大数据分析技术报告（一）

姓名 金家耀

学号 1193210320

班级 人工智能 2103

学院 人工智能与计算机学院

指导老师 朱书伟

2023 年 10 月 1 日

目录

大数据分析技术报告（一）

1 实验要求

任选一个 (或多个）推荐系统算法，完成小报告，不限于下面提供的材料:

1. 面向电影推荐的一个算法实现, A system to recommend movies according toratings provided by users using Collaborative Filtering Learning A1gorithm. github 链接: https://github.com/thegenuinegourav/Movies-Recommender

2. 一个用于推荐系统的包，Surprise is a Python scikit.for building and analyzing recommender systems that deal with explicit rating data. github 链接: https://github.com/NicolasHug/Surprise

3. 关于推荐系统的重要论文合集 (Must-read papers on Recommender System)github 链接: https://github.com/hongleizhang/RSPapers#poi-recommender-system

4. 课本第 3 章。

2 协同过滤算法介绍

协同过滤算法是一类常用于推荐系统的算法，其主要思想是利用用户与项目（或物品）之间的交互行为（如用户的评分、点击、购买历史等）来预测用户对未来项目的喜好或兴趣。这些算法依赖于用户之间或项目之间的协同行为，因此称为协同过滤算法。

协同过滤算法主要分为两种类型:

1. 基于用户的协同过滤（User-Based Collaborative Filtering):

基于用户的协同过滤算法首先找到与目标用户具有相似兴趣的其他用户集合，这些用户通常被称为“邻居”。

然后，通过分析邻居用户对某个项目的评分或行为来预测目标用户对该项目的兴趣。基于用户的协同过滤算法的优点是简单易懂，但它可能受到用户冷启动问题和数据稀疏性问题的影响。

2. 基于物品的协同过滤（Item-Based Collaborative Filtering):

基于物品的协同过滤算法首先计算项目之间的相似性，通常使用项目的评分或行为数据计算相似性。然后，根据用户过去的行为和评分，预测用户对未评价项目的兴趣。

基于物品的协同过滤算法通常在数据稀疏性方面表现更好，因为物品比用户数量多，更容易找到相似的物品。

协同过滤算法的优点是能够提供个性化的推荐，因为它们考虑了用户和项目之间的直接交互信息。然而，它们也存在一些挑战，如数据稀疏性、冷启动问题（新用户或新项目的推荐困难）以及可伸缩性问题（当用户和项目数量庞大时，计算成本较高)。

2.1 计算相似性

2.1.1 杰卡德相关系数

$$J(A,B) = \frac{\|A \cap B\|}{\|A \cup B\|}$$

两个集合 A 和 B 交集元素的个数在 A、B 并集中所占的比例，称为这两个集合的杰卡德系数，用符号 J(A,B) 表示。杰卡德相似系数是衡量两个集合相似度的一种指标。

2.1.2 余弦相似度

$$\text{Cosine Similarity}(A,B) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

余弦相似度通过计算两个向量之间的夹角余弦值来计算物品之间的相似性，其中分子为两个向量的内积，即两个向量相同位置的数字相乘。

2.1.3 皮尔逊相关系数

$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}}$$

相比余弦相似度，皮尔逊相关系数通过使用用户平均分对各独立评分进行修正，减小了用户评分偏置的影响。

2.2 矩阵分解

矩阵分解是推荐系统中常用的技术之一，用于预测用户对项目的评分或偏好。这些方法通常尝试将用户-项目评分矩阵分解成两个或多个低维度的矩阵，以捕获潜在的用户和项目特征。以下是一些常见的矩阵分解方法：

