Amazon Automotive Products Recommender System

Simrandeep Singh
Student ID: 010219339

Haasitha Pidaparthi
Student ID: 012669254

Do Hyeong Kim
Student ID: 014641497

Team Name
Team 13

Project Repo:
https://github.com/brightdo/CMPE256_Project

***Abstract*—The purpose of this project was to create a recommender system for amazon automotive products using reviews and other metadata provided by the dataset. The dataset is from UCSD's recommender systems repository. We used the 5-core Automotive dataset from the Amazon section for a "small" subset for experiments. The total file size is 713 MB but we reduced this in half or a third to around 200-250 MB in order for the execution times to be in scope for this class and feasible. The dataset has approximately 1,700,000 rows (reviews) and about 9 features (columns). Due to the large size of the dataset, we performed heavy preprocessing in order to get the data down to a manageable level. There were some columns that are erroneous and not included in our models. We removed the rows with reviews that are deemed false to be "not-verified" by Amazon based on the "Verified" column. During our modeling process, we will be taking into consideration the following columns: "Overall" which is the rating, "Verified" which is the verified status of the review/reviewer, "ReviewTime" which is the time at which the review was published. The algorithms to calculate the similarities were split along 3 libraries for each group member. The final models/libraries used were SKLearn, Surprise, and CNN.**

Keywords—Amazon; Recommender System; dataset; SKLearn; Surprise; CNN;

I. Introduction

Recommender systems have been used throughout the internet for ecommerce for a considerable amount of time. Many big companies have implemented these predictive algorithms to boost their website traffic and performance. Some of these companies include the likes of: Amazon, Walmart, Ebay, etc. There are many datasets available online that allow for independent researchers/students to create their own recommender systems. The dataset used in this project was from UC San Diego's publicly available online database for different Amazon product reviews and its respective metadata. Our group decided to use the automotive products dataset because we have seen that automotive products have the tendency to have the most reviews on Amazon. This was due to the fact that automotive products are one of the most purchased products on Amazon and have the most detailed reviews. Our group wanted to build a recommender system using this dataset in order to build some predictive models for this dataset. This would hopefully help us gain a better understanding of Amazon's item-item collaborative filtering. Also, the main objective is to show how different ways to procure this recommender system can be compared and contrasted. We decided to use very different algorithms and libraries in order to compare the RMSE score.

To our knowledge there are many amazon product recommender systems on the internet. There are many different implementations of various algorithms on datasets compiled in a similar manner. There have been different versions of the dataset released as well. Amazon also makes their own online competitions on datasets they release for students and independent researchers [1].

II. System Design & Implementation details

A. SK Learn Algorithms

The first library used is SK Learn which is the basis for many data science concepts. It has a large amount of machine learning algorithms that can be used for our recommender system. All of the modeling done in this implementation was done in Jupyter Notebook using Python 3. This was because the group was most comfortable using the technology. Jupyter Notebook allows for excellent testing and debugging of the code. Google Colab was considered for its online resources like the GPU which would decrease the runtime of our code. Finally, SJSU's HPC was considered to be used in order to perform the preprocessing of our dataset. This was very helpful because the initial dataset was too large to be manipulated by our local machines. Out of the Sci Kit Learn library the following algorithms were considered: Linear Regression, Logistic Regression, Support Vector Classification, Gradient Boost, K-Nearest Neighbors, Gaussian Naive Bayes, Multinomial Naive Bayes, and Random Forest Classification. This large number of algorithms was chosen from online forum recommendations and also prior knowledge from previous course work in Data Science. The final algorithms used for the recommender system were: Linear Regression, SVC, Gradient Boost, KNN, Gaussian Naive Bayes, and Random Forest. These were chosen from the considered algorithms above where the models that had similar architectures were removed. The recommender system was designed so that previously

preproceed dataset was ingested and then some important facts about the dataframe were exported.

