Movie Recommendation System using Neural Network Embeddings

Harpreet Virk
CS - 2490 - Advanced Machine Learning
Prof. Ravi Kothari
May 5, 2020

Abstract— **In this project, we implement a system that generates movie recommendations by predicting the likability of a movie by a user based on the response of other similar users, where this similarity is calculated based on the movie ratings left by other users. Our system is based on Collaborative Filtering (CF) with a modification, where we embed our users and movies into a low dimensional space to learn these similarities among users and movies.**

I. Introduction

Recommendation systems are all around us. Since the advent of Machine Learning, scientists have been studying the machines ability to learn about the preferences of individual users in order to predict their future behaviour. From *Amazon* to *Facebook* and *Netflix*, learning about user preferences for customising user experience has always been of prime concern to drive continuous user engagement statistics.

Recommendation systems use the user, item, and ratings data to predict the likability of a particular item by other users. Several recommendation system designs have been proposed over the recent years. These systems can generally be categorised into two main categories - content based recommending systems, and collaborative filtering recommendation systems.

Content based recommendation systems rely on item features such as keywords, descriptions, genres and user reviews to make recommendations, but they can also depend upon other characteristics including demographics and user preferences. However, they are limited by their ability to recommend only those items that have similar features to the queried item. This can often limit the scope of their recommendations, which can result in surfacing items with low ratings or relevance to the user.

Collaborative Filtering (CF) generates user recommendations by taking advantage of the collective knowledge from all users (Jannach *et al*, 2016). This means that it focuses on finding *item-item* and *user-user* similarities, instead of focusing on the *item-user* similarities like content based recommendation algorithms.

In the context of movie recommendation systems, collaborative filtering works based on the simple assumption that if a user A rates certain movies similar to how another user B rates them, then user A is more likely to have a similar opinion as user B on an unwatched movie, than any randomly chosen user.

In our implementation of a movie recommendation system, we will be using neural network embeddings to represent our movies (and users) data to build a modified version of collaborative filtering. Neural network embeddings map high-dimensional categorical variables (here movies and users) to a low-dimensional learned representation that places similar entities closer together in the embedding space. This allows us to represent each movie (and each user) using a low dimensional vector, where movies (and users) that are more similar in the context of our learning problem are closer to one another in the embedding space.

II. Methodology

A. Dataset

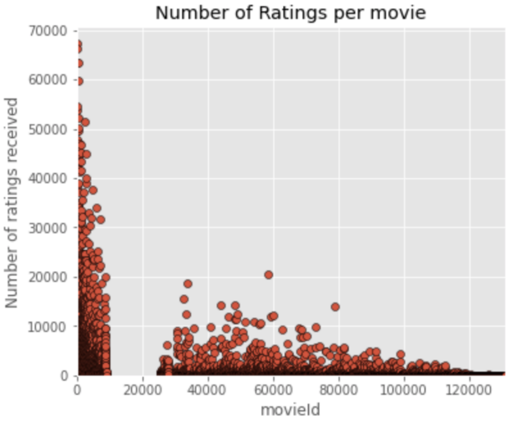
We will be using the MovieLens 20M dataset made available by GroupLens. This dataset contains 20 million ratings and 465, 000 tag applications applied to 27, 000 movies by 138, 000 users.

B. Data Pre-processing

The MovieLens 20M dataset consists of two smaller datasets. The ratings dataset consists of all the ratings for movies by individual users. The movies dataset consists of movie metadata for all movies, namely their movieIds, title, and genres. On reading the dataset using the pandas data-frame, the following statistics are revealed:

- Number of unique users: $138,493$
- Number of unique movies: $26,744$

We then find the count of user ratings that each movie has received, the results of which can be seen with the following visualisation:



We can observe a significant difference between the number of ratings received for different movies. Certain popular movies have received as many as 70,000 ratings. On the other hand, there are also some movies that have not received any rating.

Thus, we will use only those movies that have received more than at least 5,000 user ratings to train our recommendation system. This will help us focus on recommending more popular movies, as well as help us increase our training and prediction speed.

