SOCIAL MEDIA ANALYSIS AND NETFLIX SHOW RECOMMENDATION SYSTEM USING NLP AND DEEP LEARNING

FINAL RESEARCH PROJECT REPORT - 2020

AMOD: BIG DATA ANALYTICS

Project By:

Aiswarya Nellayi Venkitaraman (0667620)

Rakhi Konar (0670029)

ABSTRACT

Recommendation systems are one of the most popular intelligent systems which enable users to discover new and relevant products as per their interest. Traditional approaches such as collaborative filtering and content-based recommender systems have gained popularity over the years. In this project, we propose a deep neural network-based user-item collaborative filtering recommendation system of Netflix shows for twitter users. This is built using sentiment analysis of tweets about Netflix shows and movies. Twitter data is collected using REST API in python and NLP techniques are performed to extract show names. The proposed approach is build solely using implicit feedback of the user and thereby its purely based on user's social media interaction. Deep learning MLP model produced promising results with decent accuracy and low errorrates.

INTRODUCTION

In today's world recommendation system have become integral part of our day-to-day life. We will be provided with n number of results if we are searching for anything online. This massive amount of information is a little overwhelming for the users. Recommender systems provide relevant suggestions to users based on their history of searches or by finding similarity with the history of users with the item they are looking for. Hence recommender systems are used largely by retail industries, service providers as well as entertainment industries and help their users to make decisions much faster.

Twitter is one of the largest microblogging and social networking service where registered users interact via tweets, retweets, likes etc. Users express their interest on different topics via tweets and hence they are excellent source to understand people's opinion, likes and dislikes. Even though the number of twitter users are not as big as many other mainstream applications such as YouTube or WhatsApp, we can get a lot of opinions, mentions and hashtags about topics which provides a clear understanding about the user's stand. Moreover, the textual data can be analysed and processed to interpret the underlying sentiment and hence emotions can be classified into positive, negative, and neutral. We leveraged this aspect and attempted to build a recommendation system unlike the conventional systems which uses user ratings to understand the user-item interactions.

In this project, we focus mainly on building a Netflix show recommendation system based on twitter user interests about different shows which they expressed through tweets. Deep learning and NLP (Natural Language Processing) have had immense impact on wide variety of services recently and we made use of these machine learning techniques to perform sentiment analysis of tweets and provide recommendations based on sentiment score.

RELATED WORK

Recommender systems are highly efficient in knowledge management and it delivers personalized recommendations based on the historical data and similarity of items which users are searching over the internet. Many recommender systems were built over the past years and used different approaches like Collaborative Filtering (CF), Content-Based Filtering (CBF) and Hybrid Filtering. Deep learning-based approaches are also gaining popularity and many other approaches were explained in literature.

Recommender system approaches

An overview of different recommender system approaches, research trends and future direction is given by P.K. Singh, P.K.D. Pramanik, A. Dey, P. Choudhury in their research paper [1]. Apart from CF and CBF, Demographic based recommender systems (DRS), Knowledge-based (KBRS), Context-aware (CARS) and Hybrid approaches were discussed. Collaborative Filtering is the process of evaluating items based on opinions of other people. These systems produce recommendations for a user and one or more items [2]. Automated collaborative filtering (ACF) systems uses a historical database to match user opinions with other users of similar opinions. Underlying algorithms such as non-probabilistic and probabilistic algorithms for CF were also discussed and research showed that probabilistic approaches are more popular among machine learning community.[2]

Content-based recommender systems predicts or recommends items similar to those a user has liked or interacted with in the past [3]. The system matches a user profile with item representation. A high-level architecture of the recommendation process was also analysed where three steps process – Content analyzer, Profile learner and Filtering component are used in the recommendation process. Besides content-based systems are user independent, transparent, and capable of recommending new items, its prone to several disadvantages such as limited content analysis, over-specialization, and cold user problems[3].

Hybrid recommender system combines two or more recommender techniques and provides recommendations with better performance as disadvantages of individual ones will be significantly reduced. Robin Burke [4] introduced a novel hybrid-based restaurant recommender system 'EntreeC' that combines knowledge-based recommendation and collaborative filtering techniques. Oyebode, O and Orji, R proposed a hybrid recommender system that combines the item-based collaborative technique and demographic-based approach in banking sector to drive product marketing [5].

Deep learning methods have gained popularity over the past decade and there are several notable publications for deep model-based recommender systems. All deep learning work categories such as embedding vectors, feedforward networks and autoencoders for collaborative filtering, deep feature extraction methods and session-based recommendation with recurrent neural networks where explained and compared with traditional methods. These models outperformed traditional ones by a large margin [6]. A deep learning approach based on autoencoders were used to predict new movie ratings for a user with a large database of ratings from other users. This system outperformed user-based neighborhood baseline in terms of RMSE on predicted ratings[7].

Sentiment Analysis

Sentiment Analysis is one of the popular techniques used for opinion mining and to know user preferences. Sentiment Analysis is integrated with a collaborative filtering recommender system to make recommendation based on user reviews. Opinion polarity scores are detected using a semi supervised SVM to create labelled data. The experiment gave promising results with recall of 100% and precision of more than 95% [8]. The authors of [9] proposed a movie recommendation system with sentiment analysis on Spark platform which optimises the recommendation list provided by a preliminary hybrid recommender system. VADER is computationally efficient lexicon of sentiment related words and rule-based method. VADER method is used to analyse movie review tweets which is used along with ratings to make a hybrid movie recommender system as it works well with text that has slangs, emoticons, and acronyms.

