Product recommender system using Spark

Jonas Fockstedt

May 27, 2020

Abstract

This project is a result of the Big Data Parallel Programming course at Halmstad University. The goal of the project was to construct a recommendation system on the Amazon Customer Reviews dataset. The specifications of the project included to recommend products to a customer, this would be based on which product they purchased as well as the rating they gave for that product. The achieved RMSE was 1.56. With other words, the model believes that a customer could rate a product 1.56 stars higher or lower than the actual rating. This is not the best recommendation system due to a strong bias since many customers are giving products a rating of 5. But it can provide mediocre recommendations that most of the time are sufficient. For future recommendation systems, developers can instead use Spark 3 with GPU-acceleration from Nvidia as well as their Marlin framework for more efficient development.

Contents

1 Introduction

In the course Big Data Parallel Programming at Halmstad University, a project is part of the examination process of the course. The aim of the project is to let the student apply their knowledge gathered from previous exercises in the Apache Spark framework[1]. The student was given free hands on coming up with a project of their own, given an approval from the course supervisors. The only requirements that had to be fulfilled where that the project code had to utilize the Apache Spark framework, run on a cloud platform and that the results would be written to a file on the platform.

The goal of this particular project is to build a product recommendation system using Spark. The data is given from Amazon Customer Reviews Dataset[2], where a given customer (only represented with an ID) has left a review on a given product. Some of the attributes in the table are the marketplace, customer ID, product, if the purchase is verified, the rating of the product given by the customer, the review text, among others.

Today, many industries and businesses rely on sophisticated recommendation systems. Streaming sites recommend their customers new content to consume based on what they have watched in the past, and e-commerce wants to recommend their customers new products based on their previous purchases. In fact, this is such an important source of income for corporations that they have held competitions[3] where a recommendation algorithm had to be developed.

The structure of this report is as follows - in section 2 the Spark framework will be introduced, among the basic principles of a recommendation system. It will also present the structure of the data used in this project and will refer to related work. The methodology used throughout the project is specified in section 3 and the results achieved from that will be presented in section 4. Lastly, discussions about the achieved results will be discussed in section 5 and conclusions will be drawn in section 6.

2 Background

This section will present the Apache Spark into further detail as well as describe methodologies for building a recommendation system using collaborative filtering. It will also present the data set itself and describe the different attributes and will lastly bring up some related work.

2.1 Apache Spark

As a response to the growing volumes of data in the industry, many technologies have been developed to handle this problem. One of them is the Apache Spark framework[1], which was developed to bring a unified analytics engine for distributed data processing. The programming model of Spark reminds of the MapReduce framework[2], but extends this principle with Resilient Distributed Datasets (RDDs). The idea of RDDs is that the data contained within a RDD can be distributed between multiple systems, or *nodes*, while also allowing for fault tolerance.

There are many use cases for the Spark framework. The most popular is to use it for batch processing, where the task usage of Spark has been used for graph mining at Alibaba, recommendation at Yahoo!, managing data lake at Goldman Sachs, even the Chinese social network giant Tencent uses Spark, where they ingest around 1PB of data per day. Spark is also popular for processing data in real-time and for scientific research.

2.2 Collaborative filtering

A common practice for recommendation systems is *collaborative filtering*[3]. Recommendation systems who are based on collaborative filtering base their recommendations on explicit feedback received from a given user. This type of feedback is based on directly asking the user to rate a certain item. A scenario could be that a user rates a movie or product. In times where explicit feedback is not available, the *cold start* problem[4] occurs, where some users give few ratings (or they might use a new account). For this, one

[1] https://spark.apache.org/
[2] https://registry.opendata.aws/amazon-reviews/
[3] https://netflixprize.com/index.html

can take use of implicit feedback instead. This sort of feedback is data collected from a user's behavior, which in return potentially tell about their preferences. An example of implicit feedback is a user's browsing history, search patterns or purchase history.

The main principle of collaborative filtering is to find users who can be seen as "similar", and grouping them together. This is done by creating user-item pairs, where each user is paired up with items that they have interacted with. When all users have got their item pairs, the system tries to associate users with one another, based on the items they have been interacting with. This could be imagined as a situation between friends - George really likes a movie and therefore recommends it to his friends, which then are more likely to watch it, given that they previously know that they have the same movie taste as George. This also works in the opposite way, if George dislikes a movie, then he will recommend his friends not to watch it. But lets say that Linda does not agree with George's taste, but have a more similar taste as Tim. Then Linda will weigh Tim's recommendations more than George's. This scenario is what collaborative filtering tries to capture, as displayed in Figure 1.

