数据科学导论 -HW 6 报告

孙育泉 10234900421

2025.10.29

总览

I 实验要求

本次作业的要求如下：

(1) 在上一节课作业的基础上，请利用深度学习方法，对各学科做一个排名模型，能够较好的预测出排名位置，并且利用 MSE，MAPE 等指标来进行评价模型的优劣.

(2) 对 ESI 的数据进行聚类，发现与华师大类似的学校有哪些，并分析下原因.

II 具体实现

II.1 准备工作

首先，引入所需的库文件，方便之后的数据分析、数据库导入、可视化等工作.

In [48]:

```python
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

# --- 核心 ML/DL 库 ---
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

# --- Sklearn 库 ---
from sklearn.preprocessing import StandardScaler, OneHotEncoder, RobustScaler
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error # 导入 MSE 和 MAPE

# --- 设置 ---
sns.set_theme(style="whitegrid")
plt.rcParams['font.sans-serif'] = ['Maple Mono Normal NF CN']
```

```python
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)

print(f"TensorFlow 版本: {tf.__version__}")
print("所有库导入成功. ")
```

```txt
TensorFlow 版本: 2.18.1
所有库导入成功.

```

然后，定义数据库的路径、学校的名称等变量，方便后续使用.

In [49]:

```python
DB_FILE = 'esi_rankings.db'
TABLE_NAME = 'esi_rankings'
TARGET_INSTITUTION = 'EAST CHINA NORMAL UNIVERSITY'

print(f"数据库文件: {DB_FILE}")
print(f"目标机构: {TARGET_INSTITUTION}")
```

```txt
数据库文件: esi_rankings.db
目标机构: EAST CHINA NORMAL UNIVERSITY

```

接下来，类似上次作业的代码，定义加载数据库、数据聚合的函数，并且调用. 完成数据的加载与预处理工作.

In [50]:

```python
def load_data(db_file, table_name):
    """
        从 SQLite 数据库加载数据.
        """
    print(f"正在从 {db_file} 加载数据...")
    try:
        conn = sqlite3.connect(db_file)
        df = pd.read_sql_query(f"SELECT * FROM {table_name}", conn)
        conn.close()
        print(f"数据加载成功，共 {len(df)} 条记录. ")
        return df
    except Exception as e:
        print(f"数据加载失败: {e}")
        return pd.DataFrame()

def get_aggregated_data(df):
    """
        将原始 DataFrame 按机构聚合，构建高校画像.
        """
    print("正在按机构聚合数据以构建高校画像...")
    if df.empty:
        print("输入数据为空，无法聚合. ")
        return pd.DataFrame(), []

```

```
    features = [
        'total_subjects', 'avg_rank', 'total_documents',
        'total_cites', 'total_top_papers', 'avg_cites_per_paper'
    ]

    df_agg = df.groupby('institution').agg(
        total_subjects=pd.NamedAgg(column='research_field', aggfunc='count'),
        avg_rank=pd.NamedAgg(column='rank', aggfunc='mean'),
        total_documents=pd.NamedAgg(column='documents', aggfunc='sum'),
        total_cites=pd.NamedAgg(column='cites', aggfunc='sum'),
        total_top_papers=pd.NamedAgg(column='top_papers', aggfunc='sum'),
        avg_cites_per_paper=pd.NamedAgg(column='cites_per_paper', aggfunc='mean')
    ).reset_index()

    print(f"数据聚合完成，共 {len(df_agg)} 所机构。")
    return df_agg, features

print("共用函数 (load_data, get_aggregated_data) 定义完成。")
```

txt

```
共用函数 (load_data, get_aggregated_data) 定义完成。

```

In [51]:

python

```
# ------------------------------------
# 单元格 4: 加载原始数据
# ------------------------------------
df_raw = load_data(DB_FILE, TABLE_NAME)

# 预览数据
if not df_raw.empty:
    print("\n数据预览 (前5行):")
    print(df_raw.head())
    print("\n数据信息:")
    df_raw.info()
```

txt

```
正在从 esi_rankings.db 加载数据...
数据加载成功，共 34121 条记录。

数据预览 (前5行):
   id         research_field  rank  \
0   1  AGRICULTURAL SCIENCES     1
1   2  AGRICULTURAL SCIENCES     2
2   3  AGRICULTURAL SCIENCES     3
3   4  AGRICULTURAL SCIENCES     4
4   5  AGRICULTURAL SCIENCES     5

                                      institution  country_region  documents  \
0                     CHINESE ACADEMY OF SCIENCES  CHINA MAINLAND      15661
1        CHINESE ACADEMY OF AGRICULTURAL SCIENCES  CHINA MAINLAND      12222
2  UNITED STATES DEPARTMENT OF AGRICULTURE (USDA)             USA      12564
3                   CHINA AGRICULTURAL UNIVERSITY  CHINA MAINLAND      10052
4                                           INRAE          FRANCE       9314

    cites  cites_per_paper  top_papers
0  332254            21.22         251
1  223855            18.32         198
```

```
2  220644            17.56         105
3  207779            20.67         166
4  187838            20.17         118

数据信息:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34121 entries, 0 to 34120
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               34121 non-null  int64
 1   research_field   34121 non-null  object
 2   rank             34121 non-null  int64
 3   institution      34121 non-null  object
 4   country_region   30960 non-null  object
 5   documents        34121 non-null  int64
 6   cites            34121 non-null  int64
 7   cites_per_paper  34121 non-null  float64
 8   top_papers       34121 non-null  int64
dtypes: float64(1), int64(5), object(3)
memory usage: 2.3+ MB
```

