Investigating Recommender System for Online Shopping

Final Report

Information Technology Capstone Project

COMP5703

Group Members

Deniz Doruk Nuhoglu	(490283336)
Jingwen Wang	(480480329)
Madanagopal Pasupathy	(480219291)
Ming Sheng Choo	(490072615)
Mohamad Reza Pancawan	(480172262)

Abstract

Recommender system, as one of the subclasses of the information filtering system, is widely used in multiple industries. It excels at exploring and forecasting user preferences. In modern society, the rapid growth of E-Commerce business meaning that the recommender system will be an essential component for all online business platforms. For E-Commerce, it is a useful technique to help users make better decisions and enhance user loyalty. In this project, we proposed a model based on a deep sparse autoencoder that using user and item embeddings. We also provide a comprehensive analysis and comparison between the proposed model and other popular collaborative-based algorithms such as item-based KNN, SVD, and Co-clustering. The comparison will also include the exploration of the different data sampling methods in the real-life skewed Amazon dataset. Through different test scenarios, the model that we propose, is proven can produce as low as 0.7 of mean absolute error (MAE), which also makes this model has outperformed the other models in regards to MAE.

Keywords: Recommender System, deep learning, deep sparse autoencoder, collaborative filtering, mean absolute error.

Table of Contents

1. INTRODUCTION

In this modern age, recommender system is widely implemented in many areas that we interact in day-to-day basis, such as Netflix, Youtube, Spotify, Amazon and Facebook. Various concepts and algorithms have been proposed to explore and improve the effectiveness of recommender system. Three popular methods of these algorithms are collaborative filtering (CF) that make use of ratings provided by multiple similar users to make recommendations, content-based filtering (CB) which use descriptive attributes of items, and hybrid method (HB) which can be a combination of at least two different algorithms. (**Figure. 1**) below show the classification of RS based on the method used to create recommendation.

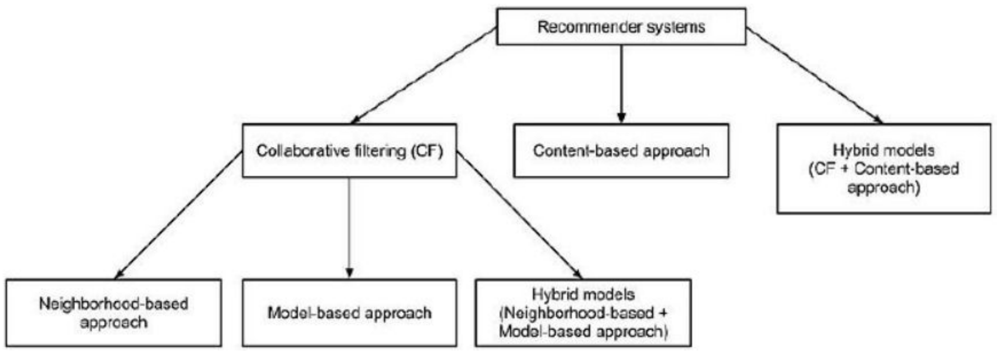

Recommender systems
Collaborative filtering (CF)
Content-based approach
Hybrid models (CF + Content-based approach)
Neighborhood-based approach
Model-based approach
Hybrid models (Neighborhood-based + Model-based approach)

Figure 1. RS classification (Eleks, 2014)

We believe that a good recommender system will not only improves the user's online shopping experience but also generate additional revenue for the business, hence we will take a look at what has been done in recommender system space and compare our proposed method with some with some of the classical baseline model.

We briefly present the work that has been done in and discusses each methods strengths and weaknesses in section two, research question-related information will be presented in part three, part four presents the approaches that we will be using as the base model and briefly discuss on dataset analysis and usage as well as the evaluation that will be used for performance measurement.

All the resources, which include the specification of hardware & software, additional materials & readings, as well as the roles and responsibilities of each member, will explain in section five. In the following section, we also provide a comprehensive project schedule and Gantt-chart along with the related milestones on each step. In section 8 and 9, we provide a detailed and in-depth analysis regarding the result of the experiment of various methodologies which has been proposed before. As for the last part, we also provide constructive future work to fellow students and researchers who have an interest in continuing this work.

2. Related Literature

2.1 Literature Review

In recent years, many researchers have proposed different methods in tackling recommender system, in CF research area some of the popular methods includes item-based and user-based collaborative filtering approach (Sarwar, B., Karypis, G., Konstan, J., & Riedl, J., 2001).

The basic idea of the CF algorithm is to generate item prediction based on known preferences of other users. In most cases, CF has proven to make better recommendations compared to the CB method. Moreover, the CF method does not require the user to state their interest explicitly. (George, T., & Merugu, S. 2005). Two types of CF's method that are commonly used in recommender systems are memory-based and model-based (Aggarwal, C., 2016). The memory-based method emphasizes the whole user and item data to provide predictions. This method uses a statistical technique in finding a set of users that share the same interest with the target user (it could be from rate or purchase similar item) known as neighbor. Therefore this method is also widely called a neighbor-based model.

Two popular techniques to determine the neighbor similarity are user-based technique item-based technique where the first finds similarity in user's preference and the latter finds similarity between items, both techniques normally make use of statistical methods such as pearson correlation and cosine similarity. Item-based CF tends to provide better prediction compared to user-based CF as shown in a previous study (Sarwar, B., et al., 2001) and the disadvantage of memory-based method has also been discussed in another study (Su, X., & Khoshgoftaar, T., 2009) which indicates that the methods will not perform well if working with sparse data. Also, one of the drawbacks of the neighbor-based model is the complexity of the offline phase that leads to resource-intensive especially when the number of data is huge (Aggarwal, C., 2016).

The other way to build RS with the neighbor-based model is by clustering to replace the expensive computation of traditional neighbor-based model. The variance of the clustering method that allows Co-clustering to allow each user or item belongs to multiple class or categories. The study from (George, T., & Merugu, S. 2005) shows that the co-clustering can become one of the alternatives to build RS since it can run faster especially compared to a model-based algorithm such as singular value decomposition (SVD) and gives the considerably same result of performance.

On the other hand, the model-based method adopts several machine learning and data mining methods to generate predictions. Some algorithms that broadly use to develop model-based CF are matrix factorization, co-clustering, and decision tree. The experiment from (Joonseok, L., Mingxuan, S., & Guy, L., 2012) concludes that the performance of model-based CF is generally better than the memory-based CF, especially when it uses some algorithms such as singular value decomposition (SVD) and probabilistic matrix factorization (R. Salakhutdinov and A. Mnih., 2008). Another work from (Aditya, P., Budi, I., & Munajat, Q., 2016) also shows that the model-based CF with improved naïve-Bayes algorithm outperformed the memory-based CF that applied the nearest neighbor technique. However, both of the work also share the same conclusion that model-based CF computation is generally slower.

Recently, one algorithm that has received much traction is none other than the latent factor model, the model gained a lot of popularity after Netflix's prize competition, SVD(Koren, Y., 2008), latent Dirichlet allocation (LDA) (Hofmann, T., 2004), pLSA (David, M.B., Andrew, N., & Michael, I.J., 2003), alternating least squares (ALS) (Jain, P., Netrapalli, P., & Sanghavi, S., 2013) are the matrix factorization method that has been proposed and explored in recommender system. SVD++ was later proposed and designed to estimate the implicit information, the result was that it surpasses both SVD and Asymmetric SVD (Koren, Y., 2008).

The other approach of RS is known as the content-based method, that unlike collaborative filtering which makes a prediction or recommendation based on neighbor user's similarity, the content-based method make use of building user profile based on user's past interaction which could include what user has read, rated as well as the attribute that is associated with the product. Many different implementations of this text-mining technique called TF-IDF in their recommender system, for example in Newsdude (Pazzani, M., & Billsus, D., 2000) where TF-IDF is implemented for their short term model within the hybrid system. Other implementation of content-based filtering can also be seen, Libra (Mooney, R.J., Roy, L, 2000), Citeseer (Bollacker, K., Lawrence, S., & Giles, C., 1998) and much more.

Since building a user profile can be seen as a learning problem, many different machine learning techniques have also been applied in the content-based filtering domain, one of which is the implementation of decision tree (Amir, G., Amnon, M., Karl-Heinz, L., Lior, R., Alon, S., Arnon, S., 2010). Other implementation approach and review can also be found on Content-Based Recommender system (Pazzani, M., & Billsus, D., 2007).

There is a lot of hype around deep learning lately, many different researchers have also came up with different deep learning models in exploring recommender system area. Neural Collaborative filtering (Xiangnan He, Lizi Liao, Hanwang Zhang et al, 2017) has proposed a feed-forward neural network that has since become a classical neural network method, other deep learning approach has also been proposed such as restricted Boltzman machine (RBM) (Salakhutdinov, R., Mnih, A., & Hinton, G., 2007), deep autoencoder has also been explored in the past decade, because of how well autoencoder works in capturing important features. Different variants of autoencoders has also been proposed, started of with AutoRec (Sedhain, S., Menon, A., Sanner, S., & Xie, L., 2015), stacked denoising auto encoder (Suzuki, Y., & Ozaki, T., 2017), Collaborative Variational autoencoder (Li, X., & She, J., 2017) where it seeks for probabilistic latent variable with the use of maximum a posteriori estimates(MAP).

With all these previous related work presented, it prompted our interest in deep autoencoders where we wanted to further explore the effectiveness in autoencoders within an e-commerce environment setup.

3. RESEARCH/PROJECT PROBLEMS

Autoencoders was chosen as our research method due to the strong capabilities of finding latent representation even in a sparse dataset similar to matrix factorization, this fits to an e-commerce environment where long-tail phenomenon is most likely to happen.

3.1 Research/Project Aims & Objectives

In our project motivations, it is a good chance to develop insights and knowledge in recommender systems. For example, know the multiple recommendation algorithms' advantages and shortcomings.

To better understand autoencoders in recommender systems and suggest improvements on existing architecture. We use Autoencoder because it is popularly used in compression and we wanted to analyze deep learning areas where it has similar functionality.

To achieve our aims:

1. We investigate how well it will perform in skewed-dataset.
2. Compare against based line methods to see how well it performs.
3. And investigate if the same could work in the collective domain.

We are interested to see how well the proposed method will perform in a close to a real-life dataset such as the Amazon dataset.

3.2 Research/Project Questions

We started our project with the following research question:

- How well our algorithm will compare with all the baseline algorithm such as item-based KNN, Co-clustering, matrix factorization and autoencoder.
- Does a skewed dataset like Amazon dataset impact the overall performance of the algorithm?

- We would like to further investigate if any algorithm will work well in a collective domain dataset.

3.3 Research/Project Scope

- For this project, due to resource constraints we are able to run only a fraction of the full amazon dataset which we will further discussed in a later section.
- We will also explore some different parameter settings and discuss on the improvement it brings towards overall performance.
- For testing and evaluation, as discussed in the first proposal report we will first evaluate the model based on various evaluation metrics result.

The scope of the study included assumptions about the testing results of data set and tested the robustness and stability of the different algorithms. In this way, different methods are rated and tested to obtain conversion rates of different recommended algorithms, and then the best-recommended algorithm for Amazon data sets is obtained. Cross-testing is performed using different evaluation methods to confirm the consistency of the results.

4. METHODOLOGIES

4.1 Methods

To analyze our proposed autoencoder, we will compare it against several algorithms other than the baseline autoencoders. Two different types (single sub-category, collective domain) of dataset will also be used to test and evaluate in all of the mentioned algorithms, which we will further discuss in section 2.2. As for the methodologies, we divided the study into four parts as follows:

1. Compare the performance of each algorithm with two famous evaluation metrics. As for the algorithms that we used are item-based KNN, Co-Clustering, Matrix Factorization (SVD), baseline autoencoder, and deep sparse autoencoder with embedding.

2. Compare the algorithm performance based on various types of data sampling (twenty cores, fifteen cores, and ten cores) on a single and collective domain dataset. As for the single domain that we used is the Electronics set, while we combine the CDs and Movies collections to form the collective set.
3. Analyze the top-N recommendation items of autoencoders. With top-N analysis we will measure the performance subjectively where we examine the correlation of user purchase history with the recommendation items produced from autoencoders.
4. Conduct a further discussion and in-depth analysis of different autoencoder setups.

Below is the explanation of the evaluation metrics, algorithms, parameter set up and autoencoders architecture that we use in the project.

- Evaluation Metrics

To measure the accuracy of predictions of tested algorithms, we decided to use the widely used evaluation matrices which are the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). Additionally RMSE was also used as the measurement metric for the Netflix Prize contest where it will penalize the significant error by using squared term in the summation (Aggarwal, C., 2016). In other words, we tend to examine the smaller score of RMSE that will produce by the algorithms, where the lower number is, the better one.

$$MAE = \frac{\sum_{(u,j)\in E} |e_{uj}|}{|E|} \qquad RMSE = \sqrt{\frac{\sum_{(u,j)\in E} e_{uj}^2}{|E|}}$$

- **Item-based KNN**

The item-based KNN is a popular recommendation algorithm that combines the traditional item-item collaborative filtering and KNN. The item-based KNN that we use in this project is explained in four steps: (1) generate item and user rating matrix, as shown below (**Table 1**).

Table 1. Example of user item matrix

	user_a	user_b	user_u
item_1	5	3	**3**
item_2	2	4	**5**
item_i	1	5	**?**

(2) Calculate the cosine similarity to compute the similarity of item_i to other items for user *u* (3) pick the top K most similar item. (4) predict the rating of item *i* for user *u* (*Rui*) with prediction function:

$$Rui = \frac{\sum_{j \in N(i;u)} sim(i,j).\ ruj}{\sum_{j \in N(i;u)} sim(i,j)}$$

Where *sim(i,j)* is the similarity of item *i* and *j*, while the *ruj* is the rating of user *u* on item *j* and *N(i;u)* is the set of items rated by user *u* that similar to *i.* As for the implementation using surprise we are following some the default parameter as follows:

Parameters:

1. ***k** (int) – The max number of neighbours Default is **40**.*
2. ***min_k** (int) – the minimum number of neighbours. Default is **1**.*
3. ***sim_options** (dict) – similarity measurement. We choose **Cosine similarity**.*

- **Co-Clustering**

According to the paper from (George, T., & Merugu, S. ,2005) the algorithm consists of three parts, which are the (1) static training which involves co-clustering the rating matrix as well as to compute several summary for prediction, (2) prediction which consists of the estimation of unknown rating, and

(3) the incremental training that includes updating the summary statistics based on the incoming rating. In summary, the user and items will be assigned to some clusters C_u, C_i, and some co-cluster *Cui*, while the prediction R_{ui} is a set as:

$$R_{ui} = \overline{C_{ui}} + (\mu_u - \overline{C_u}) + (\mu_i - \overline{C_i})$$

where $\overline{C_{ui}}$ is the average rating of co-cluster C_{ui}, and $\overline{C_u}$ is the average rating of u's cluster, and is the average rating of i's cluster. Furthermore, if the user is unknown, the prediction R_{ui} will have an equation to $R_{ui} = \mu_i$ and if the item is unknown, the prediction is $R_{ui} = \mu_u$, while if both the user and the item are unknown, the prediction will be $R_{ui} = \mu$. As for the implementation using surprise we are following some of default parameters as follows:

Parameters:
1. ***n_cltr_u*** *(int) – Number of user clusters. Default is* ***3****.*
2. ***n_cltr_i*** *(int) – Number of item clusters. Default is* ***3****.*
3. ***n_epochs*** *(int) – Number of iterations of the optimization loop. Default is* ***20****.*

- **Matrix Factorization - SVD**

We have adopted the matrix factorization based algorithm (SVD), which is also equivalent to the probabilistic matrix factorization that has been proposed by (R. Salakhutdinov and A. Mnih., 2008). To predict the ratings of user *u* to item *i* (R_{ui}) we follow the equation that represents the relationship between average rating, user and item bias as well as the user and item interaction :

$$R_{ui} = \mu + b_u + b_i + q_i{}^T P_u$$

Where $\boldsymbol{\mu}$ is the average rating, b_u & b_i is the user and item biased from the global average, while q_i is the vector that associated with each item *i* and each user *u* associated with vector p_u (Y. Koren, R. Bell, and C. Volinsky, 2009). And to estimate the unknown ratings we also adopt the following regularized squared error:

$$\sum_{r_{ui} \in R_{train}} (R_{ui}(true) - R_{ui})^2 + \lambda(b_i^{\ 2} + b_u^{\ 2} + \|q_i\|^2 + \| p_u \|^2)$$

As for the implementation using surprise we are following some the default parameter as follows:

Parameters:

1. ***n_epochs*** *(int) – Number of iterations of the SGD. Default is **20**.*
2. ***biased*** *(bool) – Default is **True**.*
3. ***lr_all*** *(int) – Learning rate for all parameters. Default is **0.005**.*
4. ***reg_all*** *(int) - Regularization for all parameters. Default is **0.02***

- **Autoencoder**

Autoencoder is widely used in denoising and decompression of image data, recently different variation of autoencoder has also been used as a generative model to generate sentences or even artwork (Jun-Yan Zhu, Taesung Park, Isola, P., & Efros, A. 2017).