Our revised data statistics are as follows:

- Number of unique users: $138,476$

- Number of unique movies: $1,005$

C. Collaborative Filtering

In traditional forms of collaborating filtering, each item is represented as an m dimensional vector, where m is the number of unique users in the catalogue (Liden *et al*, 2003). The entries in these vectors are the ratings provided by each users. These n item rating vectors are put together in a matrix, with the rows and columns representing each unique item and user respectively (Mojdeh *et al*). We get a ratings matrix of size $n \times m$ as follows, where $None$ denotes a missing entry:

	User 1	User 2	User 3	User 4
Item 1	0.5	$None$	3.0	2.0
Item 2	3.5	5.0	4.0	4.0
Item 3	$None$	$None$	3.0	$None$

In the context of our movie recommendation system, we can observe that our dataset has around 1,000 unique movie items in the catalogue with around 138,000 unique users. This would mean that each movie in our dataset will require a discrete valued vector of size 138,000 to represent its user ratings. In addition to that, because not every user often rates every other movie, these vectors will be substantially sparse due to a large number of missing rating entries.

We will be modifying our approach to collaborative filtering in our model by using neural network embeddings to help us map these high dimensional vectors to a low dimensional learned representation.

Entity embeddings are equivalent to the result of multiplying our rating vectors with a weight matrix $W \in R^{n \times m}$ as follows:

$$embeddings(x) = rating(x) * W$$

where x represents an entity and $ratings(x)$ represents the vector corresponding to that entity.

Weight matrix W is typically randomly initialised. We will be preparing our entity embedding layers by training a neural network to learning the weight matrix W for our movies (and users) data.

Learning the entity embeddings not only helps us get a reduced dimension for representing our items, we also get a representation that keeps similar items closer to each other. We can measure the similarity between two movie embedding vectors A and B by measuring the cosine of the angle between the two embedding vectors (Liden *et al*, 2003) as follows:

$$similarity(\vec{A}, \vec{B}) = \cos(\vec{A}, \vec{B}) = \frac{\vec{A} \bullet \vec{B}}{\|\vec{A}\| * \|\vec{B}\|}$$

Here, we are essentially taking the dot product of two embedding vectors, and then normalising them, to get a measure of similarity between the values 0 and 1, with 1 signifying maximum similarity.

III. Network Design

A. Approach

To learn our embedding layers, we will build an Embedding Neural

Network. We will train our network model using supervised learning with the aim of representing similar movies (and users) with closer representations in the embedding space.

After training, we will have the embedding representations for our movies and users. We will then find movies to recommend by finding similar movies to a given movie. We will be using the cosine distance to measure the similarity between the embedding vector of the input movie, and the embeddings of all other movies.

B. Learning Task

The supervised machine learning task for our network will be to predict the ratings given by a user, for a given movie. The input data (x) will consist of (userId, movieId) pairs, and our model will be predicting the ratings as the output data (y).

The network will try to predict the user rating by adjusting the embedded vector representations of that movie and user. Hence during training, our goal will be to minimise the loss in the user ratings prediction by adjusting the weight matrix W for our embedding layers.

We need to note that although we are designing our network for a supervised learning task, our end goal is not to make accurate rating predictions, but to learn the perfect entity embeddings. We will not be testing our model on new data after training. Instead, we will be using the embedding layers themselves to learn similarities between entities. Hence, we will not be creating a separate testing (or validation) set, as our prediction task is just a method by which we train our model to make the embeddings.

C. Model Architecture

The layers of our model are as follows:

1) Input Layers: We have two parallel input layers, one for users and the other for movies.

2) Embedding Layers: We have two parallel embedding layers, one for users and the other for movies. We will limit the size of the embedding vector to 50 dimensions to ensure that our embeddings are dense, and have continuous values.

