Python Movielens 大作业报告

——基于观影数据集的数据分析与挖掘

沈千帆 2200013220

2024 年 5 月 5 日

所有代码和结果都在2200013220.ipynb中按照部分标注好。

1 传统偏好分析

在这部分中，主要参考 demo 中所给方式，对于不同年龄段的观影偏好进行分析。

1.1 合理衡量偏好程度的指标并展示

这里采用 $R_a = WR + (1-W)R_0$ 作为流行统计量，先对所有电影进行筛选。

```
R = ratings.groupby('movie_id')['rating'].mean()
N = ratings.groupby('movie_id')['rating'].count()
popula r= pd.DataFrame({'rating':R, 'count':N})
W = popular['count'].apply(lambda x: max(0.5*x/popular['count'].mean(), 1))
popular['rating'] = W * popular['rating'] + ( 1 - W ) * popular['rating'].mean()
popular = popular.sort_values(by='rating', ascending=False)
popular_percentile = popular['rating'].quantile(0.9)
popular = popular[popular['rating'] >= popular_percentile]
```

然后我们确定有哪些不同的年龄段。

```
unique_age_descriptions = age_users['age_desc'].unique()
```

```
['Under 18' '56+' '25-34' '45-49' '50-55' '35-44' '18-24']
```

我的思路是与 demo 类似，对于每一类，和其他所有类的均分做差值，得分越高说明偏好越强，并且乘上之前的流行统计量作为权重，由此排名得到每个年龄段偏好的电影。（此处以 18 岁以下为例）

```
Under_18_users = age_users[age_users['age_desc'] == 'Under 18']
else_users = age_users[age_users['age_desc'] != 'Under 18']
```

```
Under_18_users_mean = Under_18_users.groupby('movie_id')['rating'].mean()
else_users_mean = else_users.groupby('movie_id')['rating'].mean()

score = Under_18_users_mean - else_users_mean
score = pd.merge(score, popular, on='movie_id', how='outer')
print(score)
score['score'] = score['rating_x'] * score['rating_y']
Under_18_preference = score.nlargest(10, 'score').reset_index()
```

ranking	title	genres
1	GoodFellas (1990)	Crime|Drama
2	Aparajito (1956)	Drama
3	Trust (1990)	Comedy|Drama
4	Pawnbroker, The (1965)	Drama
5	Paradise Lost: The Child Murders at Robin Hood...	Documentary
6	Thin Blue Line, The (1988)	Documentary
7	Seven Days in May (1964)	Thriller
8	Palm Beach Story, The (1942)	Comedy
9	Central Station (Central do Brasil) (1998)	Drama
10	Celebration, The (Festen) (1998)	Drama

