Spotify Playlist Music Recommendation System

Hongxin Fu
Candidate Number
52240

Ye Hu
Candidate Number
55974

Tiam Tee Ng
Candidate Number
48119

Abstract

In this project, we developed a music recommendation system by using the Neural Collaborating Filtering (NCF) model and Dimension Independent Matrix Square with MapReduce (DIMSUM) algorithm within a distributed computing framework. The NCF model replaced the state-of-the-art method in matrix factorization models, which applies the inner product to the latent features of users and items, with a neural network that extracts underlying concealed patterns from the data. On the other hand, the DIMSUM algorithm defines an efficient method for calculating the similarity between items using a MapReduce framework and a large and sparse user-item matrix.

1 Introduction

With the rise of digital content distribution and recent advancements in streaming technologies, people now can access music collections at an unprecedented scale [7]. As of April 2023, Spotify alone boasts a staggering collection of 100 million tracks on its platform [8], let alone all the music streaming services combined. With that many music items available on the Internet, the task of finding acceptable and preferred music has become a challenging task. Music recommendation systems help to solve this problem. They help users to filter from the many items available and discover songs of their choice. Although there have been a lot of significant advancements in data science, building a model that is capable enough to give good recommendations and efficient enough to produce results within milliseconds online remains a highly challenging task.

The primary reason for this is the scalability potential of a recommender engine to handle large-scale datasets as the data grows at an exponential scale. As the data volumes continue to increase at an unprecedented rate, traditional computing methods often struggle to process and analyse the information in a timely manner. To address the challenge, distributed computing has emerged as a promising approach, leveraging the power of multiple computing nodes to process and analyse large dataset in parallel. In this report, we will study an efficient algorithm called Dimension Independent Matrix Square Using MapReduce (DIMSUM) [10] which we believe is a ideal algorithm to present the impact on distributed framework to process large scale data. In brief, the main problem that the algorithm tries to tackle is that to compute the similarity scores between items in an efficient and scalable manner. The similarity score can be predefined according to our interest, and in our implementation, we will only consider cosine similarity. The algorithm assumes a sparse user-item matrix A with size m by n where m is much larger than n. For example, m = 10^{13} and n = 10^3. Although the shape of our input data to build the recommendation system does not align with this assumption as we are unable to make full use of the entire data set, we would like to argue that this is a reasonable and practical assumption. We will also present the results from our experimentation of manipulating the shape of the input data to fit the model's requirement.

Due to the Netflix prize, the Matrix factorization technique is also a widely used tool in machine learning for recommendation systems [3]. Such methods can be applied to analyse a wide range of data such as data like user-item ratings and document-word frequencies to discover underlying hidden patterns. However, such technique suffers from the item cold-start problem. The cold-start problem occurs when a new user enter the model without any past interactions. In the context of music recommendation, one can take it as asking the model to make recommendations given a playlist with zero tracks. Although this is not the problem we want to study in this report but it has led to an increasing attention on applying neural architecture to build recommendation systems. In addition, despite its great success and simplicity, prior efforts also suggest that such a method would have two major limitations. The first one is that the triangle inequality is violated [5], as the inner product only encourages the representations of users and historical items to be similar, but lacks guarantees for the similarity propagation between user-user relationships. Also, it models the linear interaction and may fail to capture the complex relationships between users and items [3]. However, it is also worth noting that a lot of the existing recommendation engines are still relying on the computation of the inner product between the user embedding and the item embedding[9]. Hence, in our project, we will delve into the Neural collaborative model [3]. Not only does the model extend the traditional Matrix Factorisation framework, it also aims to capture the intricate, non-linear relationships between use-item interactions.

2 Dataset Description

We will build our recommendation engines based on the Spotify Million Playlist Dataset [1] throughout this project. The data set consists of 1 million playlists with a total size of 32GB. The data set contains information about all playlists, including playlist id, number of albums, number of tracks, list of tracks in the playlist, etc. The data set also consists of information about each track in the playlist including track name, track id, and so on. Of all the information about the playlists and tracks, we are only interested in the playlist id and the track id. In essence, we filtered all features except the playlist id and the list of track ids corresponding to the particular playlist.

2.1 Preprocess

The original dataset is saved in the json format. Since we are working with Pyspark, we converted the data into Spark RDD format and organised everything into a spark dataframe. This will allow us to perform data analysis in a distributed enviroment. During our project, we worked if 5% of the entire dataset which is 50000 playlists and 1.6GB in size.

2.2 Exploratory Data Analysis

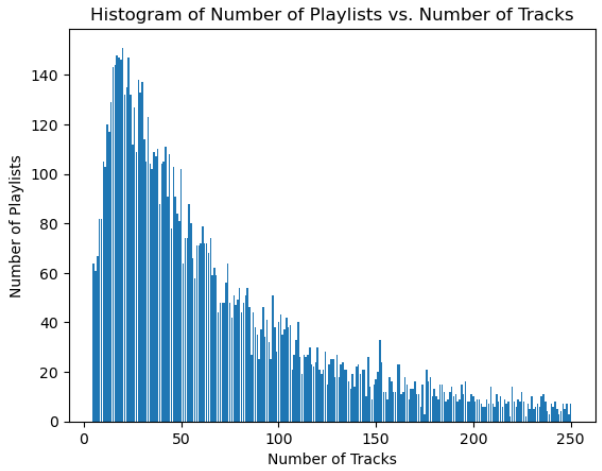


Figure 1

Processing with 50,000 playlists data, we observe that fewer than 50% of playlists contain 100 tracks or more, while playlists with 20 to 50 tracks make up a significant portion of the data. In addition, the number of playlists is going to peak at around 140 when they only have around 25 songs in each, according to Figure 1. It also shows that as the increase in the number of tracks, the number of playlists starts to decrease after its peak.

