Movie Recommender System

1st Alvaro Aquino
Electrical Engineering Department
Stevens Institute of Technology
New Jersey, USA
aaquino3@stevens.edu

2nd Xuanjin Ding
Mathematical Sciences Department
Stevens Institute of Technology
New Jersey, USA
xding22@stevens.edu

3rd Farokh Cooper
Computer Science Department
Stevens Institute of Technology
New Jersey, USA
fcooper@stevens.edu

***Abstract*—There are a vast variety of movie options to choose from when sitting down to watch a movie. This variety can create difficulties in selecting a movie, so to tackle this problem, Machine Learning algorithms such as Content Based Filtering models, Collaborative Memory Based Filter, and Collaborative Model Based Filter models were used. For the Content Based Filter model the accuracy of the movie recommender was measured using a hit-rate. The hit-rate resulted in a score of .032. We then implemented an autoencoder implementation for Content based filtering which provided an MAE of 7.4605e-04 and a RMSE of 0.0273 which is a major improvement over the previous implementations. Further we worked on another approach to create two different kinds of collaborative filters. The Memory Based Collaborative model works on an Embedded Neural Network and showed a loss of 0.0005184, which is a respectable result and the Model Based Collaborative model uses SVD to get a RMSE score of 0.9975 and a MAE score of 0.7479 which is respectable result too but shows our model is primitive compared to the systems implemented at platforms like Netflix.**

I. Introduction

The project is based on a movie recommender system. Lots of people like to go and watch movies but they have a hard time picking out between the different options available to them. Depending on the mood one is feeling one may decide that they are up for a laugh, other times if they are with a partner they may want to watch a more intimate movie such as romance, and in other cases their may be a part 2 to a movie they already watched and can't miss. It is clear that on top of the different quantity of movies available the quality in terms of the movie genre is also expansive. We are now living in the modern Internet era and on top of being occupied by commitments from work and life, people's attention is being captivated by other content that doesn't make them have to have to put in effort to consume. Even though choosing a movie is fun it can also be time consuming and not the proper allocation of one's time, plus if you are watching with others, on top of their being so many options we all have different tastes. Recommendation systems make it easier for consumers to make the right choices, without having to spend time debating on which movie to watch. A recommendation system determines movies that an consumers will find engaging to watch . Nowadays there are many recommendation systems that exist in products we consistently use. Take Youtube as an example, when you listen to a music video or watch any other videos you come across other great recommendations. By the time you notice, you have already gone down a wormhole and have spent a few hours scrolling though the internet. Moreover, in apps like Netflix and Disney the power of a recommendation systems extends much beyond just providing a recommendation. By using machine learning algorithms the recommendation system exhausts all possible options to create a personalized list for each and every individual. The days of being overwhelmed with options is now over. By compiling lots of data on consumers such as their search and browse history and what other people with similar interests are watching the algorithms can easily narrow down the results. Movie recommender systems also ask for feedback, after the user is done watching the movie, on what they thought about the recommendation. This feedback doesn't just inform the companies about the consumers opinion, about what they thought, it also helps improve the recommendations for the next time they run into the same issue of deciding what movie to watch.

In general when working on a machine learning algorithm to get started one must gather the data. An algorithm can't learn without the data and in most cases the data is the most important part when implementing an algorithm. Data can come in many different formats and can be presented in different way as well. However, just because data is uploaded to a machine that does not mean that the machine has learned something new. To be able to learn data must be manipulated, so that it can be fed and fine tuned through the model.

MovieLens is a movie recommendation dataset that was collected by the GroupLens Research Project at the University of Minnesota. It contains data on movie ratings, movie metadata, and demographic information about the users who provided the ratings. The dataset is widely used for research on recommendation systems and thus we are building our models on this dataset too.

