A Network Tour of Millenial Movies
Project for the course Network Tour in Data Science

Milena Filipović, Kristijan Lopatički, Jelena Malić and Davor Todorovski

***Abstract*— As Walt Disney once said: "Movies can and do have tremendous influence in shaping young lives in the realm of entertainment towards the ideals and objectives of normal adulthood." But what do viewers really know about movies and what makes them successful? This project, based on the TMDb dataset, offers some interesting insights into movies from the past several decades. It shows how some of the movie features are correlated, explores how movies can be classified into genres using spectral graph analysis and CNNs, and gives a simple demo of a recommender system.**

I. Introduction

A graph is generally defined as an ordered pair of sets. The first set contains vertices or nodes, while the second is a set of edges or links. Graphs can be used to describe a wide variety of different concepts, and are present in various forms, some well-known, others more obscure. In this project, graphs will be used to represent similarities between different movies released after 1989. Movies represent a very interesting form of entertainment and can be a meeting point of many different creators and topics, so it will be intriguing to see how they correlate with each other depending on their features.

The rest of the report is organized in the following manner: Section II will give a short overview of the used dataset and its preprocessing. Section III will show some of the interesting movies' features and their correlations. Most importantly, Section IV will show how spectral analysis and CNN networks can be used in this context, and will provide basics of recommender systems that can be constructed over the data.

II. Dataset

A. Description

This project is based on the data obtained from TMDb (The Movie Database [1]), a community built movie and TV database. All of the data has been added by the community, and it has a strong international focus which makes the database very broad and unique. It has been updated since 2008, and the numerous contributions come from millions of users from over 180 countries. The database contains extensive metadata about movies, TV shows and people, with the addition of high resolution posters and fan art, with over 437,000 entries only in the film dataset. The platform has been praised as the one that has maintained an API that is pragmatic, reliable, well-structured and well documented from day one.

However, our project will not use the whole database but a meaningful subsample obtained from the Kaggle dataset [2]. The dataset contains important information about 5000 most relevant movies of the past several decades. In addition to these movies, there is a supporting dataset containing the information about cast and crew employed on each of the movies. All of these employee lists are given as JSON objects with detailed information about each of the persons

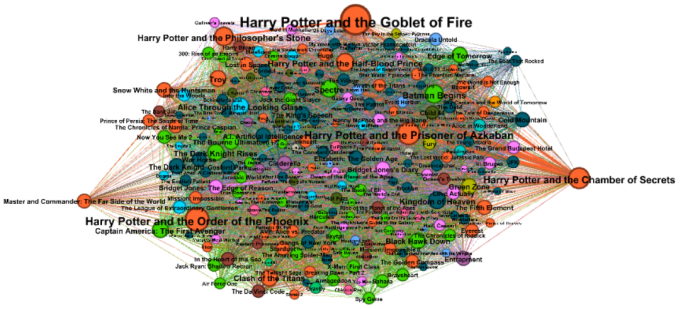


Fig. 1: Visualization of a subset of our cast and crew co-occurrence graph generated using Gephi [3].

represented in the movies' credits. Other than these, each of the movies is given 20 features, with some being used for identification like *id*, *overview* and *title*. Other useful columns that can be obtained from the dataset are related to financial aspects of the films (i.e., *budget*, *revenue*), the additional information about international movies (i.e., *original language and title*, *spoken languages*), and the movies success with viewers (i.e., *popularity*, *vote count* and *average*). Another very important feature is given in the *genres* column, which contains information about all of the genres a specific movie belongs to. In the end, it is important to note that there is also a list of all of the keywords attached to each movie.

B. Cleaning

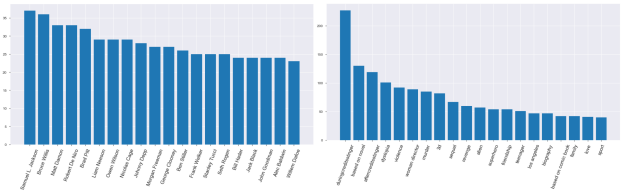
The TMDb dataset is made up of movies from various countries, and as such there is a large number of different original languages in which these films were created. For simplicity, we opted to retain only those movies where the original language in which they were filmed was English. A small number of movies in the dataset were in a state of pre- or post-production. In order to maintain consistency throughout our data, we decided to keep only movies which were released at the time that the TMDb dataset on Kaggle was created. To further sub-sample the original dataset, we also filtered the movies so that only those released after the 1st January 1990 remained, and made sure to keep only those films which had both their budget and revenue over 0. After further exploring this subset of data, we kept those movies which were in the top 3 quartiles by vote count. The reasoning behind this was that while a certain movie might have a high vote average if only a very small number of people voted for that movie at all, this vote average is meaningless. Therefore, in order to be able to compare movies, we filtered out the movies which had less than 225 votes, removing those films that were not particularly popular. The minimal budget and revenue values in the dataset at this point were both less than 10 dollars. In order to remove absurd values such as these from the data, yet not limit out analysis to extremely high budget Hollywood blockbusters, we chose to remove all movies with revenues and budgets below 10,000 dollars.