	Unnamed: 0	overall	verified	reviewTime	reviewerID	asin	style	unixReviewTime
0	1	1	True	04 19, 2018	ABCA1A8E4DGV1	0209688726	{'Color:': ' Blue'}	1524096000
1	2	1	True	04 16, 2018	A1NX8HM89FRQ32	0209688726	{'Color:': ' Black'}	1523836800
2	3	3	True	04 13, 2018	A1X77G023NY0KY	0209688726	{'Color:': ' CA'}	1523577600
3	4	5	True	04 8, 2018	A3GK37JO2MGW6Q	0209688726	{'Color:': ' Black'}	1523145600
4	5	5	True	03 24, 2018	AIY18YON1TWJJ	0209688726	{'Color:': ' Black'}	1521849600
...	...	...	...	...	...	...	...	...
936191	1711514	5	True	06 19, 2018	A3H86E5N0F1Q5R	B01HJFDJ8S	NaN	1529366400
936192	1711515	4	True	08 23, 2017	AXH645B4SSAJY	B01HJFDJ8S	NaN	1503446400
936193	1711516	5	True	08 8, 2017	AMGJLCCNVFB8	B01HJFDJ8S	NaN	1502150400
936194	1711517	5	True	08 24, 2018	A1MJUNTX7CTR5U	B01HJI17Y8	{'Size:': ' H8-NEW'}	1535068800
936195	1711518	5	True	06 26, 2018	AF67UV2T2TFWB	B01HJI17Y8	{'Size:': ' H8-NEW'}	1529971200

936196 rows × 8 columns

Fig. 1 Preprocessed Dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 936196 entries, 0 to 936195
Data columns (total 8 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   id              936196 non-null  int64
 1   overall         936196 non-null  int64
 2   verified        936196 non-null  bool
 3   reviewTime      936196 non-null  object
 4   reviewerID      936196 non-null  object
 5   asin            936196 non-null  object
 6   style           339984 non-null  object
 7   unixReviewTime  936196 non-null  int64
dtypes: bool(1), int64(3), object(4)
memory usage: 50.9+ MB
```

Fig. 2 Dataset Info

Next, the non-numeric feature values were label encoded to numeric values. Finally, the null columns were dealt with.

