Text Mining project

Giulia Beccaria [851286], Roberta Di Santo [851272], Lorenzo Lobosco [851289]

Abstract

This project focused on the analysis of movie-related textual data using advanced natural language processing techniques. The primary goal was to develop a predictive model for film ratings and identify meaningful patterns through clustering. The project involved text preprocessing, representation, classification, and clustering. The project aim to enhance the recommendation experience for movie viewers and understand user preferences towards films.

Keywords: Text Mining;Text Classification;Text Clustering; Movies

Contents

1. Introduction

Text mining is an advanced field in analyzing textual data, aimed at extracting meaningful information from unstructured texts. The goal is to understand the context and meaning of documents by identifying patterns, trends, and associations. In our context, we will apply text mining to the extensive world of movies, ratings, and tags. Specifically, we will leverage the text found in the tags assigned by people to movies in our analyses. This will allow us to explore hidden dynamics in reviews, discover correlations among assigned tags, and provide a richer understanding of user preferences towards movies.

A. Aim of the project

Our project sets out to explore the world of films through a comprehensive analysis of data. The primary objective is to develop an advanced model capable of predicting film ratings, leveraging textual reviews as a valuable source of information. Additionally, we aim to identify meaningful patterns through clustering techniques.
In more concrete terms, envision grouping films into homogeneous categories based on similar reviews and tags. This will be achieved by applying clustering algorithms, with the anticipation of discovering connections and affinities among different cinematic works, creating a richer and more detailed framework.
The crux of the project lies in the development of a predictive model. This model, relying on textual reviews, will automatically assign a rating to films. Such an approach will enable a more accurate classification, providing a nuanced view of cinematic works.
In practice, we aspire to enhance the recommendation experience. This involves identifying groups of films that might appeal to a user by analyzing their past reviews. This personalized approach has the potential to significantly improve recommendations, suggesting films aligned with individual preferences.
To accomplish these objectives, we will employ the fundamental application of text mining. By analyzing reviews and tags associated with films, we will use advanced natural language processing (NLP) techniques to extract meaningful information.

2. Dataset

For our analysis we used the dataset ml-latest-small available here[1][2].The dataset (ml-latest-small) comprises 100,836 ratings and 3,683 tags applied to 9,742 films. These data were

sourced from 610 users between March 1996 and September 2018 via MovieLens, without specific demographic information. The dataset is organized into files named links.csv, movies.csv, ratings.csv, and tags.csv, each playing a distinct role in providing details about films, their ratings, and associated tags.

- *User IDs.* MovieLens users were randomly chosen for inclusion, and their IDs have been anonymized. User IDs remain consistent between ratings.csv and tags.csv, meaning the same ID refers to the same user across both files.
- *Movie IDs.* Only movies with at least one rating or tag are part of the dataset. Movie IDs align with those used on the MovieLens website. Movie IDs are consistent across ratings.csv, tags.csv, movies.csv, and links.csv, ensuring the same ID refers to the same movie throughout these four data files.
- *Ratings Data File Structure (ratings.csv).* All ratings are found in the ratings.csv file. Each line after the header row signifies one rating of one movie by one user, with the format: userId, movieId, rating, timestamp. Lines in this file are ordered by userID and then by movieID within each user. Ratings operate on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars). Timestamps represent seconds since midnight Coordinated Universal Time (UTC) on January 1, 1970.
- *Tags Data File Structure (tags.csv).* All tags are housed in the tags.csv file. Each line after the header row represents one tag applied to one movie by one user, formatted as: userId, movieId, tag, timestamp. Similar to ratings.csv, lines are ordered first by userID and then by movieID within each user. Tags serve as user-generated metadata about movies, typically comprising a single word or short phrase. The meaning and value of each tag are determined by individual users. Timestamps represent seconds since midnight Coordinated Universal Time (UTC) on January 1, 1970.
- *Movies Data File Structure (movies.csv).* Movie information is stored in movies.csv. Each line after the header row signifies one movie, with the format: movieId, title, genres. Movie titles, entered manually or imported from [3], include the year of release in parentheses. Genres are represented as a pipe-separated list, selected from various categories.
- *Links Data File Structure (links.csv).* Identifiers for linking to other movie data sources are located in links.csv. Each line after the header row represents one movie, with the format: movieId, imdbId, tmdbId. These IDs correspond to MovieLens, IMDb, and TMDb, respectively, and allow linking to the respective websites. The use of these resources is subject to the terms of each provider.

