Netflix titles and IMDb reviews: data extraction, preparation & analysis

Ricardo Fontão
FEUP
Porto, Portugal
up201806317@up.pt

Telmo Baptista
FEUP
Porto, Portugal
up201806554@up.pt

Tiago Silva
FEUP
Porto, Portugal
up201806516@up.pt

***Abstract*—In the current age, people start to spend more and more time watching movies and TV shows in their free time, utilizing a multitude of services to provide their favorite shows. One of the most popular streaming services is Netflix, counting with an ever-increasing number of monthly users and shows. This project joined Netflix titles and IMDb user review data to later develop a search engine and evaluate it with various relevant metrics.**

***Index Terms*—data collection, data preparation, data processing, data analysis, movies, TV shows, Netflix**

I. Introduction

Netflix has been steadily growing in subscribers over the years, constantly adding new TV Shows and Movies. [1] To better understand people's opinions on these shows, in this first step of the project we combine a Netflix dataset with a dataset of reviews from the IMDb platform. In this report, we will detail all the steps taken, from downloading the data in the Data Collection section, processing it in the Data Preparation section to building a pipeline to automate the process in the Pipeline section. We will also make an exploratory data analysis to understand the characteristics of the data and how it is distributed.

For information retrieval, we used Elastic Search to upload our data and perform various queries to retrieve relevant information, providing an analysis of the results obtained.

After building a stable system, some experiments will be conducted and evaluated, such as language identification and synonym filter with the goal of improving the system.

II. Data Collection

We required both show information and their respective user reviews. As such, our search for datasets started in the Kaggle platform, chosen thanks to the various diverse and high-quality datasets it provides, most of which also offer an in-depth analysis of the data contained therein.

Our search for a comprehensive Netflix dataset yielded a continuously updated dataset [2] that contained the entirety of Netflix's library. This dataset contains the following 12 columns: the id of the show, title, type of show (TV Show, Movie), director(s), cast, country(ies) of origin, date added to Netflix, release year, parental rating, duration (minutes for movies and seasons for TV Shows), the genres of the show and its description. It contains 8807 rows and takes up approximately 3.4 MB. To download the data we decided to use Kaggle's built-in public API due to its free use, good download speeds, and simple integration into the pipeline. To use this API, Kaggle provides a command-line utility that can be installed from pip (Python's package manager).

After having the Netflix titles, we required their respective reviews. Our first approach to solve this issue revolved around using IMDb's built-in API to download the necessary reviews. This idea was quickly discarded due to the limitations imposed by the API's free tier, which allows only two queries per second, making it unfeasible to download all the required data within a reasonable time frame.

The next step was to fall back to the Kaggle platform, where we found a dataset [3] containing all of IMDb's user reviews as of January 11,$^{\text{th}}$ 2021. This dataset contains approximately 5.5 million reviews and is split into 6 different JSON files, each of which is sized at about 1.3 GB, totaling about 7.78 GB. Kaggle's public API was also used to retrieve this dataset, which downloaded a single zip file with 2.9 GB. The data contained in each row is the summary of the review and the detailed review, the user who posted it, the rating given by the user, the date of the review, a flag to indicate it contains spoilers, and the number of people who found the review useful from the total number of votes.

III. Data Preparation

To ensure effective use of the data collected, it was filtered, cleaned, and transformed from the sources mentioned above.

To better comprehend the problem domain, an initial study on the structure of the information was conducted. In this brief analysis, we focused on the data's overall format and structure, with a careful analysis of each attribute available and its pattern.

A. Filtering data

The reviews' dataset contained information relating to every entry on IMDb, which is a superset of the titles present in Netflix. As this data was not pertinent to the domain of our problem, we decided to filter it out. To perform this filtering, we used the title of the movie or show to match data between both datasets. This reduced the number of reviews from 5.5M (7.8 GB) to approximately 900k (955 MB).

A method to drop shows created before a certain year was implemented, as an option to reduce the number of reviews on our dataset, which may have an impact on later stages of the project.

B. Cleaning and Transformation of data

We cleaned each dataset individually, starting with the Netflix data. We started by looking for null values on the dataset. Attributes like *director*, *cast*, and *country* had plenty of null values, but those are used to indicate that those attributes were unknown. However, the number of null values on the attributes *date_added*, *rating* and *duration* were of small size, so we decided to analyze them further. Through that analysis, we discovered the null values on the attribute *duration* were caused by faulty data, having its value on the *rating* attribute instead. To treat this case, we swapped the values on those rows and as the number of rows was small, we searched manually on Netflix for the maturity rating and filled the correct value on those rows.

Afterward, we analyzed the rows which had the attribute *date_added* with a null value. Through research on the movies and TV shows that expounded those values, we found out that those movies and TV shows were no longer available on Netflix, and thus appeared with the attribute *date_added* as null. We decided to keep these rows as they still contain pertinent data.

For the null values present on the *rating* attribute, we filled with the correct values by manually searching the correct rating again, as the number of rows that presented the issue was small enough to handle it manually.

Other than null values, there was another issue that needed to be treated on the dataset. The attribute *country*, which contains a list of countries in which the show was produced, is a comma-separated list in string format. Various rows, however, started with a comma and a space, which we promptly removed to ease the parsing of the list later on.

After cleaning the Netflix dataset, we moved on to the IMDb review dataset. Searching for null values, we found 3 in the *review_summary* column, which we decided to delete as they were not relevant and the source of the value was unknown. The second column that contained null values was the *rating*. These values come from a time when a score was not needed to publish a review, so we let them stay, as they were still valid data.

After the search for null values was concluded, we decided to change the *helpful* column to simplify the processing later on. This column was split into three new attributes: *helpful*, *unhelpful* and *total*. These features are, respectively, the number of votes considering the review helpful, the number of votes considering the review unhelpful, and the total amount of votes.