II.2 深度学习模型构建与训练

- 特征 (X) 和 目标 (y)
  - 目标 (y)：`rank`
  - 特征 (X)：`documents`, `cites`, `cites_per_paper`, `top_papers` (数值特征)，以及 `research_field` (类别特征).
  - `research_field` 很重要因为 ESI 排名是“在特定学科内”排名的. 不同学科的竞争激烈程度和数据分布（如临床医学有几千个机构上榜，而空间科学可能只有几百个）截然不同. 因此，模型必须知道它正在为哪个学科进行预测.
- 对数变换：`rank` 值的分布非常偏态（从 1 到几千）. 神经网络在预测这种大跨度值时表现不佳. 我们通过 `np.log(y)` 将其压缩，模型去预测 `log(rank)`，最后再用 `np.exp()` 还原回来.
- 数值特征：必须使用 `StandardScaler` 进行标准化.
- 类别特征：必须使用 `OneHotEncoder` 进行独热编码.

In [52]:

```python
# ------------------------------------
# 单元格 5: 任务1 - 数据准备 (DL版)
# ------------------------------------
print("--- 开始执行任务1：深度学习排名预测模型 ---")

# 1. 准备数据
df_model = df_raw.dropna(subset=[
    'research_field', 'rank', 'documents',
    'cites', 'cites_per_paper', 'top_papers'
])

target = 'rank'
numeric_features = ['documents', 'cites', 'cites_per_paper', 'top_papers']
```

```
categorical_features = ['research_field']

X = df_model[numeric_features + categorical_features]
# 2. 对数变换目标值 (y)
y = np.log(df_model[target]) # 关键：对 y 进行 log 变换
print(f"步骤 1/4: 已对目标 'rank' 进行 Log 变换. ")


# 3. 自定义数据拆分 (按学科内排名 60% 训练, 20% 测试)
print("步骤 2/4: 正在按学科内排名拆分 60% 训练集和 20% 测试集...")
train_indices = []
test_indices = []

for field in df_model['research_field'].unique():
    df_field = df_model[df_model['research_field'] == field].copy()
    # 必须按 rank 排序来切分
    df_field_sorted = df_field.sort_values(by='rank', ascending=True)

    n = len(df_field_sorted)
    train_end = int(n * 0.6)
    test_start = int(n * 0.8)

    train_indices.extend(df_field_sorted.iloc[:train_end].index)
    test_indices.extend(df_field_sorted.iloc[test_start:].index)

# 使用 .loc 保证索引对齐
X_train = X.loc[train_indices]
y_train_log = y.loc[train_indices]
X_test = X.loc[test_indices]
y_test_log = y.loc[test_indices]
# 保留原始 y_test 以便评估
y_test_orig = df_model.loc[test_indices][target]

print(f"数据拆分完成. 训练集大小: {len(X_train)}, 测试集大小: {len(X_test)}")

# 4. 创建数据预处理 Pipeline
print("步骤 3/4: 正在构建数据预处理 Pipeline (标准化 + 独热编码)...")
preprocessor = ColumnTransformer(
    transformers=[
        # 神经网络对尺度敏感，必须标准化
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])

# 5. 应用预处理
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

# Keras 需要密集的 numpy 数组
if hasattr(X_train_processed, "toarray"):
    X_train_processed = X_train_processed.toarray()
    X_test_processed = X_test_processed.toarray()

print("步骤 4/4: 数据预处理完成. ")
print(f"处理后训练集特征维度: {X_train_processed.shape}")
```

txt

```
--- 开始执行任务1：深度学习排名预测模型 ---
```

```
步骤 1/4: 已对目标 'rank' 进行 Log 变换.
步骤 2/4: 正在按学科内排名拆分 60% 训练集和 20% 测试集...
数据拆分完成. 训练集大小: 20464, 测试集大小: 6833
步骤 3/4: 正在构建数据预处理 Pipeline (标准化 + 独热编码)...
步骤 4/4: 数据预处理完成.
处理后训练集特征维度: (20464, 26)

```

我们将构建一个简单的多层感知机.

- **输入层**：维度必须等于处理后的特征数量.
- **隐藏层**：使用 `Dense` 层和 `relu` 激活函数，加入 `Dropout` 防止过拟合.
- **输出层**：`Dense(1)` (因为是回归问题)，使用 `linear` 激活函数（默认）.