Similar to matrix factorization, autoencoder will try to learn the function $h_{W,b}(x) \approx x$, in simple form, the model will try to learn an approximation to the input x, where it has two functions during training of the neural network, namely the encode function $encode(x) : R^n \rightarrow R^d$ and the decode function $decode(x) : R^d \rightarrow R^n$ in order to find the representation of reconstruction.

- **Autoencoder with iterative refeeding**

A new deep learning model of autoencoders (Kuchaiev, O., and Ginsburg, B., 2017) with iterative refeeding feature along with custom loss functions.

Where the algorithm and refeeding steps can be defined as follows:

1. Given Sparse x, compute loss with Masked Mean Squared Error, where the equation can be defined as:

$$MMSE = \frac{m_i * (r_i - y_i)^2}{\sum_{i=0}^{i=n} m_i}$$

   Where r_i is the actual rating y_i is the predicted ratings, and m_i is the masked ratings such that $m_i = 1$ if r_i != 0 else $m_i = 0$, and RMSE is the square root of MMSE.

2. Update weights with back propagation
3. Reuse $f(x)$ as new training example, and compute $f(f(x))$ with calculation of loss MMSE (refeeding feature).
4. update weights with back propagation

Note that step 3 and step 4 can be performed multiple times during each iteration.

- **Deep Sparse autoencoder**

We have argued that the method that was proposed in the paper was not well optimized, where with large amount of data it will always converge to the expected error according to large number law. Secondly, the user item matrix generation will be an issue as the items increase where we will show the results for each different algorithm in the result section. Due to the sparsity rate the autoencoder will not be able to perform well even with the use of custom loss and custom accuracy measures as discussed in the earlier section. Lastly, we believe that with the data size of our current experiment, iterative refeeding feature will not bring any significant impact to the model, as shown in FlexEncoder(Tran, D.H., Hussain, Z., Zhang, W.E., Khoa, N. et al. 2019) where they have tried similar feature but with the conclusion that the refeeding does not bring a significant impact to the model.

So our proposed model is as follows, instead of generating user and item matrix and feed into the deep autoencoder, we will take user and item as input but convert them into two embeddings with dropouts, then concatenate them into new latent features before feeding into the deep autoencoders. In order to further enforce the autoencoder to learn only the representation of the input data without redundancy in input, we will then apply regularization constraint to the autoencoder with L2 regularization, where the L2 term can be defined as below :

$$L(x, \hat{x}) + \lambda \sum_i a_i^2$$

The final proposed method can be seen from the below figure (**Figure 2**):

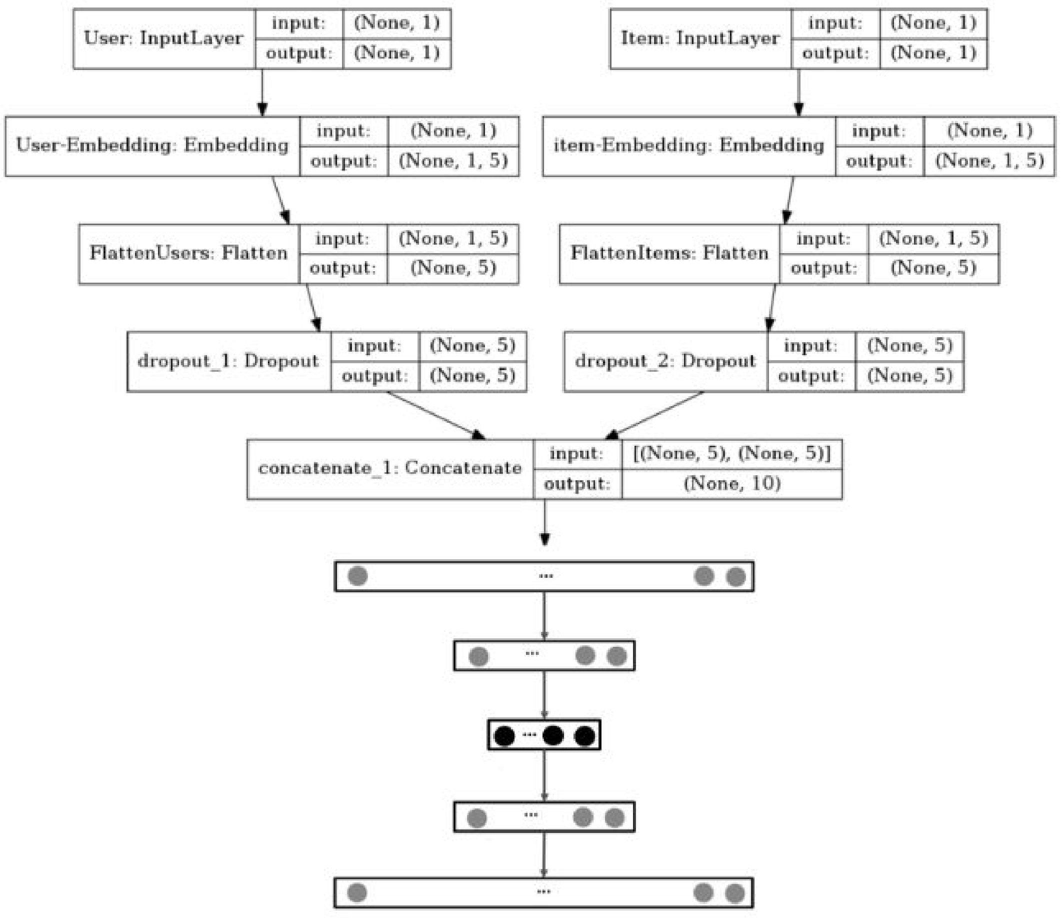


Figure 2. Deep Sparse Autoencoder structure

In short, we make use sparse autoencoders instead of the normal autoencoder with high dropout rate.

- **Hyperparameter tuning with grid-search**

  For further investigation for all the traditional collaborative filtering (item-based KNN, co-clustering, and SVD) we conducted a hyperparameter tuning with grid-search and three-fold cross-validations. Below are the parameters that we tried to optimize for each algorithm:

  1. Item-Based KNN: (1) similarity options: [cosine, pearson], (2) number of K: [40, 31, 11]. Please note, due to the limitation of resources we are unable to complete the grid-search process for item-based KNN.
  2. Co-clustering: (1) number of user clusters: [3,5], number of epoch: [10,20], number of item clusters: [3,5]. As for the search result the new parameter used respectively are [3, 10, 5].
  3. SVD: (1) learning rate: [0.005, 0.01], regularization: [0.02, 0.1], number of epoch: [10,20]. As for the search result the new parameter used respectively are [0.005, 0.1, 20].

- **Top-N analysis**

  To perform the top-N analysis, we emphasize several steps as follows:

  1. We choose the N=20, which means that we only take the top-20 purchased items of specific users from Electronics set based on their review score.
  2. Run the deep sparse autoencoders algorithm three times with different K sampling number (20, 15, 10) to generate recommendations items. We are limiting the number of recommended items to 20 as well, which sorted by the prediction score.
  3. Compare the similarity of users purchased history and preferences with the recommended items based on the general types of items.
  4. Analyze the relationship between top-N analysis and the offline evaluation metrics (MAE, RMSE).

4.2 Data Collection

Data is selected based on the following criteria, first we will evaluate based on the dataset size, so that we can stimulate a close to real-world e commerce environment for our experiment. Second, we will check if the database has a basic e-commerce platform's attributes such as product data, user's implicit and explicit ratings.

MovieLens (Grouplens, 2019), Brazillian E-Commerce Public Dataset by Olist (Kaggle, 2018), Netflix Prize dataset (Netflix, 2017), Retail rocket recommender system dataset (Retail rocket, 2017), Book-Crossing dataset (Cai-Nicolas, Z., 2004), and Amazon Recommender System dataset (McAuley, J., 2016) is all the dataset that we gathered based on the above-mentioned criteria. The data of the datasets listed above is collected by either research group or company authorized dataset for research/contest usage, where some of them can be found in Kaggle while other is from their respective research group site. We have chosen Amazon dataset for our experiment and research as we think the dataset has both the complexity and all fits all the basic requirement to start on building a recommender system where it consist of ratings data, meta data of products as well as image features if we wish to expand on the current research in the future.

4.3 Data Analysis

We conducted several data analysis in order to form a workable data. All the data analysis we have done in this project are: initial assessment, categorical analysis, k-core analysis.

- **Initial assessment**

  As shown in **Figure 3**, we analyze the complete review, and metadata with the respective size are 18GB (82.83 million reviews) and 3.1GB (9.4 million product). From the analysis we know that the average review score across the 82 million reviews in the whole amazon dataset is 4.16, while the median score is 5 out of 5. Around 60% of the ratings are 5 out of 5, and 20% of the reviews are a 4 out of 5, signaling that the distribution is highly skewed to the higher end. At the other end, 8% of customers have a review score of 1 out of 5. Meaning that most of the customers will rate and review the product if they are happy with the product or service provided by Amazon.

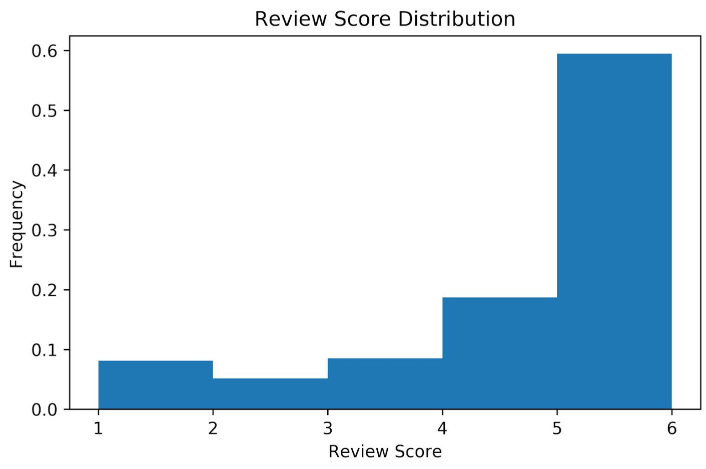


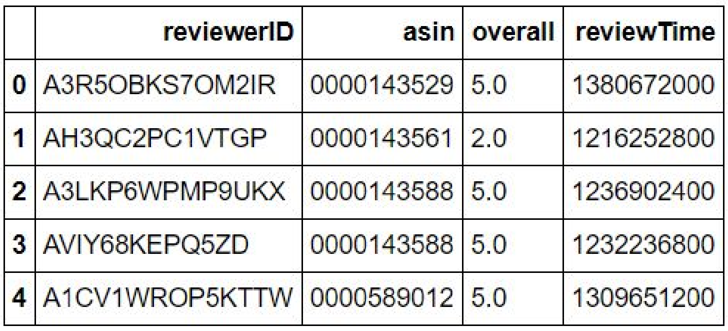
Figure 3. Review Score Distribution

The processing of the dataset itself, is highly cumbersome at this stage. The unzipped version of the reviews dataset in json format is 50GBs. In order to extract the summary statistics, we had to use iterations that used a chunksize of 1 million. As discussed in the earlier section, because of the actual dataset is extremely skewed, we have concluded that we will work with a subset of the full dataset. By doing this, we will be able to train the models more efficiently with providing meaningful data to the model. We have further breakdown the dataset into two analysis as shown below:

- **Categorical analysis**

In this part, we performed an analysis of the specific two categories single domain and the collective domain. As for the single domain, we tend to analyze the Electronics set that contains 7.8 million reviews, while for the collective domain we combined the CD and Movies set together that providing 8.4 million reviews.

The two datasets are chosen because we think that it can represent how the aggregate data is spread. The rating set consists of four-column as follows: (1) reviewerID: to represent the amazon user ID, (2) asin: to represent the amazon product code, (3) overall: to represent the ratings of item with shows by number 1-5, and (4) ReviewTime: to show the time of comment. (**Figure 4**) will show how the ratings data of Amazon dataset looks like.



	reviewerID	asin	overall	reviewTime
0	A3R5OBKS7OM2IR	0000143529	5.0	1380672000
1	AH3QC2PC1VTGP	0000143561	2.0	1216252800
2	A3LKP6WPMP9UKX	0000143588	5.0	1236902400
3	AVIY68KEPQ5ZD	0000143588	5.0	1232236800
4	A1CV1WROP5KTTW	0000589012	5.0	1309651200

Figure 4.Example of Amazon's data set

To further elaborate on how collective data is created, CD and Movies are selected because while they are two different domain but they also resembles some similarities where movie soundtrack can be released on CD, we decided to directly inner join both reviews from CD and Movies based on reviewer ID to obtain all the cross domain buyer, hence the set only contains all the user that has bought products in the two sub-category.

The picture below (**Figure 5**) shows the rating distribution of 4.201.696 users and 476.002 items in the Electronics set. Based on the data, we know that more than 54% of data are having a reasonable rate as five, while 20% of them are given rate as 4. Another percentage is for the item that just rated as one with around 12% and rated as 3 for about 8%, while the smaller portion of the item (5%) is rated as two.

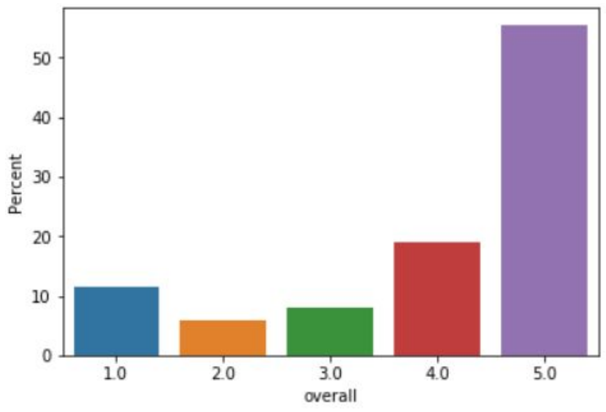
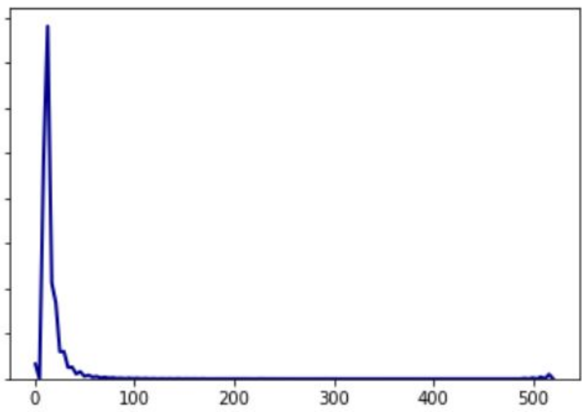


Figure 5. Review Score Distribution

Other analysis to see how much active the user in rating items in this set. As we can see in the graph and table below, the user of this set does not like to rate an item. Where almost all among 4 Million users only have one rate, with just one user with 520 rates as an exception.

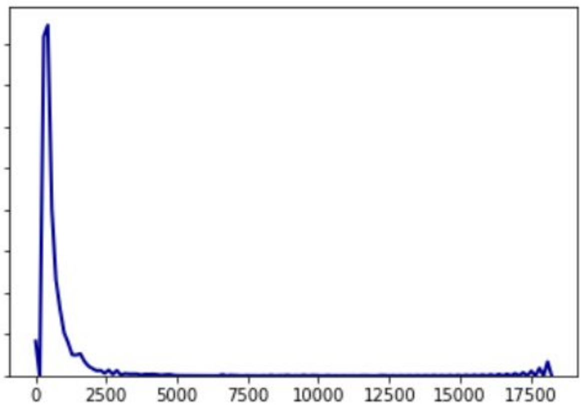
Table 2. Users ratings analysis (Single Domain-Electronics)



	Rate/User
count	4201696
mean	2
std	3
min	1
25%	1
50%	1
75%	2
max	520

A similar analysis is also conducted (**Table 3**) to see the distribution of item popularity, where we derived from how much rating that an item contains. Based on our research, there is one item that stands out with 18.244 scores. While the rest of the items only has one rating as minimum and seven as the top ratings. It means that the Electronic set that we used are only filled by niche product. As for the other possibility, it also because the items are not adequately rated despite their high sales.

