Recommendation System for Movies

Gustaf Rydholm

***Abstract*—In this project a recommendation system for movies was created, that could give recommendations of movies that were similar to a user's query. The system started by transforming a database of movies into vectors, based on a movie profile that was constructed, using a vector space model. Then, similar movie vectors could be clustered together using the *k-means* algorithm. Recommendations were made by computing the similarity between the query and other movies in the same cluster. The similarity measures that where studied was the *cosine similarity* and the *Jaccard similarity*, as both of these are widely used for information retrieval problems. To evaluate the system, the recommendations were compared to a ground truth, consisting of movie recommendations from IMDb and Google. With a statistical evaluation called *F–measure* it was concluded that the system preformed rather poorly with both similarity measurements, but the *Jaccard similarity* outperformed the *cosine similarity*, with a *F–measure* of 8.04% compared to 4.53%. These results was compared with another system implemented by another student in a parallel project, this system had a $F - measure = 7.68\%$.**

I. Introduction

RECOMMEDATION systems are a class of applications which purpose are to suggest items that a user might like, based on previous behavior of the user, e.g. product bought, movies watched or news articles read. Even though all recommendation systems are created in order to preform the same task, numerous algorithms have been developed to improve efficiency of the recommendations. Yet, these system work in one of two ways, and can therefore be classified into two broad groups [1].

- *Content-based systems* recommend items similar to the ones the user like by finding other items that have similar attributes [2].
- *Collaborative filtering systems* gives recommendations based on the relationship of users. Users with the same type of behavior, e.g. items searched, movies rated similarly or products bought, tend to share the same interests. Therefore, it is likely that a user belonging to a group will like the same items as others have liked, even though the user is unaware of these yet [3].

For Internet companies these type of systems has become an invaluable asset, where on-line retail giant Amazon experienced a 29% increase in sales during the second fiscal quarter of 2012, compared to the same time period in 2011 [4]. This growth in sales is believed to be strongly correlated to the implementation of their recommendation system called "item-to-item collaborative filtering" [4]. Another company that has seen a great value in personalizing the user's experience via recommendation systems is Netflix [5]. So much so that they announced a competition called the Netflix Prize, where they offered $1 million to whomever that could improve the accuracy of their prediction system of movie ratings by 10% [6]. Bringing together the fact that there is endless possibilities when creating a recommendation system, with a growing demand from internet companies to develop more efficient system, makes this an interesting research field.

The task of this project was two-folded, the first part was to create a *Content-based system* for movies. The program allowed the user to get recommendations on either a movie they like, or a manually enter a set items for attributes of the movie profile, see section II-C. First, the algorithm created a vector space model of all the movies in the data set. Then, a clustering algorithm called *k-means* [7] was used in order to partition similar movies into the same cluster.

The second part of this report was to evaluate the systems performance, using the F-measure [8]. This was achieved by first collecting a data set containing a sampled set of movies, together with the systems recommendations for each of them. The recommendations was then compared to a ground truth consisting of a combination of movie recommendations gathered from IMDb's *"People who liked this also liked..."* and Google's *"People also search for"*.

The performance of the system was also compared with another *Content-based system*, developed by another student, where a graph-processing approach was used instead. The reason for this was to evaluate which approach was preferable when creating a recommendation system for movies.

II. Method

In this section the we present the algorithm of the recommendation system. First, we present the language and the tools needed. Next, we present the movie profile that was used to represent each movie. Then, we present the vector space model that was created to convert the movies to vectors in $\mathbb{R}^{10}$, and the pre-processing step of normalization. After that, we explain the *k-means* clustering, the initialization algorithm and how the number of clusters was selected based on the elbow method. We then go on to present the methods of finding movies to recommend based on a query. Lastly, we present the a statistical method to determine the performance of the system, called the *F–measure*.

A. Scala & Apache Spark

Scala is a programming language that runs on the JVM. It is both an object-oriented and functional language with emphasis on scalability [9]. In the big data community Scala has become one of the most prominent languages, with the large-scale data processing engines such as Apache Spark and Apache Kafka written in it [10].

Apache Spark was selected as the engine to use for the processing of the database of movies, the reason behind this choice was because it is one of the fastest frameworks with a vibrant ecosystem, and the high level API made it easy to work with [11].

As the recommendation system relied on the Apache Spark and the scalable machine learning library, MLlib, for the data processing and the implementation of the *k-means* clustering method, the choice of writing the recommendation system in Scala was made.

