Netflix Search System

Paulo Ribeiro
up201806505@edu.fe.up.pt
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

Pedro Ferreira
up201806506@edu.fe.up.pt
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

Pedro Ponte
up201809694@edu.fe.up.pt
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

Abstract

This paper presents the creation of an information search system for Netflix shows. It will gather information from a dataset containing all the featured shows on Netflix and filter them by title, summary, release year and other pertinent criteria. The accessible content of each show on the website IMDb will ideally supplement the non-available information.

Keywords: datasets, data processing, information retrieval, retrieval evaluation, information search system

1 Introduction

This project works on top of a dataset about Netflix, an American subscription streaming service [see ref. 8]. This dataset contains information on its available shows, namely their titles, summaries and cast, for example.

The development of this project was composed of three milestones, them being:

- Data Preparation
- Information Retrieval
- Search System

From a first approach, we focused on preparing the collected data by adapting it to our preferences and fixing existing problems, such as missing or invalid values. So, it was required to define a concise pipeline for the data processing and understand our problem domain. After all the preparation, we conducted an exploration phase in which we displayed various data through various graphs to analyze the outcomes.

The second phase achieves the implementation and use of an information retrieval tool besides a few experiences of applying some queries to the data and analyzing the outcomes. It implies the construction of a solid schema so that the results are the most relevant for the user searching through the data. Then, some example queries demonstrate the behaviour of the search system in two different situations: one with no schema defined and the other with an enhanced schema that includes filters and boosts, for example. The retrieved results evaluation is present in tables and plots, measured by different metrics.

2 Milestone 1 - Data Preparation

The goals of the Data Preparation milestone are to correctly prepare the data for the information retrieval tool to be implemented in the second milestone.

The first step is to explore different datasets and choose the most captivating, which happened to be a dataset about Netflix, an American subscription streaming service. Therefore, the chosen dataset contains information on its available shows, namely their titles, summaries and cast, for example.

After being familiar with its data, the next step is to think of a possible pipeline for the data processing, adequate for our problem.

Upon defining this pipeline, some essential steps are necessary, such as assessing the data quality by looking for missing or invalid values and handling them, besides fixing other minor problems. The dataset contained unreasonable values in some fields, such as the summary and cast. Thus, data extraction was applied to fill or update the missing or invalid values (using the IMDb website to retrieve the Netflix shows data).

Finally, a data exploration stage is required, which entails gathering relevant information and creating distinctive charts to show the data characteristics. For instance, the plots relative to the genres of the shows allowed us to deduce the most common genre, which was Drama.

2.1 Data Collection

For this step, our first move was choosing an interesting collection of data. This meant picking a dataset with a somewhat large amount of information that would be useful and relevant for the given purpose.

While seeking a dataset worthy of fitting the requirements, various options that ended up not being worked upon came across. Datasets related to videogames, as a gathering of all players present in the EA sports franchise FIFA or club and sportsman data from SEGA game Football Manager, were some considered examples. The digital distribution service Steam, from Valve corporation, was also taken into account. Although these datasets were able to fill almost all the requirements needed, the lack of sizeable texts of individual information could be a non so good aspect that would limit us in the future.

So, the final choice ended up related to the topic of Netflix's TVSeries and movies. Two different datasets about this topic were found. The choice on which one to go with was based on the existence of a vaster number of relevant parameters and the presence of a summary column that would be useful for full-text searches needed for future work. This dataset was retrieved from Kaggle and was the basis of our work [see ref. 3].

The information was also gathered from the IMDb website. This would facilitate the search for missing intel and bring confidence to the work while using a trustworthy source for the data collected since IMDb is a well-known and prestigious platform for online TV series and films [see ref. 2].

The dataset chosen has information in the form of the following columns:

- *imdb_id*: Show id from the IMDb website;
- *title*: The title of the show;
- *popular_rank*: The popular rank of the show;
- *certificate*: The age restriction of a show;
- *startYear*: Release year of a show;
- *endYear*: Year the show ended;
- *episodes*: Number of episodes of a show;
- *runtime*: Mean time duration of a show;
- *type*: Type of a show;
- *orign_country*: Origin country of the show;
- *title*: The title of the show;
- *language*: The language a show was produced on;
- *summary*: Summary of the show;
- *rating*: IMDb rating of the show;
- *numVotes*: Number of votes for show rating;
- *genres*: Genres categories of a show;
- *isAdult*: Presence of adult content in the show;
- *cast*: Cast of the show;
- *image_url*: Poster of the show.

2.2 Pipeline Design

Following our decision on which dataset to explore, our next step is to define a concise pipeline for the data processing such that we can order its steps in a way that is appropriate for our problem. To better visualize the data flow, we started by documenting it with a diagram. This diagram is in the *Attachments* section, Figure **1**.

The starting point of our pipeline consists of the Netflix dataset, which we retrieved from Kaggle [see ref. 3], in a CSV format.

Then, our first step is to clean this data by fixing some issues and adapting it to our best interest. That includes removing some irrelevant columns from our dataset, renaming others and refactoring some data formats to make them more easily accessible. The previous step results in a clean CSV file, where these changes are already applied.

After this, we thought it was interesting to fill the missing values of the dataset with recent information gathered from the web. The indicated step is called Data Extraction. We decided to resort to the IMDb website and apply some scraping to complete the information about the shows since it is the world's most popular and authoritative source for this type of content. The creation of the initial dataset using data from this same source also aided in the decision. Because we wanted a consistent dataset, the freshly obtained data shouldn't deviate too much from the old.

The mentioned process results in the last CSV file, from which can now be performed the Data Exploration step, where we gather some significant statistics of our problem and generate some relevant plots better illustrating some of its characteristics.

Finally, our last step is to store the data so that its access becomes easy and efficient. For this, we decided to use a MySQL database. After all these steps, the data is now ready to be retrieved every time it is needed.

This pipeline is reproducible in a Makefile, from which it's possible to run each of its steps. For creating it, the gathering of information from the book Data Science at Command-Line [see ref 10] was essential.

2.3 Data Cleaning

Initially, we have a dataset with 7008 rows and 19 columns. Some of the columns contain useless data, some have duplicate data, while others feature similar categories that may be combined.

By analysing the dataset on *OpenRefine* [see ref. 6], we conclude that the column *isAdult* adds no value to our dataset since it has all values as '0', so we removed it. We also checked that the columns *plot* and *summary* are very similar, but the *summary* has more information than the *plot*, so we opted to delete the *plot* column.

Then, we noticed some pairs of repeated titles, although one has leading whitespace whereas the other doesn't. We found out that the ones with no initial space represent a 'tvSeries' and the others were a 'tvEpisode'. After some search on the IMDb platform using the *imdbID*, we concluded the 'tvEpisodes' are, in reality, episodes of the 'tvSerie' with the same name, so we opted to delete all the rows that have 'tvEpisode' as *type*.

Continuing *type* analysis, the dataset only has one entry with 'videogame'. Since we consider that only having one value of this type doesn't add value to our future search system, we decided to drop this entry. Two rows were also deleted since, in addition to having a null *type*, do not have other several values defined. After removing these types, we still have: 'tvSeries', 'tvMiniSeries', 'tvShort', 'tvSpecial', 'tvMovie', 'video', 'movie', 'short'. We decided to rename 'tvSeries' for 'series', 'tvMiniSeries' for 'miniSeries', 'tvShort' for 'short', 'tvSpecial' for 'special', 'tvMovie' for 'movie' and 'video' for 'animation'.

