Developing Recommendation System for Amazon Product

Milestone 4: Final Project Report

Group1

1.0 Introduction

Out team has focused our project on an Amazon dataset [1] comprised of ratings from over 200 million reviews of products across various categories. Given the large size of such a data set, we have wondered how companies like Amazon utilize the sentiments of their users and are able to overcome the difficulties of using such large amounts of data. Recommendation systems are a vital component of marketplaces online and are intertwined with user profiles and preferences. Our project attempts to learn, research, and implement the recommendation systems through collaborative filtering algorithms and explain the efficacy of academic algorithms on real world data. The report we have created shows how to use collaborative filtering to provide recommendations to users across the different categories in the data set. In this way, we would be echoing the feed provided to users on their login pages or while browsing the marketplace. These filtering algorithms take on various forms and further in this paper we will elaborate on their differences and comparative advantages. Also because of this we are able to compare and contrast the performance of sequential and distributed algorithms via the different forms collaborative filtering takes. Using Apache Spark, we are able to echo the performance of these large scale data processing tasks and move beyond the examples that we have encountered in our research. Understanding these algorithms is vital for any recommendation system and having empirical evidence from real big data datasets will give vital insight to our team and report into companies like Amazon.

2.0 Related Work

When researching recommendation systems, we came upon a number of academic articles analyzing the use of such systems as well as the number of methods applied. The first chapter of a handbook [2] on recommendation systems by Ricci, Rokach, and Shapira, an Italian and Isaraeli professors in the computer science field, outlines the importance of recommendation systems and provided a foundation for why this task is useful for businesses. However, this chapter also provided an outline to the various recommendation techniques such as content-based techniques, collaborative filtering, demographic techniques, knowledge-based techniques, community-based systems, and hybrid systems composed of the previous

list. Based on our dataset it was apparent that the data was primed for useful application to collaborative filtering, and the feedback from our first milestone included as such.

Moving from an academic to technical expertise on the subject, we found a number of blog posts, tutorials, and descriptive articles on the subject that served as the primary driver of our coding and understanding. An article [3] posted by Price Grover, a research scientist at Amazon Web Services, gives a summary of the various types of collaborative filtering algorithms and included a simple yet elaborative visual for understanding them.

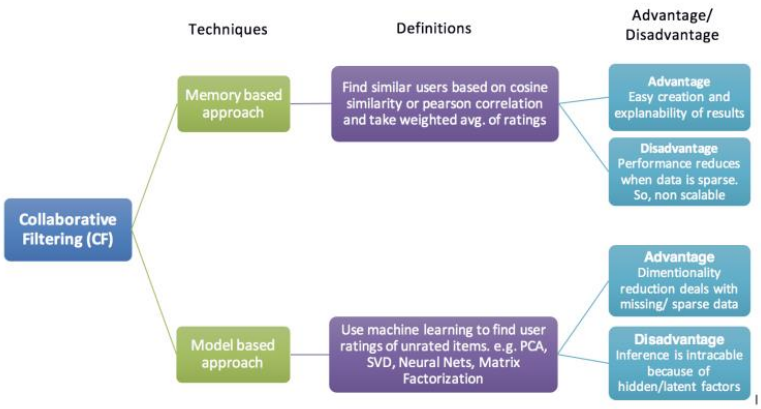


Figure 1: Types of collaborative filtering approaches

Here we see that there are two typical groupings for collaborative filtering, with memory-based techniques such as user-item or item-item filtering and model based techniques which apply machine learning via various approaches. Given the large amount of reviews on various products and the users who rated them, the memory based technique should be able to find similar users to the user in question and find an assortment of items that are new to that user that they would rate well. This approach would be the user-based filtering method. However, for item-item filtering we should be able to take the various reviews on products and find items that are also well liked with the item reviews being the focus instead of the user similarity. Both of these would perform this task by finding the cosine similarity between given vectors of reviews. This gives us a weight to discount the ratings of other users with those having more similarity having their rating contribute more to the estimated review. The summation of these and normalization gives us the rating.