2.2.1 Traditional SVD

矩阵分解第一个想到的一定是奇异值分解（SVD），其公式如下：

$$M = U\Sigma V^T$$

在推荐算法中，将分解之后的 U 作为用户特征，将 ΣV^T 作为项目特征，从而进行预测。

2.2.2 Funk SVD

传统的 SVD 矩阵分解需要将原始的 R 矩阵填充完整，再进行矩阵分解，同时存在着计算复杂度高的问题，在样本量较大时往往效果一般。于是 Funk SVD 就被发明。

$$\min \Sigma_{ij}(m_{ij} - q_j^T p_i)^2$$

利用线性回归的思想，最小化均方误差来拟合原始矩阵数据，从而实现寻找最优化的用户特征以及项目特征。同时为了降低过拟合，Funk SVD 又加入了 L2 正则化：

$$\min \Sigma_{ij}(m_{ij} - q_j^T p_i)^2 + \lambda(\|q_j\|_2^2 + \|p_i\|_2^2)$$

对于上述优化问题，我们通常采用梯度下降来进行求解。对上述式子分别对 p_i，q_j 求导：

$$\frac{\partial J}{\partial p_i} = -2(m_{ij} - q_j^T p_i) + 2\lambda p_i$$
$$\frac{\partial J}{\partial q_j} = -2(m_{ij} - q_j^T p_i) + 2\lambda q_j$$

则梯度下降的迭代公式为：

$$p_i = p_i + \alpha((m_{ij} - q_j^T p_i) - \lambda p_i)$$

$$q_j = q_j + \alpha((m_{ij} - q_j^T p_i) - \lambda q_j)$$

2.2.3 Bias SVD

$$\min \Sigma_{ij}(m_{ij} - \mu - b_i - b_j - q_j^T p_i)^2 + \lambda(\|q_j\|_2^2 + \|p_i\|_2^2 + \|b_i\|_2^2 + \|b_j\|_2^2)$$

该公式提出的依据是假设用户与项目都存在固有的属性与对方无关，每个人的评分标准不同，有的人喜欢打高分，有的人则喜欢打低分；每个项目自身的定位也不同，有些项目本身就比较差，有些项目则相对优质。该方法即避免双方之间对各自固有属性的影响。

3 面向电影推荐的一个算法实现

3.1 简介

实验要求中的第一点是一个电影推荐系统，使用 matlab 编程。该系统旨在根据输入的少量电影评价，利用大量的用户电影评价并使用协同过滤系统的 Funk SVD 算法来寻找最优的用户属性和项目属性，最后求解出输入用户的评分最高的前 10 项并推荐给他。以下是我对该系统的解读，包括代码部分的解读。

3.2 准备阶段

在系统的准备阶段，主要做的工作是读入电影列表和被推荐用户少量的电影评价。

```
movieList = loadMovieList();
```

代码中使用 loadMovieList() 函数读入电影列表。根据内部函数，我们可以得知电影总个数为 1682 个。

```
%  Initialize my ratings
my_ratings = zeros(1682, 1);

% Check the file movie_idx.txt for id of each movie in this dataset
% For example, Toy Story (1995) has ID 1, so to rate it "4", we can set
my_ratings(1) = 4;

% Or suppose did not enjoy Silence of the Lambs (1991), we can set
my_ratings(98) = 2;

% We have selected a few movies we liked / did not like and the ratings we
% gave are as follows:
my_ratings(7) = 3;
my_ratings(12)= 5;
(...)
```

首先初始化被推荐用户的评分，都设置成未看过（规定评分是 0 表示该用户未观看过该电影）。输入被推荐用户少量的电影评分，如该用户对第一个电影评分为 4，则写入 my_rating(1)=4。代码中的省略号是我自己修改的，表达的意思是之后还可以写很多电影的评分。其实这些评分都是可以修改的，甚至可以什么电影也没看过，推荐系统也依然会运转。当然写入的电影评分越多，则推荐系统推荐的电影越符合被推荐人。