3. Text Preprocessing

Text preprocessing is a crucial step in natural language processing, involving the cleaning and transformation of raw textual data. This process enhances the quality of text data, reduces dimensionality, and facilitates the effectiveness of machine learning models in tasks such as sentiment analysis, text classification, and information retrieval.

A. Explanatory analysis

We started by importing the datasets and exploring them to understand which could provide useful information for our aim. We have 4 datasets movies, links, ratings and tags. The links dataset is not useful for our analysis since it's used to retrieve more data from the IMDb and the TMDd websites. Then, concerning feature selection, we notice that from the ratings and tags datasets, the column timestamp is present: this column indicates the time in which the rating or the tag was uploaded, for the aim of our research, it is therefore not a useful information. After this initial exploration, we analyze more in depth each dataset, starting from the movies dataset: in this dataset we have 3 columns describing movie Id, title and genre. The title columns a part from the title of the film also contains the year in which the film was published and in the genre column we can find all the genres movies can be traced back to. Since most of the movies can be traced back to more than one genre, we decide to extract the genre for each movie and encode them creating new columns with each genres as the name and allot " 1 " if the movie has the genre else " 0 ".
The most popular genres are Drama, Comedy and Thriller.
The last step of pre-processing for the movie dataset is to extract the Year the film was published from the title column. Then moving on the rating dataset, we removed the timestamp and we looked at the distribution of the ratings:

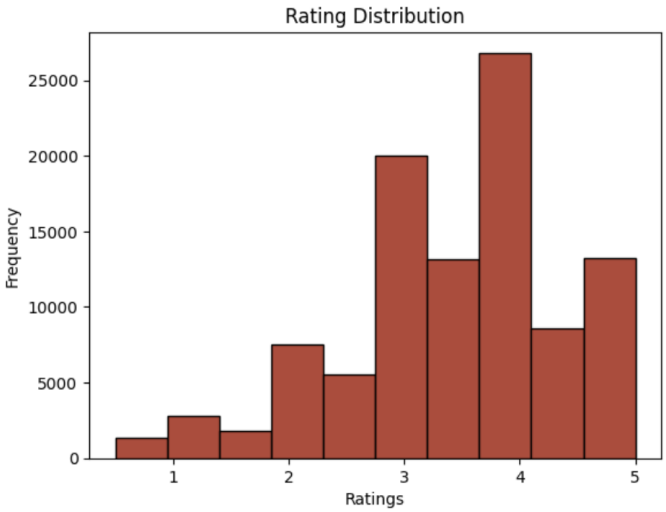

Figure 1 Rating distribution.

As we can see, the rating distribution is skewed on the right, with a predominance in the 4th class.
We check if there are some outliers:
As we can see the distribution goes from 0 to 5, the inter-quartile range goes from 3 to 4 and has a mean of 3.5 with a few outliers.

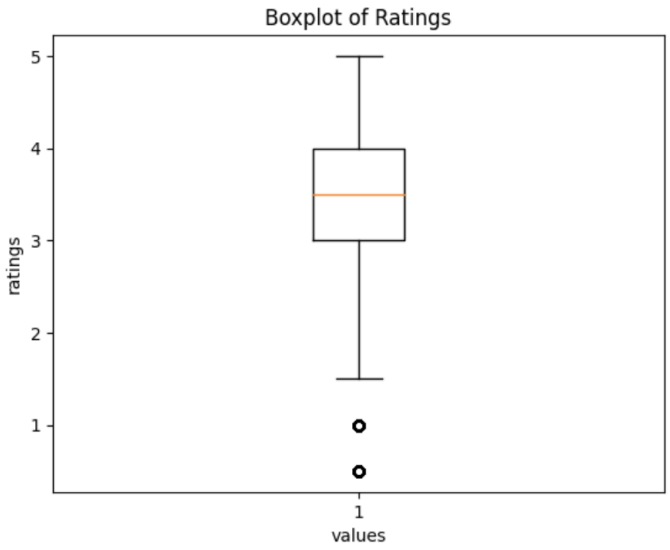

Figure 2 Rating distribution.