Finally, we decided to rename some of the columns to have them all in *lowerCamelCase* style and, for each *genres* column entry, put all its values inside an array.

After finishing the cleaning step, we have a dataset with 6216 rows and 17 columns.

2.4 Data Extraction

Upon finishing the data cleaning process, we proceed to an attempt to fill in the missing values.

To do this, we used Python and a library called IMDbPY [see ref. 1]. With this library, we can get almost all types of information about the IMDb shows.

In the case of the *certificate* column, initially, we had only a certificate for each show, but the country to which the certificate applies was not always the same. Taking this into account, we decided to extract from IMDb all the age certificates corresponding to each show and store them into a dictionary converted into a string, in the format of *country: certificate*.

To try to complete some of the missing values in the *startYear* and *endYear* columns, we filled all the shows that are missing both values simultaneously and all the series or miniSeries that don't have the *endYear* value.

To complete the null values in the *episodes* column, we have scraped from IMDb the number of episodes of each series or miniSeries which are missing this value.

For *runtime*, *originCountry*, *language*, *summary*, *rating*, *numVotes*, *genres* and *cast* columns, we try to extract from IMDb the missing values of these dataset parameters.

2.5 Data Exploration

Now that our data is finally clean and complete, we are ready to start exploring all its 6216 entries. In this stage, we'll create some relevant graphics to help us better visualize our data domain and, potentially, draw some conclusions. We used Python to achieve these results, namely the **Pandas** library to easily handle the dataset [see ref. 11], the **MatPlotLib** [see ref. 7] and **Seaborn** [see ref. 12] libraries to generate the charts and the library **Plotly** [see ref. 9] to create a statistics table.

Our first thought was to get an overview of one of the most significant attributes of our dataset: the type of the show. We decided that a Bar Chart was the most suited approach to see the dimensions of each of those types (see Figure **2** in *Attachments*).

Looking at the data, it appears that movies and series dominate the dataset, followed by specials, mini-series, shorts, and less common animations.

Then, we thought it was interesting to study the evolution of the ratings over the decades. Thus, we chose a Box Plot because it provides the most information about the distribution, such as ranges, outliers, and quartiles. Before generating the graphic, we first had to group the years by decades, exclude the "Not available" values and sort the decades.

As we can see in Figure **3** of the *Attachments*, the number of shows from recent decades is naturally higher than the number of older shows. As a result, the grading range in recent decades has widened. The 21^{st} century registers the minimum and highest rating values. The mean of the ratings has also been slightly decreasing throughout the years. It's also possible to notice some outliers, marked by the dots.

Next, we wanted to see the distribution of the languages and origin countries. There were too many different values in these columns to display in a single graph. Thus, the best and cleaner way to solve this problem was to create a Pie Chart, where one of the divisions was called "Others". This chart should show the nine most common languages/origin countries while the *Others* division would encompass all the less frequent ones. Both of these graphs are available in the *Attachments*, Figures **4** and **5**.

We also thought it was interesting to show the distribution of the genres. That was a more challenging quest since each show can have many genres associated. As a result, we developed and populated a dictionary with a count for each genre, indicating the times that genre appears in a show. Then, we generated a Bar Chart with the nine most common genres and the *Others* column encompassing the less frequent ones, which is available in Figure **6** of the *Attachments*.

Similarly, it would be relevant to know the actors with the most appearances in our dataset. We used the same technique employed in the genre chart again and ended up with a dictionary that included every actor and their number of appearances. After this, we generated the chart with the top 20 actors with the highest number of performances (see Figure **7** of the *Attachments*).

Finally, we wanted to know some basic statistics of our dataset, such as the means of the runtimes of each show-type and the total number of distinct values of some columns, which we gathered in a table (see Figure **8** of the *Attachments*).

2.6 UML Design

A well-thought-out UML was necessary to create a database capable of safely and efficiently storing all of the data, and it can bee seen in the Figure **9**.

The main class *ImdbEntry* was created to represent each show. This class is composed of the parameter *imdbID*, *title*, *popularRank*, *episodes*, *runtime*, *rating*, *summary*, *numVote* and *image*.

As there are two columns dedicated to years on our dataset and repeated values for many shows, the creation of a new class *Year* was necessary, having two different relations with *ImdbEntry* (one for the startYear and the other for the endYear).

Furthermore, we created five new classes since they had the same issue, where the values would be very repetitive. These were: - The class *genre* - with the genres a show can have - *type* - class with the various types a show can be, whether it is a TV series, film, etc. - *actor* - class with the name of an actor that can participate in a show. - *language* - languages a show can be originally made on. - country - countries where the show took place.

Finally, a new class Certificate was added, as well as a connection to the ImdbEntry. This object also links to the country class and a new classification class that will give the actual value for the certification. This new class classification will depend on the correspondent country. This architecture happens because different countries classify their shows on numerous scales and will be helpful in future work, allowing us to get search results in all the distinct classifications.

2.7 Search Tasks

In the final project it is intended to have accomplished the following search tasks with the correct values retrieved:

- Usage of full-text search applied to *title*, *summary*, *cast* members and *IMDbID*. For instance, the search "Pinocio" will retrieve values such as the show "Pinocchio";
- Filter results by parameters such as *gender*, *language*, *originCountry*, *certificate* and *type*. For instance, if we filter values by "Yoruba", only two shows will appear in that language ("The Bling Lagosians" and "Untitled Sefi Atta Project");
- Filter results by a range of years, *rating*, *runtime*, number of *episodes* and *numVotes*. For instance, if we filter the range values for years from 1933 to 1938, two shows will appear ("Kära släkten" and "Karriä");
- Sort results by numVotes, rating, runtime, popularRank and startYear. For instance, if we sort shows by popularRank the show "Lucifer" will be the first one to appear, followed by "Army of the Dead", "The Kominsky Method", and so on.
- Apply multiple filters simultaneously. For instance, it will be possible to apply a range filter such as filtering the *numVotes* from 1000 to 2000, *originCountry* as "United States", and use full-text search for the word "Payneville". The result will be the show "Along Came Jones".

3 Milestone 2 - Information Retrieval

The goals of the Information Retrieval milestone are to implement and use an information retrieval tool on the project dataset and its exploration with free-text queries. This task makes use of state-of-the-art retrieval tools and involves the view of the datasets as collections of documents, the identification of a document model for indexing and the design of queries to be executed on the indexed information.

Firstly, we have to choose an information retrieval tool and, after making some searches, we chose Solr [see ref. 5] because it's a fast, powerful, flexible and simple search engine with a good interface. To better understand and use this tool, we used the FAQ website provided by the developer, which proved to be very useful [see ref. 4].

After analysing the documents and identifying their indexable components, we start building the indexes and configuring and executing the queries in order to achieve an information retrieval system that returns great results.

Upon having a well-defined schema, we have to manually evaluate the returned results and evaluate the results obtained for the defined information needs, so we can verify if we need to improve our schema.