```
id                    0
overall               0
verified              0
reviewTime            0
reviewerID            0
asin                  0
style            596212
unixReviewTime        0
dtype: int64
```

Fig 3. Null Values in Dataset

The dataset's features were visualized as a correlation map to show which features largely affected our target variable which was the rating of the product.

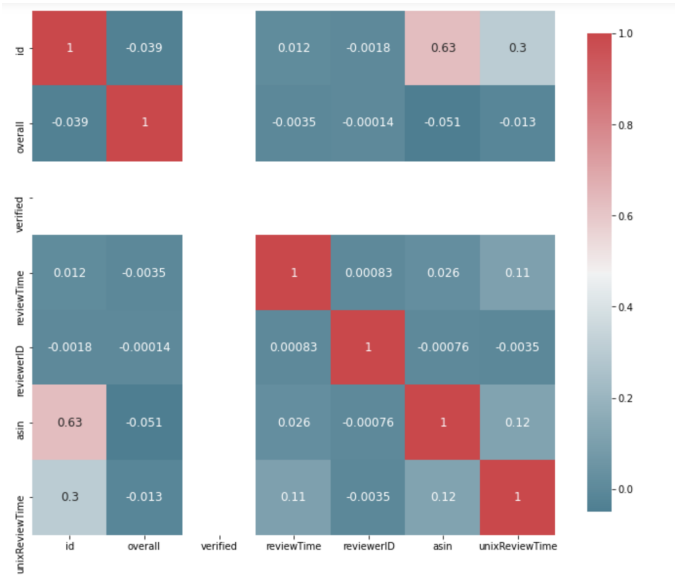

Fig. 4 Correlation Map

The dataset was split into a training and testing set to implement the hold out method of evaluation. Each algorithm was trained using the training data and validated on the testing target data. Finally, the accuracy and RMSE metrics were exported for each model to allow for easy comparison of the respective model's performance. This performance will be discussed in the next section of the report.

B. Surprise Algorithms

1. Algorithms Used

Although surprise provides with a huge variety of algorithms (SVD, NMF, Slope One, KNN, Colustering, and others), due to the limited time compared to the large amount of time it would take to create models and make predictions for each of these algorithms on our huge dataset, I had to limit the algorithms to run and compare the accuracy on. I knew SVD,SVD++, and NMF were variations of matrix factorization based algorithms, and KNNBasic, KNNWithMeans, and KNNWithZScore were variations of K-NN based algorithms. So I've decided to use only SVD and KNNBasic to represent the matrix factorization and KNN based. Co-Clustering was included for Cluster based algorithm and Slope One for item-based Collaborative filtering; KNN will also represent the user-based Collaborative filtering with user-based sim_option set to true. Along with all these algorithms provided by the surprise library, I have implemented my own variation of the KNN based algorithm that utilized the time of the ratings.

2. Technologies and Tools

For the implementation of the time-based KNN recommender, Surprise was used. Surprise is a python scikit for recommender systems that deal with explicit rating. It provides various ready to use prediction algorithms, tools to

evaluate algorithm's performance, and a simple way to implement new algorithms and use custom datasets.

3. Architecture Related Decisions

Additional preprocessing and data analysis was done before the data was used for the surprise algorithms. Since the surprise library only allows datasets with 3 columns (uid,iid,rating), all additional columns had to be dropped. For the time based knn custom algorithm, a copy of the dataset was created in order to keep the column for linuxReviewTime. After dropping the additional columns and renaming them to a more traditional context of iid,uid, and rating, additional analysis was performed on the dataset. First the number of missing values was checked(fig5), then the numbers of unique uid,iid,and ratings(fig6). Since the rating plays an important role in the algorithms, I looked into the counts of each rating(fig7) and also plotted a histogram(fig8). After the analysis, I performed a removal of duplicates. I looked for rows with a non unique uid and iid combination and dropped them from the data.

The premise of the time based algorithm was to give priority to the ratings given most recently. However since there could be ratings given more recently but with very bad user similarity score if time is blindly prioritized, we've restricted this prioritizing to K+5 most nearest neighbors (This value could not be K since even if selecting k would change which rating is more prioritized, all k rating values will still be used to create a predicted value, and not make any difference). Due to the high volume of memory required for KNN related algorithms, the dataset was truncated to a smaller size and fit into the KNNBasic algorithm from the surprise library to be used to find the nearest neighbors with the get_neighbors method. When we train the KNNBasic algorithm for the get_neighbors method, we use the build_full_trainset to fit the whole data into the algorithm. There is no danger of over fitting since the get_neighbors method will not return the user whom we want to predict the item rating for to be used to calculate the rating prediction and we need to fit the whole data in order to convert the inner ids of the nearest neighbors returned back into raw user id. After finding the top K+5 neighbors and sorting them by time, the top k ratings would be chosen and used to calculate the predicted rating by using a simple average.

```
reviewerID      0
asin            0
overall         0
dtype: int64
```

Fig. 5 Empty Values in Dataset

```
No. of Unique Users    : 161088
No. of Unique Business : 75894
No. of Unique Ratings  : 5
```

Fig. 6 Unique uid, iid, ratings Count