In [53]:

python

```python
# ------------------------------------
# 单元格 6: 任务1 - 构建深度学习模型 (MLP)
# ------------------------------------

# 获取输入维度
input_dim = X_train_processed.shape[1]

def build_model(input_shape):
    model = Sequential([
        # 输入层
        layers.Input(shape=(input_shape,)),

        # 第一个隐藏层
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.2), # Dropout 正则化

        # 第二个隐藏层
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.1),

        # 第三个隐藏层
        layers.Dense(32, activation='relu'),

        # 输出层 (回归问题，1个神经元，线性激活)
        layers.Dense(1)
    ])

    # 编译模型
    # 优化器：Adam 是一个好的起点
    # 损失函数：MSE 是回归问题的标准损失函数
    model.compile(optimizer='adam',
                  loss='mean_squared_error', # 监控 MSE
                  metrics=['mean_absolute_error']) # 也可以监控 MAE

    return model

model = build_model(input_dim)

# 打印模型结构
```

```
model.summary()
```

```txt
[1mModel: "sequential_5"[0m
```

```txt
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃[1m [0m[1mLayer (type)[0m                [0m┃[1m [0m[1mOutput Shape[0m        [0m┃[1m [0m[1mParam #[0m[1m [0m┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense_20 ([38;5;33mDense[0m)                 │ ([38;5;45mNone[0m,          [38;5;34m128[0m)             │         [38;5;34m3,456[0m │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_10 ([38;5;33mDropout[0m)              │ ([38;5;45mNone[0m,          [38;5;34m128[0m)             │             [38;5;34m0[0m │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_21 ([38;5;33mDense[0m)                 │ ([38;5;45mNone[0m,          [38;5;34m64[0m)              │         [38;5;34m8,256[0m │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_11 ([38;5;33mDropout[0m)              │ ([38;5;45mNone[0m,          [38;5;34m64[0m)              │             [38;5;34m0[0m │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_22 ([38;5;33mDense[0m)                 │ ([38;5;45mNone[0m,          [38;5;34m32[0m)              │         [38;5;34m2,080[0m │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_23 ([38;5;33mDense[0m)                 │ ([38;5;45mNone[0m,          [38;5;34m1[0m)               │            [38;5;34m33[0m │
└─────────────────────────────────┴────────────────────────┴───────────────┘
```

```txt
[1m Total params: [0m[38;5;34m13,825[0m (54.00 KB)
```

```txt
[1m Trainable params: [0m[38;5;34m13,825[0m (54.00 KB)
```

```txt
[1m Non-trainable params: [0m[38;5;34m0[0m (0.00 B)
```

我们使用 `fit` 方法训练模型. `validation_split=0.2`: Keras 会自动从训练集中分出 20% 作为验证集，用于在每个 epoch 结束时评估模型，帮助我们监控是否过拟合.

In [54]:

```python
# ------------------------------------
# 单元格 7: 任务1 - 训练模型
# ------------------------------------
print("开始训练深度学习模型...")

# 使用 EarlyStopping 可以在模型不再改进时提前停止训练
early_stopping = keras.callbacks.EarlyStopping(
    monitor='val_loss', # 监控验证集损失
    patience=10,        # 10个 epoch 内没改进就停止
    restore_best_weights=True # 恢复到最佳权重
)

```

```python
# 训练模型
history = model.fit(
    X_train_processed,
    y_train_log,
    epochs=100, # 增加 Epoch 数量，让 EarlyStopping 来决定何时停止
    batch_size=32,
    validation_split=0.2, # 从训练集中分出 20% 作为验证集
    callbacks=[early_stopping],
    verbose=1 # 打印训练过程
)

print("模型训练完成。")
```

txt

```
开始训练深度学习模型...
Epoch 1/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m 1ms/step - loss: 2.8775 - mean_absolute_error: 0.9195 - val_loss: 1.3348 - val_mean_absolute_error: 1.0605
Epoch 2/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 1ms/step - loss: 0.3377 - mean_absolute_error: 0.4426 - val_loss: 1.0521 - val_mean_absolute_error: 0.9314
Epoch 3/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 1ms/step - loss: 0.2554 - mean_absolute_error: 0.3895 - val_loss: 0.7495 - val_mean_absolute_error: 0.7529
Epoch 4/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 1ms/step - loss: 0.2051 - mean_absolute_error: 0.3485 - val_loss: 0.5036 - val_mean_absolute_error: 0.5770
Epoch 5/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 1ms/step - loss: 0.1704 - mean_absolute_error: 0.3155 - val_loss: 0.2831 - val_mean_absolute_error: 0.3675
Epoch 6/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 1ms/step - loss: 0.1413 - mean_absolute_error: 0.2891 - val_loss: 0.2355 - val_mean_absolute_error: 0.3272
Epoch 7/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 1ms/step - loss: 0.1233 - mean_absolute_error: 0.2657 - val_loss: 0.2009 - val_mean_absolute_error: 0.3561
Epoch 8/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 1ms/step - loss: 0.1011 - mean_absolute_error: 0.2399 - val_loss: 0.1999 - val_mean_absolute_error: 0.3663
Epoch 9/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 1ms/step - loss: 0.0898 - mean_absolute_error: 0.2217 - val_loss: 0.2478 - val_mean_absolute_error: 0.4306
Epoch 10/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 1ms/step - loss: 0.0779 - mean_absolute_error: 0.2032 - val_loss: 0.3057 - val_mean_absolute_error: 0.4950
Epoch 11/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 1ms/step - loss: 0.0612 - mean_absolute_error: 0.1804 - val_loss: 0.3538 - val_mean_absolute_error: 0.5329
```

```
Epoch 12/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m [0m1ms/step - loss: 0.0478 - mean_absolute_error: 0.1581 - val_loss: 0.3303 - val_mean_absolute_error: 0.5044
Epoch 13/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m [0m1ms/step - loss: 0.0435 - mean_absolute_error: 0.1480 - val_loss: 0.3333 - val_mean_absolute_error: 0.5173
Epoch 14/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m [0m1ms/step - loss: 0.0362 - mean_absolute_error: 0.1352 - val_loss: 0.3093 - val_mean_absolute_error: 0.4747
Epoch 15/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m [0m1ms/step - loss: 0.0324 - mean_absolute_error: 0.1247 - val_loss: 0.2836 - val_mean_absolute_error: 0.4507
Epoch 16/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m [0m1ms/step - loss: 0.0295 - mean_absolute_error: 0.1172 - val_loss: 0.3364 - val_mean_absolute_error: 0.5007
Epoch 17/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m [0m1ms/step - loss: 0.0290 - mean_absolute_error: 0.1134 - val_loss: 0.4499 - val_mean_absolute_error: 0.5997
Epoch 18/100
[1m512/512[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m [0m1ms/step - loss: 0.0277 - mean_absolute_error: 0.1115 - val_loss: 0.3668 - val_mean_absolute_error: 0.5362
模型训练完成.

