Distributed & Scalable Data Engineering

(DSCI-6007-2)

Project Report

Sowmya Davuluri, Naresh Venkataramanan

TITLE: Collaborative-Filtering-Netflix

ABSTRACT

The Recommendation system plays a major role nowadays, which is used for many applications. We know that the online content and service providers have a huge amount of content so the problem which arises is which data is required for whom so the problem of providing apposite content frequently. This paper represents the overview and approaches of techniques generated in a recommendation system. One of them is Collaborative Filtering which we will use for Netflix movie recommendation.

INTRODUCTION

https://github.com/Nareshvrao

A recommendation model, in simple terms, is an algorithm that aims to provide the most relevant and relatable information to a user depending on the behavior of the user. Companies like Netflix and Google have a huge database of the behaviors of data collected to be able to perform state-of-the-art recommendations so that they can display the most relevant content or services to the users to increase engagement. In part 1, we would be building a recommendation model using Collaborative Filtering to recommend movies to users.

Collaborative Filtering:

Collaborative filtering tackles the similarities between the users and items to perform recommendations. Meaning that the algorithm constantly finds the relationships between the users and in-turns does the recommendations. The algorithm learns the embeddings between the users without having to tune the features. The most common technique is by performing Matrix Factorization to find the embeddings or features that make up the interest of a particular user.

Matrix Factorization

Matrix factorization is an embedding. Say we have a user-movie matrix or feedback matrix, A^{NM}, the models learn to decompose into:

- A user embedding vector U, where row N is the embedding for item M.
- An item embedding vector V, where row M is the embedding for item N

The embedding vector is learned such that by performing UV^T, an approximation of the feedback matrix, A can be formed.

Loss Function

To approximate the feedback matrix, a loss function is needed. One of the intuitive loss functions is using mean squared error (MSE). MSE comp utes the difference in the feedback matrix A and the approximated UV^T matrix. In simple terms:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

where:

- n represents the total number of users
- Y_i represents the feedback matrix A
- Y_i_bar represents the approximated matrix UV^T

Alternating Least Square (ALS) with Spark ML

Alternating Least Square (ALS) is also a matrix factorization algorithm and it runs itself in a parallel fashion. 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.

Some high-level ideas behind ALS are:

- Its objective function is slightly different than Funk SVD: ALS uses L2 regularization while Funk uses L1 regularization

- Its training routine is different: ALS minimizes two loss functions alternatively; It first holds user matrix fixed and runs gradient descent with item matrix; then it holds item matrix fixed and runs gradient descent with user matrix

- Its scalability: ALS runs its gradient descent in parallel across multiple partitions of the underlying training data from a cluster of machines

Just like other machine learning algorithms, ALS has its own set of hyper-parameters. We probably want to tune its hyper-parameters via hold-out validation or cross-validation.

Making recommendations

Now that we know we have a wonderful model for movie recommendation, the next question is: how do we take our wonderful model and productize it into a recommender system? Machine learning model productization is another big topic and I won't get into details about it. In this post, I will show how to build a MVP (minimum viable product) version for ALS recommender.

To productize a model, we need to build a work flow around the model. Typical ML work flow roughly starts with data preparation via pre-defined set of ETL jobs, offline/online model training, then ingesting trained models to web services for production. In our case, we are going to build a very minimum version of movie recommender that just does the job.

Alternating Least Squares (ALS) is collaborative filtering method for recommender systems.

Pearson Correlation Coefficient (PCC) is one of the most popular similarity measures for Collaborative filtering recommender system, to evaluate how much two users are correlated.

Dataset [TrainingRating, TestingRating and Movies] which has data UsedId, MovieId, Rating and Movie name.

- First I load three of datasets, namely movie, train and test rating in spark environment.
- Data preprocessing.
- Build an ALS model based by tuning parameters (maxIter, rank, regParam) to obtain the model with the smallest RMSE on the validation set.

```
# Create ALS model
# Build the recommendation model using ALS on the training data
# Note we set cold start strategy to 'drop' to ensure we don't get NaN evaluation metrics
als = ALS(maxIter=5, rank=10, regParam=0.01, userCol="userID", itemCol="movieId", ratingCol="ratings",
          coldStartStrategy="drop")

#Tune model using ParamGridBuilder
# it will take long time in the cv period, so just use few parameter to try

paramGrid = (ParamGridBuilder()
             .addGrid(als.regParam, [0.01])
             .addGrid(als.rank, [10])
             .addGrid(als.maxIter, [15])
             .build())

# Define evaluator as RMSE

evaluator = RegressionEvaluator(metricName="rmse", labelCol="ratings",
                                predictionCol="prediction")

# Build Cross validation
# Create 5-fold CrossValidator
# it takes too long that I only use 2-fold
cv = CrossValidator(estimator=als, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=2)

# Run cross validations
cvModel = cv.fit(ratings_df)
# this will likely take a fair amount of time because of the amount of models that we're creating and testing
```

```
# Extract the best model selected by CV
```

- By the best model obtained from the above step, making predictions of ratings on movies in the test set and calculating the RMSE to evaluate the model performance are preparing for the next step.

Model testing

```
#Generate predictions and evaluate using RMSE
predictions=best_model.transform(ratings_Testing_df)
rmse = evaluator.evaluate(predictions)
print ("RMSE = "+str(rmse))
```

```
RMSE = 0.8539641229349001
```

```
#Extract best model from the tuning exercise using ParamGridBuilder

als_best = ALS(maxIter=15, rank=10, regParam=0.01, userCol="userID", itemCol="movieId", ratingCol="ratings",
          coldStartStrategy="drop")
model = als_best.fit(ratings_df)
alldata=model.transform(ratings_Testing_df)
rmse = evaluator.evaluate(alldata)
print ("RMSE = "+str(rmse))
```

```
RMSE = 0.8389696976575549
```

- In this step, I use the prediction results by the best model to recommend movies for random userID 1744889

```
+------+-------+----------+
|userId|movieId|prediction|
+------+-------+----------+
| 15846|  17337| 7.6803985|
| 15846|  14324| 5.3869762|
| 16386|  14648| 4.0660458|
| 16386|  14361| 3.9452302|
|128389|  16467|  5.876249|
|128389|   5142|  5.842342|
|151871|  17337| 5.2391896|
|151871|   1779| 5.0404043|
|205371|  12952| 7.8549204|
|205371|   3706|  6.840836|
+------+-------+----------+
only showing top 10 rows
```

```
In [85]: df.show(10)
```

```
+--------+--------------------+
|  userId|                name|
+--------+--------------------+
|1744889|The Great Ecstasy...|
|1744889| Andromeda: Season 5|
+--------+--------------------+
```

Conclusion:

The RMSE of the best ALS model on the test data is 0.83896, indicating the model is with good performance in predicting the ratings for movies

References

https://www.jiristodulka.com/post/recsys_cf/

https://medium.com/@sajad1009/netflix-recommendation-system-with-pyspark-3c25121a9144

https://towardsdatascience.com/build-recommendation-system-with-pyspark-using-alternating-least-squares-als-matrix-factorisation-ebe1ad2e7679

https://www.elenacuoco.com/2016/12/22/alternating-least-squares-als-spark-ml/