We are proposing a Netflix show recommender system using a deep learning MLP(Multi Layer Perceptron) model and uses tweets to extract show names. It then calculates sentiment score of tweets for the extracted shows using VADER sentiment analysis method. The proposed recommendation system will make suggestions based on sentiment scores rather than actual movie ratings.

DATASET

Our project idea is to build a recommendation system based on twitter data. There are several twitter users who express their interest about Netflix shows, post reviews through their tweets. We have analyzed such tweets to extract their interest and build a collaborative recommendation system using NLP and Deep learning model which suggest shows of the user's interests. To achieve this, we have collected tweets using Twitter API with different search keywords related to Netflix show interests. In our project, the dataset is created from social media data (Twitter) using REST api to collect data. We have used different keywords for each genre and collected directly into MongoDB. The data contains four different collections for each genre **(Comedy, Romance, Horror and Action)** and around 160000 records in total from each genre of 40000 records. The data comprises of twitter data such as **userid, username, text, created, retweet_count, source, user language and user location**. We also collected rating records from IMDB using web scraping of each shows extracted from tweets. In addition, we have collected ratings and show list of all current Netflix shows for using exact names of the shows as Twitter has unstructured data. Each user in our data has at least one tweet about a show or movie on Netflix. There is a total of 3365 users in 160000 records. Users who have not tweeted at least a show / movie are removed.

File format details of all data in the dataset

Data	Format
Tweets (160000)	code
Users (3365)	code
IMDB Rating (1-10)	code
Netflix shows	csv

DATA AND MAPPING SHOW NAMES

We extracted twitter data with a goal of collecting tweets which is related to Netflix shows. To get a sensible data from twitter and collect the Netflix show names from tweets, we used different search keywords such as “best comedy Netflix”, “best action Netflix”, “best thriller Netflix” etc.

```
for tweet in tweepy.Cursor(api.search,
                           q= "netflix comedy",
                           since=date_since,
                           #until="2020-07-31",
                           count=1000).items(5000):
```

The data gathered from the twitter looks like below,

```
['Austin', datetime.datetime(2021, 1, 6, 3, 7, 56), 'I highly recommend watching Derry girls on Netflix if you're looking for
a new comedy show! https://t.co/j3keFvt02g']
['Amanda', datetime.datetime(2021, 1, 6, 0, 25, 45), 'Everyone freaking out about the office leaving Netflix as if Comedy Cen
tral doesn't show it at least 3 times a week']
```

As we did not acquire significant number of tweets with this, we collected tweets by giving show names as keywords as shown below,

```
romance_title = romance_data["title"]
romance_title_list = romance_title.values.tolist()
romance_title_list
```

```
['6 Years',
 'Castle of Stars',
 'One Day',
 'No Tomorrow',
 'The World We Make',
```

```
for x in romance_title_list:
    for tweet in tweepy.Cursor(api.search,
                               q=x,
                               since=date_since,
                               #until="2020-07-31",
                               count=1000).items(5000):
```

Our next step was to clean the tweets and extract show names from them after which we can get the sentiment score for each tweet. To extract titles/shows, we used regex to extract words which are within quotes, starts with Capital letter and hashtags as we could see most tweets had show names mentioned in these formats. Once such words were extracted, for show mapping we used string matching and fuzzy methods. We did a full string matching where we could get an exact match between extracted words and actual show titles (Collected from Kaggle dataset). For words which did not pass an exact match test, we did a partial string-matching using Fuzzy string match. Fuzzy logic outputs a value (Lower the score, higher the match) based on how close two strings are. After trying few threshold values, we set a threshold score of 10 with which we got most similar matches and mapped actual show titles with score < 10 and selected only those extracted titles/shows.

Since we have used two methods for show extraction, we did not exclude any potential good data but fuzzy string matching which is a partial matching method, there are chances that the tweets can be unrelated to the matched names for few cases. We included such data so that we will have adequate amount of data to perform rest of the project and most importantly build a deep learning model.

As far as the subjectivity of the tweets are concerned, after processing the data using NLP, filtering it using Regex and cleaning it, we removed bad data, but we could not eliminate all bad data from our dataset. We had to compromise the subjectivity and include such data since we are predicting sentiment of the user item interaction which is mainly based on user's reaction towards a certain topic.

One example of the tweets with show title has shown below,

user_id	title	text	Genre	show_id	created	pol_sub	text_clean
202480252	Hush	HUSH (2016) REVIEW! on Netflix \n-WOOOW this is an AMAZING thriller/realistic horror movie 😱\n-use headphones to real… https://t.co/gt3OdclWi1	Horror	80091879	2020-10-14 04:27:14	(0.6000000000000001, 0.9)	hush review on netflix wooow this is an amazing thrillerrealistic horror movie use headphones to real

The text column is showing actual tweet on twitter posted by user_id 202480252. Since the text data has Netflix keyword and movie word, we have extracted the show name using Regex and fuzzy logic. In this case we know that the user is talking about the movie 'Hush'. Here, the user is talking about the movie subject only as we extracted the tweets using keywords such as Netflix or movies.

Another example, a tweet:

"The Witcher is fantastic!" → this tweet could be about the show, the novels, or the games.

Here, we can see that there is no clear subjectivity as the data collected by matching a list of show names. But our project simply considered it as a tweet about the show and sentiment analysis was performed, because our main idea was to build a recommendation system based on sentiment score.