Subsequently, we merged the rating and movies dataset obtaining a new dataframe with the columns:

- movieID
- title
- year
- rating: the mean of ratings for each film
- userId
- number of ratings

B. Handling missing values

Missing values were encountered only when merging the movie and rating dataset, because not all movies listed in the movie dataset were rated, therefore we filled the missing values with '0'.

C. Text Processing

In this fase we proceeded to the processing of text. The text is present in the tags dataset.

C.1. Normalization

We started by analyzing the text to check if there were any white spaces, we did this by defining a function to individuate some patterns that are typical HTML tags. Since there were no HTML tags, we proceeded to the next step.
We wrote a function to case fold all the tags in order to make text processing operations, such as searching and comparison, case-insensitive.
Later for the same goal of normalization we removed numbers, punctuation, links and emojis that are not useful for our analysis.
To end the normalization process we removed character repetition and we used beutifulsoup library to remove any remaining HTML tag.

C.2. Tokenization

Tokenization is the process of breaking down text into smaller pieces. To proceed in this face we used the NLTK package, and the WordPunctTokenizer function to tokenize our tags and store them in a new column 'tag_nsw'.

C.3. Stop-words removal

In this step, we eliminate commonly used words found in nearly every sentence, such as 'the,' 'a,' 'an,' 'is,' 'are,' etc. This essential process aims to filter out words that carry minimal semantic meaning for our specific scope, thereby reducing computational complexity. We used the NLTK stopwords vocabulary to do this.

C.4. Stemming/Lemmatization

In the next phase, we apply lemmatization or stemming to the text. This involves reducing words to their base or root form, such as converting 'running' and 'runs' to 'run.' The goal is to achieve a more concise and standardized representation of words, facilitating meaningful analysis and reducing the complexity of the computational task at hand. We used the function PorterStemmer() and WordNetLemmatizer() from NLTK.

Figure 3 Wordcloud of the tags.

4. Text representation

Text representation, in the context of natural language processing (NLP), refers to the process of converting textual data into a format that can be easily understood and processed by machine learning algorithms. The goal is to represent the meaning or content of the text in a way that facilitates various NLP tasks, such as classification, clustering, or language understanding.

We focus on three methods: Bag of Words, Word Embeddings and Contextualized Word Embeddings. These are approaches to convert textual data into numerical vectors, making it suitable for machine learning models. These representations capture different aspects of the text, allowing algorithms to work with the underlying semantic and syntactic information.

A. BoW

It represents a document as an unordered set of words, ignoring grammar and word order but keeping track of word frequency. Different variations, like Binary BoW, TF (Term Frequency), and TF-IDF (Term Frequency-Inverse Document Frequency), offer different ways to quantify the importance of words in the representation.

B. Word Embeddings

These methods map words into continuous vector spaces where the proximity of vectors represents semantic similarity. Models like word2vec, GloVe, and FastText learn these vector representations by capturing semantic relationships and contextual information about words.

Figure 4 t-SNE, Two dimentions.

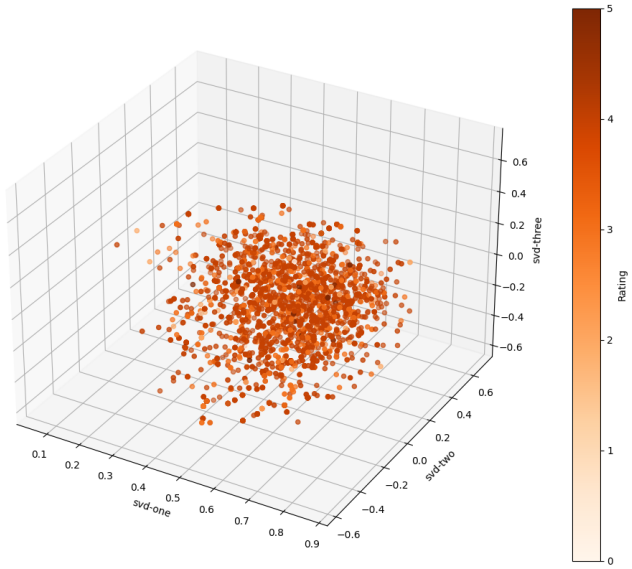

Figure 5 t-SNE, Three dimentions.

C. Contextualized Word Embeddings:

These models, like BERT and ELMo, go a step further by considering the context of each word within a sentence. They generate word representations that are contextualized based on the surrounding words, capturing nuances and complex semantic relationships within the text.

