# MOVIE RECOMMENDATION USING CONTENT-BASED AND COLLABORATIVE FILTERING

A Project Report

Submitted as a Part of Major Project-2

**Master of Technology in Information System**

**By**

**AGNIVA GOSWAMI**

**2K17/ISY/01**

**Under the Guidance of:**
**Ms. PRIYANKA MEEL**
**(Assistant Professor - Information Technology)**

**Delhi Technological University**

**(Formerly Delhi College of Engineering)**

**ShahbadDaulatpur, Bawana Road, Delhi – 110042**

**June-2019**

## CERTIFICATE

This is to certify that Major Project report-2 entitled "**MOVIE RECOMMENDATION USING CONTENT-BASED AND COLLABORATIVE FILTERING**" submitted by **AGNIVA GOSWAMI (Roll No. 2K17/ISY/01)** for partial fulfillment of the requirement for the award of degree Master Of Technology (Information System) is a record of the candidate work carried out by him under my supervision.

**Assistant Prof: Ms. Priyanka Meel**
Project Guide

Department of Information Technology

Delhi Technological University

# DECLARATION

We hereby declare that the Major Project-2 work entitled "**MOVIE RECOMMENDATION USING CONTENT-BASED AND COLLABORATIVE FILTERING**" which is being submitted to Delhi Technological University, in partial fulfillment of requirements for the award of degree of Master of Technology (Information System) is a bonafide report of Major Project-2 carried out by me. The material contained in the report has not been submitted to any university or institution for the award of any degree.

**AGNIVA GOSWAMI**

2K16/ISY/15

## ACKNOWLEDGEMENT

As a matter of first importance I might want should thank those Almighty, who need continuously guided me on worth of effort on the right way of the life. My best much appreciated would with my guardians who gave capability What's more quality over me to complete this worth of effort.

I owe An profound gratitude on my one task aide right hand **Assistant Prof. Ms. PRIYANKA MEEL** who need been An consistent sourball about impulse will me All around those time from claiming this venture. It might have been as much skillful guidance, steady consolation What's more basic assessment that aided me with create knowledge under my undertaking. Her calm, gathered and professionally immaculate style for taking care of circumstances not best steered me through each problem, as well as aided me with develop as An developed man.

**AGNIVA GOSWAMI**

**2K17/ISY/01**

# ABSTRACT

As the online entertainment industry and the e-commerce markets grows rapidly, so is the need for efficient recommendation engines and efficient algorithms, for the business of the companies so that large amount of revenue can be generated. The thesis proposes a hybrid collaborative and content based filtering algorithm so that the online entertainment market can be benefited, especially the online movie market, which gives the plus points of both, semantics and frequency based filtering along with a collaborative based approach which predicts the ratings of every movie. In the end, this thesis shows the results of the proposed hybrid algorithm, along with the other known filtering techniques and algorithms used to recommend movies.

*Keywords— recommendation engine; ecommerce; frequency; semantics; collaborative; ratings; hybrid algorithm;*

.

# TABLE OF CONTENTS

**LIST OF FIGURES……………………………………………………………………….7-8**

**LIST OF ABBREVIATION……………………………………………………………….8**

**CHAPTER 1: INTRODUCTION………….……………………………………….........1-3**
1.1 WHAT IS A RECOMMENDATION ENGINE?......................................................1
1.1.1 CONTENT-BASED FILTERING……..…………………………………2
1.1.2 COLLABORATIVE FILTEREING…...………………………………2-3

**CHAPTER 2: LITERATURE REVIEW……………………………………………….4-15**
2.1 SINGULAR VALUE DECOMPOSITION…………………………………..4-11
2.2 TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY ALGORITHM...12
2.3 AVERAGE WORD2VEC ALGORITHM……………………………………..13-14
2.4 TF-IDF WEIGHTED WORD2VEC……………………………………………15

**CHAPTER 3:THE PROPOSED WORK…………………………………………...16-24**
3.1 OVERVIEW……………………………………………………………………..16
3.2 DATA ACQUISITION…………………………………………………………16
3.3 DATA CLEANING…………………………………………………………16-19
3.4 THE PROPOSED ALGORITHM…………………………………………...19-22
3.4.1 COMPUTING TF-IDF WEIGHTED WORD2VEC MODEL…………….19
3.4.2 COMPUTING ABSENT RATINGS USING SVD……………………….20-21
3.4.3 MERGING THE TWO ALGORITHMS……………………………………..22
3.5 THE ALGORITHM………………………………………………………….22-23
3.6 THE FLOWCHART……………………………………………...……23-24

**CHAPTER 4: RESULTS……………………………………………………….....25-33**
4.1 DATA CLEANING RESULTS…………………………………………...25-27
4.2 RESULTS AFTER USING TF-IDF ALGORITHM………………………...28-29
4.3 RESULTS AFTER USING WORD2VEC ALGORITHM……………………29-31
4.4 RESULTS OF TF-IDF WEIGHTED WORD2VEC WITH PREDICTED RATINGS USING SINGULAR VALUE DECOMPOSITION……………………………………..32-33

**CHAPTER 5: SOFTWARE REQUIREMENTS……………………………………...34-35**
5.1 SOFTWARE DESCRIPTION……………………………………………………34-35
5.2 PACKAGES……………………………………………………………………35

**CHAPTER 6: CONCLUSION AND FUTURE WORK………………………………….36**

**CHAPTER 7: REFERENCES………………………………………………………..37-39**

# LIST OF FIGURES

| S No | Figure Name | Page no. |
|---|---|---|
| 1 | Geometrical representation of M= U∑V* | 05 |
| 2 | users rating to different movies | 07 |
| 3 | decomposition of the original matrix | 08 |
| 4 | Space taken by the component matrices | 09 |
| 5 | component matrices filled with random values | 09 |
| 6 | Resultant matrix after the dot product | 10 |
| 7 | resultant matrix after increasing the values of the component matrix after first epoch | 10 |
| 8 | value of the component matrix after second epoch | 11 |
| 9 | Partially filled matrix | 11 |
| 10 | Different methods to implement word2vec | 14 |
| 11 | Comparison of Adam to other optimization algorithm training a multilayer perceptron ; taken form Adam :A method for Stochastic Optimization,2015 | 21 |
| 12 | Flowchart | 23-24 |
| 13 | Data of movies.csv | 25 |
| 14 | Data of tags.csv | 25 |
| 15 | Data of ratings.csv | 25 |
| 16 | the encoded data | 26 |

| | | |
|---|---|---|
| 17 | data after cleaning | 27 |
| 18 | The initial component matrix | 27 |
| 19 | The values of tf-idf vector | 28 |
| 20 | the complete first row of the tf-idf vector | 28 |
| 21 | tf-idf based results | 29 |
| 22 | the word2vec matrix | 29 |
| 23 | part of a 300 dimensional vector | 30 |
| 24 | vector after dimensionality reduction | 30 |
| 25 | spatial positions of the movies | 31 |
| 26 | word2vec based results | 31 |
| 27 | the predicted average ratings | 32 |
| 28 | results of the proposed algorithm | 33 |

## **LIST OF ABBREVIATIONS**

| **S No** | **Abbreviated Name** | **Full Name** |
|---|---|---|
| 1 | IDF | Inverse Document Frequency. |
| 2 | TF | Term Frequency |
| 3 | SVD | Singular Value Decomposition |
| 4 | CSV | Comma Separated Values |
| 6 | PCA | Principal Component Analysis |

# CHAPTER 1

# INTRODUCTION

With the increase in the number of entertainment and e-commerce websites, people have move towards watching movies and television shows and purchase of different products online. But the information present on the websites is too much, that it becomes difficult for people to find useful and relevant information. Hence, Recommender Systems play a very important role in making the user buying experience hassle free. Recommender Systems apply machine learning algorithms, to predict the rating or preference a buyer would give to an item and the suggest top-n items according to the ratings predicted. Different companies are working on the recommendation algorithms so that the existing algorithms can be improved. Recommendation System algorithms can be content-based or collaborative-based.

Companies like Amazon, generates about 35% of revenue from product recommendation, which equals approximately more than 40billion$ in 2016, according to statistics. In this project, we have used content based approach on both title and tags of the movies, and collaborative approach to find out the ratings of all the movies by all the users for product recommendation. This project will describe methods to recommend movies to the users for the purpose of suggesting relevant movies. Different approaches have been used on text, collaborative based approach like singular value decomposition, content based approach such as Tf-Idf (which stands for, Term frequency-Inverse Document Frequency) and Word2Vec, developed by Google in the year 2013, for developing content based recommendation system. They are being modified in different ways so that the recommendations to the results improve. Some examples where recommendation engines are used is as follows, “you might also like” in Amazon.com, youtube.com suggestions, Netflix’s movie recommendation.

Recommendation of movies and products makes it easier for the users of the websites and buyers and also generates profits to the companies as it attracts the customers into buying more products, watching more movies and makes them spend more time in the websites and thus making more profit.

## 1.1 What is a Recommender Engine?

An information system that suggests only the relevant results to the users by removing all the unnecessary and redundant information by implementing different algorithms that are computerized and automated is known as recommender system. Some real world examples are recommendation of book on Amazon.com, recommendation of movies on Netflix. All of the recommendation is dependent on the characteristics of the data available to us and the domain. The recommender system was developed to bridge the gap between the collection of information and analysis of it by removing all the redundant data and presenting to the user all the essential data.

The possible ways to suggests the recommendations to the users by the recommender system is by one the two ways: 1.Content based filtering 2.Collaborative filtering

**1.1.1 Content-based Filtering**: The most common approach while designing a recommender system. This filtering method filters the data based the item description and according to the data collected by tracking the user's preferences [1]. The user's search history is taken into consideration and the recommendation is done based on this search history. The data is provided to the system either implicitly when a link is clicked by an user or explicitly when an user rates an item, this data is then further used by the recommender system to recommend products. All the items are the compared with the query item which the user had rated earlier. After every user activity the suggestion is updated, for example when the user gives good ratings to a particular product, the probability of that item in future recommendation increases. For example if a user like a page containing words such as pen drive, RAM, mobile, then content based filtering will recommend web pages related to electronics. Content based filtering tried to recommend items based on the similarity count. A content based recommender engine works with what data the user provides, either implicitly by clicking on a link or explicitly by adding tags and comments, based on this , a user's profile is created which is in turn used to recommend products. The more the user provides inputs or takes action, the better the recommender engine becomes.

**1.1.2 Collaborative Filtering**: in this type of filtering the behaviors of users that are somehow related to the a user is taken into consideration while recommending product of a user. For example, if an user A likes an item and an user B does not like that same item, if an user C is more closely related to user A then that particular item is recommended to user C.

So in this filtering, the rating of other close users is taken into consideration, the closest item that other user has liked is also taken into consideration.
Recommendations are given to the users based on what the other user of the same group has preferred. The algorithm calculates the similarities between different items by using various similarity measures. , and then these same similarity measures are used to predict the ratings for a user-item pairs that is absent in the dataset. The prediction is not based on similar people’s ratings but rather on the weighted average of all the people. This weight is based on the similarity between two users, that is a person and the person for whom prediction or recommendations have to be made, the Pearson correlation coefficient can be used to measure the similarity.

# CHAPTER 2

# LITERATURE REVIEW

## 2.1 Singular Value Decomposition Algorithm

Given an M*N complex or real matrix, the singular value decomposition of this matrix is the factorization of the form U∑V*, where U means it is an M*M unitary matrix, V is an N*N unitary matrix and V* is the conjugate transpose of V and ∑ is a M*N rectangular diagonal matrix which is filled with non-negative real numbers, these values are known as singular values of the original matrix. The columns of U and V are known as left and right singular values of the original matrix respectively.

SVD or singular value decomposition helps in decomposition of matrix into constituent parts thus reducing the matrix size and to make calculations simpler. SVD can be used to calculate matrix inverse, helps in data reduction in machine learning, also in image compression and to de-noise data.

When the original matrix is a real square matrix with positive determinants, then ∑ is regarded as a scaling matrix and U and V* is known as rotation matrix and thus the decomposition can be translated as rotation or reflection and scaling. In the below figure 1[7], the upper top right shows the transformed matrix with singular values of $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$, the bottom left show the action of V* on the original matrix, which is just rotation. And the bottom right shows the effect of ∑V* on the original matrix where ∑ just scales the disc vertically and horizontally.

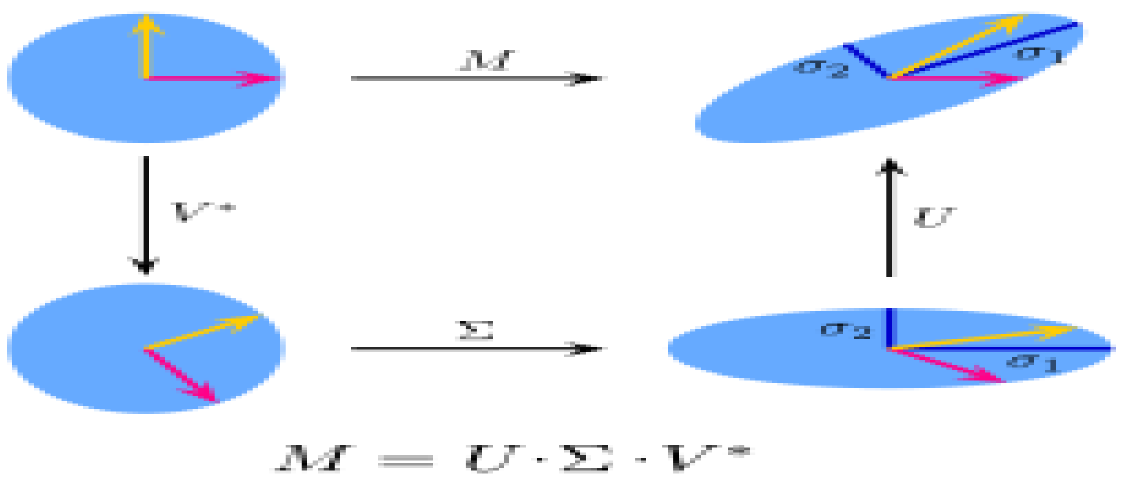


Figure 1 Geometrical representation of M= U∑V*

We can also consider the original matrix M as a product of R and P, thus M=RP, which means M is a composition of stretch (P=V∑V*) with eigenvalues of $\sigma_i$ along the eigen vectors $V_i$ of P followed by a single rotation (R=UV*). If in case the rotation is done first then M is represented as P'R where R is the same but P'=U∑U*, here P has the same eigen values but is stretched along different directions.

We can conclude that the columns of $U(U_{1,\ldots\ldots,}U_{m)}$ and $V(V_{1,\ldots\ldots,}V_n)$ yield a ortho normal basis of $B^m$ and $B^n$ respectively, thus the linear transformation $T{:}B^n \rightarrow B^m$ can be described simply with respect to these bases. We have,

$T(V_i)= \sigma_i U_i$ ,    i=1,…..,minimum(m,n),

Where $\sigma_i$ is the i$^{th}$ diagonal entry of ∑.

Thus geometrically we can summarize the following: for every map $T{:}B^n \rightarrow B^m$ , we can find orthogonal bases so that it maps a basis vector of $K^n$ to a basis vector of $K^m$ and puts the rest of the vector values as zero, thus the map T is represented with respect to these orthogonal bases by a diagonal matrix with real non-negative diagonal entries.