IV. Pipeline

To streamline and automatize the collection and processing of data described in both the Data Collection and Data Preparation sections, we created a pipeline. Its purpose is to collect all the necessary datasets, process them and then save them to relational storage at a later stage. This pipeline can be triggered with a single command by using a Makefile. A visual representation of this pipeline can be found at Fig. 1

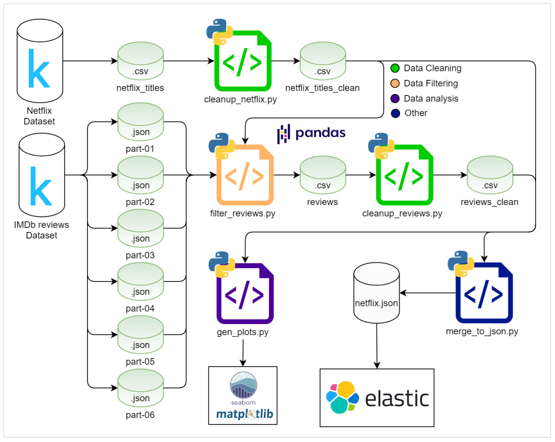

Fig. 1. Data Pipeline

V. Conceptual Model

After the data processing, we created a conceptual model. It can be found in Fig. 2.

The two main classes consist of *Show* and *Review*. *Show* refers to a Netflix show with its attributes brought from the clean dataset. It has associations with the *Person*, *Country* and *Genre* as these are repeated a lot through the entire dataset. The *Person* class represents either an actor or a director, as there are examples of elements belonging to both at the same time. As for the *Review* class, it associates with the *Reviewer* class, which also has a lot of repeated elements across the dataset.

VI. Dataset Characterization

To better understand the data, we conducted an exploratory analysis by creating a set of plots and visual representations of the data. This helped us better understand certain aspects of the data that weren't apparent at first glance. As mentioned previously, the reviews dataset contains about 900k reviews and the Netflix dataset contains 8807 unique titles, from which 2676 are TV Shows and 6131 are movies.

A. Maturity Rating Distribution

To understand the maturity rating distribution of both Movies and TV Shows, we plotted a bar plot that can be found in Fig. 3

By analyzing this plot, it can be concluded that most of the shows present on the Netflix platform are catered to a more mature and adult demographic.

B. Duration Distribution

To analyze the TV Show's duration, we decided to again use a bar plot that can be found in Fig. 4. By analyzing it, we arrive at the conclusion that most of the TV Shows released never

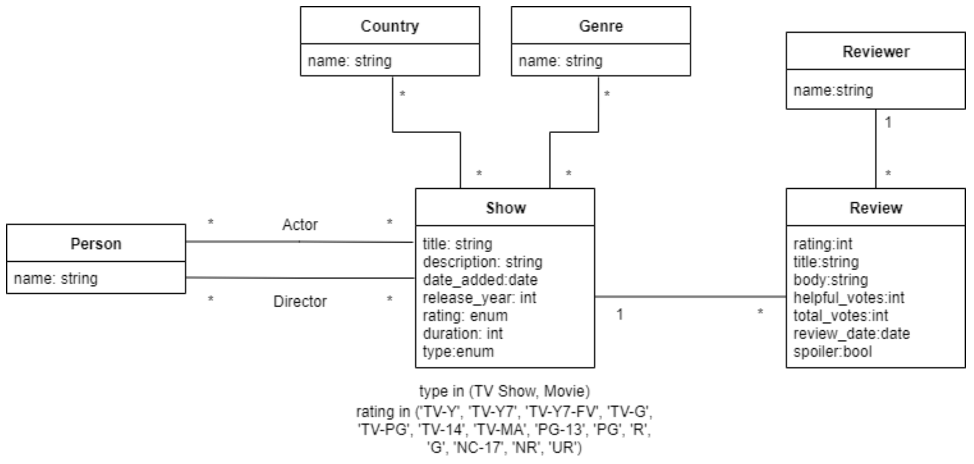

Fig. 2. Conceptual Model

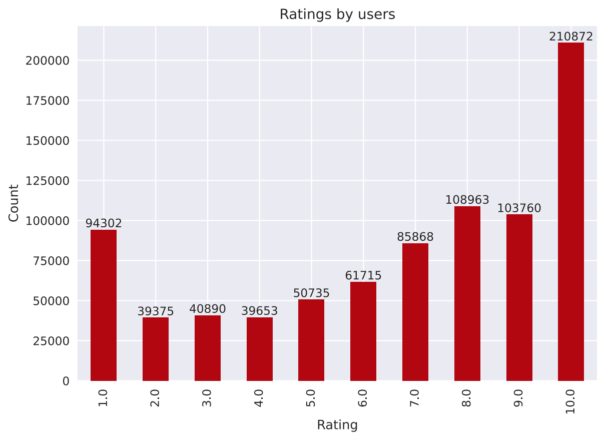

Fig. 3. Amount of shows by Maturity Rating

get a second season. From the 2676 TV Shows on Netflix, only 883 have 2 seasons or more and only 17 shows have 10 seasons or more.

Regarding Movie durations, we decided a histogram was the best way to represent them because it's less discrete than the TV Show's duration. It can be found in Fig. 4. By analyzing the plot, it can be asserted that most movies have durations in the 90 minutes to 110 minutes range.

