Jack Jakober

Dr. Pallickara

CS 435

17 November 2023

Product Recommendations for eCommerce Website

Introduction:

Online shopping has become a significant source of income for companies in recent years. Customers are finding and purchasing items exclusively online and it is important that customers are able to find the products they want. There are many factors that determine what products a customer might want to buy. Some products might be purchased impulsively while others might be purchased because the customer is already familiar with the product. In an online landscape, customers are mostly likely only going to see what the website thinks is relevant to them. Online stores need a way to determine what products a customer might want to purchase.

The goal of this project is to analyze E-commers data from an online store and create product recommendations for each customer based on their purchase history. It is important that users are recommended products that they actually want. If users are recommended products that they are not interested in, then they might stop looking at their recommended products in the future. On the other hand, if a user is recommended a product that they like, they will be more likely to look at and purchase their recommended items in the future.

It is clear that quality recommendations can be an invaluable tool for getting people to buy more. Unfortunately, the mind of a customer cannot be read. Instead, the interests of a customer must be determined from limited, unexplained data. In this case, the only measured interaction that a customer has with a product is whether they view, add it to their cart, or purchase it. This is an interesting Big Data problem because each interaction gives no information about how a customer feels about a product. For example, a customer might view a product accidently or purchase a product as a gift for someone else. If this problem were solved, it could give insight into how people think and interact with the world. Online stores usually rely on customer reviews to determined how a user feels about a product. If stores could estimate how a customer feels about a product before they write a review, then similar products could be recommended to the customer much sooner.

There have been many different approaches to recommendation systems. One such approach is Content Filtering, where a product is classified based on its features (5). This approach generates recommendations by recommending products that are similar to what a customer has already purchased. Another approach is Collaborative Filtering, which this model uses. Instead of calculating which products are similar to each other, this approach calculates which users are similar to each other (5).

Methodology:

As mentioned above, this model uses Collaborative Filtering to provide recommendations for each user. We want to figure out if a customer is likely to purchase an item that they have not purchased before. To do this, we need to determine if similar customers have also purchased the item in question. If two customers have similar purchase history, then it is likely that customer A will like most of the things that customer B has purchased. So if customer B has purchased an item that customer A has not, there is a good chance that customer A will purchase the item if they are recommended it. I decided to use collaborative filtering as opposed to another approach because customers purchase a wide variety of products, if two customers are similar to each other and the model works well, then they will get a wide variety of recommendations. This is the overall logic behind how collaborative filtering works, to implement it, a technique called matrix factorization is needed.

Matrix factorization is used to calculate similarities among customers. The data is represented as a matrix A where the rows are the customers, and the columns are the products. A given customer-product entry in this matrix represents how a customer feels about a certain item on a scale of 1 to 5 (I will explain how these ratings are calculated later. This matrix is spare, meaning it has a lot of missing entries because each customer does not purchase every single item. Matrix factorization is then performed in order to generate estimates for all of the missing entries in the table. After all of these estimates are generated, users can be recommended the products that had the highest rating estimates.

The algorithm used to perform the matrix factorization is known as the alternating least squares algorithm, or ALS for short (1). In the algorithm, a user matrix X, and a product matrix Y are created and each value in these matrices represent a latent feature in the model (the latent features are unknown features in the data that correspond to how a user feels about a movie). These two matrices are then multiplied together to fill in the previous sparse matrix A.

The first step of the ALS algorithm is to compute the X matrix (4). This is done by initializing the Y matrix with random values and solving for the best X given Y and A. The best X in this case is the X that minimizes the sum of the squared distances between the matrices. The next step is to compute Y from the X computed in the previous step (4). The logic here is the same as the previous step except for the fact that X is now known. These steps are then repeated (X is calculated with Y and Y is calculated with X) until they converge into a potential solution. Once X and Y are computed, they are multiplied together in order to fill in the sparse matrix A.

This recommendation system uses the ALS algorithms provided in Apache Spark mllib. I decided to use the spark implementation of ALS because we have been using it in class and there is a lot of documentation on it. The advantage of using Apache Spark is that it can be used to effectivly handle large amounts of data.