This concludes the data cleaning performed on the initial TMDb dataset, which reduced the set from 4185 movies to 1943. Further analysis of the dataset was performed upon a network constructed from these movies. To create this network we created an adjacency matrix. Since no edges were explicitly provided in the TMDb dataset, we decided to use the crew and cast data to generate edges that would connect the movies (nodes). The edges are created between any two movies that shared a common cast or crew member. The weights of these edges were determined depending on the number of shared cast and crew members. This gave us a network of 1943 nodes and 209,095 edges, however, it was composed of two components, one containing 1942 nodes, and the other just one node. This unconnected node is an independent documentary and it was removed from the dataset for simplicity. The analysis described in later sections was performed without it.

A visualization of a subset of our cast and crew co-occurrence graph can be seen on Figure 1.

III. Data Exploration

In order to better understand our data, we must look into some properties of our features in detail. Firstly, we explored which 20 actors have been most popular and present in the movies from past several decades. It can easily be seen that the most popular actors are Samuel L. Jackson and Bruce Willis that both appeared in over 35 movies from the subsample. An interesting observation is that among 20 most represented actors there is almost no diversity, as there is only one actress – Cameron Diaz and three non-white actors.



(a) Most represented actors. (b) Most frequent tags.

Fig. 2: Movie data properties.

When observing the keywords associated with the movies, we can see that they can be divided into several categories. The first and most common category marks the existence of a during- or post- credits scene. Another category of keywords informs that the movie is based on a novel or a comic book, or supplements the genre of the movie. In the end, we have the woman director keyword, which tells us that there is a specific 'tag' for movies that were directed by a woman, while a movie directed by a man is almost a default.

Afterwards, we have decided to test how different features, specifically movie budgets and revenues are represented or correlated with different movie genres. In Figure 3 we can see that the lowest budgets are given to movies of genre Documentary, Horror and Music. On the other hand, the movies with the highest budgets are usually the Adventure, Animation and Fantasy movies, which can be attributed to incredibly high costs of animation and special effects required for their successful filming. It is also important to note that even with sporadic flops that leave the movies with revenues lower than their initial budgets, average revenue by genre is always higher than the average budget which means that flops are much rarer than they appear to be.

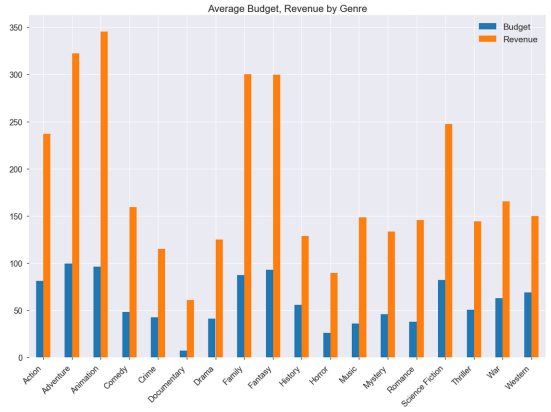


Fig. 3: Distribution of movie budget and revenue by genre.

As can be seen in Figure 4, there is a relatively strong correlation between the number of viewer votes and vote average. This is due to the fact that when so many viewers have seen the movie and took the time to review it, it means that it is a generally very popular movie. It can be seen that when a movie has above 8000 voters the average vote is always higher than 6.