B. Movie database

The database of movies that was used for the recommendation system was called *"The Open Movie Database"* [12]. With this database we had access to the attributes shown in Table I for each movie.

TABLE I
ATTRIBUTES OF EACH MOVIE IN THE OMDB DATABASE.

imdbID	Genre	Metacritic	FullPlot
Title	Released	imdbRating	Language
Year	Director	imdbVotes	Country
Rating	Writer	Poster	Awards
Runtime	Cast	Plot	lastUpdated

These attributes were considered to give a detailed enough description of each movie, so that an extensive enough movie profile could be created.

The original size of the database was 1128654 movies, however most of these movies were considered irrelevant. Therefore, most of the movie in database was filtered out. The filtering was based on the popularity on IMDb, where the movies had to have rating above 5.0 and more than 1000 votes. The remaining size of the database was 21555 movies.

C. Movie Profile

To convert all movies to vectors a movie profile was needed. The movie profile was constructed with a set of characteristics considered to be the most significant to a movie. The attributes that considered important were the ones that allowed a user of the recommendation system to get relevant results, based on the query. Some attributes were considered to have little or no relevance to a viewer, e.g. writer, rating or run time.

As there is no perfect set of attributes to use when creating a *content-based system* for movies, the profile for each movie contained the attributes considered relevant by the author in [1]. With the addition of the language spoken and popularity, given by the rating and number of votes the movies had on IMDb. Thus, the movie profile contained the following profile:

i. A subset of the cast from the movie. The actors considered relevant were the first three, as they are most likely to be the stars of the movie and have the most screen time. As people often have some favorite actors, other movies that have one or more of them in the cast will be likelier to be a relevant recommendation.

ii. The director, regarded as one of the most important attributes of the movie profile, as the director have the responsibility of bringing the movie to life. Some directors have a distinct way of portraying a movie, via visual effects and storytelling, e.g. the Coen brothers [13] or Wes Anderson [14], which in turn have lead to a large fan base. Therefore, even if the viewer is unaware of who made the film, it is plausible they will enjoy other work from the same director. Even more so than the other work of an actor.

iii. The year in which the movie was released. If a viewer likes old western movies, the probability is higher that the same viewer prefer other western movies from the same era, rather than newer productions.

iv. The subset of the two first genres of the movie. Together with the director, were considered the most significant for the movie profile. If a user query a "comedy-drama" movie, the user expects that the recommendations presented to them will be of the same genres or similar. The reason two genres was used instead of one to represent the movie, was to increase the performance of the recommendations. Often a movie can be classified by several genres, therefore many movies have a genre in common, but might not be similar at all, e.g. a "drama-war" and a "drama-romance" movie.

v. The popularity of a movie, which was determined by the rating and number of votes on IMDb. These feature were useful as a viewer might like blockbuster movies, then it was considered likelier that the user would preferred other movies of that caliber, and not a collection of lesser known movies.

vi. The main language that was spoken in the film. This will allow for a user to get recommendations based on preference of language, as one might like French-language movie, thus movies with another language spoken might not be as relevant.

To summarise the movie profile, each attribute is shown in Table II.

TABLE II
ATTRIBUTES IN THE MOVIE PROFILE.

Actor #1	Genre #1
Actor #2	Genre #2
Actor #3	imdbRating
Director	imdbVotes
Year	Language

D. Vector Space Model

To represent each movie profile as a vector, we created a vector space model. The model allowed each unique item of an attribute to be represented with a specific number. This was done by separately assigning values to each unique actor,

director, genre and language. Because the value of the year, rating and number of votes was almost continuous, these could be used without any type of transformation.

In order to prevent that several values was given to a unique item of an attribute, e.g. actor Julia Roberts, a map for each attribute was used to store the value assigned. These where used in order to check if a item had been assigned a value or not. If an attribute was empty, the value of 0.0 was assigned to it. Each unique attribute was given an incremented value of 1.0 from the previous one.

E. Normalization

Pre-processing of the data is almost always required in order to increase results' performance, and this is done by normalize the data. This is especially important when using distributed clustering algorithms, such as the *k-means* implemented in Apache Spark, where the Euclidean distance is used [15]. As the Euclidean distance is a distance metric that is sensitive to the differences in magnitude and overpowering values of attributes [15].