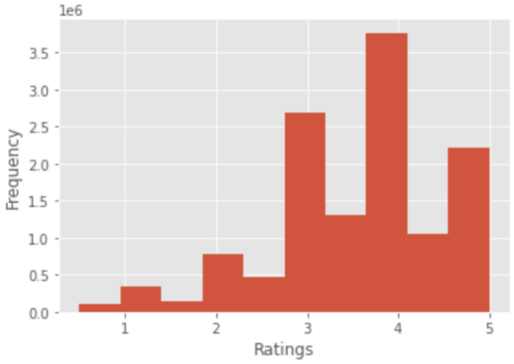
Embedding values depend upon weight matrix W which is typically randomly initialised. However, we will be initialising our W with the he_normal initialiser (He *et al*, 2015), which draws samples from a truncated normal distribution centred on 0, with standard deviation conditional upon the size of the previous layer vector. To avoid overfitting, we also regularise our embedding layers with l2 $ridge\ regression$ technique.

3) Dot Product Layer: Once we have the embedding layers for our movies and users, we need a way to merge them to get a single dimensional output. We then take a dot product of our user and movie embedding, which sums the element wise multiplication of the embedding vectors to result into a single digit number. This number can then be scaled to get the output of our rating prediction.

4) Bias Layer: Now, let's observe the frequency of each rating value in our dataset:

```
Frequency of each rating from 0.5 to 5.0:

Text(0.5, 0, 'Ratings')
```



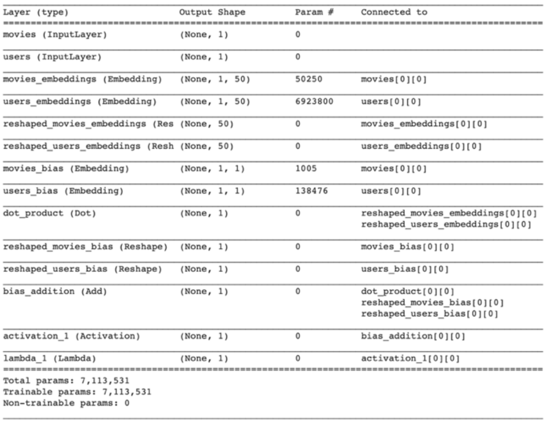
We can observe that most users rate movies in the range of 3.0 to 5.0, with 4.0 being the most common rating value. This can be attributed to several factors. Certain movies are considered good universally, and are highly rated by most users. Similarly, some users are more critical in rating their movies, while a majority are not who instinctively rate movies with popular rating values. To ensure that these inherent biases in rating practices do not affect our prediction model, we will now introduce a bias embedding layer for our movies (and users).

The bias layers for our users and movies are embedding layers with single dimension. We then account for this bias in our model by adding the output of our bias layers to the output of our dot product layer.

5) Activation Layer: We use a sigmoid activation function to map the single dimensional output of our Bias layer to values between 0 and 1. This also introduces non-linearity in our model.

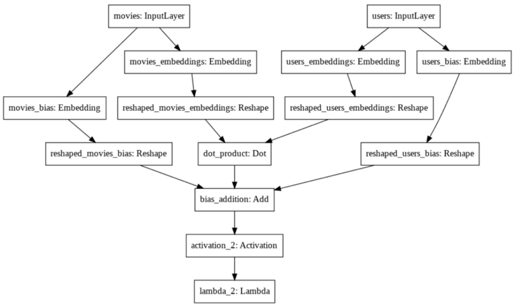
6) Lambda Scaling Layer: Now, since our user rating values range between 0.5 to 5.0, we use a scaling layer to scale the output of our sigmoid activation layer to values between the rating range [0.5, 5.0].

A summary of our model architecture is as follows:



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
movies (InputLayer)              (None, 1)             0
____________________________________________________________________________________________________
users (InputLayer)               (None, 1)             0
____________________________________________________________________________________________________
movies_embeddings (Embedding)    (None, 1, 50)         50250       movies[0][0]
____________________________________________________________________________________________________
users_embeddings (Embedding)     (None, 1, 50)         6923800     users[0][0]
____________________________________________________________________________________________________
reshaped_movies_embeddings (Res (None, 50)             0           movies_embeddings[0][0]
____________________________________________________________________________________________________
reshaped_users_embeddings (Resh (None, 50)             0           users_embeddings[0][0]
____________________________________________________________________________________________________
movies_bias (Embedding)          (None, 1, 1)          1005        movies[0][0]
____________________________________________________________________________________________________
users_bias (Embedding)           (None, 1, 1)          138476      users[0][0]
____________________________________________________________________________________________________
dot_product (Dot)                (None, 1)             0           reshaped_movies_embeddings[0][0]
                                                                   reshaped_users_embeddings[0][0]
____________________________________________________________________________________________________
reshaped_movies_bias (Reshape)   (None, 1)             0           movies_bias[0][0]
____________________________________________________________________________________________________
reshaped_users_bias (Reshape)    (None, 1)             0           users_bias[0][0]
____________________________________________________________________________________________________
bias_addition (Add)              (None, 1)             0           dot_product[0][0]
                                                                   reshaped_movies_bias[0][0]
                                                                   reshaped_users_bias[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 1)             0           bias_addition[0][0]
____________________________________________________________________________________________________
lambda_1 (Lambda)                (None, 1)             0           activation_1[0][0]
====================================================================================================
Total params: 7,113,531
Trainable params: 7,113,531
Non-trainable params: 0
____________________________________________________________________________________________________
```

We can use the following diagram to understand how the model processes input from one layer to next:


movies: InputLayer
users: InputLayer
movies_embeddings: Embedding
users_embeddings: Embedding
users_bias: Embedding
movies_bias: Embedding
reshaped_movies_embeddings: Reshape
reshaped_users_embeddings: Reshape
reshaped_movies_bias: Reshape
dot_product: Dot
reshaped_users_bias: Reshape
bias_addition: Add
activation_2: Activation
lambda_2: Lambda

D. Model Compilation

We compile our model using the mean squared error loss function. We use Adam optimiser with a learning rate of 0.001 to update our weight parameters after calculating the gradients.

E. Initialising and Training

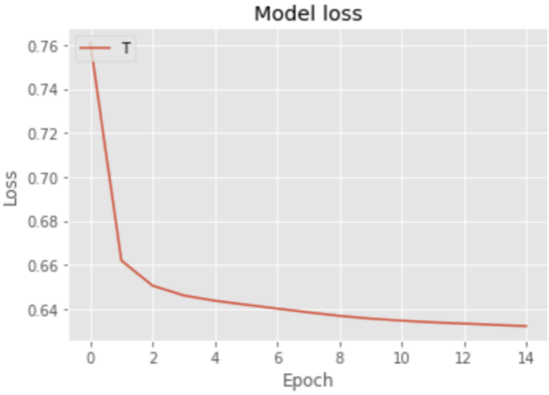
We trained our models on Google Colab with the following system configurations enabled:

- Intel(R) Xeon(R) CPU @ 2.30GHz
- 13 GB RAM

The following coding dependencies have been used:

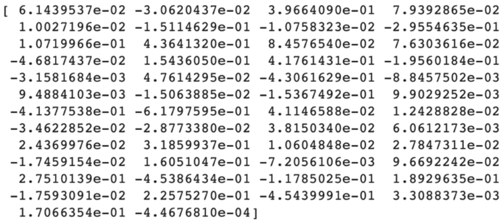
- Jupyter
- NumPy
- Keras
- TensorFlow
- Matplotlib
- Pandas
- TSNE
- SKLearn

We train our model for 15 epochs, and measure the loss reduce as follows:



F. Model Output

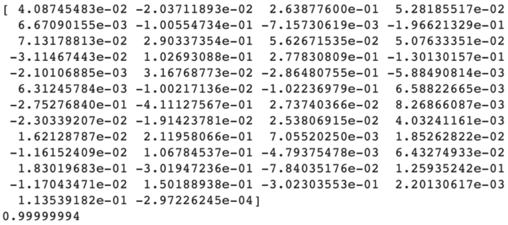
The resultant movie embedding layer that we get, after the model finishes learning, for a randomly chosen movie is as follows:



```
[ 6.1439537e-02 -3.0620437e-02  3.9664090e-01  7.9392865e-02
  1.0027196e-02 -1.5114629e-01 -1.0758323e-02 -2.9554635e-01
  1.0719966e-01  4.3641320e-01  8.4576540e-02  7.6303616e-02
 -4.6817437e-02  1.5436050e-01  4.1761431e-01 -1.9560184e-01
 -3.1581684e-03  4.7614295e-02 -4.3061629e-01 -8.8457502e-03
  9.4884103e-03 -1.5063885e-02 -1.5367492e-01  9.9029252e-03
 -4.1377538e-01 -6.1797595e-01  4.1146588e-02  1.2428828e-02
 -3.4622852e-02 -2.8773380e-02  3.8150340e-02  6.0612173e-03
  2.4369976e-02  3.1859937e-01  1.0604848e-02  2.7847311e-02
 -1.7459154e-02  1.6051047e-01 -7.2056106e-03  9.6692242e-02
  2.7510139e-01 -4.5386434e-01 -1.1785025e-01  1.8929635e-01
 -1.7593091e-02  2.2575270e-01 -4.5439991e-01  3.3088373e-03
  1.7066354e-01 -4.4676810e-04]
```

We can see that each movie (and user) is now represented as a 50 dimensional embedding vector. Now, we need to normalise these embeddings to ensure that the dot product between any two embedding vectors gives us the cosine similarity.

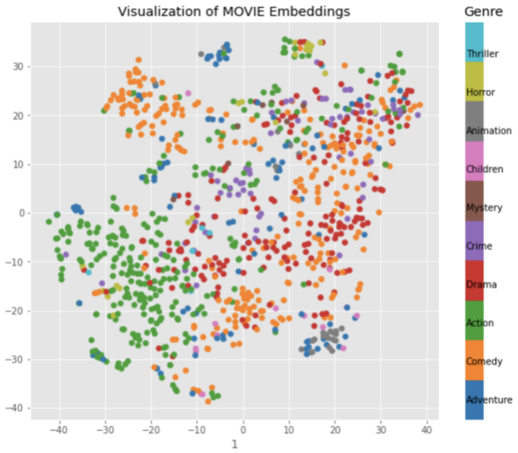
We do this normalisation by dividing each vector by the square root of the sum of all of its squared components. Our resultant embedding layer for the same movie is as follows:



```
[ 4.08745483e-02 -2.03711893e-02  2.63877600e-01  5.28185517e-02
  6.67090155e-03 -1.00554734e-01 -7.15730619e-03 -1.96621329e-01
  7.13178813e-02  2.90337354e-01  5.62671535e-02  5.07633351e-02
 -3.11467443e-02  1.02693088e-01  2.77830809e-01 -1.30130157e-01
 -2.10106885e-03  3.16768773e-02 -2.86480755e-01 -5.88490814e-03
  6.31245784e-03 -1.00217136e-02 -1.02236979e-01  6.58822665e-03
 -2.75276840e-01 -4.11127567e-01  2.73740366e-02  8.26866087e-03
 -2.30339207e-02 -1.91423781e-02  2.53806915e-02  4.03241161e-03
  1.62128787e-02  2.11958066e-01  7.05520250e-03  1.85262822e-02
 -1.16152409e-02  1.06784537e-01 -4.79375478e-03  6.43274933e-02
  1.83019683e-01 -3.01947236e-01 -7.84035176e-02  1.25935242e-01
 -1.17043471e-02  1.50188938e-01 -3.02303553e-01  2.20130617e-03
  1.13539182e-01 -2.97226245e-04]
0.99999994
```

IV. Embedding Visualisation

We visualise our movie embeddings, while sorting them by genre, using the t-Distributed Stochastic Neighbor Embedding ($t{-}SNE$) dimention reduction technique. It uses the local relationships between points to create a low-dimensional mapping, which allows it to capture non-linear structures in our embeddings.



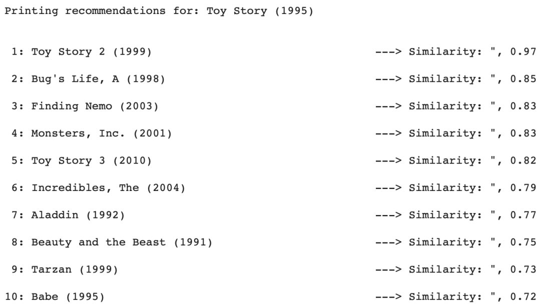
We can observe that movies with similar genres are placed near each other in the embedding layer.

V. Results: Movie Recommendation

We use our embedding layers to recommend movies based on a single input movie. We do this by finding the cosine similarity between the embedding vector of the input movie and the embedding vectors of all the other movies in our dataset.

For testing, we select three movies to test our system:

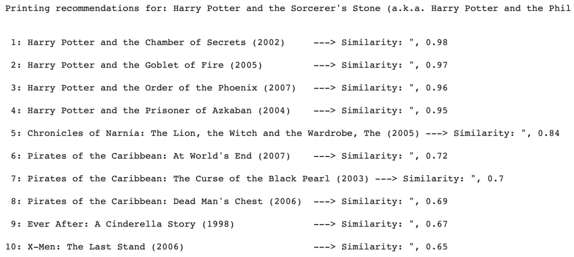
A. Movie: Toy Story 1995