```
5    686420
4    124176
3     53338
1     44603
2     27659
Name: overall, dtype: int64
```

Fig. 7 Counts of Rating Values

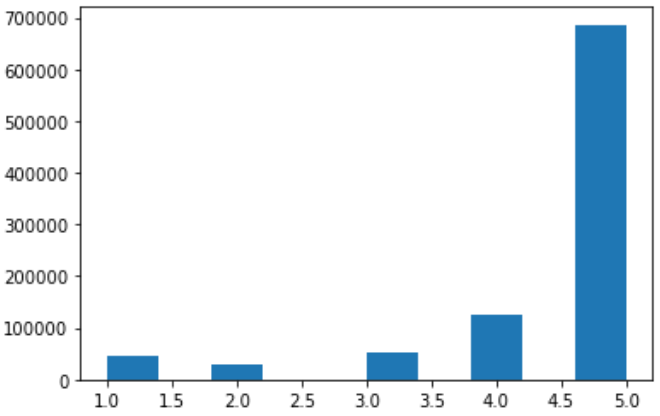

Fig. 8 Ratings Histogram

C. Convolutional Neural Networks

For this section, I used a variant of collaborative filtering recommender system by using a neural network approach. With the help of Embeddings, I created a representation of user and item from the dataset. An embedding is a mapping from discrete objects, such as user ids of items in this case, to a vector of continuous values. This can be used to find similarities between the discrete objects that wouldn't be apparent to the model if it didn't use embedding layers. The embedding vectors are low-dimensional and get updated whilst training the network.

To build the neural network, I used the TensorFlow Keras deep learning framework. Keras makes it easy to create the neural network embeddings as well as multiple input and output layers. The model was trained using Jupyter notebook with Python 3 version. However, I also tried to run the model using the TPU environment on Google Colab. Due to the amount of time it took to run the model, the session was timed out. In the end, due to the comfort of using the Jupyter notebook, I decided to build a model using this environment.

The dataset was heavily pre-processed and analyzed to consider how to use the different metadata of Amazon automobile items to best represent the ratings for the user-item recommender system. However, the meta-data cannot be directly used into the embeddings while creating the neural network model. Due to this reason, I used user and item columns to create the embedding.

The neural network uses two input embedding layers. The first embedding layer accepts the items and the second embedding layer accepts the users. Each of the two embeddings are trained separately and combined together using the similarity dot product for passing it into the model.

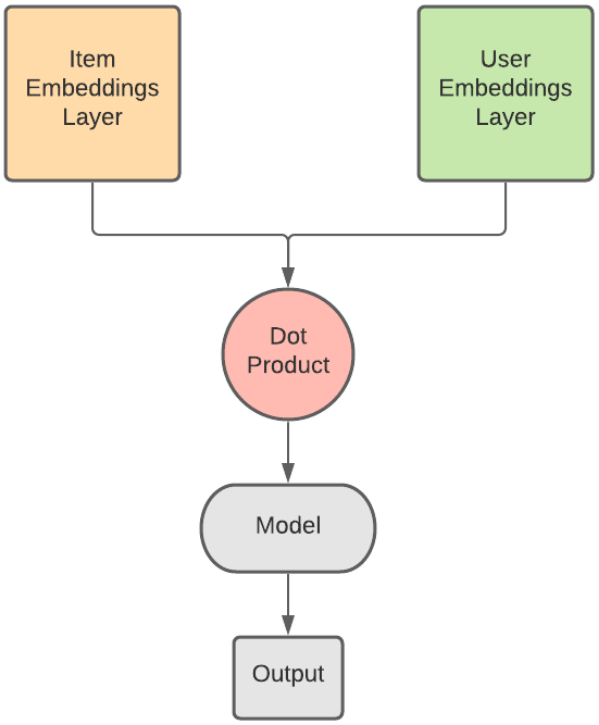

Fig 9. Neural Network Architecture

III. Experiments / Proof of concept evaluation

The dataset is from UC San Diego's vast archive of recommender systems repositories. We used the 5-core Automotive dataset from the Amazon section for a "small" subset for experiments. This was to keep the project in a manageable size so that the necessary steps could be done within the time constraints. The total file size is 713 MB but we reduced this in half or a third to around 200-250 MB in order for the execution times to be in scope for this class and feasible. The dataset has approximately 1,700,000 rows (reviews) and about 9 features (columns). The dataset was given in .json format so first we had to convert the file into .csv to work with the Jupyter Notebook libraries like numpy and pandas. We dropped columns like the index and other extraneous columns which were not needed for our recommender system.

For the preprocessing steps, we performed several techniques to bring down the size of the dataset. Firstly, we sorted the unixReviewTime and filtered out all the reviews made before the year 2016. We wanted to make sure that the recommendations are done on reviews that are more recent. Performing this step, brought down the dataset size to 971070 rows. Next, we wanted to make sure that the reviews were made with only the verified users. Removing the non-verified users brought down the size of the dataset to 936196 rows. In addition to that, we removed the reviewerName, reviewText, image of the product, style, and the numbers of votes made for the review. We did consider including the vote and style column to prioritize the reviews, but these columns were missing a lot of data so we decided to remove them. Finally, we brought down the dataset size from approximately 1.7 to 936196.

A. SK Learn Algorithms

Once the data was pre processed and manipulated as explained in Chapter II Part A, the data was fit using the models mentioned. The fitting was on the training data which was split from the whole dataset in a 70:30 split with respect to the testing data. This aligned with the recommended 70-80 % training data for the algorithms used. The models were then used on the testing feature data to generate predictions and finally those predictions were compared to the testing target data. Two methods of evaluations were used which were SK Learn's accuracy and RMSE scores. Both of these methods would allow for a holistic and unbiased evaluation of the recommendations given by the models and allow for one or a group of algorithms to be chosen as optimal for our dataset. Below are the accuracy and RMSE metrics:

```
Linear Regression Accuracy: 0.733055
SVC Accuracy: 0.733055
Gradient Boost Accuracy: 0.733033
KNN Accuracy: 0.620251
Gaussian Naive Bayes Accuracy: 0.733055
Random Forest Accuracy: 0.725756
```

Fig. 10 Accuracy Metrics

```
Linear Regression RMSE: 1.180882
SVC RMSE: 1.180882
Gradient Boost RMSE: 1.180863
KNN RMSE: 1.503637
Gaussian Naive Bayes RMSE: 1.180882
Random Forest RMSE: 1.185015
```

Fig. 11 RMSE Metrics

From both of these we can see that 3 of our algorithms had the exact same accuracy and RMSE scores. These were Linear Regression, SVC, and Gaussian Naive Bayes. Coincidentally, this was the highest accuracy and lowest RMSE scores observed in the testing. This can lead us to infer that perhaps this is the upper bound on the maximum accuracy that was achievable for our dataset given. This is further supported by the other algorithms metrics not being as good with KNN having the worst performance. KNN was not performing well due to the fact that there were not enough neighbors to our targets. The results tell us that the dataset did not provide enough results for our models to make gains on accuracy and that ~73% accuracy was based on dataset limitations. One thing to note is that KNN and Random Forest did take the longest time to fit and make predictions.

B. Surprise Algorithms

Once the data was analyzed and processed in Chapter II Part B.3, the data was fit into the reader object to parse the file; for KNN,custom time based-KNN, and SlopeOne, a subset of 50000 and 100000 were used due to the restrictions in computation memory. Then the parsed dataset split in a 80:20 split with respect to the testing data. This aligned with the recommended 70-80 % training data for the algorithms used. The algorithmic models mentioned above were fit with the training data and were used on the testing feature data to generate predictions. Finally those predictions were compared to the testing target data using the surprise accuracy module. The RMSE score was the evaluation method used to compare the results. These methods would allow for a holistic and unbiased evaluation of the recommendations given by the models and allow for one or a group of algorithms to be chosen as optimal for our dataset. Aside from the metrics of these algorithmic models with the default parameters, the metrics for GridSearch based optimal parameter models were also collected and compared. For KNN based algorithms manual prediction of various k values was run to compare its RMSE due to the restriction in memory and slope one takes no input argument. Below are the RMSE metrics.

Classifier	**RMSE**
SVD	0.9916
Co-Clustering	1.0849
Slope One	1.0323
KNN	0.9736

Fig. 12 RMSE Metrics(default parameter)

Classifier	**RMSE**
SVD	0.9823
Co-Clustering	1.0768
Slope One	1.0323
KNN	0.9659

Fig. 13 RMSE Metrics(gridSearch based parameter)

For the custom time-base Knn algorithm, I manually calculated the RMSE by summing up all the subtracted values of the predictions from the actual rating, then dividing it by the number of rows and square rooting it. I also ran this evaluation on multiple values of k to find the best k value for this algorithm. I compare this rmse directly with the KNNBasic to see how utilizing the time changed the prediction error rate The RMSE metric table is shown below

k	**RMSE(time based)**	**RMSE(surprise)**
3	1.0265	0.9942
4	1.0035	0.9916
5	0.9880	0.991
6	0978	0.9663
7	0.973	0.9662
8	0.9674	0.9660
9	0.9665	0.9659

Fig. 14 RMSE based on k value of time-based knn algorithm

