MovieHut.com - a Film Search Engine powered by Movie Scripts and other Movie Metadata

António Ferreira
Faculty of Engineering of the University of Porto
Porto, Portugal
up202004735@fe.up.pt

Francisco Serra
Faculty of Engineering of the University of Porto
Porto, Portugal
up202007723@fe.up.pt

João Maldonado
Faculty of Engineering of the University of Porto
Porto, Portugal
up202004244@fe.up.pt

Tomás Gomes
Faculty of Engineering of the University of Porto
Porto, Portugal
up202004393@fe.up.pt

Abstract

In today's day and age, the film market can be characterized as over saturated. With the immense catalog of films currently available, it's impossible to know every movie and its many details. Taking this into account, with this paper we'll present the development of MovieHut.com, a movie search engine, through the documentation of the milestones that compose this group project of the PRI course.

CCS Concepts: • **Information Systems → Information Retrieval**.

Keywords: Information Retrieval, Information Processing, Information Theory, Data Characterization, Data Sets

ACM Reference Format:
António Ferreira, Francisco Serra, João Maldonado, and Tomás Gomes. 2023. MovieHut.com - a Film Search Engine powered by Movie Scripts and other Movie Metadata . In *Proceedings of Information Processing and Retrieval (PRI).* ACM, New York, NY, USA, 4 pages.

1 Introduction

Since its introduction in 1895, cinema and the film industry have become staples of modern culture for more than a century. However, in today's day and age, due to the constant releases of new movies and the over saturation of the film market, it is impossible to have the knowledge about every piece of cinema currently available. Therefore, as we are living in the era of social media, it's not uncommon to come across, while scrolling through one of the many apps, various references to movies that are unknown to us, whether it's in the form of a single quote from the script, still images from the film or full movies divided by parts.

Considering this, we decided to tackle this problem and take it as inspiration for our project's theme. Our goal is to compile various movie scripts with metadata relative to their respective film, in order to create the ultimate movie search engine: a collection of searchable 600+ film titles, each characterized by their cast, IMDb rating, and script.

2 Dataset

Upon deciding on a topic, the first main task was focused on searching dataset sources and choosing the ones that best fit our needs. Taking into consideration the goal of this project, we prioritized datasets that included a substantial amount of text data. In our case, we tried to find **movie script datasets**.

2.1 Data Selection

The chosen dataset was a **"Movie Dialog Corpus"** dataset obtained from Kaggle [1], from 2017. It consists of a metadata-rich collection of fictional conversations from 617 raw movie scripts, accumulating to 30.12MB of data, referring to 304413 lines, 83054 conversations, and 9033 characters.

This dataset was originally used for a Cornell University paper entitled "Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogues" by Cristian Danescu-Niculescu-Mizil and Lillian Lee. The main inspiration behind this paper was to study how movie dialogues represent real life conversations and how we can identify various themes in movies from specific writers or directors.

As a way to complement this data and make our search engine more thorough, we opted to include information on the cast of each film. To complete this, we utilized the **The Movie Database** [2] API. This platform consists of a community curated database, that stores metadata relative to an ample size of movies and TV shows. Through this API, we were able to obtain the cast of each film present in our chosen dataset, which then allows to link each character to their respective actor. We also retrieved other movie metadata, like revenue, duration in minutes and promotional poster.

2.2 Data Processing and Pipeline

The first step in creating data for our project was handling the five .tsv files, included in our movie dialogue dataset.

Before selecting this data source, we not only analyzed the relevance of the information to the topic in question, but also the quality of the data. During this process, we observed that, in 3 files, many rows were not properly formatted, accumulating all the information in one column. With this in mind, using Microsoft Excel, we mended the broken columns and converted all five files to the.csv format. It should also be noted that, in this initial stage, we also verified its correctness. By randomly choosing ten movies, we compared their scripts, character names and flow of the dialogues with other trusted sources.

The next stage was setting up the data pipeline. To store the data for our search engine documents, we chose to create a SQLite database, because of its efficient way of data storage and fast reading and writing operations. That being said, in order to extract the data from the dataset files into our created database, we opted to use Python, alongside the Pandas and SQLite3 libraries.