As in most cases in machine learning there are many ways to approach a problem. In some cases, algorithms can be improved and even combined to provide the best result. In other cases, algorithms may solve the problem but they do not provide the best solution. Cost functions that will be applied include Root Mean Square Error (RMSE), normalization, Cosine Similarity index, Hit Rate, Mean Absolute Error (MAE) and loss functions. As for the algorithms, the content based filtering, memory based collaborative filtering, and model based collaborative filtering algorithm have been chosen to approach the problem. The content based filter uses item features to determine other similar items the user may be interested in. The memory collaborative based filtering takes a person's previous preferences to come up with similar recommendations. The model based filtering system uses the popular model of SVM to implement the system. Ultimately, these algorithms are encompassing the scope of the main

objective of a movie recommendation system to recommend movies.

II. Related Work

For Memory Based Collaborative filtering, several algorithms have been used to approach the problem. These algorithm included User-Based CF and Item Based CF. Although they were able to address the problem at task they did have some pro's and con's to them.The advantages of using these algorithms include that they are simple to execute, data addition is simple, content should not be considered, and they are efficient and scalable. The disadvantages include that it depends on explicit suggestions, there may be cold start issue, trouble with sparsity, and the scalability is limited. For Content Based filtering, different approaches were taken to provide recommendations. Advantages in of all content based models include that they don't require data from other users to to generate recommendations. Disadvantages include the complexity of the improved models and the time it takes to implement. Also the model is very limited in the fact that it has scalability issues when new attributes are added. For Model Based Collaborative filtering an algorithm that has been used is the slope one cf algorithm. The advantages of using this algorithm are that it enhances the efficiency of prediction and that it improves issues with scalability and sparsity for huge dataset's. The disadvantages include that the model is costly and there may be loss of information in a factorization matrix. For Content based collaborative filtering the Hidden Markov algorithm is usually implemented. Advantages include that there are no issue with scarcity and cold start and that it ensures confidentiality. Disadvantages include that it needs detailed description of items, it requires ordered user profile, and the concern is material overspecialization.

III. Our Solution

A. Description of Dataset

The data sets used in this project is the 1 million and the 20 million movie lens data set. There are 6 csv files such as movies, ratings, links, tags and etc.. Among those 6 files, we mainly use the movies and ratings data set. In these two datasets, userId, movieId and rating are consistent. Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars). Users were selected at random for inclusion. Their ids have been anonymized. Only movies with at least one rating or tag are included in the dataset. With these, the dataset is valid and suitable for this project.

When it comes to the content based filter model, for the visualization of the raw data the number of genres for each movie was calculated. This was done in order to be able to determine which genres were the most popular and which genres were the least popular. The point behind doing such was because the tfidf vectors were going to be based off of genres. In order to make it easier for the viewer to visualize a word chart was created.

A different content based model was created to generate movie recommendations. In this model the genome-tags data set file was used. This file provides tag descriptions to the related tag Ids. The genome score gives the score of the movie relevance tag to the movie.

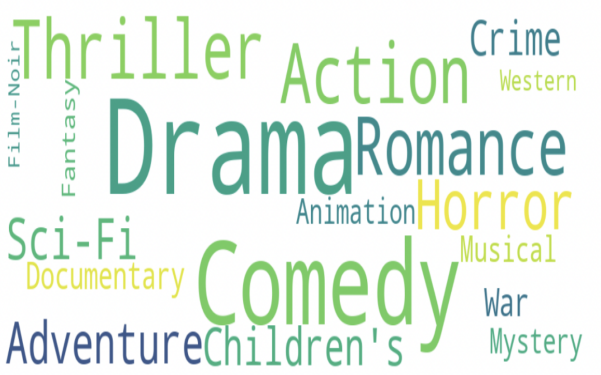

Fig. 1. Word Cloud

B. Machine Learning Algorithms

After pre-processing the data and deciding on the algorithms the group was able to implement. For content based learning TfidVectorizer was imported in order to convert non-numerical data into numerical values. This approach to the problem is appropriate because it will provide a recommendation based on the features initially made available. Once this was done the data was fitted and transformed, which provided a vector with numbers that represented the features. Afterwards, cosine similarity function can be applied to provide a similarity confidence value. The score value is stored into a list and the feature vector is fed through cosine similarity to identify which movies are similar to one another. By using the input function, the model is given a movie title, which is then compared to all other movies. Depending on movie name given by the user, the difflib function that was imported allows to pinpoint the movie that is the closest match in the list of movies.Once the closest match is identified, the index of the movie is found using the title. The point of finding the index value is to identify what the similarity score was for the movie. A variable labeled as similarity score was set equal to the list of movies. By using the enumerating function the model goes through all the data providing a score for each index. Ultimately, a model listing all the movies and their score will be provided based on the users movie selection.