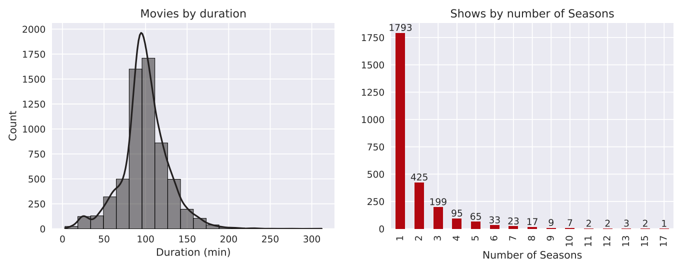

Fig. 4. Duration

C. Show scores

From the rating scores present in the reviews dataset, it is trivial to calculate the score of each title. To analyze this feature, we plotted a histogram that can be found in Fig. 5.

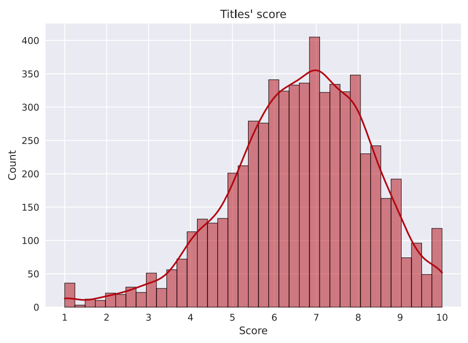

Fig. 5. Title score distribution

By analyzing the plot, we can conclude that most shows are seen in a positive light, since most shows have a score between 6 and 8. On top of that, scores between 0 and 4 seem less prevalent than scores between 8 and 10.

D. User Review Distribution

In a platform like IMDb, there's always a small group of people who have the most presence on rating and reviewing the movies and TV shows that are released every year. Taking this in mind, we decided to analyze the number of reviews made by each reviewer. From the almost 479 thousand reviewers, on average each one made 8 reviews in its lifetime, however, this statistic in itself doesn't accurately represent the distribution. By further analyzing the distribution we found out that the third quartile is equal to 1, this is, 75% of the reviewers had only made a single review in their lifetime. By looking for outliers, the most accusatory value was the reviewer that presented over 1500 reviews, and, among others, caused the huge discrepancy in the distribution.

E. Review Length

To understand better how a review length is affected by other features in the dataset, two plots were created.

The first one can be found in Fig. 6 and presents the distribution of the review lengths across the dataset. From this plot, we conclude that most reviews are of very short length. Even though this doesn't mean the review is of low quality, it probably means the review is a low-effort one. There are, however, a considerable amount of reviews in the 200 to 400 words which represent higher effort reviews.

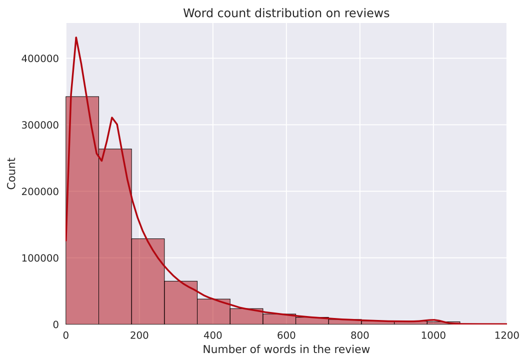

Fig. 6. Amount of Reviews by word count

The second plot is a violin plot and can be found in Fig. 7 and presents the distribution of the review lengths based on its rating. This is an interesting plot that shows that extreme reviews (1 and 10) tend to have fewer words and therefore be of less effort than the more moderate ones (5 to 8).

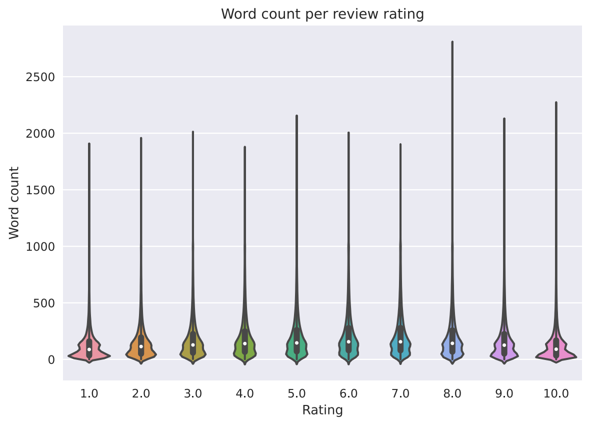

Fig. 7. Word Count by Review Rating

F. Countries

Studying the distribution of the countries of production of Netflix movies and TV shows that can be found in Fig. 8, we found, as expected, that the United States leads with the most produced TV shows and movies. After that, the country with the most produced TV shows varies from the one with the most movies produced. United Kingdom, Japan, and South Korea lead the TV show production after the United States, with practically all the other countries having a significantly lower amount of TV shows produced. Meanwhile, on the movie production, India leads after the United States followed by the United Kingdom with approximately half the amount of movies produced by India. Like on the TV shows, the rest of the countries have significantly lower movies produced.

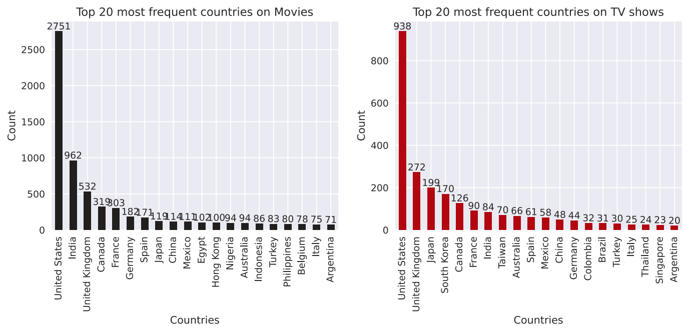

Fig. 8. Top 20 most frequent countries on Netflix titles

G. Genre

In search engines that operate through media like ours, it's also important to know how the genre of the movies and TV shows are distributed, as they can be used as keywords to search for content of a similar type. As such, we plotted the genres of TV shows and movies to analyze their distribution. From the plot in Fig. 9, we can see that dramas and comedies are the most occurring type of movies and TV shows on Netflix. All other genres have significantly lower occurrence rates save for crime, targeted for kids, romances, and documentary type of TV shows that are relatively close to the most occurring.