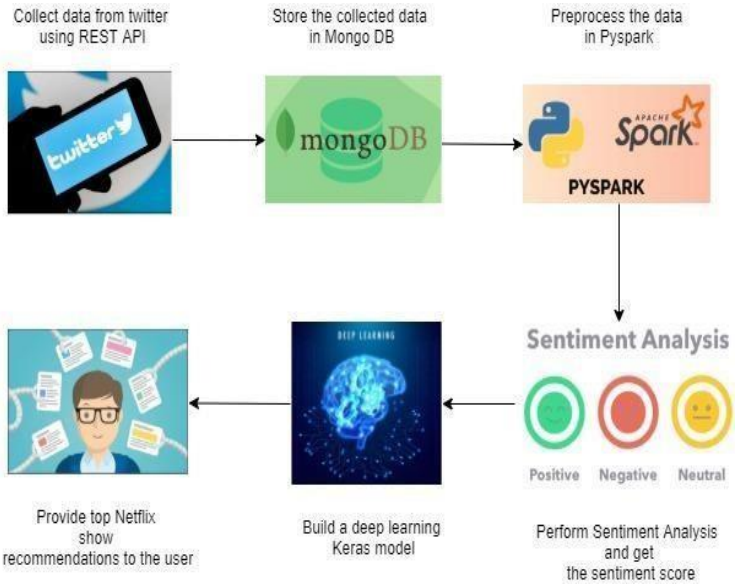
Overall, we tried to build a Netflix show recommender system for twitter users who express their interest about shows through tweets. Since there are above mentioned inconsistencies(partial string matching) involved in the data, as NLP is a vast field, we believe this could be improved by collecting tweets which are more specific about shows. Since Twitter contains many user specific unstructured data, we believe that the veracity of this time series data extracted from tweets can be further analyzed to get better accuracy which we considered as future scope.

PROPOSED SYSTEM

In this project, the applied techniques for building the recommendation system using social media has the below five approaches,

- **Problem Statement:** In this project, we have analyzed and identified a common problem that we see for any ecommerce website. Most of the recommendation system is based on user's implicit feedback i.e. ratings, comments, and view details. But social media platforms are becoming very popular nowadays and helping ecommerce domains to know customer behavior and their taste in order to sell products or to suggest items. We have chosen social media (Twitter) analysis for our project to build a Netflix recommendation system.
  This recommender system uses social media data to recommend movies or shows on Netflix which is based on user's sentiment. Based on our analysis with social media user data, we further characterize the problem and establish the requirement of the Netflix recommender system. Finally, our goal is to recommend movie or shows on Netflix based on Twitter user data and it's interaction with Netflix items (shows/movie) through tweets.

Architecture of Recommender System

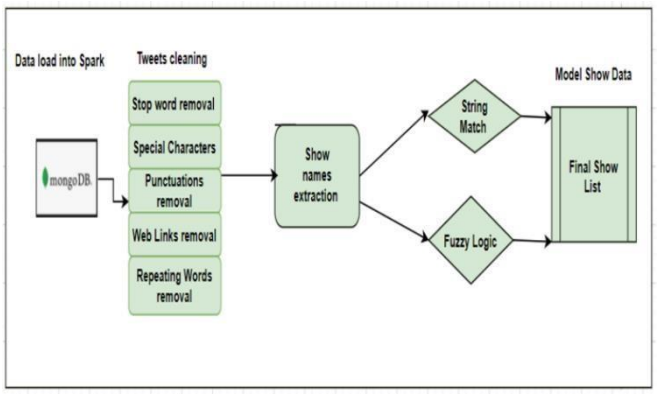


- **Data Gathering**: In this step, we check the design of recommendation system then produce prototype for the recommendation system and gather data based on our requirement. From the previous step, we can tell that our project is based on social media data (text data only), so we have chosen Twitter platform as it is very popular and has various user data with variety of real-time data. We have written a python code using tweepy module which helps to connect to the twitter services. We have used Rest API for data collection. The total data collected is around 160k (40k for each genre). Our python code is making a connection to Twitter API and loading the data directly into MongoDB. The output is a top level json array which consists of the below variables. We have categorized the data into 4 genres (action, horror, comedy, and romance).

Example of json array variable names in the MongoDB document,


```
_id: "5f85e3e28e4d9797365994a4"
user_id: "26843687"
user_name: "M. Gage"
text: "Recasting Avatar The Last Airbender for the Live-Action Netflix Series..."
created: "2020-10-13T17:03:02.000Z"
source: "Twitter for iPhone"
retweet_count: "0"
user_location: ""West Memphis"
user_lang: " AR""
```

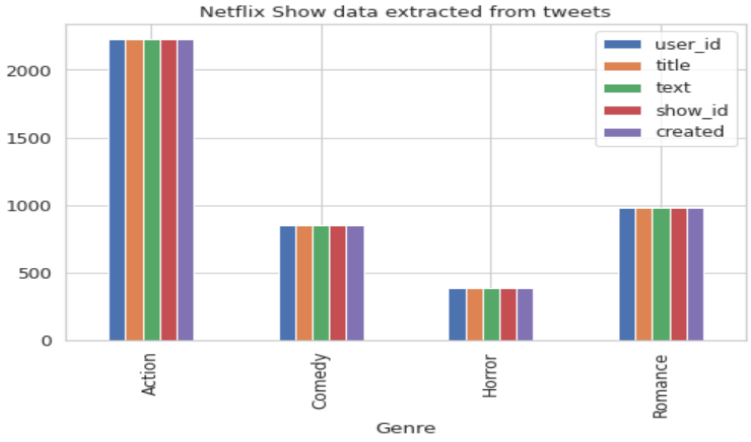

- **Preprocessing**: In our proposed approach, we have used text data from twitter. Since text data is unstructured, we have cleaned the data and removed all uninformative data from thedataset.