$$sim(u,u') = cos(\theta) = \frac{\mathbf{r}_u \cdot \mathbf{r}_{u'}}{\|\mathbf{r}_u\| \|\mathbf{r}_{u'}\|} = \sum_i \frac{r_{ui} r_{u'i}}{\sqrt{\sum_i r_{ui}^2} \sqrt{\sum_i r_{u'i}^2}}$$

Following these techniques is the model-based approach and here we are introduced to the use of neural nets and matrix factorization as techniques to do collaborative filtering. Rather than using a distance metric through cosine similarity, the matrix factorization tries to estimate a number of embeddings that represent unknown influences upon a review. Intrinsic to this is matrix decomposition so in the case of our data, where a user may only have a few reviews, the decomposition would combat the issue of having sparse matrices. For the memory-based approach this would be an issue for the cosine similarity calculation and something we would hope to see in our data results. Finally, the article introduced us to the Surprise package which has functionality for implementing these functions and algorithms. This will be elaborated on further in the model section.

While the previous paper is a good introduction to the algorithms we would be using we needed more elaboration on matrix factorization and we found an article at datasciencemadesimpler.wordpress.com [4] that elaborated on using Alternating Least Squares and Singular Value Decomposition. The greatest take away here is that these techniques of matrix factorization allow us to perform dimensionality reduction and address the issue of having spare matrices.

Finally, it is important to note some of the shortcomings and issues surrounding recommendation systems. In an article [5] written by Kevin Liao a data scientist the article comments on ALS in collaborative filtering as well but mentions issues of popularity bias, item cold-start, and scalability issues intrinsic to recommendation systems. The cold start problem [6] reflects the sparse matrix issue where an individual that we are trying to recommend items to doesn't have an adequate amount of reviews to generate a similar user to compare to. Popularity bias reflects the issue of models simply just reflecting the most popular item, and despite complex calculations the result is just the most reviewed item. Recommender systems are meant to provide value beyond this and personalize the experience for the customer in order to provide a unique experience. Scalability issues are the result of data sets being too large to meaningful calculate and reflect an experience that is enjoyable. By analyzing the distributed algorithm detailed later, we hope to better understand this issue.

3.0 Algorithms, Method, and Model

The articles that we read were informative but did not expand upon the use of these algorithms on large data. Since our data is composed of hundreds of thousands of reviews, we are fortunate to have this to challenge the algorithms and compare them.

3.1 Data preparation

Our first task is to parse the data held in JSON files. The code for this is held in a screenshot in the appendix and it details how we converted the json files into easily read csv files. CSV files are easily read into data frames by the pandas library and are fundamental for use of the algorithms. While our dataset is composed of several attributes of metadata we are only focused on the user_id of the person doing the review, the review rating, and asin which corresponds to the item identification number. Some collaborative filtering algorithms consider context of the user detailing attributes that could be used to associate users to one another, but this was not required by the algorithms we chose to compare.

3.2 Sequential Algorithm

Our choice of sequential algorithm was instead a choice of comparing a few of the algorithms we found in our research. Item-item technique was used for performing our memory-based sequential algorithm. With this approach we used the cosine similarity coupled with K-Nearest-Neighbors to perform collaborative filtering. For user-time technique we were able to perform similarly, but instead used a person correlation to find similar users and consider their ratings to generate a recommendation. For model-based sequential algorithms we used both Singular Value Decomposition and Alternating Least squares. The Surprise package [7] in python was used for these three collaborative filtering algorithms. This package is specifically for recommendation systems and has built in algorithms for performing each. With these algorithms we were able to train against 80% of the data set for particular categories of items. With the trained data, we were able to then run the test set against the holdout and obtain metrics for RMSE, MSE, and MAE that will be shown in the results section of this report. Each result is then followed by taking a particular user and running a prediction for that user. We return a top 10 items for that user in the category. This is to echo the search results or automatic marketing of items to a user in the marketplace.

3.3 Distributed Algorithm

For the distributed algorithm we performed ALS on the Amazon data. To perform the distributed algorithm, we computed this on a Spark Cluster [8].