We initially experimented with Word2Vec but discarded it due to unsatisfactory model results, it usually not works well on short texts as it requires a lot of data to learn meaningful embeddings. Then we used TF-IDF (Term Frequency-Inverse Document Frequency) for text vectorization to consider word frequencies, crucial for classification and clustering tasks.
Here's an explanation of why TF-IDF was applied to the DataFrame: The objective is to convert textual data (in the 'tag_nsw' column) into a numerical format that machine learning models can understand and work with. TF-IDF is a common technique in natural language processing that assigns weights to words based on their importance in a document relative to a collection of documents (corpus). It considers both the term frequency (TF), which measures how often a word appears in a document, and the inverse document frequency (IDF), which penalizes words that are common across many documents. The resulting TF-IDF matrix represents each document as a vector, where each element corresponds to the importance of a specific word in that document. We used the TfidfVectorizer from scikit-learn is used to create a TF-IDF matrix from the 'tag_nsw' column. The TF-IDF matrix is then converted into a DataFrame (tfidf_df), where columns represent unique words in the corpus.

Then we implement the t-SNE to reduce the TF-IDF data to two and three dimensions and creates a scatter plot where each point represents a movie.The points are colored based on their 'rating_class,' providing a visual representation of the TF-IDF features in a lower-dimensional space. t-SNE is commonly used for visualizing high-dimensional data in a way that preserves local relationships between points.

5. Text Classification

Text classification is the task of automatically assigning predefined categories or labels to a given text document. It is done to categorize and organize textual data, making it easier to analyze, search, and extract meaningful information. Text classification is commonly used in various applications, such as spam detection, sentiment analysis, topic categorization, and content recommendation, to enhance information retrieval and decision-making processes.

We decided to try two methods for text classification, one where we try to classify films based on the tags we get in one of the rating classes and the second one to classify the films based on the rating, when the rating is lower than 3 we have a negative tags and when it is higher we have positive tags.

A. Ordinal Classification

For the first type of classification we decided to do Ordinal Classification, this type of classification is a supervised learning task that consists of estimating the rating of a data item on a fixed, discrete rating scale. To employ this task we divided the ratings in 6 classes from 0 to 5. Then we vectorized the data items in order to obtain data types useful to train a model. We tried 2 ways of vectorizing and we chose the TF-IDF for the reasons described earlier in section 4. The model's effectiveness is evaluated through a confusion matrix and a classification report, providing insights into its ability to classify instances into different classes.

A.1. Performance

The confusion matrix shows how our model has an accuracy of 72% and performs really well especially in class 4 which is the most popular class.

B. Binary Classification

For the binary classification task, we categorized our data into two classes: positive for ratings higher than 3 and negative for ratings lower than 3. Subsequently, we assigned the label 1 to instances in the positive class and 0 to those in the negative class.

Recognizing a significant imbalance between the two classes, with one being substantially more prevalent, we addressed this issue by applying an oversampling technique to rebalance the class distribution. Once the classes were balanced, we proceeded with the vectorization of both the training and testing data. Subsequently, we employed various models to explore the predictive performance on the balanced dataset. The models employed were:

- **Logistic Regression**
  Logistic Regression is a statistical model commonly used for binary classification tasks, predicting the probability that an instance belongs to a particular class.

- **Support Vector Machine**
  Support Vector Machine is a powerful supervised learning algorithm. SVM works by finding the optimal hyperplane that maximally separates data points of different classes in a high-dimensional feature space.

- **Random Forest**
  Random Forest is an ensemble learning method widely used for both classification and regression tasks. It builds multiple decision trees during training and merges their predictions to improve accuracy and reduce overfitting.

- **Gradient Boosting (XGBoost)**
  Gradient Boosting, specifically exemplified by XGBoost (Extreme Gradient Boosting), is a powerful machine learning technique that builds an ensemble of weak learners, typically decision trees, in a sequential manner. XGBoost minimizes a loss function by adding new trees that correct errors made by the existing ensemble. It incorporates regularization terms to prevent overfitting and employs efficient algorithms for parallel processing, making it computationally efficient.

B.1. Performance

Between these models the one performing better is the XGBoost with a 0,87 Accuracy score, followed by Logistic Regression with an Accuracy of 0,807 and the Random Forest with Accuracy score of 0,805.