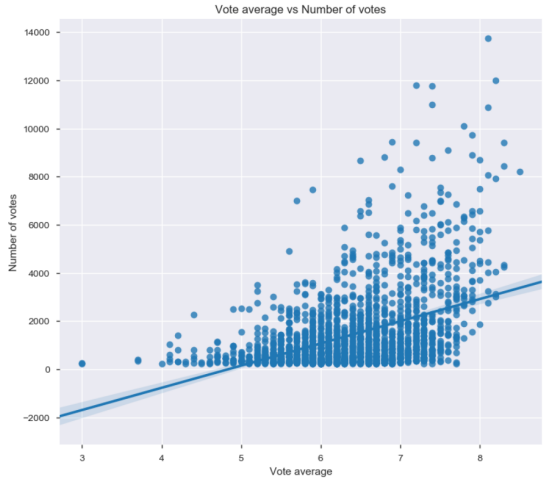


Fig. 4: Distribution of votes.

In order to see more interesting insights into the dataset make sure to check the Data Exploration notebook.

IV. Data Exploitation

A. Spectral Analysis

Laplacian eigenmaps [4] and graph embeddings in general are used for dimensionality reduction and can help in visualizing certain graph properties. They are used to learn a mapping from a graph to a vector space, creating a latent vector representation of the graph while preserving relevant graph properties. Graphs are complex structures with nodes and edges and it is often not possible to perform certain operations on these structures. Vector spaces on the other hand can be used in many mathematical, statistical and machine learning operations, and typically these operations are a lot simpler and computationally faster than their equivalent graph operations. There has recently been a lot of progress in this area, for example the well known application in the natural language processing field - "Word2Vec" [5].

Laplacian eigenmaps is a method to embed a graph $\mathcal{G}$ in a d-dimensional Euclidean space. That is, it associates a vector $z_i \in R^d$ to every node $v_i \in V$. The graph $\mathcal{G}$ is thus embedded as $Z \in R^{N \times d}$. This method finds the low dimensional

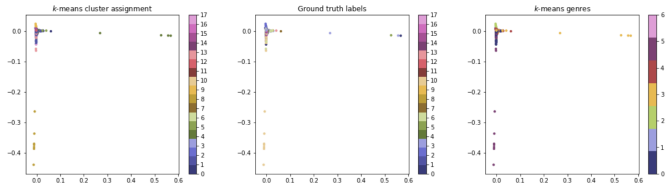


Fig. 5: The cluster assignment, ground truth first genre labels, and the genres assigned with k-means on the 2D embedding computed with Laplacian eigenmaps from the cast and crew graph.

representation of data using a spectral decomposition of the graph Laplacian. Laplacian eigenmaps outputs coordinate maps that are smooth functions over the original graph.

Spectral clustering [6] is a method to partition a graph into distinct clusters. The method associates a feature vector $z_i \in R^d$ to every node $v_i \in \mathcal{V}$, then runs k-means in the embedding space R^d to assign each node $v_i \in \mathcal{V}$ to a cluster $c_j \in \mathcal{C}$, where $k = |\mathcal{C}|$ is the number of desired clusters.

We decided to explore the distribution of genres in the spectral domain and wanted to see whether we could use spectral clustering as a method to partition the data into clusters that were defined by their respective genres.

In this part of our analysis, we first used the graph of cast and crew co-occurrences, and following this created a new graph from keyword co-occurrences.

In both cases we opted to keep only the first genre of each movie, as it is the one that carries the most value, and to see whether we could use spectral clustering and k-means to model these genres. We have 18 unique genres that occur in our dataset, thus we divide the dataset into 18 dimensions and k = 18. We embedded our graphs in R^d as $Z \in R^{N \times d}$, and tried k-means clustering with and without re-normalizing the eigenvectors by the degrees. Judging from the results obtained with k-means, we decided to proceed using the normalized graph Laplacian eigenvectors for both the cast and crew, and the keywords graphs. This is due to the fact that the re-normalized Laplacian produced a clustering where almost all of the nodes were in one cluster.

In order to attach a specific genre to each cluster, we looked at the ground truth labels in each cluster, determined which ones were the most common, and then assigned this label to the whole cluster.

1) Cast and Crew Co-Occurrence Graph: We applied spectral clustering on the movie graph built on cast and crew co-occurrences. Using the k-means algorithm we achieved 38.5% accuracy of correctly classified nodes, which implies that the algorithm was not overly successful in assigning the movie genre labels. This is partly due to the subsampling and partly because of the fact that k-means is not the ideal algorithm for recognizing labels in this dataset. Another problem is the number of genre combinations in the ground truth dataset, which is very high compared to the number of nodes in our subset.