3.1 Data Collection and Indexing Process

In this step, we start preparing the materials to use in further steps to construct our retrieval system. That includes:

- Documents creation and analysis
- Indexing process
- Schema and Indexed fields

3.1.1 Documents Creation and Analysis

From the previous milestone, we have a dataset that contains all shows data in *CSV* format, where each row corresponds to a different show and each column matches a different attribute: *imdbID*, *title*, *popularRank*, *certificate*, *startYear*, *endYear*, *episodes*, *runtime*, *type*, *orignCountry*, *title*, *language*, *summary*, *rating*, *numVotes*, *genres*, *cast* and *imageURL*.

Based on this structure, we only need to create one *core*, composed by all the shows, where each document corresponds to a *show* (row) in the dataset. All documents have a similar structure, as all of them have the same fields.

Listing 1. Example of a document

```
{
    "imdbID":"tt2092588",
    "title":"Frozen Planet",
    "popularRank":2165,
    "certificate":"{'Brazil': 'Livre', '
       Germany': '0', 'Netherlands': '
       AL', 'Singapore': 'PG', 'United
       States': 'TV-PG'}",
    "startYear":2011,
    "endYear":2012,
    "episodes":10,
    "runtime":333,
    "type":"miniSeries",
    "originCountry":"United Kingdom",
    "language":"English",
    "summary":"Focuses on life and the
       environment in both the Arctic
       and Antarctic.",
    "rating":9.0,
    "numVotes":27484,
    "genres":"['Documentary']",
    "cast":"['David Attenborough', 'Alec
         Baldwin', 'Chadden Hunter', '
       Ted Giffords', 'John Aitchison',
        'Doug Allan', 'Doug Anderson',
       'Miles Barton', 'Vanessa
       Berlowitz', 'Lewis Brower', '
       John Durban', 'Faye Hicks', '
       Alun Hubbard', 'Michael Kelem',
       'Mark Linfield', 'Red McBrian',
       'Jamie McPherson', 'Hugh Miller'
       ]",
    "id":"2b511d95-d3ce-4c2f-a857-c7baf7
       9313c5",
    "version":1718968596823015424,
    "imageURL":"https://m.media-amazon.
       com/images/M/MV5BOGM5YWU2N2
       QtYjVhZi00MzYyLTk0
       ODctYmV1NDZlMjU5N2Q5
       XkEyXkFqcGdeQXVyMzU3MTc5OTE@._V1
       _FMjpg_UX1000_.jpg%22%7D
}
```

Since our dataset is condensed into a single file and has consistent data, we don't need to split it into different fields and create more than an index.

3.1.2 Indexing process

To index and parse all the data, first, we converted our *CSV* file that contains the dataset into a *JSON* file to be easier to import it to *Solr*. To run *Solr* in our machines, we have a *Docker* image, based on Solr's, that performs all preparation operations when booting, so that we don't have to perform them manually every time you start working. We have a *Dockerfile* that defines a custom image in this manner and performs all the basic setup operations, like copying the original dataset and other sub-datasets that we have created to evaluate our results. This file also calls a script where we create our core *shows*, define our *schema* and populate our database.

3.1.3 Schema and Indexed Fields

Solr stores details about the field types and fields it is expected to understand in a schema file. To do that, we have created *schema.json* file.

In *schema*, we create some *field-types*: *id*, *title*, *certificate*, *type_plural*, *lower_clean*, *summary* and *cast*.

In *id* field-type, we decided to use the same *tokenizer* and *filters* to *indexAnalyser* and *queryAnalyser*. For *tokenizer*, we used *solr.StandardTokenizerFactory*. For *filters*, we use:

- *solr.LowerCaseFilterFactory* because this converts any uppercase letters in a token to the equivalent lowercase token. That allows, for example, that inputs 'tt4052886' and 'TT4052886' match the same document. Otherwise, only 'tt4052886' matches a document, as there is no document with id 'TT4052886';
- *solr.PatternReplaceCharFilterFactory*, so we can remove all the letters that the input or the indexed id contains and compare only the numbers. As an example, if we have as input 'ttb4f05D2a88X6', this will be transformed into '4052886' and will match the document with id 'tt4052886'.

Looking for *title* field-type, we decided to use the same tokenizer for *indexAnalyser* and for *queryAnalyser* - *solr.StandardTokenizerFactory* - because we want to split the text into its different tokens treating whitespaces and punctuation as delimiters. In relation to *filters*, we use for both *indexAnalyser* and *queryAnalyser*:

- *solr.ASCIIFoldingFilterFactory* because this filter converts alphabetic, numeric, and symbolic Unicode characters which are not in the Basic Latin Unicode block (the first 127 ASCII characters) to their ASCII equivalents, if one exists, so words like 'Lúcifer' and 'Lucifer' return the same output;
- *solr.EnglishMinimalStemFilterFactory* because this filter stems plural English words to their singular form, so inputs like 'Friends' and 'Friend' return the document 'Friends';
- *solr.LowerCaseFilterFactory* for the same reasons as in *id* field-type;
- *solr.KStemFilterFactory* because that allows us to reduce inflected (or sometimes derived) words to their word stem, base or root form. As an example, if we search for the word 'walking', then should return documents that contain both 'walk' or 'walking' in the title and searching for 'walk' should also return documents with 'walk' or 'walking' in the title.

In *indexAnalyser* filters, we also use *solr.EdgeNGramFilterFactory* with 'minGramSize' = 4 and 'maxGramSize' = 10, because this allows us to generate edge n-gram tokens with sizes between 4 and 10, so we can obtain titles that contain substrings of the searched word. For example, if we search for the word 'English', we get results containing 'English' and 'Englishman' in the title.

In case of *certificate* filter-type, we use *solr.StandardTokenizerFactory* as *tokenizer* for both analysers for the same reasons as above. For filters, we also use the same for both: *solr.ASCIIFoldingFilterFactory* and *solr.LowerCaseFilterFactory* for the reasons that we have already explained.

Then, we have *type_plural* filter-type that uses *solr.WhitespaceTokenizerFactory* tokenizer for both analysers because we are interested in only splitting the text stream on whitespaces and maintaining the words that contain '-' together. Looking at filters for both we use:

- *solr.PatternReplaceCharFilterFactory* to remove all '-' introduced when searching for a particular *type*. This allows to present results for 'miniSeries' when the user search by 'mini-series';
- *solr.EdgeNGramFilterFactory* with 'minGramSize' = 2 and 'maxGramSize' = 5, because this allows us to get edge n-gram tokens with size between 2 and 5, in order to allow a greater margin of error to the user if he does not know the specific names of the types when searching. For example, if the user searches for 'serie' *type*, he will get documents with *type* 'series';
- *solr.ASCIIFoldingFilterFactory* and *solr.LowerCaseFilterFactory* for the reasons already explained.

For filter-type *lower_clean*, we use for both analysers the tokenizer *solr.StandardTokenizerFactory* to split the text field into tokens, treating whitespace and punctuation as delimiters. Then we apply two filters that we have already used in other filter-types: *solr.ASCIIFoldingFilterFactory* and *solr.LowerCaseFilterFactory*.