Furthermore, from Figure 2, we observe that the common track counts over 100 times in all playlists have a very skewed distribution. Most tracks are counted less than 500 times, and tracks in each playlist are not overlapped too much. Additionally, there is a significant decrease in track counts over 100 times but under 200 times and over 200 times but under 300 times, which infers that it is difficult for good music songs to remain popular over time.

According to Table 1, the Top 10 Tracks, comprises primarily of Hip-hop and Pop music genres, with all artists being either rappers or pop vocalists.

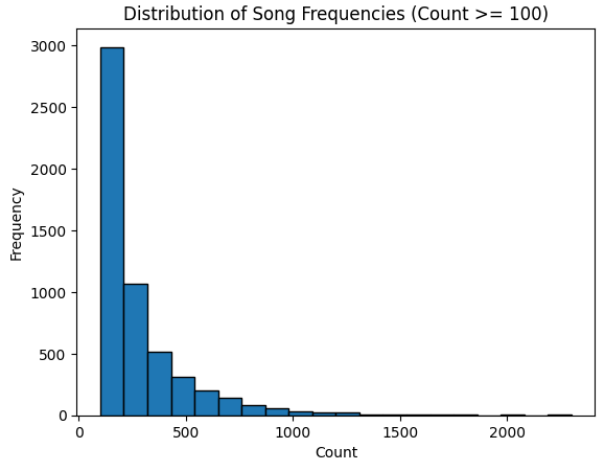


Figure 2

Top 10 Tracks

track_name	album_name	artist_name	count
HUMBLE.	DAMN.	Kendrick Lamar	2301
One Dance	Views	Drake	2208
Broccoli (feat. Lil Yachty)	Big Baby DRAM	DRAM	2106
Closer	Closer	The Chainsmokers	2053
Congratulations	Stoney	Post Malone	2024
Caroline	Good For You	Aminé	1815
iSpy (feat. Lil Yachty)	iSpy (feat. Lil Yachty)	KYLE	1778
Location	American Teen	Khalid	1763
Bad and Boujee (feat. Lil Uzi Vert)	Culture	Migos	1736
Bounce Back	I Decided.	Big Sean	1714

Table 1

In corresponding to Top 10 Tracks, from Figure 3, The majority of the Top 10 Artists are rappers, such as Drake, Kanye West, Kendrick Lamar, etc, while others are pop vocalists.

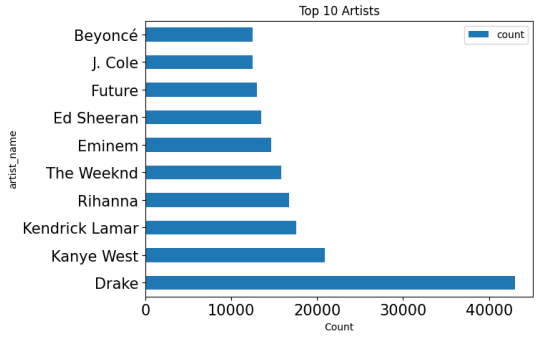


Figure 3

From the Figure 3, though the plot only shows the top 10 artists, it is not hard to imagine that the data distribution is skewed in the sense that famous artists and popular songs have much higher probability to be included into user's playlist. In such a case, one of the most naive implementation of a music recommender is to recommend everyone the top 10 or top 100 tracks that receive the most clicks (implicit feedbacks) from the users. Though this might yield a decent results in terms of the performance of the recommendation engine but this is not what we strive for. We would like a more personalised recommendation that we believe is better suit for users.

3 Methodology

Regarding the variety that people have individual special music tastes since not everyone will like the most popular songs, we decide to analyze the dataset with Dimension Independent Matrix Square using the MapReduce algorithm and Neural Collaborative Filtering Model to learn to develop a more comprehensive music recommender.

3.1 Dimension Independent Matrix Square using MapReduce

The Dimension Independent Matrix Square Using MapReduce (DIMSUM) is an algorithm that aims to compute the similarity score to understand the relationship between items based on implicit feedback data. In essence, given a sparse user-item matrix, this model

aims to find all similar item vector pairs according to a similarity function according to some similarity threshold. The similarity function we are using in our experiment is cosine similarity. We will formally define the preliminaries as follows: denote a matrix $\boldsymbol{A}$ of size m × n whose columns as $c_1, \ldots c_n$ and rows as $r_1, \ldots r_m$. (1)

$$\cos\left(c_i, c_j\right) = \frac{c_i^T c_j}{\|c_i\|_2 \left\|c_j\right\|_2} \tag{1}$$

Formula (1) represents the cosine similarity formula.