From the listed original labels and the labels obtained from the cluster by majority voting, the algorithm assigned only 7 different genre labels to the clusters it created, namely: Drama (777), Action (703), Comedy (330), Animation (88), Horror (26), Adventure (14), Science Fiction (4). This is probably due to the fact that some of the genres are highly underrepresented in our dataset. Additionally, many of the movies that were wrongly labeled most probably contain the genre somewhere in the list of their ground truth list labels, which we had to summarize to only one as explained previously.

Figure 5 shows first the cluster assignment, then the ground truth first genre labels, and the third plot shows the genres assigned with k-means. As can be seen from these plots, it is hard to distinguish between different label assignments due to the shape of the 2D embedding computed with Laplacian eigenmaps. For this reason, we decided to shift our focus from the cast and crew graph and to explore the genre distributions using a graph based on keyword co-occurrences.

2) Keyword Co-Occurrence Graph: We apply spectral clustering on a new movie graph built on keyword co-occurrences. The data used for the creation of this graph was extracted from the previously performed data cleaning completed on the initial TMDb dataset, which reduced the set to 1942 movies. The nodes in this new graph tend to have a degree of 100 and below, and have a distribution is markedly different from the one obtained on the previous graph constructed from cast and crew co-occurrences. The average degree in that graph is also 1.66 times larger than the average degree in this newly constructed graph, indicating that these two graphs provide us with two very different insights into the world of movies and film. While this graph has 19 disconnected components, one of them is by far the largest and contains 1924 of the original 1942 nodes. The the other components contain only one disconnected node each. Further analysis is done with the largest component.

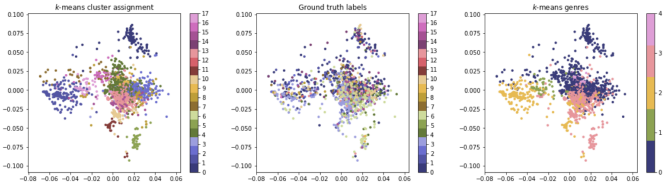


Fig. 6: The cluster assignment, ground truth first genre labels, and the genres assigned with k-means on the 2D embedding computed with Laplacian eigenmaps from the keywords graph.

Using the k-means algorithm we achieved 32.95% accuracy of correctly classified nodes, which confirms the results gained from the crew and cast co-occurrence graph in implying that the algorithm was not highly successful in assigning the movie genre labels.

From the listed original labels and the labels obtained from the cluster by majority voting, the algorithm assigned only 5 different genre labels to the clusters it created: Action (840), Drama (681), Comedy (301), Adventure (80), Horror (22).

As before we plotted the the cluster assignment, then the ground truth first genre labels, and the genres assigned with k-means, as can be seen on Figure 6. Comparing these plots with the ones generated using the cast and crew co-occurrence graph we can see that these cluster assignments are far clearer and seem to indicate that a graph based on keywords may be able to give us better insight into how genres are distributed throughout the dataset.

Based on these findings, we decided to further explore the possibility of genre prediction using Convolutional Neural Nets on Graphs with Fast Localized Spectral Filtering.

B. Convolutional Neural Nets on Graphs with Fast Localized Spectral Filtering

This section is devoted to our experimentation with the implementation of Convolutional Neural Networks (CNNs) translated onto the problem of a word embedding graph. The libraries, methods, and techniques presented here utilize the work introduced by Defferrard et al. [7]. This work generalizes CNNs from low-dimensional regular grids (i.e., images, videos, speech) to high-dimensional irregular domains (i.e., social networks, brain connectomes words embedding), represented by graphs, and presents a formulation of CNNs in the context of spectral graph theory.

In order to extract the elements necessary for forming the CNNs, we re-use data obtained during the implementation of previous steps, such as data created in the data cleaning and exploration phases, detailed in Sections III and II-B.

To build the graph used here, we first created a dataset containing the movies, genres and the keywords used to describe each of these movies. Next, we divided this data into a *training* (60%) and a *testing* (40%) set, taking care to make sure that our subsampling is stratified. It is important that all categories (in our case movie genres) are equally represented in the sampled data, as this ensures that future results are not biased.

In order to implement the CNN, the *training* data had to be preprocessed and prepared for use. All characters were transformed to lower case letters, then whitespaces and stopwords commonly found in the English language were removed and the movie-keywords were transformed to bags-of-words. Short 'documents' with less than 5 words were also removed, and from the remaining words, the ones which have embeddings were retained. Since there was not a large number of words at this point (666), we chose to keep all of them, and not filter any further. Finally, the *training* data was normalized, using the l_1 norm.