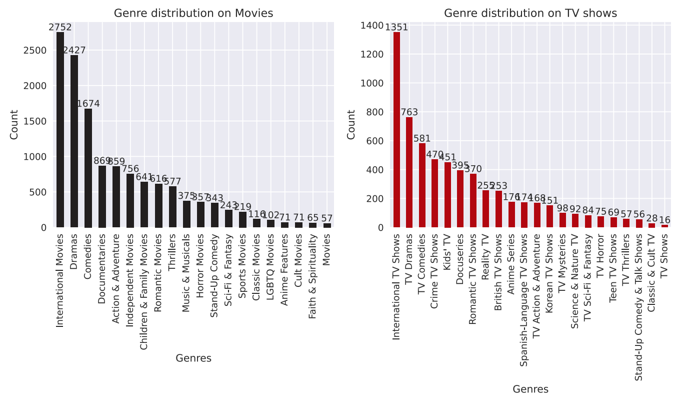

Fig. 9. Genre distribution on Netflix titles

VII. Possible Search Tasks

Given the data available, the following example queries can be made to the database originated from the combined data:

- Search for highest-rated titles of a specific actor or director
- Search for movies and/or TV shows where an actor is playing a certain role (e.g. actor X is a criminal in Tokyo)
- Search for movies that contain actors of certain nationality (e.g. Spanish actors movies)
- Search the top animation TV shows that involve a different world
- Search by keywords that aren't explicit on movie genres or description but still describe the type of movie or characters in it (e.g. searching ”Jotaro” should give me results such as ”JoJo's Bizarre Adventure”)

VIII. Information Retrieval

Information retrieval is the process of obtaining information that is relevant to an information need from a collection of resources. [4] In the context of this project, documents will be retrieved from a collection based on a provided search need that is, in this case, a text query.

A. Tool Selection

The only tools considered for this work were ElasticSearch [5] and Apache Solr [6]. Both engines are open-source and built on top of Apache Lucene. After some thorough consideration, the final decision was ElasticSearch. The research done into both these tools revealed that even though they were both popular, ElasticSearch Was leading the market at the moment [9]. It also appears to have an overall better documentation, whilst not missing any relevant features for the task at hand. To create an index and insert the data in ElasticSearch, we used its official python client, *elasticsearch-py* [7]. Alongside ElasticSearch, its companion application, Kibana, was also used to visualize and test queries and analyzers.

B. Documents

In previous sections, the data was manipulated and stored in two separate CSV files. Since ElasticSearch's preferred method of handling files is in the JSON format, all data was converted to JSON. Thus, a single JSON file was created, containing all the data ready to be indexed. The file is composed of a list of documents, each representing a single Netflix show. In a nested list of documents associated with each Netflix show, there are all the reviews associated with that specific show.

C. Indexing

A single index was created in ElasticSearch. Before inserting the data, a mapping was created to define how data should be indexed and which data should be indexed. To choose which data to index, all fields that were not relevant to the retrieval tasks are set to be not indexed by ElasticSearch. To improve the indexing performance, fields should have their type specified in the mapping file. The mapping can be found in tables I and II.

While the field *reviews* is technically a nested type, we do not specify that type on the mapping. This is because

Field	Type	Index
show_id	integer	No
type	keyword	Yes
directors	text	Yes
cast	text	Yes
countries	keyword	Yes
date_added	date	No
release_year	short	Yes
duration	short	No
rating	keyword	Yes
genres	text	Yes
description	text	Yes
avg_rating	double	No
reviews	object	-

TABLE I
Netflix Show Document Mapping

Field	Type	Index
review_id	integer	No
reviewer	keyword	No
rating	short	No
review_summary	text	Yes
review_detail	text	Yes
spoiler_tag	boolean	No
review_date	date	No
total	integer	No
helpful	integer	No
unhelpful	integer	No

TABLE II
Review Document Mapping

ElasticSearch will flatten that nested object into multiple fields to improve performance on the schema, and, since we do not have deeply nested objects, we don't require the reviews to remain nested.

After inserting all the data, ElasticSearch will create indexes that allow for search on the data. Before this step, however, analyzers, tokenizers, and token filters can be specified to instruct ElasticSearch on how to index the data. The ElasticSearch indexing process on full-text fields (fields with type *text*) is separated into three steps. In the first step, the character filters are applied. For example, the *HTML Strip* Filter that removes HTML tags or the *Mapping* Filter that replaces based on a provided map. No character filter was needed for the dataset. The second step is the Tokenizer, where the text is separated into different tokens. There can only be one Tokenizer Some examples are the *Standard* Tokenizer that splits the text according to the Unicode Text Segmentation algorithm or the *Whitespace* Tokenizer, which simply splits the text by white spaces. Being the most general-purpose Tokenizer, our choice was the *Standard* one. The third and final steps are the Token Filters, where filters are applied to the tokens returned by the Tokenizer. Some examples are the *ASCIIFolding* Filter, which converts non-ASCII characters to their ASCII counterparts if they exist, and the *Stop* Filter, which removes words from a provided list, usually common words. As for these filters, our choice was the *ASCIIFolding* Filter and the *Lowercase* Filter. The resulting analyzer is very similar to the default Standard analyzer, but with the addition of the *ASCIIFolding* Token Filter.

IX. Evaluation methodology

This section describes the method and metrics used to evaluate the search system.