6. Text Clustering

The motivation for applying text clustering to our film dataset stems from the need to organize and categorize films based on inherent similarities in their associated tags. By doing so, we aim to achieve several objectives:
Clustering films with similar tags allows for the creation of clusters that may be of interest to users who have shown a preference for specific themes or genres.
Identifying clusters of films with similar tags enables the analysis of emerging trends or popular themes within the dataset.
For datasets with a large number of films, text clustering can efficiently organize the catalog, facilitating easy navigation for users to find similar or related films.
To implement text clustering on this dataset, we will represent the textual data, such as tag labels, in a format suitable for clustering algorithms. Subsequently, algorithms like k-means, hierarchical clustering, DBSCAN, and agglomerative clustering can be applied to obtain desired results.

A. Models

In this subsection, we will delve into various clustering algorithms that we employed for text clustering on our film dataset.

- **K-Means.**
  K-means is a fundamental flat clustering algorithm that assumes documents are represented as length-normalized vectors in a real-valued space. Its objective is to minimize the average squared Euclidean distance of documents from their cluster centers.

- **Hierarchical Clustering.**
  Hierarchical clustering creates a hierarchy of clusters, either bottom-up (agglomerative) or top-down (divisive). Agglomerative clustering starts with individual documents as singleton clusters and merges them iteratively, forming a dendrogram.

- **DBSCAN.**
  DBSCAN is a density-based clustering algorithm that groups together data points based on their density in the feature space. It is particularly effective in identifying clusters of varying shapes and sizes.

- **Agglomerative Clustering.**
  Agglomerative clustering is a hierarchical clustering method that starts with each document in a separate cluster and successively merges the closest pairs of clusters until a single cluster remains.

B. Evaluation

we assess the quality of our clustering results using both internal and external evaluation metrics.

- **Silhouette Score.**
  The Silhouette analysis measures how well an observation is clustered. It estimates the average distance between clusters, providing insight into the cohesion within clusters and separation between them.

- **Purity.**
  Purity is an external evaluation metric that measures the agreement between the clusters and a predefined ground truth. It assesses how well documents are assigned to the correct classes.

- **Rand Index.**
  The Rand Index is another external evaluation metric that measures the percentage of correct decisions. It considers true positive, true negative, false positive, and false negative decisions to evaluate the accuracy of clustering results compared to ground truth.

- **Precision, Recall, and F-measure.**
  Precision, recall, and the F-measure are widely used metrics in information retrieval tasks. Precision measures the accuracy of positive predictions, recall assesses the ability to capture all positive instances, and the F-measure balances both metrics.

C. Results

Firstly, we sought to determine the optimal number of clusters (K) within our dataset. We obtained interesting yet challenging-to-interpret results. As depicted in the following graphs, we

explored K values using the k-means algorithm and sought the highest possible Silhouette Score.

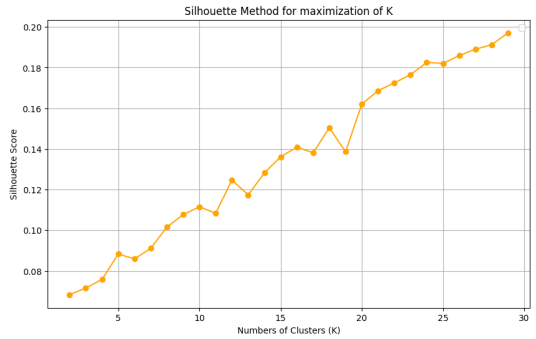

Figure 6 K values from 2 to 30

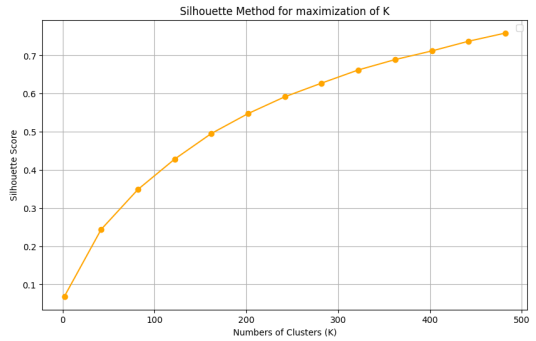

Figure 7 K values from 2 to 500

As observed in the graphs, the Silhouette Score significantly increases with an exponential growth in the number of clusters. Given the text-based nature of clustering, it is plausible for this indicator to approach 1 as the number of clusters approximates the number of words. Subsequently, we applied alternative clustering algorithms, such as hierarchical clustering.

