IMDB/Rotten Tomatoes Movies Ratings and Reviews

Caio Nogueira
up201806218@edu.fe.up.pt
University of Porto

Carlos Lousada
up201806302@edu.fe.up.pt
University of Porto

Tomás Mendes
up201806522@edu.fe.up.pt
University of Porto

ABSTRACT

With the constant growth of information available online, it has become extremely important to come up with information systems able to process, index, and search that information effectively. In this article, we study the case of movies and respective reviews with information available in the 2 most popular movie-related websites on the whole Internet: Rotten Tomatoes and IMDB. The datasets used are public and available on Kaggle. The data was then analyzed and prepared for the retrieval phase. To implement the Information Retrieval system, we used the Solr platform. After indexing the documents and defining our collection, we chose a set of information needs that would allow us to explore the engine's functionalities and evaluate the search system - what fields are most important to improve the search results. Furthermore, to achieve better performance in the Information Retrieval phase, we implemented improvements to both syntactical and semantical analysis of the documents. Moreover, some machine learning methods were also used to increase the scope of the search. In a final stage, we needed to analyze the impact of the improvements implemented in the system, having a system without those same improvements as a guideline.

CCS CONCEPTS

• **Information systems → Evaluation of retrieval results**.

KEYWORDS

Datasets, Data Preparation, Movies, Search Systems, Information Retrieval

1 INTRODUCTION

Cinema was not invented by one person. The first to present projected moving pictures were the Lumière brothers in December 1895 in Paris. They decided to use a device made by themselves, the Cinématographe, which was composed of a camera, a projector and a film printer. Since then, movies evolved tremendously: from basic frames in black and white colours with no sound to an amazing explosion of colours with beautiful soundtracks and dialogues, passing through a golden age (during the 1930s and 1940s) and a rapid improvement of the digital technology (3D, IMAX, CGI, ATMOS, Real|Laser, among other tools).

Bearing this in mind, this article describes the development phases of a movie platform that aims to break any geographical barrier and intends to connect fans of the seventh art from all over the world, using a friendly and intuitive search engine with several applicable filters.

Firstly, we collect datasets with relevant information (*Data Collection*). Then, the data should go through a preparation stage (*Data Preparation*), i.e., a variety of refinement tasks, such as data scaling, normalising and cleaning, to be easier to handle. Furthermore, the *Conceptual Model* demonstrates how the dataset is structured and by using and interpreting diverse graphs (*Data Characterization*), we can have a better perception of the collected data.

The second phase is focused on *Information Retrieval*. Initially, we compared the most relevant search engines - *Solr* and *Elasticsearch*. This process is described as *Information Retrieval Tool Selection*. Then, the collections and documents are specified in *Collections and Documents*, describing our different approaches to solve our problem of joining movies and their respective reviews. In the following sections, the field types were applied to the respective relevant fields (*Indexing Process*) and, finally, to evaluate the system's performance, five different information needs were defined (*Retrieval Process*) and their results were discussed based on the configurations set.

Lastly, the third and final stage is mainly centered in improvements of our search system, passing through several topics, such as *OpenNLP* integration (divided in two steps), *Learning to Rank* and the usage of synonyms. Moreover, to verify if our recent experiments had a great impact, we compared the results of Schema with weights with the new System (*Overall improvement results*).

2 DATA COLLECTION

The data used for the information retrieval system was collected from *Kaggle* [27], which is a well-known data science website that allows users to publish and use free datasets in a collaborative environment.

In order to collect the necessary data, we downloaded two different datasets from *Kaggle*: one of them containing information scrapped from the IMDB's database[7] and the other from the Rotten Tomatoes's database[8].

The IMDB dataset contains information about over 85000 movies, dated between 1915 and 2020 (when both datasets were scrapped). The data is composed of 4 different *.csv* files, from which we selected the ratings and the movies' general information: title, actors, directors, release date, age suitability. By doing this, we can retrieve information about the ratings given to the movies.

The Rotten Tomatoes dataset is less extensive when it comes to the number of movies: it contains information about roughly 17,000 movies. Despite covering fewer movies, we can use this dataset to collect textual critics for each movie, which will be useful. In addition, we also collected numeric classification for each movie, such as the *Tomatometer* score and *Audience Score* (rating given by the users of Rotten Tomatoes). For these reasons, we can say that the datasets complement each other.

Both datasets are owned by the same *Kaggle* user: Stefano Leone (a data Analyst at a financial company). The fact that the data was collected by the same developer using the same methods (Python with the Requests [16] library) proved to be a huge advantage as the datasets have a very high usability score and the data types are consistent.

3 DATA PREPARATION

After having decided which datasets would be used for the information retrieval system, we needed to refine the data. This section consists of the different steps taken to prepare the data for the information retrieval phase.

For this task, we used Pandas [15] library from Python, which allows reading, writing and manipulation of *csv* files.

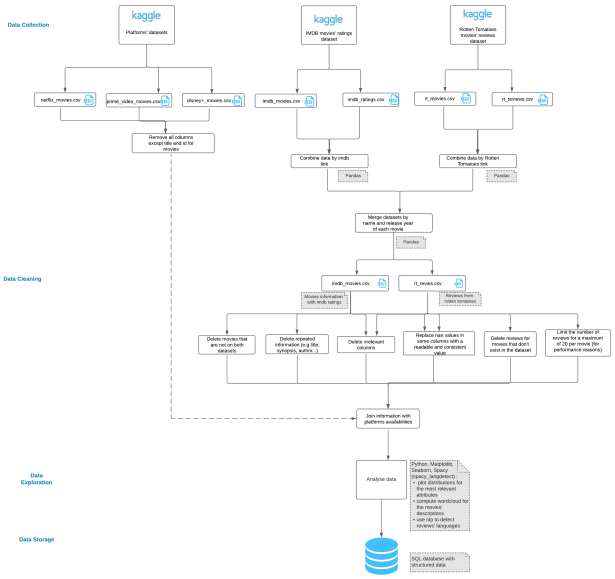

Figure 1: Pipeline design

3.1 Merging

First of all, since we selected two independent datasets we would need to merge them into one. For this reason, we could only use rows about movies that existed in both datasets. However, before merging, we dropped the common attributes (movie title, director, actors, genre, duration).

The next step in this process was the proper merging of the datasets. We decided to join the files by title and release date.

The merging process resulted in a *csv* file containing approximately 9,000 movies and over 80 columns. In the reviews file, we dropped the rows that didn't refer to movies that existed in the merged dataset.

3.2 Refinement

After having merged the data, we started looking into the different columns. Since some of the columns had NaN (Not a Number) values, we decided to replace them with a more readable text ("Unknown" or "Not defined") depending on the respective attribute. At this point, we had over 80 columns, which translates into an excessive amount of data (per movie) that would not be used further on, so we disposed of it: some columns had specific data that we were not planning to keep for the information retrieval phase (e.g., average score among females between 18 and 30 years old), or even redundant data: for example, the attribute *certified fresh* is true for movies whose *Tomatometer score* is above 85

The critics dataset, on the other hand, contained a reasonable amount of columns (8), but an excessive amount of reviews per movie (about 500,000 reviews in total). For performance reasons, we limited the maximum amount of critics for each movie to 20. In order to get the most relevant ones, we sorted the reviews by top critic and text length. This means that if a movie has, for instance, 50 reviews, where **1)** 15 of them are top critics and the rest of them are not, we will pick the 15 top critics and the 5 normal critics with higher word count; **2)** all of them are top critics, we will pick the 20 top critics with higher word count. After this operation, the reviews file was slightly less than 13,0000 rows.

