推荐算法

Item-Item Recommender

结构

数据分析——>构建数据(用户->电影)——>不同df直接用索引map——>计算相似度——>根据相似度使用不同的推荐算法

example

data

rating

userId	movieId	rating	timestamp

movies

movieId	title	genres

关注每个用户的评分数，使用 pandas 的 groupby() 和 count()，按用户 ID 对数据进行分组，合并相同ID并计算每个用户 ID 的评分数。

```
1 user_freq = ratings[['userId',
  'movieId']].groupby('userId').count().reset_index()
2 #此时列名需要重新定义
3 user_freq.columns = ['userId', 'n_ratings']
```

此处按movieId分类后选rating列计算平均，得到的新的df只有两列，movieId和rating(average)

```
1 mean_rating = ratings.groupby('movieId')[['rating']].mean()
```

在选取平均最受欢迎的电影时发现会由于样本过少而出现分数不正常，因此使用baye‘s平均代替平均数

再按照movieId将movie_stats和movies的title列合并

```
movie_stats = movie_stats.merge(movies[['movieId', 'title']])#会自动对应
movie_stats.sort_values('bayesian_avg', ascending=False).head()#ascending=False 说明按高到低排序
```

协同过滤（Collaborative Filtering）

协同过滤的核心思想是，如果一个用户A在某方面（如电影、音乐、书籍等）的喜好与另一个用户B相似，那么A用户可能会喜欢B用户喜欢的其他东西，反之亦然。

协同过滤的主要类型

1. 用户基于协同过滤（**User-Based** Collaborative Filtering）：
    - 在这种方法中，系统会寻找与目标用户相似的用户群体，然后根据这个群体的喜好向目标用户推荐物品。
    - 相似性通常基于评分模式的比较，例如使用皮尔逊相关系数或余弦相似性。
2. 物品基于协同过滤（**Item-Based** Collaborative Filtering）：
    - 这种方法关注于物品之间的相似性，而不是用户之间的相似性。
    - 系统会识别用户已经评价过的物品，然后找出与这些物品相似的其他物品来进行推荐。

Example

定义一个create_X() 函数会输出一个包含四个映射字典的稀疏矩阵 X：

user_mapper：将用户 ID 映射到用户索引

movie_mapper：将电影 ID 映射到电影索引

user_inv_mapper：将用户索引映射到用户 ID

movie_inv_mapper：映射电影索引到电影 ID

我们需要这些字典，因为它们分别映射了实用程序矩阵中哪一行和哪一列对应哪一个用户 ID 和哪一部电影 ID。

X（行:用户，列:电影）矩阵是一个 scipy.sparse.csr_matrix，可以稀疏地存储数据

```
def create_X(df):
    """
    Generates a sparse matrix from ratings dataframe.

    Args:
        df: pandas dataframe

    Returns:
        X: sparse matrix
        user_mapper: dict that maps user id's to user indices
```

```
        user_inv_mapper: dict that maps user indices to user id's
        movie_mapper: dict that maps movie id's to movie indices
        movie_inv_mapper: dict that maps movie indices to movie id's
    """
    N = df['userId'].nunique()
    M = df['movieId'].nunique()

    user_mapper = dict(zip(np.unique(df["userId"]), list(range(N))))
    movie_mapper = dict(zip(np.unique(df["movieId"]), list(range(M))))

    user_inv_mapper = dict(zip(list(range(N)), np.unique(df["userId"])))
    movie_inv_mapper = dict(zip(list(range(M)), np.unique(df["movieId"])))

    user_index = [user_mapper[i] for i in df['userId']]
    movie_index = [movie_mapper[i] for i in df['movieId']]

    X = csr_matrix((df["rating"], (movie_index, user_index)), shape=(M, N))

    return X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper
```

Tips

用下面的代码可以查看稀疏矩阵的稀疏性，并将矩阵写入文件

```
sparsity = X.count_nonzero()/(X.shape[0]*X.shape[1])
#保存文件
from scipy.sparse import save_npz
save_npz('data/user_item_matrix.npz', X)
```

KNN找到item的k部相似电影

```
from sklearn.neighbors import NearestNeighbors

def find_similar_movies(movie_id, X, k, metric='cosine', show_distance=False):
    """
    Finds k-nearest neighbours for a given movie id.

    Args:
        movie_id: id of the movie of interest
        X: user-item utility matrix
        k: number of similar movies to retrieve
        metric: distance metric for kNN calculations

    Returns:
```

```
        list of k similar movie ID's
    """
    neighbour_ids = []

    movie_ind = movie_mapper[movie_id]
    movie_vec = X[movie_ind]
    k+=1
    kNN = NearestNeighbors(n_neighbors=k, algorithm="brute", metric=metric)
    kNN.fit(X)
    if isinstance(movie_vec, (np.ndarray)):
        movie_vec = movie_vec.reshape(1,-1)
    neighbour = kNN.kneighbors(movie_vec, return_distance=show_distance)
    for i in range(0,k):
        n = neighbour.item(i)
        neighbour_ids.append(movie_inv_mapper[n])
    neighbour_ids.pop(0)
    return neighbour_ids
```

此处相似度计算base的是rating向量

冷启动问题

协同过滤法完全依赖于实用工具矩阵中用户与项目之间的交互。这种方法的问题在于，没有互动的全新用户或项目会被排除在推荐系统之外。

为了解决这一问题。可以使用基于内容的过滤，它根据用户和物品的特征生成推荐。

清理数据集

观察到电影标题后有形如《XXX》(年份）,因此为了分析时间信息，可用正则表达式 将年份单独提取出来然后将genres列用split转为list（注;x.split("|")意为将字符串x，根据|来分开，返回一个list）

创建字典：genres频率计数

服务：most-common()方法找到电影数量最多的类型

```
import re
def extract_year_from_title(title):
    t = title.split(' ')
    year = None
    if re.search(r'\(\d+\)', t[-1]):
        year = t[-1].strip('()')
        year = int(year)
    return year
```

计算感兴趣的电影间的余弦相似度矩阵。(nmovies,nmovies).

基于字符的搜索器（fuzzywuzzy)

```
from fuzzywuzzy import process

def movie_finder(title):
    all_titles = movies['title'].tolist()
    closest_match = process.extractOne(title,all_titles)
    return closest_match[0]
```

隐式反馈推荐法

隐式反馈会根据用户对物品的行为来推测其偏好。以 Netflix 为例。如果你狂看一部剧，并在一周内看完所有季，那么你很有可能喜欢这部剧。但是，如果你开始看一部剧集，却在第一集看到一半时就停止了，那么就有理由怀疑你可能不喜欢这部剧集。

Implicit package可以帮助我们发现用户与电影之间互动的潜在特征。这些潜在特征可以更简洁地表示用户口味和项目描述。虽然我们无法解释隐空间的每个特征维度是什么，但是确实可以捕捉到相关性。

此处并没有使用神经网络训练求解，而是采用ALS直接求解方程