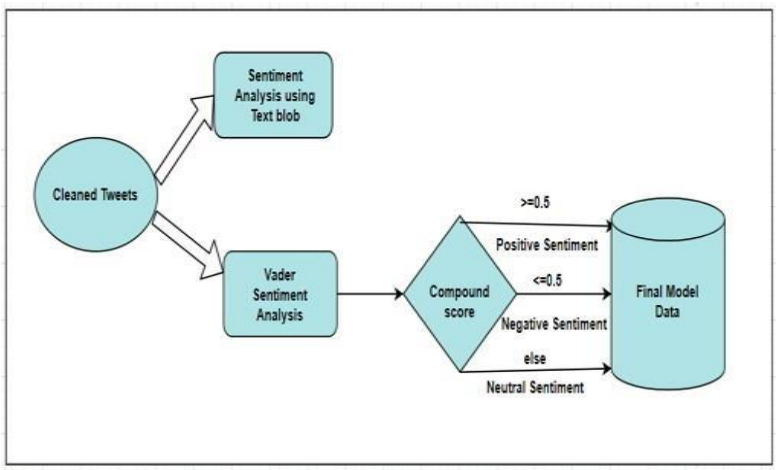
This is the diagram of our preprocessing method. We have cleaned the tweets by removing stop words, special characters, punctuations, web links and repeating words. These are typically the noise in the text data which we do not require in our data analysis. After cleaning the data, we extracted the Netflix show names from tweets. To match the tweeted Movie or show names with the actual Netflix show names, we have applied two methods a. String Matching, b. Levenshtein Fuzzy Logic.

String matching is a simple matching of two strings and taking all matched show names into a dataframe. Levenshtein fuzzy logic is a similarity matching method which compares two words based on the distance between them. Levenshtein distance is measured by the minimum number of edits required to change one string into the other one to match both the strings. After matching using both the logic, we have used that original show and movie lists as our model input data for the proposed recommendation system.

Details of the extracted shows from tweets after preprocessing is shown below,

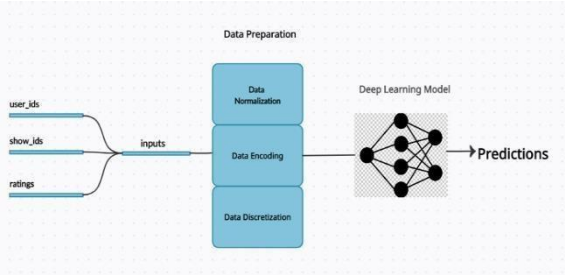


- **Sentiment Analysis**: After preprocessing the data, we have done sentiment analysis of the tweets. Sentiment analysis is a process by which we try to analyse the underlying emotion of the text posted by the user. Users tweets have been analyzed using two methods as shown in the diagram,

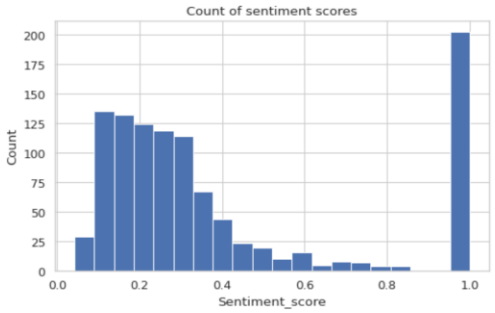


The text blob sentiment method has two components (Polarity, Subjectivity). It is an NLP process, which analyse the text data and assign scores to rate it positive or negative sentiment. The polarity value lies between -1 and +1, where polarity of -1 means negative sentiment and that of 1 means positive sentiment. Subjectivity checks if the text data is a public or factual statement. Subjectivity score lies between [0,1], where higher value means public opinion and lower value means factual statement. We have not considered this score for our recommendation system since we must take in account two scores for deciding the sentiment and subjectivity. The other method is VADER (Valence Aware Dictionary and Sentiment Reasoner) sentiment analysis which is a dictionary-based approach that maps each word with it's emotion by building a lexicon. In this process, the subjectivity of the text data is handled by the sentiment process itself. This sentiment analysis has four components i.e. positive, negative, neutral, and compound score. Compound score value is between [-1,1], -1 means least preferred and +1 means most preferred. For VADER sentiment if the compound score is greater or equal to 0.05 then text data sentiment is positive and if the compound score is less than or equal to -0.05 then it's a negative sentiment, it's a neutral sentiment if the sentiment score is neither of the above. For our recommendation system, we used VADER Sentiment score for the model data.

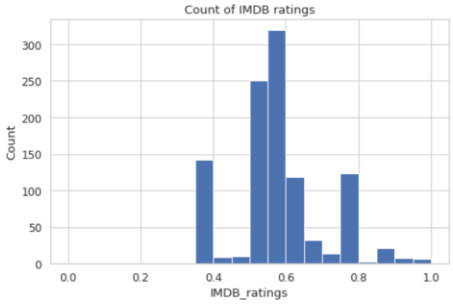
- **Model Building**: We have chosen deep learning for building the model. Before building the model, we have normalized our data and prepared it for feeding to themodel.