Let us understand with an example,

Consider the 4*5 matrix,

[[1 0 4 0 5]

 [7 0 3 1 0]

```
 [5 8 0 0 2]
 [0 2 0 1 0]]
```

After decomposition,

```
U= [[ 0.27049746  0.61564858 -0.73949489  0.03090716]
    [ 0.52042844  0.54542687  0.64878066  0.10366971]
    [ 0.80198973 -0.5339519  -0.1601416  -0.2146218 ]
    [ 0.11312752 -0.19591254 -0.08115184  0.97068781]]

∑=[[11.09181846 0 0 0 0]
   [0    6.7692937 0 0 0]
   [0  0  5.34593965 0 0]
   [0 0 0 1.25265932 0]]

V*=[[ 0.71435042  0.59883534  0.23830855  0.05711921 0.26654482]
 [ 0.26057035 -0.68891091  0.60550999  0.05163232  0.29697915]
 [ 0.56140959 -0.27000612 -0.18923476  0.10617943 -0.75155313]
 [-0.25267352  0.17913988  0.34697204  0.85766138 -0.2192997 ]
 [ 0.20683794 -0.24860329 -0.64835739  0.49720659  0.47731832]]
```

SVD can also be used for **Dimensionality Reduction,**

One of the most popular applications of SVD is for dimensionality reduction. Many times the number of features (columns) in a data corpus can be huge, svd can thus be used to reduce the feature to only a few features that depict the most relevant features of the problem. We therefore get a matrix which approximately resembles the original matrix and has a lower rank. To do this we apply the SVD algorithm on the original matrix and select the best ‘x’ largest values of sigma, the columns are selected from Sigma and the rows are selected from $V^T$. Let us see with an example,

Let A be the original matrix of size 3*10,

```
[[ 1  2  3  4  5  6  7  8  9 10]
 [11 12 13 14 15 16 17 18 19 20]
 [21 22 23 24 25 26 27 28 29 30]]
```

Decompose the original matrix into U∑V*

Let us reduce the matrix size of ∑ to 3 rows and 2 columns,

Then to finally calculate the resultant matrix we apply the following formula,

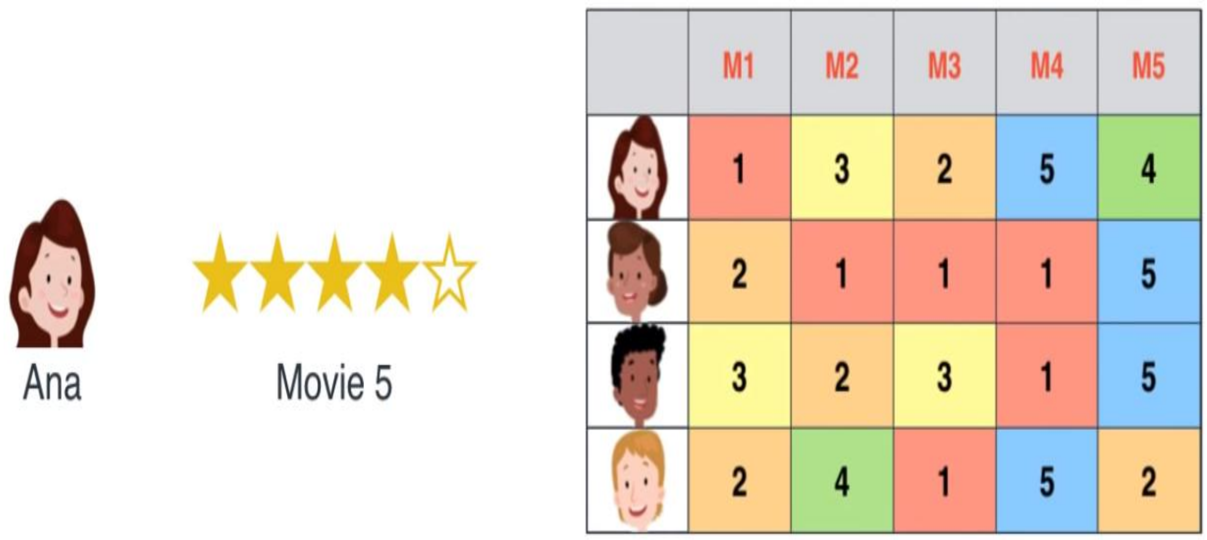
U.∑ to get the below dimensionally reduced matrix,

```
[[-18.52157747   6.47697214]
 [-49.81310011   1.91182038]
 [-81.10462276  -2.65333138]]
```

**SVD in Movie Recommendation**

Let us look with some examples and diagrams, how the SVD algorithm helps us in movie recommendation by predicting user ratings to a particular movie, whose rating was not known.

The image below shows the rating given by different persons to different movies,



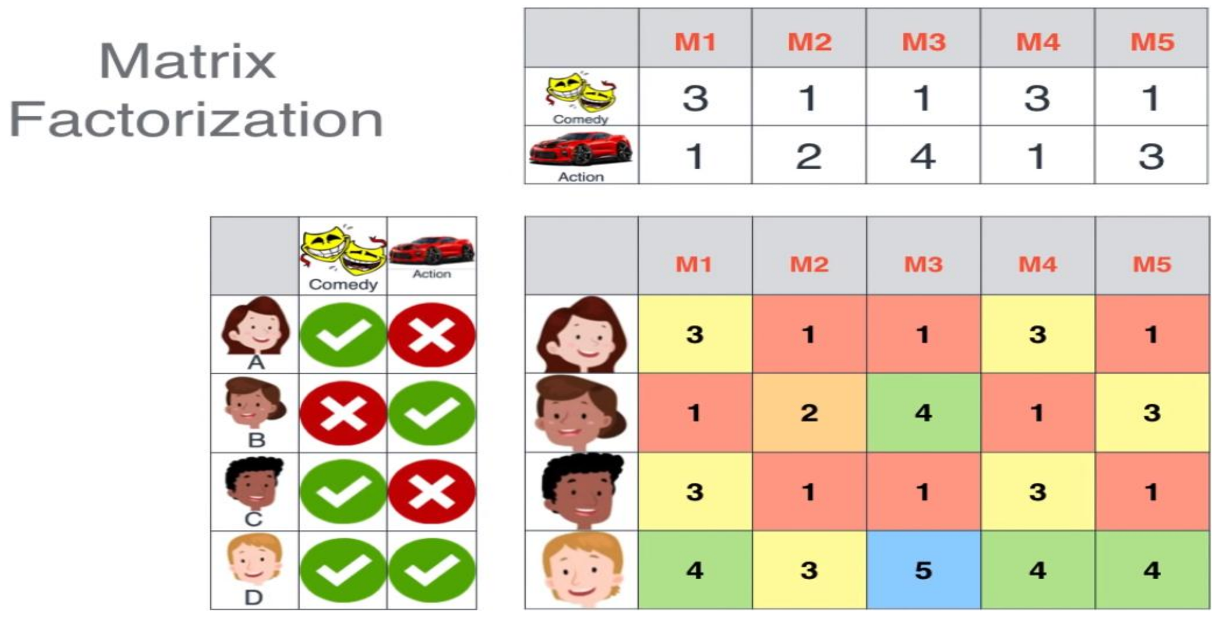
| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| | 1 | 3 | 2 | 5 | 4 |
| | 2 | 1 | 1 | 1 | 5 |
| | 3 | 2 | 3 | 1 | 5 |
| | 2 | 4 | 1 | 5 | 2 |

Figure 2 users rating to different movies

The matrix in the above image is the rating matrix, where each cell values corresponds to the rating given out of 5, by a particular user to a particular movie (depicted by M1,…,M5).

Now we want to decompose this matrix into components, so that if there is any vacant cell, we can calculate its value using the component matrix.

Below is a simple diagram which shows how we can decompose the given rating matrix into very simple components. There are two components, the movie component has two features namely comedy and action, the other component or the user component has also the same two features. The ‘comedy’ row in movie component tells us how much a movie is into ‘comedy’, same with ‘action’ row. And the comedy column in user component tell us whether a user likes ‘comedy’ or  not , same with the ‘action’ column. The components matrices are filled in such a way, that it person A likes comedy and does not like action, and a particular movie has comedy percentage as 3 and action percentage as 1 then the corresponding cell in the matrix is filled with 3 only, and if a person likes both comedy and action then the cell is filled with 4 that is (3+1). We keep on doing the same and keep filling the matrix, as our resultant matrix is same as the original one thus our decomposition is perfectly done.


Matrix Factorization

| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| Comedy | 3 | 1 | 1 | 3 | 1 |
| Action | 1 | 2 | 4 | 1 | 3 |

| | Comedy | Action |
|---|---|---|
| A | ✓ | ✗ |
| B | ✗ | ✓ |
| C | ✓ | ✗ |
| D | ✓ | ✓ |

| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| | 3 | 1 | 1 | 3 | 1 |
| | 1 | 2 | 4 | 1 | 3 |
| | 3 | 1 | 1 | 3 | 1 |
| | 4 | 3 | 5 | 4 | 4 |

Figure 3 decomposition of the original matrix

Another utility of SVD is decreasing the space occupied by the data matrix, in the below image it shows that the original data matrix have 2 million entries, and if it can be decomposed into components, then the number of entries reduces drastically to 300 thousand. The values of each cell can be calculated by the dot product of the corresponding column of the upper component matrix and the corresponding row of the left component matrix, like in the below the cell in brown can be calculated by the dot product of the column (in brown) in the upper component matrix and the row (in brown) in the left component matrix.

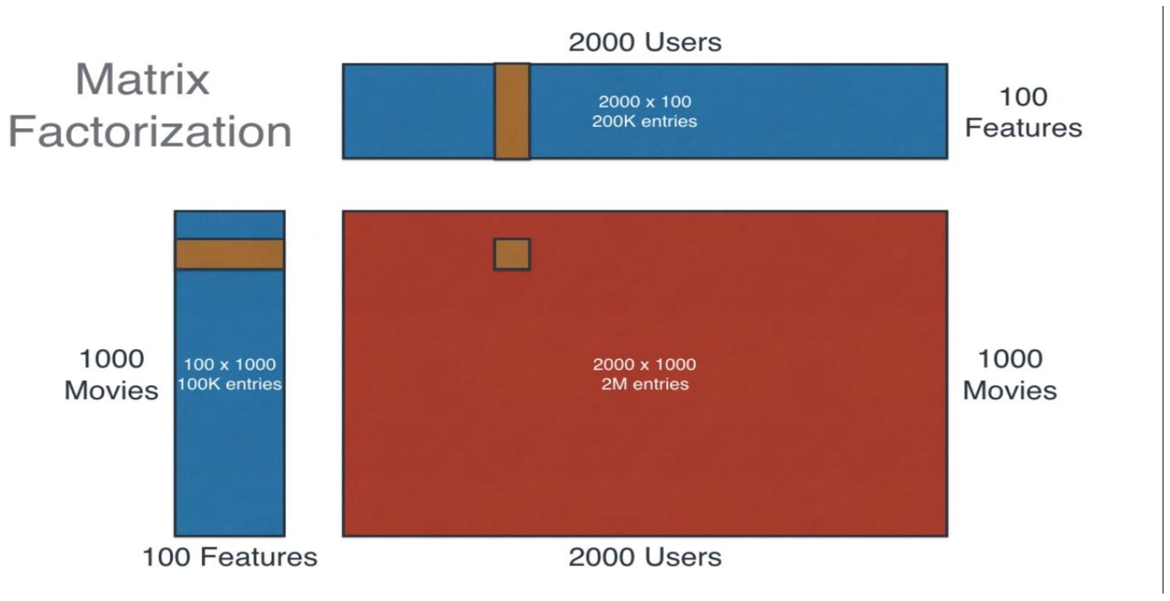
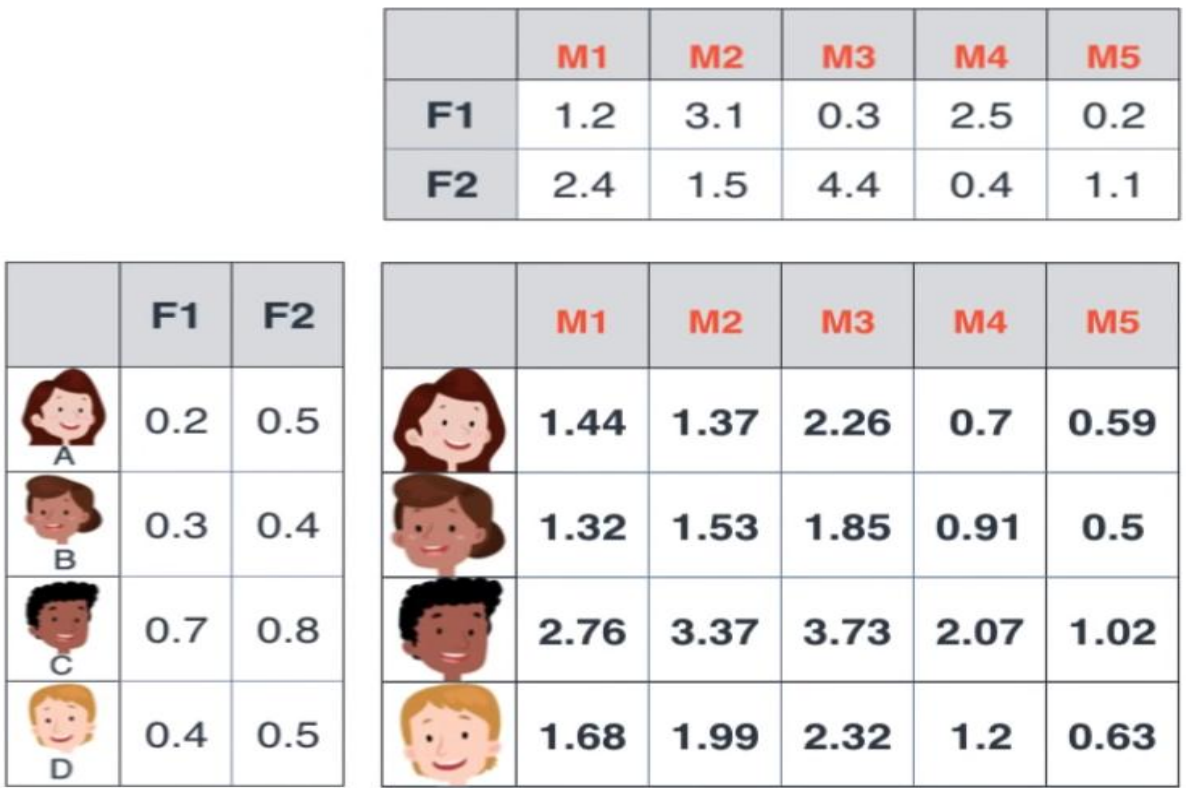


Figure 4. Space taken by the component matrices

Now let us see, how we can decompose the original matrix with an example and how to calculate the error and improve our component matrix in each epoch.

We initially take two component matrices, each with two features and fill the values randomly as shown below,



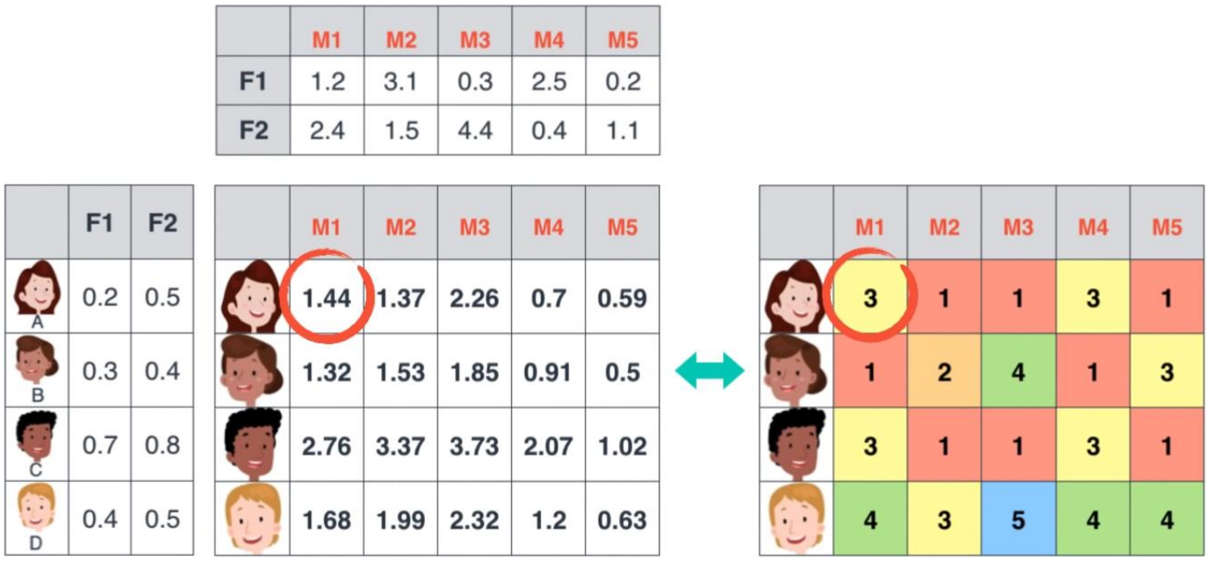
| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| F1 | 1.2 | 3.1 | 0.3 | 2.5 | 0.2 |
| F2 | 2.4 | 1.5 | 4.4 | 0.4 | 1.1 |

| | F1 | F2 |
|---|---|---|
| A | 0.2 | 0.5 |
| B | 0.3 | 0.4 |
| C | 0.7 | 0.8 |
| D | 0.4 | 0.5 |

| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| | 1.44 | 1.37 | 2.26 | 0.7 | 0.59 |
| | 1.32 | 1.53 | 1.85 | 0.91 | 0.5 |
| | 2.76 | 3.37 | 3.73 | 2.07 | 1.02 |
| | 1.68 | 1.99 | 2.32 | 1.2 | 0.63 |

Figure 5. Component matrices filled with random values

We calculate the values of each cell by the dot product of the corresponding column of the above component matrix and the row of the left component matrix,
(1.2*0.2)+(0.5*2.4)=1.44 , we keep on filling the cells of the matrix the same way.
Next we compare with the original matrix, and calculate the difference in the values of each cell, like the first cell we had calculated the value as 1.44 but the original value is 3 as shown below in (Figure 6)[6], which is more so we increase the values in the component matrix as shown in (figure 7)[6] and the value increase to 1.92. For the next cell we calculate the value as 1.83 which is more than 1 (in the original matrix) so we decrease the values of the corresponding row and column of the component matrix. We keep on doing these steps until the total error is minimum. We calculate the total error as the sum of errors of each cell.