Start up spark cluster

```
appName="Collaborative Filtering with PySpark"
# initialize the spark session
spark = SparkSession.builder\
    .appName(appName)\
    .config('spark.driver.memory', '15g')\
    .getOrCreate()
# get sparkcontext from the sparksession
sc = spark.sparkContext
```

The data is loaded into a spark data frame, which allows the data to be operated on in a parallelized way. The spark cluster allows tasks performed on the data to be performed on separate threads and has a driver scheduler to optimize the tasks. For ALS this would translate into separate threads performing the matrix algebra for determining the latent factors that summarize the information from a large sparse matrix into more succinct matrices. As well as this, ALS performs gradient descent in parallel [5] as it trains the data.

3.4 Hyperparameter Tuning

In order to increase the accuracy of our model, we had to tune our hyperparameters. We performed grid search on a number of hyperparameters in order to get the lowest testing error. The below graphs show how the training and testing errors changed when the regularization parameter, and the cap on max iterations changed:

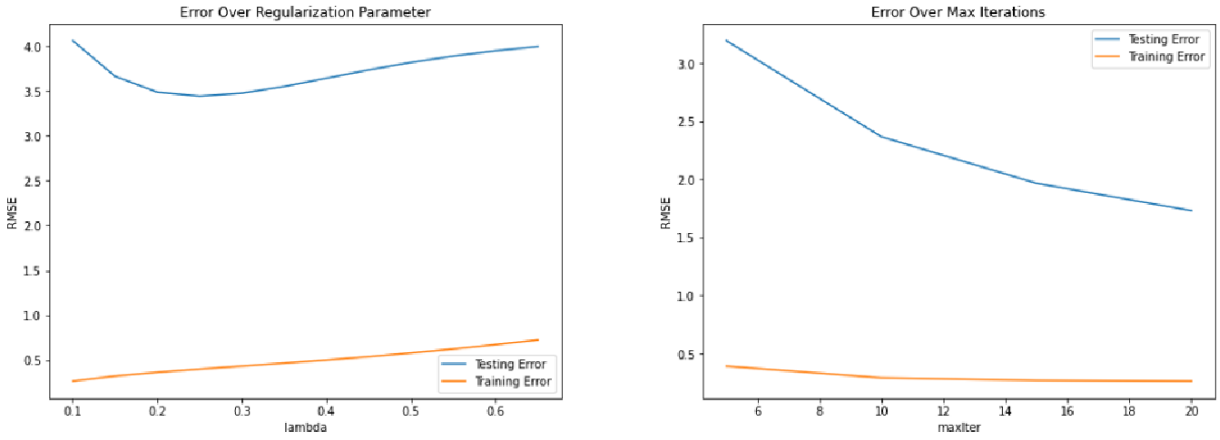


Figure 2: Accuracy statistics for the hyper tuned models

We were able to decrease RMSE from 3.9 to about 2 using hyperparameter tuning.

4.0 Results and Discussion

In the appendix you can find captured images of the available top rated items for a particular user. This is the result of running the distributed ALS algorithm on the All_Beauty data set. The goal of our project was to replicate the presentation of items that a user would be interested in and this visual component is an attempt of replicating this. The findings of these results indicate that while the memory-based and model-based approaches composing collaborative filtering are very useful for approximating with large data sets of reviews, the hybrid approach of combining these collaborative filtering techniques with context-based recommendation algorithms would most likely be the most useful. Our data set does not

include data on the user, but having this data would be incredibly useful for determining which kinds of sets of items within a category would be most relevant. Perphas pulbic information contained in cookies from web search history and other personal information from the user's profile would be impactful for determining a smaller set of items toconsider. In the case of hybrid approaches, the data filtering could be performed by context based algorithms and then using these items to train the collaborative filtering mechanism would be most useful. Only using collaborative filtering for determining highly rated items based on similar users or similar items is no doubt useful, but as you can see with the results it is somewhat random. However, this randomness is useful in providing a diverse set of items for the user to consider. A balance would then be needed in the hybrid model to not recommend already seen items or 'old' items to the user based on a context set and instead be able to present new and relevant items for the user experience to be maximized.

