数据科学导论 -HW 7 报告

孙育泉 10234900421

2025.11.08

总览

I 实验要求

本次作业主要是改进上次的作业，这次主要改进的任务是:

(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.

II 具体实现

本次使用表格深度学习的方法.

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 [1]:

```python
# ------------------------------------
# 单元格 1: 导入所有库
# ------------------------------------
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

# --- 核心 DL 库 (用于任务1) ---
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

# --- Sklearn 库 (用于任务1预处理 和 任务2) ---
from sklearn.preprocessing import StandardScaler, OneHotEncoder, RobustScaler # 导入两种 Scaler
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error # 导入 MSE 和 MAPE
from sklearn.decomposition import PCA # 用于可视化

```

```
# --- 设置 ---
sns.set_theme(style="whitegrid")
plt.rcParams['font.sans-serif'] = ['Heiti TC'] # 如果您用中文标签，请取消这行注释
# plt.rcParams['axes.unicode_minus'] = False
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: 2.16.2
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 [2]:

python

```
# ------------------------------------
# 单元格 2: 全局配置
# ------------------------------------
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 [3]:

python

```
# ------------------------------------
# 单元格 3: 共用函数
# ------------------------------------

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}")
```

```python
        return pd.DataFrame()

def get_aggregated_data(df):
    """
        将原始 DataFrame 按机构聚合，构建高校画像 (用于任务2).
        """
    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.

```

python

In [4]:

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

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

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
```

```
14 1        CHINESE ACADEMY OF AGRICULTURAL SCIENCES  CHINA MAINLAND      12222
15 2  UNITED STATES DEPARTMENT OF AGRICULTURE (USDA)             USA      12564
16 3                   CHINA AGRICULTURAL UNIVERSITY  CHINA MAINLAND      10052
17 4                                           INRAE          FRANCE       9314
18
19     cites  cites_per_paper  top_papers
20 0  332254            21.22         251
21 1  223855            18.32         198
22 2  220644            17.56         105
23 3  207779            20.67         166
24 4  187838            20.17         118
25
```

II.2 深度学习模型构建与训练

(1) 对目标 y (rank) 进行 `np.log()` 变换，使分布更平滑，便于神经网络学习.

(2) 对数值特征使用 `StandardScaler` (因为是对原始数据，不是聚合数据，`StandardScaler` 在这里是合适的).

(3) 对类别特征使用 `OneHotEncoder`.

(4) 对数据集进行打乱后，按照 80% / 20% 划分训练集和测试集，然后进行分层抽样，尽可能让训练集与测试集同分布.

In [11]:

python

```python
# ------------------------------------
# 单元格 5 (改进版): 任务1 - 数据准备 (随机打乱 + 分层抽样)
# ------------------------------------
print("--- 开始执行任务1 (改进版)：深度学习排名预测模型 ---")

# 导入用于拆分的库
from sklearn.model_selection import train_test_split

# 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. [关键改进]：随机打乱拆分 (80% 训练, 20% 测试)
print("步骤 2/4: 正在进行随机打乱 (Shuffle) 和分层抽样 (Stratify)...")

# 我们使用 train_test_split 来随机拆分数据
# test_size=0.2 表示 20% 的数据作为测试集
# stratify=X['research_field'] 确保每个学科在训练集和测试集中的比例与原始数据一致
```

```python
X_train, X_test, y_train_log, y_test_log = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42, # 确保每次运行结果一致
    stratify=X['research_field'] # 按学科分层
)

# 保留原始 y_test 以便评估
# 我们从 y_test_log 的索引中找到原始的 rank 值
y_test_orig = df_model.loc[y_test_log.index][target]

print(f"数据随机拆分完成. ")
print(f"  训练集大小: {len(X_train)}")
print(f"  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 (StandardScaler + OneHotEncoder)...")
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])

# 5. 应用预处理
# [重要]：预处理器 (preprocessor) 必须在训练集 (X_train) 上 'fit'
# 然后用它来 'transform' 训练集和测试集
print("步骤 4/4: 正在对训练集 'fit' 预处理器并 'transform' 数据...")
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("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 (改进版)：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: 正在进行随机打乱 (Shuffle) 和分层抽样 (Stratify)...
数据随机拆分完成.
  训练集大小: 27296
  测试集大小: 6825
步骤 3/4: 正在构建数据预处理 Pipeline (StandardScaler + OneHotEncoder)...
步骤 4/4: 正在对训练集 'fit' 预处理器并 'transform' 数据...
数据预处理完成.
处理后训练集特征维度: (27296, 26)

```

接下来，构建一个简单的多层感知机（MLP）.

In [12]:

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([
        layers.Input(shape=(input_shape,)),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.2),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.1),
        layers.Dense(32, activation='relu'),
        layers.Dense(1) # 输出层 (回归问题)
    ])

    model.compile(optimizer='adam',
                  loss='mean_squared_error', # 损失函数用 MSE
                  metrics=['mean_absolute_error'])

    return model

model = build_model(input_dim)
model.summary()
```