Due to the fact that we are finding the similarity score between all pairs of items, computing $\boldsymbol{A}^T\boldsymbol{A}$ for large $\boldsymbol{A}$ is a process that we cannot skip and it is also the component which makes the whole cosine similarity computation extremely time-consuming. A naive approach to compute $\boldsymbol{A}^T\boldsymbol{A}$ will be to run the element-wise multiplication between every value from $\boldsymbol{A}^T$ and $\boldsymbol{A}$. This approach will result in a complexity of $O(n^2)$. It is a well-known fact that it is not possible for one to perform matrix multiplication that is more efficient than the naive approach without any margin of error in terms of time complexity. Hence, with the heuristics regarding the sparsity of the matrix, the paper proposed a simple non-adaptive sampling scheme in the process of multiplication with some margin of error to improve the computational efficiency. Since the randomized nature of the algorithm, it is hard to provide an exact complexity formulation but it can be broadly characterized as having a sub-quadratic complexity which is considered as more efficient than $O(n^2)$

To compute the similarity score between tracks given the user-item matrix, in our implementation, we turn the dense matrix into a sparse matrix representation. Recall that our user-item matrix is made of binary entries and there are more zeros than ones. Hence, to make the computation more efficient, we only store the coordinate value of the entries whose value is 1. Keep in mind that we denote the row id as playlist and the column as track and we may use them interchangeably.

On a high level, the algorithm performs the computation using MapRedcue paradigm which is by splitting the computation into mapper and reduce, the mapper is as shown in the below:

Algorithm 1 Lean DIMSUM Mapper

for all a_{ij} in r_i
  With probability $\min(1, \frac{\sqrt{\gamma}}{\|c_j\|})$
  for all a_{ik} in r_i
    With probability $\min(1, \frac{\sqrt{\gamma}}{\|c_k\|})$
    Emit $b_{jk} \rightarrow \frac{a_{ij}a_{ik}}{\min(\sqrt{\gamma}, \|c_j\|)\min(\sqrt{\gamma}, \|c_k\|)}$
end for

Source: Adapted from Zadeh et al. (2014) [10]

According to the author, we should define the parameter γ as follows:

$$\gamma = \frac{4log(n)}{s} \tag{2}$$

Source: Adapted from Lin et al. (2014) [6]

The n is the number of rows of the input matrix and it is used to normalise the gamma value based on the matrix size and the s is the similarity threshold to be set manually which filters out less significant pairs of items in terms of similarity scores. The larger the threshold would lead to a smaller value of gamma which controls the sampling probability of matrix entries in the mapper function.

The reducer is as follows:

Algorithm 2 DIMSUM Reducer

1. output $c_i^T c_j \rightarrow \sum_{i=1}^{n} \frac{a_{ij}a_{ik}}{\min(\sqrt{\gamma}, \|c_j\|)\min(\sqrt{\gamma}, \|c_k\|)}$

Source: Adapted from Zadeh et al. (2014)[10]

Since we implement the algorithm on Pyspark which is a distributed framework that utilizes the MapReduce paradigm as well, hence it is important to dive into understanding the low-level operation of the computation. When we load the data in the format of Spark Dataframe into Spark, it will automatically partition the data set across the worker nodes in the cluster. The partitioning is done on the basis of the Resilient Distributed Dataset (RDD) data structure which is an immutable distributed collection of objects. Since our computation involves a mapper and reducer which is not the same as the mapper and reducer in low-level Spark, the mapper in Spark will distribute the computation that was being defined in the function's mapper to each partition of the data set in each worker node. Upon completion of computation in all worker nodes, the reducer in low-level Pyspark will aggregate the results back from all worker nodes to the master node. Spark handles the aggregation process by shuffling the data between the worker nodes as needed and combining the results. The same process will be applied to the function's reducer. We will also discuss the effect of the size of the cluster on the computation speed in the experiment section.

The details of the algorithmic implementation of Algorithm 1 and 2 are provided in the Python notebook.

3.2 Neural Collaborative Filtering Model

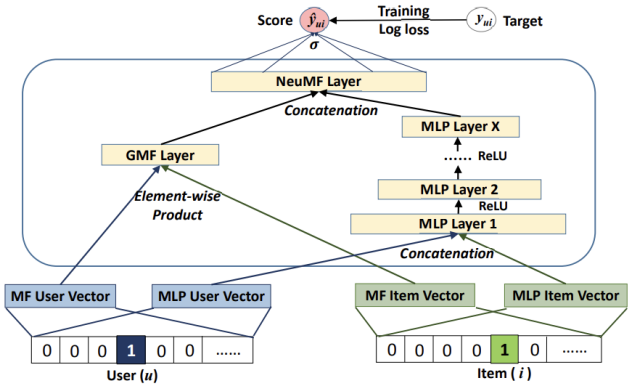


Image Source: He et al. (2017) [3]

Figure 4: Neural matrix factorization model.

The Neural Collaborative Filtering (NCF) model is a deep learning based approach for collaborative filtering in recommendation systems. It is made of two parts the Generalized Matrix Factorisation (GMF) and the Multi-Layer Perceptron (MLP). It aims to

address the limitations of traditional matrix factorization methods by leveraging the power of neural architecture to learn the complex and nonlinear relationship between the user and item-based on implicit feedback data.