Table 3. Items ratings analysis (Collective domain - CD & Movies)



	Rate/Item
count	476002
mean	16
std	113
min	1
25%	1
50%	2
75%	7
max	18244

In analysis of CD & Movie set (collective set), we found that more than 60% of the items are rated as 5, and around 18% of the items are rated as four which follow by items that have three rates with 9%. As for the items that have 1 and 2 rates are distributed to 8% and 5% respectively. Moreover, this set are containing 3.266.499 of unique users and 674.246 items.

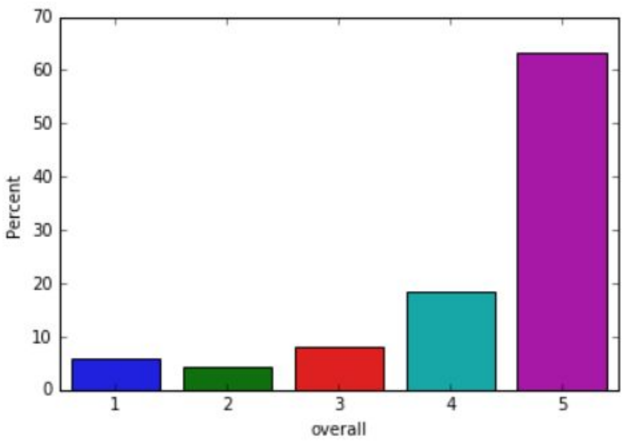
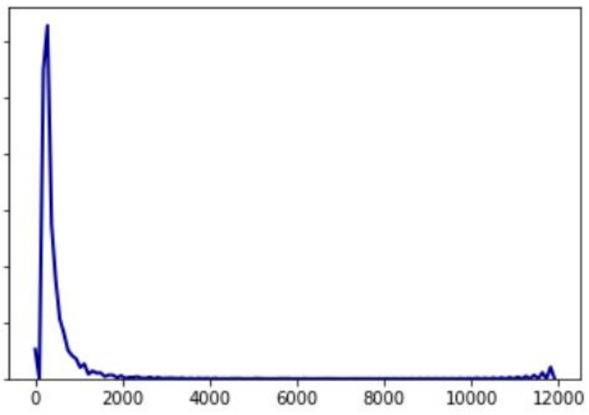
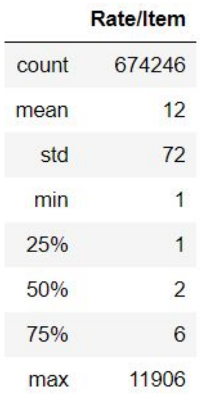


Figure 6. Distribution of ratings on Collective set

As we can see from the figure above (**Figure 6**), the rating data are very sparse where half of the total items only contain two rates. 25% of it only has 1 ratings and 6 ratings. However the top-rated item has been rated 11.906 times, which is drastically different compared to other popular set such as movilens or even Netflix dataset.

Table 4. Items ratings analysis (Collective domain - CD & Movies)

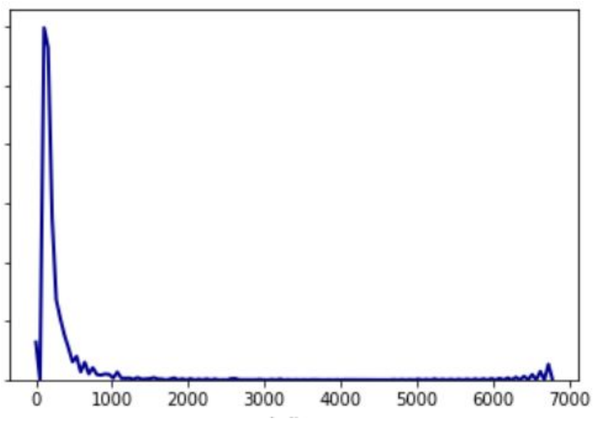


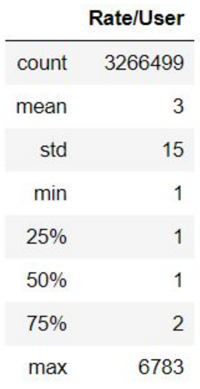


	Rate/Item
count	674246
mean	12
std	72
min	1
25%	1
50%	2
75%	6
max	11906

Further analysis shows also shows that almost all of the users that belong on CD & Movie Set are just made one or two ratings, despite the outliers that rated 6.783.

Table 5. User Ratings analysis (Collective domain - CD & Movies)





	Rate/User
count	3266499
mean	3
std	15
min	1
25%	1
50%	1
75%	2
max	6783

Finally, we believe that by conducting this categorical analysis, we can focus our research on the specific categories and get a much smaller data set to work with. But we also realize the problem of sparsity that these two sets have. Therefore, we were continuing our analysis of the k-core analysis as one of the possible methods to produce more meaningful data.

- **K-core analysis**

In this section, we conduct further analysis of the distribution of categorical data. The main objective of this analysis is to filter out less meaningful data to work with. We adopted k analysis, where we will only select items that has only been rated K times and users that has done reviews at least K times. The three-types of k core analysis that we have are k equals to 20, 15, and 10. A higher K setting will always give us a smaller set while with lower K setting will also give us a more prominent and sparse set.

As in the first step of the analysis, we were taking the result of the categorical analysis. Based on the previous result we know that the original Electronics set contains a sparse 7.82 million reviews record that spread to 4.2 million users and

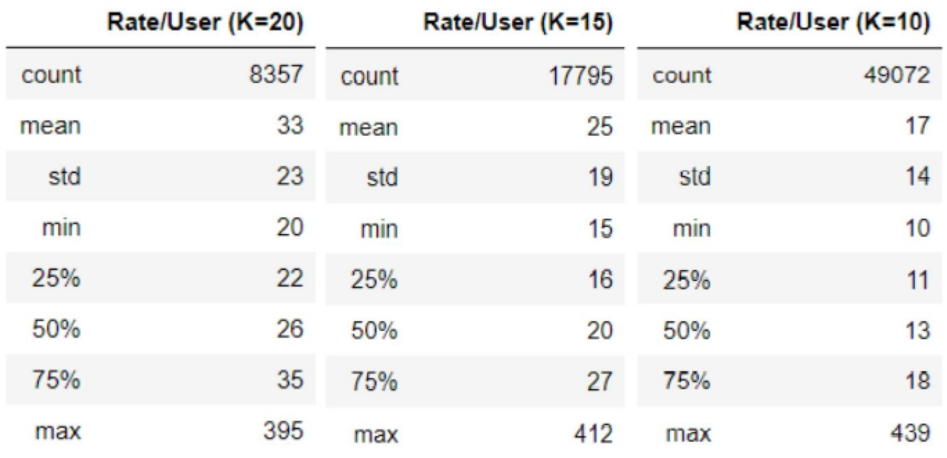
476 K items. Similarly in CD & Movies sets are having 4.61 million reviews and 2.08 million unique users with 200 K items.

Next, we want to analyze the impact of the application of each cores number (20,15,10). In regards to the Electronic set, we are able to reduce the number of reviews by 64% (2.79 K reviews), 42% (4.50 K reviews), and 7% (8.35 K reviews), respectively. By following the k-core sampling technique we are able to trim down the dataset to retrieve more meaningful data, the actual review user and item count is shown in the table below, where S is representing single domain and C represent the collective domain:

Table 6. K-cores comparison in different user and items count

	K=20		**K=15**		**K=10**	
	S	**C**	**S**	**C**	**S**	**C**
Total Users	8,357	22,933	17,795	37,796	49,072	76,219
Total Items	42,771	65,429	57,341	84,277	8,.932	119,374
Total Reviews	278,970	1,303,270	449,841	1,634,836	835,578	2,210,181

Several figures below also show the number of changes in the rate per user and rate per item distribution for each of the K values, which start on the Electronics set below.



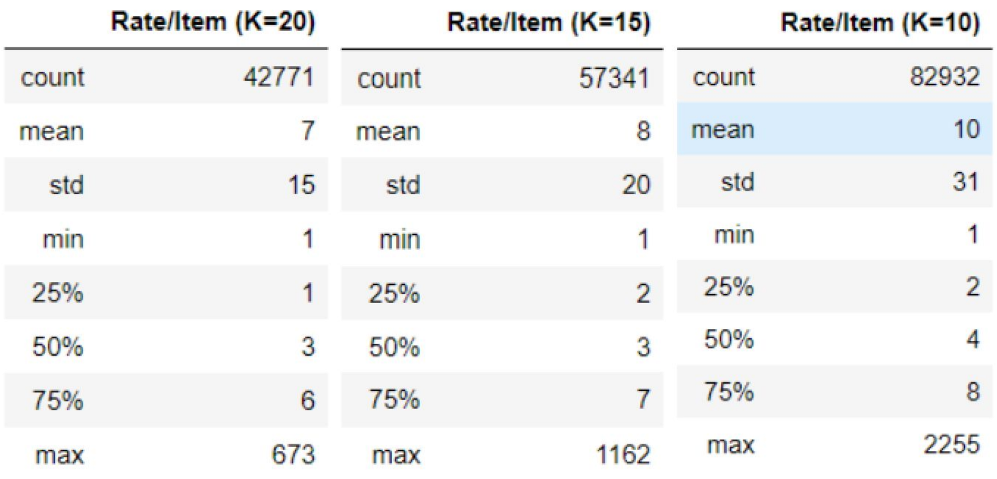
	Rate/User (K=20)
count	8357
mean	33
std	23
min	20
25%	22
50%	26
75%	35
max	395

	Rate/User (K=15)
count	17795
mean	25
std	19
min	15
25%	16
50%	20
75%	27
max	412

	Rate/User (K=10)
count	49072
mean	17
std	14
min	10
25%	11
50%	13
75%	18
max	439

Figure 7. K-core Analysis on users Single Domain (Electronics)

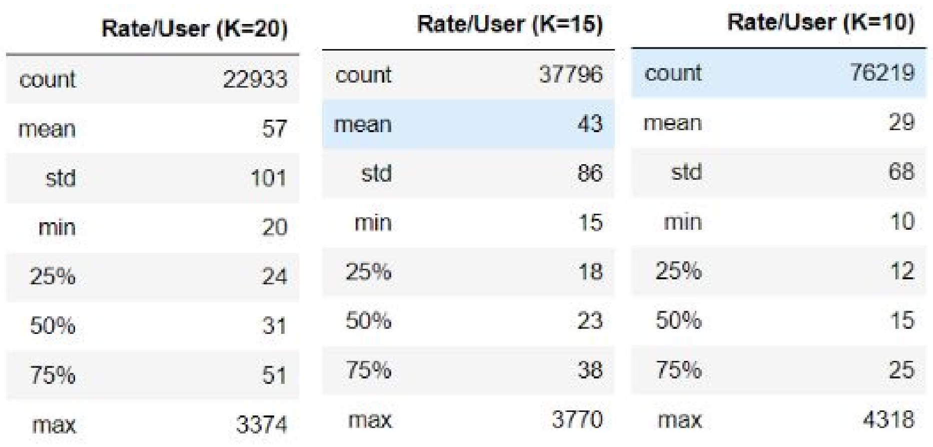
From the generated figure (**Figure 7**), we know that when K = 20 can produce more meaningful data and better spread out. Where around 50% of the user are made rate 26 times other than 20 and 13 times with other cores (15 and 10 respectively). However, applying k-cores sampling did not make much improvement to fix the poor sparsity on the rating per item distribution. Referring to **Figure 8** below we can see that the rating per item for all cores values are still not very good where almost 75% of objects are only rated one to three times.



	Rate/Item (K=20)		Rate/Item (K=15)		Rate/Item (K=10)
count	42771	count	57341	count	82932
mean	7	mean	8	mean	10
std	15	std	20	std	31
min	1	min	1	min	1
25%	1	25%	2	25%	2
50%	3	50%	3	50%	4
75%	6	75%	7	75%	8
max	673	max	1162	max	2255

Figure 8. K-core Analysis on items Single Domain (Electronics)

Continue to the CD & Movies (Collective set) in **Figure 9**, utilizing k-core sampling techniques are also produce a similar result where we are also able to get a smaller data compared to the original data. The cut-down percentage of each core for this set is 84% (22.93 K users and 65.43 K items), 80% (37.78 K users and 84.28 K items), and 74% (76.22 K users and 119.37 K items) for cores equal to 20, 15, 10 respectively. The figure below shows the new distribution of rate per user after using k-core sampling



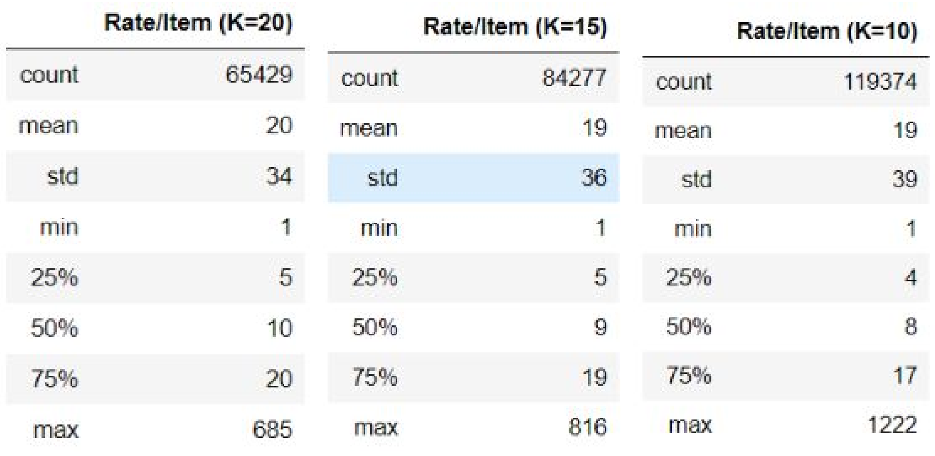
	Rate/User (K=20)
count	22933
mean	57
std	101
min	20
25%	24
50%	31
75%	51
max	3374

	Rate/User (K=15)
count	37796
mean	43
std	86
min	15
25%	18
50%	23
75%	38
max	3770

	Rate/User (K=10)
count	76219
mean	29
std	68
min	10
25%	12
50%	15
75%	25
max	4318

Figure 9. K-core Analysis on users Collective Domain (CD & Movies)

From the data above (**Figure 9**), the cores sampling able to improve the distribution of rate per user. Where the cores equal to 20 is considered as the most better cores. And not like the Electronics set, the distribution of rate per item seems to be slightly improved, where we can now have around 10 rated items in 50% on each population, and top 25% is measured 20 times (for 20 cores). The pictures **(Figure 10)** below show the updated rate per item distribution.



	Rate/Item (K=20)
count	65429
mean	20
std	34
min	1
25%	5
50%	10
75%	20
max	685

	Rate/Item (K=15)
count	84277
mean	19
std	36
min	1
25%	5
50%	9
75%	19
max	816

	Rate/Item (K=10)
count	119374
mean	19
std	39
min	1
25%	4
50%	8
75%	17
max	1222

Figure 10. K-core Analysis on items Collective Domain (CD & Movies)

In conclusion, we can generate and analyze the various data based on specific cores for the two categorical data that we use. We will then proceed in working with the three different K's that we have discussed earlier for data sampling to analyze the relationship between how the data are distributed and the performance of the system.

4.4 Testing

To test the implemented system, we follow a specific direction :

- We run each of algorithm three times to find the average MAE and RMSE.
- All of the algorithms have also experimented with the tuned parameter, such as setting the number of epoch, number of clusters and K values.
- The test method are conducted in the same environment and specification to minimize bias in the result.

5. Resources

5.1 Hardware & Software

To run the project, we are using the google cloud platform virtual machine with specification as follows:

- Hardware: 8 vCPUs with 40 & 60 GB of RAM
- Software: Python 3.6, python library for data analysis, visualization and recommendation system (matplotlib, seaborn, surprise, tensorflow, Keras)

5.2 Materials

Additional readings material that did not get to be used as a supporting argument:

- WTF : The who to follow service at Twitter (Gupta, P., Goel, A., Lin, J., Sharma, A., Wang, D., & Zadeh, R., 2013)
- Deep Neural Networks for YouTube Recommendations (Covington, P., Adams, J., & Sargin, E., 2016)
- Spatially Aware Recommendations Using K-D Trees (Das, J., Majumder, S., & Gupta, P., 2013)
- Deep content based music recommendation (Aaron, V.D.O, Sander, D., Benjamin, S.2013).