From the comparison between the KNNBasic and our own time based knn-algorithm, we can see that the KNNBasic starts with a better RMSE Score but starts to have a very similar value as the value of k increases. I think that this is because as more and more k increases, both KNNBasic and custom knn starts to have not enough neighbors with the rating for the item we would like to predict and had to resolve the default option of returning the global_mean as the predicted rating. This is a problem due to the limited number of users and items in the truncated dataset. We hope to resolve this issue with the use of a HPC. But based on the RMSE of the two algorithms on smaller values of k that ran normally with enough neighbors to make the prediction normally, it looks like the KNNBasic algorithm performed better.

C. Convolutional Neural Networks

To work on this model, we used the userId (reviewID), itemID (asin), and ratings (overall) columns. We split the dataset into train and validation by 80% and 20%, respectively. The shape of the dataset turned out to be (749199, 3) (186997, 3).

We also created two variables to give the unique number of users and items. The model of the neural network structure consists of Input (users and items), Embedding Layers (embedding for users and items), and Dot (dot product of the combined embeddings). The embedding layer contains the weights the model learns during training. These embeddings our vectors with the size of n-factor. They start off as a random number and begin to fit by the model to capture the essential qualities of each user-item. Because we are using two input embedding layers, we will specify an array of training data as our X. They are not only used for extracting the information about the data but we can also use them to visualize them.

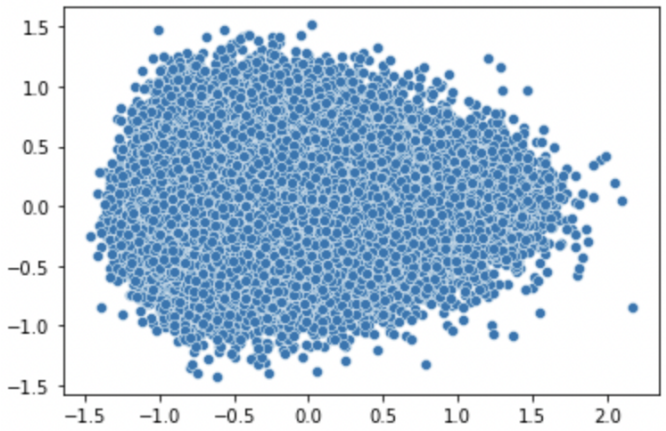

Fig. 16 Embeddings Visualization

The following diagram below shows the structure of how the model layers are implemented.

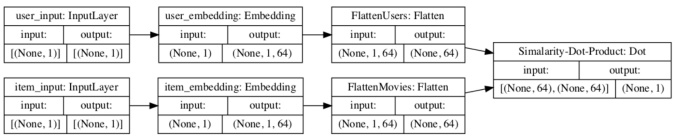

Fig. 17 Model Layers

To build a model, we used the Adam optimizer with loss function of a MSE and metrics RMSE. We ran the model with batch-size 128 and epochs 50. The following plot below shows the relationship between training and validation of RMSE according to epochs. According to this graph, although the RMSE for training was really low, the validation score dropped only to a certain threshold. This could indicate that the model is overfitting.

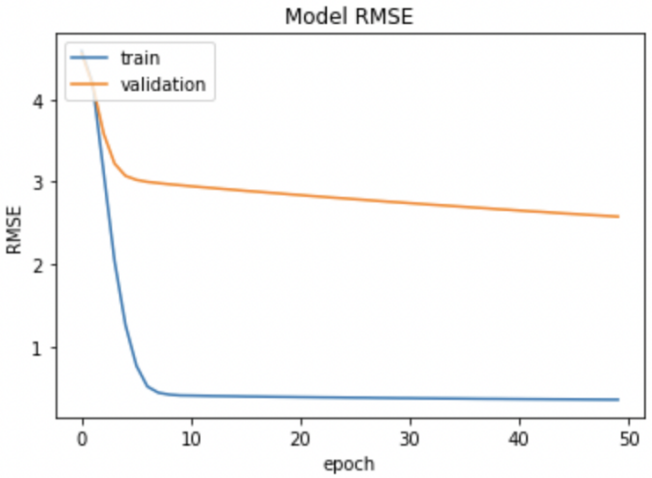

Fig. 18 Model RMSE

IV. Discussion & Conclusions

A. SK Learn Algorithms

From our initial data exploration stage we can see that we had a wide range of directions to move our project. The preprocessing we performed as a group worked well because we got rid of the extraneous columns in our dataset. This made it easier for our algorithms to perform and also freed up space in our system along with shorter run times. Another decision made was to choose a small subset of the original dataset. This was a good decision because it allowed the group to have ample time to work with the data and do some initial data exploration.

The Sci Kit Learn library was chosen because of the familiarity with it along with the large amount of online support, tutorials, and tips for recommender systems. The library was a good choice for our project because it did not require much research on how to use the library. Furthermore, the amount of algorithms available made it a suitable choice. The final list of algorithms used was an easy choice to make due to the amount of similar review based recommender systems projects available online.

