Comparative Performance Analysis on Model Based vs Memory Based Collaborative Filtering

Aadit Mehrotra

Abstract

This paper performs a comparative performance analysis on Model Based Collaborative Filtering (CF) and Memory Based Collaborative Filtering (CF). The paper utilizes the Movielens 20m dataset to showcase how the item based memory model and the SVD model perform on accuracy metrics and scalability. The performance analysis on both approaches was conducted utilizing the same dataset for both training and testing. The results showcased that both models perform exceedingly well on the dataset in terms of accuracy with the SVD model performing better with regard to RMSE, MAE and the F1_score on a sample size of 20000 ratings. However, as scalability was tested the increasing data size showed an improvement in the F1_score in both models with SVD performing better than the memory based model. However, the run time for SVD's fine tuned version was exponentially greater than the memory based model and SVD model that wasn't fine tuned.

Introduction

In today's information age, data is key to understanding and characterizing behavior. As the amount of data on the internet continues to grow there is a larger need for filtration and processing algorithms to be able to recognize and search for relevant data and make it usable. A method used to tackle this problem is Collaborative filtering[1] that decreases the time used in searching and retrieving accurate results. It is the process of filtering that utilizes similarities between users and items to provide recommendations to users based on a set of data.

The Tapestry project, created by Xerox PARC in 1992, can be credited with the invention of collaborative filtering algorithms[2]. Moreover, the phrase "collaborative filtering" was first used during this project. The project allowed users to annotate emails on the Tapestry platform that also offered a way to perform search queries to find specific emails. Although the system did not automatically filter information, it made significant advancements in the process by using user-specified criteria. Furthermore, as Amazon in the 1990s adopted similar recommendation system algorithms[3] it made them common knowledge amongst the software community. An anecdote that was based on the recommendation system that amazon used to advertise was "users who bought this item also bought these other items" which essentially was a fundamental building block to item based memory based collaborative filtering[4].

In recommendation systems it is a widely used technique to be able to provide accurate recommendations to users based on their interests. The primary theory behind collaborative filtering is to make predictions based on the opinions of users with similar traits. It is widely used in many industries ranging from e-commerce to audio/video streaming to social networks. For-example,

Spotify uses memory based collaborative filtering to recommend songs based on previously liked songs by the user[5]. Therefore, there is a huge need for understanding and investigating the most optimal collaborative filtering techniques as it could really benefit in providing the users with the best recommendations and increasing user engagement on platforms/services.

This paper is structured in a manner to first provide a conceptual understanding of memory based filtering and model based filtering. Then, it focuses on the algorithms used in implementing these techniques after which the actual results of the paper is presented. Following this, the evaluation section displays the final output of the paper along with the conclusion and next steps.

Memory Based

Memory based collaborative filtering has been around for the better part of the 21st century[6]. Before Netflix made model based filtering popular, the memory based systems were the first in class to be used in most recommendation system algorithms. This collaborative filtering algorithm can be performed in two different manners. Firstly, user-based[7] performs recommendations utilizing user's preferences that are similar and correlated to other users. For instance, if a user rates movies similarly to the active user. It's reasonable to infer that they share interests. As a result, we would suggest a movie if the other user has seen it and enjoyed it. Secondly, item based[8], as mentioned earlier this method was developed by Amazon that finds comparable items that consumers have previously positively connected with or liked. For both these methods, the similarity is calculated using a predetermined correlation method which in our case is the cosine measure. Cosine similarity[9] is a measure of similarity between two vectors in a multi-dimensional space. It is calculated as the dot product of the two vectors divided by the product of their magnitudes. In other words, it is a measure of the cosine of the angle between the two vectors

Model Based

Model Based Recommendation systems were first popularized by Netflix in 2006[10]. The goal of a model-based algorithm is to condense a large database into a model, which then performs recommendation tasks by incorporating a reference mechanism. The model based recommendation system utilizes the Singular Value Decomposition method[11]. This method is a matrix factorization approach of features of a dataset by reducing the space dimension. In this scenario, it uses a matrix structure where each row represents a user and each column represents an item and the elements of this matrix are the ratings that are given to items by users. The decomposition results in 3 matrices - orthogonal left singular matrix, that represents the relationship between users and latent factors, diagonal matrix, that describes the strength of each latent factor and diagonal right singular matrix, that indicates the similarity between items and latent factors. The latent factors here are the characteristics of the items, for example, the genre of the movies.