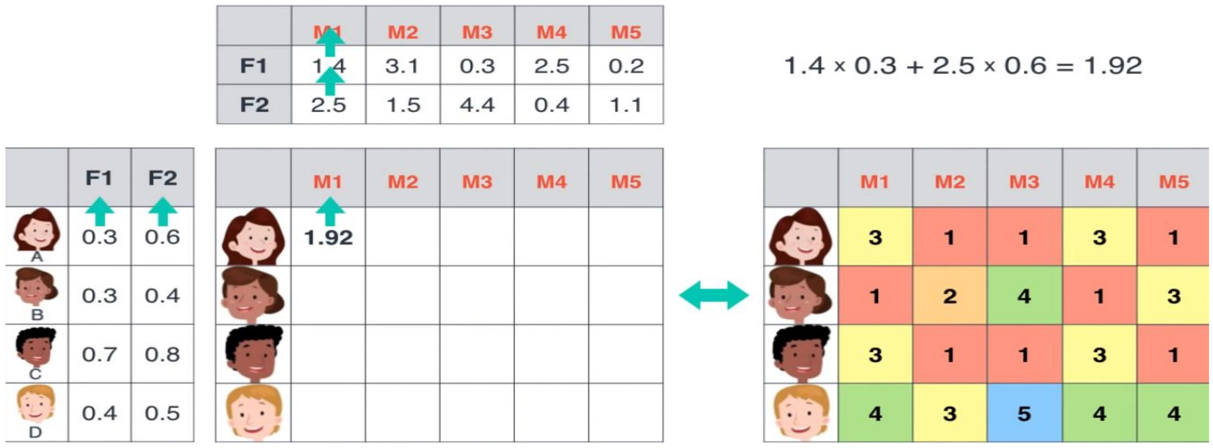
| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| F1 | 1.2 | 3.1 | 0.3 | 2.5 | 0.2 |
| F2 | 2.4 | 1.5 | 4.4 | 0.4 | 1.1 |

| | F1 | F2 |
|---|---|---|
| A | 0.2 | 0.5 |
| B | 0.3 | 0.4 |
| C | 0.7 | 0.8 |
| D | 0.4 | 0.5 |

| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| | 1.44 | 1.37 | 2.26 | 0.7 | 0.59 |
| | 1.32 | 1.53 | 1.85 | 0.91 | 0.5 |
| | 2.76 | 3.37 | 3.73 | 2.07 | 1.02 |
| | 1.68 | 1.99 | 2.32 | 1.2 | 0.63 |

| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| | 3 | 1 | 1 | 3 | 1 |
| | 1 | 2 | 4 | 1 | 3 |
| | 3 | 1 | 1 | 3 | 1 |
| | 4 | 3 | 5 | 4 | 4 |

Figure 6. Resultant matrix after the dot product.



| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| F1 | 1.4 | 3.1 | 0.3 | 2.5 | 0.2 |
| F2 | 2.5 | 1.5 | 4.4 | 0.4 | 1.1 |

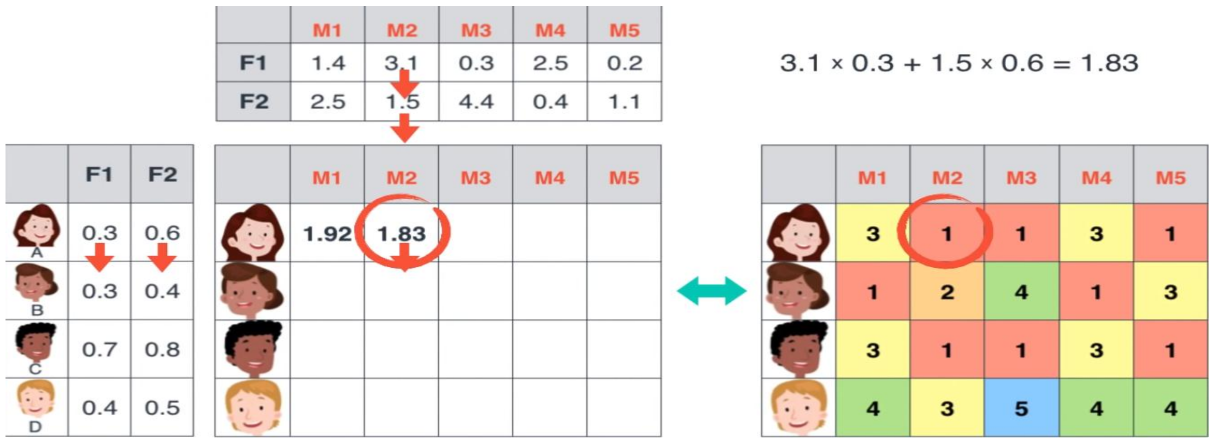
$1.4 \times 0.3 + 2.5 \times 0.6 = 1.92$

| | F1 | F2 |
|---|---|---|
| A | 0.3 | 0.6 |
| B | 0.3 | 0.4 |
| C | 0.7 | 0.8 |
| D | 0.4 | 0.5 |

| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| | 1.92 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| | 3 | 1 | 1 | 3 | 1 |
| | 1 | 2 | 4 | 1 | 3 |
| | 3 | 1 | 1 | 3 | 1 |
| | 4 | 3 | 5 | 4 | 4 |

Figure 7.resultant matrix after increasing the values of the component matrix after first epoch



|  | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| F1 | 1.4 | 3.1 | 0.3 | 2.5 | 0.2 |
| F2 | 2.5 | 1.5 | 4.4 | 0.4 | 1.1 |

$$3.1 \times 0.3 + 1.5 \times 0.6 = 1.83$$

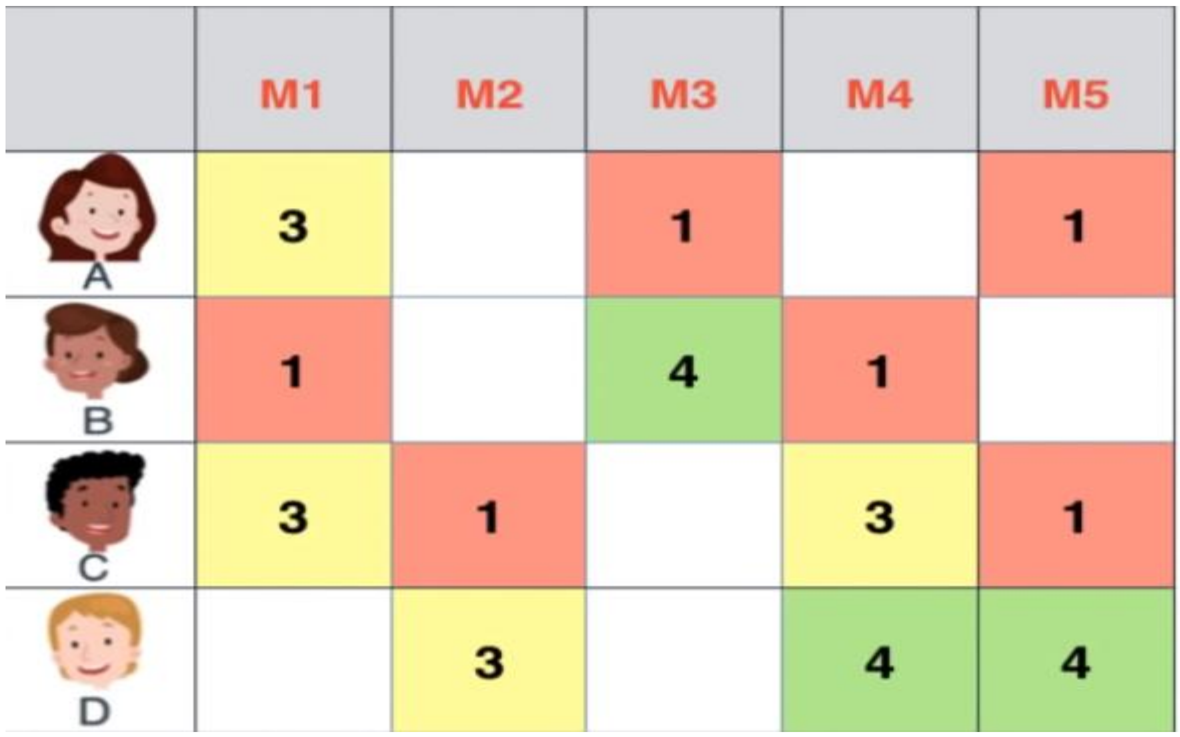
|  | F1 | F2 |
|---|---|---|
| A | 0.3 | 0.6 |
| B | 0.3 | 0.4 |
| C | 0.7 | 0.8 |
| D | 0.4 | 0.5 |

|  | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| A | 1.92 | 1.83 |  |  |  |
| B |  |  |  |  |  |
| C |  |  |  |  |  |
| D |  |  |  |  |  |

|  | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| A | 3 | 1 | 1 | 3 | 1 |
| B | 1 | 2 | 4 | 1 | 3 |
| C | 3 | 1 | 1 | 3 | 1 |
| D | 4 | 3 | 5 | 4 | 4 |

Figure 8.value of the component matrix after second epoch

Now if we have a partially filled matrix as below (Figure 8)[6], and we calculate the component matrix iteratively on the cells that are filled and bring the error down , then we can predict the values of the empty cells, which is what is done in this project that is we calculated the ratings that could have been given by a user to a certain movie.



|  | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| A | 3 |  | 1 |  | 1 |
| B | 1 |  | 4 | 1 |  |
| C | 3 | 1 |  | 3 | 1 |
| D |  | 3 |  | 4 | 4 |

Figure 9. Partially filled matrix

## 2.2 Term frequency-Inverse document frequency algorithm

Also termed as tf-idf, is the weight which is used in mining of text and retrieval of information. This measurement of weight measures the importance of a particular word in a document in a data corpus. The importance is proportional to the frequency of the word in the document. Different variation of the tf-idf algorithm is used by search engines to rank the relevance of a document given a user query. Tf-idf can also be used for filtering stop-words including text classification.

The tf-idf weight is a mixture two terms: the first is call term frequency, and the second term is known as inverse document frequency,

- Term frequency (Tf): measures the frequency of a term in a document, the term frequency is divided by the length of the document.
  TF(x) = (number of times the term x appears in a document)/ (total number of terms in the document)
- Inverse document frequency (idf): detects the importance of a term. We know that stopwords appears a lot of time with very little importance, thus weighing them down and weighing up the rare terms will increase the efficiency,
  IDF(x)= log_e(Total number of documents / number of documents having term 'x' in it)

Let us say, that we have a data corpus having titles T1,T2,……,Tn

TF(Wj,Ti) of the given word Wj in the given title Ti is given by, the value obtained by dividing frequency of Wj in the title Ti divided with the total words contained in title Ti. Let us assume that the total words in title Ti is 'n' and number of times Wj occurs is 5, then TF(Wj,Ti)=5/n

Thus for the word Wj in title Ti in data corpus D, w calculate TF(Wj,Ti)*IDF(Wj,D), and we create the TF_IDF vector for title Ti and in each cell corresponding to word Wj we fill the cell with the TF_IDF value of that particular word, and for each absent word we put 0. Thus we can say that the tf-idf value is high if the word appears very less number of times in the whole data corpus and more number of times in the given title Ti.

## 2.3 Average Word2Vec Algorithm:

Word2Vec is a text processing two layer neural network. For every text corpus as input, the output is a set of vectors, known as the feature vectors for that particular work in the data corpus. Word2vec is not a deep neural network, but tit turn the text into some numerical form that is understandable by deep neural networks.

Word2vec can as well be applied  to code, genes, playlists, social media graphs and other symbolic and verbal series which has some underlying patterns in it, because words is same as the other data mentioned before which has discrete states, and we are simply looking for the probability of likelihood among themselves. As a result gene2vec, like2vec, followers2vec all are possible.

The usefulness of using word2vec is that it groups all the similar words together in vectors space, by detecting the similarities mathematically.  Word2vec algorithm creates vectors with numerical representations of the words according to the features of contextual similarity between words. It does so with human interventions.

Word2vec makes highly accurate predictions about a word's meaning based on previous occurrences, given enough data and the contexts. These predictions can be utilized to make association between words such as "boy" is to "man" what woman is to "girl" or clustering documents by topics. These clusters form the basis of sentimental analysis, search and recommendations in various fields.

Output of word2vec network is a vocabulary of vectors, which corresponds to each item, which is then fed to the deep neural network.

The measurement of cosine similarity, no similarity is shown as a 90 degree angle, while complete similarity is shown by 0 degree that is cosine distance between Norway and Sweden is 0.760124, and cosine distance between Sweden and Sweden is 1.

Word2vec works similar to auto encoder, which encodes each word to a vector, but instead of training against the given words by reconstructing it like a Boltzmann machine, word2vec trains a word against the other words in the input which is in the neighbor.

This is done in two ways [31], the continuous bag of words method in which, to predict a word the context is used, and the skip gram method in which to predict a target context the particular word is used. The skip-gram model is used as it produces better results on large datasets.

When the feature vector assigned to the word cannot predict the context of the word accurately, the values and components of the vector are corrected and adjusted. The word's context is just like a teacher sending error messages so that adjustment can be made. The vectors of the word which are calculated to be similar based on context are brought closer to one another by adjusting the numbers.

A result of a well trained vector is that all the similar words are close to on another in the vector space. For example, words like elm, oak, birch are in cluster in one part of the vector space and words like guns, wars, conflict are clustered in another region of the vector space.

Similar words and ideas are close to one another, the relative meaning are transformed to distances, the qualities of each word are changes into quantities so that algorithms are useful. Also word2vec can establish relations between words and one language and can map them to another.