3.3 Enrichment

In order to provide more information and, therefore more possibilities to query the dataset on future milestones, we used another dataset from *Kaggle*, with information about movies from three popular streaming services: *Netflix* [9], *Amazon Prime Video* [11] and *Disney Plus* [12]. The analysis of this new information resulted in three more columns added to the previous (movies) dataset. Each of the additional columns contains boolean values indicating whether the corresponding movie is available on the respective streaming service or not.

4 CONCEPTUAL MODEL

Having completed all the necessary data preparation tasks, we designed a domain conceptual model, which is displayed in Figure 2.

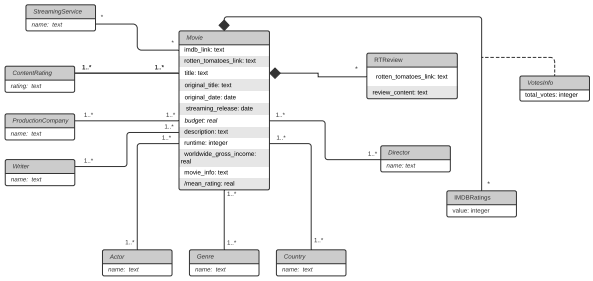

Figure 2: Conceptual Model

Movie is the main class since it connects all other entities in the domain. This class contains relevant information about each movie. The classes *ContentRating*, *ProductionCompany*, *Writer*, *Director*, *Actor*, *Genre* and *Country* are association classes that provide additional information about the movies.

Furthermore, the classes *RTReview* and *IMDBRatings* provide information about the Rotten Tomatoes critics and IMDB ratings,

respectively: *RTReviews* entries have as attributes not only the review itself but also some information about the critic who published it (name and reputation on the website). *IMDBRatings'* entries have the rating itself associated (1-10). This table will then be analyzed to compute the mean rating from *IMDb*.

5 SEARCH TASKS

The data collected and prepared can be obtained by the information retrieval system in several ways. Some of the possible retrieval tasks are listed below:

- Search movies about specific historical events
- Search movies that approach certain topics (e.g., social issues)
- Search movies based on real events
- Search movies that other users found emotive
- Search for criminal investigation movies
- Search reviews for a movie made by a verified Rotten Tomatoes user (top critic)

6 DATA CHARACTERIZATION

With the data prepared, we started to analyse its distribution and characterization. To complete this task, we used the following Python modules: Matplotlib [4], Seaborn [25] and WordCloud [13].

One of our concerns was to have a wide variety of data. This means that we wanted our final dataset to contain a decent number of movies (and respective reviews and ratings) from a relevant range of years. This proved to be the case as seen in figure 3.

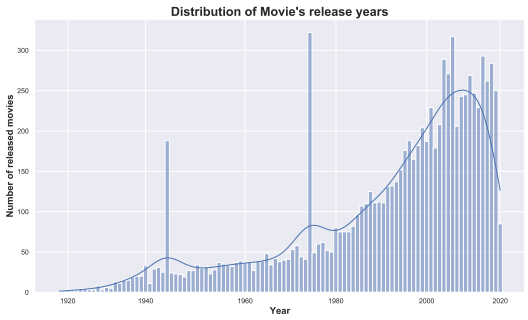

Figure 3: Number of movies per year

As expected, on average, our dataset contains more movies that were released in the 2000s. However, there's still a good representation of movies from the previous century.

Following this line of thought, we also strived for a wide representation of movie production countries, genres and content ratings. This can be observed in figures 9,10 and 11 (Attachments).

Analysing these graphs, we can see that there is a decent distribution of movies in all of the mentioned parameters. We found the results to be very accurate, as it shows Drama, Comedy and Action/Adventure as the most common genres and that R-Rated is the most popular content rating.

Furthermore, regarding rating data, we felt the urge to verify if there's a wide range of ratings and differently rated movies. To do that, we can analyse figures 7,8 and 9.

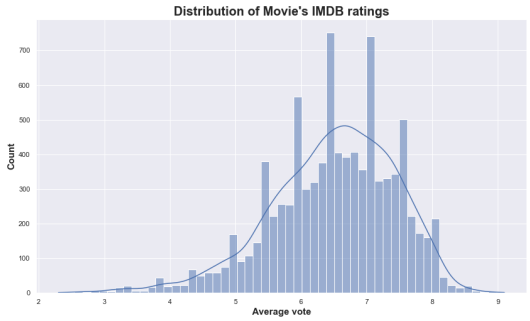

Figure 4: IMDB's rating distribution

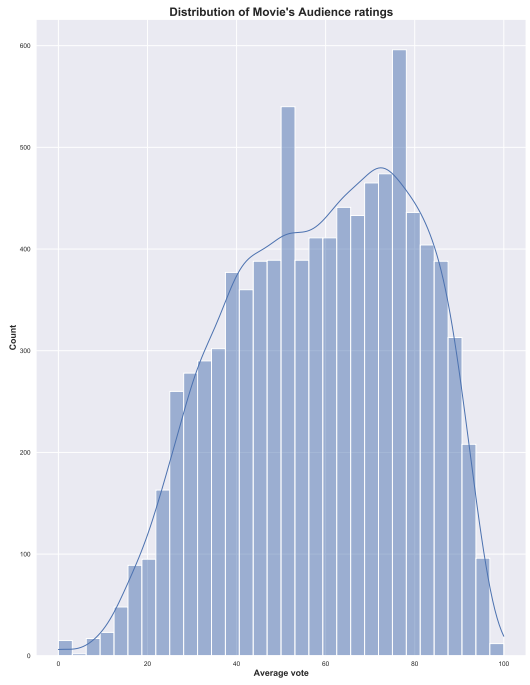

Figure 5: Rotten Tomatoes's Audience rating distribution

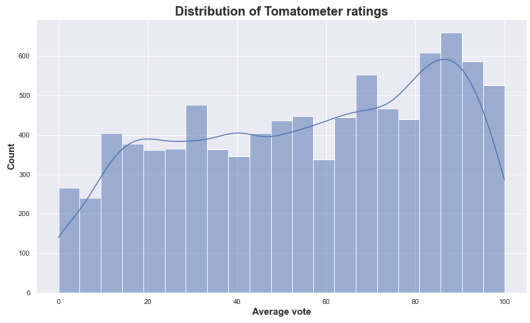

Figure 6: Tomatometer rating distribution

Once again, we can conclude that our dataset includes a wide variety of movies, rated in very different ways in several different ratings.

Finally, we decided to analyse a bar plot to check the evolution of ratings throughout the years, and a Word Cloud plot, to verify that there are several different plot points in the movies taken into account (for instance, family, love, friend, life, etc). Those plots can be seen in figures 10 and 11, respectively.

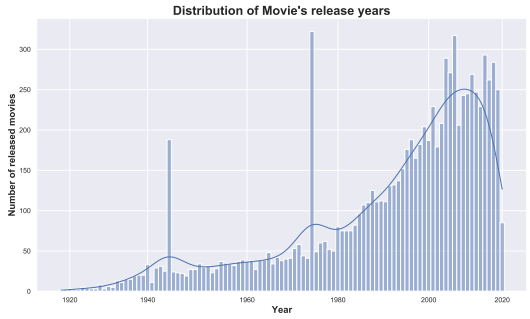

Figure 7: Average IMDB's ratings per year

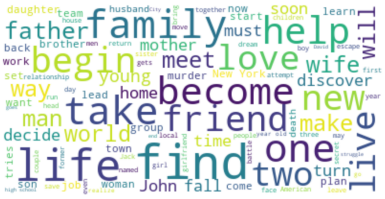

Figure 8: Movie description's Word Cloud

6.1 Text characterization

In our dataset, there are two main textual values for each movie: its description and the critics. Our goal at this point is to improve the understanding of both and, to achieve that objective, we computed the word cloud for the movies' descriptions, which is displayed on **Figure 11**. As to critics, we used NLP (Natural Language Processing) with *Spacy* [19] to detect the percentage of English reviews in the dataset. The NLP algorithm pointed out that approximately 99 percent of the reviews were English. The other reviews are written in Portuguese.