Research Methodology

The research methodology applied for this project consists of 4 steps as seen in figure 1 below.

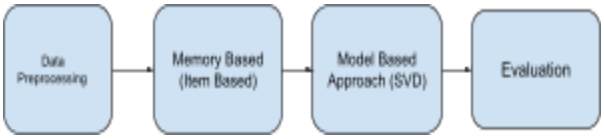

Figure 1 - Overview of Research Methodology

Data Preprocessing

For the scope of this project, I utilized the Movielens 20m dataset[12]. This dataset contains 20 million ratings and 465,000 tag applications applied to 27,000 movies by 138,000 users. The dataset has multiple csv files with information about ratings, movies, tags, links and genomes. However, for the scope of this project I would only be focusing on using the 'ratings.csv' that contains a table of 20000263 rows and 3 columns. The columns consist of the userId, movieId and ratings. In addition, I also merged the movies table from the movies.csv file to get the movie titles, genres and timestamp into a single pandas Dataframe called df_combined as seen in Figure 2.

	userId	movieId	rating	timestamp	title	genres
0	1	2	3.5	1112486027	Jumanji (1995)	AdventureIChildrenIFantasy
1	5	2	3.0	851527569	Jumanji (1995)	AdventureIChildrenIFantasy
2	13	2	3.0	849082742	Jumanji (1995)	AdventureIChildrenIFantasy
3	29	2	3.0	835562174	Jumanji (1995)	AdventureIChildrenIFantasy
4	34	2	3.0	846509384	Jumanji (1995)	AdventureIChildrenIFantasy
...	...	...	...	...	...	...
20000258	138301	121017	3.5	1420558479	The Gentleman from Epsom (1962)	ComedyICrime
20000259	138301	121019	4.5	1420558606	The Great Spy Chase (1964)	ActionIComedyIThriller
20000260	138301	121021	4.5	1420558687	Taxi for Tobruk (1961)	DramaIWar
20000261	138406	110167	4.5	1396184127	Judge and the Assassin, The	CrimeIDrama

Figure 2 - Movies and Ratings DataFrame

As we can see from the dataframe, every user is represented by an unique Id, every movie is represented by an unique Id and rating represents the rating given by the user to the corresponding movie out of 5. To understand these numbers better, the following figure 3 displays some insights drawn from the data. From figure 3, we can see that the average rating given to a movie by a user is 3.52 with the min being 0.5 and max being 5. Therefore, there are no ratings with 0 or NA making calculations easier as we build our models in the following steps.

	userId	movieId	rating
count	2.000026e+07	2.000026e+07	2.000026e+07
mean	6.904587e+04	9.041567e+03	3.525529e+00
std	4.003863e+04	1.978948e+04	1.051989e+00
min	1.000000e+00	1.000000e+00	5.000000e-01
25%	3.439500e+04	9.020000e+02	3.000000e+00
50%	6.914100e+04	2.167000e+03	3.500000e+00
75%	1.036370e+05	4.770000e+03	4.000000e+00
max	1.384930e+05	1.312620e+05	5.000000e+00

Figure 3 - Insights from DataFrame

Lastly, to visualize our data better and understand where the majority of the ratings lie I plotted the total number of ratings vs the mean ratings in Figure 4 below.

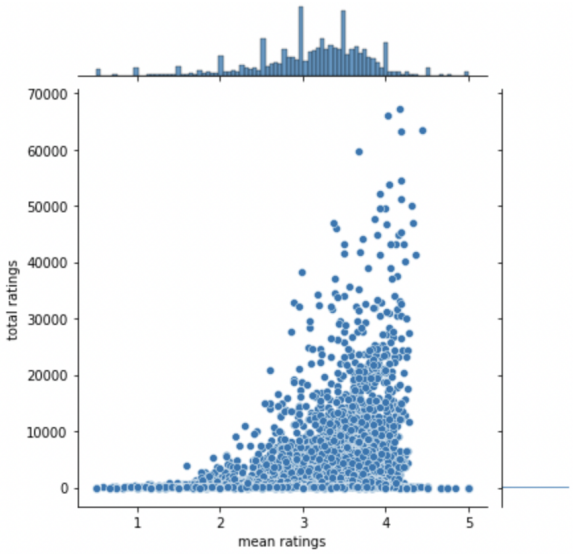

Figure 4 - Total Ratings vs Mean Ratings

From Figure 4, we can see every data Point represents a distinct Movie, with x-coordinate representing the mean of all the ratings of the corresponding users and y-coordinate - the total no of users which has rated that movie and. In addition, it's valuable to note that there is a huge density in the region corresponding to 0-100000 no of users and between mean ratings 3 and 4 .