Continuing looking at filter-types, we have *summary* that uses the same features for *indexAnalyser* and *queryAnalyser*. In tokenizer, we use *solr.StandardTokenizerFactory* for the same reasons explained above. For filters, we have:

- *solr.StopFilterFactory* in order to stop analysis of tokens that are on the given stop words list. Through this, if a user search for 'story man leverages single', the result returned is a document that contains 'The story of a man who leverages his single' in *type* field;
- *solr.EdgeNGramFilterFactory*, *solr.ASCIIFoldingFilterFactory* and *solr.LowerCaseFilterFactory* for the reasons already explained.

Finally, we have *cast* field-type, using *solr.ClassicTokenizerFactory* tokenizer for both analysers and for filters:

- *solr.ClassicFilterFactory* that takes the output of the classic tokenizer and strips periods from acronyms and "'s" from possessives. As it's applied to query and index, if a user searches by 'DB Woodside', the system will return the documents that contain 'D.B. Woodside'.
- *solr.EdgeNGramFilterFactory*, *solr.ASCIIFoldingFilterFactory* and *solr.LowerCaseFilterFactory*, as we explianed for other field-types.

Now, let's look at the final type for each attribute:

- *name*: imdbID, *type*: id, *indexed*: true, *stored*: true;
- *name*: title, *type*: title, *indexed*: true, *stored*: true;
- *name*: popularRank, *type*: pint, *indexed*: true, *stored*: true;
- *name*: certificate, *type*: certificate, *indexed*: true, *stored*: true;
- *name*: startYear, *type*: pint, *indexed*: true, *stored*: true;
- *name*: endYear, *type*: pint, *indexed*: true, *stored*: true;
- *name*: episodes, *type*: pint, *indexed*: true, *stored*: true;
- *name*: runtime, *type*: pint, *indexed*: true, *stored*: true;
- *name*: type, *type*: type_plural, *indexed*: true, *stored*: true;
- *name*: originCountry, *type*: lower_clean, *indexed*: true, *stored*: true;
- *name*: language, *type*: lower_clean, *indexed*: true, *stored*: true;
- *name*: summary, *type*: summary, *indexed*: true, *stored*: true;
- *name*: rating, *type*: pfloat, *indexed*: true, *stored*: true;
- *name*: numVotes, *type*: pint, *indexed*: true, *stored*: true;
- *name*: genres, *type*: lower_clean, *indexed*: true, *stored*: true;
- *name*: cast, *type*: cast, *indexed*: true, *stored*: true;
- *name*: imageURL, *type*: string, *indexed*: true, *stored*: true;

3.2 Information Retrieval

For the information retrieval process, we created four queries so that it was possible to get specific show information from the dataset based on the filters applied. We developed a subset of shows for each one with some values that satisfy some fields and do not satisfy others, mixed with random shows that have nothing to do with our search and others that match it perfectly. The last step was essential to make our results more efficient and quickly check if correct.

3.2.1 Query 1: English-related shows

For the first query, the goal was to find all the shows with the word 'English', either because it appears in the summary or the original language is English. Finally, we needed to assign the results weights such that, for instance, if the language of 2 shows were both English, the one who also had the word in the summary would appear first. It was, then, required to change the defType to edismax and the field qf to language^2 summary. This way, the language will be the field with the most weight since if someone tries to search for English, it is more relevant to show first the shows in English. As a result, since there will be a tie, the field summary will not affect the weight where the language is English. Thus, the field tie must be filled with a value multiplied by the summary's weight and added to the language's one. If the field tie has the value 1, the results will appear as predicted. Meaning, all English shows with English in the summary field appearing first, followed by those with only the language English, and finally those with only the word English in the summary field.

Querie url: http://localhost:8983/solr/shows/query?q=english&q.op=AND&defType=edismax&indent=true&debug

Query=false&qf=language%5E2%20summary&rows=20&tie=1

3.2.2 Query 2: Shows containing occurrence of verb Do

For the second query, the objective was to find all results where the origin country was the 'United States' and the range of years it was released was between 2017 and 2021. We also wanted to include some title text searching. So, the results have to include the word 'Do' and its conjugated forms, such as 'doing', 'done', 'did', etc. In this case, we also needed to assign the results weights. So it was required to change the defType to edismax and the field tie to 1.

Querie url: http://localhost:8983/solr/shows/select?defType=edismax&indent=true&q.op=AND&q=title%3ADoing%20originCountry%3A%22United%20States%22%20startYear%3A%5B2017%20TO%202021%5D&tie=1

3.2.3 Query 3: High rating with Russian or Portuguese certificate

For the third query, the goal was to find all documents where the rating is higher than seven. Besides this, the show must have been released after 2010 and ended until 2020. Furthermore, we also wanted to get all the previous results but were only interested in the ones having the values for the certificate in Russian or Portuguese. We based this choice on the fact that our search should handle words written with accents like 'Rússia'. In this case, we also needed to assign the results weights. So it was required to change the defType to edismax and the field tie to 1. The order of the results relies on the number of votes to get the most viewed and popular shows first.

Querie url: http://localhost:8983/solr/shows/select?defType=edismax&indent=true&q.op=AND&q=rating%3A%5B7%20TO%20*%5D%20startYear%3A%5B2010%20TO%20*%5D%20endYear%3A%5B*%20TO%202020%5D%20(certificate%3AR%C3%BAssia%20OR%20certificate%3APortugal)&rows=40&sort=numVotes%20DESC&tie=1

3.2.4 Query 4: Action Series Actors

For the fourth query, the purpose was to find all results with cast members by the name of 'D.B.', 'Lesley-Ann', 'J.K.' or 'Matt'. Furthermore, it also contemplated the restriction of the 'Action' genre. This choice came to mind so we could guarantee the search would still work if we did not put the punctuation dividing the names in the search query. We also ordered the results by their number of episodes, where movies should not be relevant. In this case, we also needed to assign the results weights. So it was required to change the defType to edismax and the field tie to 1.

Querie url: http://localhost:8983/solr/shows/select?indent=true&q.op=OR&q=(cast%3A%22DB%22%20cast%3A%22Lesley%20Ann%22%20cast%3A%22jk%22%20cast%3AMatt)%20AND%20genres%3A%22Action%22&rows=30&sort=episodes%20ASC

3.3 Evaluation

The next step is to evaluate the relevance of the results retrieved for each of the queries described in the previous subsection. The use of distinct evaluation metrics clarifies the discrepancy between two different situations: a schema-less system and one with an enhanced schema. The used metrics are the following:

- **Average Precision:** mean of the precision scores after each relevant document is retrieved
- **Precision at N (P@N):** proportion of retrieved documents that are relevant
- **Recall at N (R@N):** proportion of relevant documents retrieved
- **F1 at N (F1@N):** harmonic mean of the precision and recall

The Precision-Recall Curve, which shows the trade-off between precision and recall for different thresholds, is the same for every query, where the precision is constant independently of the recall due to an Average Precision of 100%, with an exception of the fourth query. The graph is displayed in the Figure 10.

Each query is executed upon a subset to control the existing documents and easily explain the evaluation process.

3.3.1 Query 1: English-related shows

The subset used for this query included 20 shows: five had the word 'English' on the summary, another five in the language, another five in both the language and summary and finally, the last five had neither the word in the summary field or language. Therefore, there are 15 relevant documents since we desire to know every English-related show. There is one particular document where only the word 'Englishman' appears. That document will only be retrieved in the enhanced schema version due to the filters applied. Consequently, the precision, recall and f1 score of the schema-less version are lower.