- An Approach to A University Recommendation by Multi-Criteria Collaborative Filtering and Dimensionality Reduction Techniques (Bokde, D., Girase, S., & Mukhopadhyay, D., 2015)
- A Hybrid system, for personalized content recommendation (Ye, B., Tu, Y., & Liang, T., 2019)

5.3 Roles & Responsibilities

Below is a table listed to indicates the current role and responsibility relative to the project progress only, note that team management role and responsibilities is not listed in this table.

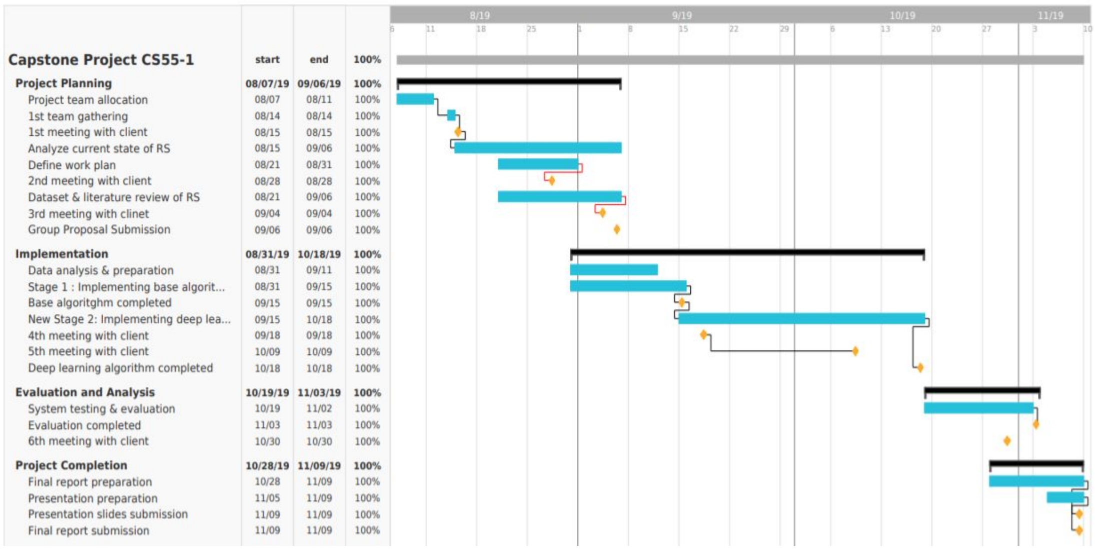
Table 7. Team members roles and responsibilities

No	Group Member	Roles & Responsibilities
1	Deniz Doruk Nuhoglu	1. Analyze and Research paper reading 2. Deep learning implementation with focus on baseline autoencoder 3. Data cleaning
2	Jingwen Wang	1. Analyze and Research paper reading 2. Deep learning Implementation with emerging factor on how to extend current algorithm to solve some specific questions (research side) 3. Data sampling discussion
3	Madanagopal Pasupathy	1. Analyze and Research paper reading 2. Deep learning Implementation with emerging factor on how to extend current algorithm to solve some specific questions (research side) 3. Reference list gathering
4	Ming Sheng Choo	1. Analyze and Research paper reading 2. Deep learning Implementation with focus on deep sparse autoencoders with embeddings 3. Model Evaluation analysis 4. Final report write up

5	Mohamad Reza Pancawan	1. Analyze and Research paper reading 2. Baseline algorithm Implementation with focus on collaborative filtering 3. Implementation of utility function (graphs functions) 4. Final report write up

6. MILESTONES / SCHEDULE

We completed the project in 14 weeks, started on 7th August and ended on 9th November. During the project phases, we achieved 12 milestones, which include the completion of base algorithm, implementation of deep learning algorithm, system testing & evaluation, as well as the end of the final report. The completed project schedule along with the list of milestones in each phase are shown below



Capstone Project CS55-1	start	end	100%
Project Planning	**08/07/19**	**09/06/19**	**100%**
Project team allocation	08/07	08/11	100%
1st team gathering	08/14	08/14	100%
1st meeting with client	08/15	08/15	100%
Analyze current state of RS	08/15	09/06	100%
Define work plan	08/21	08/31	100%
2nd meeting with client	08/28	08/28	100%
Dataset & literature review of RS	08/21	09/06	100%
3rd meeting with clinet	09/04	09/04	100%
Group Proposal Submission	09/06	09/06	100%
Implementation	**08/31/19**	**10/18/19**	**100%**
Data analysis & preparation	08/31	09/11	100%
Stage 1 : Implementing base algorit...	08/31	09/15	100%
Base algoritghm completed	09/15	09/15	100%
New Stage 2: Implementing deep lea...	09/15	10/18	100%
4th meeting with client	09/18	09/18	100%
5th meeting with client	10/09	10/09	100%
Deep learning algorithm completed	10/18	10/18	100%
Evaluation and Analysis	**10/19/19**	**11/03/19**	**100%**
System testing & evaluation	10/19	11/02	100%
Evaluation completed	11/03	11/03	100%
6th meeting with client	10/30	10/30	100%
Project Completion	**10/28/19**	**11/09/19**	**100%**
Final report preparation	10/28	11/09	100%
Presentation preparation	11/05	11/09	100%
Presentation slides submission	11/09	11/09	100%
Final report submission	11/09	11/09	100%

Figure 11.Project schedule gantt-chart

As shown in the Gantt-chart (**Figure. 11**), we completed four milestones in the Planning phase (7th August to 6th September), which are the first three group meetings with the project client, and completed the group proposal. Then we can finish four milestones in Implementation phase in around one and a half months (31st August – 18th October). The completed milestone in this phase is the implementation of base and deep learning algorithms as well as completion of two meetings with the client. As for the next two milestones are done in the Evaluation & Analysis phase, while another two milestones which related to the final report and presentation slides are also finished in the following stage (Project Completion).

7. Risk Assessment

To ensure the uncertainty are properly managed, we listed some of the high risks that might occur during project completion along with its impacting level and mitigation plan in table below.

Table 8. Risk assessment (High Risk)

Risk Name	Risk description	Impact	Action & Mitigation Plan
Dataset issue	Cannot load the full set of Amazon dataset on our local machine.	High	● In talks with the course coordinator in granting access to cloud VM **(Appendix A).** ● As an alternative google GCP free account could also be considered **(Appendix B)**. ● Use subset data based on category (e.g. Electronics, CD, Movie, etc).
Resources issue	The free-tier cloud environment may be running out of credit or has library issues that make the code won't work properly	High	● Create several cloud environments with the same specifications. ● Only install the application, library, modules that are needed on the project (Anaconda, Jupyter, Tensorflow, etc.).
Methods complexity	There are too many combinations of algorithms, and it	High	● Choose some of the most popular and widely used algorithms. ● Focus the proposed method to one specific algorithm which is the autoencoders

	is hard to find the best one.		
Poor communication	Improper communication between team members and project client	High	● Held regular meetings with project clients to ensure all the issues are well communicated **(Appendix C)**. ● Weekly progression update with the tutor to ensure we are moving on the right track. ● Online daily collaboration on Slack, Trello. **(Appendix E)**. ● Communicate individual's findings to other team members.
The project failed to finish on time	Delayed in project completion	High	● Day-to-day meeting ● Follow the planned agenda **(Appendix D)**. ● Better time management.
Quality risks	The risk that related to the project scope & goals with the result	High	● Define the clear project scope and objectives. ● Set a comprehensive method to tackle project problems. ● Define all the project limitations related to the project outcome.
Program failure	The codes failed to run on a different machine that affects the project time and quality.	High	● Only install the application, library, modules that are needed on the project (Anaconda, Jupyter, Tensorflow, etc.). ● Test the codes on multiple environment.

8. RESULTS

As we have discussed in earlier section, Amazon dataset is extremely skewed dataset as compared to other traditional research dataset such as Movielens, Jester dataset, Book Crossing and also Netflix data, we argued that the dataset is handpicked hence it is in a more controlled environment, hence it is relatively easy for algorithms to achieve good results, so we are really interested in applying our own approach and some well known state-of-the-art algorithm to see if the methods can still be robust in such a skewed setup.

- **Single Domain - Electronic set**

  The table (**Table 8**) shows the results of all the algorithms that we have run in both single domain and collective dataset. The parameter chosen for autoencoders is [512, 256] for both encoder and decoder hidden layer with 0.01 L2 regularization, and with an arbitrary value of 5 as the latent factor for both users and items embeddings, tanh is the chosen activation function as we found that it provides the best convergence rate which we will further discuss in later sections.We have ran deep sparse autoencoder with minimal tuning done to the model, and we would like to see how well it can compare with the other model.

Table 9. Performance of each algorithm in K = 20, lower is better

Algorithm	MAE	RMSE
Deep Sparse Autoencoder	**0.700(±0.005)**	1.07(±0.006)
Autoencoder	2.024(±0.019)	1.486(±0.022)
SVD	0.735(±0.0004)	1.007(±0.014)
SVD (Tuned)	0.735 (±0.0014)	**1.003(±0.002)**
Co-clustering	0.758(±0.001)	1.103(±0.013)
Co-clustering (Tuned)	0.761(±0.001)	1.106(±0.010)
Item-based KNN	0.794(±0.001)	1.146(±0.002)

When K of 20 is used for the dataset, SVD has the overall best performance out of all the algorithm that we have in the table, surprisingly our deep sparse autoencoder is closely after where the best MAE score but with worst RMSE result compared to funk SVD. The rest of the algorithm has struggled to achieve RMSE of lower than 1.1, the baseline autoencoder with high dropout rate (0.8) is just as we have predicted where it will not work well in a skewed dataset, and baseline autoencoder seems to run for longer compared to other algorithms.

Table 10. Performance of each algorithm in K = 15, lower is better

Algorithm	**MAE**	**RMSE**
Deep Sparse Autoencoder	**0.7285(±0.002)**	1.079(±0.006)
Autoencoder	2.03(±0.025)	1.586(±0.022)
SVD	0.756(±0.0001)	1.032(±0.002)
SVD (Tuned)	0.757 (±0.0004)	**1.029(±0.002)**
Co-clustering	0.777(±0.002)	1.117(±0.004)
Co-clustering (Tuned)	0.761(±0.001)	1.127(±0.005)
Item-based KNN	N/A	N/A

Similar situation can be observed from the table above when K=10 is used for pre-processing for the review data, where again SVD proven to be closer to the true ratings while our proposed method has achieved lower MAE scores. Item-based KNN failed to run in this experiment as it is expected due to the scalability issue of collaborative filtering, again baseline autoencoder still does not perform well when more users and items is included for the training, even though more data does not mean that we will converge to expected error, it could also be a risk we are adding in more potential outlier into the model.

Table 11. Performance of each algorithm in K = 10, lower is better

Algorithm	**MAE**	**RMSE**
Deep Sparse Autoencoder	0.795(±0.002)	1.153(±0.038)
Autoencoder	N/A	N/A
SVD	0.7867(±0.0009)	1.069(±0.002)
SVD (Tuned)	**0.735 (±0.0014)**	**1.064(±0.002)**
Co-clustering	0.803(±0.001)	1.146(±0.013)
Co-clustering (Tuned)	0.81(±0.001)	1.161(±0.001)
Item-based KNN	N/A	N/A

From the table above, when K=10 is used for filtering for users and items we can see that the previous observation does not hold true anymore as SVD has both the best in MAE and RMSE score while our proposed has similar score comparable with co-clustering in this situation. Both KNN and baseline autoencoder still continue to suffer from scalability issue when trying to construct users and items rating matrix.

From all the table in analysing the K with all the different algorithm so far, we know that SVD is robust in most situations where it managed to achieve the best RMSE score, even with a high sparsity rate (lower K) it continues to have the best overall recommender system in terms of RMSE score, while our proposed method managed to have better results in MAE meaning that our model is sensitive to extreme cases and outliers but abit more robust compared to SVD, we believe that with further tuning we will be able to get better results compared to SVD.

- **Collective Domain - CD & Movies**

  As mentioned in earlier section, we are interested to see if we stacked up two different domains will the algorithm still remain robust. Hence we have proceed to work on creating a collective domain dataset by taking from 2 different sub-category such as Movies & TV and CDs and Vinyl, by inner join the two dataset by using reviewerID as key, we have managed to retain all the users that has cross buying history, hence we can continue our experiment with the collective dataset created.

Table 12. Performance of each algorithm in K = 10, lower is better

K= 20		
Algorithm	**MAE**	**RMSE**
Deep Sparse Autoencoder	**0.704**	1.165
Autoencoder	N/A	N/A
SVD	0.711	0.962
SVD (Tuned)	0.713	**0.955**
Co-clustering	0.712	0.995
Co-clustering (Tuned)	0.714	0.997
Item-based KNN	N/A	N/A
K=15		
Algorithm	**MAE**	**RMSE**
Deep Sparse Autoencoder	**0.711**	1.06
Autoencoder	N/A	N/A
SVD	0.713	0.965
SVD (Tuned)	0.715	**0.957**
Co-clustering	0.712	0.998
Co-clustering (Tuned)	0.713	1.002
Item-based KNN	N/A	N/A
K=10		
Algorithm	**MAE**	**RMSE**
Deep Sparse Autoencoder	0.726	1.08
Autoencoder	N/A	N/A
SVD	0.719	0.972
SVD (Tuned)	0.72	**0.965**

Co-clustering	**0.713**	1.011
Co-clustering (Tuned)	0.715	1.015
Item-based KNN	N/A	N/A

Similar outcome could be observed on the experiment in collective domain, where our proposed method has still the best MAE compared to all of the listed algorithm but this time our proposed deep sparse autoencoder achieved worst RMSE score compared to single domain while the other algorithm seems to have improved RMSE score. With the current result, one can bring up a conclusion where collective domain is great for recommendation system where it will bring positive impact to the algorithm in learning patterns of the user, but we need more test and other different domains combination before we can make that conclusion. But as a first step, we think that collective domain does not seem to add additional noise to the model but it may also brings improvement to the algorithm RMSE score.

From all the experiments so far we believe that our proposed model has potential for further improvement and getting a better RMSE score is not impossible. From all the listed results from either single domain and collective domain we know that our model is capable of capturing the latent factor of each user and item pair but still not quite good at predicting the actual ratings of the user.

- **Top N analysis**

In this section, we tend to analyze the top-N list that has been generated by deep sparse autoencoders. The study will compare the purchase history for a random user and the top-n list from three different k sampling (20,15,10) of the Electronics dataset. The table below is the purchase history of the user 1812. To make the analysis easier, we also conducted a further investigation to classified the item type in general such as camera, bags/case, cable, tools/utilities, etc.

Table 13. Top-20 user (ID=1812) purchase history

Rating	Item	Type
5	Canon HF-DC1 High Power Flash for Digital Cameras	Camera
5	Clamshell CD Protective Case, 25 Pack with Rack	tools / Utilities
5	1-Foot Extension Power Cable, 5-Pack	Cable
5	Philips SPP1591WA Squid Surge Protector with 5 Outlets	Cable
5	Memory Card Carrying Case - Black (Generic)	Bags / Case
5	USA Gear Large GPS Navigation Travel Case for ...	Bags / Case
5	Case Logic DCB-302 Compact Camera Case (Gray)	Bags / Case
5	Olympus CS-125 Soft Carrying Case for WS Serie...	Bags / Case
5	2 Pack Battery And Charger Kit For Canon Power...	Camera
5	The Friendly Swede (TM) Bundle of 4 Colorful 6...	Cable
5	ClearMax&reg; NB-6L Lithium Ion Replacement Ba...	Camera
4	Premium 10 feet Canon HTC-100 Upgrade Replacem…	Cable
4	Koss 'The Plug' In-Ear Headphones (Black)	Earphone
4	Skooba Design Cable Stable DLX	Bags / Case
3	Canon PowerShot SX260 HS 12.1 MP CMOS Digital ...	Camera
3	Fosmon Trio Capacitive Stylus	tools / Utilities
2	Handstands Original Sticky Pad (13810) (13810)	tools / Utilities
2	Canon PSC-3200 Deluxe Leather Case	Bags / Case
2	Olympus WS-802 Voice Recorder	tools / Utilities

Referring to the table above, we know that the user bought six bags/case, four items each for camera-related, cables, and other tools/utilities respectively. In this case, now we can make a subjective conclusion about the user preferences that can we use to measure the recommendation system performance differently (without evaluation metrics). As for the top-20 list generated for K=20 is shown in the table below.