After reading the *movie _titles _metadata.csv* file into a Dataframe variable, we extracted the genres, normalized them (by removing unnecessary characters), stored them in a set of genres, which was then transformed into a list of tuples of the type (id, genre) and finally stored the data into a **genre** table. Since there's a many-to-many relationship between movies and genres, we then created a **movie _genre** pivot table, linking each genre id to a movie id. Subsequently, from the Dataframe variable, we populated the **movie** table. Then, after retrieving the data from the *movie _characters _metadata.csv* file, since the columns relative to the gender and the position in the credits of the character had many '?' values, we converted them into NULL, before populating the **character** table. Finally, upon extracting the data from the *movie _conversations.csv* and *movie _lines.csv*, besides normalizing the values, we dropped the lines column from the conversations Dataframe variable, added a conversation id to each line of the lines Dataframe variable and created the **conversation** and **line** tables.

The next step was to integrate the data obtained from the chosen API into the database. Besides the libraries referenced above, we also utilized the Requests and JSON Python libraries, in order to send HTTP requests and read the resulting data. After fetching all movie titles and all the characters and their respective movies, from the database, we created four dictionaries where the keys were film titles and the values were: character names from our database; and character names from the API; actor names from the API; and movie metadata from the TMDB API. Besides that, we also created an **actors** table and added the additional information to the database.

The main challenge in this part of the project was connecting each character to their respective actor. Even though we had the ability to extract a film's cast through the API, the names present in the resulting JSON file did not fully match those in the database. Given this circumstance, we resorted to the use of fuzzy string matching [3], by importing the Thefuzz Python library. Fuzzy matching consists of determining how close two strings are by calculating the "edit distance" between them. In this context we used the **Levenshtein Distance** to calculate the differences between sequences of characters. Taking this into account, we performed a O(n^2 * m) search through each movie title, iterating through the list of character names in the film, stored in the database, to check if there is a 90% or higher fuzzy match with a character name present in the cast obtained through the TMDB API. If this condition reveals to be true, we stored the best found match (the character name with the highest fuzzy match ratio) and connected the corresponding actor to the character. However, due to poor results, we also considered a valid condition to check if the name of the database character is a substring of the API character (and vice versa). Consequently, we obtained a match rate of**70%**. Although these represent great results, there is still the underlying issue of characters without matching actors.

Finally, after going through all of these steps, we achieved a robust pipeline (fig.1) that will later power the documents that compose our search engine.

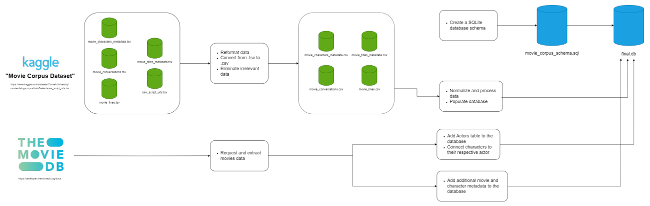
kaggle
"Movie Corpus Dataset"
https://www.kaggle.com/datasets/Cornell-University/movie-dialog-corpus/data?select=raw_script_urls.tsv
movie_characters_metadata.tsv
movie_titles_metadata.tsv
movie_conversations.tsv
raw_script_urls.tsv
movie_lines.tsv
Reformat data
Convert from .tsv to .csv
Eliminate irrelevant data
movie_characters_metadata.csv
movie_titles_metadata.csv
movie_conversations.csv
movie_lines.csv
Create a SQLite database schema
movie_corpus_schema.sql
final.db
Normalize and process data
Populate database
THE MOVIE DB
https://developer.themoviedb.org/docs
Request and extract movies data
Add Actors table to the database
Connect characters to their respective actor
Add additional movie and character metadata to the database

Figure 1. Pipeline Diagram

2.3 Data Conceptual Model

The conceptual model (fig. 5) consists of several interconnected tables that represent various aspects of the movie dataset:

- **Character Table**: Stores character information with attributes like name, associated movie, gender, and credit position. It references the actor who portray a character.
- **Conversation Table**: Records character conversations in movies. Includes references to characters and movies involved.
- **Genre Table**: Stores movie genres with unique genre IDs and genre names.