Another implementation uses an Autoencoder to implement Content based filtering. An Autoencoder is a type of neural network that is used for unsupervised learning. It is called an autoencoder because it is composed of two parts: an encoder and a decoder. The encoder maps the input data to a lower-dimensional representation, or encoding, and the decoder maps the encoding back to the original input data. The goal of an autoencoder is to learn a compressed representation of the input data that captures the most important features or patterns in the data. This can be useful for a variety of tasks, such as dimensionality reduction, data denoising, and feature learning. An autoencoder consists of an input layer, an encoding layer, and a decoding layer. The encoding layer typically has fewer units than the input and decoding layers, which forces the network to learn a compact representation of the input data. The autoencoder is trained by minimizing the reconstruction error between the input data and the output of the decoder.

Within recommendation systems, there is a group of mod-

els called collaborative-filtering. In these models, it is to find the similarities between users or between items based on user preferences or ratings. In model-based filtering. there are three most used algorithms: SVD, NMF and SVDpp. SVD model uses an estimating rating users on different item i. In the calculation it first calculates the overall average rating, the every other parameter in this model is calculated using the gradient descent method. Therefore, the model will try to fit the estimated rating on all known ratings, minimise the MSE and return the closet fit. Also, in this model, each user will be represented by their matrix, which is the matrix formed by the hyperparameter of the model. These matrices will capture their essence in n numbers and user pairs to calculate the ratings. Even though SVD model seems simple, the analysing the result is a bit more complicated. Different factors will have different impact on the model, sometimes, one factor can fit the model well. In this case, to further analysis the model, using more factor as a basis for a cluster analysis would be a good option. Hence, we will choose to use SVD model for model-based filtering in the collaborative-filtering.

We also used a Memory based collaborative-filtering algorithm in the recommendation system. Memory based algorithms can again be classified into two parts, item based approach and user based approach. Memory algorithms in general provide predictions based on previous interactions, for user based approached its based on previous users data and choices (items) for similar users and for item based approach its based on previous items that have been chosen by the user to make predictions on the users next choice. The simplest way to implement such a rank for user preference would be by calculating a simple average of ratings for all the users which doesn't take into consideration that the users are also similar to one another. This is combated by calculating a similarity index too using the Cosine Similarity index. While this is an efficient method, it doesn't account for the fact that some people give more lenient or strict ratings naturally, so to counter that we will normalize the ratings by subtracting the item weight from average ratings of that user. After this we can classify all users into Neighborhoods based on similarity with other users. Neighborhoods solve the problem of very time-consuming process of calculating all possible user-combinations, probably in the millions and also help us solve the problem of sparsity of data.

For this, we implemented a Neural Network using the PyTorch framework. PyTorch is a popular open-source deep learning framework used for designing, training, and evaluating deep learning models. To build a neural network in PyTorch, we define the layers of the network and the forward pass, which specifies how the input data is transformed as it passes through the layers of the network. We use PyTorch's built-in layer, the fully connected layer

Once we have defined our neural network, we can train it using PyTorch's optimization algorithms and loss functions. PyTorch provides a range of optimization algorithms, such as stochastic gradient descent and Adam, and a variety of loss functions, including cross-entropy loss and mean squared error. We used the Adam optimizer in our PyTorch Neural Network. To use Adam in a neural network, we specify the learning rate and other hyperparameters, such as the loss function, the evaluation metrics and decay rates for the exponential moving averages. Adam (Adaptive Moment Estimation) is a popular optimization algorithm for training deep learning models. It is a variant of stochastic gradient descent that uses adaptive learning rates, which means that the learning rates for each parameter are adjusted based on the historical gradient information. In Adam, the learning rate for each parameter is updated using an exponentially decaying average of the past squared gradients, as well as an exponentially decaying average of the past gradients. The learning rate is then adjusted based on the ratio of these two averages. One of the key benefits of Adam is that it can adapt the learning rates of different parameters at different rates, which can help the model converge more quickly and achieve better performance. Adam also requires fewer hyperparameters to be set compared to other optimization algorithms, which makes it easier to use and tune.Overall, Adam is a widely used optimization algorithm for training deep learning models, and it has been shown to be effective for a variety of tasks including recommendation systems.