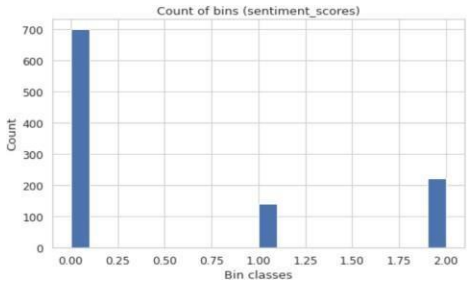
After preprocessing and sentiment analysis, we have scaled, encoded, and discretized the data into 3 classes to remove data fluctuations. After fine tuning thedata, we have built a deep learning model for the proposed recommendation system.



Fig(a)



Fig(b)



Fig(c)

Fig(a): Sentiment score distribution after normalization.

Fig(b): IMDB rating after scaling the data.

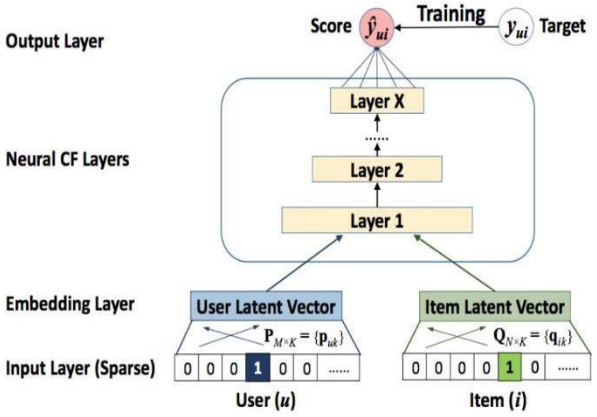
Fig(c): Sentiment classification after discretization of the training data

In this step, we are trying to predict the sentiment score for twitter users which is a regression framework, where the target value is sentiment score, and dependent variables are user ids, show ids, ratings. We used neural collaborative filtering technique with two models for Netflix show/Movie recommendations-

a. Single layer neural network Model

b. Multi-layer perceptron Model.

A typical neural collaborative filtering has the below architecture:



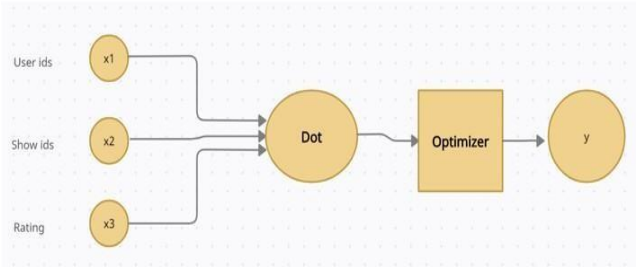
Input layer: It is a binary input based on user and item(show/Movie) interaction. The value being 1 means the user and item has positive interaction.

Embedding layer: It is used to vectorize the inputs before feeding it to the dense layers. Embedding layer transforms the inputs into a dense vector.

Neural CF layers: A set of layers to map the latent input vectors to its prediction score.

Output layer: The output layers give the predicted score which is the target value by minimizing the loss.