Therefore, we scaled all attributes between a lower boundary C and a upper boundary D, with the help of Min-max normalization. As this is a linear transformation, the relationship between values of different items was kept [15]. Let x be some value of an unique item of an attribute $\mathcal{A}$, assume that $min_{\mathcal{A}}$ and $max_{\mathcal{A}}$ are the minimum respectively maximum value for that attribute. Then the normalize value given by x', is given by the following equation [16].

$$x' = \frac{x - min_A}{max_A - min_A} \cdot (D - C) + C \tag{1}$$

F. Clustering Method

As we now had an unlabeled dataset of movies, we now wanted to group the data into coherent clusters. In order to do this we used the *k-means* clustering algorithm, as its one of the most widely used unsupervised learning algorithms for this type of problem [17]. In this section we start by presenting the mathematical notation of the k-means. Then, we present the initialization algorithm, *k-means*$\|$. Lastly, explain the Lloyd's algorithm that computes the convergence of the centroids. As these algorithms already existed in the scalable machine learning library, MLlib, in Apache Spark, there was no need to implement these in this project. However, as both of these algorithms were vital to the project, it is necessary to describe the theory behind them.

Let start by introducing the set of movies, as $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, where each movie, $\mathbf{x}$, is represented by a point in the 10-dimensional vector space. The number of clusters used is given by k, where $S_1, S_2, ..., S_k$, each corresponds to a specific cluster [18], with a centroid in the set $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_k\}$. Each point in the set X, will be partition into a cluster $S_1, S_2, ..., S_k$, such that the centroids of the cluster will minimize the cost function given by

$$\phi_X(\mathcal{C}) = \sum_{\mathbf{x} \in X} d^2(\mathbf{x}, \mathcal{C}) = \sum_{\mathbf{x} \in X} \min_{i=1,...,k} \|\mathbf{x} - \mathbf{c}_j\|^2 \tag{2}$$

where $\|\mathbf{x} - \mathbf{c}_j\|$ denotes the Euclidean distance between two points. The minimum distance of a point, $\mathbf{x}$, and the subset of the centroids is defined as $d(\mathbf{x}, \mathcal{C}) = \min_{c \in \mathcal{C}} \|\mathbf{x} - \mathbf{c}\|^2$ [17]. When all points in X have been partitioned into clusters, the new centroids can be calculated using the following equation.

$$\mathbf{c}_j = \frac{1}{|S_j|} \sum_{\mathbf{x} \in S_j} \mathbf{x} \tag{3}$$

In [17] Bahmani et al. describes the crucial part of initializing the set of centroids in an nearly optimal way. An initialization of the centroids is needed in order to guarantee that the final position will be close to the optimal solution. They describe that one of the most popular initialization algorithms, *k-means++*, does not scale well, due to its inherently sequential nature. Therefore, they created a parallel version of this algorithm called *k-means*$\|$, thus making the *k-means* applicable on large scale data sets.

Their parallel version for initializing the centroids, see Algorithm 1, starts off by sampling a point uniformly at random and assigning it to the set $\mathcal{C}$. In Step 2, the initial cost, ψ, of this centroid is computed. The algorithm then proceeds to iterate $\log \psi$ times, where each iteration the number of expected points, x, sampled are given by the *oversampling factor* $\ell = \Omega(k)$, where each point is sampled with probability $\ell \cdot d^2(x, \mathcal{C}) / \phi_X(\mathcal{C})$. In Step 5, the sampled points in $\mathcal{C}'$ are then added to the set of centroids, $\mathcal{C}$, and the cost function, $\phi_X(\mathcal{C})$, recalculated. When the iteration is completed, the number of sampled points is expected to be $\ell \log \psi$. Often, this will mean that the number of points in $\mathcal{C}$ exceeds the predefined number of clusters, k. To reduced the set to k centroids, Step 7 assigns a weight to each point, and in Step 8 the set $\mathcal{C}$ is reclustered into a set of k centroids.

Algorithm 1 ***k-means*$\|$*(k, l)*** initialization [17].

1: $\mathcal{C} \leftarrow$ sample a point uniformly at random from X
2: $\psi \leftarrow \phi_X(\mathcal{C})$
3: **for** $\mathcal{O}(\log \psi)$ times **do**
4: $\quad \mathcal{C}' \leftarrow$ sample each point $x \in X$ independently with probability $p_x = \frac{\ell \cdot d^2(x, \mathcal{C})}{\phi_X(\mathcal{C})}$
5: $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$
6: **end for**
7: For $x \in \mathcal{C}$, set w_x to be the number of points in X closer to x than any other point in $\mathcal{C}$
8: Recluster the weighted points in $\mathcal{C}$ into k clusters

However, $\log \psi$ number of iteration seldom necessary to achieve a good initial set of centers. Therefore, its by default set to 5 [19]. Which has proven to be enough to achieve better clustering than any other initialization algorithm [17].

