DISTRIBUTED AND SCALABLE DATA ENGINEERING
FINAL PROJECT REPORT

On

Collaborative Filtering using the Netflix Data

By
Thrinath Nalajala

Under the guidance of
Vahid Behzadan

Tagliatela College of Engineering

Department of Computer Science
(Data Science)

300 Boston Post Rd, West Haven, CT 06516

COLLABORATIVE FILTERING

These days whether you look at a video on YouTube, a movie on Netflix or a product on Amazon, you are going to get recommendations for more things to view, like or buy. You can thank the advent of machine learning algorithms and recommender systems for this development.

Recommender systems are far-reaching in scope, so we are going to zero in on an important approach called collaborative filtering, which filters information by using the interactions and data collected by the system from other users. It is based on the idea that people who agreed in their evaluation of certain items are likely to agree again in the future.

Motivation :

Collaborative filtering comes from the idea that people often get the best recommendations from someone with tastes similar to themselves. Collaborative filtering encompasses techniques for matching people with similar interests and making recommendations on this basis.

RECOMMENDER SYSTEMS:

A recommender system is a subclass of information filtering that seeks to predict the "rating" or "preference" a user will give an item, such as a product, movie, song, etc.
Recommender systems provide personalized information by learning the user's interests through traces of interaction with that user. Much like machine learning algorithms, a recommender system makes a prediction based on a user's past behaviors. Specifically, it is designed to predict user preference for a set of items based on experience.
Mathematically, a recommendation task is set to be:

- Set of Users
- Set of Items that are recommended to the User
- Learn a function based on user's past interaction data that predicts the likeliness of item to user

Recommender systems are broadly classified into two types based on the data being used to make inferences:

- Content based filtering, which uses item attributes.
- Collaborative filtering, which uses user behavior in addition to item attributes.

COLLABORATIVE FILTERING:

Collaborative filtering filters information by using the interactions and data collected by the system from other users. It is based on the idea that people who agreed in their evaluation of certain items are likely to agree again in the future.

The concept is simple: when we want to find a new movie to watch we will often ask our friends for recommendations. Naturally, we have greater trust in the recommendations from friends who share tastes similar to our own.

Most collaborative filtering systems apply the so-called similarity index-based technique. In the neighborhood-based approach, number of users are selected based on their similarity to the active user. Inference for the active user is made by calculating a weighted average of the ratings of the selected users.

Collaborative-filtering systems focus on the relationship between users and items. The similarity of items is determined by the similarity of the ratings of those items by the users who have rated both items.

There are two classes of Collaborative Filtering:

- User-based, which measures the similarity between target users and other users.
- Item-based, which measures the similarity between the items that target user rate or interact with and other items.

COLLABORATIVE FILTERING USING PYSPARK:

The project is done in Jupyter notebook created on AWS EMR Cluster.

The implementation of project starts with creating an EMR cluster on AWS:

1. Choose Create cluster.

2. On the Create Cluster - Quick Options page, accept the default values except for the following fields:

- Enter a Cluster name.
- In Software configuration select the applications as spark.
- Under Security and access, choose the EC2 key pair that you created in Create an Amazon EC2 Key Pair.

3. Choose Create cluster.

Create EMR notebook using EMR Cluster:

EMR Notebook:

EMR Notebooks, a managed environment, based on Jupyter Notebooks that allows data scientists, analysts, and developers to prepare and visualize data, collaborate with peers, build applications, and perform interactive analysis using EMR clusters. EMR Notebooks is pre-configured for Spark. It supports Spark magic kernels allowing you to interactively run Spark jobs on EMR clusters written in languages such as PySpark, Spark SQL, Spark R, and Scala.

Steps to create EMR notebook:

1. Choose Notebooks, Create notebook.
2. Enter a Notebook name and an optional Notebook description.
3. select Choose, select a cluster from the list (cluster created above), and then Choose cluster. Only clusters that meet the requirements are listed.
4. For Security groups, choose Use default security groups.
5. For AWS Service Role, leave the default or choose a custom role from the list. For more information, see Service Role for EMR Notebooks.
6. For Notebook location choose the location in Amazon S3 where the notebook file is saved.

Open the Notebook created and select pyspark in jupyter notebook.

BUILDING THE RECOMMENDER SYSTEM

Also called User-User Filtering, it uses other users to recommend items to the input user. It attempts to find users that have similar preferences and opinions as the input and then recommends items that they have liked to the input. There are several methods of finding similar users and the one we will be using here is going to be based on the Pearson Correlation Function**.**

Implementation of Collaborative filtering on Netflix data in pyspark:

Analyzing the Netflix Data:

- Import pyspark
- The text data files are stored in S3. Import the text files and formed the Spark data frame.