Table 14. Top-20 user (ID=1812) recommended items with K=20

Prediction	Item	Type
5	XtremeMac White Silicone Case for iPod Mini	Bags / Case
5	Nikon SK-9 Flash Bracket Coolpix 2100, 3100 Cameras	Camera
5	AMD ATHLON XP 2000 CPU PALOMINO CORE ...	Computer
5	JVC RX-8000VBK Dolby Digital/DTS Audio/Video R...	tools / Utilities
5	Lexar Media 1 GB USB Jump Drive Elite (JDE1GB-...	tools / Utilities
5	Vivitar PZ3140 Zoom Date 35mm Camera	Camera
5	70IN Zipper Bag for Tri-pod Screen	Bags / Case
5	Canon WL-DC100 Wireless Controller PowerShot P...	Camera
5	Generic Standard Red LED Wall/ AC/ Home Charge...	tools / Utilities
5	InFocus LP640 Business LCD Video Projector	tools / Utilities
5	Nikon Coolpix 8400 8MP Digital Camera with 3.5...	Camera
5	Samsung HT-DB120 Progressive-Scan DVD Home The...	Home theatre
5	Cambridge SoundWorks Outdoor 42 All-Weather Ou...	Speaker
5	Panasonic Lumix DMC-FZ1S 2 MP Digital Camera w...	Camera
5	Phoenix Gold VRR-120 BONE 120 Watt Transformer...	Cable
5	Pyramid 2706D AM/FM/Cassette Player with Detac...	Music Player
5	StarTech PXT101NB 6 ft Standard Laptop Power C...	Cable
5	SMARTHOME INC ATH-AAA INT RF ADAPTER...	tools / Utilities
5	Averatec AV3255P1-01 Laptop (AMD Athlon XP-M 2...	Computer

According to the top-20 recommendations, the deep sparse autoencoders can generate products that share a similar type with his purchase history, where the system recommended six items related to the camera, five pieces of tools/utilities, two items for bags/case and cables. It means that the system 75% (15 out of 20) of items. In regards to the result, we also aware that the top-10 results (except computer) heavily related to the user's most wanted items, which are the bags/case and camera. Therefore, we think the model is able to learn user preferences and histories. However, the system also generates some types of items for the user, such as a computer, home theatre, music player, and speaker.

As for the top-20 result of the k=15 sampling method below, we argued that the system is generated an imperfect list compared to when we are using the k=20. As we can see on the top-10 items of the records, the systems were produced random items such as sports tools, monitors, speakers, and walkie talkie, which is not present on the top preferences and history of the users. Additionally, we are also unable to find the popular items for the user, such as bags/case and camera on the list. However, despite the poor result, the system can also generate seven out of twenty items (35%) similar items with user preferences such as cable and tools/utilities that might attract the user interest.

Table 15. Top-20 user (ID=1812) recommended items with K=15

Prediction	Item	Type
5	Golf Stat Tracker II G-100	Sport tools
5	Samsung LNS2352W 23-Inch LCD HDTV	Monitor
5	Sony SS-MF550H 3-Way Floorstanding Speakers (P...	Speaker
5	JVC FS-X1 CD Shelf System (Silver)	Speaker
5	Happy Hacking Keyboard Lite2 USB (Black)	tools / Utilities
5	Unwired Technology UFR6802 2-Mile 14-Channel F...	Walkie talkie
5	JVC XL-PG39 Personal CD Player with ASP-EXtrem...	Music Player
5	Sony ZSSN10PS PSYC CD Boombox (Black)	Music Player
5	Belkin Surgemaster 6 Outlet 3 Foot Cord 585 Jo...	Cable
5	Palm TX - Palm OS Garnet 5.4 312 MHz	Phone
5	Sharp SD-AT1000 600 Watt Home Theater Audio Sy...	Home theatre
5	Vantec Tornado TD8038H 80x80x38mm Double Ball ...	Computer
5	iAudio U3 2 GB MP3 Player	Music Player
5	Belkin 3-Button Mouse PS2 Only. Black (F8E813...	tools / Utilities
5	HP DVD+R 4.7GB 16X White Inkjet Printable 100 ...	tools / Utilities
5	Interlogix NetworX 16 Zone Hardwired Expansion...	tools / Utilities

5	Slimline Null Modem Adapter, DB9 Male / Female	tools / Utilities
5	Memorex MVD4540 DVD-VCR Dual-Deck Player	DVD Player
5	Philips AZ1565 CD Radio Cassette Recorder with...	Music Player

Lastly, the following table shows the top-20 items for when we are using K=10. Based on the result we can conclude that the K=10 were also produced several random things just like we got from the previous result. Nonetheless, we also think that the system can generate more related items such as two items related to camera and bags/case, which we know that items are the user's most purchased items. Additionally, the list is also contained several items that still correspond to user preferences such as cable, tools/utilities despite its also recommend items like music player, speaker, and computer-related items. As for the conclusion, we see that half of the recommended items (50%) are similar to the user history and preferences.

Table 16. Top-20 user (ID=1812) recommended items with K=10

Prediction	Item	Type
5	Pre-Owned 80GB Video iPod - Black (5.5 Generate...	Music Player
5	F8V3080-DL2 -TUNECAST II TRNSMTTR DIGTLW/...	Music Player
5	Prime CR220625 25-Feet 16/3 SJT Cord Caddy wit...	tools / Utilities
5	Bazooka MT8002WS 8-Inch Marine Tubbie (White)	Speaker
5	StarTech.com 4 DRIVE 68 PIN U320 SCSI (SCSI32...	Computer
5	Adorama Pro Photo Album, Burgundy Leatherette ...	Cable
5	Digipower TP-S032 Compact tripod with two sect...	Camera
5	Intel P4-2.26ghz 533mhz Fsbpga478	Computer
5	Creative Labs Webcam Notebook Camera with Clip	Computer
5	BoxWave Elite Leather Apple Macbook Air 11&quo...	Computer
5	Rotating Imitation Security Camera	tools / Utilities
5	CAT5E, UTP, with Molded Boot, 350MHz, White, 7...	Cable
5	C2G / Cables to Go 43053 .68in Self-Adhesive C...	Cable

5	Sony Clip-on Stereo Headphones with Retractabl...	Headphone
5	AC-LS1A SONY AC-LS1A SONY AC-LS1A	Camera
5	JBL L810 3-Way High Performance 5.25&quot; Wal...	Speaker
5	BoxWave Google Nexus 7 Swivel Stand Case - Syn...	Gadget
5	Grey 2.4GHz wireless optical mouse	tools / Utilities
5	Tonino Lamborghini LM-2104 Camera Case Carry Bag	Bags/Case

9. Discussion

- **Top N discussion**

Based on our top N analysis, we conclude that the bigger K number is, the better. We think this is also related to the long tail phenomenon, which illustrated in the (**Figure. 12**) below.

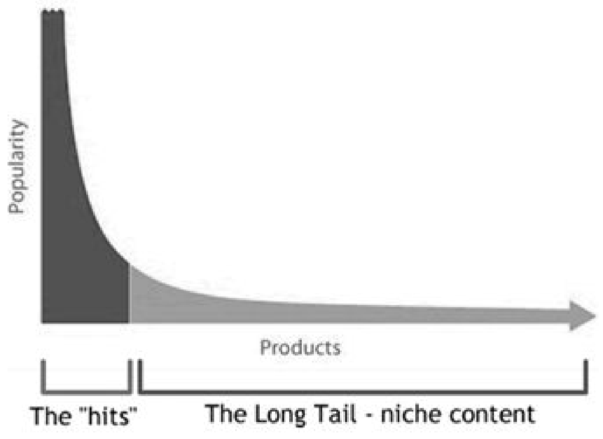


Figure 11. The long-tail curve (Krasinski, M. 2019)

A big K number is not only giving us a smaller set to work with, but it also allows us to generate more meaningful data, which is similar to get a ‘hits’ or popular items. The small K number, on the other hand, will give us a more significant set as well a bunch of unpopular things from the tail of the curve, which might provide some diversity in a sense. As for the further discussion of our findings, the table below shows the summary of the item similarity between the user purchase history & preferences and all the generated top-20 & 10 items for each K.

Table 17. Top-N similarity comparison

Number of K	Top-20 Similarity	Top-10 Similarity
20	75%	90%
15	35%	20%
10	50%	30%

Based on the table, we can argue that the proposed deep sparse autoencoders can learn and predict better with the bigger K number, which also supported by another experiment result that using offline evaluation metrics. For example, with K=20 we can get 15 from 20 items that are similar to the user preferences, but we only get seven items from twenty when we changed the K to 15.

Further analysis also shows that K=20 can get nine related items out of the first ten items. On the other hand, when we applied K=15, we end up generating items that are matched with the user preferences that can also be considered as the unpopular items, despite the system seen them as similar items. However, we believe that an excellent recommendation system should not recommend something out of user preference or unpopular things every time.

Finally, we realize that this top-N analysis might be subjective, and the real performance of the systems can only be determined through the real online environment. However, we also believe that the top-N analysis can also be an alternative or to supporting our study in defining the performance of the system rather than only considering the offline metrics measurement like MAE and RMSE.

- **Latent factor of user and item embeddings**

Our proposed model has a pre-processing layer before stacking on top of our deep sparse autoencoders, the final model that we have chosen is to concatenate instead of dot product of two embeddings layer, we have managed to achieve a slightly better results when we use concatenate before feeding into the sparse autoencoder.

For this experiment we will use the following fixed other parameters and only change the latent factor size in embedding layer, and to simplify the experiment we will assume that both user and item has the same importance hence the size will stay the same.

below is the settings used for the experiment:

activation = 'TANH', reg_rate = 0.001, pre-process method = concatenation, layers = [512, 256]

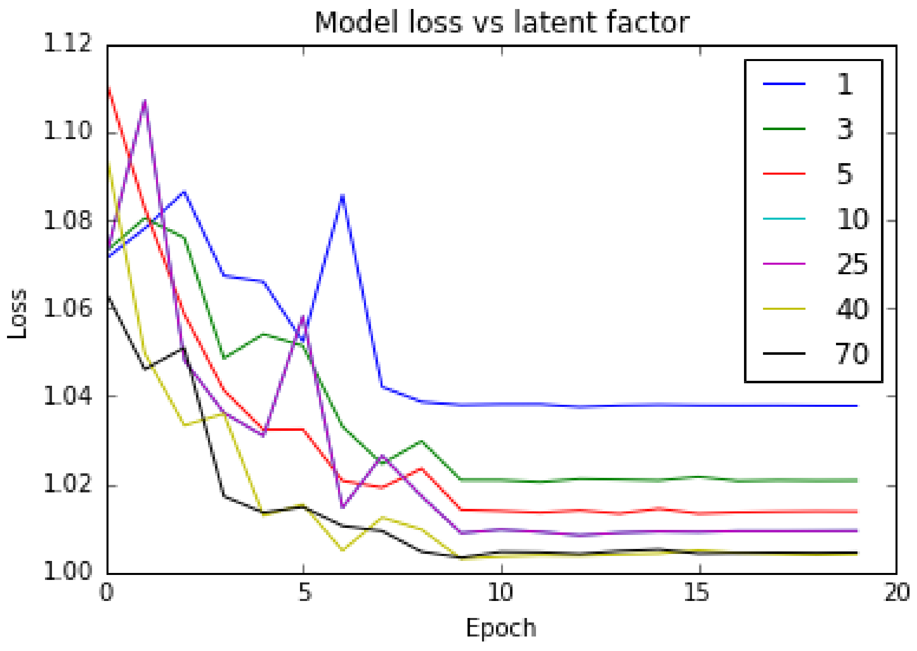


Figure 12. Comparison of different latent factor settings

As shown in the figure above **(Figure 12)**, validation error will go down when we gradually increase the size of latent factor, however it will stop converging after the size at around 40 and it will stop converging beyond that point, grid search seems to be the only option on selecting of the best K or Rank in matrix factorization.

- Convergence rate of different activation settings

In this section we will explore the convergence rate of different activation settings, for this we chose the following activation for our experiment TANH, RELU, SELU, LINEAR. From our experiment and the graph provided below **(Figure 13)**, we found that tanh, selu, and linear works better than relu, which is surprising where in the deep autoencoder paper they find that ELU, SELU works better than RELU, SIGMOID and TANH, which is not really the same findings as our own, probably mosting due to our structure is different compared to theirs now. The experiment for this activation run is listed as follows:

latent_factor = 40, reg_rate = 0.001, pre-process method = concatenation , layers = [512, 256]

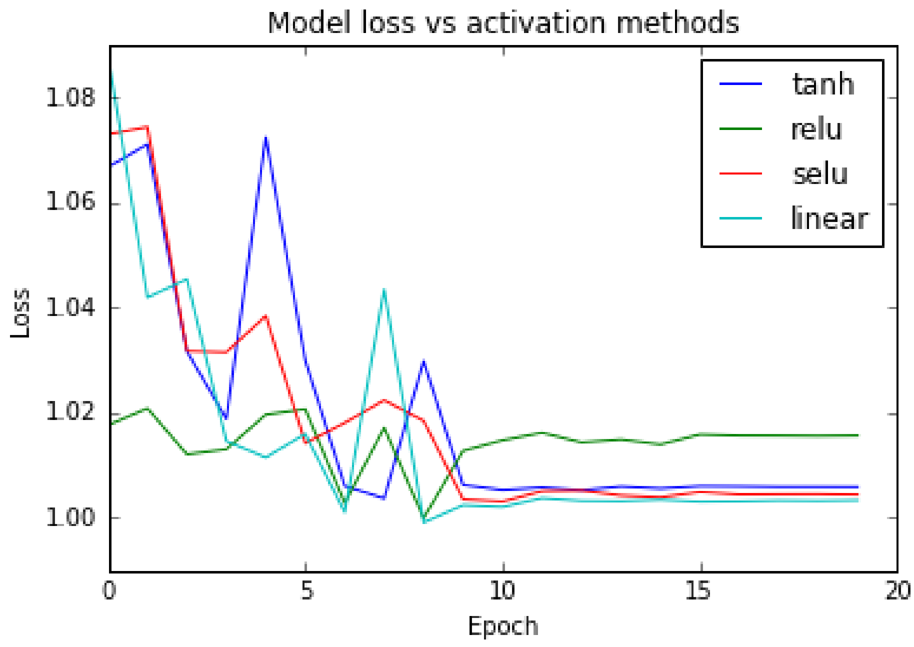


Figure 13. Different Activation methods impact to model convergence

- Depth of autoencoder analysis

In this section we wanted to see if adding depth to our autoencoder increase the performance of the mode, we have done two different starting layer size with a total of 8 different layering settings. Some notation to note, [512, 256] meaning that we will be using a 4 layer autoencoders such that 512 , 256 for the encoding layer and 256, 512 for the decoding layer, hence [512, 256, 128] is a 6 layers autoencoder, [512, 256,128,64] will be an 8 layer autoencoder, layer also does not include the bottleneck layer that is used as a dropout layer.

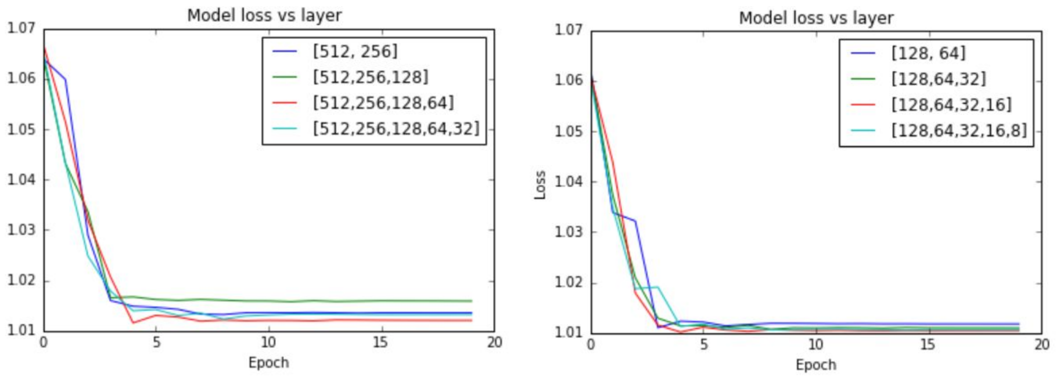


Figure 14. Layer numbers comparison

From the experiment above **(Figure 14)**, we can observe that the assumption where deeper autoencoders yield better compression results, and for our case 8 layers autoencode actually provides better results compared to the other layering setups. To better understand the experiment we will also provide a table with all the details of the experiments with RMSE results.