With the nearly optimal set of k-clusters obtained from the *k-means*$\|$ algorithm, Lloyd's algorithm was used in the *k-means* algorithm implemented in MLlib to minimize the cost function, $\phi_X(\mathcal{C})$ [20]. In [21] Hamerly et al. explains the three basic steps of this algorithm. The first step consists of initializing a set of centroids, which in our case was already

done. Then, in step two, each point in the data set is assigned to the closest centroid. In step three, each centroid will be moved to the mean of its assigned points, see equation 3. The algorithm will alternate between step two and three until the convergence criterion's are met. The criterion's set for the Lloyd's algorithm was the maximum number of iterations, and a threshold for the minimum movement of each center, ε, in the previous iteration [20].

In this project the maximum number of iterations was set to 50, and $\varepsilon = 1 \cdot 10^{-4}$. The number of clusters used was $k = 180$, see section II-G for detailed explanation.

G. The Elbow Method

For this clustering problem we did not have a optimal set of clusters, therefore a method was needed to determine a good value of k. One method for finding a suitable k is called the elbow method. According to the elbow method, a good value for k is the point where there the curve goes form having a sharp descending to a more flattened appearance [22], hence the curve will look like an arm with an distinct elbow. As the curve flattens out, the gain of increasing k will be relatively little [22].

To collect a set of data points to construct this curve, the data set of movie vectors was clustered 26 times for different number of k-clusters. During each run, the cost function, seen in Figure 1 as *Within Set of Squared Errors*, was computed. As the data set was quite large, the starting value for k was set to 50. For each run the value of k was incremented with 10, until a final value, which was set to $k = 300$.

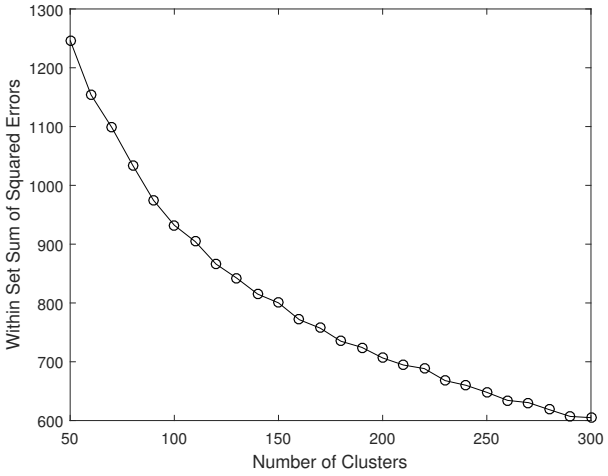

Fig. 1. The cost function, *Within Set Sum of Squared Errors*, for different values of k.

In Figure 1, we can observe that the curve has a sharp descending in the beginning and then it flattens out when the number of clusters exceeds $k = 180$. Thus, the number of clusters was set to $k = 180$.

H. Predictions

To find similar movies to the user's query, the query is first transformed into a vector representation. Then the vector is assigned to the cluster its closest to, based on the Euclidean distance to the center [20], [23]. After the cluster was obtained, the movies that was most likely to be of interest to the user had to be sorted out. One way of finding the similarity between object in a vector space model is to use the cosine similarity [1], defined as

$$\text{Cosine similarity} = \cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|} \tag{4}$$

To get a good intuition of how the cosine similarity works, we can look at Figure 2. Here we see the vector representation of two arbitrary movies, m_1, m_2, and a query, q. The angle between the query and the movies is represented by θ_1 and θ_2. The vectors will be more closely spaced if they share common attributes, e.g. actors, director or genre, therefore the angle between these vectors is small, as θ_1, and value of cosine close to 1. However, if movies does not share a lot of common features they will be further apart, as m_2 and the query, contributing to a larger angle of θ_2, and a small value of cosine. Thus, movie 2 will not be as relevant to the query as movie 1.