7 INFORMATION RETRIEVAL

The main goal of an Information Retrieval System is to retrieve documents from a collection. These documents obtained should be as relevant as possible to a given information need. This work focuses on *ad hoc* retrieval tasks, where the information need is obtained by a specific query.

In order to study and evaluate this process, we will be using 3 different systems:

- A baseline (*schemaless*) system, with basic matching on key fields;
- A system where the main text fields are indexed with the use of filters (*schema*);
- A system with indexed fields and which uses weights to choose the most relevant documents in the retrieval phase (*boosted*).

To evaluate the queries, we will use *Precision at 10* (P@10) [2] which calculates precision at a specific cut point (in this case, 10 documents), and *Average Precision* (AP) [1] that is defined as the average of the precision values obtained for the set of top N documents existing each time a new relevant document is retrieved. To compare the different systems, we will be using *Mean Average Precision* (MAP) [10].

8 INFORMATION RETRIEVAL TOOL SELECTION

In order to develop an Information Retrieval System, we could have opted for several platforms. However, there were 2 main options to choose: *Solr* [18] and *Elasticsearch* [3]. Both tools are similar built on *Apache Lucene* [23] and offer a wide variety of features: e.g., full-text search, support for No-SQL data and high availability. In this project, we used *Solr* as the search tool, due to the fact that *Solr* supports text search while Elasticsearch is mainly used for analytical querying, filtering, and grouping. For our specific use case, *Solr* is the most adequate tool.

9 COLLECTIONS AND DOCUMENTS

The most important documents for this dataset are the movies and their respective reviews.

In order to use multiple types of documents, we followed different approaches. Firstly, we tried to use nested documents: each movie would have as its children, the respective reviews. However, to query this collection we would need to use a complex syntax, that was hard to find on *Solr*'s documentation. In the second attempt, we tried to use different documents without nesting, but

we encountered problems when trying to use weights to improve search results.

Our final solution was to have the movies as a single type of document, where each movie has a list of its textual reviews.

10 INDEXING PROCESS

One of the most important steps involved in the development of an Information Retrieval System is the Indexing Process. Before this phase, we needed to decide which fields should be indexed. After analyzing the existing fields, we decided to index the ones which we believed to be useful when searching for a movie.

The indexed and stored fields with numeric values were defined with *Solr*'s default field type *pint* and the dates were defined with the *update* field type. The most important textual fields were subjected to an analyzer pipeline, including a *tokenizer* and optional filters to process each token (e.g., evaluate them in lower case and without accent marks).

With this in mind, two custom field types were created: *standart_text* and *daterange*.

- The *standart_text* field type is used in the main textual fields. This way, we applied three different filters: *LowerCaseFilterFactory* to match tokens given by a user query in a non-case-sensitive way; *ASCIIFoldingFilterFactory*, which converts alphabetic, numeric, and symbolic Unicode characters which are not in the Basic Latin Unicode block to their ASCII equivalents, if one exists and *SynonymGraphFilterFactory*, that allows matching a token with its synonyms (defined in a *synonyms.txt* file).
- The *daterange* field type applies the *DateRangeField* that allows indexing date ranges.

The summary of the custom field types can be found in Table 1.

Table 1: Custom Field Types

Field Type	Filter	Index	Query
standart_text	ASCIIFoldingFilterFactory	X	X
	LowerCaseFilterFactory	X	X
	SynonymGraphFilterFactory	X	X
daterange	DateRangeField		

Table 2 contains information about the fields relative to each document in the collection.

Table 2: Schema Fields

Field	Field Type	Indexed
imdb_title_id	standard_text	false
rotten_tomatoes_link	standard_text	false
original_title	standard_text	true
original_release_date	daterange	true
streaming_release_date	daterange	true
country	standard_text	true
language	standard_text	true
production_company	standard_text	true
directors	standard_text	true
writer	standard_text	true
budget	standard_text	false
worlwide_gross_income	standard_text	false
genres	standard_text	true
content_rating	standard_text	true
runtime	pint	true
movie_info	standard_text	true
audience_rating	pint	true
tomatometer_rating	pint	true
total_votes	pint	true
mean_vote_imdb	pfloat	true
votes_10	pint	false
votes_9	pint	false
votes_8	pint	false
votes_7	pint	false
votes_6	pint	false
votes_5	pint	false
votes_4	pint	false
votes_3	pint	false
votes_2	pint	false
votes_1	pint	false
available_netflix	standard_text	true
available_prime_video	standard_text	true
available_disney_	standard_text	true
review_content	standard_text	true

11 RETRIEVAL PROCESS

After finishing the indexing process, we have a complete schema to add the fields with the correct field types and use the *post* tool to populate the collection. To evaluate the system's performance, we defined 5 different information needs, based on the search tasks previously discussed (Section 5). *Solr* offers different query parsers. During the development of this project, we explored *DisMax* (*Maximal Disjunction*) and EDisMax (*Extended Maximal Disjunction*).

The query fields used were the indexed ones. Moreover, for each information need, we created a list with the relevant documents, thus allowing us to evaluate the results using the metrics referred in Section 7.

A. Space movies on Netflix (IN1)

Information need: Movies set in a spatial environment and are available to watch on Netflix.

User story: "As a Netflix client and interested in science fiction movies featuring intergalactic traveling, I want to find movies related to that theme."

Relevance Judgement: On this query, our goal is to retrieve movies with a spatial theme. To obtain good results, we need to use different fields: we can search for the *Science Fiction & Fantasy* genre, as well as space-related keywords on the textual reviews, movie title and description. Since we are only interested in movies available on Netflix, we need to apply a filter (*fq*) to the *available_netflix* field.

Query (q): "space sci-fi"
Filter query (fq): 'available_netflix: "True"'
Query fields (qf): *genres*, *original_title*, *movie_info* and *review_content*

Table 3: Weights applied to IN1

Field	Weight
genres	3
original_title	1
movie_info	5
review_content	2

Results: The first two systems resulted in a poor information retrieval performance, as can be seen in Table 4. However, it is noticeable that, when applying boosts (Table 3) to certain fields, the *Average Precision* (*AvP*) increased up to 93%. This means that relevant results appear much sooner on system 3 (schema with boosts). In this query, we can prove that weights have a high influence over the relevance of the first 10 documents retrieved. The precision-recall curve for this information need is displayed in Figure 12 (*Attachments*). From this curve, we can infer that the system with the weights filter is the one with the best performance. This shows that the weights hold a significant importance in the relevance of the documents.

Table 4: IN1 Results

Rank	Schemaless	Schema	Boosted
1	N	N	R
2	N	N	R
3	R	R	R
4	N	N	R
5	R	N	R
6	N	N	N
7	N	N	N
8	N	N	N
9	N	R	N
10	N	R	R
AvP	0.366667	0.285185	0.933333
P@10	0.20	0.30	0.60

B. Movies about slavery (IN2)

Information need: Movies that portray slavery and the different points of view over time

User story: "As a History enthusiast, I want to know more about the slavery topic"

Relevance Judgement: On this query, our goal is to retrieve movies that touch on the slavery subject. In order to obtain good results, we made our search by using slavery-related keywords on the movie title, movie info and review content.