C. Implementation Details

We firstly start with pre-processing the Movies data set as it contains errors. We looked for special characters except for !,-, , ?, (and). This will help us identify the movie titles that contain errors in their title columns and ignores the commonly used special characters to not create false positives. Once we fixed this problem we also excluded from the users data set, the timestamp, since it is an irrelevant feature which is not relevant to any algorithm which we have implemented in our systems. We also ran the algorithms on the users data-set and We will also train test split the ratings from each user to compare the accuracy.

a) Content Based Filtering System:: For the first implementation of the content based filter, TFIDF and cosine similarity was applied. The TFIDF vector will provide a vector representation of the data we will be using to create the recommender. Hence, why the value counts functions was used, as this will provide us with the term frequency. This is multiplied by the weight of the term frequency, which is also based on how many time it appears. The movies with less frequency were assigned a higher weight. Given a string of a specific movie, the TFIDF vectorizer helps provide us with a sequence of features, which gives us the TFIDF Vectors. Then by using cosine similarity, we can find vectors that are similar. By using the vector with inputs and comparing it to another, both vectors are compared to determine a similarity measure.

For the second implementation of the Content Based Filter, a genre vector is mapped out for each movie. What this does is label each genre with a 1 and the genres that a movie excludes with a 0. By merging the dataframes, for one movie we are able to see the many tags that are associated with it. Then the relevance score for each tag is listed in the same dataframe and the top ten tags are listed in the relevance rank section.

As the Relevance Rank increases the median relevance score decreases as well. Therefore, in the chart below the box plot we are able to visually see what the ideal relevance rank score would be.

This approach is different from the prior content based filter because instead of using cosine similarity as a way of

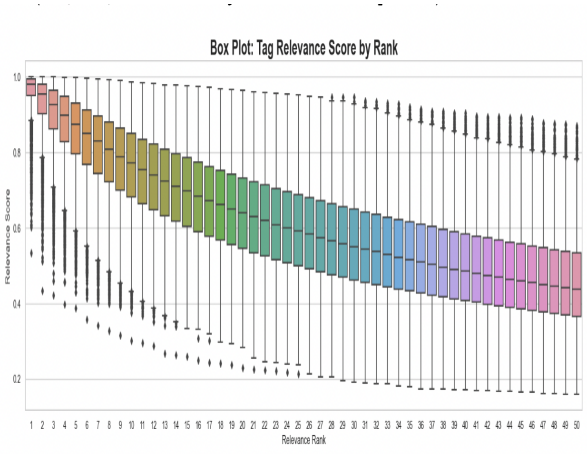

Fig. 2. Tag Relevance Score

measuring similarity the Jaccard Index was used. The Jaccard index compares members from two indexes to see which members are shared and which are distinct. In our case, If three movies are compared depending on which two vectors are the closely similar they are assigned a higher Jaccard similarity score.

Since a ratings file was provided RMSE and MAE was calculated. However, recommendations for the movies was based on top-N recommenders. Typically recommender systems rely on RMSE and MAE as a measure of accuracy, but for the content based filter it is ideal to measure the model's accuracy by applying a Hit Rate. The hit rate measures whether one of the top n recommendations was a hit. By applying the Leave One Out Cross-Validation method (LOOCV), the algorithm can be applied once for each instance. Since we have a huge data set this approach becomes very valuable. The problem that arises with the hit rate is that it results in a very low score as we have a very basic setup and the algorithm is not meant to handle a huge dataset accurately. To resolve this issue we decided to improve the model and use a collaborative base filter model.

```
def MAE(predictions):
        return accuracy.mae(predictions, verbose=False)
def RMSE(predictions):
        return accuracy.rmse(predictions, verbose=False)

print("RMSE: ", RMSE(predictions))
print("MAE: ", MAE(predictions))
```

```
RMSE:  0.8781708043233208
MAE:  0.690406590065625
```

Fig. 3. RMSE

We then used an Autoencoder based Content Filtering model which we implement by finding similiar movies using the information provided by the tag csv file. We first create a Panda Dataframe of all the tags associated with a particular movie using the groupby function and obtain a movie-tag association list. This tag dataframe obtained is then merged with the movie dataset by applying a merge based on MovieId. This creates a tag document listing all unique movies with its associated tags and genre values.