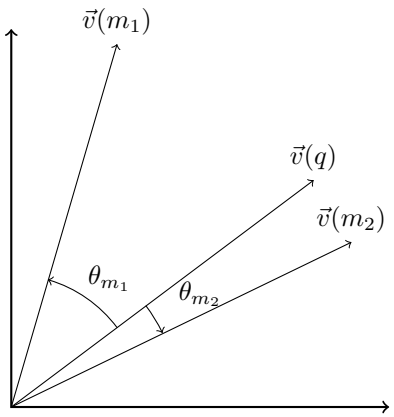

Fig. 2. The cosine similarity between the query, q, and two movies, m_1 and m_2, given by $\cos(\theta_{m_1})$ and $\cos(\theta_{m_2})$.

Another useful similarity measurement for information retrieval is the *Jaccard similarity*, see Figure 3. This gives us a measurement of how similar two movies are to each other based on their textural data [24]. The *Jaccard similarity* is defined as the intersect of movie A and B, dividen by the union of the two movies, that is

$$\text{Jaccard similarity} = \frac{|A \cap B|}{|A \cup B|} \tag{5}$$

When implemented, the measure was made more dynamic. Instead of comparing the year, number of votes and the ratings, the measure compared if the movies had been made in the same decade, had the same magnitude of votes and if the integer value of the ratings was equal.

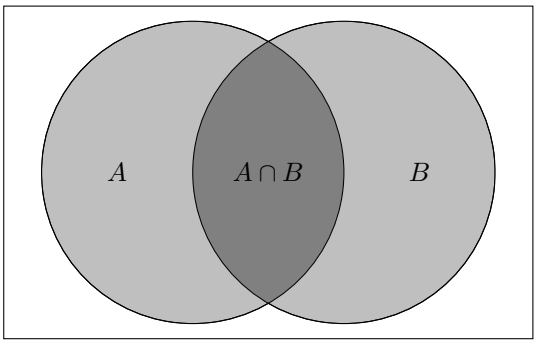

Fig. 3. *Jaccard similarity between movie A and B.*

Two experiments with these similarity measurements was made, one with the *cosine similarity* and the second one with *Jaccard similarity*.

I. Performance evaluation

To evaluate the recommendation system performance, the a statistical measure of the recommendations had to be computed. A common method in data mining is to use the *F–measure* [8], which was done by comparing the predictions to a ground truth. For this project the ground truth was selected as the first eight movies recommended on IMDb, i.e. *"People who liked this also liked..."*, and a random selection of eight movies from Google's *"People also search for"*.

The predictions could be classified into three possible outcomes, true positive, false positive or false negative [8]. A true positive, TP, meant the prediction was correct, i.e. belonged to the ground truth. If a prediction did not belong to the ground truth, it was considered a false positive, FP. All the movies that belonged to the ground truth but was not classified as TP or FP, where considered as false negatives, FN. With these outcomes defined, we could now compute the *precision*, defined as

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

which gave us a measurement of how many of the predictions was correctly given by the system. To measure the number of the correct movies selected from the ground truth, the *Recall* measurement was used.

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

In other words, we can say that *Precision* says how good the system is at identifying similar movies, and *Recall* says how good the system is at finding the correct movies to recommend. This means that the more interesting of these two measures is the precision, as this tell us how good our system is at presenting a correct recommendation. A common way to evaluate the performance of a system is to take the harmonic mean of the *Precision* and *Recall*, called the *F–measure*, as this measure assesses the trade off between the two [8].

$$F\text{–measure} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{8}$$

This gave us a statistical measurement of how well the system preformed. The number of movies used in order to evaluate the system was 100, and these movies were sampled from the data set at random with a function in Spark called *takeSample* [25].

III. Results

In this section we present an evaluation of the recommendation system. To do an evaluation of the system, two experiments was made with different similarity measures, the *cosine similarity* and the *Jaccard similarity*. The number of cluster was set to $k = 180$, with maximum number of iterations set to 50 for the Lloyd's algorithm. A sample set of 100 movies was used as queries. For each query the system was set to give a set of five movie recommendations. The upper and lower boundary of D and C was determined with experimental analysis. These differed between the two experiments and are presented in Table III. The change in the upper boundary for the *Jaccard similarity* experiment made the predictions more dependent on the actors, director and genres.

TABLE III
THE ASSIGNED UPPER AND LOWER BOUNDARY FOR EACH EXPERIMENT.