The GMF block which is the left block of the neural network as shown in Figure 6 above. This block aims to generalise matrix factorisation techniques. It maps users and items into a shared latent space representing their preferences and characteristics. To delve into the GMF block, we will first examine the main components of the block, the embedding layers, the flattening layers, and the multiplication layer. The block first takes in a user vector and an item vector and passes them into two separate embedding layers to embed the vectors into the predefined number of dimensions which can be thought of as a latent dimension which is a hyper-parameter to be tuned and it determines the expressiveness of the latent space. One can also intuitively imagine that the number of latent dimensions is the number of neurons within the layer and it justifies how wide the layer is. The outputs from both embedding layers are then flattened and passed into a multiplication layer. The multiplication layer is to perform element-wise multiplication of both the user latent vector and the item latent vector. Let the latent user vector be P_u and the latent item vector is Q_v. The whole mapping process as mentioned can be just thought of as applying a mapping function ϕ to the latent vectors as follows:

$$\phi(P_u, Q_v) = P_u \cdot Q_v \tag{3}$$

Source: Adapted from He et al. (2017)[3]

To understand why it can generalise the matrix factorisation framework, we can apply an activation function to the product from the mapping function. Intuitively, if the function $f_activation$ is the uniform vector of one, we can recover the matrix factorisation model. The formulation is as follows:

$$Y = f_{activation}(W^T(P_u \cdot Q_v)) \tag{4}$$

Source: Adapted from He et al. (2017)[3]

The MLP layer which is the right block in Figure 4, is basically just an extension of the GMF layer. Recall from equation 4 above, here we extend the concept of applying a non-linear activation function to the product from the mapping function and pass the output through multiple hidden layers to capture the complex underlying patterns. The common non-linear activation functions are ReLU and tanH and the number of hidden layers and the number of units of neurons are hyperparameters to tune as well.

We then concatenate the output from the two blocks and pass the product to the final output layer which consists of a softmax activation function to output a vector of probability corresponding to all inputted items and user pairs.

4 Experiments

In our experiment, we applied the models to different technologies. The DIMSUM algorithm was implemented in Google Cloud Console Dataproc API with a master node (n1-standard-16) and 2 worker nodes (n1-standard-8) for our standard computation. We also compare the performance with different numbers and different types of worker nodes. On the other hand, for our NCF model, we did our computation in GCP Vertex AI workbench API with a single Nvidia T4 GPU. Due to the limitation on GPU quota on GCP, we are only able to perform our computation with one GPU but it is possible to perform model parallelization on GCP given our implementation by using build-in tensorflow API tf.distribute.MirrorStrategy.

4.1 Neural Collaborative Filtering Model

Here, we aim to deliver a comprehensive understanding of the model's effectiveness. Recall that our model consists of multiple hyper-parameters that need to be tuned. Hence, we will try to delve into each of them in this section. In our experiment, we first filtered out playlists consisting of less than 20 songs. It is theoretically feasible to include a playlist with few tracks according to the paper as the model is robust against the cold start problem, but in our analysis, we are only interested in analysing the ability of the model in predicting the majority of the playlists and are not interested in studying the performance of the model upon edge cases.

After filtering, we split the data set into a train set (80%) and a test set (20%). Notice that we have not had a data label yet. Hence, we have to manually generate negative data for each playlist which is to randomly sample tracks from the set consisting of all tracks that exist in the entire data set (train and test) but not in the particular playlist. This is to ensure that our data is balanced in the sense that we have both playlist and track paired with label 1 and label 0. To do this, we first create another column in the data frame and use apply function in Python Pandas API to generate negative data for every playlist. We then separate them into two data frames and explode every list in the track list column. Now, we will have a data frame with playlist and track, hence we manually assign a third column which stands for label, and make the label 1 if the exploded column represents a positive track list and 0 otherwise.

Track Count per Playlist in train set

Statistics	Value
count	33041
mean	77.999395
std	52.692228
min	21
25%	37
50%	61
75%	104
max	250

Table 2

Track Count per Playlist in test set

Statistics	Value
count	8261
mean	78.879797
std	53.31419
min	21
25%	38
50%	63
75%	104
max	250

Table 3

From the table 2 and 3 above, we can see that in the train set, the mean of the number of tracks in all playlists is roughly 78 and the median is 61. In our implementation, we used the median value to justify how many zero-labeled tracks to generate and add to each playlist. For the test set, we did not generate negative tracks because we only need to measure how well the model predict a relevant recommendation, hence we only have to count the number of 1s in the prediction and compare it against the actual results.

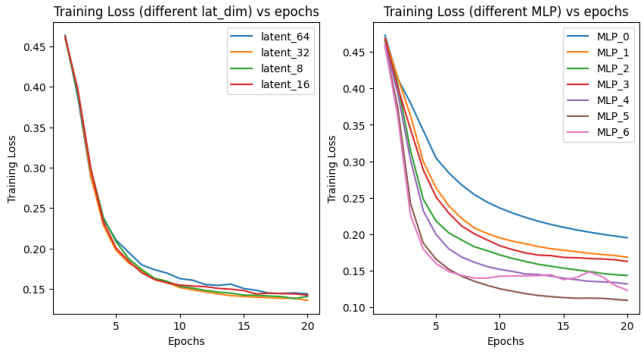
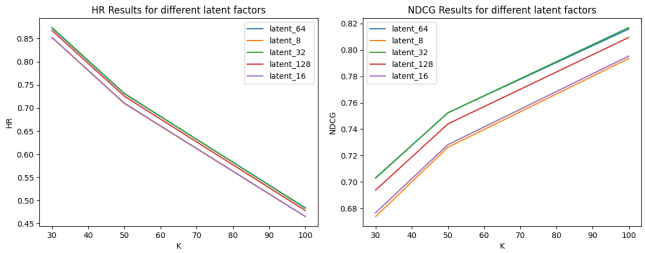


