大数据原理与技术 Lab7

20337025 崔璨明

1、实验要求

- 实现Latent factor model和Collaborative filtering model
- 找公开的MovieLens任何一个数据集(https://grouplens.org/datasets/movielens/)进行测试（随机划分80%训练、20%测试）
- 对比两个算法的RMSE

2、实验原理

1. Latent Factor Model（潜在因子模型）：
    - 潜在因子模型是基于矩阵分解的思想，它假设用户的评分是由潜在因子（latent factors）决定的。
    - 潜在因子可以理解为描述用户和物品的特征或属性，例如在电影推荐中，潜在因子可以表示电影的类型（如喜剧、动作、恐怖等）和用户的偏好（如喜欢悬疑、不喜欢浪漫等）。
    - 通过对用户-物品评分矩阵进行矩阵分解，将其分解为用户矩阵和物品矩阵的乘积形式，其中用户矩阵表示用户与潜在因子的关系，物品矩阵表示物品与潜在因子的关系。
    - 通过学习得到用户矩阵和物品矩阵，可以对用户未评分的物品进行预测，从而实现推荐。
2. Collaborative Filtering Model（协同过滤模型）：
    - 协同过滤模型是基于用户行为数据的相似性来进行推荐的方法。
    - 它假设用户对物品的偏好与与其具有相似兴趣的其他用户或物品的评分有关。
    - 协同过滤模型可以分为两种类型：基于用户的协同过滤和基于物品的协同过滤。基于用户的协同过滤通过计算用户之间的相似度，将相似用户的评分进行加权平均，预测目标用户对未评分物品的偏好。而基于物品的协同过滤通过计算物品之间的相似度，将目标用户对已评分物品的评分与相似物品的评分进行加权平均，预测目标用户对未评分物品的偏好。

3、实验过程

采用的数据集名为`ml-100k`，是由明尼苏达大学的GroupLens研究项目收集的MovieLens数据集。

Latent Factor Model实现如下，首先读取数据，设置用户和物品的数量，并计算训练样本的数量。接着初始化模型参数，包括隐因子数量`k`等。创建了用于快速查询的字典`item_user_dict`和`user_item_dict`，以避免生成大型稀疏矩阵。模型中通过随机梯度下降算法更新模型参数，计算误差并打印每个epoch的训练RMSE。最后在测试集上进行测试，详细描述见代码注释：

```
# 读取训练集和测试集数据
train_data = pd.read_table('ml-100k/u1.base', sep='\t', header=None).iloc[:,
:3].values
test_data = pd.read_table('ml-100k/u1.test', sep='\t', header=None).iloc[:,
:3].values
# 用户数量和物品数量（索引从1开始）
users_num, items_num = 944, 1683
# 样本数量
samples_num = train_data.shape[0]
print(train_data.shape, test_data.shape)
# 隐因子的数量
```

```
k = 20
# 全局均值
mean_arr = np.mean(train_data[:, 2])
# 物品和用户的偏置项
b_item = np.random.randn(items_num)
b_user = np.random.randn(users_num)
# 物品和用户的隐因子矩阵
qi = np.random.randn(items_num, k)
pu = np.random.randn(users_num, k)
# 用于快速查询物品和用户的字典
item_to_user = dict()
user_to_item = dict()
# 训练
max_iteration = 50  # 最大迭代次数
learning_rate = 0.01  # 学习率
alpha = 0.1  # 正则化项系数
for epoch in range(max_iteration):
    MSE = 0
    index = np.random.permutation(samples_num)
    for idx in index:
        user_id, item_id, rating = train_data[idx]
        # 预测评分
        y_pred = mean_arr + b_item[item_id] + b_user[user_id] +
np.dot(pu[user_id], qi[item_id].T)
        err = rating - y_pred
        MSE += err**2
        # 更新偏置项和隐因子矩阵
        b_user[user_id] += learning_rate * (err - alpha * b_user[user_id])
        b_item[item_id] += learning_rate * (err - alpha * b_item[item_id])
        temp = qi[item_id]
        qi[item_id] += learning_rate * (err * pu[user_id] - alpha * qi[item_id])
        pu[user_id] += learning_rate * (err * temp - alpha * pu[user_id])
    MSE /= samples_num
    RMSE = math.sqrt(MSE)
    print("epoch:", epoch+1, "train_data RMSE:", RMSE)
Y_pred = list()
test_data_mse = 0
# 计算rmse
for sample in test_data:
    user_id, item_id, rating = sample
    y_pred = mean_arr + b_item[item_id] + b_user[user_id] + np.dot(pu[user_id],
qi[item_id].T)
    test_data_mse += (rating - y_pred)**2
test_data_mse /= len(test_data)
test_data_rmse = math.sqrt(test_data_mse)
print("test_data RMSE: ", test_data_rmse)
```

Collaborative filtering model实现如下，首先读取训练集和测试集的数据，定义相关变量。然后，计算训练集中所有用户对物品的评分，创建用户物品评分矩阵。预测函数`predict`基于用户和物品的均值来预测用户对物品的评分。

测试时，遍历测试集中的每个用户物品对，获取实际评分和预测评分，并计算它们之间的平方误差。最后，将所有平方误差相加并计算均方根误差RMSE，将结果打印出来。具体代码：

```
import pandas as pd
```