图 1: 18 岁以下最喜欢的 10 个电影

1.2 基于电影风格的可视化

这部分也是按照 demo 的方式使用在不同年龄上，此处之展现结果。

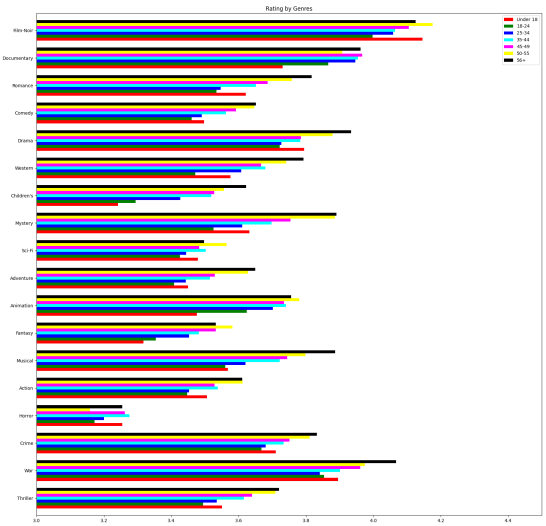

图 2: 不同年龄在不同风格电影中的评分均值对比

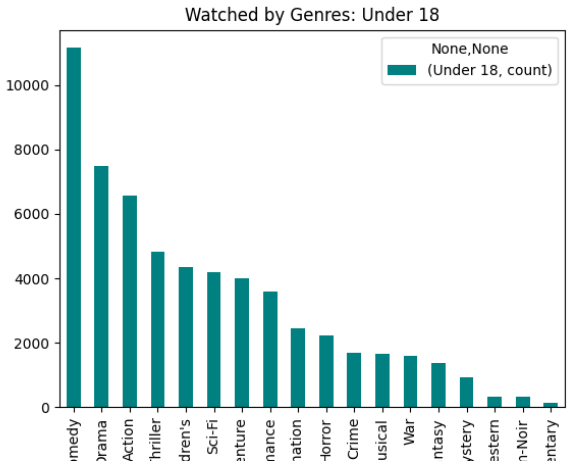

图 3: 不同年龄观看不同风格电影的数量对比

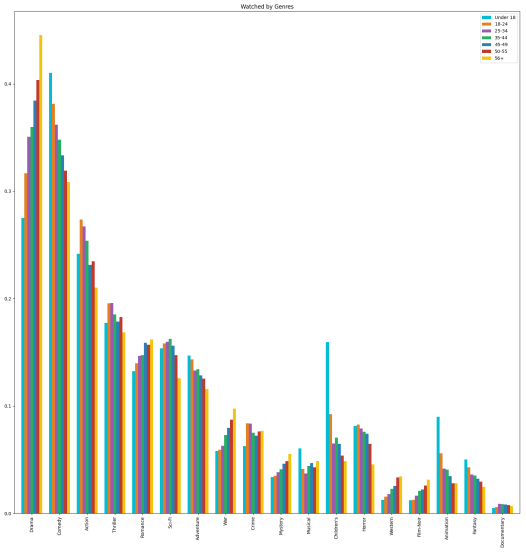

图 4: 不同年龄在不同风格电影中的比例对比

2 用户评分预测

2.1 特征工程

2.1.1 对于性别，年龄，职业，电影类型分别进行处理

对于用户的性别，年龄，职业直接用 One-Hot Encoding

```
encoder = OneHotEncoder()
user_features = encoder.fit_transform(users[['gender', 'age_desc', 'occ_desc']])
user_features_df = pd.DataFrame(user_features.toarray(),
columns=encoder.get_feature_names_out(['gender', 'age_desc', 'occ_desc']))
```

对于电影类型，要先进行分词，再用 MultiLabelBinarizer 进行 One-Hot Encoding

```
movies['genres'] = movies['genres'].apply(lambda x: x.split('|'))
mlb = MultiLabelBinarizer()
movie_genres_encoded = mlb.fit_transform(movies['genres'])
movie_genres_df = pd.DataFrame(movie_genres_encoded, columns=mlb.classes_)
movie_genres_df['movie_id'] = movies['movie_id'].values
```

2.1.2 降维

使用标准化 + 累计方差 PCA 进行降维（在之前先将所有特征进行融合，并把 ratings 这一列去掉以防作弊）。

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(user_ratings_df)
X_scaled_df = pd.DataFrame(X_scaled, columns=user_ratings_df.columns,
                           index=user_ratings_df.index)

pca = PCA()
pca.fit(X_scaled)
cumulative_variance = pca.explained_variance_ratio_.cumsum()
# 选择达到 90% 累计方差的组件数量
n_components = (cumulative_variance > 0.9).argmax() + 1
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_scaled)
```

```
Number of components to keep for 90% variance: 260
```

2.2 模型选择与训练

划分训练集和测试集，由于选用了 lightgbm 模型，因此还额外在训练集中划分了验证集。

```
X_train, X_test, y_train, y_test = train_test_split(user_ratings_df,
ratings['rating'], test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,
random_state=42)
```

选用 lightgbm 模型进行训练。