A. Retrieval Methods

To understand how the indexing and retrieval process behaves and performs on different conditions, three retrieval methods were created. These can be found in table III. The first method, Retrieval Method 1 (RM1), consists of only a simple text query, for example, an actor name or a movie genre. Retrieval Method 2 (RM2), will contain a text query, but some field weights will be tweaked on a per-query basis, for example, when searching for an actor's name, the *cast* field will have a greater weight than other fields. For the final method, Retrieval Method 3 (RM3), on top of the text query and the fields weights, a term boosting to the query words will be applied. For example, when searching for a movie with the name New York City, New York can be boosted so that the common word City produces less undesired results. The boost operator used can be found at the *query_string* documentation page. [8]

	Simple text	Field Weights	Term Boosting
RM1	X		
RM2	X	X	
RM3	X	X	X

TABLE III
Retrieval Methods

B. Evaluation Metrics

To evaluate each method, we will use a variety of metrics. Due to the huge amount of documents, we need to limit the universe of evaluation to a subset of all the documents. Therefore, we'll be using the following metrics: Precision @ 10 (P@10), the Average Precision (AP), and Recall @ 10 (R@10). Precision @ 10 is used to measure what portion of the retrieved data on the subset chosen is relevant to the search query, in this case, what portion of the 10 best-ranked documents from the search query results are relevant to it. Recall @ 10 is used to measure what portion of the relevant data on the subset chosen is being returned as search results and, due to the huge amount of documents, we will consider the universe of all documents as the first 20 search results and then calculate the recall on the first 10 results. The Average Precision is used to measure the capacity of the system to correctly identify the positives, this is, to fetch the relevant data without fetching irrelevant entries.

To represent the relevant results, the letter R will be used and for the non-relevant result, the letter N will be used.

Additionally, it will be used the Normalized Discounted Cumulative Gain @ 10 (NDCG@10) to evaluate the search results of the first query whose search need highly values the order of the results, as its objective is to give the highest rated movies of a certain actor or director. Gain is defined as the relevance score, and in this measure, documents will be graded with relevance scores ranging from 0 to 5 (5 being the most relevant). Cumulative Gain is therefore the sum of gains up until a certain position on the search results. The drawback of the Cumulative Gain is the inability to measure if the most relevant results are actually at the top of the search results, and to solve this it's used the Discounted Cumulative Gain (DCG) that penalizes relevant documents appearing lower in the search results. However, this metric still has drawbacks, since the DCG score adds up with K, thus we can't compare scores between for example the top 5 results and the top 10 results (DCG@5 and DCG@10), as the latter will probably have a better score not because of better quality but because of the pure length of search results. To solve this we apply normalization by using the Ideal Discounted Cumulative Gain (IDCG) which is the DCG with the most ideal ranking, and IDCG@K being the ideal ranking up until position K. We can now obtain the Normalized Discounted Cumulative Gain @ K (NDCG@K). However, since the amount of documents is huge, we consider again only the 20 first results as our universe of search results and then calculate the NDCG@K up to 10.

X. Search Needs

This section contains 3 search needs used to benchmark and evaluate each retrieval system.

A. Search Need 1

Search Need: Highest-rated shows of a specific actor. In this case, the search is for the highest-rated shows of Adam Sandler.

Query Text: adam sandler

Field Weights: directors^3, cast^3

Term Boosts: Opposed to other queries, there aren't boosts on terms in specific, and for the third model we boost the weight that the field *avg_rating* takes in the final score, doubling the boost. The final score in this query is calculated using the query score *_score* and multiplying it by $1 + weight * \frac{avg\_rating}{10}$. On the first and second models, it was used weight equal to 1, and on the third, it was used weight equal to 2.

Results: Found in tables IV and V. The Precision-Recall curve can be found in Fig. 10 and the NDCG study in Fig. 11.

Method	Result	P@10	R@10	AP
RM1	R R R R R R R R N R R	0.9	0.8181	0.9765
RM2	R N N N N N R R R R R	0.6	0.5454	0.5696
RM3	R R R R R N N R R R	0.8	0.6667	0.9160

TABLE IV
Search Need 1 - Rank-unaware results

In this query, there's a difficulty in labeling what results are relevant or not. Movies and TV shows that do not have Adam Sandler are not relevant to our query, but would the lowest rated movie that featured Adam Sandler be relevant to our query? To solve this, it was decided to only consider the first 12 out of the 20 best rated as relevant (60%) when calculating

Method	Result	NDCG@10
RM1	5 5 5 4 4 4 4 0 3 3	0.8887
RM2	5 0 1 2 1 4 5 5 4 4	0.6463
RM3	5 5 5 3 4 2 1 4 4 3	0.8384

TABLE V
SEARCH NEED 1 - RANK-AWARE RESULTS

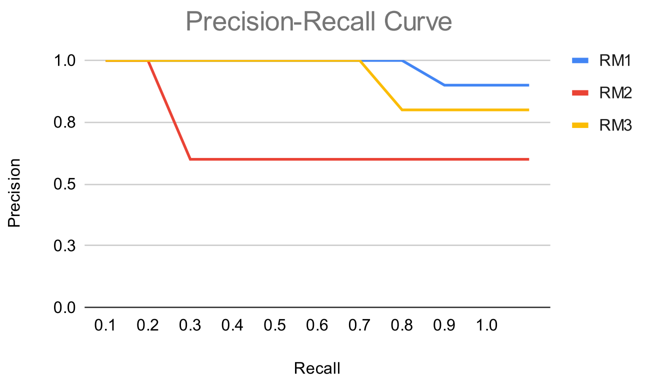

Fig. 10. Precision-Recall Curve of Search Need 1

the precision, recall, and average precision. However, while calculating the normalized discounted cumulative gain the grades will be distributed on the first 20 results evenly, this is, the 4 best-rated movies and TV shows will have a grade of 5, the next 4 with a grade of 4, and so on. The bottom 4 will have a grade of 1, and all the other movies or TV shows have a grade of 0, even if they feature Adam Sandler, as it is outside our first 20 results that we are using as our universe of search results.