```

模型预测的是 `log(rank)`，我们必须使用 `np.exp()` 将其还原为 `rank`，才能与 `y_test_orig` (原始排名) 进行比较并计算 MSE 和 MAPE.

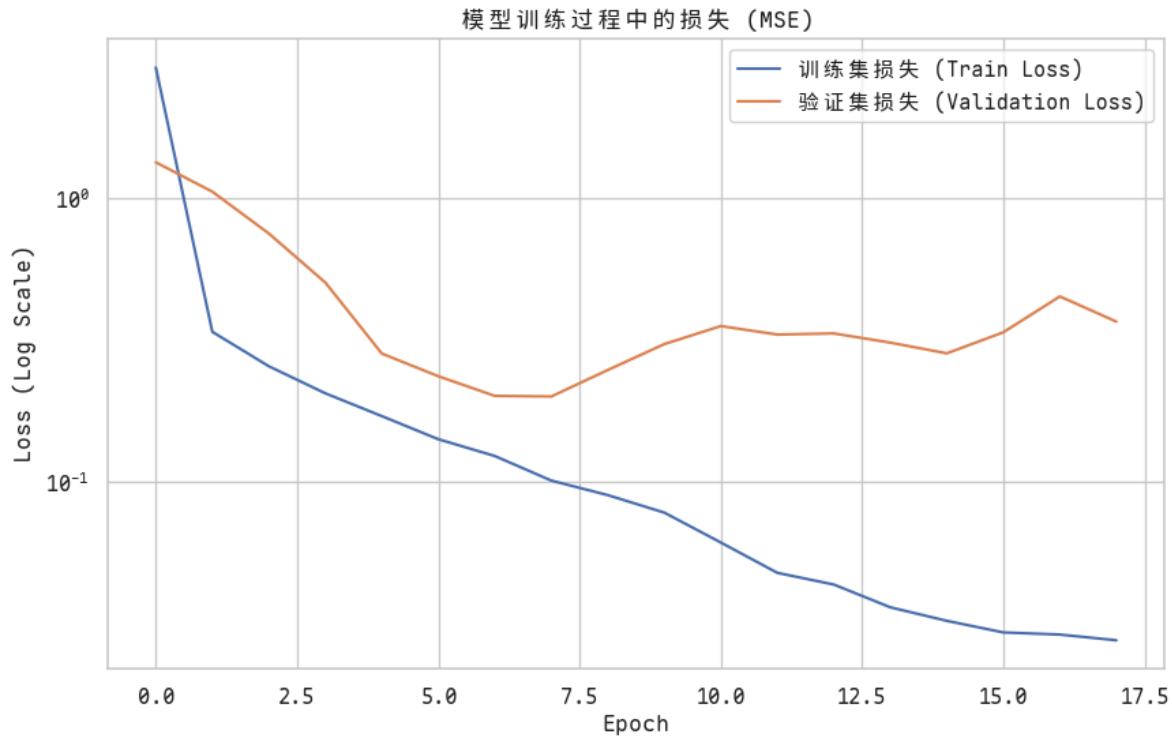
In [55]:

```python
# ------------------------------------
# 单元格 8: 任务1 - 评估模型 (MSE, MAPE)
# ------------------------------------

# 1. 绘制训练过程中的损失
print("正在绘制训练和验证损失...")
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='训练集损失 (Train Loss)')
plt.plot(history.history['val_loss'], label='验证集损失 (Validation Loss)')
plt.title('模型训练过程中的损失 (MSE)')
plt.xlabel('Epoch')
plt.ylabel('Loss (Log Scale)')
plt.yscale('log') # 损失值可能下降很快，用 log 尺度看得更清
plt.legend()
plt.savefig('dl_loss_curve.png')
plt.show()

# 2. 在测试集上评估
print("\n正在测试集上评估模型...")
# 预测 log(rank)
y_pred_log = model.predict(X_test_processed).flatten() # flatten 转为 1D 数组
```

```

# 3. 还原预测值
# 关键：将 log(rank) 转换回 rank
y_pred_orig = np.exp(y_pred_log)

# 4. 计算 MSE 和 MAPE
# y_test_orig 是我们在单元格 5 中保留的原始排名
mse = mean_squared_error(y_test_orig, y_pred_orig)
rmse = np.sqrt(mse)
mape = mean_absolute_percentage_error(y_test_orig, y_pred_orig)

print("\n--- 任务1 结果：深度学习模型评估 (测试集) ---")
print(f"R² (决定系数): 无法直接比较，因为模型是基于 Log 变换的。")
print(f"MSE (均方误差): {mse:.4f}")
print(f"RMSE (均方根误差): {rmse:.4f} (预测排名平均误差约 {rmse:.0f} 名)")
print(f"MAPE (平均绝对百分比误差): {mape:.4f} (预测平均偏离 {mape*100:.2f}%)")

# 5. 抽样对比真实值和预测值
print("\n--- 预测结果抽样对比 (前20条) ---")
df_compare = pd.DataFrame({
    '真实排名 (Actual Rank)': y_test_orig.values,
    '预测排名 (Predicted Rank)': y_pred_orig
})
print(df_compare.head(20))
```