Query (q): slavery slave
Query fields (qf): *original_title*, *movie_info* and *review_content*

Table 5: Weights applied to IN2

Field	Weight
original_title	1
movie_info	5
review_content	3

Results: By analyzing the Table 6, we noticed that the precision (*P@10*) had a slight increase (about 10 percentual points) with the boosts in Table 5, relative to the other systems. The Average Precision achieved with boosting was significantly better as well (aproximately 25 percentual points). However, Schema and Schemaless systems produced similar results, as the values were identical in both fields. Bearing this in mind, as we wanted to avoid the keywords passed in *the query (q)* to point to non-relevant or out-of-context movies, the movie_info field had the highest boost value to ensure that our results were suited to the input. The precision-recall curve for all the different systems relative to this information need can be consulted in Figure 13 (*Attachments*). This curve shows us that the systems have similar performance in the first few documents. After that, the performance of both the *Schemaless* and *Schema* systems fall.

Table 6: IN2 Results

Rank	Schemaless	Schema	Boosted
1	R	R	R
2	N	N	R
3	R	N	R
4	N	R	R
5	R	N	N
6	N	N	N
7	N	R	N
8	N	R	R
9	R	N	N
10	R	N	R
AvP	0.642222	0.607143	0.870833
P@10	0.50	0.40	0.60

C. Emotive movies about World War II (IN3)

Information need: Emotive movies about World War II with high IMDB rating.

User Story: "As a history enthusiast, I want to find emotive movies about World War II with a high IMDB rating."

Relevance Judgement: On this query, we are expecting to retrieve documents related to emotive movies set in World War 2 with a high IMDB rating. To obtain movies with a high average score on IMDB, we apply a filter (*fq*). To retrieve movies considered emotive, the textual reviews are very important, since they contain the users' subjective thoughts on the respective movie (e.g., a user that found the movie touching and emotive will likely express his feelings in this field).

Query (q): "World war II" emotional
Filter query (fq): "mean_vote_imdb: [8.0 TO 10.0]"
Query fields (qf): *movie_info* and *review_content*

Table 7: Weights applied to IN3

Field	Weight
movie_info	2
review_content	4

Results: For this Information Need, we can see in Table 8 that the *Precision@10* was similar for all the 3 systems. The chosen keyword ("emotional") is a keyword mentioned in several non-relevant documents. When comparing the *Average Precision* metric, we can see that the *Boosted* system is the best one. This means that, for this use case, the boosts (Table 7) are indeed correct and make the relevant documents appear earlier. The precision-recall curve relative to this information need is identical for both *Schemaless* and *Schema* systems, as can be seen in figure 14 (*Attachments*).

Table 8: IN3 Results

Rank	Schemaless	Schema	Boosted
1	N	N	R
2	R	R	R
3	N	R	R
4	R	R	R
5	R	N	R
6	R	R	N
7	N	N	N
8	N	N	N
9	R	R	R
10	N	N	N
AvP	0.56	0.63	0.94
P@10	0.50	0.50	0.60

D. Movies about true crime stories (IN4)

Information need: Movies with a plot based on true crimes.

User story: "As a user interested in criminal stories based on real events, I want to find movies that describe such occurrences."

Relevance Judgement: On this query, our goal is to retrieve movies with a plot that is based on real crimes. In order to obtain good results, we searched for true crime-related keywords in movie information, as well as in review contents.

Query (q): "true crime story"
Query fields (qf): *movie_info* and *review_content*

Table 9: Weights applied to IN4

Field	Weight
movie_info	3
review_content	5

Results: The *schemaless* and *schema-only* systems both resulted in poor but similar results, as we can see in Table 10. Nonetheless, it is clear that boosting certain fields (Table 9) and applying phrase boost slop (3 words) - ***ps**: 3* - influences the results, since both the precision for the first 10 documents (P@10) and Average Precision (*AvP*) increased. With this in mind, we can conclude that, in this query, boosting and *ps* have a visible influence over the relevance of documents retrieved. The precision-recall curve for this information need can be found in Figure 15 (*Attachments*). Similar to what happens in IN1, the weights filter has a noticeable influence on the performance of the information retrieval. The *schemaless* system has an identical performance compared to the system with *schema-only*.

Table 10: IN4 Results

Rank	Schemaless	Schema	Boosted
1	N	N	R
2	R	R	R
3	R	R	R
4	N	N	R
5	R	R	R
6	N	R	N
7	N	N	N
8	R	R	R
9	N	N	N
10	N	N	N
AvP	0.57	0.61	0.96
P@10	0.40	0.50	0.60

E.Family movies to watch on Christmas (IN5)

Information need: Family-friendly movies to watch on the Christmas holiday.

User story: "As a user, I want to find an appropriate movie (suited for family) to watch with my family on Christmas eve."

Relevance Judgement: On this query, our goal is to retrieve movies that other users consider to be about Christmas or appropriate to watch during that time. Therefore, it is important to use textual reviews, as well as movies' titles and descriptions.

Query (q): "Christmas time"
Filter query (fq): 'genres:"Kids & Family"'
Query fields (qf): *original_title*, *movie_info* and *review_content*

Table 11: Weights applied to IN5

Field	Weight
movie_info	3
review_content	4

Results: In this need, every system showed similar results regarding *P@10*, as can see in Table 12. On the other hand, when measuring the *AvP* metric we can see that the boosted system was the best one, obtaining an Average Precision close to 90%. Bearing this in mind, we can conclude that the keywords were somewhat ambiguous on their own, which was taken care of by adding a phrase boost slop (5) to remove out-of-context keyword references. The boosts applied in this Information Need can be seen in Table

11: the *review_content* field is the most important one, since it is more likely to find the information about a certain movie being suited to watch during Christmas than on the description of the movie or its title.

Table 12: IN5 Results

Rank	Schemaless	Schema	Boosted
1	R	R	R
2	R	R	R
3	N	N	R
4	R	R	R
5	N	N	N
6	N	N	R
7	R	R	N
8	R	R	R
9	N	N	R
10	R	R	N
AvP	0.757738	0.757738	0.908730
P@10	0.60	0.60	0.70

12 EVALUATION CONCLUSIONS

After collecting the results from the information needs experiments described, we can now compute the *Mean Average Precision* (MAP) of the 3 systems. Table 13 contains MAP values for each system.

Table 13: *MAP* Values for each system

Schemaless	Schema	Boosted
0.5793254	0.5780132	0.9225792

After evaluating the results from the Information Needs, we can see that system 3, which applies weights with a schema is the one with the best results. As expected, the weights improve the relevance of the documents retrieved. However, despite being the best system (with the highest *MAP*), it is still vulnerable to ambiguous keywords - keywords that exist in several non-relevant documents in the collection.

On the other hand, system 2 (*schema-only*) is not an improvement over the *schemaless* system. This means that the *schema* is something that could use improvement. It is also important to notice that the schema does not improve results for all information needs, since every information need is specific and may benefit from different *schemas*.

13 SEARCH SYSTEM IMPROVEMENTS

In this section, we will describe several improvements to the search system. There are many approaches that can be followed to improve search results. After carefully evaluating many possibilities, we decided that, given our context, we should target the following concepts:

- integrate *OpenNLP* for Natural Language Processing tasks
- integrate *Learning to Rank* to reorder the most relevant documents
- improve synonym term matching with *Python* libraries
- develop a graphical interface

In order to measure the practical improvement of the search systems, we must use the same queries as the ones used in Section 11 and guide ourselves by the same metrics.

13.1 OpenNLP

Apache OpenNLP [22] is an open-source library that supports the most common Natural Language Processing (**NLP**) tasks. We opted for this library because it is already included with *Solr* distribution.

A. Filters and Tokenizers

With *OpenNLP* now integrated in *Solr*, we can modify the schema, to apply a more complex *tokenization* method in full-text fields (e.g., *review_content*, *movie_info* and *original_title*). In order to accomplish this task, we can use the class *OpenNLPTokenizer* that detects sentences and tokenizes punctuation.