On the first retrieval method, all fields had the same weight to the relevance of a document and resulted in the best results overall. Surprisingly, the query with more weight on the *cast* and *directors* fields performed the worst, obtaining results that indeed had Adam Sandler but messing up the ranking order by *average rating* as it was putting too much relevance on the cast and directors overshadowing all the other fields. This was later verified with Retrieval Method 3, that improved its results to a similar level to that of the unweighted Retrieval Method by increasing the weight of the *average rating* on the relevance of the document.

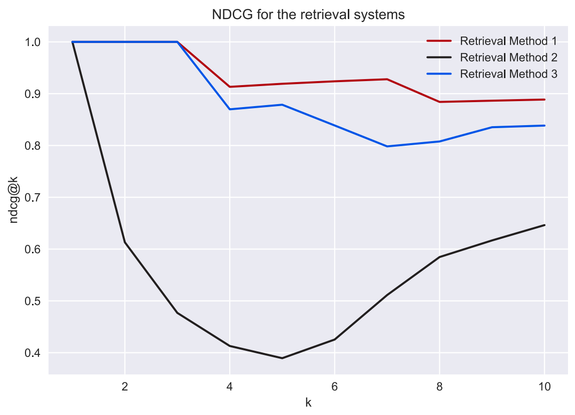

Fig. 11. NDCG@K at 1<K<10 of Search Need 1

B. Search Need 2

Search Need: Search for cartoon shows that target an adult demographic or can also be enjoyed by one

Query Text: adult cartoons

Field Weights: review_summaryˆ5

Term Boosts: cartoonˆ2

Results: Found in table VI. The Precision-Recall curve can be found in Fig. 12.

Method	Result	P@10	R@10	AP
RM1	R R R R R N R N R N	0.70	0.78	0.95
RM2	R R R R R N R N R N	0.70	0.78	0.95
RM3	R R R R N N R R N N	0.60	0.86	0.91

TABLE VI
SEARCH NEED 2 RESULTS

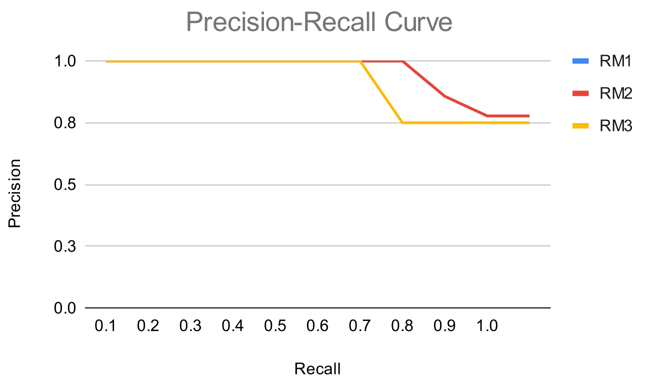

Fig. 12. Precision-Recall Curve of Search Need 2

This Search Need yielded the best results but was also the simplest of the three. It performed well across all the Retrieval Methods.

C. Search Need 3

Search Need: Documentaries about the ocean not narrated by David Attenborough

Query Text: documentary ocean NOT (David Attenborough)

Field Weights: descriptionˆ5, review_detailˆ5, genresˆ5

Term Boosts: documentaryˆ2, oceanˆ5

Results: Found in table VII. The Precision-Recall curve can be found in Fig. 13.

This Search Need had some unexpected results. The initial assumption was that RM2 and RM3 were going to be better than RM1. In this query, this didn't happen since the best was only RM2. While the cause of the overall poor results

Method	Result	P@10	R@10	AP
RM1	R N N N N N N R N N R	0.30	0.38	0.53
RM2	R R R R N R N N N N	0.50	0.83	0.97
RM3	R R N N N N N N R N	0.30	0.5	0.78

TABLE VII
SEARCH NEED 3 RESULTS

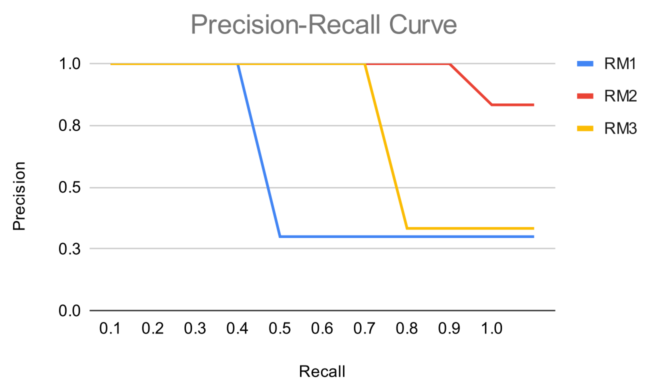

Fig. 13. Precision-Recall Curve of Search Need 3

of the query could not be found, it can be speculated that the bad results were due to a lot of documents in the dataset being similar to the Search Need but weren't quite relevant. For example, a documentary about the life of someone who lost an arm to a shark attack got considered not relevant to the Search Need since it wasn't a documentary about the ocean. Some movies such as ocean-themed cartoons also appeared in the results. Especially in Search Need 1, a bunch of relevant results appeared just outside the top 10 document limit of the results, lowering the recall as a result.

D. Overall evaluation

To compare the different Retrieval Methods, the Mean Average Precision of each Retrieval Method can be calculated. This metric provides a good way to compare the Retrieval Methods over different queries. The values of the Mean Average Precision can be found in table VIII.

