
# CS425: Algorithms for Web-Scale Data Project

# The Effect of Movie Script Similarity on Movie Recommendation

## Final Report

**Group 14, Members:**
21502312 - Eliz Tekcan
21402334 - Ezgi Çakır
21501348 - Mehmet Enes Keleş
21401258 - Metehan Kaya

# Table of Contents

# 1. Introduction

Internet allows people to access abundant resources online, and there are various enormous movie collections accessible online. With the increase of available information, a new issue occurs, that is how to select the ones that a particular user is interested in. Recommendation systems seek to determine the ratings or preference of a user based on available data in order to make suggestions. In our project, we tried to identify similar movies based on the content of their scripts, genres and also used movie ratings data and combined our results to make suggestions.

To determine similarity between movie scripts we used various methods, including cosine similarity based on tf-idf, jaccard similarity based on genres, kshingling, minhashing, and locality sensitive hashing, word2vec averages and k-means clustering using word2vec vector representations of movie scripts.

As our validation method, we used collaborative filtering with interpolation weights, and also SVD with stochastic gradient descent (SGD).

# 2. Problem Description

In this project, we basically focused on movie recommendation systems. Currently, Netflix recommendation system relies on user ratings given to movies. Apart from the Netflix recommendation system, we aim to observe whether movie script similarity can be used for recommendation of movies rather than user ratings. Furthermore, we try to observe if web-scale algorithms are sufficient to find similarity of movie pairs by processing their scripts.

# 3. Methods

## 3.1. Data Collection

### 3.1.1. Imsdb Movie Scripts

Imsdb Movie Scripts dataset contains 1,093 movie scripts collected from imsdb.com, and each script has a separate text file [1].

### 3.1.2. MovieLens 20M Dataset

MovieLens 20M Dataset consists of 20 million ratings and 465,000 tag applications applied to 27,000 movies by 138,000 users [2]. We used this dataset for collaborative filtering which is a method for predicting interests of a user using information from many others. For 1,093 movies that we have the scripts of, we tried to gather the rating information from 20 million movies, and we realized that only 808 of them were common, and our final dataset had 808 movies with ratings and scripts.

### 3.1.3. OMDb API

OMDb API is a web service to obtain movie information. We used this API to collect genre information of the 808 movies that we already gathered scripts and ratings of and finalized our data collection [3].

## 3.2. Algorithms

### 3.2.1. Collaborative Filtering

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

Eqn. 1: Collaborative Filtering Formula

For collaborative filtering algorithm, the formula above is used (Eqn. 1). In this formula, $r_{xi}$ is the rating given by user x to movie i. $b_{xi}$ is the baseline estimate for

$r_{xi}$. $s_{ij}$ is similarity between movie i and movie j. N(i;x) includes in ratings of user x for all movies in our calculation. As a result, all movies rated by user x contribute to the rating estimation of movie i. While we were implementing this algorithm, we used ratings data in order to calculate baseline estimates.

Sij is generated by various combination of 4 different algorithms. These algorithms will be described below in detail. We multiplied different coefficients with outputs of four algorithms. By doing this, we gave different weight for each output of each algorithm. Since the sum of coefficients is 1 and outputs of all implemented algorithms is within 0 and 1, the similarity between two movies is within 0 and 1. It will be discussed which coefficients are given in discussion. We compared our different results with the result of our validation method.

### 3.2.2. Cosine Similarity based on TF-IDF

TF-IDF is a numerical statistic that reflects the importance of a word for a document. TF stands for term frequency which means frequency of each word in the document. IDF stands for inverse document frequency that reflects how important this term is for the document. TF-IDF score of a word is calculated by multiplying TF and IDF scores of that word. Generally, stop words have the lowest TF-IDF scores, for example, but, however, etc. Special terminologies and character names of movies have higher TF-IDF score. When we run this algorithm for all movie scripts, it outputs a TF-IDF score vector for each script which are composed of the score of each word that is detected in any scripts. After obtaining score vectors, cosine similarity is calculated for each script pair. It is within the range [0,1].

### 3.2.3. Jaccard Similarity based on Genres

For this method, we represented each script with the list of their genres and calculated jaccard similarities for each movie pair based on this list we obtained. The following formula is used for calculation.

$$sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$

C1 and C2 represents the genre lists. The output was within the range [0,1]. Where 0 means no similarity and 1 means the highest similarity.