	Cosine similarity		**Jaccard similarity**	
	D	**C**	**D**	**C**
Actors	1.0	0.0	7.0	0.0
Director	1.0	0.0	18.0	0.0
Year	1.0	0.0	3.0	0.0
Genre	1.0	0.0	7.5	0.0
imdbRating	1.0	0.0	2.0	0.0
imdbVotes	1.0	0.0	3.5	0.0
Language	1.0	0.0	80.0	0.0

TABLE IV
THE CLASSIFIED OUTPUT OF THE SYSTEM FOR THE TWO DIFFERENT SIMILARITY MEASUREMENTS.

	TP	**FP**	**FN**
Cosine similarity	36	464	1052
Jaccard similarity	65	435	1052

TABLE V
THE COMPUTED VALUES OF *Recall*, *Precision* AND *F–Measure* OF THE TWO SIMILARITY MEASURES.

	Recall	**Precision**	**F–Measure**
Cosine similarity	0.0331	0.0720	0.0453
Jaccard similarity	0.0582	0.1300	0.0804

Table IV show the classified predictions of the system. The *Jaccard similarity* preformed slightly better than what the *cosine similarity*, however most of the predictions did not exist

in the ground truth. In table V, the computed results for *Recall*, *Precision* and *F–Measure* is presented.

IV. Discussion

The systems preformed poorly when the recommendations where compared to the ground truth. Between the *cosine similarity* and *Jaccard similarity* an improvement of almost the double could be seen in the *Precision* and *F–Measure*. The predictions were altered quite heavily in some cases, where all or most movie recommendations differed. The *Jaccard similarity* was better at recommending movies that shared either actors, director or genre with the query. Therefore it was able to find more movies in the ground truth. As the vector space model did not take into consideration how movies was related, movies with almost no similarity could end up very close to each other. For example, an American blockbuster could have previously been assigned unique points in the vector space, next a Bollywood movie might have been assigned a unique numbers also. This would have meant that difference between the items of each attribute would have been 1.0, which might not have been a good representation of how similar the movies was. Therefore, the *cosine similarity* was not the optimal measure to find similar movies. However, as one of the attributes was language, the chance that the movie predictions would be of a different language was reduced somewhat. With the previous example its easy to realize that the *Jaccard similarity* would get more relevant results. But, recommendations get somewhat limited to the items of each attribute, which is not the case for the *cosine similarity*.

As the system created was an implementation of a *Content-based system*, and the ground truth consisted of recommendations from *Collaborative filtering systems*, i.e. IMDb's *"People who liked this also liked..."* [26] and Google's *"People also search for"*, the expectations for the evaluation were not that high. As *Content-based systems* only finds similarities based on the item profile, but a *Collaborative filtering systems* is much more dynamical, as it adapts to the users behavior. Therefore, it was not necessary that the recommendations in the ground truth had anything in common with the query, other than that users had rated or searched for them. This would also partly explain why the *Jaccard similarity* did not find many of the recommendations in the ground truth.

When these results was compared to the other project, where a student had used a graph-processing approach to create the recommendation system, it was shown that the other system preformed better. It was especially better at predicting correct, $precision = 0.1608$. But was almost equally bad at finding the correct recommendations, $recall = 0.0505$. The *F–measure* for the other system was 0.0768, this was better when compared to the *cosine similarity* experiment, however not when compared to the experiment with the *Jaccard similarity*. The other system was not as dynamical as it gave at least one recommendations that contained either an actor or the director of the query. This was not the case for the system in this report, as it could recommend movies that was close in the vector space, but would not necessarily have a director or actors in common.

As the results of this project were heavily dependent on how good the vector space model was, it would be interesting to see if the performance could be increase with a more advanced model. This vector space model could for example sort movies based on countries, popularity, the frequency of directors and actors. Sorting the countries after relation to each other would be a good thing, for example the data set could be dissected into region, such as Western and Eastern. Popularity would allow for the blockbuster movies to be more closely related. Together with the popularity, a weight for each actor and director based on frequency could be implemented in order to make big stars more important, thus assigned closer together in the vector space. The genres could also be sorted so similar ones would be closer together in the vector space, as there might be a problem if comedy and horror are the ones that are the closest to each other. This would be a good starting point in order to make the recommendation system in this project better at predicting movies that a user might like.

Otherwise I would recommend to try create a *Collaborative filtering system* as these are far superior when it comes to being dynamical and general in predicting items.

Acknowledgment

The author would like to thank Kambiz Ghoorchian for his help and supervision during the project.

