Project on
MOVIES RECOMMENDATION ENGINE

Submitted by

SHUBHAM BALASAHEB PATIL

Internship at

ECKOVATION CAREERS

Course Name

PYTHON PROGRAMMING

Index

ABSTRACT

Recommender Systems Have Become Ubiquitous In Our Lives.Yet, currently, they are far from optimal. In this project, we attempt to understand different kinds of recommendations and compare their performance on the Movie Lens dataset.We attempt to build a scalable model to perform this analysis. We start by preparing and comparingthevariousmodelsonasmaller dataset of100,000ratings.Then,we try to scale the algorithms that are able to handle 20 million ratings by using Apache Spark. We find thatforthesmaller dataset,using user-based collaborative filtering results in the lowest Mean Squared Error on our dataset.

INTRODUCTION

A recommendation system isatype of information filtering system which attempts to predict the preferences of a user, and make suggestions based on these preferences. There are a wide variety of applications for recommendation systems. These have become increasingly popular over the last few years and are now utilized inmostonlineplatforms whoweuse. The Content of such platforms varies from movies,music, books and videos, to friends and stories on social media platforms, to products one-commerce websites,to people on professional and dating websites,to search results returned on Google. Often, these systems are able to collect information about a user's choices, and can use this information to improve their suggestions in the future. For example, Facebook can monitor your interaction with various stories on your feed in order to learn what types of stories appeal to you. Sometimes, the recommender systems can make improvements based on activities of large number of people.For example, if Amazon observes that a large number of customers who buy the latest Apple Macbook also buy a USB-C-toUSB Adapter, they can recommend the Adapter to an user who has just added a Macbook to his cart. Due to the advance in recommender systems, users constantly expect good recommendations. They have a low threshold for services that are not able to make appropriate suggestions. If a music streaming app is not able to predict and play music that the user likes, then the user will simply stop using it. This has led to a high emphasis by tech companies on improving their recommendation systems. However, the problem is more complex than it seems. Every User has different preferences and likes. In Addition,even the taste of a single user can vary depending on a large number of factors, such as mood, season, or type of activity the user is doing. For example, the type of music one would like to hear while exercising differs greatly from type music he'd listen to when cooking dinner.Another Issue That Recommendation Systems Have Search Engine Architecture, Spring 2017, NYU Courant to solve is the exploration vs exploitation problem. They must explore new domains to discover more about the user, while still making the most of what is already known about the user. Two main approaches are widely used for recommender systems. One is content-based filtering, where we try to profile the users interests using information collected, and recommend items based on that profile. The other is collaborative filtering, where we try to group similar users together and use information about the group to make recommendations. Both approaches are discussed in greater detail.

TECHNOLOGY USED

1. Python

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

2. Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Jupyter Notebook provides a browser-based REPL built upon a number of popular open-source libraries:

- IPython
- ØMQ
- Tornado (web server)
- jQuery
- Bootstrap (front-end framework)
- MathJax

Jupyter Notebook can connect to many kernels to allow programming in many languages. By default Jupyter Notebook ships with the IPython kernel. As of the 2.3 release (October 2014), there are currently 49 Jupyter-compatible kernels for many programming languages, including Python, R, Julia and Haskell.

PROBLEM STATEMENT

The problem that we address in this project can be formulated as follows. Let R be the ratings matrix with dimensions (num_users × num_items). The entry ri j in the ratings matrix R contains a non- zero rating value given by the user i for the item j. The matrix R is sparse in nature i.e. most of the entries ri j are missing. We generally have a few ratings or some purchase history for each user and similarly each item will have been rated by a few users, but most of the entries in the ratings matrix are generally missing. The task at hand is to predict the missing entries in the ratings matrix R.

The data that constitutes the ratings matrix R can be collected either explicitly by asking the users to rate the items or by implicitly deriving the ratings for items based on measures such as whether the user purchased the item, or whether the user clicked a certain page and likewise. We work with the MovieLens dataset, which contains explicit ratings given by users to movies. The smaller version of the dataset can be processed by using dense matrices and non-vectorized operations. However, if we try to use the simple implementations on the larger 20 million dataset, the code breaks. If we store the ratings matrix for the 20 million dataset in a dense format, it would take up around $140000 \times 27000 \times 8 = 28$GB of memory, assuming 8 bytes per matrix entry. Thus we need to use sparse matrix representations to be able to handle the bigger dataset effectively. Also, the matrix operations have to be as vectorized as possible to make efficient use of threads. We observed that even after our best attempts to optimize the code as much as possible, the algorithms still needed a lot of time to be able to process such huge amounts of data. We thus decided to make use of Apache Spark to parallelize the operations and improve the runtime performance of the algorithms by running them on Spark clusters.