The testing data was then processed in a similar way as the training data. All characters were transformed to lower case letters and whitespaces. From this set of words only those which occur in the training data vocabulary were retained. The testing data was also normalized using the l_1 norm.

With these steps completed we moved on to the creation of the feature graph. We convert the *training* and *testing* datasets into the required data type, and then proceed to create the graph data based on the embeddings of all words in our training dataset (these same words are used also in the testing dataset). We then calculated the cosine distance between each pair of words and keep only those that were most strongly connected as neighbors in the graph. Using this, we created an adjacency matrix that was used to generate a Laplacian matrix. This Laplacian matrix is used directly in the CNNs.

We tested a number of models, the details of which can be found in the Data Exploitation - CNNs Notebook. The results of these trials are shown on Figure 7. The initial model configurations were taken from the previously referenced paper [7], however, they were optimized for our specific dataset.

On Figure 7 we can observe the testing accuracy of the different models. From this, we can see that the best model for our data was the 'fc_softmax' model, with one hidden layer and one output layer with a softmax function. Although this was our best model, the maximum test accuracy achieved was 44.69%. This is considerably better than what we were

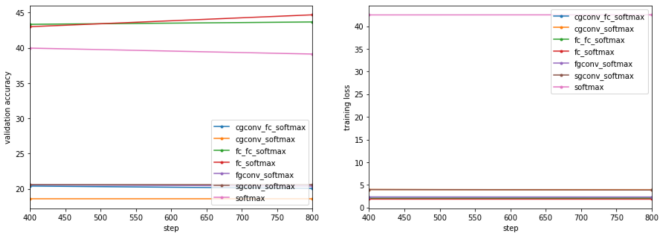


Fig. 7: Validation accuracy and testing loss achieved with the 7 models.

able to accomplish with the spectral clustering methods, but not the result we were hoping for. However, it is again important to underline that we are classifying only by the first genre of each movie and that keywords used for movies with ['Action', 'Adventure'] genres, and those used for movies with ['Adventure', 'Action'] genres are generally very similar. We believe that this, coupled with the small set of words used in the embeddings graph contributed considerably to the poor performance of the models with convolutional layers.

C. Movie Recommender Systems

1) Graph-based recommender: In this part we will present a Graph-based recommender which makes movie recommendations based on a very simple method.

Here we used the cast and crew graph where the edges are created between two movies whenever they share a common cast or crew member,with the weights determining the number of shared cast and crew members. Thus, for a given movie, the graph-based recommender will recommend the top 10 movies which have the largest connecting weights with the movie we want to have recommendations for.

This model suffers from some severe limitations. It is only capable of suggesting movies which are best connected to a certain movie through their cast and crew members. Furthermore, it is not capable of capturing user tastes and providing recommendations across genres or other movie content.

This is obviously a very naive approach, but the results are satisfying for this level of complexity as it is shown on Figure 8.

2) Content-based recommender: Instead of only taking into account the cast and crew the movies share, in this recommender system the content of the movie (genres, cast, crew, keywords, etc.) is used to find its similarity with other movies and recommend those most likely to be similar.

This is also referred as Content Based Filtering - where the recommender suggests similar items based on a particular item. The general idea behind these recommender systems is that if a person liked a particular item, then will also like an item that is similar to it.

The quality of our initial recommender would be increased with the usage of better metadata and a wider spectrum of content features. Thus, we built a recommender based on the following content of each movie: the top 4 actors, the director, related genres and the movie plot keywords. Based on these features we compute pairwise similarity scores for given movies. We use the cosine similarity score since it is independent of magnitude and represents a judgment of orientation, and also is relatively easy and fast to calculate. For a given movie, we compute the K best movie neighbors based on the similarity score and recommend K movies.