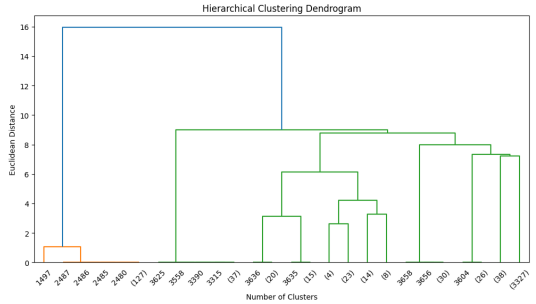

Figure 8 Hierarchical clustering

In selecting the number of clusters and the optimal algorithm, we conducted multiple trials and evaluated each using the metrics explained earlier. We present results for three cases: K=3, K=8, and K=14.

Algorithm	Silhouette Score	Purity	Rand Index	Precision	Recall	F-measure
K-Means	0.072	0.752	0.006	0.752	0.961	0.844
Hierarchical	0.079	0.752	-0.001	0.752	0.953	0.841
DBSCAN	0.498	0.764	-0.012	0.764	0.385	0.512
Agglomerative	0.079	0.752	-0.001	0.752	0.953	0.841

Figure 9 Results for K=3

Algorithm	Silhouette Score	Purity	Rand Index	Precision	Recall	F-measure
K-Means	0.102	0.753	0.007	0.753	0.918	0.827
Hierarchical	0.114	0.752	-0.029	0.752	0.897	0.818
DBSCAN	0.498	0.764	-0.012	0.764	0.385	0.512
Agglomerative	0.114	0.752	-0.029	0.752	0.897	0.818

Figure 10 Results for K=8

Algorithm	Silhouette Score	Purity	Rand Index	Precision	Recall	F-measure
K-Means	0.128	0.752	-0.041	0.752	0.88	0.811
Hierarchical	0.152	0.752	-0.023	0.752	0.858	0.802
DBSCAN	0.498	0.764	-0.012	0.764	0.385	0.512
Agglomerative	0.152	0.752	-0.023	0.752	0.858	0.802

Figure 11 Results for K=14

In most cases, the results improve with an increasing number of clusters. To illustrate concrete outcomes, we selected K=8 and examined how tag groups and word clouds for each group were created. The most frequent words in each cluster are depicted below.

```
Most representative word in Cluster 0: awesom
Most representative word in Cluster 1: atmospher
Most representative word in Cluster 2: psycholog
Most representative word in Cluster 3: alien
Most representative word in Cluster 4: comedi
Most representative word in Cluster 5: netflix
Most representative word in Cluster 6: scifi
Most representative word in Cluster 7: school
```

Figure 12 Most representative tag for each custer

7. Conclusions

The project aimed to leverage advanced natural language processing techniques to analyze movie-related textual data. The primary objective was to develop a predictive model capable of predicting film ratings and identifying meaningful patterns through clustering techniques.

Throughout the project, various steps were undertaken, including text preprocessing, representation, classification, and clustering. Text preprocessing involved tasks such as normalization, tokenization, stop-words removal, and stemming or lemmatization.

In the context of text representation for our project, our approach involves carefully selecting the most efficient vector method to accurately represent the textual data. Notably, we have taken the initiative to exclude less suitable methods from consideration, ensuring a focused and effective choice for our project's objectives These techniques allowed for the extraction of meaningful features from the text, enabling further analysis and modeling.

The project's findings and results contributed to a deeper understanding of user preferences towards movies, as well as

the potential for enhancing the recommendation experience for movie viewers.

Overall, the Text Mining project demonstrated the application of advanced NLP techniques in the analysis of textual data, with a specific focus on movie-related information. The project's outcomes may have implications for improving movie recommendations and understanding user sentiments towards films.

References

[1] GroupLens. *MovieLens Small Dataset*. Small: 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. Last updated 9/2018. 2018. URL: https://grouplens.org/datasets/movielens/latest/.

[2] Analytics India Magazine. *10 Open-Source Datasets For Text Classification*. Mar. 2020. URL: https://analyticsindiamag.com/10-open-source-datasets-for-text-classification/.

[3] The Movie Database (TMDB). *The Movie Database (TMDB)*. Millions of movies, TV shows, and people to discover. Accessed. URL: https://www.themoviedb.org/.