Table 18. Evaluation of RMSE with different number of layers

layers	Number of Layer	params	RMSE
[128,64]	4	2,076,433	1.0766914
[128,64,32]	6	2,077,521	1.0771241
[128,64,32,16]	8	2,077,809	**1.0749421**
[128,64,32,16,8]	10	2,077,889	1.0792022
[512,256]	4	2,415,889	1.0798911
[512,256,128]	6	2,432,529	1.0792271
[512,256,128,64]	8	2,436,753	**1.0777065**
[512,256,128,64,32]	10	2,436,753	1.0779641

From result table above **(Table 18)**, we can observe that we can achieve the lowest RMSE with 8 layers of autoencoders, which proves that the basic assumption of depth can lead to better performance still holds in this situation. However when layers goes beyond 8 then we can observe that the RMSE starts to increase again. And starting with 128 layer size is better for our experiment, might have due to our dataset is extremely skewed and causing the K-core to pick up lesser data hence we need to employ model with lesser complexity to effectively generalize the data.

- Hyperparameter tuning

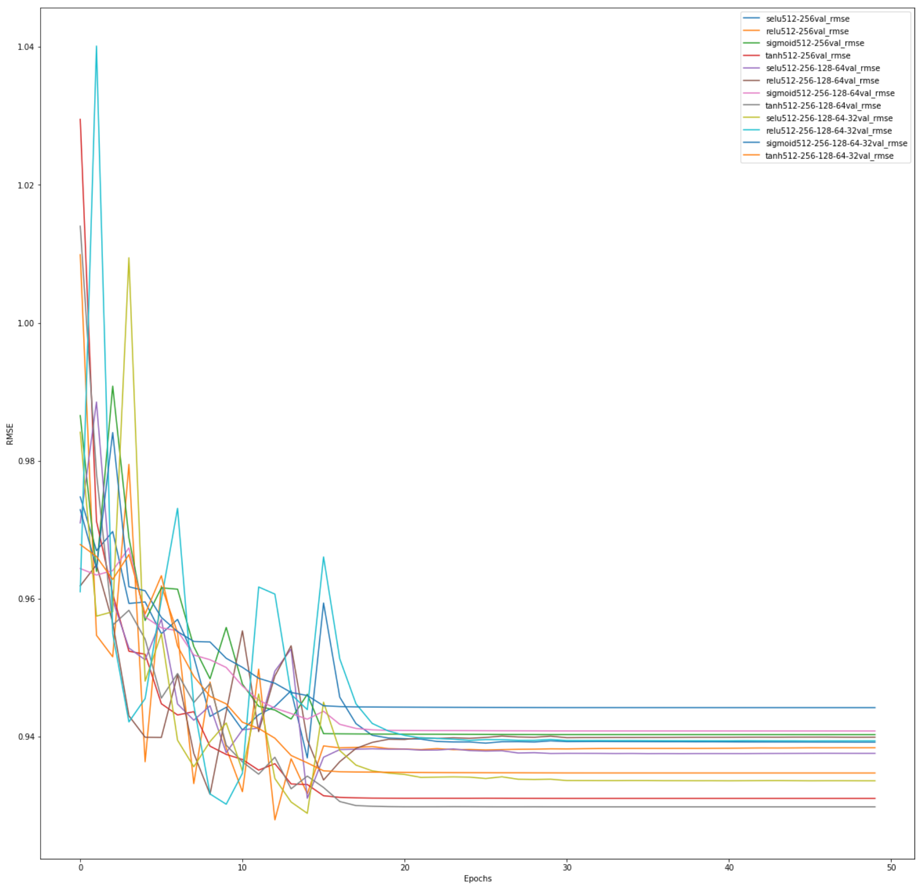


Figure 15. Layer numbers comparison

For hyperparameter tuning, we have used both Talos and sklearn gridsearch for this purpose, this is to find the best combination of hyperparameter to better generalized the given dataset. In other words, we are trying to have the model to have the best performance possible when given the dataset. TANH with 8 layers autoencoder seems to work the best in terms of the actual convergence rate when compared to other combination, the final tuned parameters is similar to our findings and discussion in the individual parameter evaluation above.

- Conclusion

  In conclusion, we have proposed an autoencoder approach where we have achieved better mean absolute error result across all other baseline methods that we have compared with, though it slightly fall short in RMSE score where SVD is still the overall more robust algorithm. But we believe the proposed method by blending in our own approach and unique ideas from other papers, the current autoencoder shows good potential, it will have better accuracy performance with more tuning and some change of structure. Though how to filter out meaningful data to train the algorithm remains an open question and we hope that we can further look into the matter in the future.

10. Limitations and Future Works

We have defined some limitations related to this project which we describe as follows:

- Collective domain currently only contains 2 subcategories out of the full 22 categories, but by doing that we can better analyze and run the experiments without any resource constraints.
- Due to the resource constraints and scalability issue of some algorithms, we were not able to run KNN and baseline autoencoders on lower K's and collective domain, due to the memory constain when trying to generate users and items matrix.
- We are not able to fully utilize TPU/GPU training as there is credit constraints on cloud VM and also memory constraints on google collab.
- We were not able to fully utilize the Amazon dataset as we are only working on collaborative filtering, some of the meta data feature is ignored for now.

As for several future works that we suggested for the next research are:

- As discussed earlier, as we have laid down a solid foundation, utilizing auxiliary information on items instead of just feeding the autoencoders with ratings. To elaborate further, we could use an item description and the product review as additional input data to the autoencoders. Other attributes such as sales-rank could also be applied to create a variational top-n recommendation list.
- Currently we are relying on K-core to filter out unnecessary data combining with stratified sampling methods during train, test and validation split, we can also investigate other sampling methods such as Markov Chain sampling, to see if that will provide any significant impact in training the network.
- Implement the current deep sparse autoencoder structure to different variant of autoencoders, such as LSTM or GRU autoencoders.
- Though we have done some evaluation on different parameters, but we have yet to investigate in-depth on other layering option such as the potential of batch normalization of layers, training batch size, and even different optimizer.
- Though we do have additional features to round up and clip the prediction of ratings, we suggest to investigate into different loss functions such as cauchy loss function and welch loss function, and custom activation methods to increase the efficiency and overall performance of the model.
- Generate more evaluation matrices such as diversity, hit-rate that could be used to learn the performance of the system.

11. DOCUMENTATION

Here we will provide some documentation of the codes and the programs we used.

Table 19. List of Python's libraries

library	version
pandas	0.23.4
numpy	1.15.4
tensorflow	1.13.1
keras	2.2.4
sklearn	0.21.2

DeepSparseAutoencoders with embeddings.ipynb

The file contains two classes and all the necessary codes to run the model.

data_prep class: call this class to perform data pre-processing on the amazon dataset.

data_prep(fileName, K_core_item, K_core_user, convert)

parameters :

fileName (string): path to the ratings file.

K_core_item (int): Include in the dataset only if item has at least K ratings.

K_core_user (int): Include in the dataset only if user has done K ratings

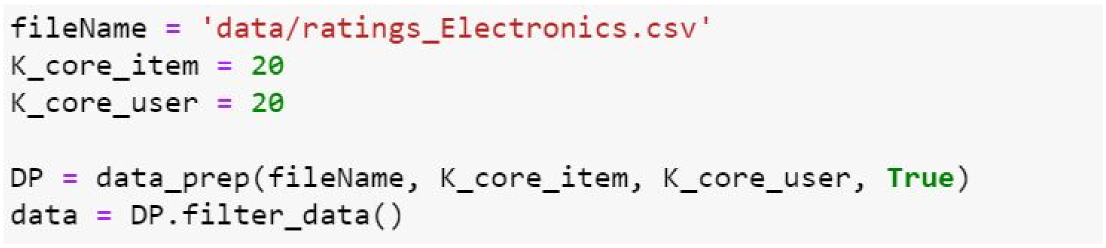
convert (Boolean): conversion of both reviewerID and asin(ProductID) into categorical.

filter_data()

return filtered dataset.

Example usage:

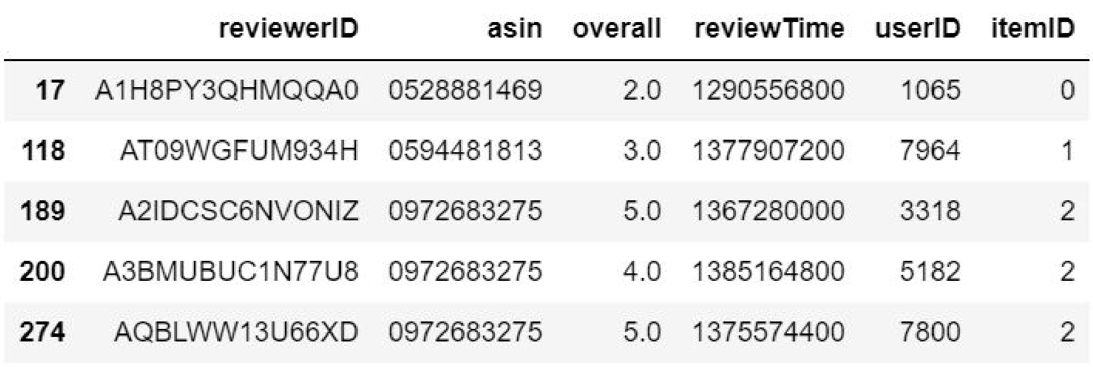
Call the data prep class and function to load electronics dataset.



```
fileName = 'data/ratings_Electronics.csv'
K_core_item = 20
K_core_user = 20

DP = data_prep(fileName, K_core_item, K_core_user, True)
data = DP.filter_data()
```

Figure 16. Example of data loader and preparation codes

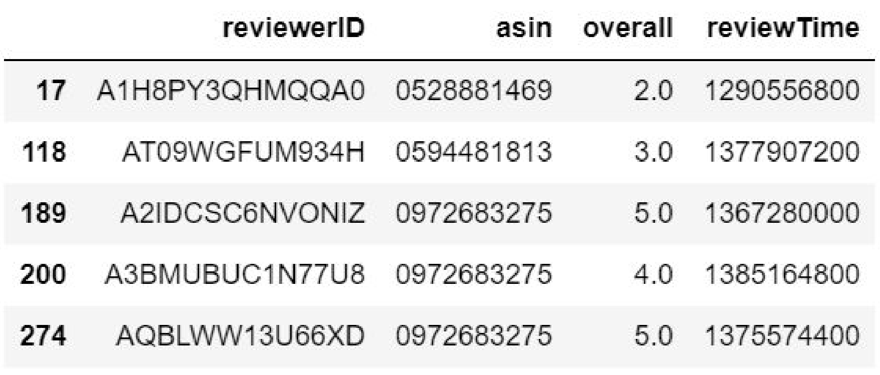
Dataset loaded when conversion flag set to **True**.



	reviewerID	asin	overall	reviewTime	userID	itemID
17	A1H8PY3QHMQQA0	0528881469	2.0	1290556800	1065	0
118	AT09WGFUM934H	0594481813	3.0	1377907200	7964	1
189	A2IDCSC6NVONIZ	0972683275	5.0	1367280000	3318	2
200	A3BMUBUC1N77U8	0972683275	4.0	1385164800	5182	2
274	AQBLWW13U66XD	0972683275	5.0	1375574400	7800	2

Figure 17. Result of data conversion flag = True

Dataset loaded when conversion flag set to **False**.



	reviewerID	asin	overall	reviewTime
17	A1H8PY3QHMQQA0	0528881469	2.0	1290556800
118	AT09WGFUM934H	0594481813	3.0	1377907200
189	A2IDCSC6NVONIZ	0972683275	5.0	1367280000
200	A3BMUBUC1N77U8	0972683275	4.0	1385164800
274	AQBLWW13U66XD	0972683275	5.0	1375574400

Figure 18. Result of data conversion flag = False

<u>Deep_SparseAutoEncoder class:</u>

Deep_SparseAutoEncoder(n_users, n_items, latent_factor, activation, reg_rate, pre_feed, layers)

<u>parameters:</u>

n_users (int): Total number of users in the rating table, for embedding usage.

n_items (int): Total number of items in the rating table, for embedding usage.

latent factor (int): rank/sparsity level, for embedding usage.

activation (string): Activation methods for each encoder/decoder layer.

reg_rate (float): L2 regularization strength.

pre_feed (string): what to do with the embedding layers, before feeding into autoencoders.

option :

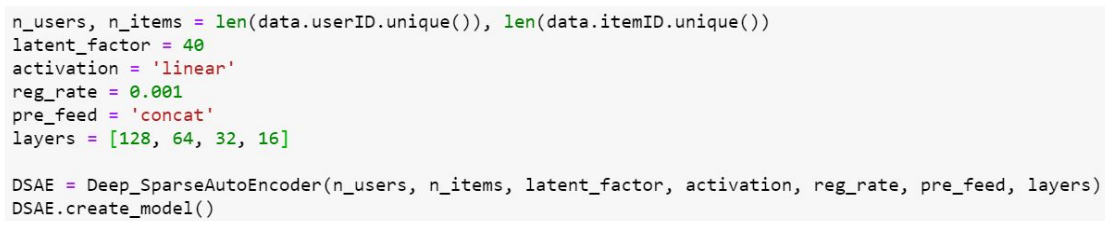
concat (concatenate of two embeddings into new features),

dot (dot product of two embeddings)

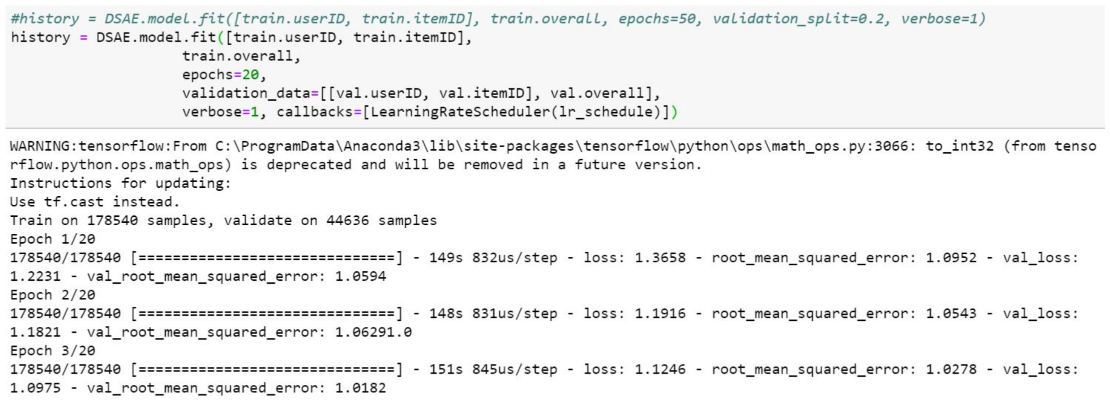
DSAE.create_model()

compile and create deep sparse autoencoder.

Example usage:



```
n_users, n_items = len(data.userID.unique()), len(data.itemID.unique())
latent_factor = 40
activation = 'linear'
reg_rate = 0.001
pre_feed = 'concat'
layers = [128, 64, 32, 16]

DSAE = Deep_SparseAutoEncoder(n_users, n_items, latent_factor, activation, reg_rate, pre_feed, layers)
DSAE.create_model()
```



```
#history = DSAE.model.fit([train.userID, train.itemID], train.overall, epochs=50, validation_split=0.2, verbose=1)
history = DSAE.model.fit([train.userID, train.itemID],
                         train.overall,
                         epochs=20,
                         validation_data=[[val.userID, val.itemID], val.overall],
                         verbose=1, callbacks=[LearningRateScheduler(lr_schedule)])
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tenso
rflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 178540 samples, validate on 44636 samples
Epoch 1/20
178540/178540 [==============================] - 149s 832us/step - loss: 1.3658 - root_mean_squared_error: 1.0952 - val_loss:
1.2231 - val_root_mean_squared_error: 1.0594
Epoch 2/20
178540/178540 [==============================] - 148s 831us/step - loss: 1.1916 - root_mean_squared_error: 1.0543 - val_loss:
1.1821 - val_root_mean_squared_error: 1.06291.0
Epoch 3/20
178540/178540 [==============================] - 151s 845us/step - loss: 1.1246 - root_mean_squared_error: 1.0278 - val_loss:
1.0975 - val_root_mean_squared_error: 1.0182
```

Figure 19. Example of model settings

<u>Utility functions:</u>

evaluateModel(model, data)

getting model prediction of the test data and return the MAE and RMSE results.