COLLABORATIVE FILTERING

Collaborative Filtering techniques make recommendations for a user based on ratings and preferences data of many users. The main underlying idea is that if two users have both liked certain common items, then the items that one user has liked that the other user has not yet tried can be recommended to him. We see collaborative filtering techniques in action on various Internet platforms such as Amazon.com, Netflix, Facebook. We are recommended items based on the ratings and purchase data that these platforms collect from their user base. We explore two algorithms for Collaborative filtering, the Near- est Neighbors Algorithm and the Latent Factors Algorithm. We implement the nearest neighbors and latent factor methods for Collaborative filtering.

Nearest Neighbors Collaborative Filtering:

This approach relies on the idea that users who have similar rating behaviors so far, share the same tastes and will likely exhibit similar rating behaviors going forward. The algorithm first computes the similarity between users by using the row vector in the ratings matrix corresponding to a user as a representation for that user. The similarity is computed by using either cosine similarity or Pearson Correlation. In order to predict the rating for a particular user for a given movie j, we find the top k similar users to this particular user and then take a weighted average of the ratings of the k similar users with the weights being the similarity values.

```
In [10]: from scipy.sparse import csr_matrix

         movie_features_df_matrix = csr_matrix(movie_features_df.values)

         from sklearn.neighbors import NearestNeighbors


         model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')
         model_knn.fit(movie_features_df_matrix)

Out[10]: NearestNeighbors(algorithm='brute', leaf_size=30, metric='cosine',
                          metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                          radius=1.0)
```

Latent Factor Methods:

The latent factor algorithm looks to decompose the ratings matrix R into two tall and thin matrices Q and P, with matrix Q having dimensions num_users × k and P having the dimensions numi tems × k where k is the number of latent factors. The decomposition of R into Q and P is such that

R = Q.PT

Any rating ri j in the ratings matrix can be computed by taking the dot product of row qi of matrix Q and pj of matrix P. The ma- trices Q and P are initialized randomly or by performing SVD on the ratings matrix. Then, the algorithm solves the problem of mini- mizing the error between the actual rating value ri j and the value given by taking the dot product of rows qi and pj . The algorithm performs stochastic gradient descent to find the matrices Q and P with minimum error starting from the initial matrices.

Content Based Recommendations

Content Based Recommendation algorithm takes into account the likes and dislikes of the user and generates a User Profile. For generating a user profile, we take into account the item profiles(vector describing an item) and their corresponding user rating. The user profile is the weighted sum of the item profiles with weights being the ratings user rated. Once the user profile is generated, we calculate the similarity of the user profile with all the items in the dataset, which is calculated using cosine similarity between the user profile and item profile. Advantages of Content Based approach is that data of other users is not required and the recommender engine can recommend new items which are not rated currently, but the recommender algorithm doesn't recommend the items outside the category of items the user has rated.

In [4]:
```
combine_movie_rating = df.dropna(axis = 0, subset = ['title'])
movie_ratingCount = (combine_movie_rating.
     groupby(by = ['title'])['rating'].
     count().
     reset_index().
     rename(columns = {'rating': 'totalRatingCount'})
     [['title', 'totalRatingCount']]
    )
movie_ratingCount.head()
```

Out[4]:

	title	totalRatingCount
0	'71 (2014)	1
1	'Hellboy': The Seeds of Creation (2004)	1
2	'Round Midnight (1986)	2
3	'Salem's Lot (2004)	1
4	'Til There Was You (1997)	2

Dataset

We used the MovieLens movie ratings dataset for our experiments1. The experiments are conducted on two versions of the dataset. The first version consists of 100004 ratings by 671 users across 9125 movies. The ratings allowed at intervals of 0.5 on a 5-point scale, starting from 0.5 and going to 5. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided. The dataset has additional information about the movies in the form of genre and tags, however we use only the ratings given by the users to the movies and ignore the other information for the collaborative filtering techniques. For the content filtering portion, information was scrapped from the IMDB website for the corresponding movie. The links to the IMDB page for each movie is provided in a separate file.

The second and bigger version of the dataset consists of 20000263 (20 million) ratings by 138493 users across 27278 movies. Apart from this, the structure of the two datasets also is identical. The movie ids for a particular movie is the same in both datasets, but the user id for the same user is different for the two datasets. We split each dataset into three partitions: training, validation and test by sampling randomly in the ratios 80%, 10%, 10% respec- tively. The validation partition is used to tune the hyper-parameters for the nearest neighbor and latent factor algorithms. The bigger version of the dataset presents significant scalability challenges.