Furthermore, we apply a new set of filters implemented by *OpenNLP* classes. The first filter that we used was *OpenNLPPOSFilter*, allowing part-of-speech tagging. Each token is assigned a special label in a text corpus indicating the part of speech and other grammatical categories. These tags are used by other classes, that perform other NLP tasks.

The second filter used in the schema was the *OpenNLPChunkerFilter*. This class allows the identification of short phrases in a given sentence (chunks).

Both filters were used with pre-trained models for English text found in OpenNLP Sourceforge website [24].

B. Named-entity Recognition

Named-entity Recognition (NER) is a very common NLP task. It aims to identify and categorize key information (entities) in a text field, using machine learning models. Our goal with the integration of NER was to enrich our dataset, creating new fields with the entities recognized in full-text fields. The *OpenNLP* library implements extraction of several types of entities. For our specific use case, we decided that it only makes sense to recognize people's names(e.g., "Tony Stark"), organizations (e.g., "FBI") and dates (e.g., "90s").

To integrate this NLP task, we used an *OpenNLP* processor class added to the default update request chain (chain of events that occur when an update is called), that extracts entities on the text. Similarly to the classes added to the schema, we used pre-trained models for entity recognition. However, it is noticeable that the system fails to recognize some entities, as they were not in the training set.

Having incorporated this NLP task to our indexing process, we enhanced the scope of our system. There is a considerable amount of information recognized by this NER mechanism that allows us to clearly identify important entities on full-text fields. Regarding concrete results, we are now able to gather information (e.g., characters' names, organizations, dates or time periods) that further improve the search functionality.

Unfortunately, due to the ambiguity of natural language and the fact that the machine learning models were not trained for

our specific use case, there are some organizations that don't get recognized.

13.2 Learning to Rank

Learning to Rank (LTR) [26] is a module that exists in the **contrib** package of *Solr*'s distribution. It allows for configuration of machine learning models in *Solr*. The main concept in this module is **Query Re-ranking**, which runs a more complex query to re-rank the top N retrieved documents. This re-ranking process is achieved though a ranking model. By default, *Solr* supports three different models: *Linear Model*, which is the simplest one, calculating an overall score using the dot product between the weights vector and the features vector; *Multiple Additive Trees Model*, that uses *Gradient Boosted Trees* and *LambdaMart* [5] technique; *Neural Network Model* which is a ranking model based on neural networks, also known as *RankNet* [6].

Generally, linear models are trained using a **pointwise** approach (i.e., the model asks, "Is this document relevant to the query or not?"), whereas tree based models and neural network models are trained using a **pairwise** approach (i.e., given a pair of documents, they try and come up with the optimal ordering for that pair and compare it to the ground truth). For LTR problems, **pairwise** approaches work better, since predicting relative order is closer to the nature of ranking than predicting a class label.

However, it is still possible to train linear models, using a **pairwise** approach, with the *SVMRank* [21] implementation. This learning retrieval function is a variant of *Support Vector Machine* [20] algorithms that transforms the ranking problem into a two-class classification problem, by calculating $(x_k^{'}, y_k^{'}) = (x_i - x_k, sign(y_i, y_j))$ for all comparable pairs. This operation is called a *pairwise transformation*. The goal is to find an optimal vector of weights (coordinates of the resultant hyperplane) to pass as parameters to the linear model. In order to find this vector, we use the *sklearn* [17] *Python* library to implement *SVMRank*.

Before building the model, we need to extract features from the documents being scored. This feature engineering phase used the *ltr* module class *org.apache.solr.ltr.feature.SolrFeature*. The original score is also retrieved as a feature using *OriginalScoreFeature*.

Table 14: LTR Features

Name	params
maximize_votes	{ "q" : "{!func}scale(total_votes,0,1)" }
maximize_rating	{ "q" : "{!func}scale(mean_vote_imdb,0,1)" }
review_bm25	{ "q" : "{!dismax qf='review_content'}${text}
description_bm25	{ "q" : "{!dismax qf='movie_info'}${text}" }
original_score	{ }

The features extracted are available in Table 14. It is important to notice that two of the features (*review_bm25* and *description_bm25*) use external feature information (efi). In Subsection 13.4, we explain the additional syntax necessary.

The data that we used to train this model was retrieved from the information needs described on Section 11: for each query, we gather the features belonging to the top 40 most relevant documents, for a total of 200 tuples (before *pairwise transformation*). The short amount of examples, combined with the imbalance of labels (only 34 positive labels before *pairwise transformation*) makes it so that we cannot ensure that the model is not *overfitting*/*underfitting* the data. In order to mitigate this problem, we would need to feed a considerable amount of additional data to the model, which was not possible in the context of this project.

13.3 Synonyms

The main point behind the usage of synonyms was to make our search system recognize and retrieve documents that were being considered insignificant by *Solr* initially, but ended up being relevant to the query.

In order to maximize the number of synonyms, we decided to use *Natural Language Toolkit* [14] (NLTK), since it is a leading platform for building *Python* programs with NLP and it provides easy-to-use interfaces to over 50 corpora and lexical resources (e.g., *WordNet*), along with a suite of text processing libraries for classification, tokenization, stemming, among other tasks.

This way, by using NLTK, all nouns present in the content of the reviews (*review_content* field) were extracted, in order to build a set of nouns from where we generated synonyms that were later integrated in the search system.

Finally, we ended up producing several synonyms for more than 21000 nouns that were stored in *synonyms.txt* and incorporated them in our schema file (*schema_movies.json*), more precisely in the *SynonymGraphFilterFactory* filter.

This approach has some drawbacks. Since natural language is ambiguous and the terms' meanings are highly dependent on the context, some terms are matched with others without necessarily having the same conotation.

13.4 Overall System Performance

In this subsection, we will compare the results obtained from the *boosted* system (using a schema and applying boosts to the query fields) with the results obtained using the enhanced system (after implementing and applying the improvements described in Section 13). In order to establish this comparison, we will be using some of the Information Needs described in Section 11. The weights remain the same, except for the new fields (e.g., *ner_dest_org*, *ner_dest_date*, *ner_dest_person*).

Furthermore, to enable query re-ranking with LTR (Subsection 13.2), we need to pass an additional parameter in the query's syntax - the **rq** parameter (ranking query). Inside this field, we also need to specify the *efi* parameters used to calculate the features' respective scores for each document.

A. Movies about slavery (IN2)

Information need: Movies that portray slavery and the different points of view over time.

Query (q): slave

Query fields (qf): *original_title*, *movie_info* and *review_content*

Ranking Query (rq)= {!ltr model=*my_svm_model* efi.text="slave" reRankDocs=50}

Table 15: IN2 Result Comparison

Rank	Schema with weights	Improved System
1	R	R
2	R	R
3	R	R
4	R	N
5	N	R
6	N	R
7	N	R
8	R	R
9	N	R
10	R	R
AvP	0.870833	0.906041
P@10	0.60	0.90

Results: Regarding Average Precision, the results using both systems are very similar, with a slight increase in the improved system. Nonetheless, taking a look at Table 15, there is a considerable increase in number of relevant results in the 10 first documents retrieved (up 30 percentual points). In such a query, where a general topic like "slave" is used, these improvements can be explained by the usage of semantic analysis and an accurate matching of synonyms to expand the search. For this Information Need, the precision-recall curve is represented on Figure 18.

B. Movies about true crime stories (IN4)

Information need: Movies with a plot based on true crimes.