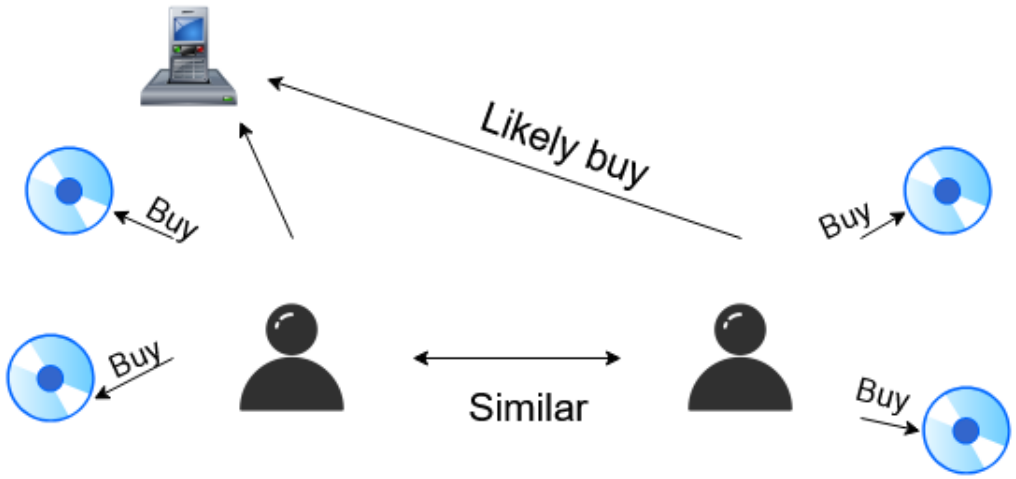

Figure 1: The principle of collaborative filtering. Seeing two users as "similar" can enable assumptions that if two customers buy the same products, then one customer might like something that the other customer has purchased.

2.3 The data

The data set for this project is supplied by Amazon and contains user reviews on their products. The data set is also divided into smaller samples, where each sample specifies the category of the product, one sample for wireless products, one for mobile, one for books, and so on. For this project, the mobile electronics sample has been chosen.

All samples from the original data set contains the same attributes, the only property that changes is the product category. The different attributes are the following:

- marketplace - Describes the market with a 2 letter country code.
- customer_id - Random identification number that is unique for each customer.
- review_id - Random and unique ID for the review.
- product_id - Unique ID for a given product.
- product_parent - Identifier used to aggregate reviews for a given product.
- product_title - Title of the product.
- product_category - Category of the given product.
- star_rating - The rating of the review, ranges from 1-5.

- helpful_votes - Number of helpful votes.
- total_votes - Total votes the review has received.
- vine - Review was written by a trustworthy reviewer.
- verified_purchase - The purchase of the reviewed product has been verified.
- review_headline - The review title.
- review_body - The review text.
- review_date - The date of when the review was written.

For this project, the most significant attributes are the *customer_id*, *product_id*, *product_title*, *star_rating* and *verified_purchase*. Different customers have to be identified so recommendations can be made for each customer, and the ID and title of a product is necessary in order to make a translation from ID (which the algorithm will work on) to title (which is visible for the customer). It is also vital to know how a given customer rated the different products, as this takes a huge role when recommending new products to customers. It is also helpful to know that a review has been on a product where the purchase has been verified. This strengthens the validity of the rating.

2.4 Related work

There have been numerous other projects whose purpose has been to implement a recommendation system, but most of them have been focusing on movie recommendations. However, the principle still remains the same - to give recommendations to a given user based on the behaviour of other, similar users with the collaborative filtering principle.

As mentioned in section 1, Netflix held a competition in 2009 where the goal was to improve their recommendation system. The winning team managed to improve the algorithm by 10.06%[5]. This was achieved by collaborative filtering as well as matrix factorization, which can be seen as a standard procedure when it comes to recommendation systems[6][7][8].

3 Method

This section brings up the methodology used in order to achieve the results. It will describe the Google Cloud Platform and its role in the project as well as the Jupyter project. The Alternating Least Squares algorithm will then be described on how it can be used for recommendation system. Lastly, value conversion of some of the attributes in the data set will be motivated and the training process of the model will be described.