MOVIE PLOT KEYWORD SEARCH

We also implemented an additional feature that provides a dis- tributed indexer on the movie synopsis as extracted from the IMDB website to allow searching for movies based on keyword searches. It is built on a framework to build an inverted index. The framework for the indexer is the same as the one developed earlier in the semester for the assignments. It has been modified to allow searching on the movie information dataset. First, the dataset is generated by scraping data from the IMDB pages of the respective movies. The IMDB movie id is provided along with the movie lens dataset. After accessing the webpage, the metadata from the movie is collected and then pickled and stored in a dump. This same dump can also be used to implement and extend the content based model.

In the current model, the synopsis refers to the first of the pro- vided plot summaries. The summaries tend to be much shorter than the synopsis and can be extracted much more easily. Also, the size of the dump increased by a factor of almost 100 when using synopses instead of plot summaries. One more reason to select plot summaries is that detailed synopsis are not available for a large portion of the movies, which would result in a bias in the search. The rest of the search is similar to the one provided in the assign- ments. The reformatter has been adjusted to match the format of the dump. The rest of the search can be run in the identical manner as the assignment.

The user bias is given by the difference between the average user rating and the global average rating. The movie bias is given by the difference between the average movie rating and the global average rating. Consider the following example which demonstrates the working of the adjusted average method. Let the global average rating be 3.7. The user A has an average rating of 4.1. Thus the bias of the user is (4.1 - 3.7) I.e the user rates 0.4 stars more than the global average. The movie Fast and Furious has an average rating of 3.1 stars. Thus the bias for the movie is -0.6. the adjusted average method will predict that user A will give the movie Fast and Furious a rating of 3.7 + 0.4 - 0.6 = 3.5 . We implemented the algorithms such that they could be run on an Apache Spark cluster. We then run the algorithms on a cluster of 20 AWS EC2 instances. We were able to achieve significant improve- ments in the running time by using Spark. The nearest neighbors algorithm took around 30 hrs to run on a single machine, whereas the Spark implementation on the 20 machine cluster was able to run in around 50 minutes.

CREDITS, GENRES AND KEYWORDS BASED RECOMMENDER

It goes without saying that the quality of our recommender would be increased with the usage of better metadata. That is exactly what we are going to do in this section. We are going to build a recommendation based on the following metadata: the 3 top actors, the director, related genres and the movie plot keywords. From the cast, crew and keywords features, we need to extract the three most important actors, the director and the keywords associated with that movie. Right now, our data is present in the form of "stringified" lists , we need to convert it into a safe and usable structure

Advantages of content-based filtering are:

● They are capable of recommending unrated items.

● We can easily explain the working of the recommender system by listing the Content features of an item.

● Content-based recommender systems use only the rating of the concerned user,and not any other user of the system.

Disadvantages of content-based filtering are:

● It does not work for a new user who has not rated any item yet as enough ratings are required content based recommender evaluates the user preferences and provides accurate recommendations.

● No recommendation of serendipitous items.

● Limited Content Analysis- The recommendation does not work if the system fails to distinguish the items that a user likes from the items that he does not like.

```
In [9]: movie_features_df=rating_popular_movie.pivot_table(index='title',columns='userId',values='rating').fillna(0)
        movie_features_df.head()
```

Out[9]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
title																					
10 Things I Hate About You (1999)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	3.000	0.000	5.000	0.000	0.000	0.000	0.000	0.000
12 Angry Men (1957)	0.000	0.000	0.000	5.000	0.000	0.000	0.000	0.000	0.000	0.000	...	5.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2001: A Space Odyssey (1968)	0.000	0.000	0.000	0.000	0.000	0.000	4.000	0.000	0.000	0.000	...	0.000	0.000	5.000	0.000	0.000	5.000	0.000	3.000	0.000	4.500
28 Days Later (2002)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	3.500	0.000	5.000
300 (2007)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	3.000	...	0.000	0.000	0.000	0.000	3.000	0.000	0.000	5.000	0.000	4.000

5 rows × 606 columns

3. Collaborative filtering based systems:

Our content based engine suffers from some severe limitations. It is only capable of suggesting movies which are close to a certain movie. That is, it is not capable of capturing tastes and providing recommendations across genres. Also, the engine that we built is not really personal in that it doesn't capture the personal tastes and biases of a user. Anyone querying our engine for recommendations based on a movie will receive the same recommendations for that movie, regardless of who she/he is. Therefore, in this section, we will use a technique called Collaborative Filtering to make recommendations to Movie Watchers. It is basically of two types:-

a) User based filtering-

These systems recommend products to a user that similar users have liked. For measuring the similarity between two users we can either use 16 pearson correlation or cosine similarity. This filtering technique can be illustrated with an example. In the following matrix's, each row represents a user, while the columns correspond to different movies except the last one which records the similarity between that user and the target user. Each cell represents the rating that the user gives to that movie. Assume user E is the target. User Based Filtering Although computing user-based CF is very simple, it suffers from several problems. One main issue is that users' preferences can change over time. It indicates that precomputing the matrix based on their neighboring users may lead to bad performance. To tackle this problem, we can apply item-based CF.