```
Printing recommendations for: Toy Story (1995)

 1: Toy Story 2 (1999)                                     ---> Similarity: ", 0.97
 2: Bug's Life, A (1998)                                   ---> Similarity: ", 0.85
 3: Finding Nemo (2003)                                    ---> Similarity: ", 0.83
 4: Monsters, Inc. (2001)                                  ---> Similarity: ", 0.83
 5: Toy Story 3 (2010)                                     ---> Similarity: ", 0.82
 6: Incredibles, The (2004)                                ---> Similarity: ", 0.79
 7: Aladdin (1992)                                         ---> Similarity: ", 0.77
 8: Beauty and the Beast (1991)                            ---> Similarity: ", 0.75
 9: Tarzan (1999)                                          ---> Similarity: ", 0.73
10: Babe (1995)                                            ---> Similarity: ", 0.72
```

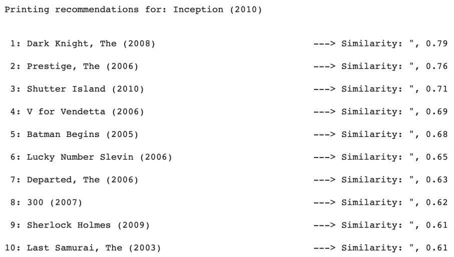
B. Movie: Harry Potter and the Sorcerer’s Stone (2001)



```
Printing recommendations for: Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Phil

 1: Harry Potter and the Chamber of Secrets (2002)     ---> Similarity: ", 0.98
 2: Harry Potter and the Goblet of Fire (2005)         ---> Similarity: ", 0.97
 3: Harry Potter and the Order of the Phoenix (2007)   ---> Similarity: ", 0.96
 4: Harry Potter and the Prisoner of Azkaban (2004)    ---> Similarity: ", 0.95
 5: Chronicles of Narnia: The Lion, the Witch and the Wardrobe, The (2005) ---> Similarity: ", 0.84
 6: Pirates of the Caribbean: At World's End (2007)    ---> Similarity: ", 0.72
 7: Pirates of the Caribbean: The Curse of the Black Pearl (2003) ---> Similarity: ", 0.7
 8: Pirates of the Caribbean: Dead Man's Chest (2006)  ---> Similarity: ", 0.69
 9: Ever After: A Cinderella Story (1998)              ---> Similarity: ", 0.67
10: X-Men: The Last Stand (2006)                       ---> Similarity: ", 0.65
```

C. Movie: Inception 2010



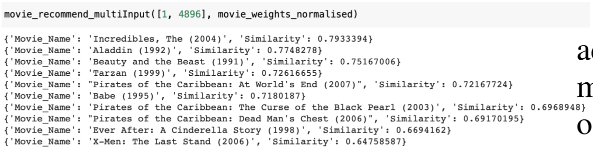
```
Printing recommendations for: Inception (2010)

 1: Dark Knight, The (2008)                            ---> Similarity: ", 0.79
 2: Prestige, The (2006)                               ---> Similarity: ", 0.76
 3: Shutter Island (2010)                              ---> Similarity: ", 0.71
 4: V for Vendetta (2006)                              ---> Similarity: ", 0.69
 5: Batman Begins (2005)                               ---> Similarity: ", 0.68
 6: Lucky Number Slevin (2006)                         ---> Similarity: ", 0.65
 7: Departed, The (2006)                               ---> Similarity: ", 0.63
 8: 300 (2007)                                         ---> Similarity: ", 0.62
 9: Sherlock Holmes (2009)                             ---> Similarity: ", 0.61
10: Last Samurai, The (2003)                           ---> Similarity: ", 0.61
```

We can also input a list of movies together in the recommendor, and then the system will compare the cosine similarity of those movies with the embeddings of other movies in the dataset to give recommendations based on multiple inputs.

Here, we have tried to test the recommendation system with two inputs together:

- Toy Story 1995
- Harry Potter and the Sorcerer's Stone (2001)