Memory Based

Next, we begin constructing our memory based collaborative filtering function. Here the

focus is on doing item based collaborative filtering. Therefore, to implement this method, the data is split into a training set and a test set with a 9:1 ratio using the train_test_split module from the 'surprise' package. The training set is then used to calculate the similarity between movies in the rating dataframe by using KNN (k-nearest neighbors)[13], which involves identifying the k most similar movies and then combining their ratings to make a recommendation. Therefore, to implement this I imported the KNNWithMeans module from the 'surprise' package[14]. Following this, I choose the cosine similarity score to measure the similarity between items, which is a measure of the similarity between two vectors, or in this case, the ratings of two movies. The cosine similarity between two movies A and B is calculated by the formula in figure 5.

$$\text{cosine similarity} = S_C(A,B) := \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Figure 5 - Cosine Similarity[9]

Where the numerator is the dot product of the vectors and the denominator is the Euclidean norm of the vectors. After calculating the similarity score between movies using KNN and cosine similarity, the recommendations are made for every movie in the test set by combining the ratings of the k most similar movies through the weighted average of the ratings, where the weights are determined by the similarity between the movies. This array of recommendations are the predictions the memory based model makes on the testset and will further be used when calculating different metrics in the performance evaluation section.

Model Based

Firstly, similar to the memory based model a matrix is created of userIds and movieIds with the cells in the matrix containing the ratings the users have given to the movies. The same train test split as the memory based model was made using a random state of 1 to ensure similar subsets were tested. Next, utilizing the surprise package the SVD model is used to decompose this matrix into a product of three lower-dimensional matrices: a user matrix, a movie matrix, and a diagonal matrix of singular values. The user matrix would contain information about how users rate different movies, while the movie matrix would contain information about the characteristics of different movies. The singular values would represent the relative importance of the different factors in the data.

$$\underset{m\times n}{\hat{X}} \approx \underset{m\times r}{U} \; \underset{r\times r}{S} \; \underset{r\times n}{V^{\mathsf{T}}}$$

Figure 6 - SVD Decomposition[11]

In Figure 6, X denotes the utility matrix, and U is a left singular matrix, representing the relationship between users and latent factors. S is a diagonal matrix describing the strength of each latent factor, while V transpose is a right singular matrix, indicating the similarity between movies and latent factors. The latent factors are abstract values that represent different characteristics of movies such as their genres. The model decreases the dimension of the utility matrix by extracting its latent factors. This maps each user and each movie into a latent space with dimension r as seen in Figure 6. Therefore, helping to better understand the relationship between users and movies as it makes them more directly comparable.

For this project, two types of SVD models were run - one which was tuned with the most optimal hyper parameters and the other not being tuned. The hyperparameters for the

SVD model include the number of factors, number of epochs and the learning rate. For the tuned model a different combination of a range of each of these hyper parameters was tested to use the most optimal ones for each sample size. Firstly, for the learning rate I used 4 different values in the range 0.005 and 0.1. Secondly for the number of factors, I used 4 different values in the range 25 to 100. Lastly, for the number of epochs, I used 4 different values in the range 10 to 40.

Next, using the trained SVD model and SVD tuned model we evaluate it with the test data to generate recommendations for each movie in the test dataset. To generate predictions, the user and movie matrices are used to compute the predicted rating for a given user and movie. This is done by first representing the user and movie in the latent space using the user and item matrices, and then taking the dot product of these representations to compute the predicted rating. The predicted rating is then used to recommend unseen movies from the test dataset to users.

Evaluation Metrics

To evaluate both model based and memory based collaborative filtering offline testing was performed. The offline testing focused on analyzing quantitative metrics for both approaches focusing on accuracy and scalability and runtime.