- **Line Table**: Contains individual movie dialogue lines with attributes like character, movie, and line text. Linked to conversations.
- **Movie Table**: Stores movie data, including title, release year, IMDb rating, IMDb votes, generated revenue, runtime (in minutes) and a poster url.
- **Movie Genre Table**: Establishes movie-genre relationships using movie and genre IDs, ensuring uniqueness.
- **Actor Table**: Keeps track of actors, assigning unique IDs and storing their names.

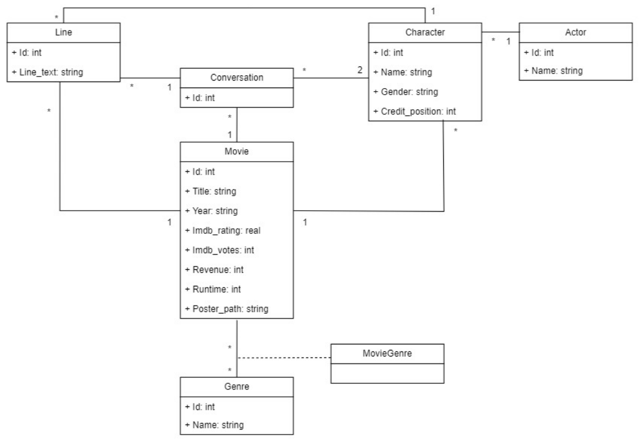

Figure 2. Dataset Conceptual Model

2.4 Data Characterization

In our exploration of the movie dataset, it was essential to assess its quality. We beginned by analyzing the distribution of IMDb ratings for each movie (fig. 3). IMDb is a well-known platform for evaluating movies, and high ratings often indicate popularity and quality.

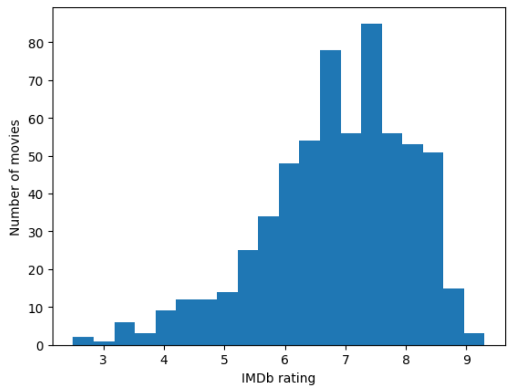

Figure 3. Distribution of IMDb ratings.

The majority of movies in the dataset have IMDb ratings of 6 or higher, which is a positive sign. Typically, movies rated 7 or higher on IMDb are considered good. However, IMDb ratings alone do not provide a complete picture of movie quality. We must also consider the quantity of votes each movie has received. A single high rating may not accurately represent a movie's quality.

To address this, we had categorized movies based on the number of IMDb votes they have received into the following groups:

- 0-100 votes
- 100-1000 votes
- more than 1000 votes

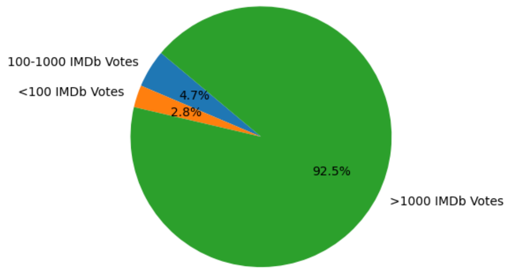

Figure 4. Distribution of the quantity of IMDb votes.

To gain a deeper understanding of the dataset's richness, it was essential to assess the quantity of lines in each movie (fig. 5). This metric can provide insights into the depth and complexity of the dialogues within the films.

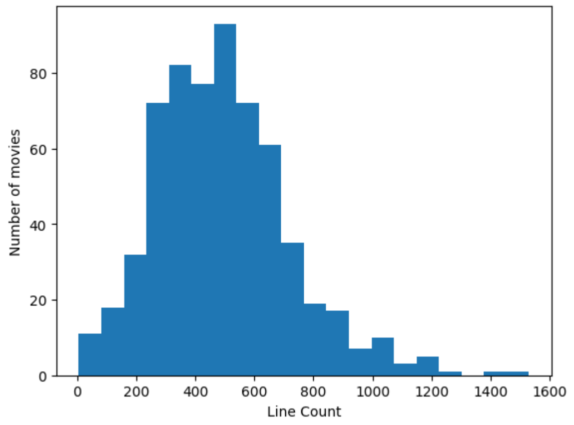

Figure 5. Distribution of Lines in the Movies Dataset.