txt

```
[1mModel: "sequential_1"[0m
```

txt

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃[1m [0m[1mLayer (type)                        [0m[1m [0m┃[1m [0m[1mOutput Shape              [0m[1m [0m┃[1m [0m[1m      Param #[0m[1m [0m┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ dense_4 ([38;5;33mDense[0m)                 │ ([38;5;45mNone[0m, [38;5;34m128[0m)            │ [38;5;34m3,456[0m │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_2 ([38;5;33mDropout[0m)              │ ([38;5;45mNone[0m, [38;5;34m128[0m)            │ [38;5;34m0[0m │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_5 ([38;5;33mDense[0m)                 │ ([38;5;45mNone[0m, [38;5;34m64[0m)             │ [38;5;34m8,256[0m │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_3 ([38;5;33mDropout[0m)              │ ([38;5;45mNone[0m, [38;5;34m64[0m)             │ [38;5;34m0[0m │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_6 ([38;5;33mDense[0m)                 │ ([38;5;45mNone[0m, [38;5;34m32[0m)             │ [38;5;34m2,080[0m │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_7 ([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)
```

然后，训练模型.

In [ ]:

```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',
    patience=20,
    restore_best_weights=True
)

history = model.fit(
    X_train_processed,
    y_train_log,
    epochs=100, # 训练 100 轮，但 EarlyStopping 会提前停止
    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
)

print("模型训练完成. ")
```

```txt
开始训练深度学习模型...
Epoch 1/100
[1m683/683[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m7s[0m 10ms/step - loss: 7.6332 - mean_absolute_error: 1.2838 - val_loss: 0.8267 - val_mean_absolute_error: 0.6330
Epoch 2/100
[1m683/683[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m8s[0m 12ms/step - loss: 7.9564 - mean_absolute_error: 1.3544 - val_loss: 0.9225 - val_mean_absolute_error: 0.6038
Epoch 3/100
[1m683/683[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m8s[0m 11ms/step - loss: 18.5334 - mean_absolute_error: 1.6748 - val_loss: 1.7577 - val_mean_absolute_error: 0.4860
Epoch 4/100
[1m683/683[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m7s[0m 10ms/step - loss: 6.4972 - mean_absolute_error: 1.2122 - val_loss: 94.4050 - val_mean_absolute_error: 5.1616
Epoch 5/100
[1m683/683[0m [32m━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m7s[0m 10ms/step - loss: 9.2174 - mean_absolute_error: 1.4926 - val_loss: 2.5527 - val_mean_absolute_error: 1.1138
Epoch 6/100
```