```
movie_recommend_multiInput([1, 4896], movie_weights_normalised)

{'Movie_Name': 'Incredibles, The (2004)', 'Similarity': 0.7933394}
{'Movie_Name': 'Aladdin (1992)', 'Similarity': 0.7748278}
{'Movie_Name': 'Beauty and the Beast (1991)', 'Similarity': 0.75167006}
{'Movie_Name': 'Tarzan (1999)', 'Similarity': 0.72616655}
{'Movie_Name': "Pirates of the Caribbean: At World's End (2007)", 'Similarity': 0.72167724}
{'Movie_Name': 'Babe (1995)', 'Similarity': 0.7180187}
{'Movie_Name': 'Pirates of the Caribbean: The Curse of the Black Pearl (2003)', 'Similarity': 0.6968948}
{'Movie_Name': "Pirates of the Caribbean: Dead Man's Chest (2006)", 'Similarity': 0.69170195}
{'Movie_Name': 'Ever After: A Cinderella Story (1998)', 'Similarity': 0.6694162}
{'Movie_Name': 'X-Men: The Last Stand (2006)', 'Similarity': 0.64758587}
```

We can see that the recommendations that the system generates are accurate, with high values of similarity between the input movie and the recommended movies.

VI. Conclusion

A good recommendation system should be able to scale very well over large customer bases with huge product catalogues, in order to provide an accurate, fast and personalised experience for each user. Our project provides insight into this problem of building a recommendation system for recommending items to users, based on the interactions with the same items by other users.

We designed and implemented a system which recommends movies using neural network embeddings. The system is able to recommend movies by finding the similarities in their embedding vectors.

We also calculated that our model takes around 11ms to recommend a movie based on a single input movie, and 24ms to recommend a movie based on a list of three input movies. This is a tolerable delay amount in website querying, and can easily be applied to an online movie platform to provide movie recommendations on the go.

In future work, we would like to address the issue of recommending movies that have extremely low number of user ratings to begin with. This may be improved by combining both collaborative filtering and content based recommendation, which could allow us to incorporate such cases with sparse data availability in our recommendation system.

References

[1] Dietmar Jannach, Paul Resnick, Alexander Tuzhilin, and Markus Zanker. 2016. Recommender systemsbeyond matrix completion. Commun. ACM (2016).

[2] Linden, G.; Smith, B.; York, J.; , ”Amazon.com recommendations: item-to-item collaborative filtering,” Internet Computing, IEEE , vol.7, no.1, pp. 76- 80, Jan/Feb 2003

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. ICCV 2015: 1026-1034

[4] Mojdeh Saadati, Syed Shihab, Mohammed Shaiqur Rahman: Movie Recommender Systems: Implementation and Performace Evaluation.

[5] Koehrsen, Will. Neural Network Embeddings Explained. Medium, Towards Data Science, 2 Oct. 2018, towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526.

[6] Koehrsen, Will. Building a Recommendation System Using Neural Network Embeddings. Medium, Towards Data Science, 6 Oct. 2018, towardsdatascience.com/building-a-recommendation-system-using-neural-network-embeddings-1ef92e5c80c9.

[7] Candillier, L., Meyer, F., Boull'e Marc. (2007). Comparing state-of-the-art collaborative filtering systems. Proceedings of the 5th International Conference on Machine Learning and Data Mining in Pattern Recognition, Leipzig, Germany. 548-562. doi: 10.1007/978-3-540-73499-4_41

[8] Chih-Ming Chen, Chuan-Ju Wang, Ming-Feng Tsai, Yi-Hsuan Yang, Collaborative Similarity Embedding for Recommender Systems.