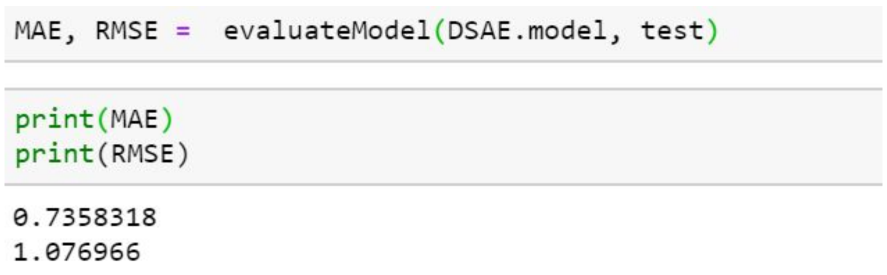
<u>parameters:</u>

model (object): The trained model of recommender system.

data (dataset): Data that is used for testing.

Return **MAE** and **RMSE** results.

Example usage:



```
MAE, RMSE =  evaluateModel(DSAE.model, test)
```

```
print(MAE)
print(RMSE)
```

```
0.7358318
1.076966
```

Figure 20. Example of the evaluation metrics function

getRecList(model, data, meta_data, user, top)

Get top-n item recommendation list for user.

getRecList(model, data, meta_data, user, top)

Get top-n item recommendation list for user.

<u>Parameters:</u>

model (Object): Trained model

data (dataset): The full rating dataset

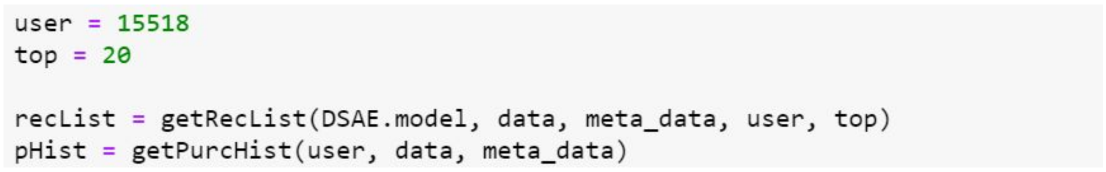
meta_data (dataset): Meta data of product, item title.

user (int): UserID/ReviewerID that will have recommended list generated.

top (int): Top-N highest rating items to be recommended to the user.

Returns a table of items with prediction score and product title, example can be seen in the discussion of top-n analysis.

Example usage:



```
user = 15518
top = 20

recList = getRecList(DSAE.model, data, meta_data, user, top)
pHist = getPurcHist(user, data, meta_data)
```

Figure 21. Example of the Top-N generator function

getPurcHist(user, data, meta_data)

Get all purchase made by the user.

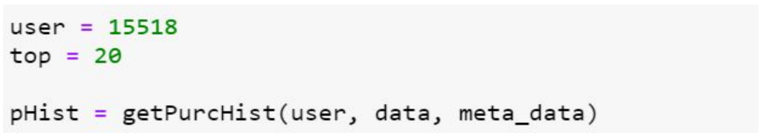
<u>Parameters:</u>

user (string): UserID that will be analyzed.

data (dataset): The ratings dataset

Returns the full review history that user has made, example of the purchase history can be seen in the previous section where we discussed on top-n.

Example usage:



```
user = 15518
top = 20

pHist = getPurcHist(user, data, meta_data)
```

Figure 21. Example of user history function

References

Aaron, V.D.O, Sander, D., Benjamin, S.(2013). Deep content-based music recommendation. Proceedings of the 26th Advances in Neural Information Processing Systems

Aditya, P., Budi, I., & Munajat, Q. (2016). A comparative analysis of memory-based and model-based collaborative filtering on the implementation of recommender system for E-commerce in Indonesia: A case study PT X. 2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS), 303–308. https://doi.org/10.1109/ICACSIS.2016.7872755

Aggarwal, C. (2016). Recommender Systems The Textbook . Cham: Springer International Publishing.

Amir, G., Amnon, M., Karl-Heinz, L., Lior, R., Alon, S., Arnon, S. (2010). A Decision Tree Based Recommender System. Informatica Intelligent Cloud Services.

Billsus, D., & Pazzani, M. (2000). User Modeling for Adaptive News Access. User Modeling and User-Adapted Interaction, 10(2), 147–180. https://doi.org/10.1023/A:1026501525781

Bokde, D., Girase, S., & Mukhopadhyay, D. (2015). An Approach to a University Recommendation by Multi-criteria Collaborative Filtering and Dimensionality Reduction Techniques. 2015 IEEE International Symposium on Nanoelectronic and Information Systems, 231–236. https://doi.org/10.1109/iNIS.2015.36

Bollacker, K., Lawrence, S., & Giles, C. (1998). CiteSeer: an autonomous Web agent for automatic retrieval and identification of interesting publications. Proceedings of the Second International Conference on Autonomous Agents, 116–123. https://doi.org/10.1145/280765.280786

Cai-Nicolas, Z. (2004). Book-Crossing Dataset. Retrieved 7 September 2019, from http://www2.informatik.uni-freiburg.de/~cziegler/BX/

Covington, P., Adams, J., & Sargin, E. (2016). Deep Neural Networks for YouTube Recommendations. Proceedings of the 10th ACM Conference on Recommender Systems, 191–198. https://doi.org/10.1145/2959100.2959190

Das, J., Majumder, S., & Gupta, P. (2013). Spatially aware recommendations using K-d trees. IET Conference Proceedings, 2013(646), 209–217. https://doi.org/10.1049/cp.2013.2593

David, M.B., Andrew, N., & Michael, I.J, (2003). Latent Dirichlet Allocation, Journal of Machine Learning Research 3, 993–1022.

Eleks. (2014.). Data Science in Action: Unlocking the Power of Recommender Systems. Retrieved 15 November 2019, from lab.eleks.com: https://labs.eleks.com/2014/10/data-science-in-action-unlocking-the-power-of-recommender-systems.html

George, T., & Merugu, S. (2005). A scalable collaborative filtering framework based on co-clustering. Fifth IEEE International Conference on Data Mining (ICDM'05), 4 pp. https://doi.org/10.1109/ICDM.2005.14

GroupLens. (2019). MovieLens 20M Dataset. Retrieved 1 September from Kaggle.com: https://www.kaggle.com/grouplens/movielens-20m-dataset

Gupta, P., Goel, A., Lin, J., Sharma, A., Wang, D., & Zadeh, R. (2013). WTF: the who to follow service at Twitter. Proceedings of the 22nd International Conference on World Wide Web, 505–514. https://doi.org/10.1145/2488388.2488433

Hofmann, T. (2004). Latent semantic models for collaborative filtering. ACM Transactions on Information Systems, 22(1), 89–115. https://doi.org/10.1145/963770.963774

Jain, P., Netrapalli, P., & Sanghavi, S. (2013). Low-rank matrix completion using alternating minimization. Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, 665–674. https://doi.org/10.1145/2488608.2488693

Joonseok, L., Mingxuan, S., & Guy, L. (2012). A Comparative Study of Collaborative Filtering Algorithms. ArXiv. doi: 10.5220/0004104001320137

Jun-Yan Zhu, Taesung Park, Isola, P., & Efros, A. (2017). Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. 2017 IEEE International Conference on Computer Vision (ICCV), 2017-, 2242–2251. https://doi.org/10.1109/ICCV.2017.244

Kaggle.(2018). Brazilian E-Commerce Public Dataset by Olist. Retrieved 7 September 2019, from https://www.kaggle.com/olistbr/brazilian-ecommerce/home#olist_order_reviews_dataset.csv

Krasinski, M. (2019). The Long Tail Effect theory in practise explained. Retrieved 15 November 2019, from miloszkrasinski.com: https://miloszkrasinski.com/the-long-tail-effect-theory-in-practise-explained/

Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 426–434. https://doi.org/10.1145/1401890.1401944

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. Computer, 42(8), 30–37. https://doi.org/10.1109/MC.2009.263

Kuchaiev, O., and Ginsburg, B. (2017). Training Deep AutoEncoders for Collaborative Filtering, ArXiv:1708.01715.

Li, X., & She, J. (2017). Collaborative Variational Autoencoder for Recommender Systems. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 129685, 305–314. https://doi.org/10.1145/3097983.3098077

McAuley, J. (2016). Amazon review data. Retrieved 7 September 2019, from http://jmcauley.ucsd.edu/data/amazon/links.html

Mooney, R., & Roy, L. (2000). Content-based book recommending using learning for text categorization. Proceedings of the Fifth ACM Conference on Digital Libraries, 195–204. https://doi.org/10.1145/336597.336662

Netflix.(2017).Netflix Prize Data. Retrieved 1 September 2019, from https://www.kaggle.com/netflix-inc/netflix-prize-data

Pazzani, M., & Billsus, D. (2007). Content-Based Recommendation Systems. In The Adaptive Web: Methods and Strategies of Web Personalization (Vol. 4321, pp. 325–341). https://doi.org/10.1007/978-3-540-72079-9_10

Retail Rocket.(2017). Retail Rocket Dataset. Retrieved 1 September 2019, from https://www.kaggle.com/retailrocket/ecommerce-dataset

Salakhutdinov, R., Mnih, A., & Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. Proceedings of the 24th International Conference on Machine Learning, 227, 791–798. https://doi.org/10.1145/1273496.1273596

Salakhutdinov, R., & Mnih, A. (2008). Probabilistic matrix factorization. In Advances in Neural Information Processing Systems, volume 20, 2008.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. Proceedings of the 10th International Conference on World Wide Web, 285–295. https://doi.org/10.1145/371920.372071

Sedhain, S., Menon, A., Sanner, S., & Xie, L. (2015). AutoRec: Autoencoders Meet Collaborative Filtering. Proceedings of the 24th International Conference on World Wide Web, 111–112. https://doi.org/10.1145/2740908.2742726

Su, X., & Khoshgoftaar, T. (2009). A Survey of Collaborative Filtering Techniques. Advances in Artificial Intelligence, 2009, 1–19. https://doi.org/10.1155/2009/421425

Suzuki, Y., & Ozaki, T. (2017). Stacked Denoising Autoencoder-Based Deep Collaborative Filtering Using the Change of Similarity. 2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA), 498–502. https://doi.org/10.1109/WAINA.2017.72

Tran, D.H., Hussain, Z., Zhang, W.E., Khoa, N.L., Tran, N.H., & Sheng, Q.Z. (2019). Deep Autoencoder for Recommender Systems: Parameter Influence Analysis. *ArXiv, abs/1901.00415*.

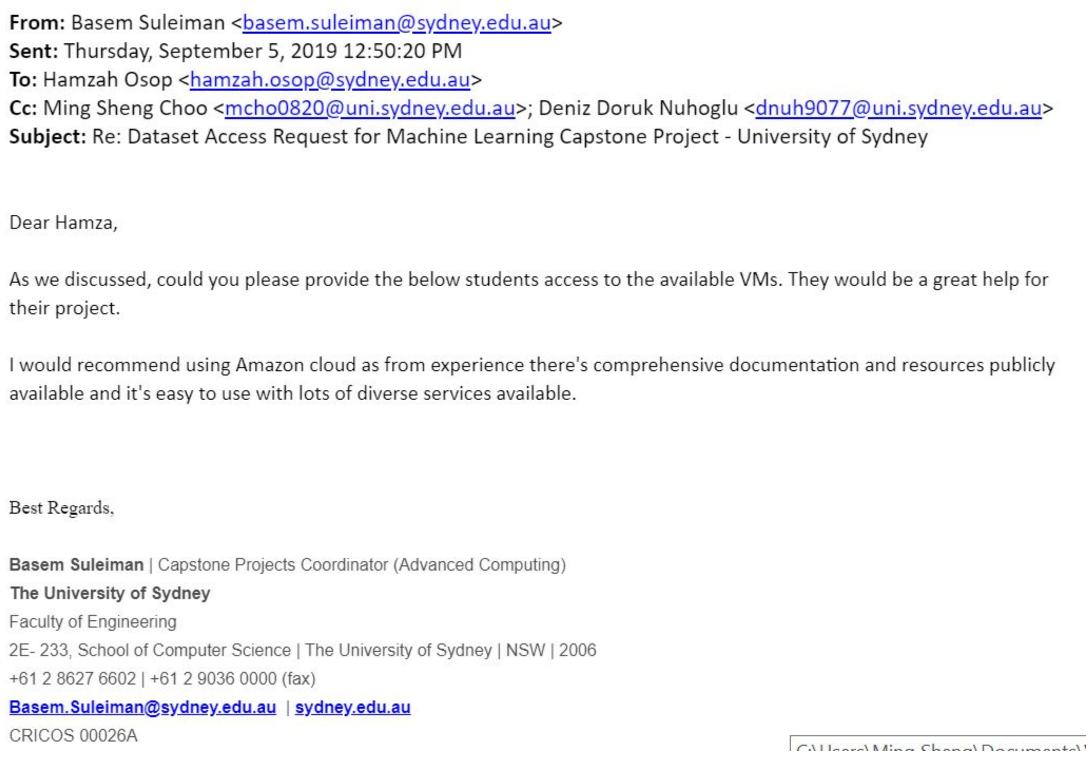
Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. (2017). Neural collaborative filtering. In Proceedings of the 26th International

Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 173–182.

Ye, B., Tu, Y., & Liang, T. (2019). A HYBRID SYSTEM FOR PERSONALIZED CONTENT RECOMMENDATION. Journal of Electronic Commerce Research, 20(2), 91–104. Retrieved from http://search.proquest.com/docview/2226386463/

Appendix

Appendix A. Cloud virtual machine discussion.



From: Basem Suleiman <basem.suleiman@sydney.edu.au>
Sent: Thursday, September 5, 2019 12:50:20 PM
To: Hamzah Osop <hamzah.osop@sydney.edu.au>
Cc: Ming Sheng Choo <mcho0820@uni.sydney.edu.au>; Deniz Doruk Nuhoglu <dnuh9077@uni.sydney.edu.au>
Subject: Re: Dataset Access Request for Machine Learning Capstone Project - University of Sydney

Dear Hamza,

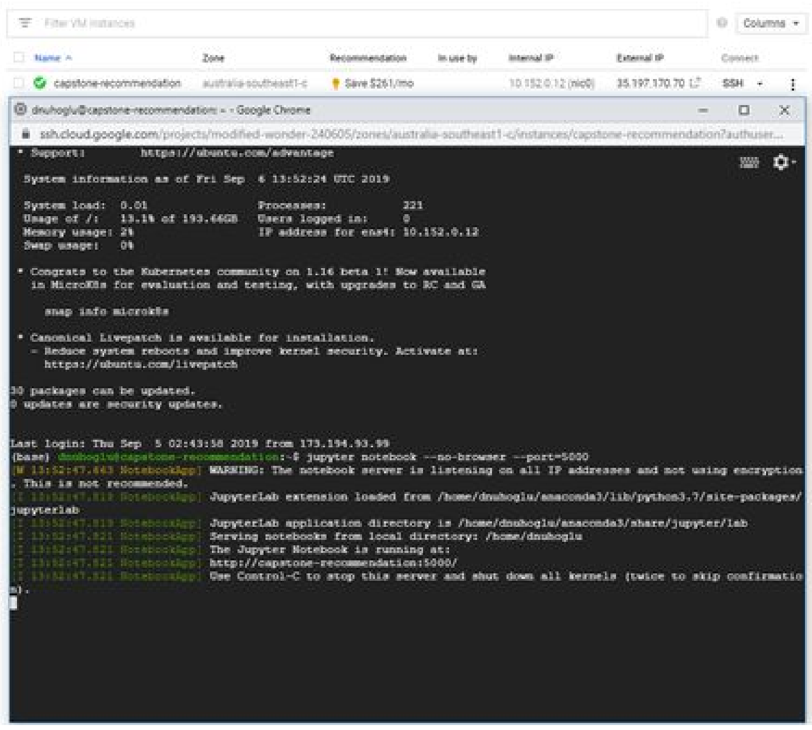
As we discussed, could you please provide the below students access to the available VMs. They would be a great help for their project.

I would recommend using Amazon cloud as from experience there's comprehensive documentation and resources publicly available and it's easy to use with lots of diverse services available.