b) Item Based Collaborative Filtering -

Instead of measuring the similarity between users, the item-based CF recommends items based on their similarity with the items that the target user rated. Likewise, the similarity can be computed with 17 Pearson Correlation or Cosine Similarity. The major difference is that, with item-based collaborative filtering, we fill in the blank vertically, as opposed to the horizontal manner that user-based CF does. The following table shows how to do so for the movie. Item Based Filtering It successfully avoids the problem posed by dynamic user preference as item-based CF is more static. However, several problems remain for this method. First, the main issue is scalability. The computation grows with both the customer and the product. The worst case complexity is O(mn) with m users and n items. In addition, sparsity is another concern. Take a look at the above table again. Although there is only one user that rated both Matrix and Titanic rated, the similarity between them is 1. In extreme cases, we can have millions of users and the similarity between two fairly different movies could be very high simply because they have similar rank for the only user who ranked them both.

RESULTS

For movie with ID 369, we get an estimated prediction of 0.5051. One startling feature of this recommender system is that it doesn't care what the movie is (or what it contains). It works purely on the basis of an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie.

In [15]:
```
query_index = np.random.choice(movie_features_df.shape[0])
print(query_index)
distances, indices = model_knn.kneighbors(movie_features_df.iloc[query_index,:].values.reshape(1, -1), n_neighbors = 6)
```

```
369
```

In [16]:
```
movie_features_df.head()
```

Out[16]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
title																					
10 Things I Hate About You (1999)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	3.000	0.000	5.000	0.000	0.000	0.000	0.000	0.000
12 Angry Men (1957)	0.000	0.000	0.000	5.000	0.000	0.000	0.000	0.000	0.000	0.000	...	5.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2001: A Space Odyssey (1968)	0.000	0.000	0.000	0.000	0.000	0.000	4.000	0.000	0.000	0.000	...	0.000	0.000	5.000	0.000	0.000	5.000	0.000	3.000	0.000	4.500
28 Days Later (2002)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	3.500	0.000	5.000
300 (2007)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	3.000	...	0.000	0.000	0.000	0.000	3.000	0.000	0.000	5.000	0.000	4.000

5 rows × 606 columns

In [17]:
```
for i in range(0, len(distances.flatten())):
    if i == 0:
        print('Recommendations for {0}:\n'.format(movie_features_df.index[query_index]))
    else:
        print('{0}: {1}, with distance of {2}:'.format(i, movie_features_df.index[indices.flatten()[i]], distances.flatten()[i])
```

```
Recommendations for Space Jam (1996):

1: Hook (1991), with distance of 0.5051567554473877:
2: Happy Gilmore (1996), with distance of 0.6052191853523254:
3: Dogma (1999), with distance of 0.6162223219871521:
4: Toy Story (1995), with distance of 0.6174311637878418:
5: Dragonheart (1996), with distance of 0.617621898651123:
```

CONCLUSION

A hybrid approach is taken between context based filtering and collaborative filtering to implement the system. This approach overcomes drawbacks of each individual algorithm and improves the performance of the system. Techniques like Clustering, Similarity and Classification are used to get better recommendations thus reducing MAE and increasing precision and accuracy. In future we can work on hybrid recommenders using clustering and similarity for better performance. Our approach can be further extended to other domains to recommend songs, video, venue, news, books, tourism and e-commerce sites, etc.

REFERENCES

[1]A Survey of Collaborative Filtering Techniques; Su et al; https://www.hindawi.com/journals/aai/2009/421425

[2]Google News Personalization: Scalable Online Collaborative Filtering; Das et al; https://www2007.org/papers/paper570.pdf

[3]Intro to Recommender Systems: Collaborative Filtering; http://blog.ethanrosenthal.com/2015/11/02/intro-to-collaborative -filtering

[4]Collaborative Filtering Recommender Systems;
Stanford Student project; http://cs229.stanford.edu/proj2014/Rahul%20Makhijani,%20Saleh%20Samaneh,%20Megh%20Mehta,%20Collaborative%20Filtering%20Recommender%20Systems.pdf

[5]MMDS course slides; Jeffrey Ullman; http://infolab.stanford.edu/ ullman/mmds/ch9.pdf

[6]Recommending items to more than a billion people; Kabiljo et al; https://code.facebook.com/posts/861999383875667/recommending-items-to-more-than-a-billion-people