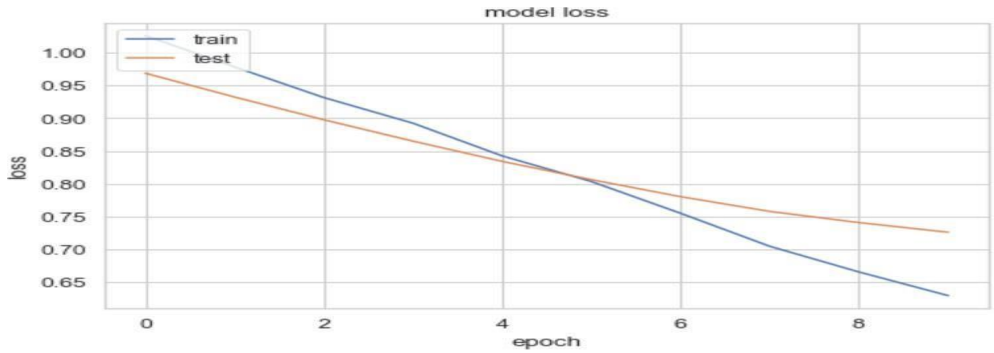
Single layer deep learning model



This is the architecture of single layer model that we used to build a collaborative recommendation system. We have given embedded latent inputs. We are feeding these dense vector inputs to the dot function where all the inputs are getting multiplied and then optimised by **Adam** optimiser, where Adam(adaptive moment estimation) adapts the learning rate of each parameters by small updates for frequent parameters and large updates for infrequent parameter. This whole process helps to regulate the training of the model and preventing abrupt changes in the model training. After training the data, we got below performance of the model which shows accuracy improvement after each epoch (with 2000 input data).

```
Epoch 1/10
14/14 [==============================] - 0s 28ms/step - loss: 1.0261 - accuracy: 0.6269 - val_loss: 0.9687 - val_accuracy:
0.6518
Epoch 2/10
14/14 [==============================] - 0s 5ms/step - loss: 0.9783 - accuracy: 0.6269 - val_loss: 0.9325 - val_accuracy: 0.
6518
Epoch 3/10
14/14 [==============================] - 0s 6ms/step - loss: 0.9317 - accuracy: 0.6269 - val_loss: 0.8978 - val_accuracy: 0.
6518
Epoch 4/10
14/14 [==============================] - 0s 5ms/step - loss: 0.8926 - accuracy: 0.6269 - val_loss: 0.8652 - val_accuracy: 0.
6518
Epoch 5/10
14/14 [==============================] - 0s 4ms/step - loss: 0.8427 - accuracy: 0.6269 - val_loss: 0.8344 - val_accuracy: 0.
6518
Epoch 6/10
14/14 [==============================] - 0s 6ms/step - loss: 0.8036 - accuracy: 0.6269 - val_loss: 0.8065 - val_accuracy: 0.
6518
Epoch 7/10
14/14 [==============================] - 0s 5ms/step - loss: 0.7552 - accuracy: 0.6269 - val_loss: 0.7806 - val_accuracy: 0.
6518
Epoch 8/10
14/14 [==============================] - 0s 4ms/step - loss: 0.7049 - accuracy: 0.6448 - val_loss: 0.7580 - val_accuracy: 0.
6518
Epoch 9/10
14/14 [==============================] - 0s 4ms/step - loss: 0.6657 - accuracy: 0.6716 - val_loss: 0.7410 - val_accuracy: 0.
6518
Epoch 10/10
14/14 [==============================] - 0s 5ms/step - loss: 0.6297 - accuracy: 0.6761 - val_loss: 0.7263 - val_accuracy: 0.
6205
```

The Single layer model evaluation looks like below,

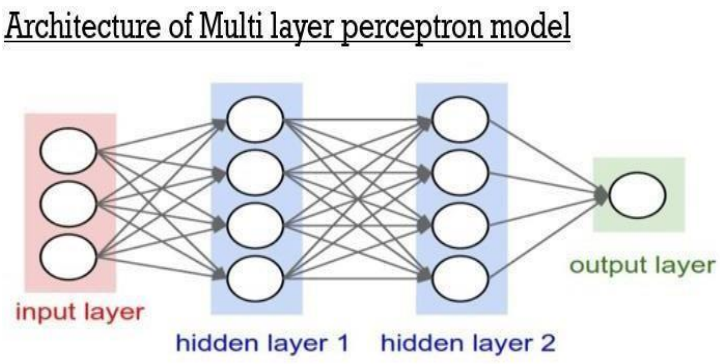


But with the increase in data (4000 data for each input) to the model, the performance of the single layer model decreases, and the mean squared error is very high as shown below,

```
history_1 = model_1.fit(x=X_train_array, y= data_trans , batch_size=50, epochs=10,
                    verbose=1, validation_data=(X_test_array, data_trans_test))

Epoch 1/10
15/15 [==============================] - 2s 57ms/step - loss: 1.1327 - mse: 1.1327 - accuracy: 0.6099 - val_loss: 0.8179 - val_mse: 0.8179
Epoch 2/10
15/15 [==============================] - 0s 5ms/step - loss: 1.0606 - mse: 1.0606 - accuracy: 0.6096 - val_loss: 0.8048 - val_mse: 0.8048
Epoch 3/10
15/15 [==============================] - 0s 5ms/step - loss: 1.0343 - mse: 1.0343 - accuracy: 0.6086 - val_loss: 0.7921 - val_mse: 0.7921
Epoch 4/10
15/15 [==============================] - 0s 4ms/step - loss: 0.9430 - mse: 0.9430 - accuracy: 0.6500 - val_loss: 0.7801 - val_mse: 0.7801
Epoch 5/10
15/15 [==============================] - 0s 4ms/step - loss: 0.9675 - mse: 0.9675 - accuracy: 0.6283 - val_loss: 0.7684 - val_mse: 0.7684
Epoch 6/10
15/15 [==============================] - 0s 4ms/step - loss: 0.9504 - mse: 0.9504 - accuracy: 0.6283 - val_loss: 0.7573 - val_mse: 0.7573
Epoch 7/10
15/15 [==============================] - 0s 5ms/step - loss: 0.9582 - mse: 0.9582 - accuracy: 0.6281 - val_loss: 0.7467 - val_mse: 0.7467
Epoch 8/10
15/15 [==============================] - 0s 5ms/step - loss: 0.9588 - mse: 0.9588 - accuracy: 0.6234 - val_loss: 0.7365 - val_mse: 0.7365
Epoch 9/10
15/15 [==============================] - 0s 5ms/step - loss: 0.9202 - mse: 0.9202 - accuracy: 0.6270 - val_loss: 0.7272 - val_mse: 0.7272
Epoch 10/10
15/15 [==============================] - 0s 5ms/step - loss: 0.8878 - mse: 0.8878 - accuracy: 0.6279 - val_loss: 0.7184 - val_mse: 0.7184
```