To measure accuracy of the two different approaches, four different metrics were used.

Firstly, I used RMSE, which stands for root mean squared error, and it measures the difference between the predicted ratings and the actual ratings. This metric is used because it is sensitive to large errors, which can be particularly problematic in a recommendation system. Thus, punishing large errors more than small ones. Figure 7 shows the formula used to calculate this measure.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Figure 7 - Formula for RMSE[15]

Secondly, I used MAE, which stands for mean absolute error, and it measures the average difference between the predicted ratings and the actual ratings. This metric is less sensitive to large errors than RMSE, but it is still a useful measure of the overall accuracy of a recommendation system

$$mae = \frac{\sum_{i=1}^{n} abs\,(y_i - \lambda(x_i))}{n}$$

Figure 8 - Formula of MAE[16]

Lastly, I used the F1 score which is a measure of the balance between precision and recall. It is calculated as the harmonic mean of precision and recall, and is expressed as a value between 0 and 1, with higher values indicating better performance. For the scope of this project, I have taken only the top 10 recommendations to calculate precision and recall and the relevancy to be a rating greater than or equal to 3.5. To calculate precision and recall I used the formula in Figure 9.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

TP = True positive
TN = True negative
FP = False positive
FN = False negative

Figure 9 - Formula for Precision, Recall and F1_score[17]

Where TP is the count of all predicted ratings greater than 3.5 that are correct, TN is the count of all predicted ratings less than 3.5 that are correct, FP is the count of all predicted

rating that are greater than 3.5 but are incorrect and FN is the count of all predicted ratings that are less than 3.5 but are incorrect. Finally, to measure scalability of the two different models, I created a range of 5 increasing sample sizes and ran both the approaches on them to calculate and plot the varying performance metrics as the dataset grows. In addition to this, run time was evaluated as the difference between start and end time of each model to generate predictions and plotted against the different sample sizes.

Results

Metrics	SVD Model CF	SVD Tuned Model CF	Item Based Memory CF
MAE	0.72477	0.745832	0.757293
RMSE	0.93426	0.947861	0.971166
F1_Score	0.404	0.424	0.398
RunTime	0.10312	158.87165	0.02143

Table 1 - Metrics for 5000 sample size

On the sample size of 5000, the memory based model performed the worst with higher MAE and RMSE scores along with lower F1_scores than the SVD or SVDTuned models. However, SVDTuned took exponentially more time than the other two models although the additional runtime compared to the SVD model did show 0.02 greater F1_score than the SVD model and 0.026 greater F1_score than the memory based model. Next, I tested these different metrics on the scalability factor. This was done by creating subsets of the dataset from different data sizes between a range of 20000 and 80000.

Firstly, as we can see from figure 10 as the sample size increased the RMSE value of the SVD Tuned model and Memory model both decreased with the SVD Tuned model performing the best on the largest data sample size of 80000 with the lowest RMSE.

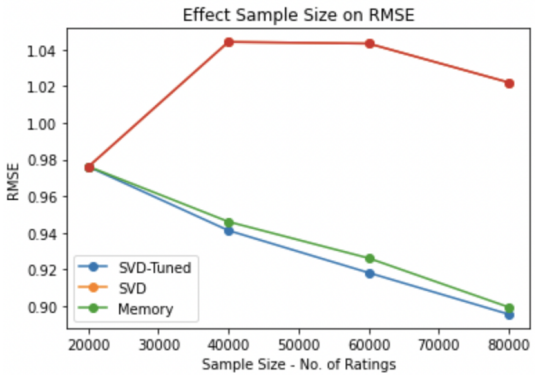

Figure 10 - Effect of Sample Size on RMSE

However, the SVD model that wasn't tuned did not perform as well as the tuned or the memory based models as it RMSE increased as the data sample size increased.