```
params = {
    'boosting_type': 'gbdt',
    'objective': 'regression',
    'metric': 'rmse',
    'num_leaves': 400,
    'learning_rate': 0.05,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose': 0
}
gbm = lgb.train(params,
                train_data,
                num_boost_round=750,
                valid_sets=[val_data],
                callbacks=[lgb.callback.early_stopping(stopping_rounds=20)])
```

2.3 预测与评估

2.3.1 自己的 MSE 函数

```
def my_mean_squared_error(y_true, y_pred):
    # 计算 y_true 和 y_pred 之间的差异
    differences = y_true - y_pred
    # 计算差异的平方
    squared_differences = np.square(differences)
    # 计算均方误差
    mse = np.mean(squared_differences)

    return mse
```

2.3.2 预测与评估

在经过不同的特征选取，融合和处理之后得到的最高 MSE 为 0.892，具体的特征选取在之后会提到。

```
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration)

mse = my_mean_squared_error(y_test, y_pred)
```

2.4 高等级分数

- 加入了 userid，movieid 两个特征，有明显的提高；
- 加入了电影简介的特征，利用 td-idf 模型进行向量化加入了特征，这个相比原来有较大提高，不加基本都在 0.98-0.99 左右；
- 尝试了随机森林和 lightgbm 模型对于降维和没降维的数据分别进行训练、预测。随机森林只能达到 1.0 左右。

```
#对于电影简介应用TF-IDF
movies_info['intro'] = movies_info['intro'].fillna('')
tfidf_vectorizer = TfidfVectorizer()
movie_synopses_tfidf = tfidf_vectorizer.fit_transform(movies_info['intro'])
svd = TruncatedSVD(n_components=300)
reduced_tfidf_matrix = svd.fit_transform(movie_synopses_tfidf)
print(reduced_tfidf_matrix.shape)
movie_synopses_df = pd.DataFrame(reduced_tfidf_matrix,
columns=[f'component_{i}' for i in range(300)])
movie_synopses_df['movie_id'] = movies_info['movie_id'].values
```

- 除此之外，在一开始对于 ratings 进行数据清洗，删去那些打分的 z-score 非常大，也就是异常评分的数据，收获了特别好的效果，降到 0.5 左右。

```
# 计算每部电影的平均分和标准差
rating_mean = ratings.groupby('movie_id')['rating'].transform('mean')
rating_std = ratings.groupby('movie_id')['rating'].transform('std')
ratings['z_score'] = (ratings['rating'] - rating_mean) / rating_std
# 筛选出异常打分，我们这里以 z-score 绝对值大于1.5作为判定标准，并删除异常评分
outliers = ratings[ratings['z_score'].abs() > 1.5]
ratings = ratings.drop(outliers.index)
```

```
MSE: 0.5269190152975602
```

3 海报按内容聚类

3.1 图像特征提取

对于颜色直方图和灰度直方图分别进行了提取，但并没有直接 concat，因为效果非常不好，主要还是利用 imgvec 提取的特征。

```
# 特征向量
vector = img2vec_model.get_vec(image)
image = np.array(image)
# 颜色直方图特征
hist_list_r.append(cv2.calcHist([image], [0], None, [256], [0, 256]))
hist_list_g.append(cv2.calcHist([image], [1], None, [256], [0, 256]))
hist_list_b.append(cv2.calcHist([image], [2], None, [256], [0, 256]))
# 转换为灰度图并计算灰度直方图
gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
hist_list_gray.append(cv2.calcHist([gray_image], [0], None, [256], [0, 256]))
# 添加图像文件前缀（电影ID）和特征向量到特征列表中
features.append([filename.split('.')[0], vector, hist_list_r, hist_list_g,
hist_list_b, hist_list_gray])
```

3.2 降维

3.2.1 使用累计 PCA