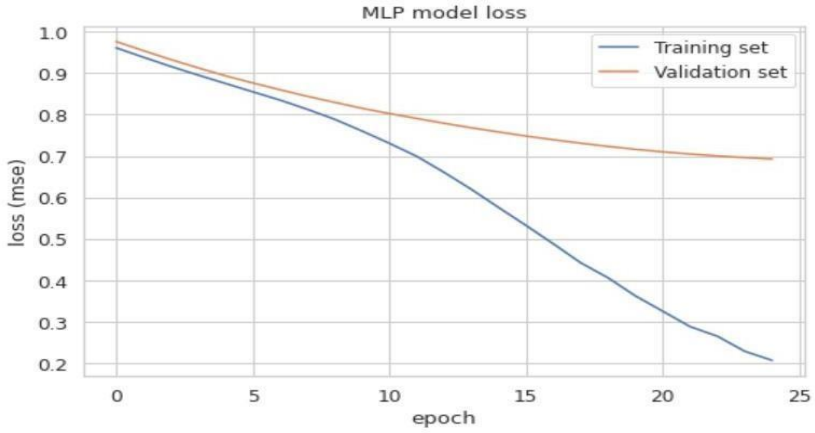
From the above evaluation, we can see that the training loss is higher than the testing loss and is not suitable for prediction. To improve the performance the model, we have added more parameters such as more layers, activation function and drop out layers to our model which improves the model performance. Next, we have used multi-layer perceptron model to improve our recommendation model. The details of multilayer model and its architecture is shown below.



This architecture takes user and item embedded inputs. It uses multiple neural layers to layer nonlinear interactions. We have used twohidden layers and each layer has 4 neurons; we have chosen 4 neurons as it is fitting well with our input data. One can take many neurons depending on the data requirement in the model. MLP model takes latent vector inputs and feed it to the dense layer. The hidden layers have activation function and optimizers.

Activation function adds nonlinearity to the model, it learns the user-item interaction and endowed with a lot of flexibility to learn more from the data. Optimizers improves the performance of each parameters in the training data. The output layers return the target value based on the element wise latent vector product of user item vectors.

After training the data, we got the below performance characteristics of the model,



By definition, a model works best when the validation loss is a little higher than the training loss. In this case, our model is working fine as there is no overfitting or underfitting of the data as we can see from the evaluation graph that the validation loss is a little higher than the training loss and gradually after each epoch the model is performing well with the validation data. The accuracy of the model has been improved with comparison to single layer model as seen below,

```
Epoch 1/5
20/20 [==============================] - 0s 6ms/step - loss: 0.2086 - mse: 0.2086 - accuracy: 0.7956
Epoch 2/5
20/20 [==============================] - 0s 4ms/step - loss: 0.1924 - mse: 0.1924 - accuracy: 0.7946
Epoch 3/5
20/20 [==============================] - 0s 4ms/step - loss: 0.1682 - mse: 0.1682 - accuracy: 0.7956
Epoch 4/5
20/20 [==============================] - 0s 4ms/step - loss: 0.1429 - mse: 0.1429 - accuracy: 0.7956
Epoch 5/5
20/20 [==============================] - 0s 4ms/step - loss: 0.1325 - mse: 0.1325 - accuracy: 0.7956
```

RESULTS

Based on the above model evaluation data, we used MLP model for our recommendation system. To check the prediction of the model, we passed user details to the model and based on the user-item interaction the model predicts the sentiment score for the list of movies. We took top 3 movies/shows score for the recommendation.

Lets first check how the user data looks like after show name extraction from Twitter and cleaning its tweet,

user_id	title	text	Genre	show_id	created	pol_sub	text_clean	Sentiment_score
8431998005248	Moonwalkers	Moonwalkers on July 24th vs July 25th https://t.co/opu2yBkhXR	Action	80046728	2020-10-06 15:16:21	(0.0, 0.0)	moonwalkers on july th vs july th	1.0

From the above data we can see that user id 8431998005248 tweeted about Moonwalkers movie release date and expressed interests about this movie. The corresponding sentiment score for this movie of this user is 1 i.e. positive interaction. We are feeding this data to our model and after training the data, the recommendation model is predicting the suitable movies for this user,

```
#Recommendation based on prediction

df_model_data1.iloc[pred_ids]
```

	show_id	title	IMDB_rating
11	70242081	Arrow	7.5
45	80223052	Polar	6.3
56	80227516	Chosen	7.4

After checking the user-item interaction with each movie and its interaction with this user has been checked by the model. Based on the given prediction results, we havetaken the top 3 predictions for recommendation. The recommendation is done by measuring every movie or show items interaction and its corresponding sentiment value for this user in our dataset. Based on social media data analysis, our model suggests that there is 79% chance that the user will pick at least one of the recommended movies.

Another example for the below user (Top 8 recommendation)

user_id	title	text	Genre	show_id	created	pol_sub	text_clean
202480252	Hush	HUSH (2016) REVIEW! on Netflix \n-WOOOW this is an AMAZING thriller/realistic horror movie 😱\n-use headphones to real… https://t.co/gt3OdclWi1	Horror	80091879	2020-10-14 04:27:14	(0.6000000000000001, 0.9)	hush review on netflix wooow this is an amazing thrillerrealistic horror movie use headphones to real

The user has tweeted about **Hush** movie and the sentiment score for the user is 0.651,

show_id	title	user_id	Genre	Sentiment_score	IMDB_rating	genre_encoded
80091879	Hush	202480252	Horror	0.651	6.6	2

We can see that the user has watched horror genre and shown positive interactions on Twitter. After using deep learning and NLP, MLP model predicted the below recommended movies for this user,

show_id	title	IMDB_rating	genre_encoded	Sentiment_score
80091879	Hush	6.6	2	0.651
80158104	Taken	7.8	0	1.000
60028294	Popeye	5.3	0	1.000
70001082	Run	8.4	0	1.000
81087094	Lifechanger	5.4	2	0.321
81168340	Medium	6.9	2	1.000
80207371	Close	5.7	0	1.000
81021831	Sabrina	6.6	2	0.139

From the above results, we can see that the model is showing 4 horror(genre encoded value : 2) movies (Hush, Life changer, Medium and Sabrina) and 4 action (genre encoded value: 0) movies(Taken, Popeye, close and Run) on Netflix.