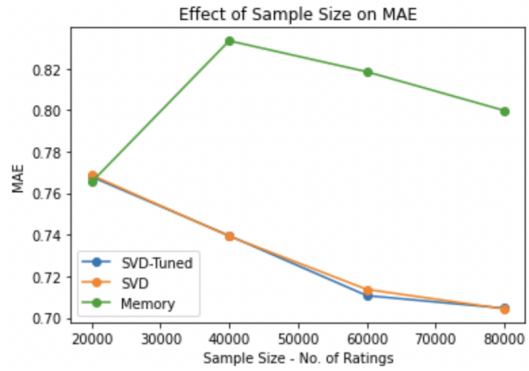

Figure 11 - Effect of Sample Size on MAE

Secondly, as we can see from Figure 11 as the sample size increased the MAE value of the SVD Tuned model and SVD model both decreased with the SVD model performing the best on the largest data sample size of 80000 with the lowest RMSE. However, the Memory model did not perform as well as the SVD models as its MAE increased as the data sample size increased.

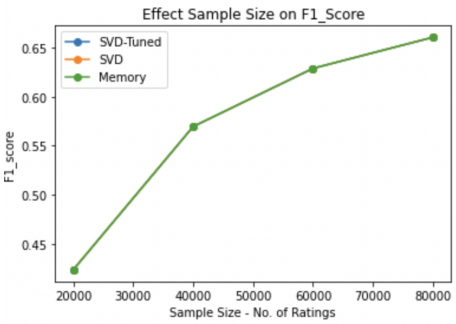

Figure 12 - Effect of Sample Size on MAE

Thirdly, as we can see from Figure 12 as the sample size increased the F1_score of the SVD Tuned model, SVD model and Memory Model all increased significantly.

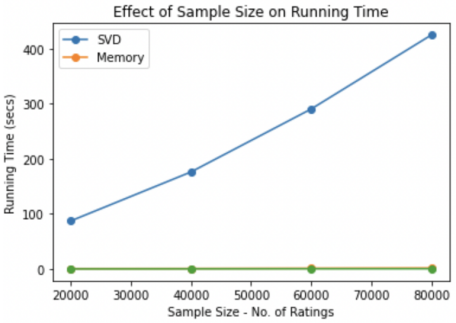

Figure 13 - Effect of Sample Size on RunTime

Lastly, as we can see from Figure 13 as the sample size increased the Runtime of the SVD Tuned model, SVD model and Memory Model all increased. However with the SVD Tuned model the runtime was exponentially greater compared to the other two models.

Analysis

Based on the results from the performance analysis on the movielens 20m dataset, the SVD Tuned model performed the best on all accuracy measures with the increase in data sample size. This was primarily as the model was fine tuned to utilize the most optimal hyperparameters of epochs, learning rate and latent factors. However, the runtime of this model was exponentially more than the other 2 models that could be a significant shortcoming of this model if utilized as a recommender system. On the other hand the SVD model that wasn't fine tuned did perform poorly on the RMSE score that implied that it made a few predictions that were way off the actual predictions. However the model performed well on both the F1_score and MAE as the data sample increased implying that except for a few ratings on average the model was accurate in its predictions while having a low runtime. Lastly, the memory based item CF performed badly on the MAE as the sample size increased implying that this model did not have too many ratings that were way off but on average produced ratings that weren't as accurate as the other two models. Although, the model had a better runtime than the fine tuned SVD model but as the data size increased the SVD model that wasn't fine tuned performed better in terms of runtime.

Conclusion

In Conclusion, from the comparative performance analysis on the movielens 20m dataset, the SVD Fine Tuned model would be the most effective model with regard to accuracy of predicted ratings between a sample size of 20000 and 80000 if runtime wasn't a point of concern for the user. However, if runtime is a major factor for the end user that is usually the case when utilizing recommendation services such as Spotify or Netflix, the SVD model would be the most preferred recommendation system as on average it produces accurate results with the lowest runtime compared to any of the other models.

Limitations

For the scope of this project the dataset was limited to a max sample size of 80,000 ratings due to time constraints. However, it would be interesting to see which model performs

better if the complete 20m ratings were trained upon. In addition, the hyperparameters of the fine tuned model were restricted to a discrete range of 4 values that could have limited the potential of the model to use the most optimal hyperparameters to predict ratings. Lastly, this comparative analysis was performed on a singular dataset with no Nan or 0 values making it easier for the model to predict ratings. However, in the real world all items/movies would not have ratings available, thus might affect the performance of models on predicting the correct rating for such movies.