The collaborative filtering approaches lives from finding many similar users (for a user-user model) or many similar items (item-item model):

- **User-User collaborative filtering:**
  The method identifies users that are similar to the queried user and estimate the desired rating to be the weighted average of the ratings of these similar users.

- **ITEM-ITEM collaborative filtering**:

  ITEM-ITEM collaborative filtering look for items that are similar to the articles that user has already rated and recommend most similar articles. But what does that mean when we say item-item similarity? In this case we do not mean whether two items are the same by attribute like Fountain pen and pilot pen are similar because both are pen. Instead, what similarity means is how people treat two items the same in terms of like and dislike.

ITEM-ITEM Based Approach:

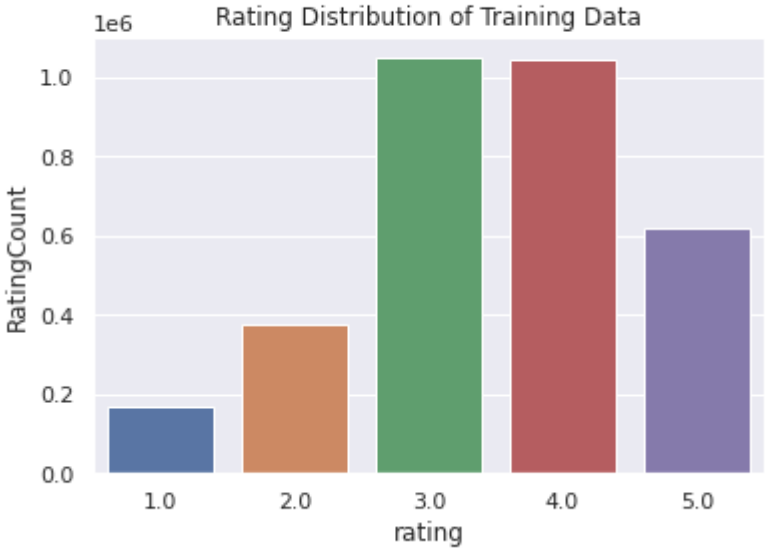
This method is quite stable as compared to User based collaborative filtering because the average item has a lot more ratings than the average user. So, an individual rating does not impact as much. To calculate similarity between two items, we look into the set of items the target user has rated and computes how similar they are to the target item i and then selects k most similar items.

To determine the most-similar match for a given item, the algorithm builds a similar-items table by finding items that customers tend to purchase together. We could build a item-to-item matrix by iterating through all item pairs and computing a similarity metric for each pair. However, many product pairs have no common customers, and thus the approach is inefficient in terms of processing time and memory usage. The following iterative algorithm provides a better approach by calculating the similarity between a single item and all related items:

Distinct Movies and distinct Users in the Training set and Testing set :

Rating Distribution of Training Data

```
Out[14]: Text(0, 0.5, 'RatingCount')
```



Rating Distribution of Testing Data:

```
Out[15]: Text(0, 0.5, 'RatingCount')
```

If we find similar users, then we only have to do the process once for user.From the set of similar users we can estimate all the blanks in the utility matrix for User.If we work from similar items, we have to compute similar items for almost all items, before we can estimate.

On the other hand, item-item similarity often provides more reliable information, because of the phenomenon observed above, namely that it is easier to find movies of the same rating than it is to find users that like only movies of a single rating.

Once we have the similarity between the items, the prediction is then computed by taking a weighted average of the target user's ratings on these similar items. The formula to calculate rating is very similar to the user based collaborative filtering except the weights are between items instead of between users. And we use the current users rating for the item or for other items, instead of other users rating for the current items.

. $$s(i;u) = \mu_i + \frac{\sum_{j \in I_u}(r_{uj} - \mu_j)w_{ij}}{\sum_{j \in I_u}|w_{ij}|}$$

The above formula has been implemented as code

Problem 3: Collaborative Filtering Implementation:

Step 1: Implementation: By using the Item-Item based approach predicted the ratings of the Test set.

Collaborative filtering (CF) is a technique used by recommender systems. ... In the newer, narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users

Execution and Evaluation:

```
seed = 1800009193
(split_60_df, split_a_20_df, split_b_20_df) = trainingRatings_df.randomSplit([6.0, 2.0, 2.0], seed)

# Let's cache these datasets for performance
training_df = split_60_df.cache()
validation_df = split_a_20_df.cache()
test_df = split_b_20_df.cache()

print('Training: {0}, validation: {1}, test: {2}\n'.format(
  training_df.count(), validation_df.count(), test_df.count())
)

#pandasDF = trainingRatings_df.toPandas()
pandasDF = validation_df.toPandas()  #taking more time to execute so given validation data as input
reader = Reader(rating_scale=(1, 5))
dataset = Dataset.load_from_df(pandasDF,reader)
"""
benchmark = []
# Iterate over all algorithms
for algorithm in [KNNBaseline(), KNNBasic(), KNNWithMeans(), KNNWithZScore()]:
    # Perform cross validation
    results = cross_validate(algorithm, dataset, measures=['RMSE'], cv=5, verbose=False)

    # Get results & append algorithm name
    tmp = pd.DataFrame.from_dict(results).mean(axis=0)
    tmp = tmp.append(pd.Series([str(algorithm).split(' ')[0].split('.')[-1]], index=['Algorithm']))
    benchmark.append(tmp)

pd.DataFrame(benchmark).set_index('Algorithm').sort_values('test_rmse')
```