3.1 Google Cloud Platform

As previously mentioned, a requirement for this project was to utilize a cloud platform. For this project, the Google Cloud Platform (GCP)[4] was chosen. The students were also offered a $50 coupon to use on the platform. By using a cloud platform, developers can instead focus on the important aspects of their applications - writing code.

In order to use Apache Spark on GCP, one can use the Dataproc[5] API that is available on the platform. Before using the API, one has to create a bucket (storage reservoir). When that is set up, a cluster can be created with the Dataproc API where the virtual machines (VM's) can be configured to meet the performance demands that is required for the given project. When creating a cluster, an option is to install Jupyter compatibility, which has been done for this project. More on that in subsection 3.2.

During the cluster creation on GCP, the master node was chosen as a VM which holds 2 vCPUs (virtual CPU) and 7.5GB of memory. For the worker nodes, two identical VM's as the master node where set up (2 vCPUs and 7.5GB of memory).

[4]https://cloud.google.com/
[5]https://cloud.google.com/dataproc/

3.2 Jupyter

The Jupyter project[6] is an open-source project for interactive data science and scientific programs. For larger applications, it is a great tool to use since it divides up the code into different segments, called *cells*, and each cell can be run independently. But a later cell can not use variables from previous computations if not the previous cells have been run before it. Thanks to the cell structure, a heavy computation does not have to be run every time something has to be checked, the result can instead be stored in a variable which can be used in later cells. This ends up saving lots of time considering the amount of time it would take to run a whole application for each new update, where training and testing the model can take some time.

3.3 Alternating Least Squares

Alternating Least Squares (ALS) is a matrix factorization algorithm[9] that is used for collaborative filtering when building recommendation systems. While given a user matrix U and a item matrix I, a rating matrix R can be calculated by the dot product $r_{ui} = u_x \times i_y^T$, where u_x is a vector which represents the interest a user u would have on a given item i, and i_y is a vector which measures the extent of how much the item possesses those factors. The rating matrix holds information of what a given user u would rate an item i. ALS operates by alternating between calculating the least squares problem for U, for a fixed I, and then for I, for a fixed U. This process is repeated until convergence, or after a set of iterations.

For systems that can perform parallel computing, the ALS algorithm is seen as a reliable and scalable method. This due to that each u_x is computed independently of the other user factors, and each computation of i_y is independent of all other item factors. This makes a perfect fit for the Spark framework. So much that the ALS algorithm has been implemented into the MLlib package[7]. This means that straight out of the box, the ALS algorithm will take advantage of the parallelism that the Spark framework provides.

The ALS algorithm in Spark lets the developer specify the column in which it will treat as the user column, the item column and the rating column. Conveniently, the current names of the columns makes the decision straightforward - The *customer_id* will be the user column, *product_id* the item column and *star_rating* the rating column. However, since all attributes are of type string in the data set, and the model only works with numerical attributes, they have to be converted.

3.4 Converting to numerical values

Many machine learning algorithms prefer to work with numbers over text. In the data set that has been picked for this project, all attributes are of the type string. In order for the ALS algorithm to work as intended on the data, there has to be a conversion from text to numerical representation. The StringIndexer[8] method is the perfect candidate for the task. The method works by specifying which attribute to convert into numerical values, and what the new attribute should be called. When executing, it transforms strings into labels in numerical form and by default labels the most frequent string at index 0, the second most common with index 1 and so on. This process is applied to the *customer_id*, *product_id* and *star_rating* attributes. The corresponding numerical attributes will be kept alongside the original valued attributes and will have the term *_num* added to them. The new attributes are consequently *customer_id_num*, *product_id_num* and *star_rating_num*. These new attributes are the ones to be selected as the user, item and rating columns for the ALS model.

3.5 Training the model

In order to make predictions with the ALS model, one has to first train it on a subset (or whole) part of the data set. For this project, the model will be trained on a subset of the original one. After this, a test will be done using cross-validation to perform tuning of the hyper-parameters. When the optimal hyper-parameters have been found, then it is time to make recommendations for all the customers, where one can specify the amount of products to recommend for each customer.

[6] https://jupyter.org/

[7] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.recommendation.ALS

[8] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html?highlight=stringindexer#pyspark.ml.feature.StringIndexer

4 Results

The data set was split in a 80/20 portion, where the training set consisted of 80% of the original data set, and 20% for the test set. Hyper-parameter tuning was applied to the test set with 4 folds. The hyper-parameters that where tested was the maximum number of iterations, the regularization parameter as well as the rank parameter (describes the dimensions of the U and I matrices). Among the tested values, displayed in Table 1, the most optimal values found were 10 for the rank parameter, 0.1 in regularization value and 15 max iterations. With this, the achieved RMSE (Root Mean Squared Error) was 1.56. This means that the model believes that a given user would rate a given product 1.56 stars more or less than they actually will, since the model uses the star rating column as the label column. In Table 2, a small subset of the original data set is displayed where the predicted values are compared to the actual values. The model is trying to predict the value of the *star_rating_num* attribute and the *star_rating* attribute is the actual value that the customer left on the review.