- Schema-less:

	Metric	Value
1	Average Precision	1.0
2	Precision at 15 (P@15)	0.933333
3	Recall at 15 (R@15)	0.933333
4	F1 at 15 (R@15)	0.933333

- Enhanced schema:

	Metric	Value
1	Average Precision	1.0
2	Precision at 15 (P@15)	1.0
3	Recall at 15 (R@15)	1.0
4	F1 at 15 (R@15)	1.0

3.3.2 Query 2: Shows containing occurrence of verb Do

The subset used for this query also included 20 shows: five have some conjugated form of the verb To Do in the title, and the other fifteen don't. Therefore, there are only five relevant documents since the user searches the word 'Doing' and might be interested in shows whose title may have a different conjugation of that verb. In the schema-less version, the system retrieves only one show, whose title has the exact conjugation ('Tom Papa: You're Doing Great!'). With the enhanced schema, the system retrieves four additional documents, for example, the show 'What Did Jack Do?'. The respective scores are:

- Schema-less:

	Metric	Value
1	Average Precision	1.0
2	Precision at 5 (P@5)	0.2
3	Recall at 5 (R@5)	0.2
4	F1 at 5 (R@5)	0.2

- Enhanced schema:

	Metric	Value
1	Average Precision	1.0
2	Precision at 5 (P@5)	1.0
3	Recall at 5 (R@5)	1.0
4	F1 at 5 (R@5)	1.0

3.3.3 Query 3: High rating with Russian or Portuguese certificate

The subset used for this query includes 100 shows, of which 40 are relevant according to the query specifications. The enhanced system retrieves all of them, but the schema-less version only results in seven. This is because shows with a certificate from 'Russia' are not retrieved since the query specifically requested the word 'Rússia', which is only detected in the enhanced situation with the filters applied. Therefore, the schema-less system only returns the shows with a Portugal certificate. The resulting scores are:

- Schema-less:

	Metric	Value
1	Average Precision	1.0
2	Precision at 40 (P@40)	0.175
3	Recall at 40 (R@40)	0.175
4	F1 at 40 (R@40)	0.175

- Enhanced schema:

	Metric	Value
1	Average Precision	1.0
2	Precision at 40 (P@40)	1.0
3	Recall at 40 (R@40)	1.0
4	F1 at 40 (R@40)	1.0

3.3.4 Query 4: Action Series Actors

The subset used for this query includes 30 shows, of which only five are relevant according to the actors the user is looking for and that it should be sub intended that the user is only looking for series because it limited the results by the number of episodes. The enhanced system retrieves movies since no limitation to the type of show was applied. So, in this case, neither of the versions will have an Average Precision of 100%, so the Precision-Recall curve shown in the Figure 11 is more interesting.

- Schema-less:

	Metric	Value
1	Average Precision	1.0
2	Precision at 10 (P@10)	0.2
3	Recall at 10 (R@10)	0.4
4	F1 at 10 (R@10)	0.266667

- Enhanced schema:

	Metric	Value
1	Average Precision	0.224162
2	Precision at 10 (P@10)	0.5
3	Recall at 10 (R@10)	1.0
4	F1 at 10 (R@10)	0.666667

4 Conclusion

In conclusion, the project successfully terminated since we fulfilled the required steps and achieved the proposed goals. Although, in other circumstances where the development time would not be so limited, exploring many more possible approaches and extra functionalities could increase the complexity and value of the project. In the following list it's presented the interpretation of each of the three milestones:

- In the Data Preparation step, despite the accomplishment of the requirements, the resulting data could be plentiful and more valuable by including new interesting columns, such as the directors of the shows and, in a more complex process, information about the episodes of a series, for example.
- In the Information Retrieval step, the group succeeded in the indexation process and the development of queries to test and prove the strength of our schema. Even though meeting the requirements, current queries could be even more difficult by adding constraints and filters, such as the case of result sorting.

References

[1] Davide Alberani. *IMDbPY Python Library*. 2018. URL: https://imdbpy.readthedocs.io/en/latest/.

[2] Amazon. *IMDb Website*. 1990. URL: https://www.imdb.com/.

[3] Snehaan Bhawal. *Initial Kaggle dataset*. 2021. URL: https://www.kaggle.com/snehaanbhawal/netflix-tv-shows-and-movie-list.

[4] Atlassian Confluence. *SolrRelevancyFAQ*. URL: https://cwiki.apache.org/confluence/display/solr/SolrRelevancyFAQ.

[5] Apache Software Foundation. *Solr*. URL: https://solr.apache.org/.

[6] Freebase. *OpenRefine Documentation*. 2010. URL: https://docs.openrefine.org/.

[7] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[8] Netflix Inc. *Netflix Website*. 1997. URL: https://www.netflix.com/.

[9] Plotly Technologies Inc. *Collaborative data science*. 2015. URL: https://plot.ly.

[10] Jeroen Janssens. *Data Science at the Command Line - Chapter 6*. 2014. URL: https://www.datascienceatthecommandline.com/2e/chapter-6-project-management-with-make.html.

[11] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[12] Michael L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