We were able to apply the ALS, SVD, and K-Nearest-Neighbor sequential algorithms to the Gift_Card, Software, All_Beauty, and Appliances datasets. In doing so we trained on 80% of the data and tested on 20%. RMSE, MAE, and MSE were used as accuracy measures on the training data and this is included in the appendix. Our findings show that the RMSE of trained models can mis-label ratings by anywhere from 1 to 3 stars. While this is not ideal, many of the reviews that are being fitted by these algorithms are dealing with very sparse matrices. Some individuals have very few reviews, so when building the review matrix for a user the values may be mostly empty, and this should explain the inaccuracy.

The below table shows the times and errors on testing set for some categories after the ALS model has been fit:

Table 1: Diagnostic measures for the fitted ALS model

Category	Users	Items	Reviews	Type	Time (s)	RMSE	MAE
Software	375147	21663	459436	Sequential	1.704	1.501	1.282
				Distributed	65.10	2.444	1.859
Gift_Cards	128877	1548	147194	Sequential	0.488	0.936	0.539
				Distributed	22.90	2.022	1.442
All_Beauty	324038	32586	371345	Sequential	1.389	1.276	0.997
				Distributed	46.60	2.793	1.821

The sequential algorithm performed better than the distributed algorithm in both the time and error dimensions. For scalability comparison we were able to run our algorithms against data sets ranging from over 100,000 reviews to 600,000 reviews. We ran into issues

with data sets comprising over 1 million reviews. We believe that the surprise package that runs these algorithms has issues with data larger than this sadly. We also believe that this may have to do with the limitation of the spark instance that we are running on since we obtain the memory errors for being unable to allocate enough memory. A screen shot in the appendix shows this. Our confusion is compounded by the fact that Scott was able to run more intensive and larger items on his local computer for a different course. However, with the data that we have the scalability appears to be linear. The below graphs indicate that with increased amount of reviews, the time to train the data against each sequential algorithm, including ALS which was used in the distributed algorithm, increases linearly and not exponentially. Because of the scale of the graph, the time values for training the sequential is difficult to see but they are linear and included in the appendix.

5.0 Conclusion

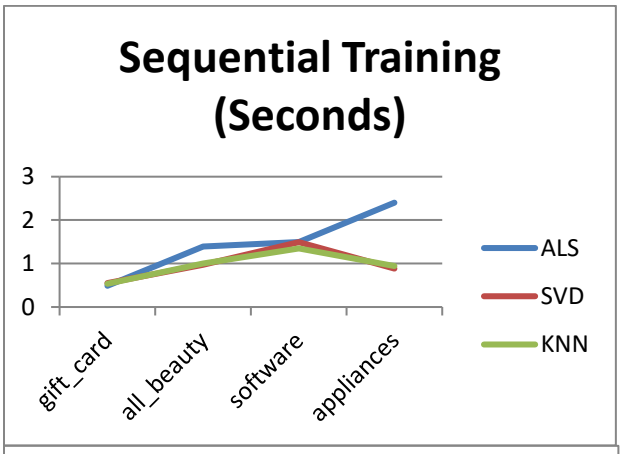
We learned a lot during this project. We started out with a very basic algorithm that performed quickly, but did not give very good recommendations. We then developed an algorithm that took a lot more time to run, but gave more meaningful results. This project not only make us more comfortable with algorithms that could be used on big data, but also also frameworks (like Spark) that could be used to process large amounts of data. One perplexing result that came from our project is that our sequential algorithms seemed to run quicker, and perform better out of sample. We were expecting the opposite result. Despite this, both sets of algorithms (sequential and distributed) ran relatively quickly, and output results that made sense.

