差10000倍的数据，变换后变成了等间距的 [0,1,2,3,4]！')
```

“预期输出”：

```
原始数据：     [    1    10   100  1000 10000]
对数变换后：   [0. 1. 2. 3. 4.]
```

4.2 可视化：对数变换前后的分布变化

教授：我们来生成一组右偏分布的数据（模拟收入数据），看看对数变换如何改变分布形态。

```
# 生成右偏分布数据（模拟收入）
np.random.seed(42)
```

```
income = np.random.exponential(scale=5000, size=1000)


# 对数变换（用 log1p 处理可能的零值）
income_log = np.log1p(income)


fig, axes = plt.subplots(1, 2, figsize=(14, 5))


axes[0].hist(income, bins=50, color='#FF6B6B', alpha=0.7, edgecolor='white')
axes[0].set_title('原始收入分布（右偏）', fontsize=14)
axes[0].set_xlabel('收入')
axes[0].set_ylabel('频数')


axes[1].hist(income_log, bins=50, color='#4ECDC4', alpha=0.7, edgecolor='white')
axes[1].set_title('对数变换后（接近正态）', fontsize=14)
axes[1].set_xlabel('log(收入)')
axes[1].set_ylabel('频数')


plt.tight_layout()
plt.savefig('03_log_transform.png', dpi=150)
plt.show()
print('☑ 图表已保存：03_log_transform.png')
```

教授：左图的原始分布明显右偏，大量数据挤在左边，右边拖了一个很长的尾巴。右图经过对数变换后，分布变得更对称、更接近正态分布了。

实验五：Robust Scaling（鲁棒缩放）

教授：讲义中我们还提到了Robust Scaling，它用中位数和四分位距来缩放数据，对离群值非常鲁棒。公式是：X_robust = (X - 中位数) / IQR。

```
# Robust Scaling
scaler_rb = RobustScaler()


data_robust = pd.DataFrame(
    scaler_rb.fit_transform(data),
    columns=data.columns
)


print('—' * 50)
print('Robust Scaling 后的数据：')
print(data_robust.head())
print('—' * 50)
print('统计信息：')
print(data_robust.describe().round(4))
```

实验六：三种方法综合对比

教授：现在让我们把三种主要的缩放方法放在一起来对比，看看它们对数据的影响有什么不同。

```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))


# 原始数据
data.boxplot(ax=axes[0, 0])
axes[0, 0].set_title('原始数据', fontsize=13)


# Min-Max 归一化
data_minmax.boxplot(ax=axes[0, 1])
axes[0, 1].set_title('Min-Max 归一化', fontsize=13)


# Z-score 标准化
data_zscore.boxplot(ax=axes[1, 0])
axes[1, 0].set_title('Z-score 标准化', fontsize=13)


# Robust Scaling
data_robust.boxplot(ax=axes[1, 1])
axes[1, 1].set_title('Robust Scaling', fontsize=13)


plt.suptitle('四种数据缩放方法对比', fontsize=16, fontweight='bold')
plt.tight_layout()
plt.savefig('04_all_comparison.png', dpi=150)
plt.show()
print('☑ 图表已保存：04_all_comparison.png')
```

实验七：实际工作流程 — 训练集/测试集的正确缩放方式

教授：这是今天最重要的实验！讲义中强调过：先分割训练集和测试集，再做缩放。而且测试集要用训练集的统计量。这是防止信息泄露的关键。

7.1 错误做法 vs 正确做法

```
from sklearn.model_selection import train_test_split


# ── ❌ 错误做法：先缩放，再分割 ──
print('=' * 50)
print('❌ 错误做法：先缩放，再分割')
print('=' * 50)


scaler_wrong = StandardScaler()
data_scaled_wrong = scaler_wrong.fit_transform(data)  # 用全部数据缩放！
X_train_w, X_test_w = train_test_split(data_scaled_wrong, test_size=0.2, random_state=42)
```

```
print('训练集均值：', X_train_w.mean(axis=0).round(4))
print('测试集均值：', X_test_w.mean(axis=0).round(4))
print('⚠ 问题：缩放时用了测试集的信息，造成信息泄露！')


# ── ☑ 正确做法：先分割，再缩放 ──
print()
print('=' * 50)
print('☑ 正确做法：先分割，再缩放')
print('=' * 50)


# Step 1: 先分割
X_train, X_test = train_test_split(data, test_size=0.2, random_state=42)


# Step 2: 在训练集上 fit（学习参数）
scaler_correct = StandardScaler()
X_train_scaled = scaler_correct.fit_transform(X_train)  # 只用训练集


# Step 3: 在测试集上只 transform（用训练集的参数）
X_test_scaled = scaler_correct.transform(X_test)  # 注意：只用transform，不用fit！


print(f'训练集均值： {X_train_scaled.mean(axis=0).round(4)}')
print(f'训练集标准差：{X_train_scaled.std(axis=0).round(4)}')
print(f'测试集均值： {X_test_scaled.mean(axis=0).round(4)}')
print()
print('☑ 正确！训练集均值≈0、标准差≈1')
print('   测试集均值不一定为0——这才是正常的！')
```

“核心要点”：

- fit_transform() = 学习参数 + 变换 → 只用于训练集
- transform()　= 只变换（用已学的参数） → 用于测试集
- 绝对不能在测试集上用 fit_transform()!

实验总结：方法选择指南

教授：最后，我们来总结一下今天学习的所有方法，以及如何选择。

方法	适用场景	优点	注意事项
Min-Max	神经网络、图像处理	固定范围[0,1]，直观	对离群值敏感
Z-score	线性回归、SVM、PCA	对离群值鲁棒	无固定范围
对数变换	偏斜分布、幂律数据	压缩极端值，接近正态	不能处理零/负值

Robust	含大量离群值的数据	不受离群值影响	结果范围不固定

“记住数据预处理的正确顺序”：

先处理离群值 → 分割训练/测试集 → 用训练集 `fit` → 分别 `transform` → 建模

教授：同学们，今天的实践课就到这里。希望通过今天的Python实验，大家对归一化、标准化和对数变换有了更直观的理解。课后请大家自己动手运行代码，多试试不同的参数和数据，加深理解。下节课我们将学习缺失值处理。谢谢大家！