	RM1	RM2	RM3
Mean AP	0.82	0.83	0.87

TABLE VIII
MEAN AVERAGE PRECISION

Some obtained results were unexpected. The first assumption was that field weights and term boosting were going to improve all queries. That was proven to not be correct since in the Search Need 1 and Search Need 2 these results proved to be worse than the simple text query.

In spite of that, the results were overall satisfactory, with all the Mean Average Precisions being around 0.75. This meant that overall, the Retrieval Methods performed evenly. An objective for future work should be improving these scores by studying how to properly weight fields to match a specific search need and add other filters to the query text like fuzziness to have a tolerance to spelling mistakes and possible synonyms for words.

XI. IMPROVING THE SYSTEM

As seen in the previous sections, a good search system was achieved. In the following sections, different approaches will be tried in an attempt to further improve the system and add new functionality. In a first stage, language detection will be used to improve multilanguage querying and at a later stage, synonyms will also be tried and evaluated.

XII. LANGUAGE IDENTIFICATION

The first approach tried was to use Elastic's built-in language identifier to identify the language in which every review was written. Most of this process is based on a blog post by Elastic, which can be found at [10].

A. Ingest Pipelines

To perform Language Identification, the text must be analyzed by a machine learning algorithm before it is indexed. For that purpose, Elastic's Ingest Pipelines are used. These mechanisms allow various preprocessing functions to be run on the data before it is indexed. Some examples are: creating new fields based on conditions, deleting fields, or even modifying fields. These functions are called Processors and are run sequentially. The pipeline runs once for each document added to the index.

The first step is to iterate over each review of each show added. To do this, the foreach processor was used and as the name implies it just iterates over a list and runs a specific processor for each object or field found. However, this processor is very limited because only a single processor can be run for each list entry. Since we need more than one processor, we had to circumvent the problem by using a nested pipeline. This allowed us to use as many processors as we needed inside the foreach processor.

The second step is to run Elastic's language identifier. For this, the inference processor was used. This processor could also be used for classification and regression tasks if needed. The model used, lang_ident_model_1, can identify about 109 different languages. We chose to only have it output the 3 most likely results.

Finally, two new fields are created. The first one is the language field, which contains the most likely language of the review in a two character code. The second field is a duplicate of the review_detail field, and its name is the same value as the language field.

B. Indexing

On the referenced blog post, two approaches are presented for language identification and storage of that information, per-field and per-index. Our choice was the per-field approach, as it is the one that can be integrated into the work already done more easily. It has, however, some drawbacks, such as duplicating fields for each language that is used. The other approach, per-index, involves creating a new index for each language that the system supports, removing the duplication of data.

After the new fields are created, we are ready to make use of them. On the mapping of the documents, we can specify an analyzer for each language that requires support from our system. For example, if our system wants to support German and Spanish, the field ”de” will have the analyzer set as German and the field ”es” will have the analyzer set as Spanish. Elastic provides these analyzers out of the box.

C. System performance

Taking into account the sheer volume of our dataset (900k reviews to classify), we expect the indexing time to increase. Before using language identification, indexing all the data took about 120 seconds, representing about 13 milliseconds to index each show on average. After using language identification on the review_detail field, the total time soared to about 4400 seconds, which is about 57 minutes. That is about 420 ms per document on average. Language identification slows down indexing by a factor of 38, making it a very costly operation. In an attempt to mitigate this slowdown, the memory available to Elastic was increased from 2 GB to 8 GB but to no effect.

D. Evaluation

To evaluate the language identification, the queries run will only look for information in the specific language field. For example, if the search is in Spanish, it will be confined to the reviews' es field. To make the comparison with the default system fair, it will be limited to the review_detail field. The chosen language for the evaluation was Spanish, for a variety of reasons: there was a healthy amount of Spanish reviews in the sample, and it's similar to Portuguese (easier to understand). Over the next subsections, the Spanish method refers to searching with a Spanish query string on a field with analyzed with a Spanish analyzer and the default method refers to searching with a Spanish query string on a field analyzed with the standard_folding analyzer referenced earlier.

E. Search Need 4

Search Need: Search for horror movies

Query Text: película horror

Results: Found in table IX. The Precision-Recall curve can be found in Fig. 14.

Method	Result	P@10	R@10	AP
Spanish	R R R N N N R N N R	0.50	0.50	0.81
Default	N N R N R N N N N R	0.30	0.75	0.34

TABLE IX
SEARCH NEED 4 RESULTS

In this Search Need, the Spanish method presents better results overall than the default method

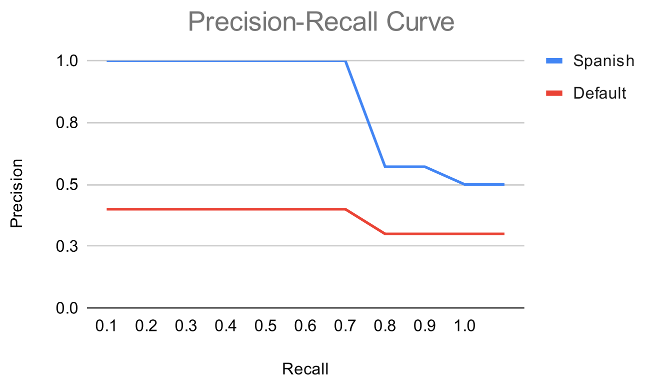

Fig. 14. Precision-Recall Curve of Search Need 4

F. Search Need 5

Search Need: Search for movies with aliens

Query Text: película extraterrestres

Results: Found in table X. The Precision-Recall curve can be found in Fig. 15.