Figure 5

The training loss plots showcase the model's convergence during the training process. By examining this plot, we observe how the training losses converge across the different values of latent dimensions in the GMF block and different numbers of latent dimensions as well as layers in the MLP layers. From the left plot of Figure 7, we observe that the loss of each value of latent dimension shows roughly the same convergence value at approximately 0.15. But as we increase the number of latent dimensions and the number of layers in the MLP block, the model exhibits very low training loss. This might result in over-fitting in the test set. The values of hyper-parameters in different MLP architectures are shown in the table in Appendix A. For example, in MLP_6 we used [256, 128, 64, 32]. The length of the list justifies how many layers the MLP block has and the value of each element in the list justifies the number of neurons within the layer. The design of a wider-to-narrower architecture on MLP block has been a trend in deep learning literature. For example, the idea of learning hierarchical feature representations in deep networks in [2] and the encoder layer design used in [4] has inspired researchers to adopt such designs. This is to enable the model to learn more complex, high-order features and high-order feature interactions between users and items.



Data Source: Appendix A - NFC RESULTS

Figure 6

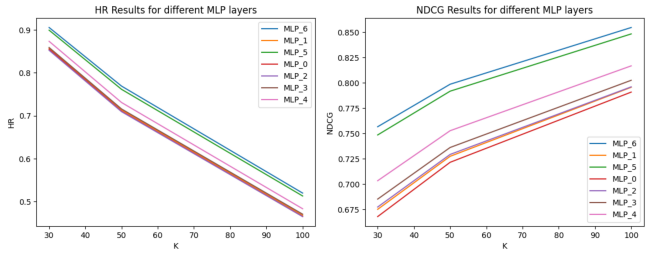
Lastly, We evaluated the performance of the NCF model in terms of Hit Rate and Normalized Discounted Cumulative Gain (NDCG) for various k. The formula for Hit Rate and NDCG is as follows:

$$\text{DCG}_\text{p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i+1)} \tag{5}$$

$$\text{IDCG}_\text{p} = \sum_{i=1}^{|REL_p|} \frac{2^{\text{reli}} - 1}{\log_2(i+1)} \tag{6}$$

$$\text{NDCG}_\text{p} = \frac{DCG_p}{IDCG_p} \tag{7}$$

In brief, the hit rate metric measures the ability of the recommender system to include relevant items in its top recommendations. To our surprise, our results showed a Hit Rate that was not what we were expecting. The Hit rate curve shows an inverse relationship with the increasing number of recommendations. This unusual behavior could potentially be caused by the nature of the input data did not meet what the model expected. In the official implementation of the model by the author, the optimal number of negative samples is set to be 4 while ours was set to be 61. On the other hand, the NDCG exhibits a normal pattern between k and NDCG values. NDCG takes into account both the relevance and the rank of the recommended items, which also indicates satisfactory performance in terms of ranking quality. As we increased the number of latent dimensions (neurons) in the GMF block, the NDCG shows a sign of improvement until when the number of neurons reached 64. It then drops to a lower level. This is a sign of overfitting.



Data Source: Appendix A - NFC RESULTS

Figure 7

We continued our experimentation with different MLP architectures after getting results regarding the effects of different latent dimensions on training loss and hit rate. Notice that the training loss from Figure 5 across different latent dimensions are roughly the same, and the differences between the hit rate curves are quite small as well. Hence, we used the default value as suggested in the official implementation for the latent dimension which is 8, and altered the MLP architecture design. It is quite interesting to see that the NDCG performs better as the complexity of the MLP block increases. There is a big jump between MLP_5 and MLP_4. Notice that the complexity between the two architectures is actually almost doubled, hence we can conclude that a more complex MLP architecture is able to capture the complex non-linear pattern in the input data.

Recall that the output from the NCF model is an array of probability corresponding to all inputted user-item pairs. Hence, we