3.3 训练阶段

在训练阶段，主要工作是构建损失函数，通过梯度下降算法来寻找最优的用户属性和项目属性。

```
%  Load data
load('ex8_movies.mat');

%  Y is a 1682x943 matrix, containing ratings (1-5) of 1682 movies by
%  943 users
```

```
%
%  R is a 1682x943 matrix, where R(i,j) = 1 if and only if user j gave a
%  rating to movie i

%  Add our own ratings to the data matrix
Y = [my_ratings Y];
R = [(my_ratings ~= 0) R];

%  Normalize Ratings
[Ynorm, Ymean] = normalizeRatings(Y, R);

%  Useful Values
num_users = size(Y, 2);
num_movies = size(Y, 1);
num_features = 10;
```

上述代码首先将 ex8_movies.mat 中存放的 943 个用户对 1682 个电影的评分矩阵读入内存，Y 矩阵表示评分矩阵（评分分数为 1 5），R 矩阵存放的是某个用户是否给出某个电影评价，若给出了，则是 1。并且将被推荐人的评分向量插入到两个矩阵中。而后计算得评分矩阵的标准化之后的矩阵（这里的标准化的意思是在 Y 矩阵中给出评分的电影评分中去中心化）和均值，并设置假设用户和电影的特征个数都为 num_features，这里的特征个数被设置成了 10，其实这个特征个数可以自己不断调参优化。

```
% Set Initial Parameters (Theta, X)
X = randn(num_movies, num_features);
Theta = randn(num_users, num_features);

initial_parameters = [X(:); Theta(:)];

% Set options for fmincg
options = optimset('GradObj', 'on', 'MaxIter', 100);

% Set Regularization
lambda = 10;
theta = fmincg (@(t)(cofiCostFunc(t, Ynorm, R, num_users, num_movies, ...
                                  num_features, lambda)), ...
                initial_parameters, options);

```

```
% Unfold the returned theta back into U and W
X = reshape(theta(1:num_movies*num_features), num_movies,
   num_features);
Theta = reshape(theta(num_movies*num_features+1:end), ...
                num_users, num_features);
```

将用户属性矩阵 X 和电影属性矩阵 $Theta$ 初始化为 0，并拼接在一起作为初始参数。并通过构建优化器 optimse 使用梯度下降迭代 100 次，设置正则化参数 λ 为 10 来进行训练。代码中的 cofiCosrFunc 函数为手动构建的价值函数（损失函数），相应代码如下：

```
function [J, grad] = cofiCostFunc(params, Y, R, num_users, num_movies
   , ...
                                  num_features, lambda)
%COFICOSTFUNC Collaborative filtering cost function
%   [J, grad] = COFICOSTFUNC(params, Y, R, num_users, num_movies, ...
%   num_features, lambda) returns the cost and gradient for the
%   collaborative filtering problem.
%

% Unfold the U and W matrices from params
X = reshape(params(1:num_movies*num_features), num_movies,
   num_features);
Theta = reshape(params(num_movies*num_features+1:end), ...
                num_users, num_features);


% You need to return the following values correctly
J = 0;
X_grad = zeros(size(X));
Theta_grad = zeros(size(Theta));


J = (1/2).*sum(sum(((X*Theta').*R-Y.*R).^2))+(lambda./2.*sum(sum(
   Theta.^2)))+(lambda./2.*sum(sum(X.^2)));
% Only predict rating X*Theta' if user has rated (i.e. R=1)

X_grad = (((X*Theta').*R*Theta-Y.*R*Theta)+lambda.*X);
Theta_grad = ((X'*((X*Theta').*R)-X'*(Y.*R)))'+lambda.*Theta;


```

```
grad = [X_grad(:); Theta_grad(:)];

end
```

该函数输入训练参数 $params$、标准化之后的评分矩阵 Y、是否给出评分矩阵 R、用户个数 num_users、电影个数 num_movies、特征个数 $num\_features$、正则化系数 $lamda$，输出损失值 J、两个带训练参数的梯度值 $grad$。利用第 2 节中所述的 Funk SVD 来进行梯度下降求解，最终得出最优化的用户属性矩阵和电影属性矩阵。