Our next step is to create a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer using the tag document. The parameters used for the vectorizer is an n-gram range between 0 to 1 which creates a continous set of data with values between 0 to 1. We then use min-df parameter to be 0.0001 which removes all values occuring less than 0.1 percentage in the tag document. We use stopwords parameter as English to remove redundant English stopwords like 'The' and 'And'. This creates our TF-IDF vocabulary in the form of a vector matrix. Words that occur frequently in a document but are rare across the collection of documents will have a higher TF-IDF score.

We then define the main function of our Autoencoder which implements the Encoder and Decoder functionalities of the neural network. Both use the 'Relu' activation function, both are regularized with a value of 0.2 and then both are normalized using the BatchNormalization function.

We then run the autoencoder system on a train-test split of 80-20 dataset. The loss function is set to MSE and the evaluation metrics are the Mean Squared Error and the Root Mean Squared Error. We use 10 epochs to train and test our model with a batch size of 500, with shuffle parameter = true to randomize the data further. The training and testing losses are returned in a .csv file and the loss per epoch is plotted. We then

```
input_size = tfidf_mat.shape[1]
intermediate_size=800
encoded_size=90

def autoencoder():

    #encoder
    model = Sequential()
    model.add(Dense(intermediate_size,input_shape=(input_size,)))
    model.add(Dropout(0.2)) #regularization
    model.add(Activation('relu'))
    model.add(BatchNormalization(axis=1, name='bn_encoder1'))

    model.add(Dense(encoded_size))
    model.add(Dropout(0.2))
    model.add(Activation('relu'))
    model.add(BatchNormalization(axis=1, name='bn_encoder2'))

    #decoder
    model.add(Dense(encoded_size))
    model.add(Dropout(0.2))
    model.add(Activation('relu'))
    model.add(BatchNormalization(axis=1, name='bn_decoder1'))

    model.add(Dense(intermediate_size))
    model.add(BatchNormalization(axis=1, name='bn_decoder2'))
    model.add(Dropout(0.2))
    model.add(Activation('relu'))

    model.add(Dense(input_size,))
    model.add(Activation('sigmoid'))
    return model
```

Fig. 4. Autoencoders encode and decode functions

use the weights of the encoding layer as the embeddings for the input data. These embeddings will capture the most important features or patterns in the data, as learned by the autoencoder. Based on the embeddings and using Cosine Similarity as our Similarity matrix function, we obtain a list of the 10 similar items to a given movie name inputted which will return a csv file containing the top 10 recommendations.

b) Collaborative Filtering Model Singular Value Decomposition (SVD): In collaborative filtering, it uses the past iterations between users and the target variables. Singular Value Decomposition is a collaborative filtering method I used

for movie recommendation in this project. In recommender system, SVD is a k-rank approximation to the original SVD. Singular value Decomposition is a k-rank approximation to the original SVD.It decomposition of original sparse matrix into product of 2 low rank orthogonal matrices.

$$M = U \cdot S \cdot V^t.$$

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 factors, V transpose is a right singular matrix,indicating the similarity between items and latent factors The aim of the code implementation is to provide users' with different movie recommendation from the latent features of item-user matrices. The input of the data to the system will be all historical data of one user's interactions with different movies, which will stored in a matrix that the rows are the users and the columns are the items.