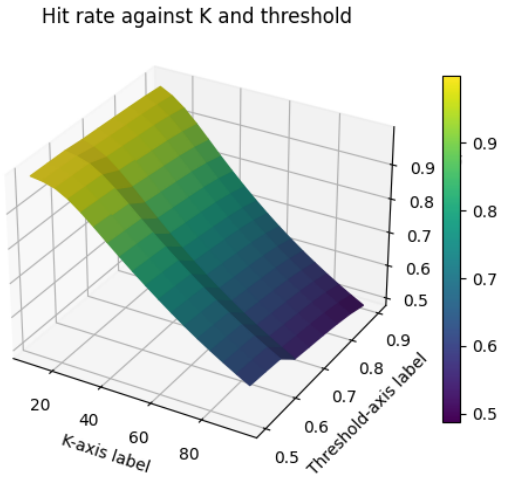


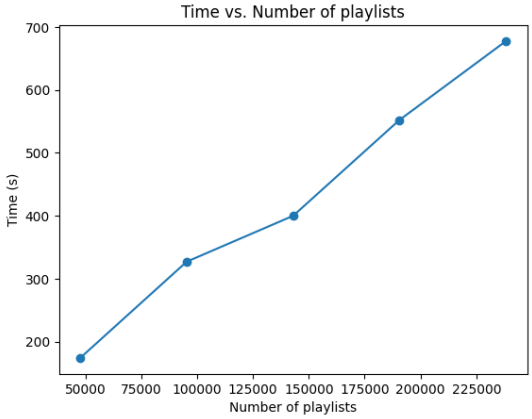
Figure 8

have to set a threshold to turn the probability measure into a binary label to compare them with the actual label in the test set which is all one. Figure 8 displays the relationship between k (number of top recommendations), hit rate, and the probability threshold. It shows a problem that we have discussed above which is that the hit rate and k should suggest an upward trend. But here we would like to focus on discussing the relationship between the probability threshold and the hit rate. But here we would like to discuss about the relationship between probability threshold and hit rate. Within the range of k values between 40 and 100, the plot shows a correct sign which tells us that as we increase the probability threshold, the hit rate decreases. Also, notice that there is an abnormal fluctuation between the probability threshold of 0.5 and 0.7. This is also showing a potential sign of the problem of data imbalance.

4.2 DIMSUM Different Similarity Values

In our experiment with the DIMSUM algorithm, we first investigate the performance of the algorithm with respect to the increase in the number of rows in the input matrix. Our analysis focuses on the relationship between the matrix scale and the algorithm's time complexity. The configuration of the Dataproc cluster we used here was a master node with n1-standard-32 with 4 n1-standard-16 worker nodes.

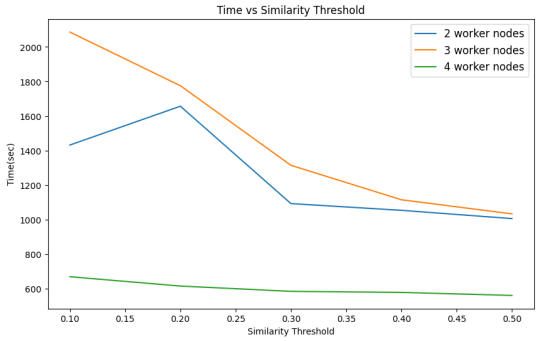
Since the paper mentioned that the algorithm managed to achieve linear complexity as the number of rows scale, we are interested in implementing it. We think that it is a powerful achievement in practise because most of the companies in the industry= usually have more users than items. The result as shown in Figure 9 shows that as the number of rows in the matrix increase, the time increase in an almost linear scale. The potential error making the graph looks slightly non-linear is because of the communication overhead problem in the distributed computation system. In the experiment, we restricted the size of playlists to 10, and at every iteration, we increased the number of playlists by 50000. With such an operation,



Data Source: Appendix C - DIMSUM PERFORMANCE

Figure 9

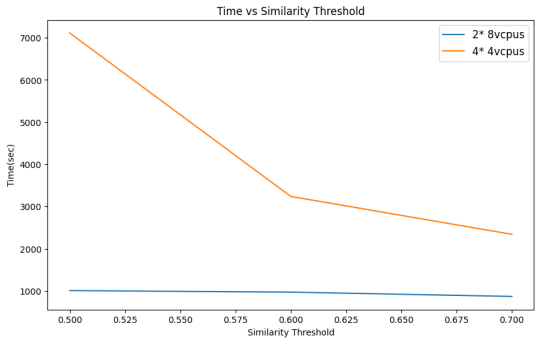
we are able to ensure that the increase in the number of columns is not as much as the increase in the number of rows and we collect the time taken for the algorithm to finish the computation.



Data Source: Appendix B - DIMSUM RESULTS

Figure 10

We have also tested running the computation across different similarity thresholds using different numbers of worker nodes with the same configuration. Recall that the similarity threshold is the hyperparameter that decides the number of pairs of items that are considered similar enough to be worth computing their similarity. By setting a higher similarity threshold, the algorithm can eliminate some column pairs with low similarity, hence lowering the complexity of the algorithm. In Figure 11, we examine the relationship between the differences in values of similarity thresholds and the time taken for finishing the computation across different numbers of worker nodes. Here, we were using a cluster with an n1-standard-16 master node and 2, 3, or 4 n1-standard-8 worker nodes. One can notice the decreasing trend in every line plot which shows the reduction in algorithmic complexity as the similarity threshold increases. Also, it is very obvious that the whole line plot drops as a whole as we increase the number of worker nodes. Notice that there is a sudden spike with the blue line when the similarity threshold is at 0.2, a potential reason that causes this is the communication error between the master node and worker nodes during the data transferring process.



Data Source: Appendix B - DIMSUM RESULTS

Figure 11

Lastly, we have also conducted a horizontal comparison to evaluate the performance of our system across different numbers of worker nodes while maintaining the same total amount of RAM and number of cores. This would allow us to understand the impact of the number of worker nodes on system performance and efficiency. One can notice from Figure 11 that as we increase the number of worker nodes while keeping the total number of RAMs and cores, the time consumption of the computation is more than doubled. The phenomena can be explained by communication overhead in the sense that the increase in worker nodes results in an increase in communication between them. It could also be caused by the higher probability of crashing of worker nodes due to the imbalance of task distribution by the master node and results in bottlenecks of nodes. We have also included the output of the algorithm in Appendix B.

4.3 Limitation

It is not hard to notice that the DIMSUM algorithm seeks to address only the computational efficiency problem underlying the traditional similarity problem. There are also models in the industry that outperform traditional algorithms such as cosine similarity in terms of efficiency and other evealuation metrics. Additionally, our data does not match the model's problem configuration, as the number of columns exceeds the number of rows since it only guarantees linear complexity as the number of rows increases, but not if the number of columns grows quicker than the number of rows. And if we are not analysing all users, we are likely dealing with a matrix with fewer rows than columns.