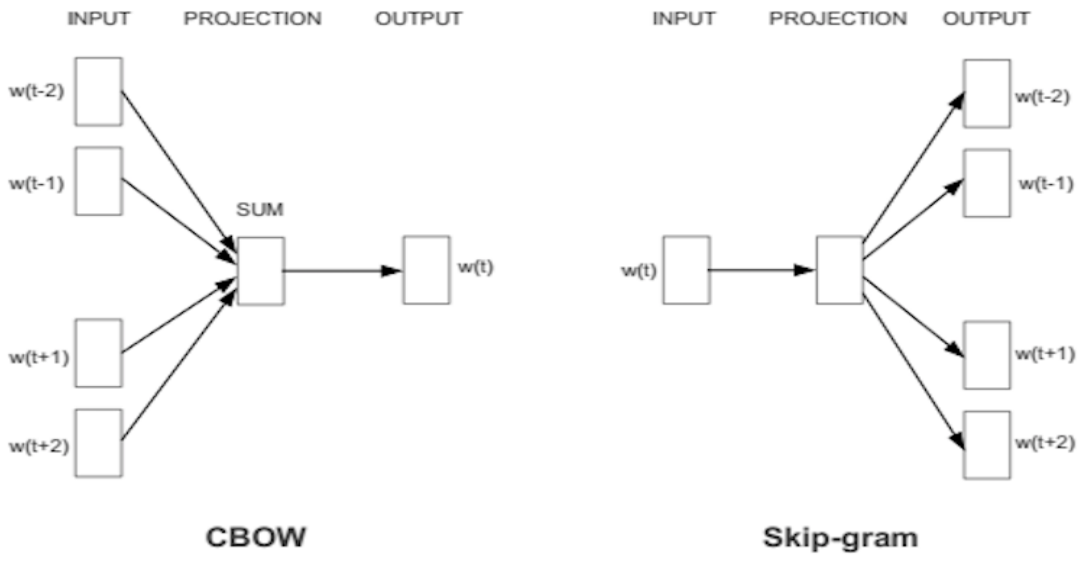


Figure 10 Different methods to implement word2vec

Methodology: given a title Ti, consisting of n words, w1,w2,……,wn. For each word wj we calculate using the word2vec model to get a 300 dimensional dense vector. To get the word2vec of the complete title Ti, we add the values of first cell of each word's word2vec and add them up and divide them by the number of words which is n here and insert the value in the first ell of the resultant new vector, we do the same with the second cell and so on till n cells.

## 2.4 TF-IDF Weighted Word2Vec Algorithm

The similarity between average word2vec and tf-idf weighted word2vec is, here we take the tf-idf value in average word2vec supposedly to be 1, but in tf-idf weighted word2vec, for each word Wj, we get the 300 dimensional word2vec and multiply the tf-idf value of the word and them add up all the corresponding cell of each word and then finally divide it with the sum of tf-idf values of each word to get the final vector for the given title.

# CHAPTER 3

# THE PROPOSED WORK

## 3.1 Overview

User ratings are an important aspect in movie recommendation besides the comments given by the user or the tags generated by the user. Rating tells us about the quality of the movie, what kind of movie a user likes based on the rating a user gives to a movie. But the problem with user ratings is that it is incomplete. It is because; every user will not rate each and every movies, so here comes the utilization of Singular Value Decomposition algorithm. This algorithm looks for pattern in the already given ratings by different users, and based on that it predicts the rating of a particular movie by a particular user. As a result we get all the rating of all the movies. This algorithm is the collaborative filtering technique used in this recommendation, because it uses information of other users to predict the results. We have also used tf-idf weighted word2vec technique, which looks both at the frequency of the genres and the words in the title of the movie, and also look at the semantic aspect by the use of word2vec algorithm. Thus our proposed algorithm makes use of the ratings predicted by the SVD algorithm, and also the frequency and semantic based techniques made effect with tf-idf weighted word2vec, this is done by assigning weights to both according to the user's preferences.

## 3.2 Data Acquisition

The data has been acquired from movielens.org website. The data set contained three datasets, namely movies.csv file which contained movie id, movie title and the tags, tags.csv file which contains more tags given to a movie by a user, ratings.csv file which contains the rating given to a movie by a particular user.

## 3.3 Data Cleaning

There were three data sets that are used in this project, and was downloaded from movielens.org. A total of 138493 users gave the ratings to many movies between 09th January, 1995 and 31st March, 2015 and generated on March 31, 2015, and updated on 17th

October, 2016. All the randomly selected users had rated a minimum of 20 movies each. No geographic locations were used of the users. Each user has an unique id, and no other information is provided to us. The data are contained in three files, `movies.csv`, `ratings.csv` and `tags.csv`.

UserId- the users of Movielens.org were selected randomly. The ids allotted to them were unique and anonymized. Userid is relatively consistent between `ratings.csv` and `tags.csv` which means that the same user id refers to the same user in both the files

.

MovieIds- Only those movies were included which had one or more than one rating. These movie ids are same as that given on Movielens.org for example movieid `1` corresponds to this link ( <https://movielens.org/movies/1>).

Movieids are consistent between `ratings.csv`, `tags.csv`, `movies.csv` which means that the same movieid is the same movie in all these four data files

Movies Data Structure (movies.csv):

The information about the movie is given in the file called `movies.csv`. Each row of this file after the header row has one movie in it.

The given format is as follows: movieId | title | genres

The titles of the movies are either manually entered or are imported from the following link: <https://www.themoviedb.org/>,

The data also has the release year of the movies included in parentheses. There were amny errors and inconsistencies in these titles which needed to be cleaned.

The Genres column contained data which were separated by a pipe '|', which included one or many of the following genres:

Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western, (no genres listed).

The total number of rows that is the total number of movies in movies.csv file is 27278, now as we see that the genres in movies.csv file is '|' separated, thus we need to remove these and add these genres tags to a list so that later we can convert the list into vectors and apply our proposed algorithm to calculate the distances between movies.

Also there is a genre known as "(no genres listed)" in the movies.csv dataset, this is an inconsistency and will cause error later, so we had to remove this particular genres tag from the movies.csv dataset.

The algorithm for removing ‘|’ and the “(no genres listed)” is as follows:

- Iterate through each genres tag in the movies.csv file
- Split the text at ‘|’
    - For each word in the splitted text check if it is not equal to “(no genres listed)”,if it is not then append it to an empty string
- Add the final string to a list

The Tags Data Structure (tags.csv)

All the corresponding movie tags are present in the file called `tags.csv`. Each row of the file after the header row contains some tags which were given to a particular movie by a particular user. The header column had the following format:

userId, movieId, tag, timestamp

each row of this file are ordered according to the userid first , then by movieid. The tags were given by the user according to what they felt after watching the movie. Thus each tag is a phrase, or a simple word. Each tag has some meaning and purpose which are user relevant. There was also a column named as “Timestamp” which showed number of seconds after the 1st of January, 1970 which elapsed when the user gave the corresponding tag.

This file is very important, as it gives us an description about what different users thought about the movie and thus the tags increase and thus become more descriptive and more the recommendation becomes accurate. We accumulated the different tags that different users gave to a particular movie and appended it to the genres tag to the list made in movies.csv data set. The algorithm is as follows

- Iterate through all the data points of tags.csv file
- Store the movie by the user, and store the tags by splitting at space, in a list.
    - Iterate through the movie.csv data set and if the movie id number matches with the stored movie id then append the stored tags list with the corresponding list that was created before during iterating through movies.csv file.

Now we have a ready list of tags corresponding to every movie, but the problem with this list is that the tags in it contains many useless items such as punctuation marks like ‘!’, ‘[‘,’]’ etc and also it contains many useless words which will give inconsistency in the results. So we remove all these useless items.

Data Preprocessing- stopwords such as ‘down’, ‘had’, ’few’, ‘both’, ‘after’, ‘my’, ‘was’, ‘because’, ‘during’, ‘about’, ‘again’, ‘y’, ‘because’ etc have been downloaded from the NLTK library and have been removed by iterating over the data set, as such words are not at all useful while recommending.

As many users tags had been appended to the same movies, so there were many duplicate words which have to be removed along with all the punctuations.

So we iterated through every item present in the list, and removed all the punctuations and added all the words to a set, which automatically deleted the duplicate words if any.

Next while iterating through the list there were many NaN values or Null values, such movies were deleted, because there is no point in keeping movies with no genre or tags present.

## 3.4 The Proposed Algorithm

### 3.4.1 COMPUTING TF-IDF WEIGHTED WORD2VEC MODEL

In average word2vec we take the tf-idf(term frequency-inverse document frequency) value to be 1, but here that is for every items in list that is the genres T(i), we run the Word2Vec model for each word W(j) and we get a 300 dimensional vector and then we do a product of the words tf-idf value, next we sum up all the cell corresponding to each column and then dive the value with the total sum of the tf-idf value of each word W(j) of each title. In this way we get the final vector for each title.

The algorithm is as follows:

1. Computing TF-IDF:
    - Tf stands for term frequency and is calculated by , total number of time a word appears in a title divided by the number of words in the whole data corpus D.
    - Idf is measured for a word W(j) to a given data corpus D, IDF(Wj,D) is equal to logarithm(number of titles in the data corpus D divided by the number of titles that contains the given word Wj).
    - Next we multiply both the values, thus the tf-idf value is high if a word appears large number of times in a given title and few number of times in the given data corpus D.

2. Computing Word2Vec:
    - Let us denote a particular title as T(i), consisting of ‘k’ number of words, w1 to wk. For every word Wj we run the word2vec model to get a 300 dimensional vector.
    - For each title T(i), we take each and every word’s Wj’s 300 dimensional vector, derived from previous step, and then do a product of the word’s tf-idf value with the 300 dimensional word2vec.
    - We now create a 300 dimensional vector, where each cell has a value after we add all the values of the corresponding cell of each word divided by the sum of tf-idf values of each word.

### 3.4.2 COMPUTING ABSENT RATINGS USING SINGULAR VALUE DECOMPOSITION

Before applying svd algorithm, we encoded the data. As the data present in Ratings.csv file was haphazard and not ordered, we had to encode the data and order it according to the userid and then by movie id. The userid which started from 1 in the original data from ratings.csv was changed and made to start from 0. The resultant encoded data had userids in increasing order and for a particular userid , the movieids were in increasing order.

Next for training our model we stored the number of unique userids and number of movie ids present in the dataset. Then we choose our embedding size as 100, which meant that we will two component matrices of size number of users cross 100 and another number of movies cross 100. These matrices were initially filled with random values, to do this we had used ‘torch’ library.

Training of our model was done with 100 epochs and a learning rate of 0.01. we initially did a dot product of the corresponding row of movie component matrix and the corresponding row of the user component matrix to find the value of a the associated cell, next we stored the original rating given to that particular movie by the particular user and then calculate the loss, by calculating the difference between both the values, if the value of the rating is more then we increase the values of the corresponding column of movie component matrix and the corresponding row of the user component matrix in the next iteration.

The process of training the sample was done by using Adam optimization algorithm; it is just an extension to the known stochastic gradient descent that is very well known in natural language processing and computer vision. This particular algorithm can be used instead of the stochastic gradient descent algorithm to update the weights of the network iteratively based on the training data. Adam is derived from ‘Adaptive moment estimation’, was presented by Diederik Kingma and Jimmy Ba in the year 2015 in their paper titled “Adam :A Method for Stochastic Optimization”[2].

The stochastic gradient descent has a rate of learning termed as Alpha for the weights updates and the rate of learning does not change during the training. Here in Adam algorithm a separate learning rate is maintained for each network weight or each parameter and adaptation happens separately as the learning unfolds. The Adam mixes the advantages of two other extensions of gradient descent, Root mean Square Propagation that maintains the per parameter learning rates that changes based on the average of the recent magnitudes of the gradient values of the weights, thus this algorithm performs well on non-stationary and online problems. And the Adaptive Gradient Algorithm that also has a per-parameter learning that performs better on problems with sparse gradients like natural language problems.

Adam identifies the benefits of both these algorithms and instead of adapting to the parameter rates of learning based on the mean as in Root mean square propagation, Adam also makes use of the uncentered variance. This algorithm specifically calculates the exponential moving average of the gradient, also the gradient square. The parameters beta1 and beta2 controls the rate of decay of the moving average.

Below is a graph showing the comparison of Adam with other algorithms,

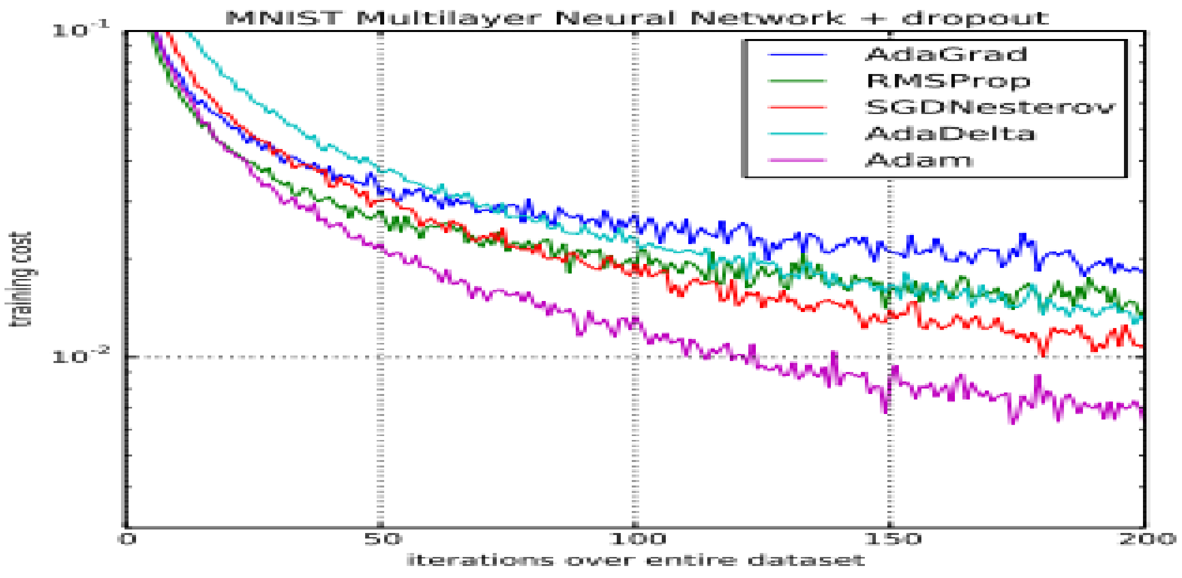


Figure 11. Comparison of Adam to other optimization algorithm training a multilayer perceptron ; taken form Adam :A method for Stochastic Optimization,2015

### 3.4.3 MERGING THE TWO ALGORITHM

Now, coming back to the implementation, after the use of a optimizer algorithm, we run the training model 100 times, or the number of epochs were 100 and the learning rate was fixed at 0.01. During each epoch we set the gradient to zero before the start of back propagation because PyTorch accumulates the gradient on subsequent backward passes. Next we compute the d(loss)/d(x) for every parameter x which has requires_grad=true, next the optimizer step updates the value x using the gradient.

After training our model and running it, we can get the rating of any movie given by an user. It is because we have trained our model so that we can guess our component matrices with very less error. The overall error has come out to be approximately 0.87.

Next we stored all the ratings for a particular movie in a list and computed the average of it, thus we get a list of average rating for a particular movie.

We then append the rating value to our previously calculated tf-idf weighted word2vector and give weights according to our preference, for example if somebody wants similar rated movies then we give more weight to the rating part of the total vector, otherwise we give equal weights to both the tf-idf weight word2vector and the rating. Also if the user wants the suggestion to be according to the ratings, then we sort the predicted movies according to the ratings.

Here we have given equal weights to the tf-idf wor2vec vector and the rating value. For example if a weight of 0 is given to the tf-idf wor2vec value and some weight to the rating value then movies of similar rating of the query movie will be shown, and when equal weights are given then movies of same genres, and tags and of same ratings will be shown.