The distribution of lines per movie revealed that the majority of movies fall within the range of 300 to 700 lines. This was an encouraging finding as it suggests that the dataset contains movies with substantial dialogues and potentially rich content.

Continuing our analysis, we turned our attention to movie genres. Understanding which genres are most popular in the dataset can provide valuable insights (fig. 6).

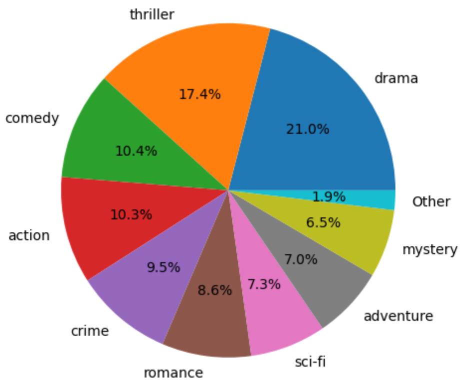

Figure 6. Distribution of movie genres.

As expected, the most common genres in the dataset are drama, thriller, comedy, and action.

To gain further insights, we exploreed the most frequently used words in the dataset (fig. 7). A word cloud visualization helps us identify these words, and **we have excluded common stop words.**

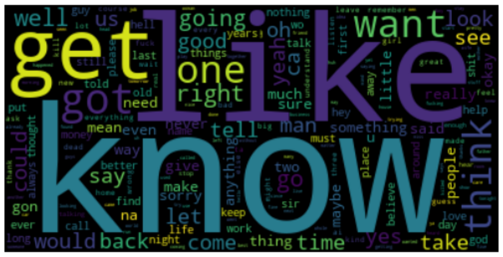

Figure 7. Word cloud of dialogue lines.

Finally, we examined the popularity of actors based on the number of characters they have portrayed in movies (table. 1).

Actor Name	Character Count
Nicolas Cage	10
Robert De Niro	8
Johnny Depp	8
Harrison Ford	8
William Shatner	7
Sigourney Weaver	7
Patricia Arquette	7
William H. Macy	6
Philip Baker Hall	6
Dustin Hoffman	6

Table 1. Top 10 actors by character count.

From the table presented, it is possible to affirm that Nicolas Cage is the actor that played the most characters.

This analysis provides a comprehensive overview of the dataset's characteristics, including movie quality, genre distribution, word usage, and actor popularity.

3 Prospective Search Tasks

In this project, the collection of documents represent the target information where each user can obtain information from.

3.1 Search Scenarios

As we have already stated, we seek to develop a search engine that focuses on responding to queries, related to this project's topic, by providing the user with relevant and useful information. Some examples of these queries include:

- "In what movie did the actor X say Y?"
- "Who said X?"
- "What character is played by X in movie Y?"
- "What was the content of the conversation between X and Y in movie Z?"
- "What comedy movies did actor X play in?"

3.2 Document Description

To meet our search scenario needs, we have defined three of documents. These documents represent the search results in our search engine and are divided in three types:

- **Movie** - will include its title, list of characters and their respective descriptions, cast, movie genres, poster, runtime, generated revenue, IMDb rating and votes.
- **Actor** - will include their name, movies they starred in and characters played and other actors they've worked with.
- **Conversation** - will include the its lines, the movie, characters and respective actors involved and previous and next conversations, to provide more context to the dialogue in question.

4 Conclusion

In the end of this first milestone, we created a pipeline, capable of serving as a base for powering the many types of documents we presented above. Our next goal is to create a system capable of satisfying the mentioned search scenarios using Apache Solr [4] and the generated dataset.

References

[1] Cornell University. Movie dialog corpus. URL https://www.kaggle.com/datasets/Cornell-University/movie-dialog-corpus.
[2] The movie database (tmdb). URL https://www.themoviedb.org/.
[3] Eric Silva. What is fuzzy matching? | redis. URL https://redis.com/blog/what-is-fuzzy-matching/.
[4] Welcome to apache solr - apache solr. URL https://solr.apache.org/.