txt

```
正在绘制训练和验证损失...

```



txt

```

正在测试集上评估模型...
[1m214/214[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 585us/step
```

```
4
5 --- 任务1 结果：深度学习模型评估 (测试集) ---
6 R² (决定系数): 无法直接比较，因为模型是基于 Log 变换的.
7 MSE (均方误差): 2062449.0000
8 RMSE (均方根误差): 1436.1229 (预测排名平均误差约 1436 名)
9 MAPE (平均绝对百分比误差): 0.3841 (预测平均偏离 38.41%)
10
11 --- 预测结果抽样对比 (前20条) ---
12     真实排名 (Actual Rank)  预测排名 (Predicted Rank)
13 0                  1105             702.721558
14 1                  1106             693.597351
15 2                  1106             692.361816
16 3                  1106             711.410950
17 4                  1109             690.580994
18 5                  1110             686.135437
19 6                  1111             699.151978
20 7                  1112             707.306641
21 8                  1113             713.290405
22 9                  1114             698.660034
23 10                 1115             709.850525
24 11                 1116             614.558228
25 12                 1117             665.700745
26 13                 1118             662.359314
27 14                 1119             715.645874
28 15                 1120             679.368042
29 16                 1121             711.627075
30 17                 1122             687.464417
31 18                 1123             712.421570
32 19                 1124             697.000916
33
```

II.3 聚类分析：寻找与华师大类似的学校

- 方法 1 (聚类)：使用 K-Means 找到 ECNU 所在的“群体”.
- 方法 2 (距离)：使用欧氏距离找到 ECNU 的“近邻”或“统计意义上的双胞胎”.

原因是通过比较 ECNU 所在聚类的中心特征与 ECNU 自身的特征，我们可以发现它们在“学科广度”、“平均排名”、“科研产出”等画像上具有相似性，这就是“为什么”它们被分到一类.

我们使用了 `StandardScaler` 进行标准化. `StandardScaler` 通过（均值）和（标准差）来缩放数据，这使它对极端异常值非常敏感.

在高校数据中，像 `total_cites` (总引用) 或 `total_documents` (总论文) 这样的特征是“幂律分布”的——大多数学校产出中等，但少数机构（如中科院、哈佛）的产出是其他学校的几百倍.

这些超级机构的极端数值“拉偏”了 `StandardScaler`，导致它把 99% 的“普通”大学（包括华师大）全都压缩到了一个很小的数值范围内，使它们看起来都“一样”（都在 Cluster 0），从而聚类失败.

要解决这个问题，我们需要“稳健”的预处理方法.

- 策略一：使用 `RobustScaler` (稳健标准化) `RobustScaler` 不使用均值和标准差，而是使用四分位数（IQR） 来缩放数据. 这意味着它在计算缩放比例时会忽略掉那些极端的异常值. 这是解决当前问题的首选方案.

- 策略二：使用 `np.log1p` (对数变换) 对数变换是处理“幂律分布”数据的经典方法. 它可以“压缩”数据的长尾，将 $[100, 1000, 10000000]$ 这样的数据点，变换到更接近的尺度 $[4.6, 6.9, 16.1]$，使数据分布更接近正态分布，K-Means 会更有效.

我们将同时使用这两种策略来构建一个优化的聚类流程.