## 3.5 THE ALGORITHM

The **algorithm** is as follows:

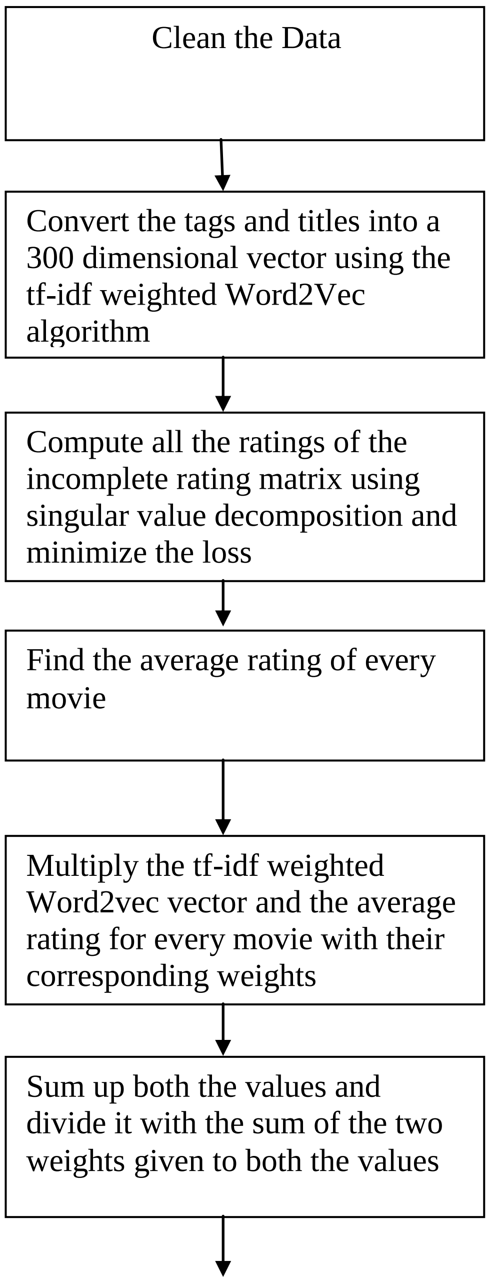
- Clean the data and insert the tags into a list
- Use the tf-idf weighted word2vec model to turn the movie titles and the tags into 300 dimensional vectors.
- Compute the rating matrix, which predicts the ratings of every movie given by every user, using the singular value decomposition algorithm
- Calculate the average rating for each movie.
- Take the 300 tf-idf weighted word2vec and multiply with the weight given to it, and

also take the average rating and multiply with the weight given to rating, sum it up and divide it with the sum of the weights.

- Now calculate the pair wise eucledian distance between the query movie and all other movies and recommend 200 movies which are closest to the query movie.

## 3.6 THE FLOWCHART

The flow chart of the algorithm is as follows,


Clean the Data
Convert the tags and titles into a 300 dimensional vector using the tf-idf weighted Word2Vec algorithm
Compute all the ratings of the incomplete rating matrix using singular value decomposition and minimize the loss
Find the average rating of every movie
Multiply the tf-idf weighted Word2vec vector and the average rating for every movie with their corresponding weights
Sum up both the values and divide it with the sum of the two weights given to both the values

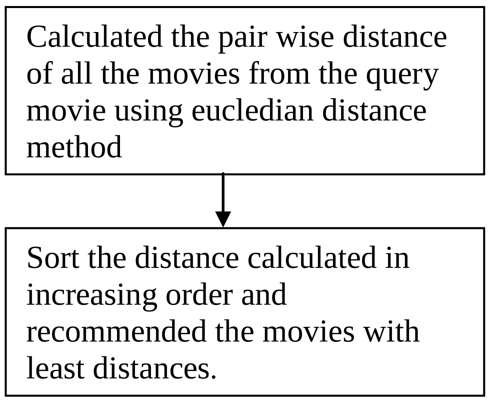

Calculated the pair wise distance of all the movies from the query movie using eucledian distance method
Sort the distance calculated in increasing order and recommended the movies with least distances.

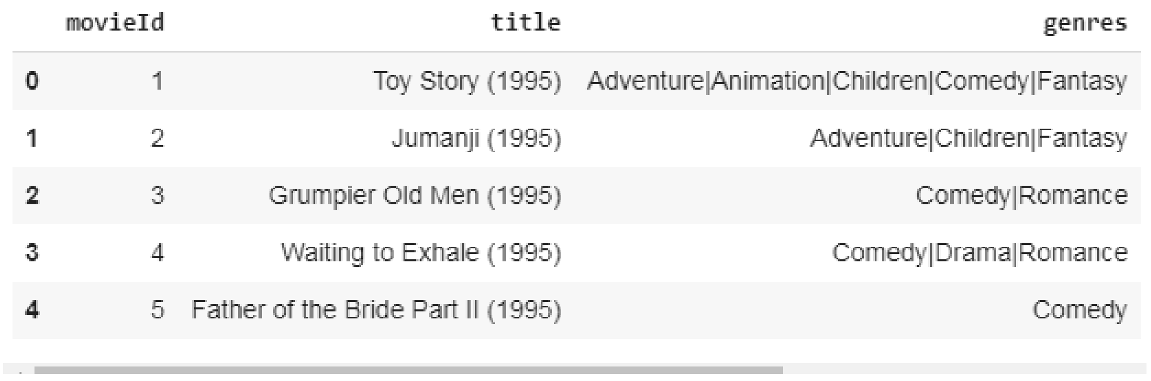
Figure 12. Flowchart

# CHAPTER 4

# RESULTS

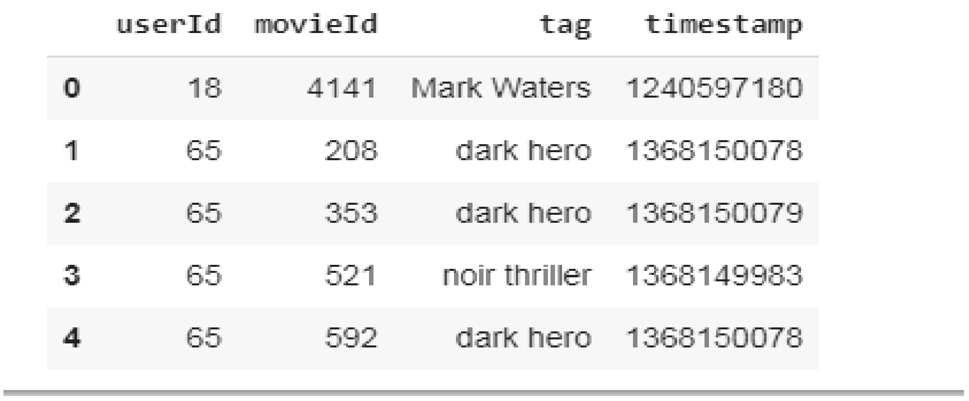
## 4.1 DATA CLEANING RESULTS

Below is the image of how the data looked in movies.csv, tags.csv, ratings.csv file,



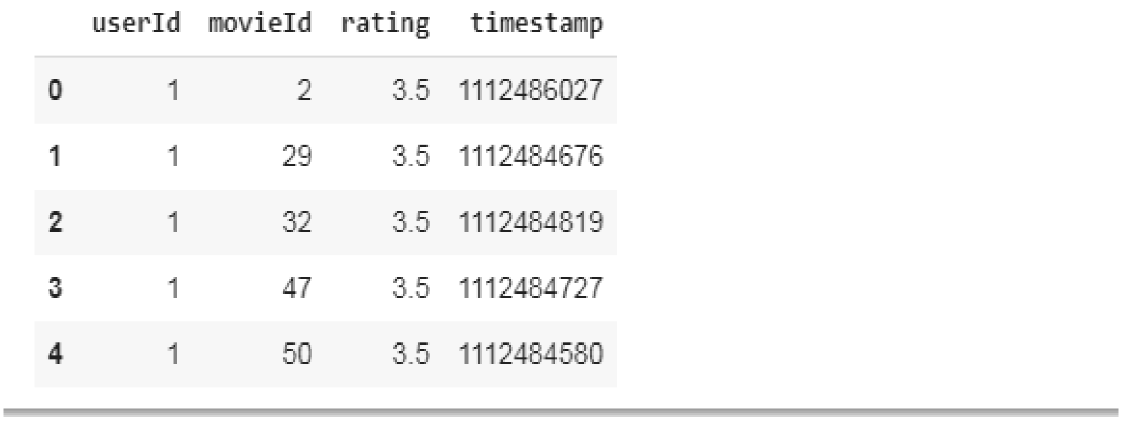
| | movieId | title | genres |
|---|---|---|---|
| 0 | 1 | Toy Story (1995) | Adventure\|Animation\|Children\|Comedy\|Fantasy |
| 1 | 2 | Jumanji (1995) | Adventure\|Children\|Fantasy |
| 2 | 3 | Grumpier Old Men (1995) | Comedy\|Romance |
| 3 | 4 | Waiting to Exhale (1995) | Comedy\|Drama\|Romance |
| 4 | 5 | Father of the Bride Part II (1995) | Comedy |

Figure 13 Data of movies.csv



| | userId | movieId | tag | timestamp |
|---|---|---|---|---|
| 0 | 18 | 4141 | Mark Waters | 1240597180 |
| 1 | 65 | 208 | dark hero | 1368150078 |
| 2 | 65 | 353 | dark hero | 1368150079 |
| 3 | 65 | 521 | noir thriller | 1368149983 |
| 4 | 65 | 592 | dark hero | 1368150078 |

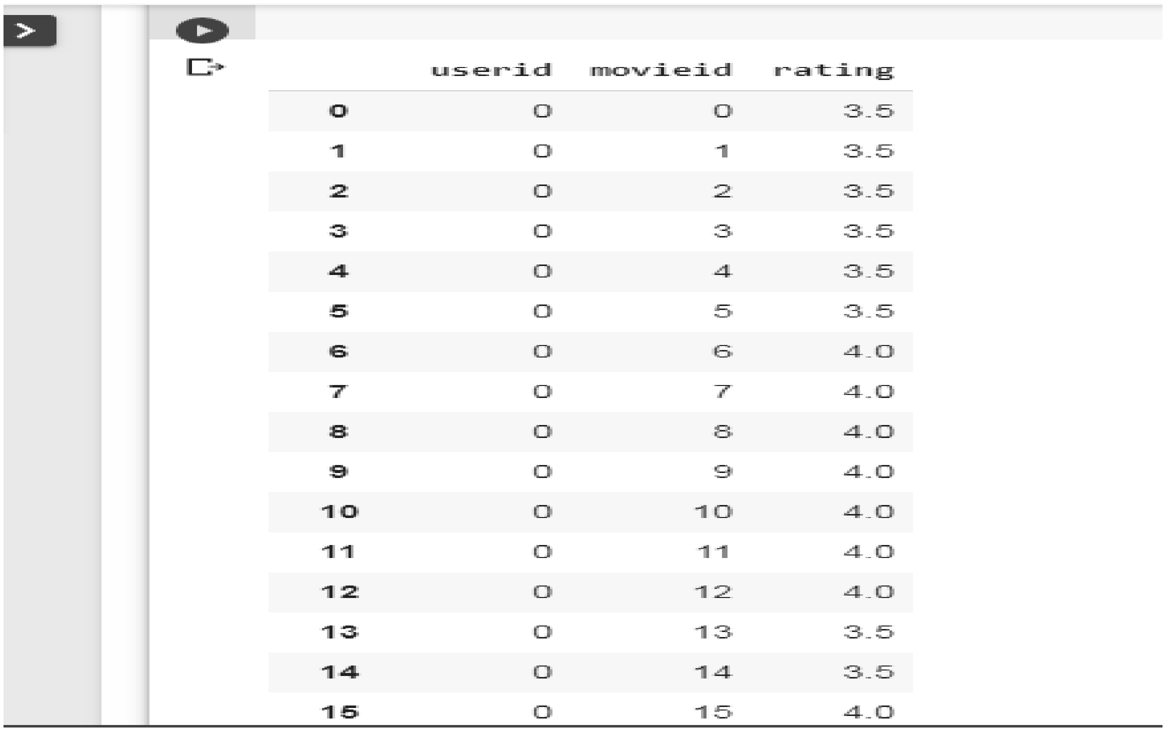
Figure 14 Data of tags.csv



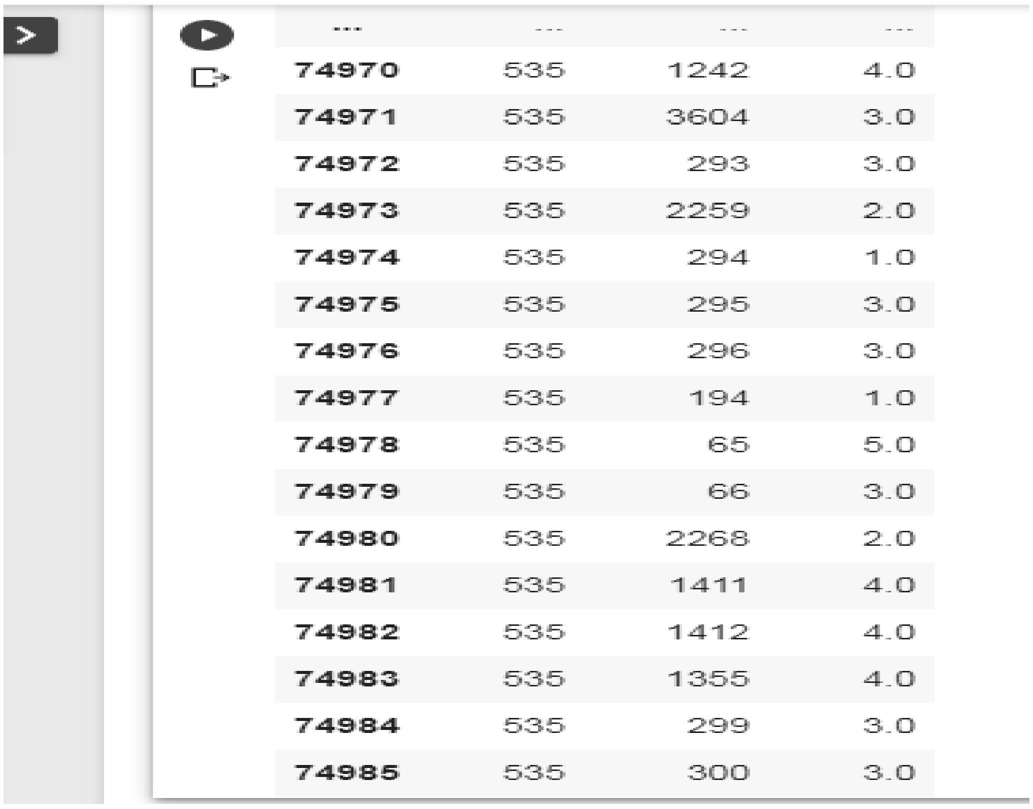
| | userId | movieId | rating | timestamp |
|---|---|---|---|---|
| 0 | 1 | 2 | 3.5 | 1112486027 |
| 1 | 1 | 29 | 3.5 | 1112484676 |
| 2 | 1 | 32 | 3.5 | 1112484819 |
| 3 | 1 | 47 | 3.5 | 1112484727 |
| 4 | 1 | 50 | 3.5 | 1112484580 |

Figure 15 Data of ratings.csv

The below image shows the data after encoding the data of ratings.csv file and after removing the timestamp column.