The ALS algorithm in Apache Spark has several attributes that need to be set (2). One such attribute is the cold start strategy. The cold start strategy specifies what the model should do with purchases that the data was not trained on. This happens when users have no purchase history or when users or products show up in the test set when they weren't in the training set. The cold start strategy is set to "drop" so any rows in the predictions dataset that are not complete are simply removed. Another parameter that needed to be set was the maximum iteration parameter. This was set to 10 so the model would only do 10 iterations.

Before the data is can be used in the collaborative filtering model, it must first be reformatted. There needs to be a rating associated with each product that a customer has

purchased. The data's simple purchase and view history needs to be converted into a number from 1 to 5 that represents how much a customer likes a product. The 1 to 5 rating scale is straight forward, 1 means a customer does not like a product, 3 means neutral, and 5 means a customer likes a product. These numbers are calculated by adding up the total number of views or purchases that a customer has for each product. 0.2 is added to the total every time a customer views a product and 1.0 is added every time a customer purchases a product. If the total is around 1, it means that the customer feels neutral about the product. This could mean that they purchased a product once or viewed it 5 times. If the total 0.4 or less, it means that the customer has viewed the product once or twice but never bought it. This signifies that a customer probably does not want a product. If the total is higher than 5, it means that a customer really likes a product as they have purchased it multiple times. Overall, the total is converted into a rating on a 5-point scale based on the logic above.

Once each customer is given an artificial rating for each of the products they have bought, the ALS recommendation model can be used. First the data is split into a training and testing set where 80% of the data goes to a training set and 20% of the data goes to a testing set. The ALS model is then ran on the training data where it can be cross validated using the test data after. The ALS function, recommend for all users function can then be called in order to generate 10 product recommendations for each user.

The processing of the data and running of this model is being done in parallel on several computers using the Apache Spark framework. The data is first loaded into a JavaRDD where the total amount of views and purchases that a customer made on each product is calculated. Once this value is calculated, it can then be converted into an artificial 1 to 5 star rating. Each line is then mapped into an object that consists of a customer, a product, and the rating. The RDD is then loaded into an Apache Spark data frame that is run on the model.

Dataset:

The data set that I am using is a collection of customer behavior data from an undisclosed eCommerce website. This dataset is opensource and can be found on Kaggle. Each row in the data represents an event that has occurred, and each column represents the attributes of that event. The original data set has columns for event type, product id, category id, category code, brand, price, user id, and user session. For the sake of this model, the only data points taken from this model are event type, user id, and product id. Since this model uses collaborative filtering, most of the attributes of this dataset become irrelevant. If this model instead used content filtering, the attributes involving brand, category, and price would be extremely useful. The original dataset is split into two separate files, one corresponding to data from November of 2019 and the other corresponding to data from October of 2019. This project only uses data from the October file. This file is 5.67 gibibytes and is in csv format. Overall, this dataset is very good. There are no missing entries or duplicates, and it is a very popular data set for machine learning.

The event type variable can either be either view, cart, or purchase. This is implicit feedback as opposed to explicit feedback. Implicit feedback represents quantitative measurements of a users behavior such as views, likes, or shares. Explicit feedback represents a customers rating of a product and can directly show if a customer likes or dislikes a product. The main difference between explicit and implicit feedback is that explicit feedback is based on a response from the customer while implicit feedback is based on the customers behavior (3). The event type variable was interpreted as representing a customer clicking on a product. Essentially a view or purchase means that a customer clicked on the product. As explained above, the total

number of clicks that a customer has for each product is added up and then converted into a rating from 1 to 5 as mentioned above. This conversion from the given implicit feedback to artificial explicit feedback makes it possible for the data to be run using the Apache Spark ALS algorithms. It is important to note that the ALS algorithm does have a method for using implicit data, however this method was not used as it makes the model more difficult to test.

It is important to note that a user can have multiple purchases in a single session. The first and last line of the file had to be removed for the data to be processed.

Link to dataset:
https://www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store/data

Discussion and Analysis

After running the model, predictions about the rating that each customer would give an item were generated. A list of ten products was generated for each customer by taking the 10 products with the highest predictions. The results show that most customers are recommended items with predicted ratings higher than neutral. The results are in the form [userId, {productid, rating}, {productid, rating},…]. One unfortunate thing about these results is that it is hard to intuitively gage weather a recommendation makes sense because the products are labeled only by their product id's as opposed to a name or description. If this dataset had product names, I could look at all the products that a customer has purchased and their recommendations and put myself in the customers shoes. For example, if a customer bought a bunch of exercise equipment, it would make sense for them to be recommended products from a similar category. There are many ways to see if a model is accurate but I think that if I could actually see what products are being recommended then I could be a lot more judgmental of my model.