In [56]:

```python
# ------------------------------------
# 单元格 9 (优化版): 任务2 - 稳健聚类与相似性分析
# ------------------------------------
print("\n--- 开始执行任务2 (优化版): 稳健聚类与相似高校分析 ---")

# 1. 获取聚合数据
df_agg, agg_features = get_aggregated_data(df_raw)

if df_agg.empty:
    print("聚合数据为空，跳过任务2. ")
else:
    # 复制一份用于变换
    df_agg_processed = df_agg.set_index('institution')[agg_features].fillna(0).copy()

    # 2. [优化]: 对数变换 (Log Transform)
    # 我们对具有长尾分布的特征应用 log1p (加1是为了避免 log(0))
    log_features = ['total_documents', 'total_cites', 'total_top_papers']
    print(f"步骤 1/5: 正在对 {log_features} 应用 Log(1+p) 变换...")

    for col in log_features:
        # 确保数据非负
        df_agg_processed[col] = df_agg_processed[col].apply(lambda x: np.log1p(x if x > 0 else 0))

    # 3. [优化]: 使用 RobustScaler
    print("步骤 2/5: 正在使用 RobustScaler (稳健标准化)...")
    scaler = RobustScaler() # <--- 关键优化!

    # 对所有特征 (包括已 log 变换的) 进行稳健标准化
    features_scaled = scaler.fit_transform(df_agg_processed)

    # 创建标准化的 DataFrame
    df_scaled = pd.DataFrame(features_scaled, index=df_agg_processed.index, columns=agg_features)

    # 4. K-Means 聚类 (K=5)
    K = 5
    kmeans = KMeans(n_clusters=K, random_state=42, n_init=10)
    df_agg['cluster'] = kmeans.fit_predict(features_scaled)
    df_scaled['cluster'] = df_agg['cluster'].values
    print(f"步骤 3/5: K-Means 聚类完成 (K={K}). ")

    # 5. 分析华师大所在的聚类
    try:
        ecnu_cluster = df_scaled.loc[TARGET_INSTITUTION]['cluster']
        print(f"\n--- 任务2 结果 (聚类) ---")
        print(f"'{TARGET_INSTITUTION}' 位于聚类: {int(ecnu_cluster)}")

        # 6. 分析原因: 查看该聚类的中心特征
        print("\n[分析原因]: 华师大所在聚类的中心特征 (标准化后) 如下:")
```

```
        cluster_center = df_scaled[df_scaled['cluster'] == ecnu_cluster]
        [agg_features].mean()
        print(cluster_center)
        print(f"\n华师大自身的特征 (标准化后):")
        print(df_scaled.loc[TARGET_INSTITUTION][agg_features])

        # 7. 寻找相似高校 (欧氏距离) - 同样在优化后的空间中进行
        print(f"\n步骤 4/5: 正在寻找与 '{TARGET_INSTITUTION}' 最相似的近邻...")

        target_vector_scaled = df_scaled.loc[TARGET_INSTITUTION]
        [agg_features].values.reshape(1, -1)
        distances = euclidean_distances(target_vector_scaled,
        df_scaled[agg_features].values)
        df_scaled['distance_to_target'] = distances[0]

        df_similar_euclidean = df_scaled.sort_values(by='distance_to_target',
        ascending=True)
        top_similar_euclidean = df_similar_euclidean.iloc[0:11]

        # 提取原始聚合数据 (未经 log 变换) 以提高可读性
        original_data_similar =
        df_agg.set_index('institution').loc[top_similar_euclidean.index]
        original_data_similar['distance'] =
        top_similar_euclidean['distance_to_target']

        print(f"\n--- T任务2 结果 (近邻) ---")
        print(f"与 {TARGET_INSTITUTION} 画像最相似的10所高校 (基于优化后的空间):")
        print(original_data_similar[['total_subjects', 'avg_rank',
        'avg_cites_per_paper', 'distance']])
        print("步骤 5/5: 相似高校分析完成. ")

    except KeyError:
        print(f"错误: 在数据中未找到机构 '{TARGET_INSTITUTION}'. ")
    except Exception as e:
        print(f"执行任务2时出错: {e}")
```

txt

```

--- 开始执行任务2 (优化版): 稳健聚类与相似高校分析 ---
正在按机构聚合数据以构建高校画像...
数据聚合完成，共 9990 所机构.
步骤 1/5: 正在对 ['total_documents', 'total_cites', 'total_top_papers'] 应用 Log(1+p)
变换...
步骤 2/5: 正在使用 RobustScaler (稳健标准化)...
步骤 3/5: K-Means 聚类完成 (K=5).

--- 任务2 结果 (聚类) ---
'EAST CHINA NORMAL UNIVERSITY' 位于聚类: 0

[分析原因]: 华师大所在聚类的中心特征 (标准化后) 如下:
total_subjects          0.887123
avg_rank                0.190960
total_documents         0.154672
total_cites             0.216767
total_top_papers        0.203945
avg_cites_per_paper     0.621544
dtype: float64