5 Attachments

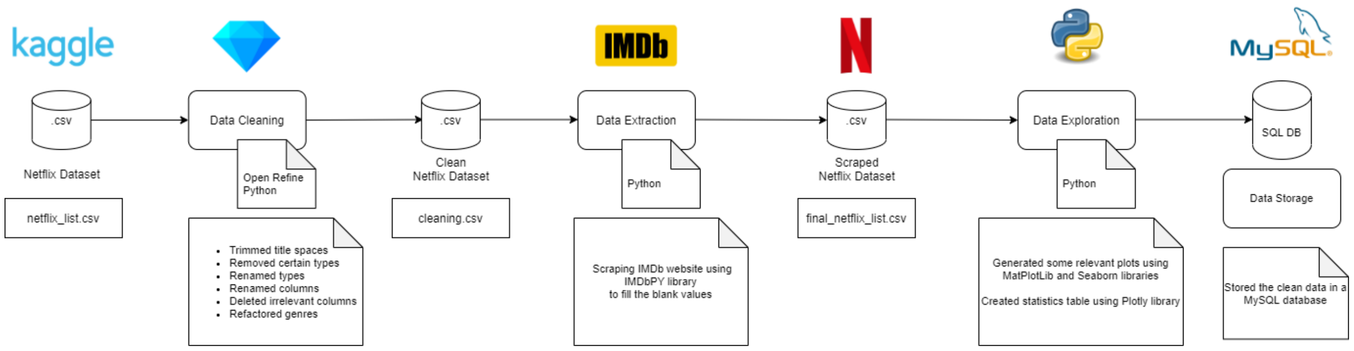


Figure 1. Data processing pipeline

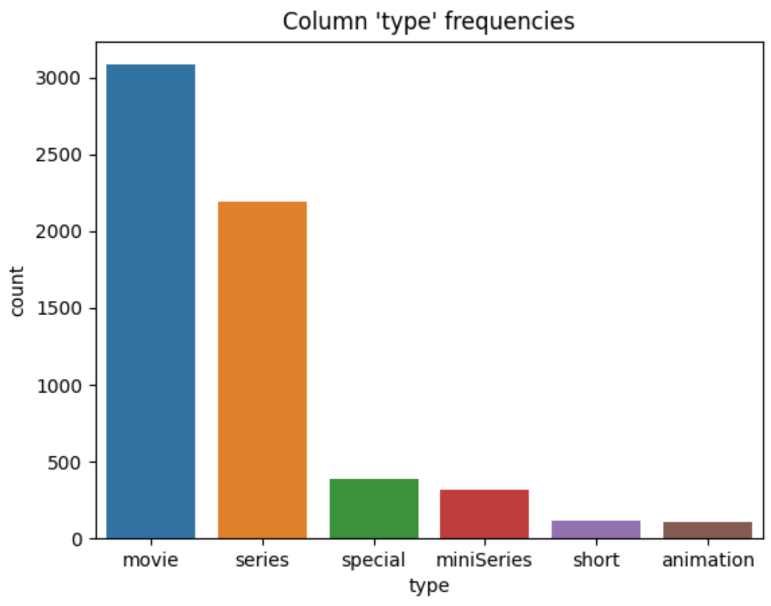


Figure 2. 'Type' Bar Chart

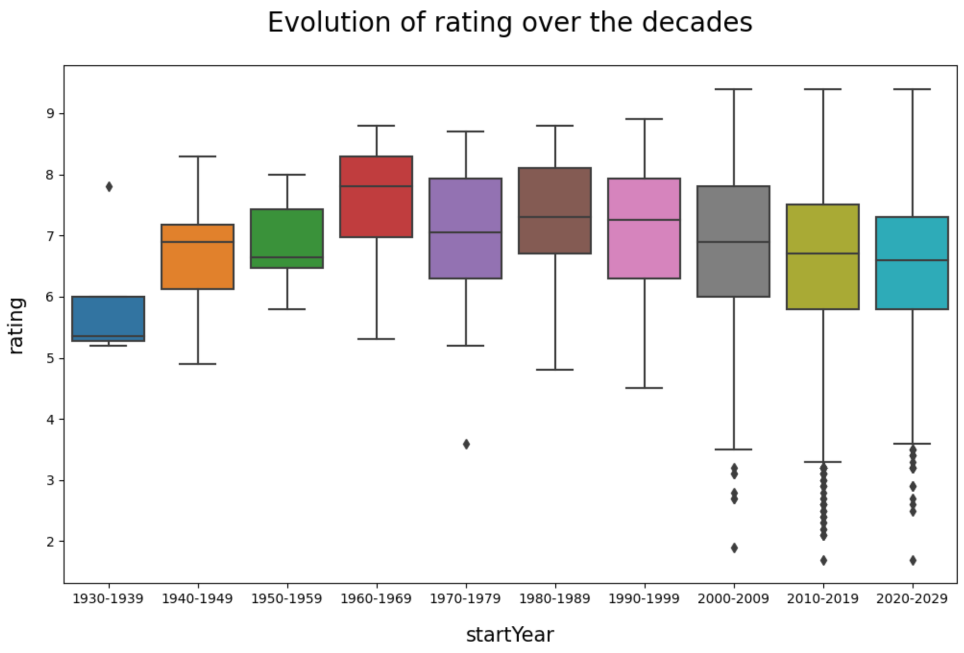


Figure 3. 'Rating' Box Plot

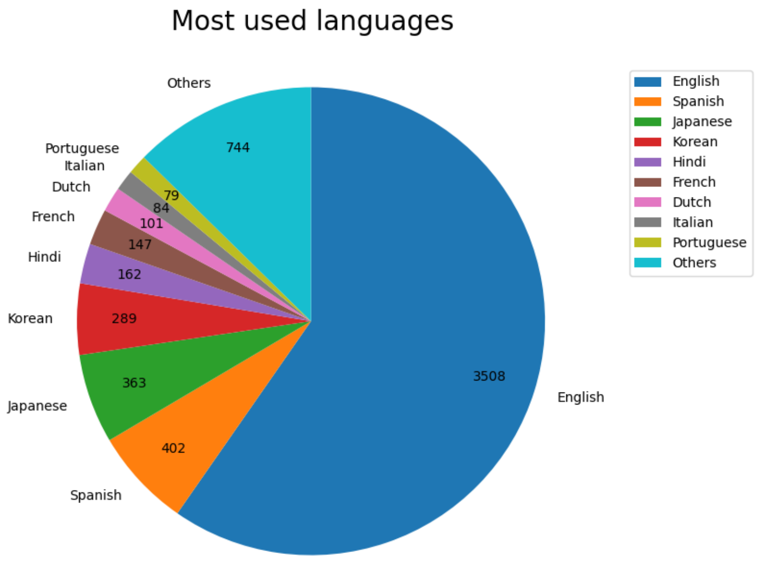


Figure 4. 'Language' Distribution Pie Chart

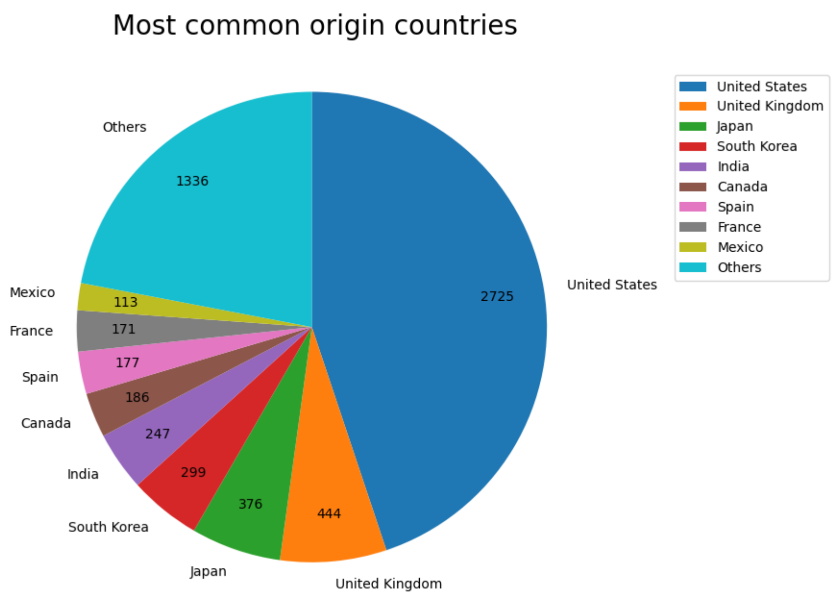


Figure 5. 'Countries' Distribution Pie Chart

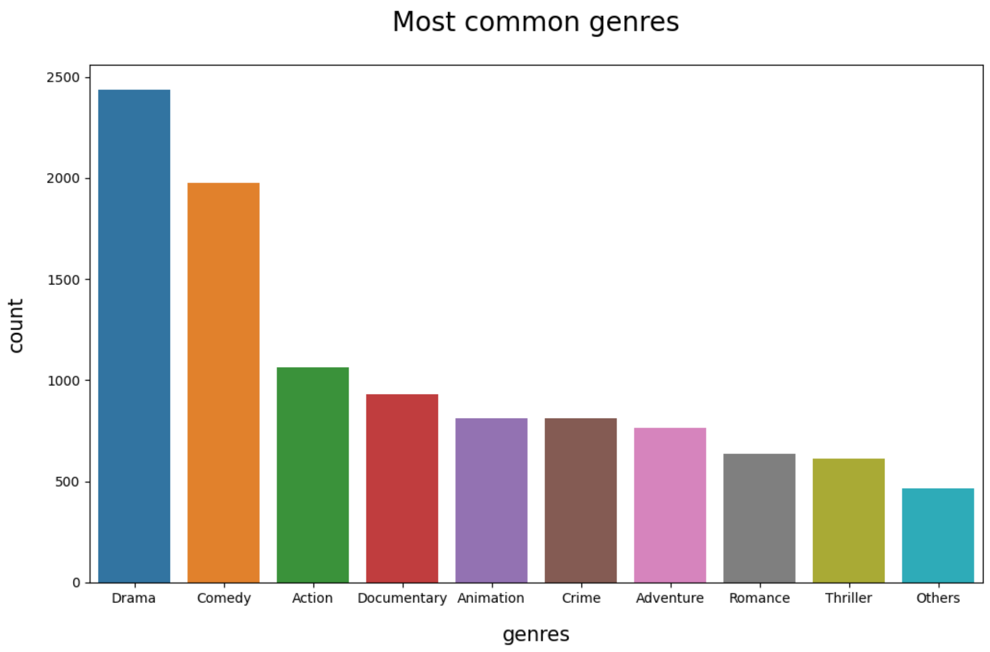


Figure 6. 'Genre' Bar Chart