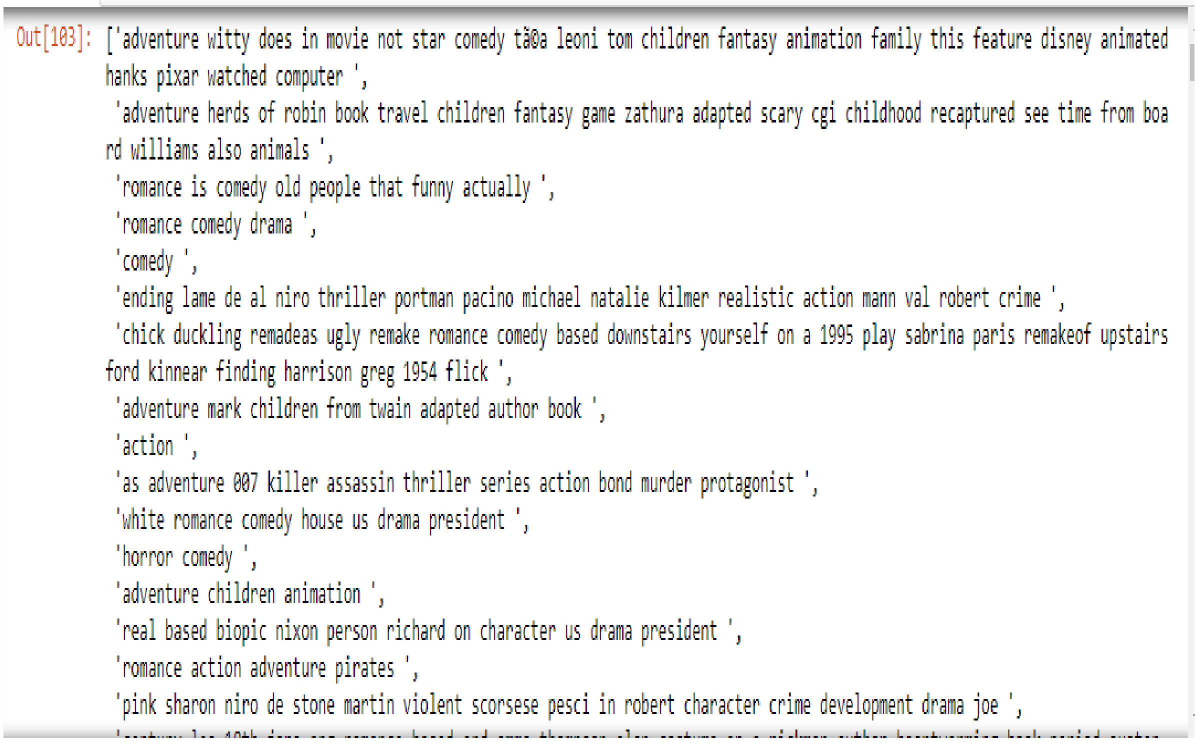
```
    userid  movieid  rating
0        0        0     3.5
1        0        1     3.5
2        0        2     3.5
3        0        3     3.5
4        0        4     3.5
5        0        5     3.5
6        0        6     4.0
7        0        7     4.0
8        0        8     4.0
9        0        9     4.0
10       0       10     4.0
11       0       11     4.0
12       0       12     4.0
13       0       13     3.5
14       0       14     3.5
15       0       15     4.0
```



```
...      ...      ...     ...
74970    535     1242     4.0
74971    535     3604     3.0
74972    535      293     3.0
74973    535     2259     2.0
74974    535      294     1.0
74975    535      295     3.0
74976    535      296     3.0
74977    535      194     1.0
74978    535       65     5.0
74979    535       66     3.0
74980    535     2268     2.0
74981    535     1411     4.0
74982    535     1412     4.0
74983    535     1355     4.0
74984    535      299     3.0
74985    535      300     3.0
```

Figure 16 the encoded data

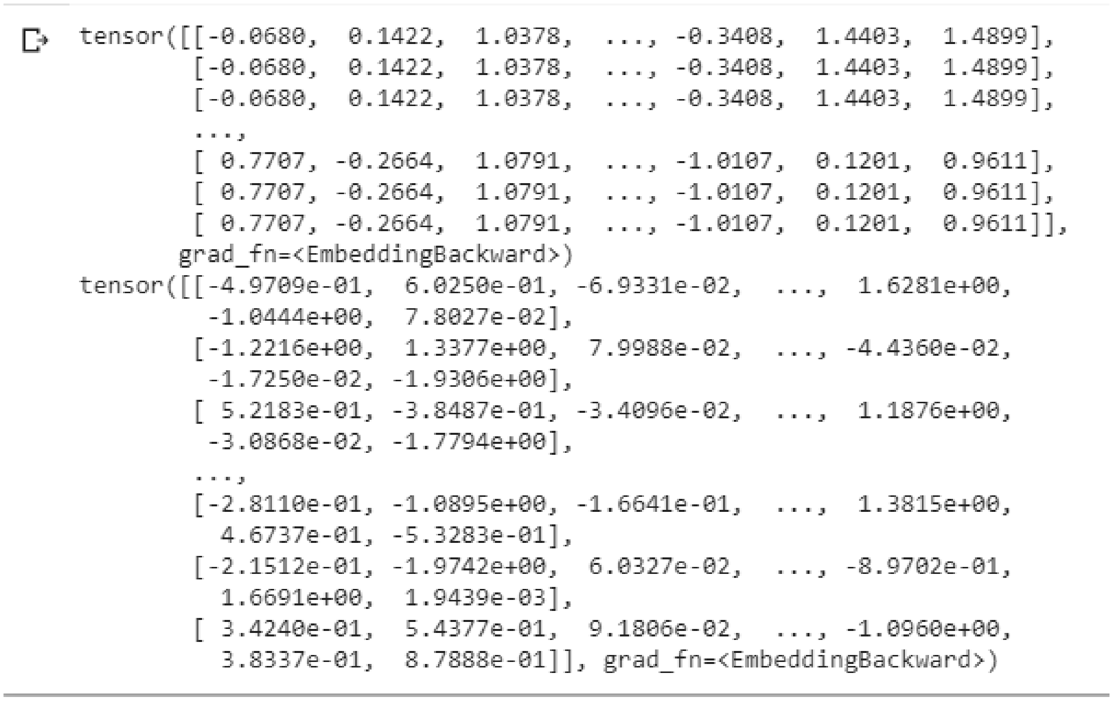
After cleaning the data and appending the tags to a list and removing the duplicates, the resulting list is shown below. The first value of the list corresponds to movie id 0, the next corresponding to movie id 1 and so on and so forth.



```
Out[103]: ['adventure witty does in movie not star comedy tã®a leoni tom children fantasy animation family this feature disney animated
          hanks pixar watched computer ',
           'adventure herds of robin book travel children fantasy game zathura adapted scary cgi childhood recaptured see time from boa
          rd williams also animals ',
           'romance is comedy old people that funny actually ',
           'romance comedy drama ',
           'comedy ',
           'ending lame de al niro thriller portman pacino michael natalie kilmer realistic action mann val robert crime ',
           'chick duckling remadeas ugly remake romance comedy based downstairs yourself on a 1995 play sabrina paris remakeof upstairs
          ford kinnear finding harrison greg 1954 flick ',
           'adventure mark children from twain adapted author book ',
           'action ',
           'as adventure 007 killer assassin thriller series action bond murder protagonist ',
           'white romance comedy house us drama president ',
           'horror comedy ',
           'adventure children animation ',
           'real based biopic nixon person richard on character us drama president ',
           'romance action adventure pirates ',
           'pink sharon niro de stone martin violent scorsese pesci in robert character crime development drama joe ',
```

Figure 17 data after cleaning

The next image shows the initial component matrix which is created by using a embedding size of 100

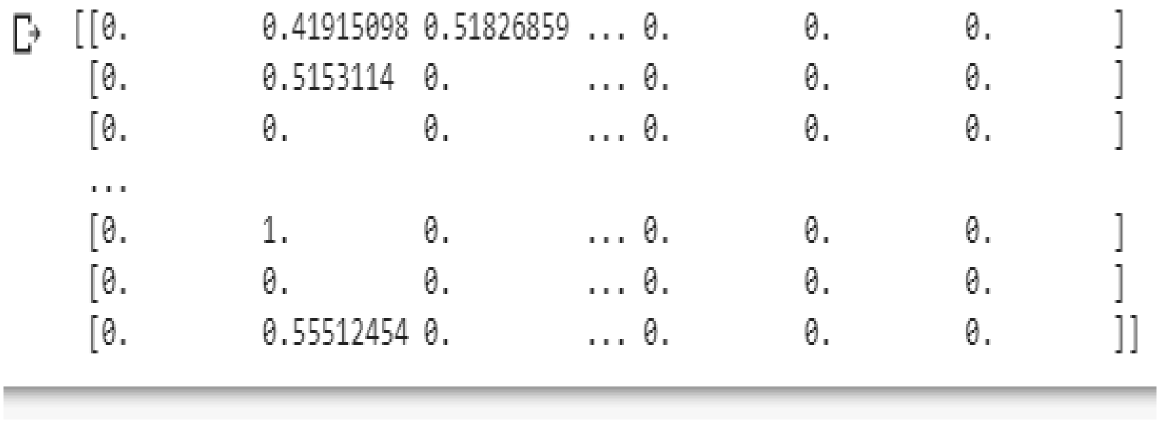


```
tensor([[-0.0680,  0.1422,  1.0378,  ..., -0.3408,  1.4403,  1.4899],
        [-0.0680,  0.1422,  1.0378,  ..., -0.3408,  1.4403,  1.4899],
        [-0.0680,  0.1422,  1.0378,  ..., -0.3408,  1.4403,  1.4899],
        ...,
        [ 0.7707, -0.2664,  1.0791,  ..., -1.0107,  0.1201,  0.9611],
        [ 0.7707, -0.2664,  1.0791,  ..., -1.0107,  0.1201,  0.9611],
        [ 0.7707, -0.2664,  1.0791,  ..., -1.0107,  0.1201,  0.9611]],
       grad_fn=<EmbeddingBackward>)
tensor([[-4.9709e-01,  6.0250e-01, -6.9331e-02,  ...,  1.6281e+00,
         -1.0444e+00,  7.8027e-02],
        [-1.2216e+00,  1.3377e+00,  7.9988e-02,  ..., -4.4360e-02,
         -1.7250e-02, -1.9306e+00],
        [ 5.2183e-01, -3.8487e-01, -3.4096e-02,  ...,  1.1876e+00,
         -3.0868e-02, -1.7794e+00],
        ...,
        [-2.8110e-01, -1.0895e+00, -1.6641e-01,  ...,  1.3815e+00,
          4.6737e-01, -5.3283e-01],
        [-2.1512e-01, -1.9742e+00,  6.0327e-02,  ..., -8.9702e-01,
          1.6691e+00,  1.9439e-03],
        [ 3.4240e-01,  5.4377e-01,  9.1806e-02,  ..., -1.0960e+00,
          3.8337e-01,  8.7888e-01]], grad_fn=<EmbeddingBackward>)
```

Figure 18 The initial component matrix

## 4.2 RESULTS AFTER USING TF-IDF ALGORITHM

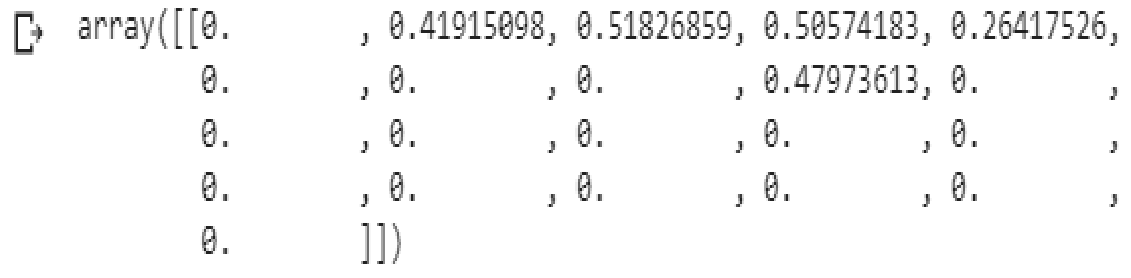
The tf-idf value matrix look like the below image



```
[[0.         0.41915098 0.51826859 ... 0.         0.         0.        ]
 [0.         0.5153114  0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         1.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.55512454 0.         ... 0.         0.         0.        ]]
```

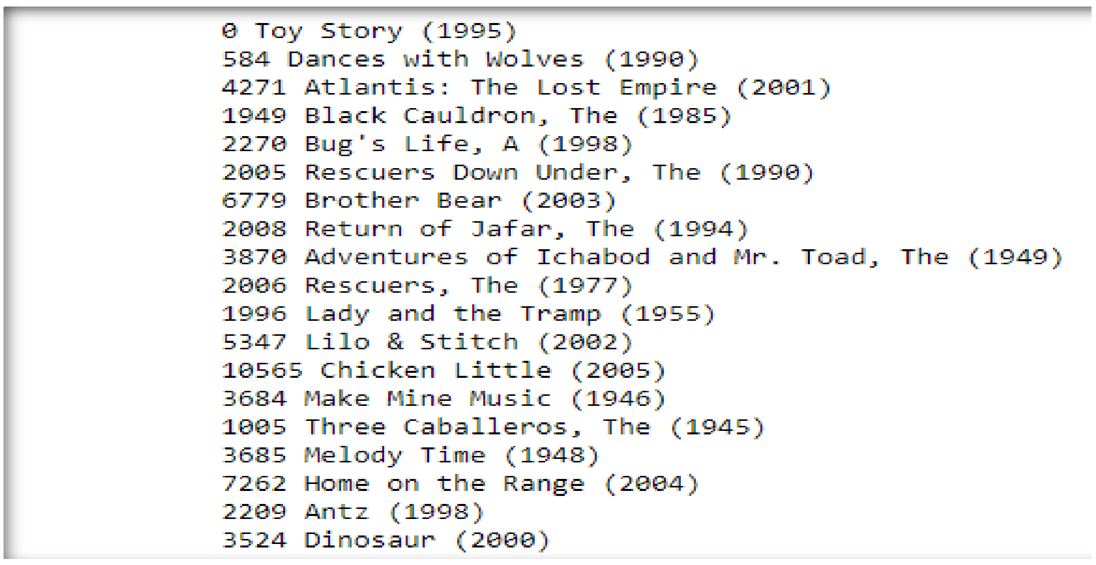
Figure 19 The values of tf-idf vector

The complete first row of the above matrix is shown below,



```
array([[0.        , 0.41915098, 0.51826859, 0.50574183, 0.26417526,
        0.        , 0.        , 0.        , 0.47973613, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        ]])
```

Figure 20 the complete first row of the tf-idf vector



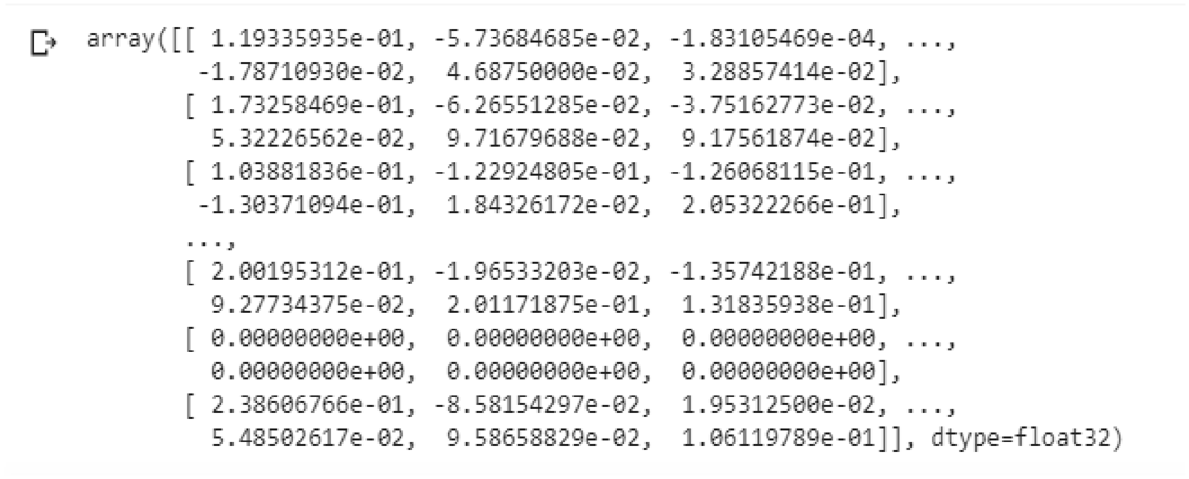
```
0 Toy Story (1995)
584 Dances with Wolves (1990)
4271 Atlantis: The Lost Empire (2001)
1949 Black Cauldron, The (1985)
2270 Bug's Life, A (1998)
2005 Rescuers Down Under, The (1990)
6779 Brother Bear (2003)
2008 Return of Jafar, The (1994)
3870 Adventures of Ichabod and Mr. Toad, The (1949)
2006 Rescuers, The (1977)
1996 Lady and the Tramp (1955)
5347 Lilo & Stitch (2002)
10565 Chicken Little (2005)
3684 Make Mine Music (1946)
1005 Three Caballeros, The (1945)
3685 Melody Time (1948)
7262 Home on the Range (2004)
2209 Antz (1998)
3524 Dinosaur (2000)
```

Figure 21 tf-idf based results

The above image shows the results, when the query movie is “toy story” and tf-idf based recommendation technique is used.