When the testing and verification of the model was complete, a simulation was made of a real-life scenario. A customer has bought the product *2-Port USB Car Charger Adapter*, and when they left a review for the product, they rated it with 5 stars, indicating that they loved it. Based on this, the system recommended 5 other items for this user, which took the bought product as well as the user rating for that product into consideration. The recommended products where the following:

1. iXCC Lightning Cable 3ft, iPhone charger, for iPhone X, 8, 8 Plus, 7, 7 Plus, 6s, 6s Plus, 6, 6 Plus, SE 5s 5c 5, iPad Air 2 Pro, iPad mini 2 3 4, iPad 4th Gen [Apple MFi Certified](Black and White)
2. New Trent Easypak 7000mAh Portable Triple USB Port External Battery Charger/Power Pack for Smartphones, Tablets and more (w/built-in USB cable)
3. iXCC Multi pack Lightning cable
4. Bluetooth Receiver, Breett Bluetooth 4.1 Receiver, Multipoint Connection Bluetooth Audio Music Receiver with 3.5mm AUX Port Hands Free Calling for Car Stereo/Home Stereo/Headphone et
5. eForCity Leather Case for Barnes and Noble Nook / Nook Color, Black

parameter	value
rank	1, 5, 10, 20
maxIter	5, 10, 15
regParam	0.001, 0.01, 0.1

Table 1: Different hyper-parameter values that where tested for hyper-parameter tuning.

customer_id	star_rating	star_rating_num	prediction
8282048	5	0	0
25660462	3	3	0
25012923	5	0	0
9025358	5	0	0
16381336	3	3	0

Table 2: A small subset of the data set where the predicted rating score is compared with the actual value. *star_rating_num* is the value that is trying to be predicted, *star_rating* is for reference to the actual value that a customer left on the review.

5 Discussion

Despite the relatively high RMSE score, it it still believed that the model can be useful. For example, if a customer is predicted to rate a product with 4 stars, the actual rating could vary from 3-5. But the system could still be confident in believing that the customer would like the product, and absolutely not hate it. The same goes for if the predicted rating would be 2, then the customer would certainly not like the product.

The results from the simulated scenario in section 4 seems reasonably accurate. If a customer would buy a charging adapter for their car, then they will most likely buy additional charging cables for that adapter. This is what was recommended as option 1 and 3 for the customer. Option 4 also seems reasonable for a customer who just bought mobile equipment for their car. If they buy a charging adapter, then it may be more likely that they use their phone in the car to play some tunes and/ or make phone calls when driving. However, the 3rd and 5th option does not typically relate to a customer who has bought a charging adapter for their car.

One of the hyper-parameters for the ALS algorithm is the number of iterations to make during the computation of the least squares between the u and i matrices. It was found that the iteration amount 20 made the Spark application crash while performing cross-validation, both on the GCP cluster as well as on a local machine. This may be due to performance issues in the application, the algorithm itself, or that the VM's on GCP were not powerful enough. Using a free trial account, the cluster was configured with the most powerful VM's available before the customer has to enable billing on their GCP account. And due to that this has not been a private project, there was of no interest to spend personal money in order to (potentially) achieve better results.

One of the main reasons for the relatively high RMSE presented in section 4 is probably due to the distribution among the different ratings, as shown in Table 3. As with many other 1-5 rating system, the distribution is formed in a J-shape[10], where a rating of 1 is the third most popular option whilst ratings 2 & 3 are less popular. Then a rating 4 is noticeably more popular than the previous ratings, and a rating of 5 is significantly more common than a rating of 4. In this case, a rating of 5 is almost 3 times as common as a rating of 4 and is representing 49.8% of the total amount of ratings. This leads to a bad representation of the real preference a given customer would have towards the given product due to that they tend to exaggerate their opinions due to a bias that occurs when purchasing an item. If a customer spends money on a product and likes it, they tend to think that the product is a great value for money. On the other hand, if they spend money on something that they do not like, they tend to become more disappointed with the product since they spent money on it, hence that a rating of 1 is almost as popular as a rating of 4. This is believed to have lead to a strong bias in the trained model, making it almost always think that a customer will rate an item with 5 stars.

star_rating	count
5	52255
4	18088
1	17587
3	9734
2	7311

Table 3: Number of ratings for each rank.