For this aspect of the project there were not many difficulties encountered. The only notable difficulty was the inability to incorporate the style column into the final dataset. This was due to the formatting of the column. Due to the time constraints we were not able to include this in our data modelling. Furthermore, the column required that we used some NLP algorithms to associate the style with the Amazon product. This was not feasible in our project due to the large size of the dataset and our limited compute resources to do this type of modelling. Another notable difficulty was that run time of the models was rather lengthy in training, fitting, and making predictions. For the SK Learn algorithms, the total runtime for the entire jupyter notebook was about ~23 hours.

We can conclude that SK Learn did successfully create a recommender system for our dataset. The optimal algorithm for our application would have to be Linear Regression due to its short execution time in conjunction with the ~73% accuracy metric. The algorithm's suitable performance can be attributed to the fact that all of the data was encoded into numeric values and Linear Regression works well with this format of data.

B. Surprise Algorithms

The Surprise library gave us a great variety of algorithms with different approaches such as item-based(Slope One), user-based (KNN), Clustering based (Co-Clustering), and Matrix Factorization based (SVD). We were also able to RMSE of these algorithms with the default parameter and the optimal one with the grid search algorithm. Even the custom time-based KNN was created based on the find neighbors method provided by surprise. Based on the wide usage of it and the predictions it provided with very low RMSE, I think the surprise library was a good decision that led to positive results.

One difficulty that I faced while working with the surprise was the memory limit. Memory-based methods(KNN and Slope One) are very memory heavy algorithms that can not be run if the computer running it does not have sufficient memory space. Unfortunately, my computer's 8G of RAM was not enough to run the whole 0.9M data so I had to create a sunset copy of 50000 for the KNN and 100000 for the Slope One. From our comparison result from above, we know that the limited number of the dataset being used for KNN is forcing it to use the default method for predicting rating rather taking the similarity weighted average of the k closest neighbors. I would have wanted to compare the result of all these algorithms from surprise when used on the same larger 0.9M dataset.

Based on the comparative result of the custom time based and Basic KNN, we can conclude that the prioritizing of the time has decreased the precision of the algorithm and increased the RMSE. However this doesn't mean that the encompassing the rating time will only result in the decrease in precision and I think more analysis on how to give more weight to ratings given more recently without over shadowing the similarity score and using a better prediction method than just normal average such as weighted average using the surprise library's similarity.cosine() method would be great areas to look into.

C. Convolutional Neural Networks

Using the train model, we can make recommendations for the user on the items that they are more likely to purchase. To make the recommendations, we need to pass a list of items and the particular user we want to make the predictions for to the model. The model and then predict and get a list of items that the user would like.