## 4.3 RESULTS AFTER USING WORD2VEC ALGORITHM

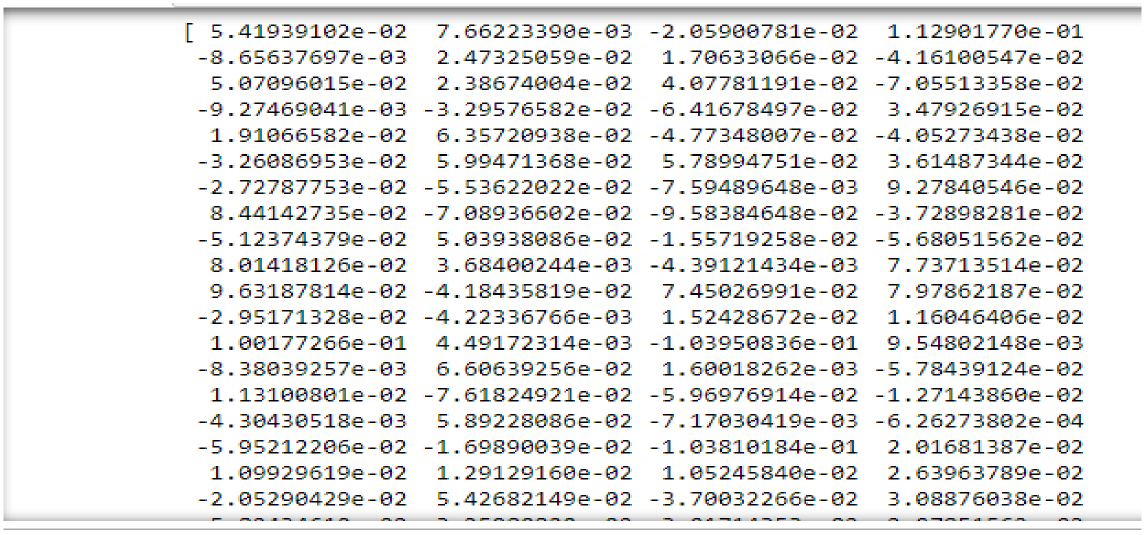
The word2vec matrix after running word2vec model on the list is shown below,



```
array([[ 1.19335935e-01, -5.73684685e-02, -1.83105469e-04, ...,
        -1.78710930e-02,  4.68750000e-02,  3.28857414e-02],
       [ 1.73258469e-01, -6.26551285e-02, -3.75162773e-02, ...,
         5.32226562e-02,  9.71679688e-02,  9.17561874e-02],
       [ 1.03881836e-01, -1.22924805e-01, -1.26068115e-01, ...,
        -1.30371094e-01,  1.84326172e-02,  2.05322266e-01],
       ...,
       [ 2.00195312e-01, -1.96533203e-02, -1.35742188e-01, ...,
         9.27734375e-02,  2.01171875e-01,  1.31835938e-01],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
       [ 2.38606766e-01, -8.58154297e-02,  1.95312500e-02, ...,
         5.48502617e-02,  9.58658829e-02,  1.06119789e-01]], dtype=float32)
```

Figure 22 the word2vec matrix

The word2vec matrix of the recommended movies for the query movie “toy story” is shown below, the first row corresponds to the query movie that is “toy story”, the next row corresponds to the nearest distant movie according to the word2vec valued vector so on and so forth. As each row is 300 dimensional, so the below image shows only a part of the first row.

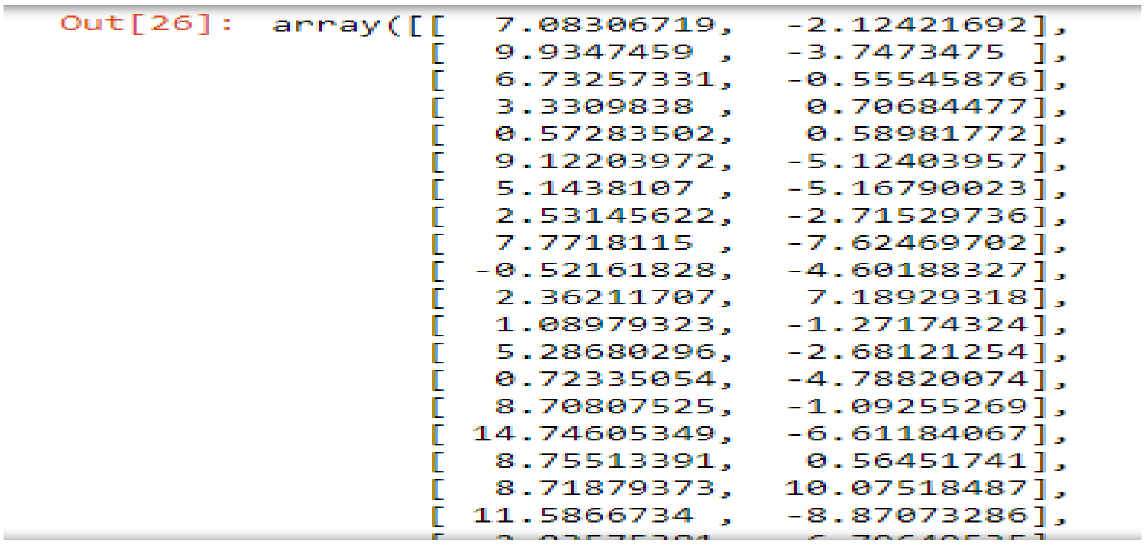


```
[ 5.41939102e-02  7.66223390e-03 -2.05900781e-02  1.12901770e-01
 -8.65637697e-03  2.47325059e-02  1.70633066e-02 -4.16100547e-02
  5.07096015e-02  2.38674004e-02  4.07781191e-02 -7.05513358e-02
 -9.27469041e-03 -3.29576582e-02 -6.41678497e-02  3.47926915e-02
  1.91066582e-02  6.35720938e-02 -4.77348007e-02 -4.05273438e-02
 -3.26086953e-02  5.99471368e-02  5.78994751e-02  3.61487344e-02
 -2.72787753e-02 -5.53622022e-02 -7.59489648e-03  9.27840546e-02
  8.44142735e-02 -7.08936602e-02 -9.58384648e-02 -3.72898281e-02
 -5.12374379e-02  5.03938086e-02 -1.55719258e-02 -5.68051562e-02
  8.01418126e-02  3.68400244e-03 -4.39121434e-03  7.73713514e-02
  9.63187814e-02 -4.18435819e-02  7.45026991e-02  7.97862187e-02
 -2.95171328e-02 -4.22336766e-03  1.52428672e-02  1.16046406e-02
  1.00177266e-01  4.49172314e-03 -1.03950836e-01  9.54802148e-03
 -8.38039257e-03  6.60639256e-02  1.60018262e-03 -5.78439124e-02
  1.13100801e-02 -7.61824921e-02 -5.96976914e-02 -1.27143860e-02
 -4.30430518e-03  5.89228086e-02 -7.17030419e-03 -6.26273802e-04
 -5.95212206e-02 -1.69890039e-02 -1.03810184e-01  2.01681387e-02
  1.09929619e-02  1.29129160e-02  1.05245840e-02  2.63963789e-02
 -2.05290429e-02  5.42682149e-02 -3.70032266e-02  3.08876038e-02
```

Figure 23 part of a 300 dimensional vector

We have applied PCA -Principal component analysis to reduce the above matrix to a 2 dimensional matrix from the above 300 dimensional matrix. We have done this to show how word2vec works to show all the nearby movies to the query movie. As we cannot plot a 300 dimensional vector, so we reduced it to a 2 dimensional vector using pca.

The pca matrix after dimensionality reduction is shown below,



```
Out[26]:  array([[  7.08306719,   -2.12421692],
                 [  9.9347459 ,   -3.7473475 ],
                 [  6.73257331,   -0.55545876],
                 [  3.3309838 ,    0.70684477],
                 [  0.57283502,    0.58981772],
                 [  9.12203972,   -5.12403957],
                 [  5.1438107 ,   -5.16790023],
                 [  2.53145622,   -2.71529736],
                 [  7.7718115 ,   -7.62469702],
                 [ -0.52161828,   -4.60188327],
                 [  2.36211707,    7.18929318],
                 [  1.08979323,   -1.27174324],
                 [  5.28680296,   -2.68121254],
                 [  0.72335054,   -4.78820074],
                 [  8.70807525,   -1.09255269],
                 [ 14.74605349,   -6.61184067],
                 [  8.75513391,    0.56451741],
                 [  8.71879373,   10.07518487],
                 [ 11.5866734 ,   -8.87073286],
```

Figure 24 word2vec vector after dimensionality reduction

The plotted graph below shows that all the recommended movies using word2vec are nearby to the query movie (here it is “toy story”)

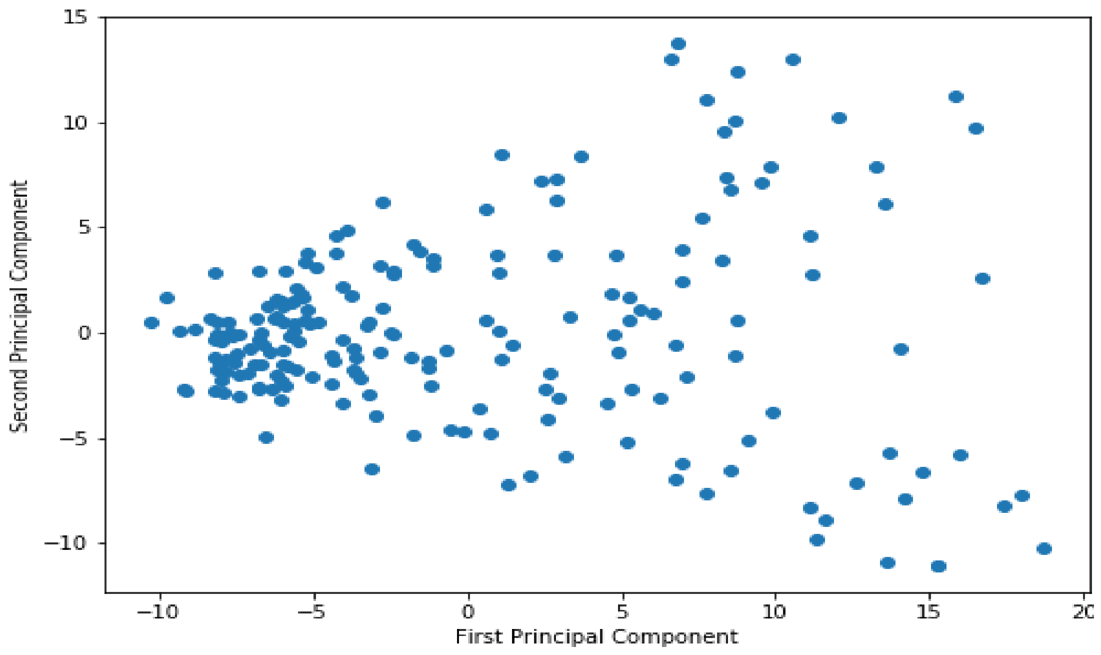
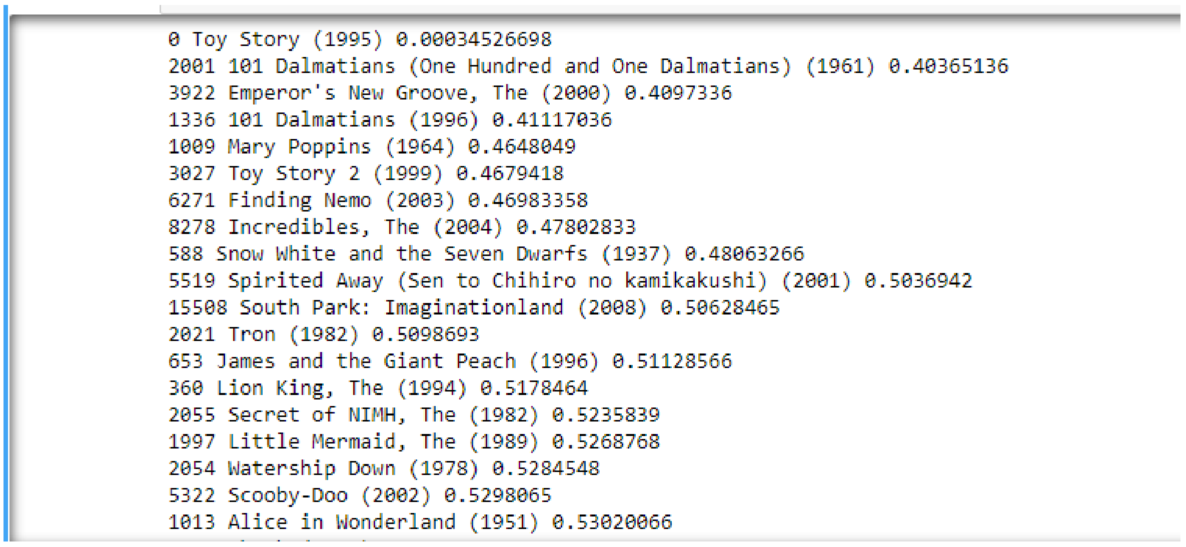


Figure 25 the spatial positions of the movies

The list of recommended movies is shown below, the first being the query movie itself, that is("toy story"). We have recommended 200 nearest movies, but below shows only a part of the recommended movies.



```
0 Toy Story (1995) 0.00034526698
2001 101 Dalmatians (One Hundred and One Dalmatians) (1961) 0.40365136
3922 Emperor's New Groove, The (2000) 0.4097336
1336 101 Dalmatians (1996) 0.41117036
1009 Mary Poppins (1964) 0.4648049
3027 Toy Story 2 (1999) 0.4679418
6271 Finding Nemo (2003) 0.46983358
8278 Incredibles, The (2004) 0.47802833
588 Snow White and the Seven Dwarfs (1937) 0.48063266
5519 Spirited Away (Sen to Chihiro no kamikakushi) (2001) 0.5036942
15508 South Park: Imaginationland (2008) 0.50628465
2021 Tron (1982) 0.5098693
653 James and the Giant Peach (1996) 0.51128566
360 Lion King, The (1994) 0.5178464
2055 Secret of NIMH, The (1982) 0.5235839
1997 Little Mermaid, The (1989) 0.5268768
2054 Watership Down (1978) 0.5284548
5322 Scooby-Doo (2002) 0.5298065
1013 Alice in Wonderland (1951) 0.53020066
```

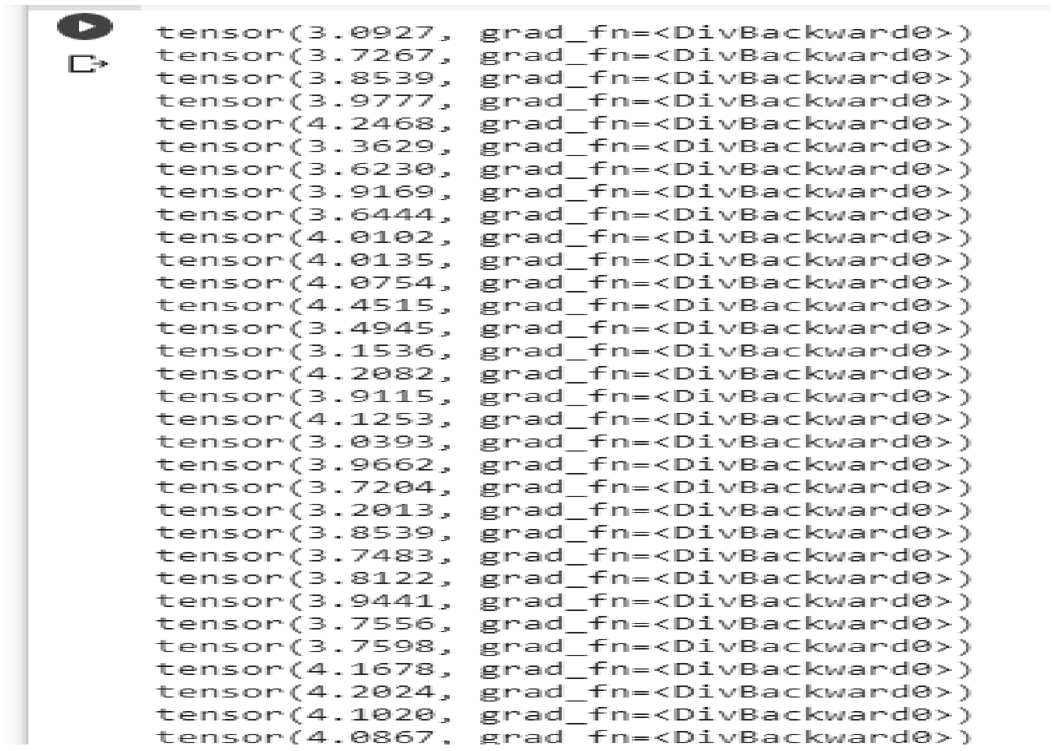
Figure 26 word2vec based results

4.4 RESULTS OF TF-IDF WEIGHTED WORD2VEC MODEL WITH PREDICTED

RATING USING SINGULAR VALUE DECOMPOSITION ALGORITHM

Below is the image showing a fragment of the predicted average movie ratings, we first stored the probable rating for every movie given by every user by minimizing the loss as far as possible, then we added the ratings for a particular movie given by all users and divided it by the number of users.