The results from this project could have been different based on a few factors. First, using a data set which only allows the customer to give a rating of thumbs up/down, as Netflix or Youtube, or a rating system which only has a like button, like Facebook or Instagram. This would most likely prevent the problem that arose here where a substantial part of the ratings where 5 stars. Second, having access to more computing resources would also benefit since Spark crashed when 20 max iterations where set for the ALS model. There might have been a better set of hyper-parameters that were just out of reach for this project. Third, the ALS algorithm could benefit from cooperating with other models, making for an ensemble machine learning model. In this way, the different models would hopefully outweigh each others weaknesses.

5.1 Future work

According to Nvidia founder and CEO Jensen Huang, the Spark framework is starting to crumble under the vast amounts of data that flows in systems as it is exponentially growing. Since Spark 2 works on CPU resources alone, it is becoming a heavy burden for the framework to try to deliver its promised performance. However, on Nvidia's GTC 2020 press-conference, Jensen announced that Spark 3 will utilize GPU-accelerated computing powered by Nvidia[9]. This will give a significant performance boost on executing jobs since a CPU typically work with 10-100MB's of cache memory, whilst a GPU nowadays can work

[9]https://www.nvidia.com/en-us/deep-learning-ai/solutions/data-science/apache-spark-3/

with 10-100GB's of internal memory when also providing multi fold the amount of cores. When developing similar projects like this one in the future, the development time would most likely reduce significantly and more analytics could be made on the data. In fact, at GTC 2020, Nvidia also announced Nvidia Merlin[10], which is a framework for building complex and vast recommendation systems. In the future, many systems will most likely take use of this framework when building recommendation systems.

6 Conclusion

The purpose of this project was to bring a recommendation system which was developed in the Spark framework and based its recommendations on the Amazon Customer Reviews Data set. It was achieved by using the principles of collaborative filtering and the ALS algorithm in order to deal with "similarity" between different customers as well as dealing with a sparse number of ratings. The results achieved from this work does not fulfill the most accurate predictions, but can be confident in that a customer would like a product if the predicted rating is 4 and not like it if it would be 2. Additional work could consist of utilizing Spark 3 with Nvidia GPU-accelerated computing as well as Nvidia Merlin to achieve better results as this would most likely have solved the bottleneck at 20 max iterations on the ALS model.

To conclude, results from this project does bring a recommendation system. However, it could (and should) bring better predictions to customers since many industries rely on this technology in order to make for a good customer experience as well as generate profits.

[10] https://developer.nvidia.com/nvidia-merlin

References

[1] Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, et al. Apache spark: a unified engine for big data processing. Communications of the ACM. 2016;59(11):56–65.

[2] Dean J, Ghemawat S. MapReduce: a flexible data processing tool. Communications of the ACM. 2010;53(1):72–77.

[3] Hu Y, Koren Y, Volinsky C. Collaborative filtering for implicit feedback datasets. In: 2008 Eighth IEEE International Conference on Data Mining. Ieee; 2008. p. 263–272.

[4] Lam XN, Vu T, Le TD, Duong AD. Addressing cold-start problem in recommendation systems. In: Proceedings of the 2nd international conference on Ubiquitous information management and communication; 2008. p. 208–211.

[5] Koren Y. The bellkor solution to the netflix grand prize. Netflix prize documentation. 2009;81(2009):1–10.

[6] Koren Y, Bell R, Volinsky C. Matrix factorization techniques for recommender systems. Computer. 2009;42(8):30–37.

[7] Schafer JB, Frankowski D, Herlocker J, Sen S. Collaborative filtering recommender systems. In: The adaptive web. Springer; 2007. p. 291–324.

[8] Zhou Y, Wilkinson D, Schreiber R, Pan R. Large-scale parallel collaborative filtering for the netflix prize. In: International conference on algorithmic applications in management. Springer; 2008. p. 337–348.

[9] Takács G, Tikk D. Alternating least squares for personalized ranking. In: Proceedings of the sixth ACM conference on Recommender systems; 2012. p. 83–90.

[10] Hu N, Zhang J, Pavlou PA. Overcoming the J-shaped distribution of product reviews. Communications of the ACM. 2009;52(10):144–147.