Furthers, as opposed to deep learning models which use GPUs, running item based collabrative filtering algorithm on Spark uses RAM. Given the advancements in high-performance computing, GPU is typically quicker than distributed computation using RAM for problems that can be solved using GPU. In our experiments, given the same amount of data, the time required to perform a computation in a distributed environment is significantly longer. For instance, when we run our computation on 50,000 playlist data on a machine with 32 vCPUs versus when we run our computation on a machine with 16 vCPUs plus 2 worker nodes with 8 vCPUs, the difference in time consumption is very substantial. However, where feasible, we can accomplish a much higher level of performance with a highly configured cluster. For example, a master node with 32 vCPUs and two worker nodes with 16 vCPUs. This is due to the fact that the improvement in completion time for computation in each worker node exceeds the cost of communication overhead.

5 Conclusion

In this project, we have implemented two different approaches to address the challenges of building scalable recommendation systems: the DIMSUM algorithm on Pyspark and the NCF model on Vertex AI Workbench. The DIMSUM algorithm, executed using the Pyspark distributed computing framework, demonstrated strong performance and potential in handling large sparse dataset and efficiently processing user-item interactions. The scalability and fault-tolarance features of Pyspark allowed us to effectively manage the growing volumes of data. On the other hand, the NCF model, implemented on Vertex AI workbench showcased its ability to capture complex, non-linear relationships between user and item. It also offers flexibility for incorporating additional features and customizations.

In conclusion, the choice of approach ultimately depends on the specific requirements of the recommendation system and the desired level of personalization. In future, we will study the data imbalance problem in our experiment and aim to discover the inherit weakness of the NCF model in the context of the quality of input data. We would also like to study into another class of models that provide a hybrid solution to recommendation system. In the context of music recommendation system based on Spotify million playlist data, we could build a better recommendation engine by scraping more content based information from Spotify official API. With more quality information, we have reasonable beliefs that a hybrid model would perform better than NCF which only based on implicit feedback data.

References

[1] 2018. Spotify Million Playlist Dataset. https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge.

[2] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2006. Greedy Layer-Wise Training of Deep Networks. In *Proceedings of the 19th International Conference on Neural Information Processing Systems* (Canada) *(NIPS'06)*. MIT Press, Cambridge, MA, USA, 153–160.

[3] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. *Proceedings of the 26th International Conference on World Wide Web - WWW '17* (2017). https://doi.org/10.1145/3038912.3052569

[4] G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science* 313, 5786 (2006), 504–507. https://doi.org/10.1126/science.1127647 arXiv:https://www.science.org/doi/pdf/10.1126/science.1127647

[5] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative Metric Learning. *Proceedings of the 26th International Conference on World Wide Web* (Apr 2017). https://doi.org/10.1145/3038912.3052639

[6] Kevin Lin. 2014. All-pairs similarity via DIMSUM. https://blog.twitter.com/engineering/en_us/a/2014/all-pairs-similarity-via-dimsum

[7] Brian McFee, Thierry Bertin-Mahieux, Daniel P.W. Ellis, and Gert R.G. Lanckriet. 2012. The million song dataset challenge. *Proceedings of the 21st international conference companion on World Wide Web - WWW '12 Companion* (2012). https://doi.org/10.1145/2187980.2188222

[8] Spotify. 2022. Spotify — Company Info. https://newsroom.spotify.com/company-info/#:~:text=Discover%2C%20manage%20and%20share%20over

[9] Le Wu, Xiangnan He, Xiang Wang, Kun Zhang, and Meng Wang. 2022. A Survey on Accuracy-oriented Neural Recommendation: From Collaborative Filtering to Information-rich Recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2022), 1–1. https://doi.org/10.1109/tkde.2022.3145690

[10] Reza Zadeh and Gunnar Carlsson. 2014. *Dimension Independent Matrix Square using MapReduce (DIMSUM)*. https://stanford.edu/~rezab/papers/dimsum.pdf

Appendix A: NFC RESULTS

Hit Rate @K with latent dim		K	thresholds	K	thresholds	K	thresholds
		100	0.85	50	0.85	30	0.85
latent dim	8	0.464996369		0.709353667		0.851940612	
	16	0.465282014		0.710278383		0.852594206	
	32	0.483076737		0.731358025		0.873727104	
	64	0.483962721		0.730583394		0.872468329	
	128	0.478138465		0.72519971		0.86724764	

NDCG @K with latent dim		K	thresholds	K	thresholds	K	thresholds
		100	0.9	50	0.9	30	0.9
latent dim	8	0.79351902		0.726213285		0.673767458	
	16	0.795323016		0.728201963		0.676618342	
	32	0.816936947		0.752358886		0.702831314	
	64	0.815860006		0.752391447		0.703154232	
	128	0.809451368		0.743951341		0.693787653	