Our proposed approach is based on regression problem, so to check the prediction results and its functionality testing we have done offline evaluation testing. For our deep learning regression problem, the below testing works well,

1. **Train-Test Split test** 2. **K-Fold Cross-Validation**

In the training and testing split, we validated the model by checking the performance of test and training data. Accuracies depends based on the actual user's data that the model has already seen. The test data generates an output which shows the ranking for the user and the prediction for the movie list given to the model.

The below data shows the prediction of actual sentiment score of users - movie interactions and recommendation prediction score,

```
# Checking the actual sentiment score and the predicted score data using MLP model

X_test_df
```

	show_Id	user_Id	actual_ss	r_predictions
0	43.0	448.0	0.097	0.353454
1	2.0	629.0	0.158	0.416772
2	68.0	1141.0	0.268	0.355421
3	68.0	89.0	0.423	0.363802
4	2.0	431.0	0.179	0.435067

Now we have the predicted data and actual data. After that, we have checked the error or loss in each prediction by using the below metrics and measures. Since we do not have access to collect user behavior data and scoring our Netflix recommendation system on Twitter. We have done the below offline evaluation measures to check the evaluation of the modelresults,

1. MSE 2. MAE 3. RMSE

```
##Evaluation using MSE, MAE and RMSE
from sklearn import metrics

print("MSE: ", metrics.mean_squared_error(X_test_df.actual_ss, X_test_df.r_predictions))
print("MAE: ", metrics.mean_absolute_error(X_test_df.actual_ss, X_test_df.r_predictions))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(X_test_df.actual_ss, X_test_df.r_predictions)))
```

```
MSE:  0.10506618873807119
MAE:  0.2590484098498251
RMSE:  0.3241391502704837
```

From the above result we can see that our model predicted values are more similar to the actual values. The error rate is very low for 3 metrics. Mean squared error value on test data is 0.105, Mean absolute error is 0.259 and root mean squared error value is 0.324. Since our deep learning model is based on a regression problem, these offline tests based on the user's historical data works well to test the prediction results. However, we have further tested the result using cross validation. Since cross validation process is computationally expensive, we have taken a sample of our model data to test the model response. For each iteration, we have seen that the root mean squared value has decreased.

CONCLUSION

Recommendation systems are important part of daily life. Scientists and researchers are exploring more on improving accuracy and to create faster models. In this report, we proposed a Netflix show recommender system which analyses twitter data to make suggestions based on sentiment score of tweets. We believe, as a tweet represent short and crisp idea of a user interest, analysing its sentiment would provide precise results about user likings and emotions towards a show and this could be used to create recommendations which are actually relevant. We used two deep learning models – a Single layer model and a multilayer perceptron model. Both the models gave comparable results with a small dataset, but multilayer model was more accurate when a large dataset was used. There is enough scope for research for this particular recommender system. More information about users and shows can be accommodated to improve the efficiency. As of now we created a static system with Netflix shows till 2019. A new database could be created for storing Netflix shows and this can be updated on a regular basis with new shows and movies to make this dynamic. Accuracy of extracted show names can be improved by refining search keywords used to collect twitter data.

REFERENCES

1. P.K. Singh, P.K.D. Pramanik, A. Dey, P. Choudhury. (2019) 'Recommender Systems: An Overview, Research Trends and Future Directions', Int. J. of Business and Systems Research, Vol. X, No. Y, pp. xxx–xxx
2. J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen, Collaborative Filtering Recommender Systems.
3. Pasquale Lops, Marco de Gemmis and Giovanni Semeraro, Chapter 3 Content-based Recommender Systems: State of the Art and Trends, F. Ricci et al. (eds.), Recommender Systems Handbook.
4. Burke, R. Hybrid Recommender Systems: Survey and Experiments. User Model User-Adap Inter 12, 331–370 (2002). https://doi.org/10.1023/A:1021240730564
5. Oyebode, O., Orji, R. A hybrid recommender system for product sales in a banking environment. J BANK FINANC TECHNOL 4, 15–25 (2020). https://doi.org/10.1007/s42786-019-00014-w.
6. Karatzoglou, Alexandros., Hidasi, B., Deep Learning for Recommender Systems, RecSys '17: Proceedings of the Eleventh ACM Conference on Recommender SystemsAugust 2017 Pages 396–397https://doi.org/10.1145/3109859.3109933.
7. J. Lund and Y. Ng, "Movie Recommendations Using the Deep Learning Approach," 2018 IEEE International Conference on Information Reuse and Integration (IRI), Salt Lake City, UT, 2018, pp. 47-54, doi: 10.1109/IRI.2018.00015.
8. Ziani, Amel & Azizi, Nabiha & Schwab, Didier & Aldwairi, Monther & Chekkai, Nassira & Zenakhra, Djamel & Cheriguene, Soraya. (2017). Recommender System Through Sentiment Analysis.
9. Yibo Wang, Mingming Wang, Wei Xu, "A Sentiment-Enhanced Hybrid Recommender System for Movie Recommendation: A Big Data Analytics Framework", Wireless Communications and Mobile Computing, vol. 2018, Article ID 8263704, 9 pages, 2018. https://doi.org/10.1155/2018/8263704