```
feature_vectors = np.array(df['features'].to_list())
scaler = StandardScaler()
features_scaled = scaler.fit_transform(feature_vectors)
pca = PCA()
pca.fit(features_scaled)
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
# 选择累积方差达到90%的主成分数量
num_components = (cumulative_variance > 0.9).argmax() + 1
pca = PCA(n_components=num_components)
features_reduced = pca.fit_transform(features_scaled)
```

```
Reduced feature dimension: 194
```

3.2.2 t-SNE 降维

```
tsne = TSNE(n_components=2, verbose=1, perplexity=25, n_iter=5000)
tsne_results = tsne.fit_transform(feature_vectors)
```

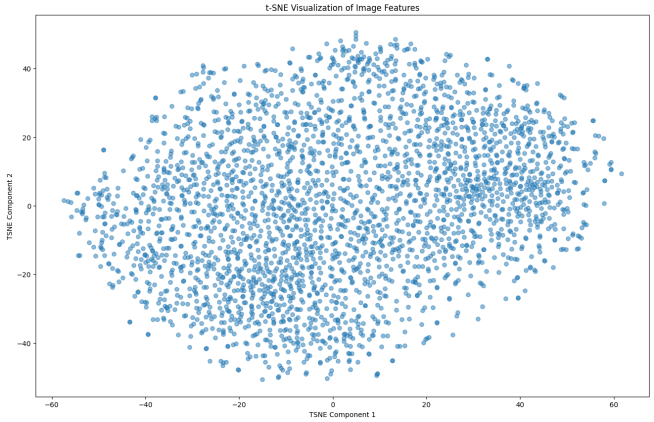

图 5: t-SNE results

3.3 无监督聚类分析

应用 k-means 等算法对电影海报进行无监督聚类，对于每个类别的中心点，选取最近的 5 个海报进行展示。

```
k = 8
kmeans = KMeans(n_clusters=k, random_state=42)
# 拟合模型
kmeans.fit(features_reduced)
# 获取聚类标签
cluster_labels = kmeans.labels_
cluster_centers = kmeans.cluster_centers_
distances = np.linalg.norm(features_reduced - cluster_centers[cluster_labels], axis=1)
representative_images = []
for i in range(len(cluster_centers)):
    cluster_indices = np.where(cluster_labels == i)[0]
    sorted_indices = np.argsort(distances[cluster_indices])[:5]
    representative_images.append(cluster_indices[sorted_indices])
```

3.4 有监督聚类分析

这里的特征将电影简介，海报 imgvec 所得到的特征和降维之后的特征进行了融合。

```
# 提取特征和标签
X = np.array(merged_df['features'].tolist())
X = np.hstack((X,info_array))
print(X.shape)
y = np.array(merged_df['genre_vector'].tolist())

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

选择 lightgbm 的多标签分类算法进行训练。

```
from lightgbm import LGBMClassifier
clf = LGBMClassifier(
    num_leaves=63,
    learning_rate=0.05,
    n_estimators=100
)
multi_output_clf = MultiOutputClassifier(clf)
multi_output_clf.fit(X_train, y_train)
y_pred = multi_output_clf.predict(X_test)
```

```
Average Accuracy: 0.9120370370370369
Accuracy: 0.1445578231292517
```

3.5 高等级分数

- 尝试将无监督聚类所得到的结果进行了 one-hot encoding，融合在之前用户评分预测的特征中，达到了前文中最好的结果（但提升的不多）；
- 将提取到的海报特征融入评分预测的特征，但效果不好，原因我认为是虽然经过处理，但是该特征无法提供明显的区分性，同时由于维度较大，反而有所影响。

```
clusters_df = pd.DataFrame({
    'movie_id': movie_ids,
    'cluster_label': kmeans.labels_
})
user_ratings_df = pd.merge(user_ratings_df, clusters_df, on='movie_id', how='left')
# 用-1表示无聚类信息
user_ratings_df['cluster_label'] = user_ratings_df['cluster_label'].fillna(-1)
user_ratings_df = pd.get_dummies(user_ratings_df, columns=['cluster_label'])
```