Query (q): "true crime story"

Query fields (qf): *movie_info* and *review_content*

Ranking Query (rq)= {!ltr model=*my_svm_model* efi.text="true crime story" reRankDocs=50}

Table 16: IN4 Result Comparison

Rank	Schema with weights	Improved System
1	R	R
2	R	R
3	R	R
4	R	R
5	R	N
6	N	R
7	N	N
8	R	N
9	N	N
10	N	N
AvP	0.96	0.97
P@10	0.60	0.50

Results: The results comparison for this Information Need is displayed in Table 16. Despite having slightly better AvP measure, the Improved System's *Precision@10* decreases by 10 percentual points relatively to the unimproved system. Moreover, by analyzing the results, we can see that non-relevant (N) documents are retrieved earlier in the Improved System. This Information Need illustrates the fact that sometimes semantic analysis may lead to inferior results. The documents retrieved from the Improved System take into account the synonyms of each keyword, suffering from the ambiguity described in Subsection 13.3: the keyword "crime" has a large set of synonyms (e.g., "massacre", "violence", "offense") associated with irrelevant documents. This example is the opposite of what we see in IN2, where the synonyms significantly boost the results obtained.

Furthermore, the precision-recall curve for this Information Need using these systems was generated and is displayed in Figure 20.

C.Family movies to watch on Christmas (IN5)

Information need: Family-friendly movies to watch on the Christmas holiday.

Query (q): "Christmas time"

Filter query (fq): 'genres:"Kids & Family"'

Query fields (qf): *original_title*, *movie_info*, *review_content* and *ner_dest_time*

Ranking Query (rq)= {!ltr model=*my_svm_model* efi.text="Christmas" reRankDocs=50 }

Table 17: IN5 Result Comparison

Rank	Schema with weights	Improved System
1	R	R
2	R	R
3	R	R
4	R	R
5	N	R
6	R	R
7	N	R
8	R	R
9	R	R
10	N	N
AvP	0.91	1.00
P@10	0.70	0.90

Results: By analyzing the results (Table 17), we can conclude that the improvements in the system are beneficial for this Information Need: in the first 10 documents retrieved, only the last one is non-relevant (N). Both *AvP* and *P@10* metrics support this conclusion, as the values increase 9 and 20 percentual points, respectively. Bearing this in mind, we consider that, for this Information Need, the *NER* task holds a significant importance, since in all relevant documents, the keyword "Christmas" is recognized as a date entity. The high boost given to this field prioritizes these documents in the retrieval phase.

The precision-recall curve that illustrates this comparison can be found in Figure 21.

13.5 Overall improvements results

From a general point of view, we consider that the improvements added to the search system and improved its overall quality. This conclusion is supported by the comparison between the *Mean Average Precision* of both systems (Table 18). The Improved System performed better then the System without improvements by aproximately 3 percentual points.

However, there are some use cases where more complex semantical analysis operations don't increase the relevance of documents

retrieved and may even slighly decrease it. This is due to the high ambiguity of natural language.

Table 18: *MAP* Values for each system

Schema with weights	Improved System
0.9228332	0.9577162

On the other hand, since we are only considering improvements of performance for a small set of Information Needs, we must not conclude that one system is better than the other. In order to state this, we would need to consider a much larger amount of Information Needs for which the improvements implemented could lead into better or worse results.

Regarding the *Learning to Rank model*, we conclude that its importance in achieving an optimal ordering in the documents retrieved is not clear by the analysis of the Information Needs in this subsection. However, by analysing Information Needs 1 (Table 19 and Figure 17) and 3 (Table 20 and Figure 19), we can see that when the metrics show very similar results, the system with LTR implemented contains relevant results earlier. This reflects the (low) impact of the machine learning model responsible for the reordering. This bellow expectations behaviour can be justified with the short amount of data fed to the model. The quality of this data was not great either, since the training dataset had unbalanced labels (large amount of non-relevant documents compared to relevant ones). On the other hand, we still consider this approach suited for our use case, especially if we consider a large scaled Information Retrieval System where it would be possible to gather a much larger amount of data.

13.6 Graphical Interface

Our last improvement was the development of a user interface. Initially, our option, in order to interact with the search system, was to use the *Solr* GUI, but this approach was not an easy-to-use choice because some intermediate level knowledge regarding GUI commands was needed.

This way, we decided to design a simple web page which was mainly made to improve the overall user experience and it was also a friendly way to interact with our Information Retrieval System. To implement our interface, we needed tools that could easily incorporate the search queries. With this in mind, *NodeJS* was used alongside the *solr-node* package for the back end. Moreover, the front end was implemented in an intuitive way and distributed in two different pages: Search Page (Figure 22) and Movie Page (Figures 23, 24, 25 and 26).

Furthermore, to complement the details, a poster and a trailer were added to every movie and, in order to obtain in-depth information about each review, another core was produced for the reviews collection. This is not an enhancement of the search system, since this core is only used to get specific points about the reviews, such as critics name, publisher and review date.

14 REVISIONS ADDED

During the process of adding new features to the search system, we also modified some aspects of the implementation carried from earlier stages.

Before including the *OpenNLP* to the schema, we added additional filters (e.g., *EnglishPossessiveFilter* and *EnglishMinimalStemFilterFactory*) that improved search results to an extent.

15 CONCLUSIONS AND FUTURE WORK

Our focus, in the first phase, was to choose a dataset with recent data and manipulate it to better understand its information and make it ready for the information retrieval phase. This goal was accomplished, despite having some problems with the merging and refinement process as well as some missing rows' values. The result is a reasonably large dataset (over 9,000 movies) with coherent data about movies, ratings and reviews.

After having the preprocessing stage done, we selected the *Solr* tool to develop an *Information Retrieval System*. We defined the indexing process, choosing what fields should be indexed and what custom field types should be added to the *Solr* schema. Having populated the collection, we chose 5 different information needs to evaluate the systems. This evaluation was done using the metrics *Precision@10* and *Average Precision*.

In a third phase, our focus was to further improve our search system. In order to achieve this, we applied common Natural Language Processing tasks with *OpenNLP*, that increases the complexity of the semantic analysis done in the indexing phase; trained a model to incorporate with *Solr Learning to Rank* to improve the order in which the documents are retrieved; and synonyms to expand the scope of our search. Generally, this new version of the search system obtains better results.

As future work to be developed in this project, it would benefit our search system to train our own models used in semantical analysis (*OpenNLP*), as well as feeding more data to our *rankSVM* reranking model. This data could be collected through a user feedback mechanism.

REFERENCES

[1] W. Bruce Croft et al. *Search engines : information retrieval in practice*. 2010. URL: https://catalogo.up.pt/F/?func=direct&doc_number=000567357 (visited on 01/20/2022).

[2] Christopher D. Manning et al. *Introduction to Information Retrieval*. 2008. URL: https://catalogo.up.pt/F/?func=direct&doc_number=000537893 (visited on 01/20/2022).

[3] *Elasticsearch is the distributed, RESTful search and analytics engine at the heart of the Elastic Stack*. Last access: 15/12/2021. URL: https://www.elastic.co/pt/elasticsearch/.

[4] John Hunter. *A comprehensive library for creating static, animated, and interactive visualizations in Python*. Last access: 15/12/2021. URL: https://matplotlib.org/.

[5] *LambdaMART is one of the Learning to Rank (LTR) algorithms developed by Chris Burges*. Last access: 17/01/2021. URL: https://www.educative.io/edpresso/what-is-lambdamart.

[6] *Learning to Rank for Information Retrieval: A Deep Dive into RankNet*. Last access: 17/01/2021. URL: https://towardsdatascience.com/learning-to-rank-for-information-retrieval-a-deep-dive-into-ranknet-200e799b52f4.