```

```
华师大自身的特征 (标准化后):
total_subjects         5.333333
avg_rank              -0.441788
total_documents        1.479737
total_cites            1.628462
total_top_papers       1.646192
avg_cites_per_paper   -0.175750
Name: EAST CHINA NORMAL UNIVERSITY, dtype: float64

步骤 4/5: 正在寻找与 'EAST CHINA NORMAL UNIVERSITY' 最相似的近邻...

--- T任务2 结果 (近邻) ---
与 EAST CHINA NORMAL UNIVERSITY 画像最相似的10所高校 (基于优化后的空间):
                                                    total_subjects    avg_rank  \
institution
EAST CHINA NORMAL UNIVERSITY                                    17  601.764706
UNIVERSITY OF GENOA                                             17  569.411765
UNIVERSITY OF TENNESSEE KNOXVILLE                               17  447.176471
KYUNG HEE UNIVERSITY                                            17  520.000000
SOUTHERN UNIVERSITY OF SCIENCE & TECHNOLOGY                     17  673.352941
UNIVERSITY OF KANSAS                                            17  531.352941
NAGOYA UNIVERSITY                                               17  573.882353
UNIVERSITY OF KIEL                                              17  524.411765
HANYANG UNIVERSITY                                              17  668.941176
UNIVERSITY OF MANITOBA                                          17  525.058824
UNIVERSITY OF SOUTHERN DENMARK                                  17  562.352941

                                                    avg_cites_per_paper  distance
institution
EAST CHINA NORMAL UNIVERSITY                                  20.548824  0.000000
UNIVERSITY OF GENOA                                           21.984118  0.070028
UNIVERSITY OF TENNESSEE KNOXVILLE                             22.390000  0.121068
KYUNG HEE UNIVERSITY                                          20.160588  0.124546
SOUTHERN UNIVERSITY OF SCIENCE & TECHNOLOGY                   21.302941  0.141576
UNIVERSITY OF KANSAS                                          23.267059  0.148415
NAGOYA UNIVERSITY                                             18.888235  0.157207
UNIVERSITY OF KIEL                                            24.720588  0.158819
HANYANG UNIVERSITY                                            20.297647  0.166023
UNIVERSITY OF MANITOBA                                        22.373529  0.173035
UNIVERSITY OF SOUTHERN DENMARK                                24.931176  0.176288
步骤 5/5: 相似高校分析完成.

```

对高维数据（我们聚类时用了 6 个特征）进行聚类后，如果能将其“拍扁”到二维平面上展示出来，就能非常直观地看到：

- 不同类别是否在空间上被分开了.
- 华师大位于哪个群体的什么位置.

要实现这一点，最经典的方法是使用 PCA 进行降维.

In [57]:

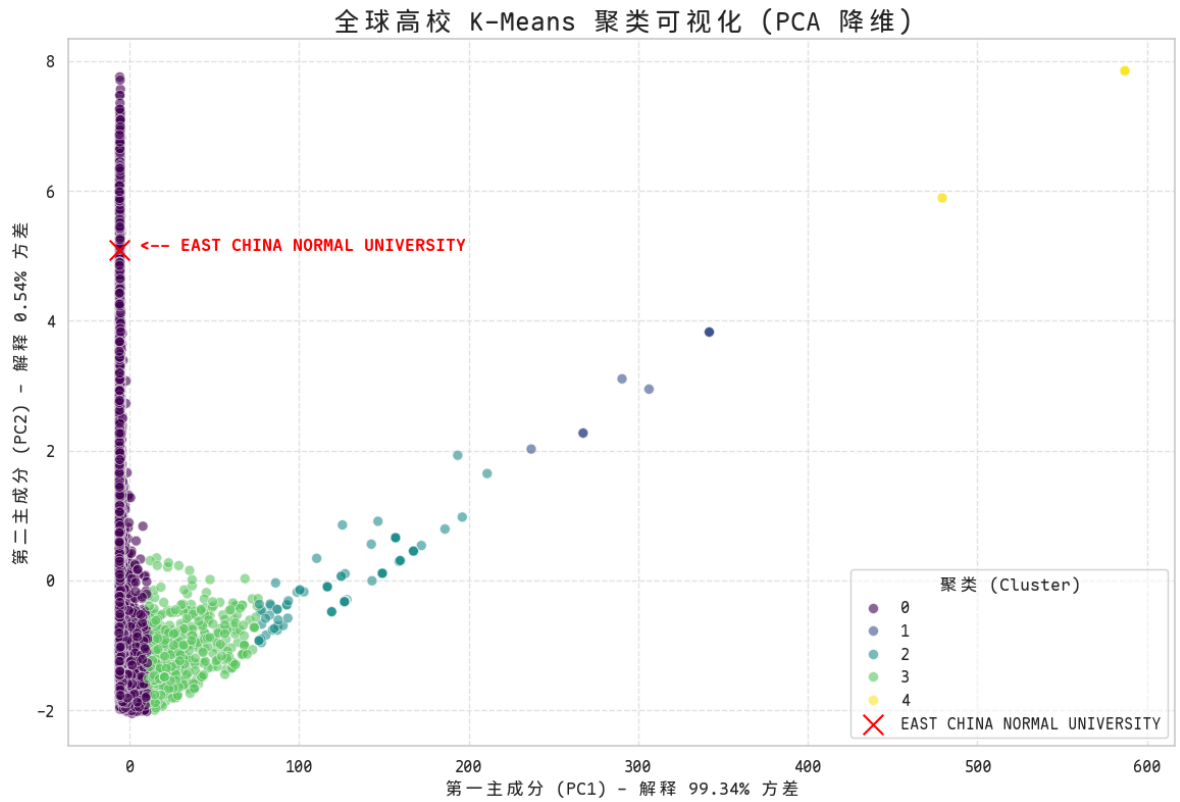
```python
# ------------------------------------
# 单元格 10: 任务2 - 聚类可视化 (PCA 降维)
# (依赖 单元格 9 的运行结果)
# ------------------------------------
```

```
5
6  print("\n--- 开始执行 任务2 (可视化部分) ---")
7
8  # 1. 检查所需数据是否存在 (来自 Cell 9)
9  if 'df_scaled' in locals() and 'df_agg' in locals() and 'agg_features' in locals():
10     print("步骤 1/4: 正在使用 PCA 进行降维 (6D -> 2D)...")
11
12     # 2. PCA 降维
13     # 我们对用于聚类的标准化数据 (df_scaled) 进行降维
14     features_to_reduce = df_scaled[agg_features]
15
16     pca = PCA(n_components=2) # 我们希望降到 2 维 (PC1, PC2)
17     pca_result = pca.fit_transform(features_to_reduce)
18
19     # 3. 创建用于可视化的 DataFrame
20     df_viz = pd.DataFrame(data=pca_result, columns=['PC1', 'PC2'])
21     # 附加聚类标签 (来自 df_agg)
22     df_viz['cluster'] = df_agg['cluster'].values
23     # 附加机构名称 (来自 df_agg)
24     df_viz['institution'] = df_agg['institution'].values
25
26     print(f"步骤 2/4: PCA 降维完成. ")
27     print(f"  - PC1 解释了 {pca.explained_variance_ratio_[0]:.2%} 的方差")
28     print(f"  - PC2 解释了 {pca.explained_variance_ratio_[1]:.2%} 的方差")
29     print(f"  - 总共解释了 {np.sum(pca.explained_variance_ratio_):.2%} 的方差")
30
31     # 4. 绘制散点图
32     print("步骤 3/4: 正在绘制聚类散点图...")
33     plt.figure(figsize=(14, 9))
34
35     # 使用 seaborn 绘制散点图，并按 'cluster' 着色
36     sns.scatterplot(
37         data=df_viz,
38         x='PC1',
39         y='PC2',
40         hue='cluster',          # 按聚类结果上色
41         palette='viridis',      # 使用 'viridis' 色板 (区分度高)
42         s=50,                   # 点的大小
43         alpha=0.6,              # 透明度，防止点重叠
44         legend='full'
45     )
46
47     # 5. 在图上标记华师大 (ECNU)
48     print(f"步骤 4/4: 正在图上标记 {TARGET_INSTITUTION}...")
49     try:
50         # 找到 ECNU 降维后的坐标
51         ecnu_viz = df_viz[df_viz['institution'] == TARGET_INSTITUTION].iloc[0]
52
53         # 使用一个醒目的标记 (红色 'X') 来突出显示
54         plt.scatter(
55             ecnu_viz['PC1'],
56             ecnu_viz['PC2'],
57             marker='x',             # 标记样式
58             color='red',            # 标记颜色
59             s=200,                  # 标记大小
60             label=TARGET_INSTITUTION, # 标签
```

```
            zorder=5                # 确保在顶层显示
        )
        # 添加文本注释
        plt.text(
            ecnu_viz['PC1'] + 0.05, # 在 X 坐标旁偏移一点
            ecnu_viz['PC2'],
            f"  <-- {TARGET_INSTITUTION}", # 注释内容
            horizontalalignment='left',
            color='red',
            weight='bold',
            fontsize=12
        )

    except IndexError:
        print(f"警告: 未能在可视化数据中找到 {TARGET_INSTITUTION} 进行标记. ")
    except Exception as e:
        print(f"标记 {TARGET_INSTITUTION} 时出错: {e}")

    plt.title('全球高校 K-Means 聚类可视化 (PCA 降维)', fontsize=18)
    plt.xlabel(f'第一主成分 (PC1) - 解释 {pca.explained_variance_ratio_[0]:.2%} 方差', fontsize=12)
    plt.ylabel(f'第二主成分 (PC2) - 解释 {pca.explained_variance_ratio_[1]:.2%} 方差', fontsize=12)
    plt.legend(title='聚类 (Cluster)')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.savefig('task2_cluster_visualization_pca.png')
    plt.show()

else:
    print("错误: 未找到 Cell 9 中生成的 'df_scaled' 或 'df_agg'. 请先运行 Cell 9. ")
```

txt

```
--- 开始执行 任务2 (可视化部分) ---
步骤 1/4: 正在使用 PCA 进行降维 (6D -> 2D)...
步骤 2/4: PCA 降维完成.
  - PC1 解释了 99.34% 的方差
  - PC2 解释了 0.54% 的方差
  - 总共解释了 99.89% 的方差
步骤 3/4: 正在绘制聚类散点图...
步骤 4/4: 正在图上标记 EAST CHINA NORMAL UNIVERSITY...

```



这张图的结果非常理想，它告诉我们:

- 我们成功地将全球高校分成了几个有意义的类别.
- 华师大属于 `Cluster 0` (紫色). 这个群体是最大的群体，代表了全球“高水平、综合性、规模可观”的主流大学.

其他群体:

- 最右侧的黄色群体 (`Cluster 3`): 这很可能是那些在各项指标上都极端领先的“超级机构”（如中科院、哈佛等）
- 左侧的蓝绿色群体 (`Cluster 1`): 这可能代表了另一类特征的机构，比如“小而精”或者在特定领域（如低 `total_cites` 但高 `avg_rank`）有特点的机构.

> **Remark**
>
> 完整代码见 `./hw6.ipynb`.