### 3.2.4. K-shingling, Minhashing and Locality Sensitive Hashing

As our first step for this approach, we obtained the set of words of length 3 (3-shingles) of each movie script. We used these shingles to create our vocabulary. We represented each movie with respect to this vocabulary. For each element in our vocabulary if it is present in a movie script a 1 is added to the corresponding index, otherwise a 0 is added. We only stored the indices that contained 1s for each movie in order to represent sparse matrix which in result asymptotically reduced the time and space complexity. We used 100 hash functions to obtain our signature matrix. After computing min-hash signatures, we applied locality sensitive hashing to determine candidate pairs. We tested our results with different band and row values. The output was a matrix where each element contained a 0 or a 1 where 0 means corresponding two movies do not fall into the same bucket, 1 means that movie script pairs fall into the same bucket.

### 3.2.5. Word2Vec Average

Word2vec is a combination of two techniques, continuous bag of words and skip-gram model. In essence, this algorithm maps words into a 300 dimensional space with regards to their semantic meanings. To this end semantically close words mapped into close locations such as “king” and “queen”, “horse” and “donkey” (Figure 1). Instead of implementing the algorithm ourselves we used pretrained vectors on wikipedia data which is published by Facebook Research. [fasttext pretrained vectors linki] For every movie, we calculated word2vec averages of words that are present in the script. We employed this method because it is widely used among many natural language processing papers to improve semantic similarity calculations. Consequently, we calculated cosine distance between word2vec average vectors which are represented in 300 dimensional space.

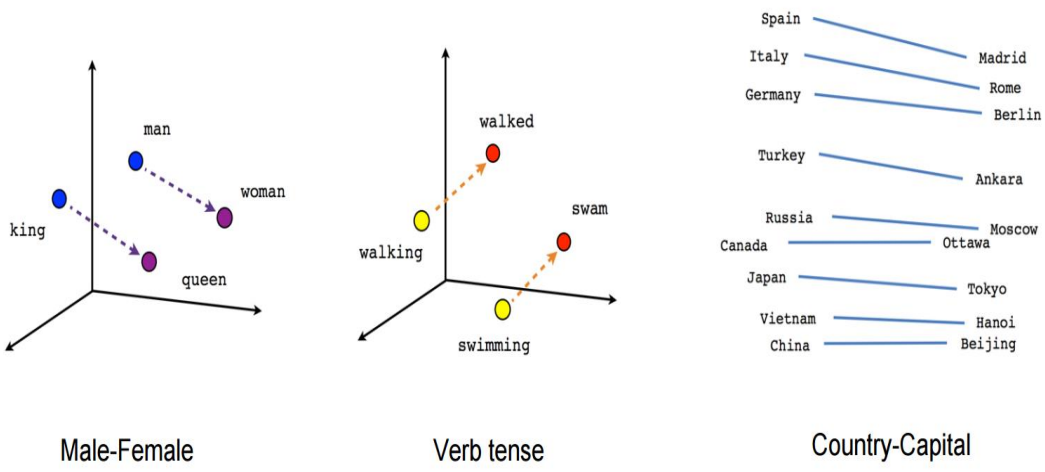



Figure 1: Word2Vec Description

### 3.2.6. K-means Clustering using Word2Vec Representations

In an effort to observe the correlation of movie genres and word vectors we depicted each movie as a set of word vectors with words corresponding to words in the script. For each set we calculated 10 centers by applying k-means clustering. We found the most similar words in euclidean distance for each center of each movie which in result give us the opportunity to represent movies using only 10 summarizing words. Using summarizing words as features we differentiated movies into 3 categories and visualized the result in 3D space after tSNE dimensionality reduction applied. An example output with 100 movie scripts can be seen below (Figure 2). Each color represents different clusters, and each node represents movie scripts. From link provided in the references section, navigating through the nodes information regarding the movies including movie names and genres can be seen [5].