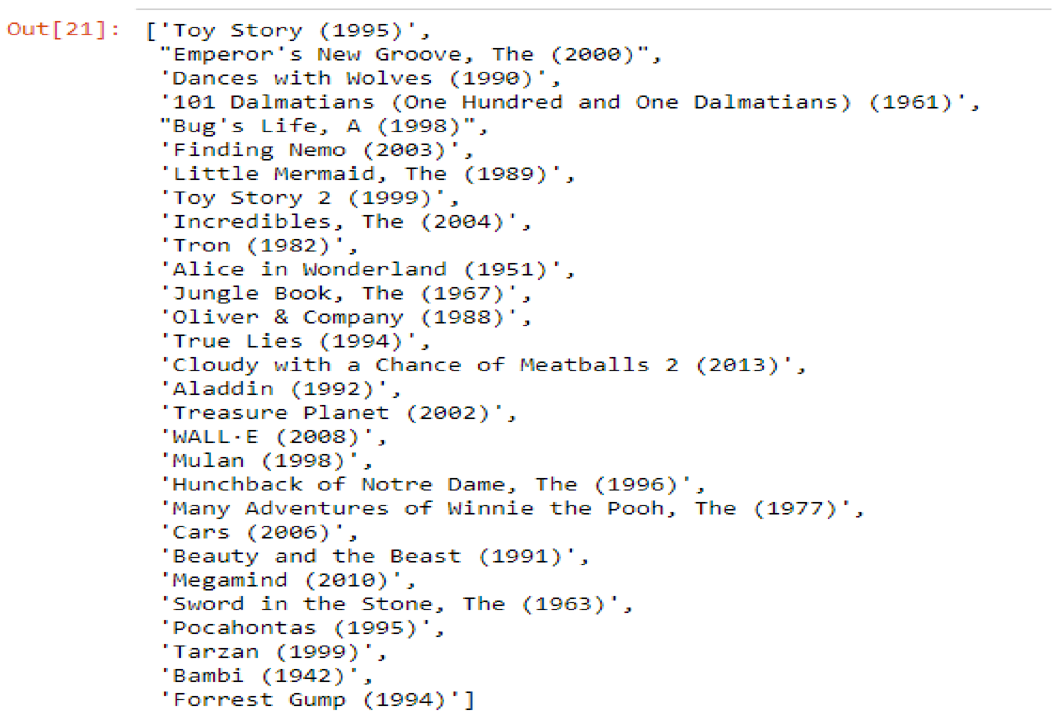
The first value in the below picture is the average predicted rating for movie 1, then next for movie 2 and so on and so forth.



```
tensor(3.0927, grad_fn=<DivBackward0>)
tensor(3.7267, grad_fn=<DivBackward0>)
tensor(3.8539, grad_fn=<DivBackward0>)
tensor(3.9777, grad_fn=<DivBackward0>)
tensor(4.2468, grad_fn=<DivBackward0>)
tensor(3.3629, grad_fn=<DivBackward0>)
tensor(3.6230, grad_fn=<DivBackward0>)
tensor(3.9169, grad_fn=<DivBackward0>)
tensor(3.6444, grad_fn=<DivBackward0>)
tensor(4.0102, grad_fn=<DivBackward0>)
tensor(4.0135, grad_fn=<DivBackward0>)
tensor(4.0754, grad_fn=<DivBackward0>)
tensor(4.4515, grad_fn=<DivBackward0>)
tensor(3.4945, grad_fn=<DivBackward0>)
tensor(3.1536, grad_fn=<DivBackward0>)
tensor(4.2082, grad_fn=<DivBackward0>)
tensor(3.9115, grad_fn=<DivBackward0>)
tensor(4.1253, grad_fn=<DivBackward0>)
tensor(3.0393, grad_fn=<DivBackward0>)
tensor(3.9662, grad_fn=<DivBackward0>)
tensor(3.7204, grad_fn=<DivBackward0>)
tensor(3.2013, grad_fn=<DivBackward0>)
tensor(3.8539, grad_fn=<DivBackward0>)
tensor(3.7483, grad_fn=<DivBackward0>)
tensor(3.8122, grad_fn=<DivBackward0>)
tensor(3.9441, grad_fn=<DivBackward0>)
tensor(3.7556, grad_fn=<DivBackward0>)
tensor(3.7598, grad_fn=<DivBackward0>)
tensor(4.1678, grad_fn=<DivBackward0>)
tensor(4.2024, grad_fn=<DivBackward0>)
tensor(4.1020, grad_fn=<DivBackward0>)
tensor(4.0867, grad_fn=<DivBackward0>)
```

Figure 27 the predicted average ratings

Below is the image of a part of the list of movies recommended using our proposed algorithm, we had given equal weight to the tf-idf word2vec and the rating vector,



```
Out[21]: ['Toy Story (1995)',
          "Emperor's New Groove, The (2000)",
          'Dances with Wolves (1990)',
          '101 Dalmatians (One Hundred and One Dalmatians) (1961)',
          "Bug's Life, A (1998)",
          'Finding Nemo (2003)',
          'Little Mermaid, The (1989)',
          'Toy Story 2 (1999)',
          'Incredibles, The (2004)',
          'Tron (1982)',
          'Alice in Wonderland (1951)',
          'Jungle Book, The (1967)',
          'Oliver & Company (1988)',
          'True Lies (1994)',
          'Cloudy with a Chance of Meatballs 2 (2013)',
          'Aladdin (1992)',
          'Treasure Planet (2002)',
          'WALL·E (2008)',
          'Mulan (1998)',
          'Hunchback of Notre Dame, The (1996)',
          'Many Adventures of Winnie the Pooh, The (1977)',
          'Cars (2006)',
          'Beauty and the Beast (1991)',
          'Megamind (2010)',
          'Sword in the Stone, The (1963)',
          'Pocahontas (1995)',
          'Tarzan (1999)',
          'Bambi (1942)',
          'Forrest Gump (1994)']
```

Figure 28 results of the proposed algorithm

# CHAPTER 5

# SOFTWARE REQUIREMENT

## 5.1 Software description

**Python:** is a high level programming language, and an interpreted language, which means that most of the implementations execute the instructions directly and not by previously compiling into machine-language. Here the interpreter translates the program directly into machine code. The language was created by Guido van Rossum and was released in 1991. Multiple programming paradigms are supported in python like functional, object oriented and procedural programming.

**Anaconda (python distribution):** is used for scientific computing like machine learning applications, data science applications, to process large scale data, predictive analysis etc. It is free and a open source distribution of Python and R. It simplifies packet management which is managed by system conda. The Anaconda distribution comes with more fifteen hundred and more packages and virtual environment manger, it also includes a graphical user interface, known as Anaconda Navigator, which serves as a replacement to the command line interface.

**Jupyter Notebook:** is a client-server architecture based application, where we can run and also edit notebook documents in a web browser. The notebook document consists of code (e.g Python) and also text elements like figures, equations, paragraphs, links etc. it can be accessed by a computer without internet access, or it can be accessed by a remote server through internet. Jupyter notebook also has a control panel which shows all the local files, allowing opening of notebook documents.

**Google colaboratory:** it is free research related tool used in machine learning in research and education. It has a jupyter notebook type environment and requires no setup. It works in all major browsers like safari, chrome, firefox. It is same as the jupyter notebook because it is based on it, and above it colaboratory allows us to use and share the jupyter notebook without actually downloading it and the code runs on Google’s server and not on the local machine.

## 5.2 Packages:

**Pandas:** is a library for Python used for data analysis and manipulation. It is free and it offers data structures and lets us operate on the data by manipulating the tables and time series.

**Scikit-Learn:** it is also a free machine learning libraries for Python. It contains various regression, clustering and classification algorithms, which includes algorithm like random forest, support vector machines, k means etc. it is designed in such a way that it operates with python libraries such as NumPy and Scipy.

**Numpy:** is a library for python, which helps to operate with large multidimensional matrices and arrays, it also has many mathematical functions included in it.

**Gensim:** it is a library used in unsupervised learning and in natural language processing. It is designed to work with large data sets and incremental online algorithms, it is different than others machine learning libraries, which targets only in-memory processing.

**Pytorch:** is a scientific computing framework and machine learning library that consist of many deep learning algorithms using the scripting language of Lua which has an C implementation underneath it. It provides a N-dimensional array also known as tensor, in which we can apply slicing, transposing, type-casting, resizing etc. it also supports matrix-multiplication with vector and matrices, dot product etc.

**Matplotlib:** is a library used for plotting in Python and Numpy. It provides an API, so that plots can be embedded using graphical user interface like GTK+, QT etc.

# CHAPTER 6

# CONCLUSION AND FUTURE WORK

The above results show the positives of using Tf idf-weighted word2vec algorithm over only tf-idf or only word2vec algorithms. Using the semantic based filtering like word2vec we can know the user’s behavior and thinking and thus predict the future needs and by using the frequency based filtering we get the results which gives recommendations based on the text of the title and previously added tags by the user. Better results are produced when we add the average rating value to the previous idf-weighted vector, which enables the user to give preference to ratings or semantics or a mixture of the previous.

As a future work, we can increase the efficiency by studying the image of the cover picture of the movie and classifying the image property using convolutional neural network and then give recommendations based on the similarity of the query movie image.

Many current techniques in collaborative filtering are unable to handle large dataset and also many user gives very few ratings and this problem cannot be easily dealt with, so here we could use Probabilistic Matrix Factorization [7]. This particular algorithm perms very well in datasets which are very large and is sparse and imbalanced.

We could clean the data accordingly so that by applying clustering algorithms like k-means we can find all the movies belonging to a particular genre.

# CHAPTER 7

# REFERENCES

[1] Priyanka Meel, Agniva Goswami. “Inverse Document Frequency-Weighted Word2VEc Model to Recommend Apparels”, 2019 6th International Conference on Signal Procesing and Integrated Networks (SPIN), 2019

[2] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*(2014).

[3] R. C. Bagher, H. Hassanpour, and H. Mashayekhi, “User trends modeling for a content-based recommender system,” *Expert Syst. Appl.*, vol. 87, pp. 209–219, 2017.

[4] M. S. Tajbakhsh and J. Bagherzadeh, “Microblogging hash tag recommendation system based on semantic TF-IDF: Twitter use case,” *Proc. - 2016 4th Int. Conf. Futur. Internet Things Cloud Work. W-FiCloud 2016*, pp. 252–257, 2016.

[5] G. Carullo, A. Castiglione, and A. De Santis, “Friendship recommendations in online social networks,” *Proc. - 2014 Int. Conf. Intell. Netw. Collab. Syst. IEEE INCoS 2014*, pp. 42–48, 2014.

[6] https://www.youtube.com/watch?v=ZspR5PZemcs

[7] Mnih, Andriy, and Ruslan R. Salakhutdinov. "Probabilistic matrix factorization." *Advances in neural information processing systems*. 2008

[8] https://en.wikipedia.org/wiki/Singular_value_decomposition

[9] J. Hannon, M. Bennett, and B. Smyth, “Recommending twitter users to follow using content and collaborative filtering approaches,” *Proc. fourth ACM Conf. Recomm. Syst. - RecSys ’10*, p. 199,2010

[10] Banerjee, Sudipto; Roy, Anindya (2014), Linear Algebra and Matrix Analysis for Statistics, Texts in Statistical Science (1st ed.), Chapman and Hall/CRC, ISBN 978-1420095388

[11] Trefethen, Lloyd N.; Bau III, David (1997). Numerical linear algebra. Philadelphia: Society for Industrial and Applied Mathematics. ISBN 978-0-89871-361-9.

[12] S. Aral and D. Walker, “Identifying Influential and Susceptible Members of Social Networks,” *Science*, vol. 337, no. 6092, pp. 337-341, June 2012.

[13] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” IEEE Trans. Knowl. Data Eng., vol. 17, no. 6, pp. 734–749, 2005.

[14] H. Kwak, C. Lee, H. Park, and S. Moon, “What is Twitter, a Social Network or a News Media?”, Int. World Wide Web Conf. Comm., pp. 1–10, 2010.

[15] Aggarwal, Charu C. (2016). Recommender Systems: The Textbook. Springer-ISBN 9783319296579

[16] Appliedaicourse.com

[17] https://deeplearning4j.org

[18] Demmel, James; Kahan, William (1990). "Accurate singular values of bidiagonal matrices". SIAM Journal on Scientific and Statistical Computing. 11 (5): 873–912. CiteSeerX 10.1.1.48.3740. doi:10.1137/0911052.

[19] Horn, Roger A.; Johnson, Charles R. (1985). "Section 7.3". Matrix Analysis. Cambridge University Press. ISBN 978-0-521-38632-6

[20] Horn, Roger A.; Johnson, Charles R. (1991). "Chapter 3". Topics in Matrix Analysis. Cambridge University Press. ISBN 978-0-521-46713-1

[21] Golub, Gene H.; Van Loan, Charles F. (1996). Matrix Computations (3rd ed.). Johns Hopkins. ISBN 978-0-8018-5414-9

[22] GSL Team (2007). "§14.4 Singular Value Decomposition". GNU Scientific Library. Reference Manual

[23] Halldor, Bjornsson and Venegas, Silvia A. (1997). "A manual for EOF and SVD analyses of climate data". McGill University, CCGCR Report No. 97-1, Montréal, Québec, 52pp.

[24] Hansen, P. C. (1987). "The truncated SVD as a method for regularization". BIT. 27 (4): 534–553. doi:10.1007/BF01937276.

[25] Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007), "Section 2.6", Numerical Recipes: The Art of Scientific Computing (3rd ed.), New York: Cambridge University Press, ISBN 978-0-521-88068-8

[26] Golub, Gene H.; Kahan, William (1965). "Calculating the singular values and pseudo-inverse of a matrix". Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis. 2 (2): 205–224. Bibcode:1965SJNA....2..205G. doi:10.1137/0702016. JSTOR 2949777.

[27] Samet, H. (2006). Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann. ISBN 978-0-12-369446-1.

[28] Strang G. (1998). "Section 6.7". Introduction to Linear Algebra (3rd ed.). Wellesley-Cambridge Press. ISBN 978-0-9614088-5-5

[29] Stewart, G. W. (1993). "On the Early History of the Singular Value Decomposition". SIAM Review. 35 (4): 551–566. CiteSeerX 10.1.1.23.1831. doi:10.1137/1035134. JSTOR 2132388

[30] Wall, Michael E.; Rechtsteiner, Andreas; Rocha, Luis M. (2003). "Singular value decomposition and principal component analysis". In D.P. Berrar; W. Dubitzky; M. Granzow (eds.). A Practical Approach to Microarray Data Analysis. Norwell, MA: Kluwer. pp. 91–109.

[31] Yoav Goldberg and Omer Levy, "word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method", arXiv:1402.3722v1, Feb 2014

[32] P. Alencar and D. Cowan, "The use of machine learning algorithms in recommender systems : A systematic review," Expert Syst. Appl., vol. 97, pp. 205–227, 2018.

[33] M. Razghandi and S. A. H. Golpaygani, "A Context-Aware and User Behavior-Based Recommender System with,Regarding Social Network Analysis," *2017 IEEE 14th Int. Conf. E-bus. Eng.*, pp. 208–213, 2017