```
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 47.1111 - mean_absolute_error: 2.8368 - val_loss: 32.2156 - val_mean_absolute_error: 2.9551
Epoch 7/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 259.1045 - mean_absolute_error: 5.4519 - val_loss: 1879.7865 - val_mean_absolute_error: 24.9025
Epoch 8/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 146.7566 - mean_absolute_error: 4.9934 - val_loss: 15.3566 - val_mean_absolute_error: 1.5577
Epoch 9/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 86.7602 - mean_absolute_error: 3.6166 - val_loss: 41.0193 - val_mean_absolute_error: 4.1226
Epoch 10/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 75.1320 - mean_absolute_error: 3.8771 - val_loss: 1.0988 - val_mean_absolute_error: 0.5494
Epoch 11/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m8s [0m 11ms/step - loss: 637.5482 - mean_absolute_error: 11.4197 - val_loss: 1.5791 - val_mean_absolute_error: 0.5270
Epoch 12/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 320.4568 - mean_absolute_error: 7.4376 - val_loss: 422.7041 - val_mean_absolute_error: 12.0179
Epoch 13/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 1049.6307 - mean_absolute_error: 11.7316 - val_loss: 253.2596 - val_mean_absolute_error: 9.1932
Epoch 14/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 141.0644 - mean_absolute_error: 5.6581 - val_loss: 78.8994 - val_mean_absolute_error: 5.4887
Epoch 15/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 615.9055 - mean_absolute_error: 8.5492 - val_loss: 295.0187 - val_mean_absolute_error: 11.5909
Epoch 16/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 2001.1785 - mean_absolute_error: 18.1557 - val_loss: 29.8246 - val_mean_absolute_error: 1.5173
Epoch 17/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 1013.9137 - mean_absolute_error: 12.9888 - val_loss: 1.5507 - val_mean_absolute_error: 0.8564
Epoch 18/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 463.5229 - mean_absolute_error: 7.6006 - val_loss: 9795.3145 - val_mean_absolute_error: 59.4567
Epoch 19/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m [1m7s [0m 10ms/step - loss: 3688.2117 - mean_absolute_error: 15.6137 - val_loss: 752.8276 - val_mean_absolute_error: 10.5576
Epoch 20/100
```

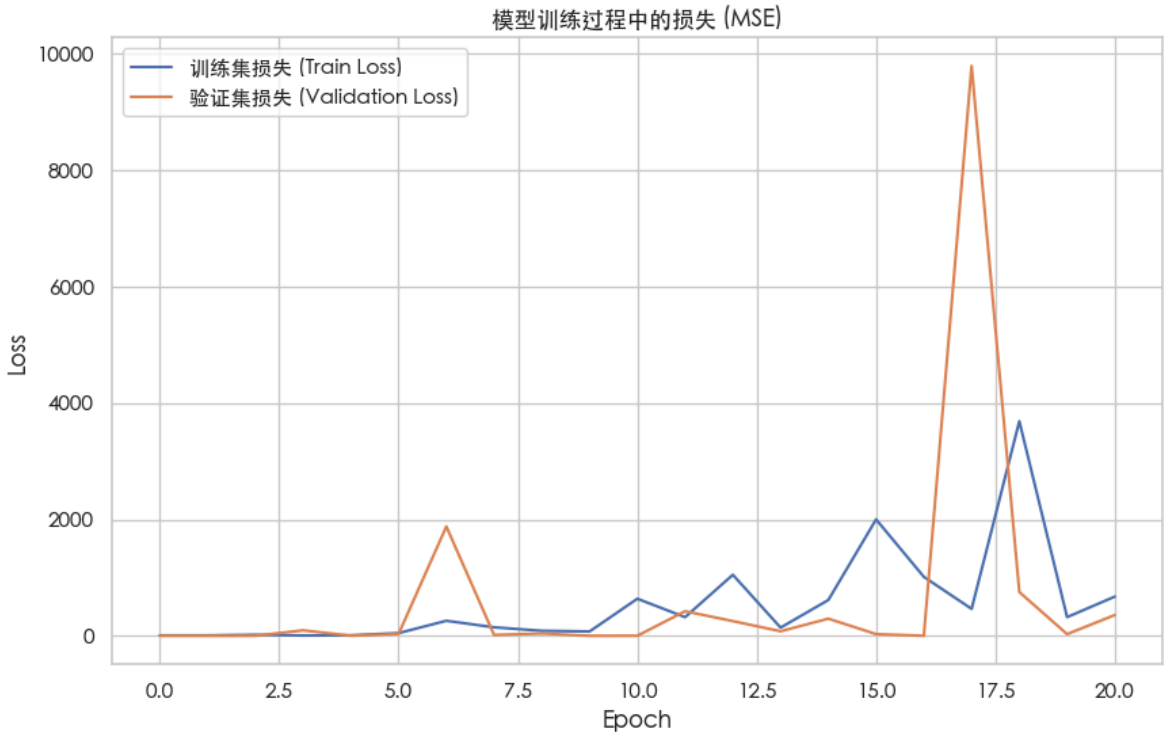
```
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m[0m [1m7s [0m 10ms/step - loss: 321.6691 - mean_absolute_error: 7.3027 - val_loss: 29.0872 - val_mean_absolute_error: 1.7844
Epoch 21/100
[1m683/683 [0m [32m━━━━━━━━━━━━━━━━━━━━[0m [37m[0m [1m7s [0m 10ms/step - loss: 674.6478 - mean_absolute_error: 8.8608 - val_loss: 357.7590 - val_mean_absolute_error: 10.0191
模型训练完成.
```

- 按作业要求，我们使用 `mean_squared_error` 和 `mean_absolute_percentage_error` 进行评估.
- 我们必须先用 `np.exp()` 将模型的预测值（log(rank)）还原回 rank，才能与 `y_test_orig` (原始排名) 进行比较.

In [21]:

```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
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')
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()

# 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 (按作业要求)
mse = mean_squared_error(y_test_orig, y_pred_orig)
mape = mean_absolute_percentage_error(y_test_orig, y_pred_orig)
rmse = np.sqrt(mse) # RMSE 只是 MSE 的平方根，更具解释性

print("\n--- 任务1 结果：深度学习模型评估 (测试集) ---")
print(f"MSE (均方误差): {mse:.4f}")
print(f"MAPE (平均绝对百分比误差): {mape:.4f} (预测平均偏离 {mape*100:.2f}%)")
print(f"RMSE (均方根误差): {rmse:.4f} (预测排名平均误差约 {rmse:.0f} 名)")


# 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

```

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

--- 任务1 结果：深度学习模型评估（测试集） ---
MSE (均方误差): 47523276.0000
MAPE (平均绝对百分比误差): 1.6086 (预测平均偏离 160.86%)
RMSE (均方根误差): 6893.7128 (预测排名平均误差约 6894 名)

--- 预测结果抽样对比（前20条） ---
    真实排名 (Actual Rank)  预测排名 (Predicted Rank)
0                   924           1034.654663
1                  1385           1088.666870
2                   948            961.737000
3                    32            439.364777
4                   292            203.373352
5                  2529           4309.211426
6                  1776           1569.565186
7                  2100           1185.154175
8                  1644           1641.129639
9                  2897           4149.728516
10                  358            685.589905
11                  328            635.259094
12                 1048           1062.924683
13                 1216           3585.914062
14                 1646            925.419312
15                  793           1039.298706
16                 4625           4410.215332
```

```
29 17                1002             664.867004
30 18                  36             233.596024
31 19                 310             435.206207
32
```

> **Remark**
>
> 完整代码见 `./hw7.ipynb`.