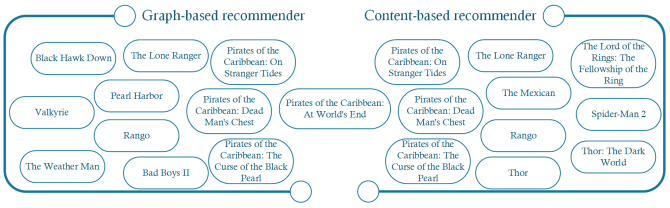


Fig. 8: Comparison between Graph-based and Content-based recommender systems

From Figure 8, it is evident that our recommender has been successful in capturing more information due to more metadata and has given us better recommendations than the Graph-based one. The example on Figure 8 for "Pirates of the Caribbean: At World's End" shows that the recommendations using the naive Graph-based recommender were limited, but here we can notice that the recommender suggested not just the other three parts from the series and those movies starring Johnny Depp, but new ones which correlate with the new given features. We can modify the described model even further by adding more metadata or by changing the weights in the sum for the total cosine similarity between two movies.

However, our content based recommender also suffers from certain limitations. It is only capable of suggesting movies which are close to a certain movie based on the set of features we provide. It is not capable of capturing preferences and providing recommendations across genres but only to similar ones. Also, this engine is not user oriented, in that it doesn't capture the personal tastes and biases of a user. Anyone querying our engine for recommendations based on a movie will receive the same recommendations for that movie, regardless of who the user is, as shown in Figure 8.

Another way of building this content based recommender is through graphs, namely by constructing four different graphs based on our selected features. These graphs should be build by calculating a chosen similarity metric (cosine similarity in our case) between every movie. Then similarly as we have done, the K best neighbors will be calculated by taking a weighted sum over the cosine similarity metric for a given movie from each graph. We believe that the results we gained with our approach are very close to the alternative graph method, as our approach simulates and captures the functionality of the graph method.

3) Collaborative-based recommender: The content based recommender system was not capable of capturing user preferences or providing recommendations across genres. Therefore, in this section, we will use a technique called Collaborative Filtering to make movie recommendations.

Collaborative filtering methods are based on collecting and analyzing a large amount of information on users behaviors, activities or preferences and predicting what users will like based on their similarity to other users. Basically there are two types of collaborative filtering: user based and item based. We will use the item based collaborative recommender system to recommend movies for a user by taking user data for movie rankings. For measuring the similarity between two users we use again the cosine similarity. However, this model has issues with sparsity and scalabillity, where the computation grows with both the users and the ratings. One way to handle the scalability and sparsity issue created by this model is to leverage a latent factor model to capture the similarity between users and items. We will use SVD, which decreases the dimension of the utility matrix by extracting its latent factors. Essentially, we will map each user and each item into a latent space with dimension r. We will do inference on the model by predicting the rating for movies given a user using Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) as metrics. This leads to the final part where we use the predicted ratings to make movie recommendations for a user. Namely, the model suggests 10 movies for a given user, based on predicted ratings for every movie that the particular user has not rated, sorted in descending order.

This system overcomes the problem of suggesting movies which are close to a certain movie based on the set of features we provide as the previous Content-based system. This engine is strictly user oriented, in a way that represents the personal tastes of a user. It works purely on the basis of an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie. The more user ratings data we have, the better.

V. Conclusion

In this project, we analyzed the TMDb dataset containing information about movies from the past several decades. We firstly created a short analysis of the different data features in order to better represent the information our dataset consists of. After that, the general idea was to explore how spectral clustering techniques can be used to classify movie genres using the k-means algorithm in different graph settings, such as cast and crew or movie keyword co-occurrence graphs. Additionally, we used Convolutional Neural Networks on graphs with Fast Localized Spectral Filtering to dive deeper into the movie genre classification problem. The initial findings obtained with spectral clustering were improved slightly by using CNNs. However, the largest problem that we faced throughout the implementation of this project was caused by the fact that our movie labels are multidimensional, and cannot be perfectly reduced to only one genre. An interesting direction of future work would be to determine more precise genre labels, or maybe group several genres into more general labels that would facilitate the correct classification. Finally, we decided to suggest movies to users by creating a graph based recommender system. The results obtained with this recommender were unsatisfying since we used a naive approach. Thus, we created one movie content based recommendation system which suggests movies based on their similarities upon different features, and a collaborative movie recommender system which suggests movies upon user preferences and tastes.

References

[1] The Movie DB. The Movie DB. `https://www.themoviedb.org`, 2018.

[2] The Movie DB. TMDB 5000 Movie Dataset. `https://www.kaggle.com/tmdb/tmdb-movie-metadata/home`, 2018.

[3] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks, 2009.

[4] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396, 2003.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[6] Ulrike von Luxburg. A tutorial on spectral clustering. *CoRR*, abs/0711.0189, 2007.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016.