[7] Stefano Leone. *IMDB movies extensive dataset*. Last access: 15/12/2021. URL: https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset.

[8] Stefano Leone. *Rotten tomatoes movies and critic Reviews Dataset*. Last access: 15/12/2021. URL: https://www.kaggle.com/stefanoleone992/rotten-tomatoes-movies-and-critic-reviews-dataset?select=rotten_tomatoes_critic_reviews.csv.

[9] *Listings of movies and tv shows on Netflix.* Last access: 15/12/2021. URL: https://www.kaggle.com/shivamb/netflix-shows.

[10] *mAP (mean Average Precision).* Last access: 17/01/2021. URL: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2.

[11] *Movies and TV Shows listings on Amazon Prime Video.* Last access: 15/12/2021. URL: https://www.kaggle.com/shivamb/amazon-prime-movies-and-tv-shows.

[12] *Movies and TV Shows listings on Disney+.* Last access: 15/12/2021. URL: https://www.kaggle.com/shivamb/disney-movies-and-tv-shows.

[13] Andreas Mueller. *Word Cloud is a data visualization technique used for representing text data.* Last access: 15/12/2021. URL: https://amueller.github.io/word_cloud/.

[14] *NLTK - Natural Language Toolkit.* Last access: 18/01/2021. URL: https://www.nltk.org/.

[15] *Pandas.* Last access: 15/12/2021. URL: https://pandas.pydata.org/.

[16] Kenneth Reitz. *An elegant and simple HTTP library for Python.* Last access: 15/12/2021. URL: https://docs.python-requests.org/en/latest/.

[17] *scikit-learnMachine Learning in Python.* Last access: 17/01/2021. URL: https://scikit-learn.org/stable/.

[18] *Solr is the popular, blazing-fast, open source enterprise search platform built on Apache Lucene.* Last access: 15/12/2021. URL: https://solr.apache.org/.

[19] *Spacy industrial-strength natural language processing in python.* Last access: 15/12/2021. URL: https://spacy.io/.

[20] *Support Vector Machine — Introduction to Machine Learning Algorithms.* Last access: 17/01/2021. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[21] *Support Vector Machine for Ranking.* Last access: 17/01/2021. URL: https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html.

[22] *The Apache OpenNLP library is a machine learning based toolkit for the processing of natural language text.* Last access: 17/01/2021. URL: https://opennlp.apache.org/.

[23] *The Apache Software Foundation provides support for the Apache community of open-source software projects.* Last access: 15/12/2021. URL: https://lucene.apache.org/.

[24] *The models are language dependent and only perform well if the model language matches the language of the input text.* Last access: 17/01/2021. URL: http://opennlp.sourceforge.net/models-1.5/.

[25] Michael Waskom. *Seaborn is a Python data visualization library based on matplotlib.* Last access: 15/12/2021. URL: https://seaborn.pydata.org/.

[26] *With the Learning To Rank (or LTR for short) contrib module you can configure and run machine learned ranking models in Solr.* Last access: 17/01/2021. URL: https://solr.apache.org/guide/8_7/learning-to-rank.html.

[27] *Your machine learning and Data Science Community.* Last access: 15/12/2021. URL: https://www.kaggle.com/.

Attachments

This page contains the attachments for all mentioned figures and tables throughout this paper.

Table 19: IN1 Result Comparison

Rank	Schema with weights	Improved System
1	R	R
2	R	R
3	R	R
4	R	R
5	R	R
6	N	N
7	N	N
8	N	R
9	N	R
10	R	N
AvP	0.933333	0.932540
P@10	0.60	0.70

Table 20: IN3 Result Comparison

Rank	Schema with weights	Improved System
1	R	R
2	R	R
3	R	R
4	R	R
5	R	R
6	N	N
7	N	R
8	N	N
9	R	N
10	N	N
AvP	0.94	0.98
P@10	0.60	0.60

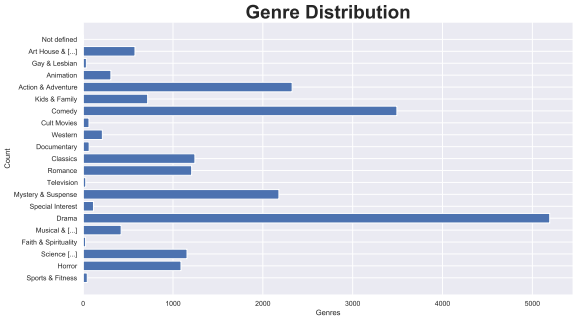

Figure 9: Genre distribution

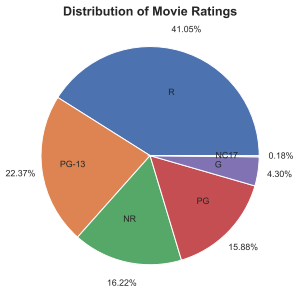

Figure 10: Content-rating distribution

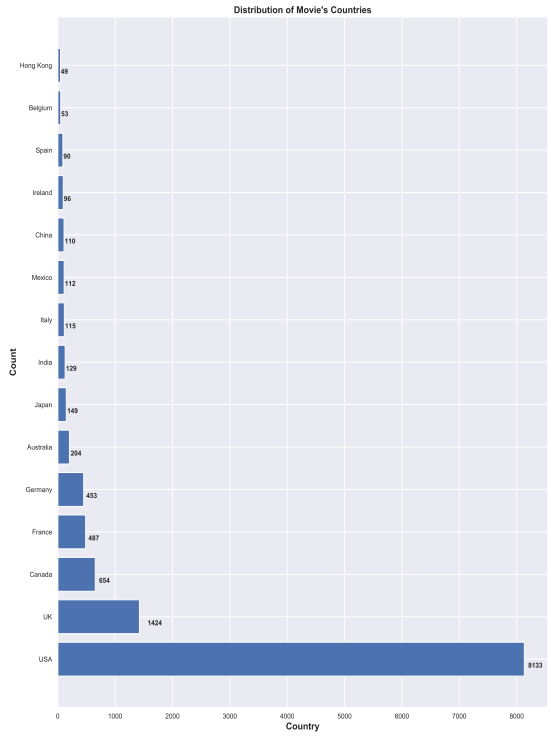

Figure 11: Production Country distribution

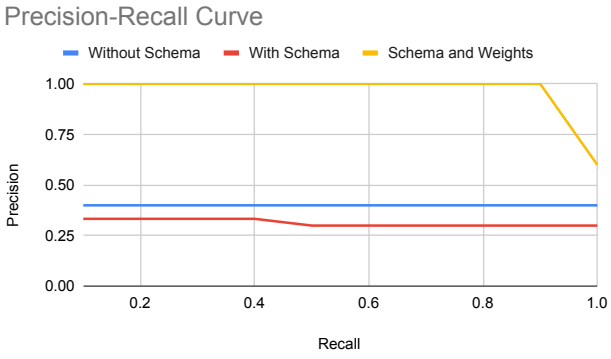

Figure 12: "Precision-recall curve for IN1"

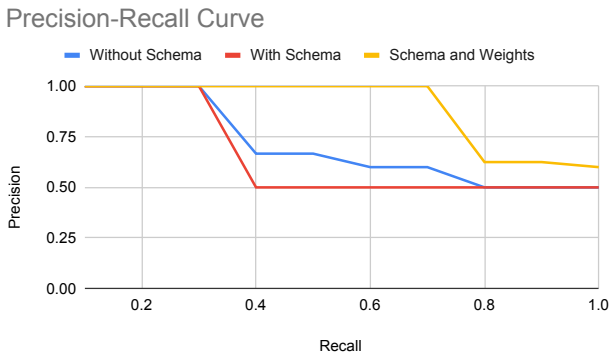

Figure 13: "Precision-recall curve for IN2"