Best Regards,

Basem Suleiman | Capstone Projects Coordinator (Advanced Computing)
The University of Sydney
Faculty of Engineering
2E- 233, School of Computer Science | The University of Sydney | NSW | 2006
+61 2 8627 6602 | +61 2 9036 0000 (fax)
Basem.Suleiman@sydney.edu.au | sydney.edu.au
CRICOS 00026A

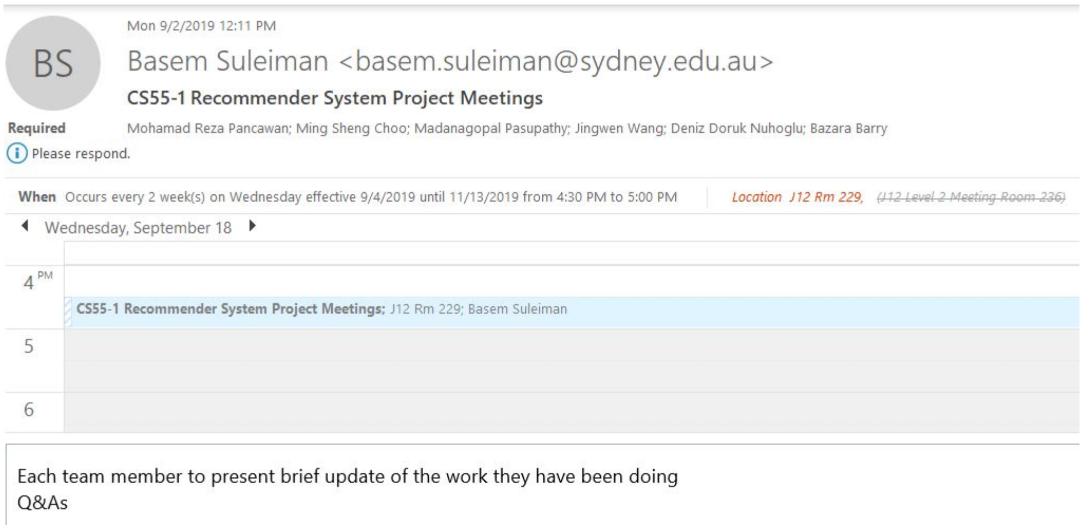
Appendix B. Establish our own virtual machine (GCP)


Filter VM instances
Columns
dnuhoglu@capstone-recommendation: ~ - Google Chrome
ssh.cloud.google.com/projects/modified-wonder-240605/zones/australia-southeast1-c/instances/capstone-recommendation?authuser...

Name	Zone	Recommendation	In use by	Internal IP	External IP	Connect
capstone-recommendation	australia-southeast1-c	Save $261/mo		10.152.0.12 (nic0)	35.197.170.70	SSH

```
 * Support:        https://ubuntu.com/advantage

  System information as of Fri Sep  6 13:52:24 UTC 2019

  System load:  0.01               Processes:           221
  Usage of /:   13.1% of 193.66GB   Users logged in:     0
  Memory usage: 2%                 IP address for ens4: 10.152.0.12
  Swap usage:   0%

 * Congrats to the Kubernetes community on 1.16 beta 1! Now available
   in MicroK8s for evaluation and testing, with upgrades to RC and GA

     snap info microk8s

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

30 packages can be updated.
0 updates are security updates.


Last login: Thu Sep  5 02:43:58 2019 from 173.194.93.99
(base) dnuhoglu@capstone-recommendation:~$ jupyter notebook --no-browser --port=5000
[W 13:52:47.463 NotebookApp] WARNING: The notebook server is listening on all IP addresses and not using encryption
. This is not recommended.
[I 13:52:47.819 NotebookApp] JupyterLab extension loaded from /home/dnuhoglu/anaconda3/lib/python3.7/site-packages/
jupyterlab
[I 13:52:47.819 NotebookApp] JupyterLab application directory is /home/dnuhoglu/anaconda3/share/jupyter/lab
[I 13:52:47.821 NotebookApp] Serving notebooks from local directory: /home/dnuhoglu
[I 13:52:47.821 NotebookApp] The Jupyter Notebook is running at:
[I 13:52:47.821 NotebookApp] http://capstone-recommendation:5000/
[I 13:52:47.821 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmatio
n).
```

Appendix C. Progress meeting with project owners/clients



Mon 9/2/2019 12:11 PM

BS Basem Suleiman <basem.suleiman@sydney.edu.au>

CS55-1 Recommender System Project Meetings

Required Mohamad Reza Pancawan; Ming Sheng Choo; Madanagopal Pasupathy; Jingwen Wang; Deniz Doruk Nuhoglu; Bazara Barry

Please respond.

When Occurs every 2 week(s) on Wednesday effective 9/4/2019 until 11/13/2019 from 4:30 PM to 5:00 PM Location J12 Rm 229, ~~(J12 Level 2 Meeting Room 236)~~

Wednesday, September 18

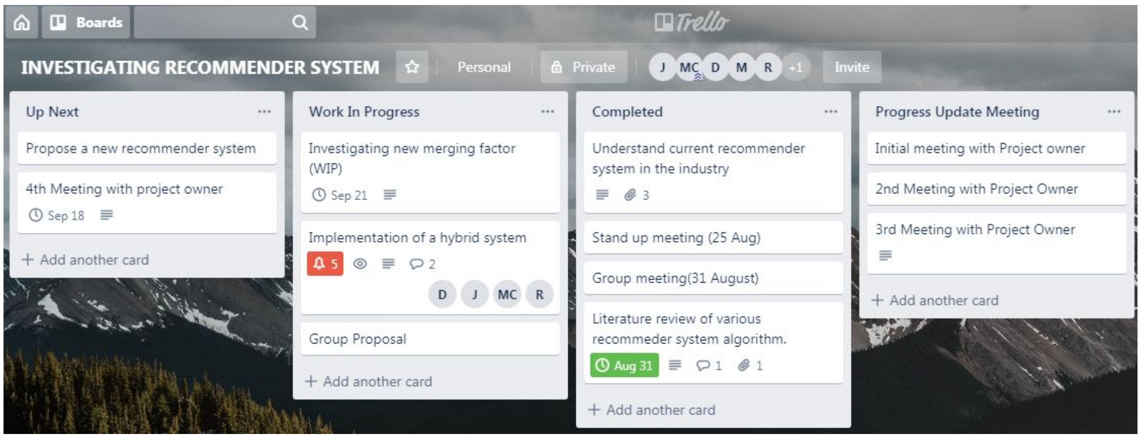
4 PM CS55-1 Recommender System Project Meetings; J12 Rm 229; Basem Suleiman

5

6

Each team member to present brief update of the work they have been doing
Q&As

Appendix D. Progress management using Trello.



Boards

Trello

INVESTIGATING RECOMMENDER SYSTEM

Personal · Private · J MC D M R +1 · Invite

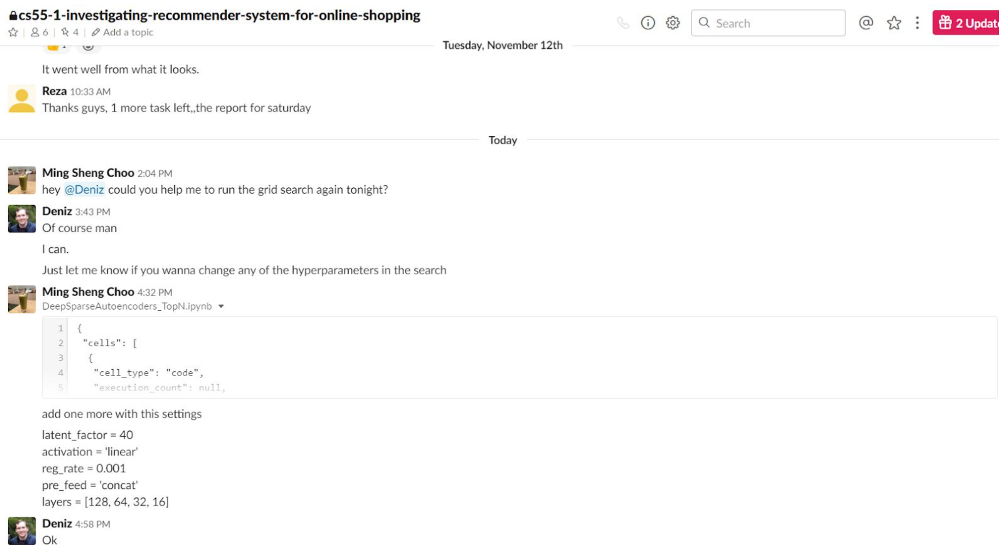
Up Next
- Propose a new recommender system
- 4th Meeting with project owner — Sep 18
- + Add another card

Work In Progress
- Investigating new merging factor (WIP) — Sep 21
- Implementation of a hybrid system — 5 · 2 — D J MC R
- Group Proposal
- + Add another card

Completed
- Understand current recommender system in the industry — 3
- Stand up meeting (25 Aug)
- Group meeting(31 August)
- Literature review of various recommeder system algorithm. — Aug 31 · 1 · 1
- + Add another card

Progress Update Meeting
- Initial meeting with Project owner
- 2nd Meeting with Project Owner
- 3rd Meeting with Project Owner
- + Add another card

Appendix E. Online group collaboration on Slack



cs55-1-investigating-recommender-system-for-online-shopping
6 | 4 | Add a topic

Search · 2 Updat

Tuesday, November 12th

It went well from what it looks.

Reza 10:33 AM
Thanks guys, 1 more task left,,the report for saturday

Today

Ming Sheng Choo 2:04 PM
hey @Deniz could you help me to run the grid search again tonight?

Deniz 3:43 PM
Of course man

I can.

Just let me know if you wanna change any of the hyperparameters in the search

Ming Sheng Choo 4:32 PM
DeepSparseAutoencoders_TopN.ipynb

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": null,
```

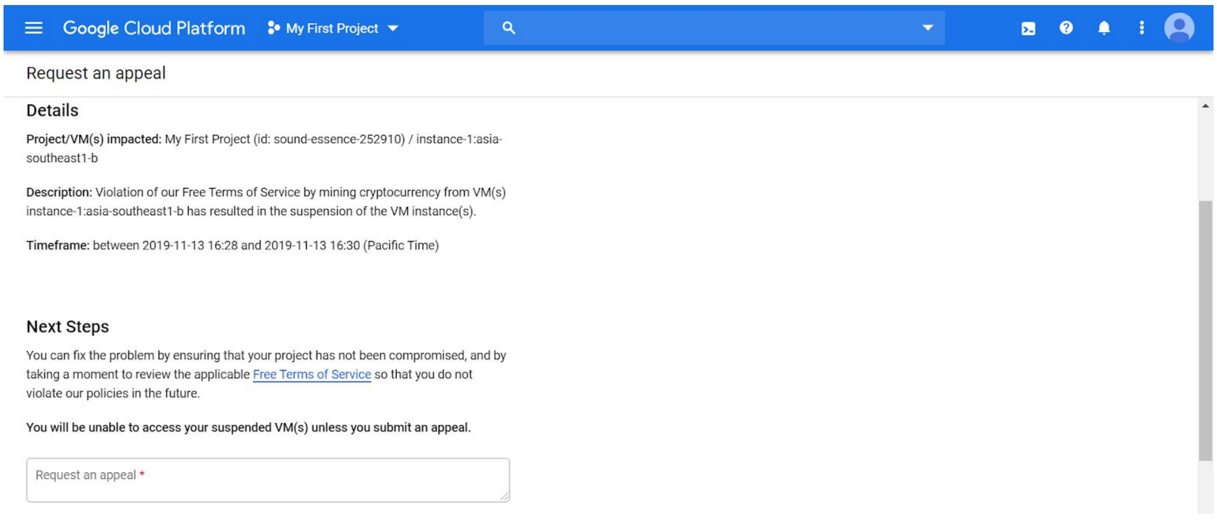
add one more with this settings

latent_factor = 40
activation = 'linear'
reg_rate = 0.001
pre_feed = 'concat'
layers = [128, 64, 32, 16]

Deniz 4:58 PM
Ok

Appendix F. Others

- **Challenge of using cloud**



Request an appeal

Details

Project/VM(s) impacted: My First Project (id: sound-essence-252910) / instance-1:asia-southeast1-b

Description: Violation of our Free Terms of Service by mining cryptocurrency from VM(s) instance-1:asia-southeast1-b has resulted in the suspension of the VM instance(s).

Timeframe: between 2019-11-13 16:28 and 2019-11-13 16:30 (Pacific Time)

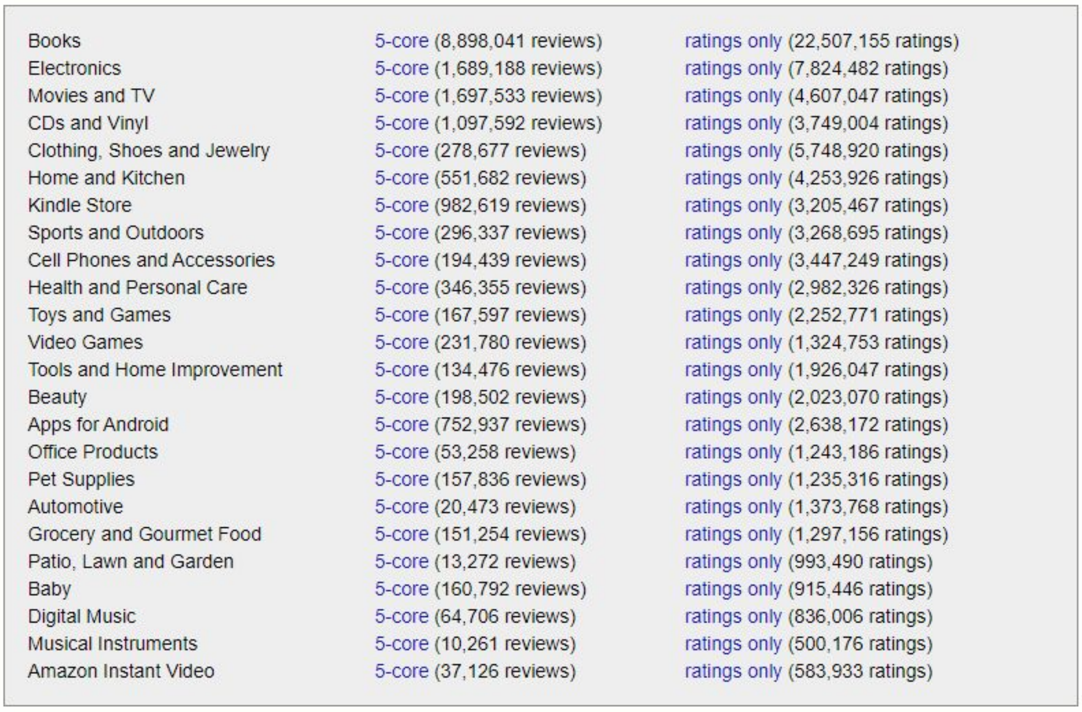
Next Steps

You can fix the problem by ensuring that your project has not been compromised, and by taking a moment to review the applicable Free Terms of Service so that you do not violate our policies in the future.

You will be unable to access your suspended VM(s) unless you submit an appeal.

Request an appeal *

- **24 Categories of Amazon dataset**



Books	5-core (8,898,041 reviews)	ratings only (22,507,155 ratings)
Electronics	5-core (1,689,188 reviews)	ratings only (7,824,482 ratings)
Movies and TV	5-core (1,697,533 reviews)	ratings only (4,607,047 ratings)
CDs and Vinyl	5-core (1,097,592 reviews)	ratings only (3,749,004 ratings)
Clothing, Shoes and Jewelry	5-core (278,677 reviews)	ratings only (5,748,920 ratings)
Home and Kitchen	5-core (551,682 reviews)	ratings only (4,253,926 ratings)
Kindle Store	5-core (982,619 reviews)	ratings only (3,205,467 ratings)
Sports and Outdoors	5-core (296,337 reviews)	ratings only (3,268,695 ratings)
Cell Phones and Accessories	5-core (194,439 reviews)	ratings only (3,447,249 ratings)
Health and Personal Care	5-core (346,355 reviews)	ratings only (2,982,326 ratings)
Toys and Games	5-core (167,597 reviews)	ratings only (2,252,771 ratings)
Video Games	5-core (231,780 reviews)	ratings only (1,324,753 ratings)
Tools and Home Improvement	5-core (134,476 reviews)	ratings only (1,926,047 ratings)
Beauty	5-core (198,502 reviews)	ratings only (2,023,070 ratings)
Apps for Android	5-core (752,937 reviews)	ratings only (2,638,172 ratings)
Office Products	5-core (53,258 reviews)	ratings only (1,243,186 ratings)
Pet Supplies	5-core (157,836 reviews)	ratings only (1,235,316 ratings)
Automotive	5-core (20,473 reviews)	ratings only (1,373,768 ratings)
Grocery and Gourmet Food	5-core (151,254 reviews)	ratings only (1,297,156 ratings)
Patio, Lawn and Garden	5-core (13,272 reviews)	ratings only (993,490 ratings)
Baby	5-core (160,792 reviews)	ratings only (915,446 ratings)
Digital Music	5-core (64,706 reviews)	ratings only (836,006 ratings)
Musical Instruments	5-core (10,261 reviews)	ratings only (500,176 ratings)
Amazon Instant Video	5-core (37,126 reviews)	ratings only (583,933 ratings)