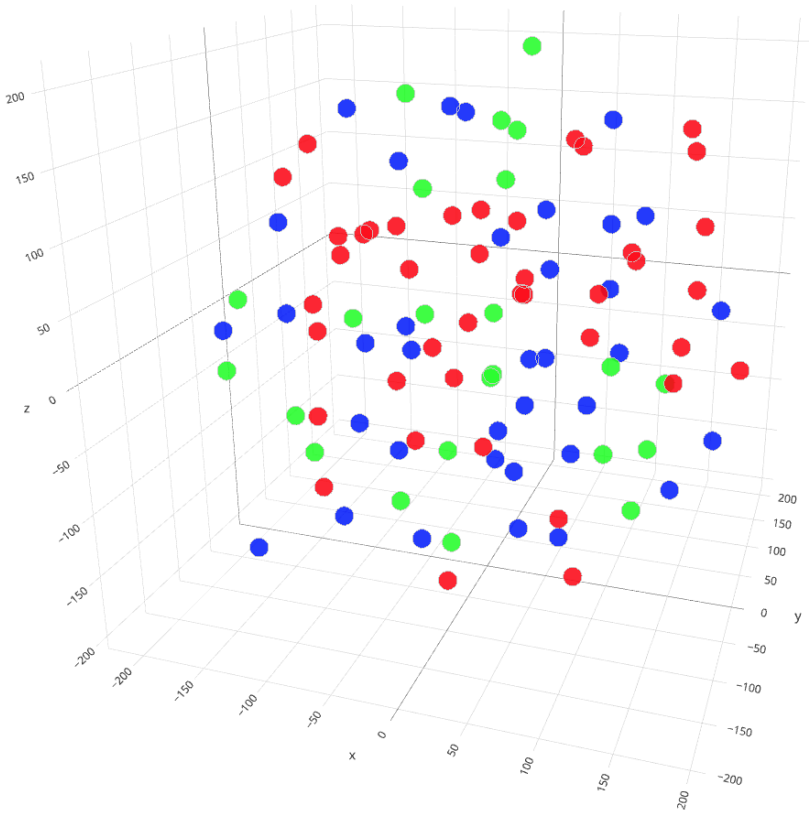



Figure 2: Visualization of Movies with Respect to Summarizing Words

## 3.3. Validation Methods

### 3.3.1. Collaborative Filtering with Interpolation Weights

$$\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i;x)} w_{ij}\left(r_{xj} - b_{xj}\right)$$

Prediction formula for this algorithm can be seen in above equation. The matrix b and the matrix r are mentioned in earlier. The matrix w is a matrix representing correlation between movies which is unknown in the beginning. In order to optimize predictions with respect to the matrix w, we need to use a metric that depends on difference between the ground truth ratings and predictions which is called error metric. Using sum square error (SSE) as error metric, the cost function is formulated as follows:

$$J(w) = \sum_{x,i} \left( \underbrace{\left[ b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj}) \right]}_{\text{Predicted rating}} - r_{xi} \right)^2$$ (True rating)

To minimize SSE, we used gradient descent algorithm where we calculated gradient of matrix w with the following formula. We trained our model with many epochs.

$$\nabla_w J = \left[ \frac{\partial J(w)}{\partial w_{ij}} \right] = 2 \sum_{x,i} \left( \left[ b_{xi} + \sum_{k \in N(i;x)} w_{ik}(r_{xk} - b_{xk}) \right] - r_{xi} \right) (r_{xj} - b_{xj})$$

### 3.3.2. Singular Value Decomposition (SVD) with Stochastic Gradient Descent (SGD)

During the computation of SSE and RMSE with SGD, unlike GD, free variables associated with a single rating are updated in one step. Also, SGD needs more steps to converge. However, each step it is faster than GD.

**For each $r_{xi}$:**

- $\varepsilon_{xi} = (r_{xi} - q_i \cdot p_x)$
- $q_i \leftarrow q_i + \mu_1 (\varepsilon_{xi} p_x - \lambda_2 q_i)$
- $p_x \leftarrow p_x + \mu_2 (\varepsilon_{xi} q_i - \lambda_1 p_x)$

Prediction formula for this algorithm can be seen in above equation. First, p and q matrices are initialized. Then, for each single rating, derivative of the error is calculated. Later on, by using this value, corresponding rows of p and q are updated. At the end of each step, SSE and RMSE are calculated after finding prediction matrix $\mathbf{p*q^T}$.