Output for Collaborative Filtering Implementation for KNNs:

]:

Algorithm	test_rmse	fit_time	test_time
KNNBaseline	0.921654	3.508207	10.072906
KNNWithMeans	0.923255	0.951716	7.636521
KNNWithZScore	0.924319	1.566507	9.154764
KNNBasic	0.992983	0.763807	7.456145

Collaborative Filtering Implementation - Using ALS:

ALS is implemented in Apache Spark ML and built for a larges-scale collaborative filtering problems. ALS is doing a pretty good job at solving scalability and sparseness of the Ratings data, and it's simple and scales well to very large datasets.

```
from pyspark.ml.recommendation import ALS

# Let's initialize our ALS learner
als = ALS()
seed = 1800009193
# Now we set the parameters for the method
(als.setMaxIter(5)
   .setSeed(seed)
   .setRegParam(0.1)
   .setItemCol("movieId")
   .setUserCol("userId")
   .setRatingCol("rating"))

# Now let's compute an evaluation metric for our test dataset
from pyspark.ml.evaluation import RegressionEvaluator

# Create an RMSE evaluator using the label and predicted columns
reg_eval = RegressionEvaluator(predictionCol="prediction", labelCol="rating", metricName="rmse")

# Create an MSE evaluator using the label and predicted columns
reg_mse_eval = RegressionEvaluator(predictionCol="prediction", labelCol="rating", metricName="mse

tolerance = 0.03
ranks = [4, 8, 12]
```

Output for Collaborative Filtering Implementation - Using ALS:

```
For rank 8 the RMSE is 0.8616004838570359
For rank 8 the MSE is 0.8616004838570359
Rank: 12
predicted_ratings_df
+-------+-------+------+----------+
|movieId|userId |rating|prediction|
+-------+-------+------+----------+
|28     |2358799|3.0   |3.6669366 |
|156    |973051 |5.0   |3.9271145 |
|851    |1189060|3.0   |3.5178668 |
+-------+-------+------+----------+
only showing top 3 rows

For rank 12 the RMSE is 0.8681582347312297
For rank 12 the MSE is 0.8681582347312297

The best model was trained with rank 8 with respect to RMSE
The best model was trained with rank 8 with respect to MSE
```

The Root Mean Squared Error- 0.833000416463466

Whichever method we choose, we should precompute preferred movies for each user, rather than waiting until we need to make a decision. Since the utility matrix evolves slowly, it is generally sufficient to compute it infrequently and assume that it remains fixed between recomputations.
Fill the null values with zero and compute the Correlation matrix for similarity measure. Correlation matrix gives the similarity between the items or movies in the train dataset and also obtained the k-most similar Users and k-most similar Items.

Step 3: Does your approach work for your own preferences

Added some movie ids with a customer id to my test set and predicted the ratings of movies. I have added movie id's with a customer id with ratings. Out of movie id's few ratings predicted match the true rating given.

Conclusion:

Recommendation algorithms provide an effective form of targeted marketing by creating a personalized shopping experience for each customer. collaborative filtering provides a powerful way for data scientists to recommend new products or items to users. If we have well-detailed metadata about your products, we can also use a content-based approach to recommendations. For large retailers like Amazon, Netflix a good recommendation algorithm is scalable over very large customer bases and product catalogs, requires only sub second processing time to generate online recommendations, is able to react immediately to changes in a user's data, and makes compelling recommendations for all users regardless of the number of purchases and ratings. Unlike other algorithms, this collaborative filtering is able to meet this challenge.

Bibliography:

https://pub.towardsai.net/recommendation-system-in-depth-tutorial-with-python-for-netflix-using-collaborative-filtering-533ff8a0e444

https://medium.com/@srinidhi14vaddy/collaborative-filtering-movie-recommendation-60461c7ef897

https://medium.com/@tomar.ankur287/item-item-collaborative-filtering-recommender-system-in-python-cf3c945fae1e

https://medium.com/sfu-cspmp/recommendation-systems-user-based-collaborative-filtering-using-n-nearest-neighbors-bf7361dc24e0

https://spark.apache.org/docs/2.2.0/sql-programming-guide.html