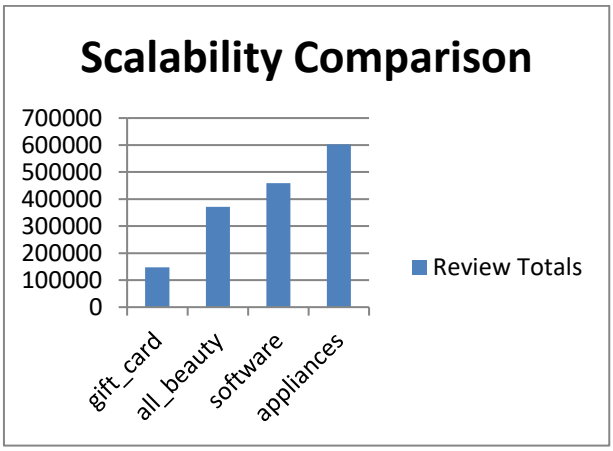
6.0 Appendix

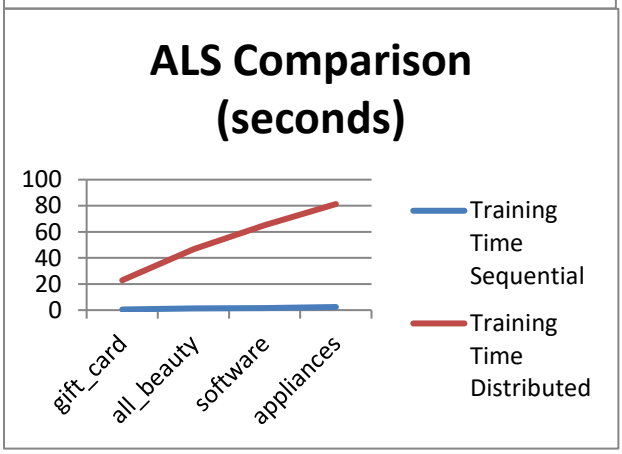
Appendix 1: GitHub Link

- Code can be found on ec2 instance as well as in the following GitHub link
https://github.com/joepatten/CPT_S_Collab_Filter

Appendix 2: Graphs on Time and Scalability Results







Appendix 3: Raw results data

data_set	algorithm	users	items	reviews	time	rmse	mse	mae
gift card	KNN-cosir	128877	1548	147194	0.53503929	0.9543	0.9107	0.535
	SVD	128877	1548	147194	0.55116	0.9397	0.883	0.5512
	ALS	128877	1548	147194	0.48815	0.9367	0.8774	0.5399
software	KNN-cosir	375147	21663	459436	1.348	1.5766	2.4856	1.348
	SVD	375147	21663	459436	1.493	2.229	1.2721	1.272
	ALS	375147	21663	459436	1.7047	1.5008	2.252	1.2829
electronic	knn	9838676	756489	20994353	x	x	x	x
automotiv	knn	3873247	925387	7990166	x	x	x	x
video_gar	knn	1540618	71982	2565349	x	x	x	x
all_beauty	knn	324038	32586	371345	1	1.3181	1.7374	1
	svd				0.97154	1.2551	1.5752	0.9715
	als				1.3896	1.2769	1.6304	0.9977
appliance	knn	515650	30252	602777	0.940498	1.2782	1.6338	0.9405
	svd				0.886989271	1.2129	1.471	0.887
	als				2.401451	1.2187	1.4853	0.8959
Digital Mu	knn	840372	456992	1584082	x	x	x	x

Appendix 4: Data Preparation

```
import pandas as pd
import gzip
import json

for f in os.listdir():
    if not f.endswith('.gz'):
        continue
    elif f.split('.')[0] + '.csv' in os.listdir():
        continue
    def parse(path):
        g = gzip.open(path, 'rb')
        for l in g:
            yield json.loads(l)

    def getDF(path):
        i = 0
        df = {}
        for d in parse(path):
            df[i] = d
            i += 1
        return pd.DataFrame.from_dict(df, orient='index')

    df = getDF(f)
    df = df[['asin', 'title', 'image']].explode('image')
    df = df.groupby('asin').first().reset_index()

    df.to_csv(f.split('.')[0] + '.csv', index=False)
```

```
random.seed(123)
mydata=pd.read_csv("/home/ubuntu/Notebooks/ratings_data/Gift_Cards.csv",names = ['item','user','rating','timestamp'])
del mydata['timestamp']
#reindex the column positions to be suitable for surprise
columns_titles = ['user','item','rating']
mydata=mydata.reindex(columns=columns_titles)
mydata.head()
```