First pre-process the data, removing all the NaNs from ratings and changing it into 0.Reshape the ratings dataset and turn the dataset into a matrix as userId be the rows, movieId be the columns and rating be the values for each cell. Then, define the train and test split and split training for validations. The validation check whether the input data is correct or not and if there is any invalid value. Then, define the error function RMSE and MAE to calculate the error between truth value and the prediction. After that, define the gradient descent for the lower rank matrices of the user matrix and item matrix, in this way, it can improve the accuracy of the prediction. And the gradient descent is applied on V,W matrices, which V is the item feature matrix and W is the user matrix. After that, create a process bar for the massive loops which helps to look at the process and better manage it. Lastly, define the final prediction function, setting the initial values with regularization parameter, learning rate and loops to updating each iteration's RSME and MAE. When all the iterations have been run, the final result of the prediction has a 0.8165 training RSME, 0.9975 testing RMSE and 0.74796 MAE. To this point, the model is finished. After that, build a recommender system based on this model. In the recommender system, it uses variables: userId, ratings, movies and the prediction model. It selects the movies with ratings above 4 and output the top 5 movies that suitable for each user.

```
Epoch number: 89, training rmse: 0.8193935265311086, Testing RMSE: 0.9957225
829913301, MAE: 0.7471018091841506
Epoch number: 90, training rmse: 0.8190872153236789, Testing RMSE: 0.9959126
4693424, MAE: 0.7471964475795534
Epoch number: 91, training rmse: 0.8187851077612592, Testing RMSE: 0.9961005
776381113, MAE: 0.7472881087245559
Epoch number: 92, training rmse: 0.8184870925787487, Testing RMSE: 0.9962863
64644183, MAE: 0.7473780825308822
Epoch number: 93, training rmse: 0.8181930620107036, Testing RMSE: 0.9964700
006930325, MAE: 0.7474651448206803
Epoch number: 94, training rmse: 0.8179029116839857, Testing RMSE: 0.9966514
814180235, MAE: 0.7475493257856238
Epoch number: 95, training rmse: 0.81761654051845, Testing RMSE: 0.996830805
0505045, MAE: 0.7476304434613339
Epoch number: 96, training rmse: 0.8173338506342727, Testing RMSE: 0.9970079
721384489, MAE: 0.7477120274334069
Epoch number: 97, training rmse: 0.8170547472646 74, Testing RMSE: 0.99718298
52798927, MAE: 0.7477964645316536
Epoch number: 98, training rmse: 0.8167791386729368, Testing RMSE: 0.9973558
488722329, MAE: 0.7478808801817177
Epoch number: 99, training rmse: 0.8165069360727488, Testing RMSE: 0.9975265
68878237, MAE: 0.7479640465179355
```

Fig. 5. Last 10 Iterations of SVD

For SVD model, there are some pros and cons. One cons is that SVD can't predict if there is a NaN value in the matrix, hence before any application of the model, those NAn values need to be changed into 0s. Also, it also cannot predict for a new user, the user must be existed in the currently known and used ratings dataset. For pros, in this model, there is no domain knowledge required, since the embeddings are learned by itself. The model can help users to discover some new interests which they may not know before. Last but not least, this model only needs the feedback matrix to train a matrix factorization model in which it can be more robust to numerical error.

c) Collaborative Filtering Model Memory Based model: In the Memory based approach, we implemented a PyTorch Neural Network to create our recommendation system using User-Item memory filtering. For our neural network we use the ratings dataset. The Dataloader loads data from the ratings dataset. A Custom dataset is then defined which stores the User and Item values in LongTensor objects.

We then create two Label Encoders which use a batch size of 256. A label encoder is a preprocessing tool that is used to convert categorical labels (strings or text) into numerical values. We convert our ratings inputs into numerical data using this. This helps us form our training dataset. We then configure the model by setting the number of users we obtain, the number of movies, and the dimension of embedding. The model has a structure that calculates the rating by dot producting the embedding values of users and movies. This creates our Matrix Factorization values. Now we have everything ready for our