3.4 推荐阶段

```
p = X * Theta';
my_predictions = p(:,1) + Ymean;

movieList = loadMovieList();

[r, ix] = sort(my_predictions, 'descend');
fprintf('\nTop recommendations for new user:\n');
for i=1:10
    j = ix(i);
    fprintf('Predicting rating %.1f for movie %s\n', my_predictions(j
        ), ...
            movieList{j});
end
```

使用优化之后的参数 X 和 $Theta$ 重构 p 评分矩阵，由于训练时使用的是标准化之后的 Y 矩阵，所以在这里重构的评分矩阵需要加上 Y 矩阵的均值。随后，系统根据重构后的评分矩阵来进行电影推荐。

3.5 实验与结果展示

我使用 Matlab R2017a 版本进行该系统的实验测试，准备阶段、训练阶段、推荐阶段的结果如图 1所示。

写入了被推荐者有限个电影评分后，电影推荐系统通过对大量用户对电影的评分使用梯度下降算法最优化用户属性矩阵和电影属性矩阵，并在最后推荐出重构的 p 中评分排名最高的 10 个且未被被推荐者观看过的电影推荐给被推荐者。

为了展现实验的准确性，我利用 PCA 算法将训练之后的电影属性矩阵 X 降至 3 维。如图 2所示，为降维之后的 X 矩阵，观察图像，样本点较符合 3 维的正态分布，说明实验结果置信度较高且较为准确。

```
New user ratings:
Rated 4 for Toy Story (1995)
Rated 3 for Twelve Monkeys (1995)
Rated 5 for Usual Suspects, The (1995)
Rated 4 for Outbreak (1995)
Rated 5 for Shawshank Redemption, The (1994)
Rated 3 for While You Were Sleeping (1995)
Rated 5 for Forrest Gump (1994)
Rated 2 for Silence of the Lambs, The (1991)
Rated 4 for Alien (1979)
Rated 5 for Die Hard 2 (1990)
Rated 5 for Sphere (1998)
```

(a) 准备阶段

```
Training collaborative filtering...
Iteration     1 | Cost: 3.281339e+05
Iteration     2 | Cost: 1.246920e+05
Iteration     3 | Cost: 1.057921e+05
Iteration     4 | Cost: 8.016496e+04
Iteration     5 | Cost: 6.207777e+04
Iteration     6 | Cost: 5.318401e+04
Iteration     7 | Cost: 4.671600e+04
Iteration     8 | Cost: 4.476383e+04
Iteration     9 | Cost: 4.315379e+04
Iteration    10 | Cost: 4.201271e+04
Iteration    11 | Cost: 4.122893e+04
Iteration    12 | Cost: 4.072081e+04
```

(b) 训练阶段

```
Top recommendations for new user:
Predicting rating 5.0 for movie Someone Else's America (1995)
Predicting rating 5.0 for movie Santa with Muscles (1996)
Predicting rating 5.0 for movie Star Kid (1997)
Predicting rating 5.0 for movie Saint of Fort Washington, The (1993)
Predicting rating 5.0 for movie They Made Me a Criminal (1939)
Predicting rating 5.0 for movie Entertaining Angels: The Dorothy Day Story (1996)
Predicting rating 5.0 for movie Marlene Dietrich: Shadow and Light (1996)
Predicting rating 5.0 for movie Aiqing wansui (1994)
Predicting rating 5.0 for movie Great Day in Harlem, A (1994)
Predicting rating 5.0 for movie Prefontaine (1997)


Original ratings provided:
Rated 4 for Toy Story (1995)
Rated 3 for Twelve Monkeys (1995)
Rated 5 for Usual Suspects, The (1995)
Rated 4 for Outbreak (1995)
Rated 5 for Shawshank Redemption, The (1994)
Rated 3 for While You Were Sleeping (1995)
Rated 5 for Forrest Gump (1994)
Rated 2 for Silence of the Lambs, The (1991)
Rated 4 for Alien (1979)
Rated 5 for Die Hard 2 (1990)
Rated 5 for Sphere (1998)
```