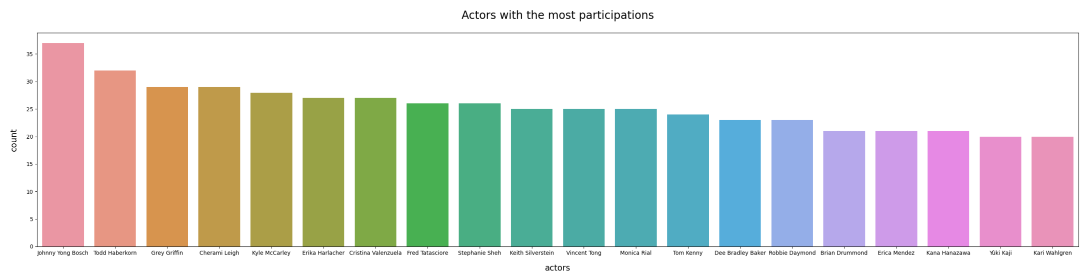
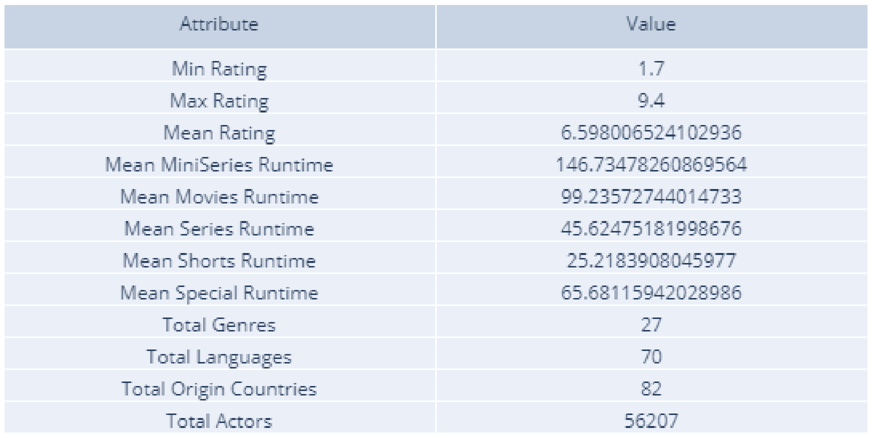


Figure 7. 'Cast' Bar Chart



Attribute	Value
Min Rating	1.7
Max Rating	9.4
Mean Rating	6.598006524102936
Mean MiniSeries Runtime	146.73478260869564
Mean Movies Runtime	99.23572744014733
Mean Series Runtime	45.62475181998676
Mean Shorts Runtime	25.2183908045977
Mean Special Runtime	65.68115942028986
Total Genres	27
Total Languages	70
Total Origin Countries	82
Total Actors	56207