# 4. Test and Validation Results

## 4.1. Cosine Similarity based on TF-IDF

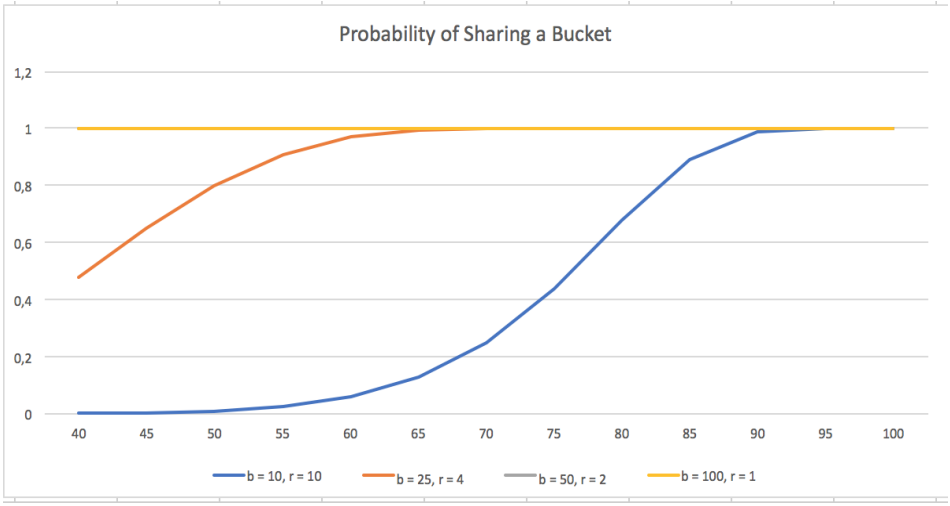
As expected, movies which belong to the same movie series turned out to be similar. For instance, cosine similarity between Star Wars - Return of the Jedi and Star Wars - The Empire Strikes Back is 0.84. It is respectively high. Similarly, movies that have different subjects turned out to be dissimilar. To give an example, cosine similarity between Austin Powers and Avengers is 0.008. Nevertheless, some movies belonging to the same movie series did not turn out to be similar as much as we expected. For instance, cosine similarity between Star Wars - Return of the Jedi and Star Wars - Revenge of the Sith is 0.08. The reason of that: many different characters exist in Return of the Jedi and Revenge of the Sith. TF-IDF scores of these characters dominate the score vectors of the movies. That is why, they turned out to be dissimilar in terms of TF-IDF scores. Consequently, we gave medium weight for this method since it is useful to some extend in terms of finding similarity of movie pairs.

## 4.2. Jaccard Similarity based on Genres

There exists an inconsistency in movie genres, that is some movie pairs that we expected to be similar were not similar in result. Since our dataset contained such inconsistencies, movies from the same series do not have jaccard similarity which is 1. It lead to wrong results.

Furthermore, we could not use movies with empty genre lists. This shrinks our dataset which results in losing data to train our methods. Nevertheless, we gave the largest weight for the result of this method since we agreed that most important similarity metric between movie pairs can be inferred from genres.

## 4.3. K-shingling, Minhashing and Locality Sensitive Hashing




Plot 1: Probability of Sharing a Bucket for Different Band and Row Values

Horizontal axis: Similarity between movie pairs

Vertical axis: Probability of sharing a bucket

For testing results of k-shingling, minhashing and locality sensitive hashing, we obtained the plot above (Plot 1). To obtain the probability values we used the following formula: $1 - ( 1- t^r)^b$ , where b is the number of bands and r is the number of rows in that band [4]. Since we had 100 hash functions band and row values always multiply to 100. Each line in the plot represent probabilities for different band and row values. For testing our results, we chose a movie script at random and modified its k-shingles by the given percentage, then added this modified version to our dataset. Finally, we checked whether they fall into the same bucket. We corrupted the k-shingles of the chosen movie from 5 percent to 100 percent, increasing the corruption rate by 5 percent in each run. Corresponding probability values and results of our tests can be seen in the table below (Table 1). For each similarity, band and row values 2 experiments were conducted. Color blue means the modified and the real version of the movie script were in the same bucket for both tests and color orange represents they were in the same bucket for only one test (Table 1). Results of our tests suggested that this algorithm works only for near duplicates. That is why we decided to give it the least coefficient in contributing the similarity of movie pairs.