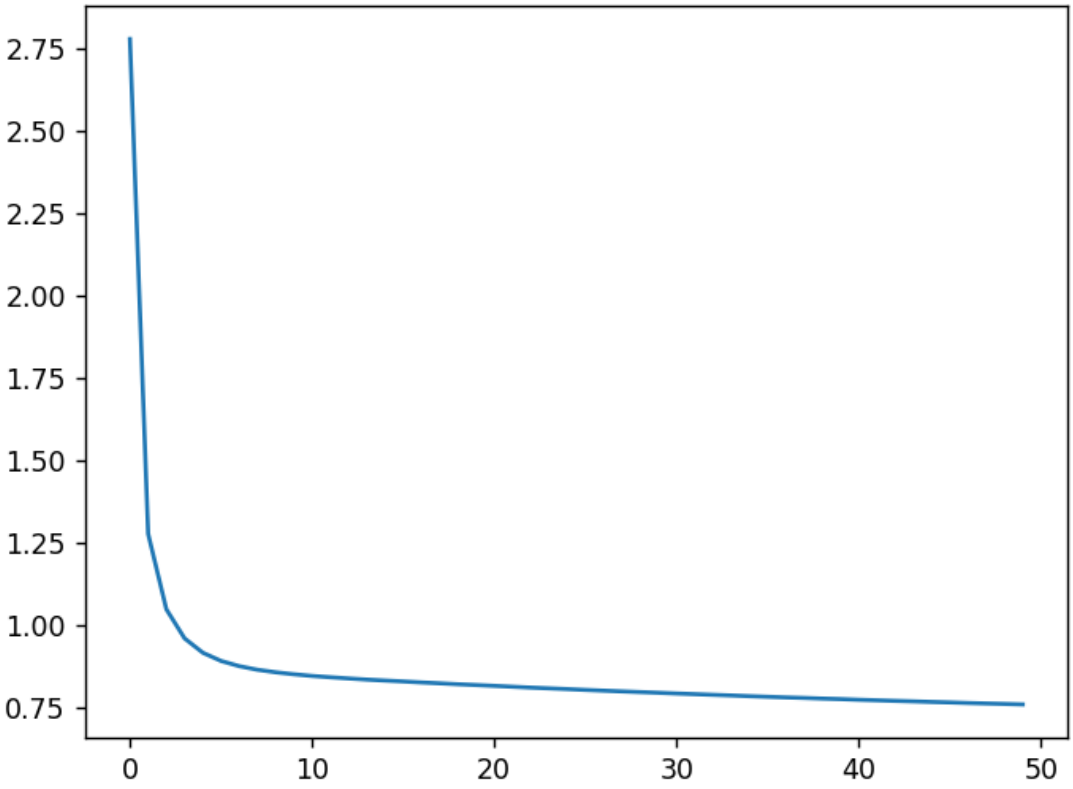
```
import math
import numpy as np
# 读取训练集和测试集数据
train = pd.read_table('ml-100k/u1.base', sep='\t', header=None).iloc[:,
:3].values
test = pd.read_table('ml-100k/u1.test', sep='\t', header=None).iloc[:,
:3].values
n_users, n_items = 943+1, 1682+1    # 数据idx从1开始
n_samples = train.shape[0]
print(train.shape, test.shape)
glob_mean = np.mean(train[:, 2])    # 全局均分
# 创建用户物品评分矩阵
rating_matrix = np.zeros((n_users, n_items))
for i in range(n_samples):
    user = train[i, 0]
    item = train[i, 1]
    rating = train[i, 2]
    rating_matrix[user, item] = rating
# 预测函数
def predict(user, item):
    user_mean = np.mean(rating_matrix[user, :])
    item_mean = np.mean(rating_matrix[:, item])
    if user_mean == 0:
        user_mean = glob_mean
    if item_mean == 0:
        item_mean = glob_mean
    return (user_mean + item_mean) / 2
# 计算RMSE
sum_squared_error = 0
for i in range(test.shape[0]):
    user = test[i, 0]
    item = test[i, 1]
    actual_rating = test[i, 2]
    predicted_rating = predict(user, item)
    squared_error = (predicted_rating - actual_rating) ** 2
    sum_squared_error += squared_error
rmse = math.sqrt(sum_squared_error / test.shape[0])
print("RMSE:", rmse)
```

4、实验结果

在测试集上进行测试，Latent factor model的RMSE为0.7427（max_iteration = 50）：

```
epoch: 42 train_data RMSE: 0.7694401744032837
epoch: 43 train_data RMSE: 0.7676365354805244
epoch: 44 train_data RMSE: 0.7661922138008089
epoch: 45 train_data RMSE: 0.764340841515364
epoch: 46 train_data RMSE: 0.7627133928734711
epoch: 47 train_data RMSE: 0.7613147210734216
epoch: 48 train_data RMSE: 0.7595855411500633
epoch: 49 train_data RMSE: 0.7580469253317815
epoch: 50 train_data RMSE: 0.7564567362137481
test_data RMSE:  0.742785353588085
PS D:\大三下资料\大数据\lab7>
```

训练过程中RMSE的变化如下：



Collaborative filtering model的RMSE为3.342：

```
PS D:\大三下资料\大数据\lab7> & E:/anaconda/python.exe d:/大三下资料/大数据/lab7/CFM.py
(80000, 3) (20000, 3)
RMSE: 3.342060680672767
```

实验结果表明，在`ml-100k`这个数据集上，Latent factor model的表现要比Collaborative filtering model要好。