Figure 8. Basic statistics of the dataset

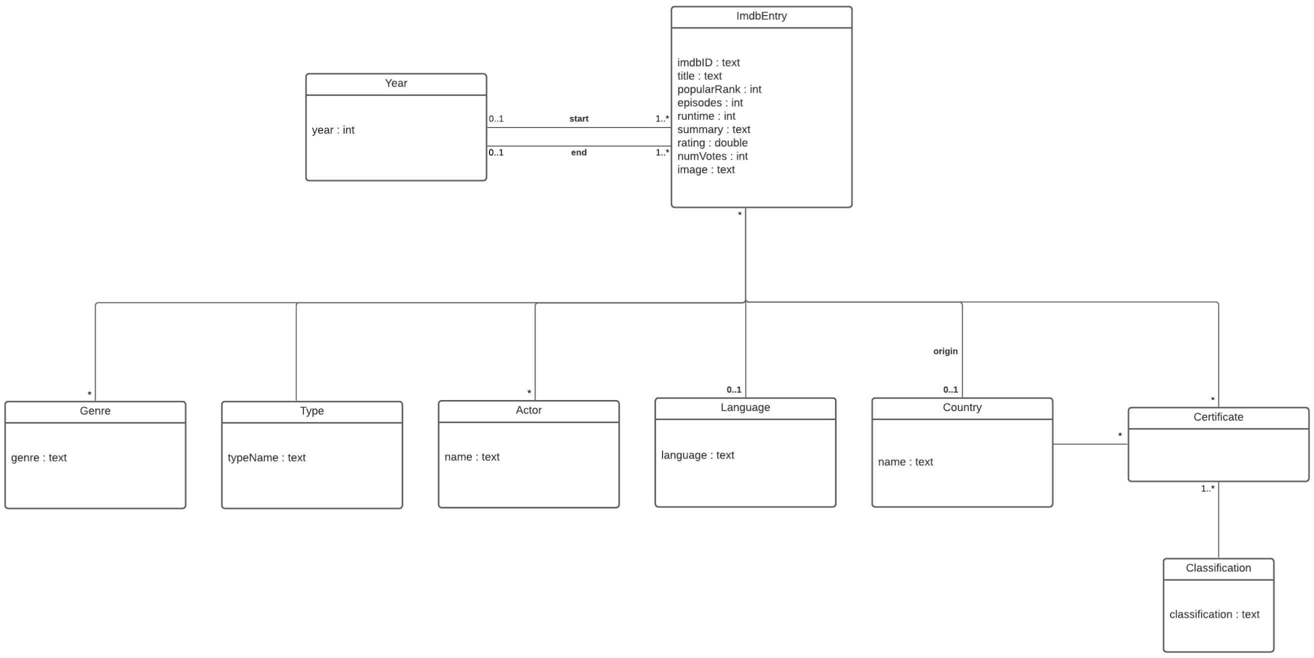


Figure 9. UML diagram for database creation

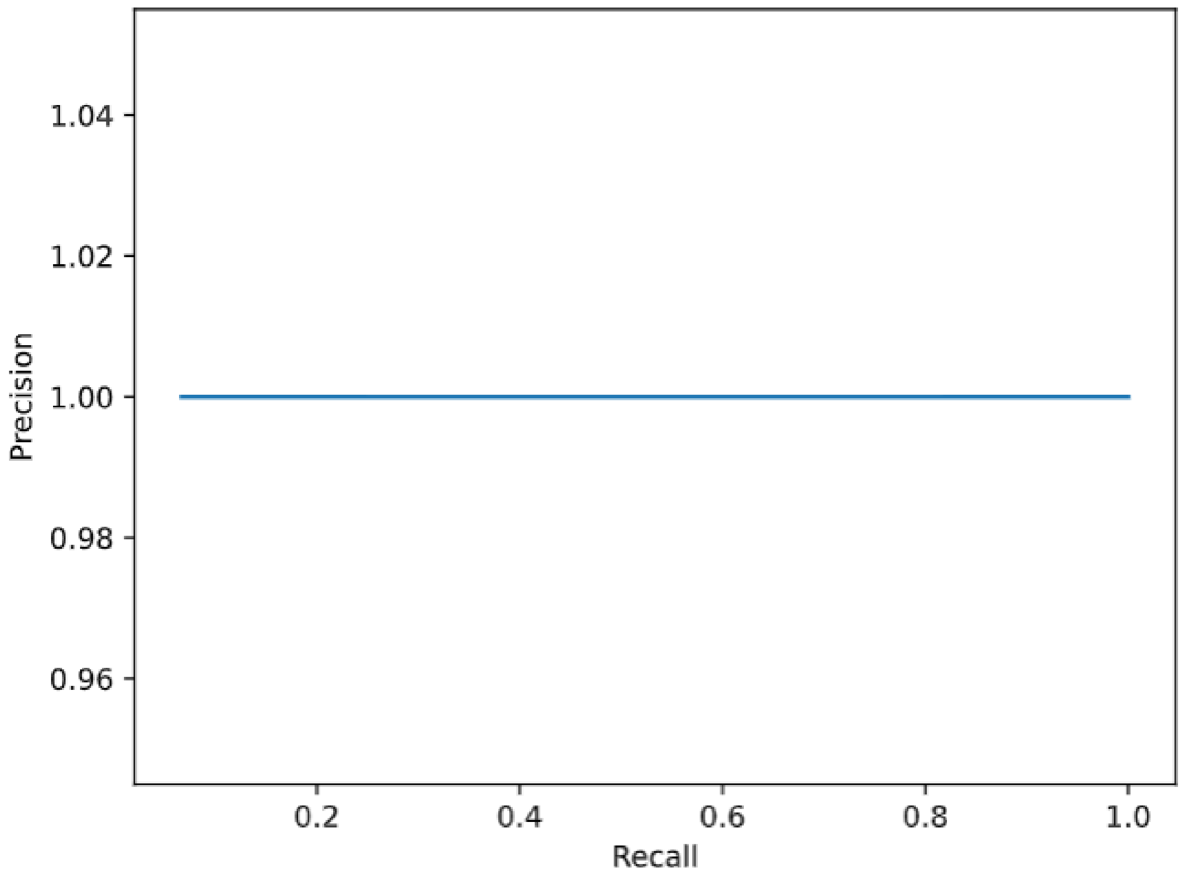


Figure 10. P-R curve with constant precision

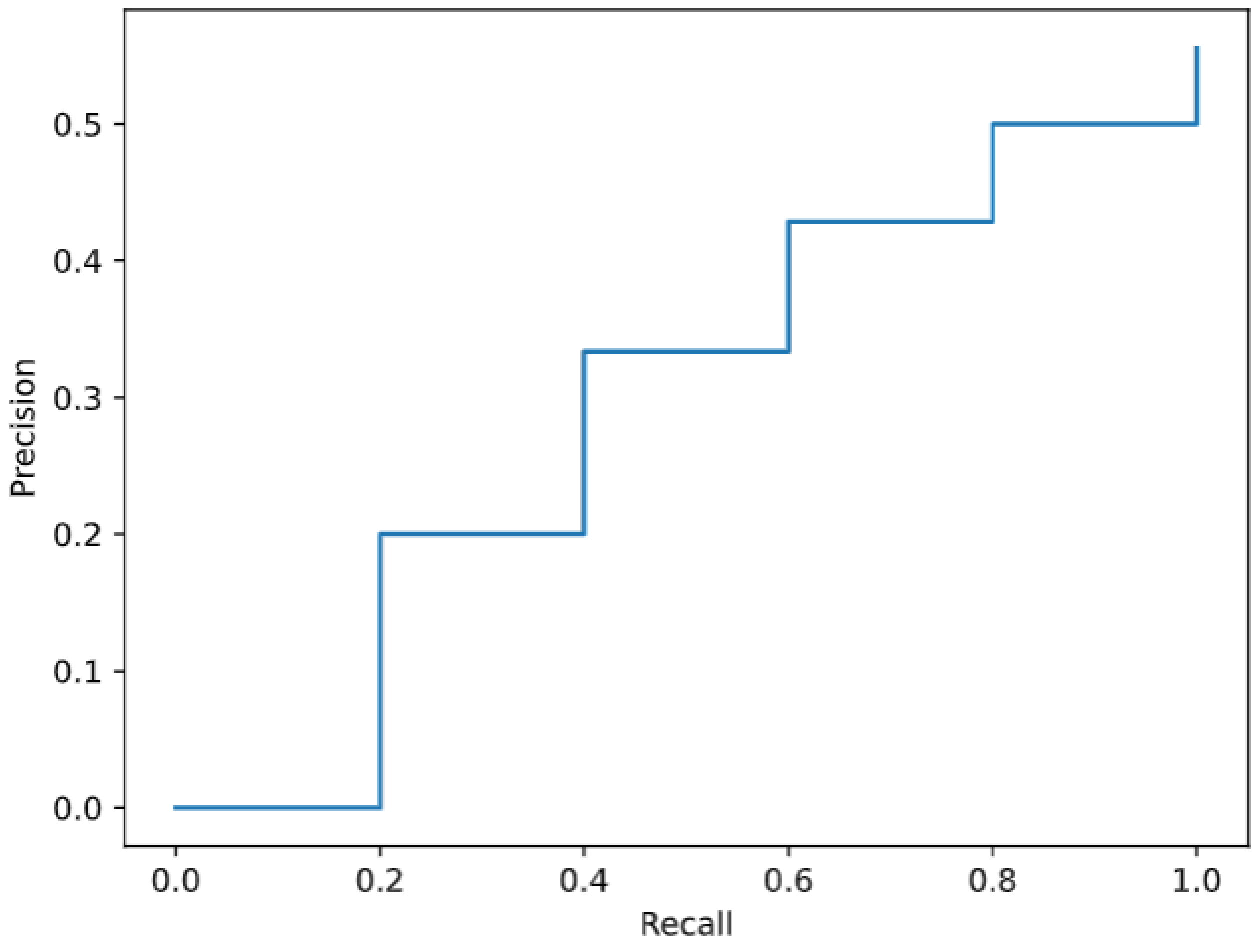


Figure 11. P-R curve of query 4