| **Similarity** | **b = 10, r = 10** | **b = 25, r = 4** | **b = 50, r = 2** | **b = 100, r = 1** |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 100 | 1 | 1 | 1 | 1 |
| 95 | 0,9999 | 1 | 1 | 1 |
| 90 | 0,9863 | 1 | 1 | 1 |
| 85 | 0,8884 | 1 | 1 | 1 |
| 80 | 0,6789 | 1 | 1 | 1 |
| 75 | 0,4399 | 0,9999 | 1 | 1 |
| 70 | 0,2491 | 0,9990 | 1 | 1 |
| 65 | 0,1268 | 0,9927 | 1 | 1 |
| 60 | 0,0588 | 0,9689 | 1 | 1 |
| 55 | 0,0250 | 0,9092 | 1 | 1 |
| 50 | 0,0097 | 0,8008 | 1 | 1 |
| 45 | 0,0034 | 0,6489 | 1 | 1 |
| 40 | 0,0010 | 0,4771 | 0,9998 | 1 |
| 35 | 0,0003 | 0,3148 | 0,9985 | 1 |
| 30 | 0,0001 | 0,1840 | 0,9910 | 1 |
| 25 | 0 | 0,0932 | 0,9603 | 1 |
| 20 | 0 | 0,0392 | 0,8701 | 1 |
| 15 | 0 | 0,0126 | 0,6795 | 1 |
| 10 | 0 | 0,0025 | 0,3950 | 1 |
| 5 | 0 | 0,0002 | 0,1176 | 1 |
| 0 | 0 | 0 | 0 | 0 |

Table 1: Probability Values and Test Results for Different Band and Row Values and Similarity Values

## 4.4. Word2Vec Average

Although we used this method in order to improve our results since it is widely used for this purpose, we did not observe any negative or positive effect of this method. That is why, we gave it a moderately low weight.

## 4.5. K-means Clustering using Word2Vec Representations

We encountered some problems while obtaining results of this method. The important problem is that the ordering of word2vec centers in the vectors results in

different outputs. That is why, similarity results of movie pairs are not consistent. Additionally, determined centers of clusters emerged as stop words. Stop words do not have any contribution to the similarity of movies. Due to these issues, we decided not to use the results of this method.

## 4.6. Validation Results

### 4.6.1. Collaborative Filtering with Interpolation Weights

- We ran collaborative filtering algorithm 200 epochs, 10000 users and 808 movie scripts.

  SSE is approximately 769,0788.

- We ran our similarity algorithm with the same number of users and movie scripts. We defined $s_{ij}$ from Eqn 1. as follows:

  A, B, C and D represent the resulting matrices of the algorithms that we decided to use for our final calculations.

  A: K-shingling, Minhashing and LSH
  B: Cosine Similarity Based on TF-IDF
  C: Jaccard Similarity Based on Genres
  D: Word2vec Averages

  $$s_{ij} = A_{ij} * 0.1 + B_{ij} * 0.3 + C_{ij} * 0.4 + D_{ij} * 0.2$$

  SSE is approximately 281770.

- Then we combined predictions from both algorithms with ratio 10000:1 as collaborative filtering has the greater portion.

  SSE is approximately 769.3347.

### 4.6.2. Singular Value Decomposition (SVD) with Stochastic Gradient Descent (SGD)

SVD with SGD has many parameters, such as two different learning rates, the number of factors, two different user set regularization parameters. Also, initialization of p and q is important. So, it takes a long time to find the best parameter combination. Furthermore, a single step requires too much time since the rating matrix is large. Therefore, we could not get perfect results.

For 8 epochs, 10000 users and 808 movie scripts:

SSE is approximately 1.9 million and RMSE is approximately 0.0034.

# 5. Discussion and Conclusions

Collaborative filtering with interpolation weights performs extremely well in predicting user ratings. It was a difficult challenge to surpass its accuracy. However, SVD with SGD was not good in terms of SSE.

We observed no correlation between semantic features of movie scripts and genres. Syntactic features extracted using our various methods, such as k-shingling and minhashing are useful only when near duplicate documents are compared. However, movie scripts do not tend to be that much similar. In conclusion, our attempt to use semantic and syntactic similarity between movie scripts caused no improvement on recommendation systems in terms of accuracy.

# References


[1] “IMSDB Movie Scripts.” [Online]. Available: https://figshare.com/projects/imsdb_movie_scripts/18907. [Accessed: 1-Nov-2018].

[2] “MovieLens 20M Dataset.” [Online]. Available: https://www.kaggle.com/grouplens/movielens-20m-dataset/version/. [Accessed: 1-Nov-2018].

[3] “OMDB API.” [Online]. Available: http://www.omdbapi.com/. [Accessed: 1-Nov-2018].

[4] Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman, Mining of Massive Datasets. Cambridge University Press New York: 2011. [Accessed: 1-Nov-2018].

[5] “ K-means Clustering Visualization Using Word2Vec Representations.” [Online]. Available: https://plot.ly/~eliztekcan/20/#/. [Accessed: 1-Nov-2018].