(c) 推荐阶段

图 1: 电影推荐系统实验结果截图展示

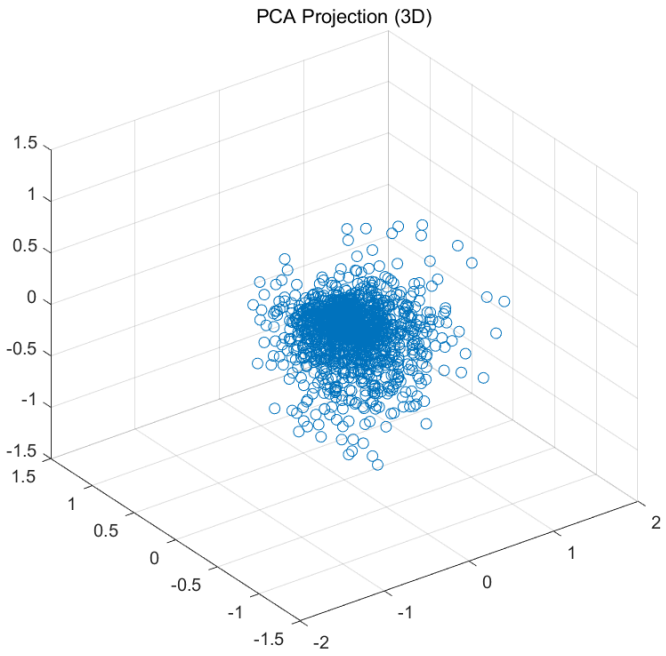

图 2: PCA 处理之后的电影属性矩阵

4 Surprise 库

4.1 介绍

Suprise 库（Simple Python RecommendatIon System Engine）是一个专为构建和评估推荐系统而设计的 Python 库。它提供了各种推荐算法和工具，以帮助数据科学家和开发人员快速创建、测试和部署推荐系统。本文提供了 Suprise 库的概述，介绍了其特点以及如何使用它。

4.2 主要特点

Suprise 具有以下主要特点：

- 实现了各种推荐算法，包括协同过滤、矩阵分解等。
- 简单直观的 API 设计，使得入门和构建推荐模型变得容易。
- 数据集加载和处理工具，包括内置数据集。
- 评估和交叉验证工具，用于评估推荐模型的性能。
- 可扩展性，允许用户实现自定义推荐算法并将其与库集成。
- 内置数据集，用于快速实验和测试。

4.3 用法示例

以下是使用 Suprise 库构建推荐模型并进行电影推荐的简单示例：

```
from surprise import Dataset, SVD
from surprise.model_selection import cross_validate

# 使用内置数据集MovieLens 100K
data = Dataset.load_builtin('ml-100k')

# 创建一个Reader对象，定义评分范围
reader = Reader(rating_scale=(1, 5))

# 加载数据集
trainset = data.build_full_trainset()

model = SVD()

# 交叉验证评估模型
cross_validate(model, data, measures=['RMSE', 'MAE'], cv=4, verbose=
    True)

# 训练模型
model.fit(trainset)

# 进行推荐
user_id = str(196)  # 示例用户ID
item_id = str(302)  # 示例电影ID

# 获取给定用户对于指定电影的评分预测
predicted_rating = model.predict(user_id, item_id).est
print(f'预测评分为：{predicted_rating}')
```

当读取 Surprise 内置的 MovieLens 100K 数据集后，我使用 SVD（奇异值分解）算法来构建推荐模型。为了评估模型的性能，我进行了经典的 4 折交叉验证，并计算了模型的 RMSE（均方根误差）和 MAE（平均绝对误差）指标，这些指标用于衡量模型的准确性和预测能力，得到以下结果：

```
Evaluating RMSE, MAE of algorithm SVD on 4 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Mean    Std
RMSE (testset)    0.9410  0.9506  0.9374  0.9389  0.9420  0.0051
```

```
MAE (testset)     0.7419  0.7511  0.7390  0.7394  0.7429  0.0049
Fit time          0.48    0.38    0.35    0.44    0.42    0.05
Test time         0.13    0.08    0.07    0.12    0.10    0.03
预测评分为: 4.035838307420791
```