One difficulty faced while training the model was the amount of time it took to run the model. We did try running the model on Google Colab, however, the session kept timing out because of the amount of time it took. Comparatively, HPC didn't seem to how much difference compared to the Jupiter notebook while running the model. Additionally, I also tried running the model with different layers such as Dense, to see if it made any difference on the model. In the end I decided to stick with the embedding modeling structure with optimizer Adam.

Looking at the observations of RMSE scores for both train and validation, we notice there is a steep decline for the training dataset and the graph levels out at about 0.35 RMSE score. However the validation score has a steep drop until 3.0 and there is a study decline up to 2.5. Although the RMSE score for validation was dropping at a steady rate, there is not much progress. Looking at the score difference between training and validation, we can say that this is a classic case of overfitting. We could further improve the model by hyperparameter tuning and possibly adding more layers to the network. Due to the time constraints and the amount of time it took to run each model, we could only hypertune the model so much. The original validation dataset RMSE score was 4.2470 before Hyperparameter tuning. This notebook can be found on the Github repository. After extensive hyperparameter tuning, the RMSE score for the validation dataset dropped to 2.579.

D. Overall Conclusions

From all of our RMSE metrics, we can conclude the Daniel's Surprise library SVD algorithm had the best results. The algorithm had the parameters of: epochs: 30, lr_all = 0.005, reg_all = 0.05, and factors = 20. This algorithm had a validation RMSE score of 0.9823 which was really good. His KNN also performed well and better than the SVD with a RMSE score of 0.9659 however this algorithm was only run on a subset of 50,000 rows. One thing to note was that Sci-Kit Learn's KNN had a worse performance than the Surprise version. This might be attributed to the algorithm structure and also the fact that it was run on the whole data set. Sci-Kit Learnn's KNN reruns the algorithm with a lower K value if it can not find enough neighbors compared to Surprise which just takes the global mean as its prediction. Further research on this topic might be required to gain a better understanding of this issue. Surprise's KNN was not able to run on the whole dataset due to the lack of memory in our local machine. For future experiments, we would like to run all of our tests exclusively on the HPC to accurately be able to compare our results.

V. Project Plan / Task Distribution

All assigned tasks are listed below and were completed successfully by the assigned team members. Group contributions are denoted appropriately with a Lead if applicable.

- Project Proposal: - All (Daniel Lead)
- Project Implementation:
  - Research Dataset - All (Simran Lead)
  - Preprocessing - All (Haasitha Lead)
  - SK Learn - Simran
  - Surprise - Daniel
  - Keras - Haasitha
- Project Report:
  - Abstract, Introduction, all SK Learn Sections, Overall Conclusion, References -Simran
  - All Surprise Sections - Daniel
  - All Keras Sections - Haasitha
- Project Presentation:
  - All Slides - All

References

[1] L. Hardesty, “The history of Amazon's recommendation algorithm,” Amazon Science, 22-Nov-2019. [Online]. Available: https://www.amazon.science/the-history-of-amazons-recommendation-algorithm. [Accessed: 01-May-2021]