These recommendations succeed at giving users items that have high predicted ratings. However, these recommendations mean nothing unless the predictions are accurate. The predictions were analyzed using Apache Sparks Regression Evaluator class. Since the data was split into a training set and a testing set before the model was ran, it is possible to compare the predicted ratings to the real ratings in the testing set. The regression evaluator object can be used to give a root mean squared error (RMSE) for the predictions. The RMSE represents the average distance between the predicted value and the real value. The RMSE for this model was 0.267. The possible values for the rating are 1 through 5 so the 0.267 means that this model is decently accurate.

Overall, I am pleased with the results of my model. I think that it generates product recommendations that users will be likely to buy. However, I did run into several challenges while working through this project. The first and most obvious one is that my group split up. Instead of working on this project with two other people, I was all the sudden on my own. This increased up the workload and made the project more difficult. I think that the challenge of doing this by myself helped me gain better understanding of machine learning and Apache Spark. The most significant challenge I faced while doing this project was trying to figure out what to do with the implicit feedback in the data. I knew that the spark algorithms could deal with implicit data (you can set the parameter implicit preferences as true), and I spent a ton of time trying to wrap my head around their implementation. I got to the point where my model was working with the number of clicks functioning as the rating. However, the predictions that the model generated were in the form of confidence values that show how likely it is that a customer will like a

project. I did not know how to compare the predicted value (confidence value) to the real value (number of clicks) and could not test my code. My solution to this problem was to convert the number of clicks into a rating. This way I could evaluate my results and not have to think about how the ALS algorithm handles implicit data. Aside from the implementation, the hardest part of this project was figuring out to do. I struggled to find datasets and come up with something to do with them when I was in my team. Another challenge I faced was figuring out how to view my results. I would prefer to do this in Python using the math plot library. I really wanted to have a graphical representation of the predicted values and real values. However, I thought that moving my data to Python seemed like a bad solution, I also could not find any mathplotlib alternative in java. I also found it difficult and tedious to print and view result dataset. The Spark Dataset was hard to output to a file and hard to view exactly what I wanted. All in all, this project was challenging but also a great learning experience.

Project Contributions:

As I mentioned before, my group split up and I worked on this project entirely alone. However, I did use the same project proposal that I used with my group. I found the eCommerce data set that we wanted to use and we discussed what we should do with it as a group. We determined that a recommendation system would be a good idea. Besides the initial forming of the project idea, I did not work on this project with any of my former team members. Since I am working alone, writing the final report seemed like a very daunting task. I think that if I worked with other people on the report, it could be of much higher quality. For example, there is a lack of visuals in this report. If I worked with even one other person, we could divide up the work such that one person writes and one person works on visuals. I think that visuals would really help to demonstrate the algorithm I used as well as the evaluation of my model. However, I had to reduce the scope of my project because I am working alone, unfortunately this meant sacrificing the visual aspects of this report to focus more on the written part.

Bibiliography:

[1] “Build a Recommendation Engine with Collaborative Filtering.” *Real Python*, Real Python, 18 Aug. 2022, realpython.com/build-recommendation-engine-collaborative-filtering/#:~:text=Collaborative%20filtering%20is%20a%20technique,similar%20to%20a%20particular%20user.

[2] “Collaborative Filtering.” *Collaborative Filtering - Spark 2.2.0 Documentation*, spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html. Accessed 2 Dec. 2023.

[3] Hu, Yifan, et al. *Collaborative Filtering for Implicit Feedback Datasets | IEEE ...*, ieeexplore.ieee.org/document/4781121. Accessed 3 Dec. 2023.

[4] “Matrix Factorization Explained: What Is Matrix Factorization?” *Great Learning Blog: Free Resources What Matters to Shape Your Career!*, 31 Oct. 2022, www.mygreatlearning.com/blog/matrix-factorization-explained/.

[5] “What Is a Recommendation System?” *NVIDIA Data Science Glossary*, www.nvidia.com/en-us/glossary/data-science/recommendation-system/#:~:text=A%20recommendation%20system%20(or%20recommender,exponentially%20growing%20number%20of%20options. Accessed 2 Dec. 2023.