Appendix 5: Sample Recommendations for user: 9386 in the All_Beauty category

Amazon Link of the Item that is Recommended	Item that is recommended
https://www.amazon.com/dp/B002Q5T1S4	Chap Lip 24's
https://www.amazon.com/dp/B00QWR5Y24	Crazy K&A Woman & Girl Small Wave Shape Rhinestone Headband Tiara Crown Hair Comb Pin For Wedding Party Banquet, Silver
https://www.amazon.com/dp/B014TII4II	Bundle of 12 Small Bar Soaps and Matching Soap Dish ("Lavender Rosemary" Set)
https://www.amazon.com/dp/B019A51QWG	Sol De Janeiro Samba 2-Step Foot Fetish Care 3 fl oz

Appendix 6: Issues with scalability

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-21-77437ec01630> in <module>
     20 print('Number of ratings in trainset: ', trainset.n_ratings,)
     21 #train the algorithm on the trainset, and predict ratings for the testset
---> 22 algo_item.fit(trainset)
     23 predictions_item = algo_item.test(testset)
     24 #then compute RMSE, MSE and MAE

~/.local/lib/python3.8/site-packages/surprise/prediction_algorithms/knns.py in fit(self, trainset)
    176
    177         SymmetricAlgo.fit(self, trainset)
--> 178         self.sim = self.compute_similarities()
    179
    180         self.means = np.zeros(self.n_x)

~/.local/lib/python3.8/site-packages/surprise/prediction_algorithms/algo_base.py in compute_similarities(self)
    247             if getattr(self, 'verbose', False):
    248                 print('Computing the {0} similarity matrix...'.format(name))
--> 249             sim = construction_func[name](*args)
    250             if getattr(self, 'verbose', False):
    251                 print('Done computing similarity matrix.')

~/.local/lib/python3.8/site-packages/surprise/similarities.pyx in surprise.similarities.cosine()

MemoryError: Unable to allocate 3.59 TiB for an array with shape (702800, 702800) and data type float64
```

7.0 References

[1] **https://nijianmo.github.io/amazon/index.html**

[2] **https://www.inf.unibz.it/~ricci/papers/intro-rec-sys-handbook.pdf**

[3] **https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0**

[4] **https://datasciencemadesimpler.wordpress.com/2015/12/16/understanding-collaborative-filtering-approach-to-recommendations/**

[5] https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1

[6] https://en.wikipedia.org/wiki/Cold_start_(recommender_systems)

[7] http://surpriselib.com/

[8] https://spark.apache.org/docs/latest/cluster-overview.html

Contribution of the Group Members

Name	Activities	Percentage
Md Muhtasim Billah	1. Coded and ran both sequential (User based, SVD, ALS) and distributed (ALS) algorithms. Compared the sequential algorithms by their evaluation metrics. 2. Helped with compiling milestone 2 report. 3. Wrote the entire milestone 3 report. 4. Helped prepare the demo video and slides. 5. Maintained communication among the group members.	**32**
Joseph Patten	1. **Environment**: Set up AWS ec2 instance and put on relevant program (python, java, scala, spark, jupyter notebook). Downloaded ratings and meta data onto instance 2. **Sequential algorithm**: Wrote User based Pearson Coefficient from scratch. Modified ALS algorithm to pull relevant meta data for recommended products 3. **Distributed algorithm**: Did hyperparameter tuning for model. Expanded on algorithm to pull data from meta data to include relevant product data for Amazon. Built front end for demo. 4. **Presentation**: Helped with slides, presented part of presentation, recorded demo video	**26**
Scott Walters	1. Wrote Milestone 2, Coding as well, 2. Wrote 90% of Milestone 4	**26**
Md Mahedi Hasan	1. Helping in preparing milestone 3 2. Help to compile the report 3. Preparing the slides for presentation and present 4. Attended the peer meetings for the project and participated in the discussions to way forward	**16**