Recommendations

As this scope of this projected was limited to a maximum of 80000 ratings, the first recommendation would be to use a gpu such as cuda to train and test the models with a large sample size to see if it changes the results.Furthermore, another recommendation would be to test the models on different datasets to understand to what extent do the latent factors impact performance of the models as different datasets would entail a variety of varying latent factors. In addition, these models should also be tested against a deep learning model that takes into account categorical features using embeddings, while processing continuous features using the bottom multilayer perceptron aiding in providing a more holistic view of recommendation systems.

References

[1]“Collaborative filtering | machine learning |google developers,” *Google*. [Online]. https://developers.google.com/machine-learning/recommendation/collaborative/basics. [Accessed: 08-Dec-2022].

[2]“Short history of collaborative filtering,” *Short History of Collaborative Filtering - information retrieval, recommendations, Tapestry, Amazon, Ringo, Firefly*. [Online]. Available: https://www.moyak.com/papers/collaborative-filtering.html. [Accessed: 08-Dec-2022].

[3]“Two decades of recommender systems at Amazon.” [Online]. Available: https://assets.amazon.science/76/9e/7eac89c14a838746e91dde0a5e9f/two-decades-of-recommender-systems-at-amazon.pdf. [Accessed: 08-Dec-2022].

[4]A. E. Nixon, “Building a memory based collaborative filtering recommender,” *Medium*, 25-Feb-2021. [Online]. Available: https://towardsdatascience.com/how-does-collaborative-filtering-work-da56ea94e331. [Accessed: 08-Dec-2022].

[5]Thecleverprogrammer, “Spotify recommendation system with MachineLearning,” *thecleverprogrammer*, 11-Apr-2022. [Online]. Available: https://thecleverprogrammer.com/2021/03/03/spotify-recommendation-system-with-machine-learning/. [Accessed: 08-Dec-2022].

[6]“User-based collaborative filtering,” *GeeksforGeeks*, 16-Jul-2020. [Online]. Available: https://www.geeksforgeeks.org/user-based-collaborative-filtering/. [Accessed: 08-Dec-2022].

[7]Y. Jeong, “Item-based collaborative filtering in Python,” *Medium*, 22-Apr-2021. [Online]. Available: https://towardsdatascience.com/item-based-collaborative-filtering-in-python-91f747200fab. [Accessed: 08-Dec-2022].

[8]R. Alake, “Understanding cosine similarity and its application,” *Medium*, 03-Nov-2021.

[Online]. Available: https://towardsdatascience.com/understanding-cosine-similarity-and-its-application-fd42f585296a. [Accessed: 08-Dec-2022].

[9]D. Chong, “Deep dive into Netflix's Recommender System,” *Medium*, 24-Sep-2021. [Online]. Available: https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48. [Accessed: 08-Dec-2022].

[10]“Singular value decomposition,” *Singular Value Decomposition - an overview | ScienceDirect Topics*. [Online]. Available: https://www.sciencedirect.com/topics/engineering/singular-value-decomposition. [Accessed: 08-Dec-2022].

[11]“Movielens 20m dataset,” *GroupLens*, 02-Mar-2021. [Online]. Available: https://grouplens.org/datasets/movielens/20m/. [Accessed: 08-Dec-2022].

[12] IBM Corporation. "The k-nearest neighbors algorithm." [Online]. Available: https://www.ibm.com/topics/knn. [Accessed: 08-Dec-2022].

[13] N. Hug, “Surprise Python,” *Surprise*. [Online]. Available: https://surpriselib.com/. [Accessed: 08-Dec-2022].

[14] “Root-mean-squared error,” *Root-Mean-Squared Error - an overview | ScienceDirect Topics*. [Online]. Available: https://www.sciencedirect.com/topics/engineering/root-mean-squared-error. [Accessed: 08-Dec-2022].

[15] Stephanie, “Absolute error & mean absolute error (MAE),” *Statistics How To*, 28-Dec-2020. [Online]. Available: https://www.statisticshowto.com/absolute-error/. [Accessed: 08-Dec-2022].

[16] J. Brownlee, “How to calculate precision, recall, F1, and more for Deep learning models,” *MachineLearningMastery.com*, 22-Aug-2022. [Online]. Available: https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/. [Accessed: 08-Dec-2022].