在训练完模型之后，我进行了一个典型的推荐操作，输出了某个用户对于某个电影的评分预测。这个预测评分是根据该用户的历史行为和模型的学习来生成的，它代表了模型对用户对电影的兴趣程度的估计。这个用户-电影的评分预测可以用于个性化推荐，为用户提供更符合其兴趣的电影推荐列表。这个推荐过程可以通过改变用户 ID 和电影 ID 来生成不同用户对不同电影的评分预测，实现个性化推荐的目标。这一操作是推荐系统的核心功能，有助于提高用户满意度和平台的用户参与度。

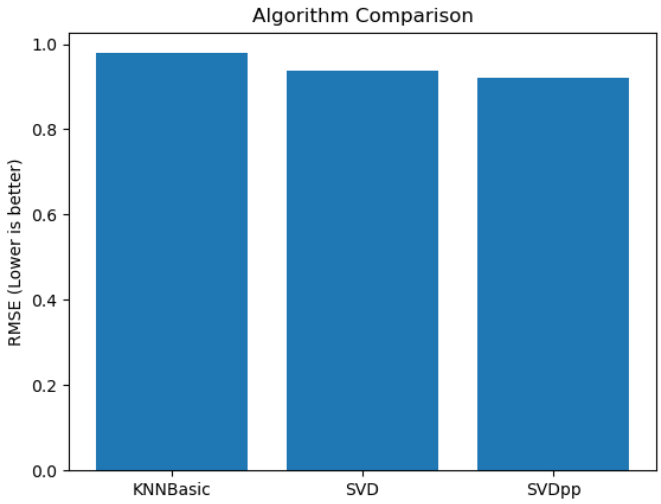

图 3: 三种不同算法的对比

4.4 不同算法的对比

由于内置数据集运算速度较快，可以检验不同算法之间的差距。于是我分别检验和对比了 SVD、KNNBasic、SVDpp 算法，并得出如图 3所示 RMSE 的结果，有图表可知，SVD++ 的误差明显最小，但使用 SVD++ 算法花费的时间相对较多。下列是相关结果：

```
Evaluating RMSE of algorithm KNNBasic on 4 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Mean    Std
RMSE (testset)    0.9880  0.9851  0.9821  0.9786  0.9835  0.0035
Fit time          0.13    0.14    0.15    0.13    0.14    0.01
Test time         1.85    1.76    1.78    1.72    1.78    0.05
```

```
Evaluating RMSE of algorithm SVD on 4 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Mean    Std
RMSE (testset)    0.9345  0.9365  0.9488  0.9360  0.9389  0.0058
Fit time          0.39    0.37    0.31    0.32    0.35    0.03
Test time         0.10    0.08    0.09    0.10    0.09    0.01
```

```
Evaluating RMSE of algorithm SVDpp on 4 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Mean    Std
RMSE (testset)    0.9226  0.9192  0.9144  0.9275  0.9210  0.0048
Fit time          7.80    7.76    7.67    7.77    7.75    0.05
Test time         1.95    1.91    1.75    2.93    2.14    0.47
```

5 总结

通过学习和实践推荐系统，我深刻理解了其在提高用户体验和商业价值方面的重要性。了解了利用梯度下降来求解矩阵分解问题进而预测推荐结果，掌握了推荐系统的基本原理，包括协同过滤和矩阵分解等核心算法，并使用 Surprise 库构建了推荐模型，通过交叉验证评估了其性能。此外，我学会了如何加载和处理推荐系统的数据集，以及如何在实际应用中进行推荐。