Hit Rate @K with MLP Layers		K	thresholds	K	thresholds	K	thresholds
		100	0.9	50	0.9	30	0.9
MLP Layers	[16, 8]	0.4652336		0.710539821		0.854643751	
	[32, 16]	0.467714839		0.713585088		0.856548051	
	[64, 32]	0.464875333		0.709353667		0.852440894	
	[32, 16, 8]	0.470467199		0.71570564		0.858880013	
	[64, 32, 16]	0.483134834		0.730752844		0.87332365	
	[128, 64, 32]	0.512895183		0.761471799		0.89930606	
	[256, 128, 64, 32]	0.519658678		0.768976035		0.905519245	

NDCG @K with MLP Layers		K	thresholds	K	thresholds	K	thresholds
		100	0.9	50	0.9	30	0.9
MLP Layers	[16, 8]	0.790791509		0.72148878		0.667894928	
	[32, 16]	0.795556196		0.727397473		0.675066102	
	[64, 32]	0.795960502		0.7292431		0.677329625	
	[32, 16, 8]	0.802452246		0.736104352		0.68515822	
	[64, 32, 16]	0.816696076		0.752677809		0.703302839	
	[128, 64, 32]	0.848239719		0.791705865		0.748495719	
	[256, 128, 64, 32]	0.854557198		0.798566221		0.756613206	

Appendix B: DIMSUM RESULTS

Similarity Threshold = 0.1			
Index	**Track1**	**Track2**	**Similarity Score**
0	HUMBLE.	DNA.	1061.63106
1	HUMBLE.	XO TOUR Llif3	994.9014964
2	HUMBLE.	Congratulations	981.1266274
3	HUMBLE.	Mask Off	974.4683146
4	XO TOUR Llif3	Congratulations	922.4131429
5	XO TOUR Llif3	Mask Off	860.3947742
6	HUMBLE.	goosebumps	852.7162099
7	Bad and Boujee (feat. Lil Uzi Vert)	Bounce Back	833.8615751
8	Broccoli (feat. Lil Yachty)	Caroline	832.5946898
9	iSpy (feat. Lil Yachty)	Bounce Back	823.795754
10	Mask Off	Congratulations	807.4225986
11	HUMBLE.	Bad and Boujee (feat. Lil Uzi Vert)	797.4613609
12	No Problem (feat. Lil Wayne & 2 Chainz)	Broccoli (feat. Lil Yachty)	778.3622727
13	HUMBLE.	Bounce Back	773.3443222
14	Congratulations	Bad and Boujee (feat. Lil Uzi Vert)	773.0929852
15	Caroline	Bounce Back	771.4214973
16	iSpy (feat. Lil Yachty)	Bad and Boujee (feat. Lil Uzi Vert)	765.0322425
17	HUMBLE.	iSpy (feat. Lil Yachty)	756.6097856
18	XO TOUR Llif3	iSpy (feat. Lil Yachty)	754.3880134
19	Congratulations	iSpy (feat. Lil Yachty)	745.9988487
20	Mask Off	Bad and Boujee (feat. Lil Uzi Vert)	735.0461676

Similarity Threshold = 0.6			
Index	**Track1**	**Track2**	**Similarity Score**
0	HUMBLE.	DNA.	1088.507847
1	Broccoli (feat. Lil Yachty)	Caroline	1057.620214
2	XO TOUR Llif3	Congratulations	1025.694946
3	iSpy (feat. Lil Yachty)	Bad and Boujee (feat. Lil Uzi Vert)	1011.055851
4	Caroline	iSpy (feat. Lil Yachty)	951.100502
5	Congratulations	iSpy (feat. Lil Yachty)	938.8668232
6	Broccoli (feat. Lil Yachty)	Bad and Boujee (feat. Lil Uzi Vert)	924.3101807
7	Caroline	Bounce Back	913.5254574
8	XO TOUR Llif3	Mask Off	888.7594147
9	HUMBLE.	Mask Off	879.2195129
10	HUMBLE.	XO TOUR Llif3	865.79216
11	XO TOUR Llif3	Bank Account	847.9604073
12	One Dance	Broccoli (feat. Lil Yachty)	843.8631935
13	XO TOUR Llif3	iSpy (feat. Lil Yachty)	821.1480732
14	Bad and Boujee (feat. Lil Uzi Vert)	Bounce Back	814.0077477
15	HUMBLE.	1-800-273-8255	813.9652061
16	goosebumps	T-Shirt	799.5028296
17	Caroline	Bad and Boujee (feat. Lil Uzi Vert)	796.7868977
18	XO TOUR Llif3	Tunnel Vision	793.0195522
19	Broccoli (feat. Lil Yachty)	Fake Love	790.9753113
20	goosebumps	Bounce Back	789.1949387

Appendix C: DIMSUM PERFORMANCE

Number of playlists (units)	Number of tracks (units)	Time Taken (sec)
47696	125633	173.9194288
95330	195847	326.9017074
143002	255778	399.887583
190544	300931	552.1276495
238230	344422	677.5848219

Individual contribution

Instructions

- By submitting this document, we declare that our whole team agrees on the filled content of this document

Contribution table

Please fill the table with the estimated percentage of contribution by each member in each area.

Members	Candidate number	EDA	DIMSUM Method	NCF Method	Report writing	Overall
Hongxin Fu	52240	33.33%	33.33%	33.33%	33.33%	33.33%
Ye Hu	55974	33.33%	33.33%	33.33%	33.33%	33.33%
Tiam Tee Ng	48119	33.33%	33.33%	33.33%	33.33%	33.33%
Total		100%	100%	100%	100%	100%

Please ensure the column sum is 100% for the last 5 columns.