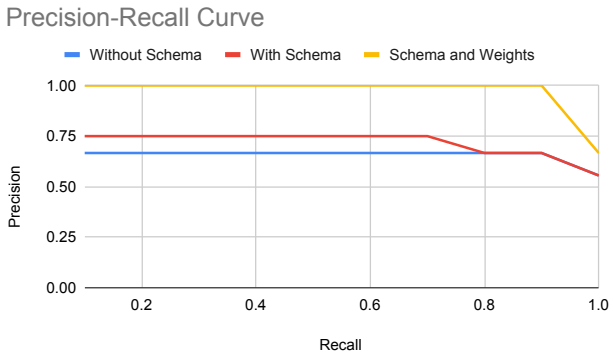

Figure 14: "Precision-recall curve for IN3"

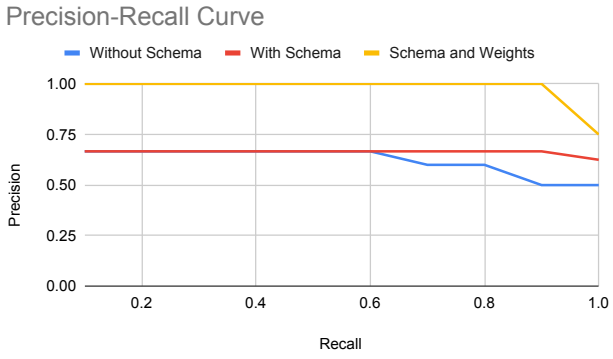

Figure 15: "Precision-recall curve for IN4"

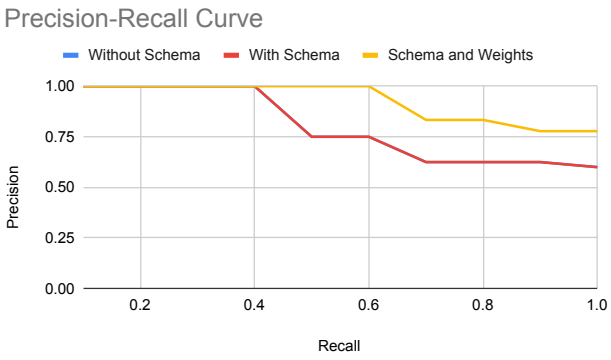

Figure 16: "Precision-recall curve for IN5"

Figure 19: "Results comparison - PR curve for IN3"

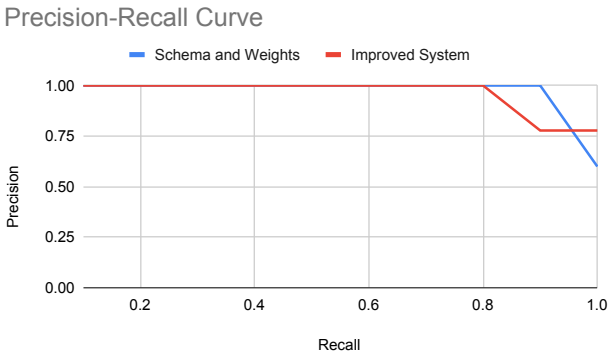

Figure 17: ""Results comparison - PR curve for IN1"

Figure 20: "Results comparison - PR curve for IN4"

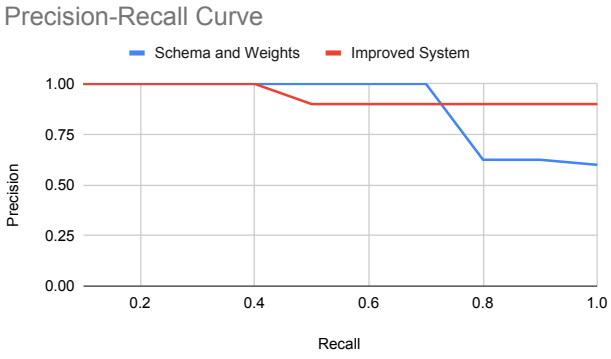

Figure 18: ""Results comparison - PR curve for IN2"

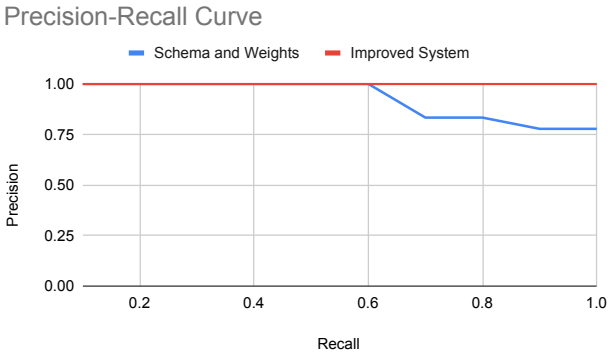

Figure 21: "Results comparison - PR curve for IN5"

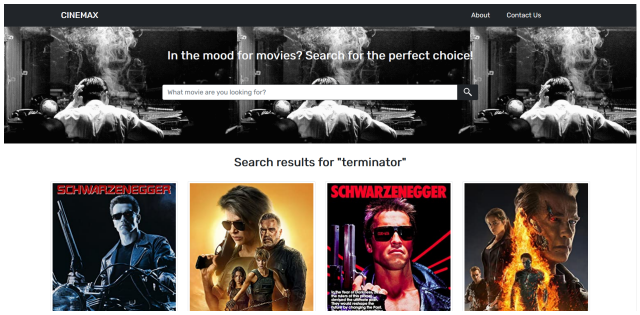

Figure 22: "Search Page"

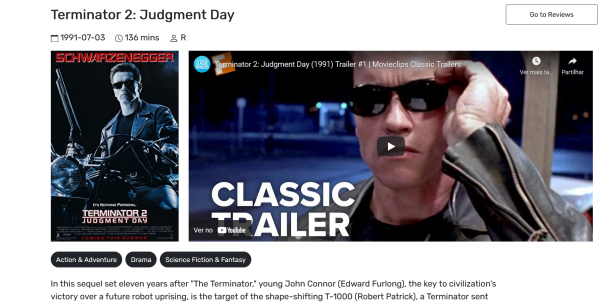

Figure 23: "Movie Page"

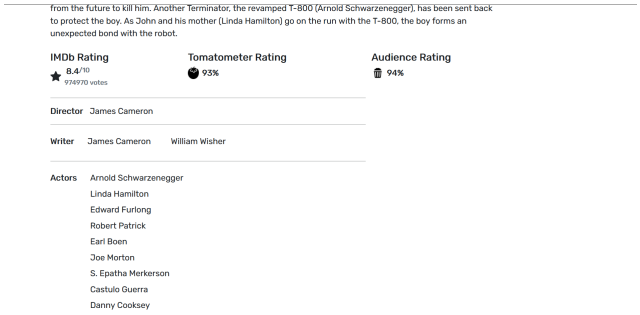

Figure 24: "Movie Page"

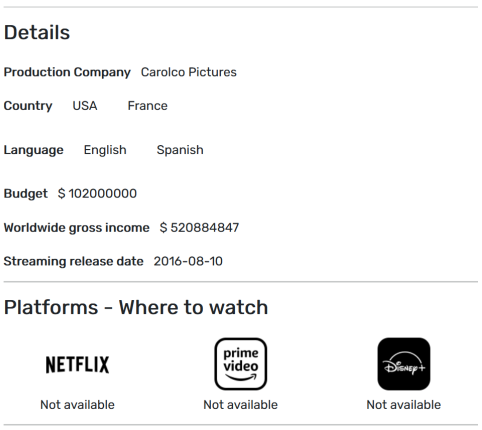

Figure 25: "Movie Page"

Figure 26: "Movie Page"