References

[1] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press, 2011. [Online]. Available: http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

[2] P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds., *The Adaptive Web: Methods and Strategies of Web Personalization*, ser. Lecture Notes in Computer Science. Berlin: Springer, 2007, vol. 4321. [Online]. Available: http://www.fxpal.com/publications/content-based-recommendation-systems.pdf

[3] M. D. Ekstrand, J. T. Riedl, and J. A. Konstan, "Collaborative filtering recommender systems," *Found. Trends Hum.-Comput. Interact.*, vol. 4, no. 2, pp. 81–173, Feb. 2011. [Online]. Available: http://files.grouplens.org/papers/FnT%20CF%20Recsys%20Survey.pdf

[4] J. Mangalindan (July 2012). (2016) Amazon's recommendation secret @ONLINE. [Online]. Available: http://fortune.com/2012/07/30/amazons-recommendation-secret/

[5] X. Amatriain and J. Basilico (April 2012). (2016) Netflix recommendations: Beyond the 5 stars (part 1) @ONLINE. [Online]. Available: http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html

[6] (2016) Netflix prize @ONLINE. [Online]. Available: http://www.netflixprize.com/

[7] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, 1967, pp. 281–297. [Online]. Available: http://projecteuclid.org/euclid.bsmsp/1200512992

[8] D. L. Olson and D. Delen, *Advanced Data Mining Techniques*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[9] M. Odersky. (2016) What is scala? @ONLINE. [Online]. Available: http://www.scala-lang.org/what-is-scala.html#a-scalable-language

[10] A. Woodie. (2015) Will scala take over the big data world? @ONLINE. [Online]. Available: http://www.datanami.com/2015/08/10/will-scala-take-over-the-big-data-world/

[11] S. Penchikala (January 2015). (2016) Big data processing with apache spark part 1: Introduction @ONLINE. [Online]. Available: http://www.infoq.com/articles/apache-spark-introduction

[12] B. Fritz. (2016) The open movie database @ONLINE. [Online]. Available: http://www.omdbapi.com/

[13] T. E. of Encyclopdia Britannica (February 2016). (2016) Coen brothers @ONLINE. [Online]. Available: http://global.britannica.com/biography/Coen-brothers

[14] R. Gray (March 2014). (2016) The science that makes wes anderson's films so good @ONLINE. [Online]. Available: http://www.telegraph.co.uk/news/good-to-share/10707340/The-science-that-makes-Wes-Andersons-films-so-good.html

[15] N. Karthikeyani Visalakshi and K. Thangavel, "Impact of normalization in distributed k-means clustering," *International Journal of Soft Computing, 4*, pp. 168–172, 2009. [Online]. Available: http://medwelljournals.com/abstract/?doi=ijscomp.2009.168.172

[16] S. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *arXiv preprint arXiv:1503.06462*, 2015.

[17] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proc. VLDB Endow.*, vol. 5, no. 7, pp. 622–633, Mar. 2012. [Online]. Available: http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf

[18] J. Hopcroft and R. Kannan, *Foundations of Data Science*, 2013. [Online]. Available: http://www.cs.cornell.edu/jeh/nosolutions90413.pdf

[19] (2016) Kmeans.scala documentation @ONLINE. [Online]. Available: http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.clustering.KMeans

[20] (2016) Kmeans.scala source code @ONLINE. [Online]. Available: https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/mllib/clustering/KMeans.scala

[21] G. Hamerly and J. Drake, "Accelerating lloyds algorithm for k-means clustering," in *Partitional Clustering Algorithms*. Springer, 2015, pp. 41–78.

[22] R. L. Thorndike, "Who belongs in the family?" *Psychometrika*, vol. 18, no. 4, pp. 267–276. [Online]. Available: http://dx.doi.org/10.1007/BF02289263

[23] (2016) Kmeansmodel.scala source code @ONLINE. [Online]. Available: https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/mllib/clustering/KMeansModel.scala

[24] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press, 2011. [Online]. Available: http://infolab.stanford.edu/~ullman/mmds/ch7.pdf

[25] (2016) Spark programming guide @ONLINE. [Online]. Available: http://spark.apache.org/docs/latest/programming-guide.html

[26] Personalized recommendations frequently asked questions @ONLINE. [Online]. Available: http://www.imdb.com/help/show_leaf?personalrecommendations&ref_=tt_rec_lm