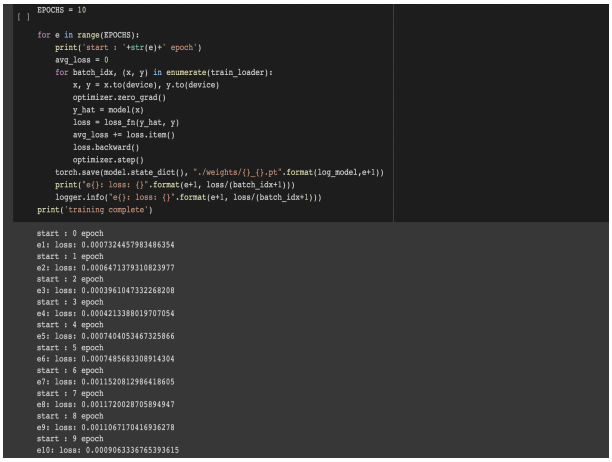

Fig. 6. Loss metrics in Memory based Neural Network

neural network, and we create a PyTorch Neural Network with learning rate set as 0.1, selecting Adam as the optimizer function and the evaluation metric as the loss function MSE. We set the number of epochs to be 10 as it is a large dataset. The loss of each epoch is returned. Once we have trained our model we return the predicted ratings for all unique MovieIds which are returned along with the actual rating for comparison. We have also used generic pieces of code to save models and to load them if the kernel is terminated. This portion is inspired by the Coursera course Special Movie Recommendation Systems.

IV. Comparison

For the content based filter model, a hit rate was implemented to measure the accuracy. In the case of the TopN Recommendations the hit rate was .03245. This was calculated by taking the number of hits/ left out predictions. This algorithm is great for content based filtering because it relies on top-n Recommendations. It would be hard to determine other accuracy scores but this performance measure is ideal.

In SVD, the accuracy is updated by learning rate and gradient descent. And RSME and MAE are used to measure the accuracy. With each iteration, the RSME and MAE are getting smaller and smaller. The first iteration has a RSME 1.3125 and MAE 1.04725 while the last iteration has a RSME 0.99752 and MAE 0.74796 which has improved by 24% in RSME and 29% in MAE. With more iteration and different learning rate, the accuracy can be improved more.

V. Future Directions

The movie recommender can be further improved by using Multi-level Ensemble Learning. Unfortunately because of time restrictions this approach wasn't able to be implemented but it would have provided for better results. In some examples that we came across, work has been performed on multi-level ensemble learning. The results reflected that the best application of Ensemble Learning relied on 2-level stacking vs single level stacking. By using two level stacking the RMSE was lowered in exchange for improved performance.

Another way the movie recommender could have been made better is by using a hybrid model. In our case a deep learning model could have been used to approach the movie recommender with a hybrid content-based/collaborative filter model. In some examples of how the approach was performed, both content based and collaborative models were treated independently and after applying an average of cosine similarities, the hybrid model can be formed.

In the implemented of autoencoder it implements for Content based filtering which provided an MAE of 7.4605e-04 and a RMSE of 0.0273 which is a major improvement over the previous implementations. The Memory Based Collaborative model works on an Embedded Neural Network and showed a loss of 0.0005184, which is a respectable result. All these models show our model is primitive compared to the systems implemented at platforms like Netflix

VI. Conclusion

All models are different and have different evaluation metrics.By including a variety of models we were able to display incremental improvement of the accuracy score. This was done by starting with simple models and working towards more complex models to reflect improvement. In the three models we did above, it shows that the accuracy of them is rank in content-based model, SVD in collaborative model based model, memory based collaborative model and autoencoder in ascending order. The problem of recommending movies was solved but with more time the problem can be improved even more.

References

[1]S. Sen, "Recommender Systems," The Owl, Jul. 06, 2020. https://medium.com/the-owl/recommender-systems-f62ad843f70c

[2]Binu Thomas and Amruth K John 2021 IOP Conf. Ser.: Mater. Sci. Eng. 1085 012011 DOI 10.1088/1757-899X/1085/1/012011

[3] Soanpef Sree Lakshmi et al 2014 (IJCSIT) International Journal of Computer Science and Information Technologies RSs: Issues and challenges Vol 5(4) 5771-72

[4]Lu, J, Wu, D, Mao, M, Wang W and Zhang G 2015 Recommender system application developments : A survey Decision Support Systems 74 12–32

[5]F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=¡http://dx.doi.org/10.1145/2827872¿

[6]Binu Thomas and Amruth K John 2021 IOP Conf. Ser.: Mater. Sci. Eng. 1085 012011