Method	Result	P@10	R@10	AP
Spanish	R R N R R R N N N N	0.50	0.83	0.88
Default	R R N R N N N N N N	0.30	0.75	0.92

TABLE X
SEARCH NEED 5 RESULTS

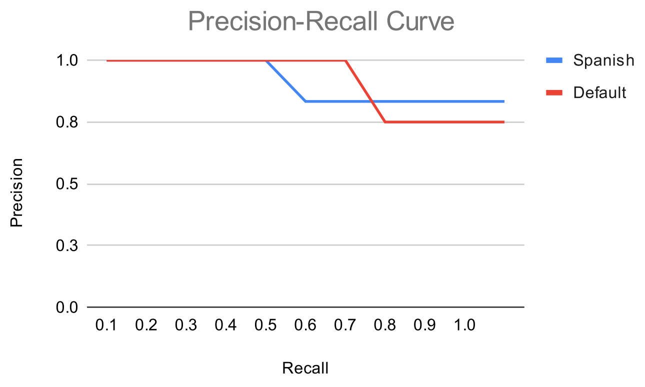

Fig. 15. Precision-Recall Curve of Search Need 5

In this Search Need, the results were similar. However, the Spanish method had a better accuracy, making it overall better for the user.

G. Average precision

The Mean Average Precision will once again be used to compare the Retrieval Methods, the Spanish one and the default one.

The Spanish retrieval system presents a Mean Average Precision vastly superior to its default counterpart for Spanish query strings.

	Spanish	Default
Mean AP	0.86	0.63

TABLE XI
MEAN AVERAGE PRECISION – LANGUAGE IDENTIFICATION

H. Overall conclusions

Language Identification presented itself as a viable solution to query string of languages different from English. Despite that, we expected better results when using the Spanish analyzer. Some probable causes for this are: There weren't as many reviews in languages other than English as we expected, making the results less far-reaching. Also, restricting the search to the review_detail fields, hampered the overall relevance of the retrieved documents.

XIII. Synonyms

As an attempt to improve the search system created, a synonym token filter was used to redirect terms to others that have similar meanings and thus give more variety and accuracy to our search results. For example, **Bollywood** is a term to refer to Hindi cinema, and thus searching for **Indian movies** or **Bollywood** should yield similar results.

Elasticsearch provides a way of handling synonyms during the analysis process. These need to be configured when defining a custom analyzer by providing it the synonym filter, mapping all the synonyms to our inverted index to be used when querying the system.

There are multiple ways of defining these synonyms, from which we explored two options: defining manually a list of synonyms and using an external tool such as Wordnet.

Defining a list of synonyms manually for each word on your database is not feasible, even if we limit to words closely related to our theme. For that reason, we initially opted to use Wordnet as Elasticsearch provides support for it. Wordnet is a large lexical database of English words, and it provides sets of cognitive synonyms (synsets), along with other concepts, providing a powerful utility tool for search engines.

After setting it up, we noticed the results given by the query were noticeably worse, obtaining practically random results rather than meaningful or related results to the query. Due to time constraints and not being able to identify the possible problem of the Wordnet setup, we discarded this option and decided to manually set up a list of synonyms.

As the synonym list is going to be defined manually, it will be restricted to some keywords related to the theme, as an extensive list is not feasible. The synonyms created mainly focus on genres of shows to map closely related words to expand the scope of our search engine, such as mapping **kids' show** to other terms such as **family friendly** or **Spanish** to **Latino**.There's also terms that refer to the same genre but are used by certain cultures. A common example being animation shows being often referred to as **cartoons** in the West but referred to as **anime** in Japan, or terms that are short versions of the original, such as **indie** when referring to **independent movies**, **sci-fi** for **science fiction** and **K-dramas** for **Korean TV shows**.

A. Evaluation

To evaluate the synonym token filter, it will be performed queries containing terms that have synonyms on our list in order to verify the impact of it on the search results. The results will be compared to the default system, this is, search without synonym token filter, to evaluate whether the synonyms used for those terms have a positive or negative impact on the relevance of the results.

B. Search Need 6

Search Need: Search for documentaries about pollution on the planet

Query Text: documentaries about pollution

Results: Found in table XII. The Precision-Recall curve can be found in Fig. 16.

Method	Result	P@10	R@10	AP
Default	N R R R R N R R N N	0.60	0.75	0.69
Synonyms	N R R N R R R R N R	0.70	0.875	0.65

TABLE XII
SEARCH NEED 6 RESULTS

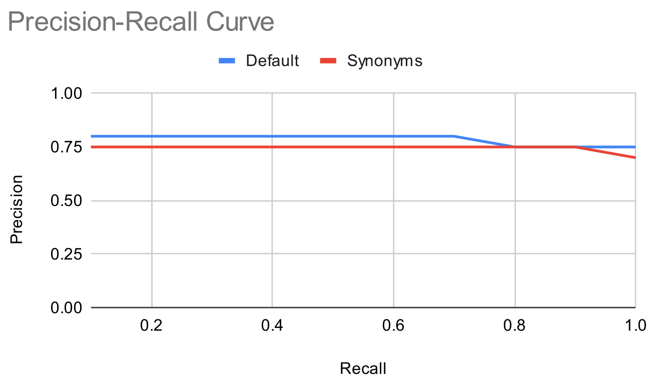

Fig. 16. Precision-Recall Curve of Search Need 6

By analyzing the search results, it can be observed that both methods were similar, favoring more the method with synonyms filter as it achieved a better precision and recall, with practically the same average precision.

C. Search Need 7

Search Need: Search for sci-fi movies with mechas

Query Text: scifi and mecha movies

Results: Found in table XIII. The Precision-Recall curve can be found in Fig. 17.

In this search need, the results obtained without the synonyms filter were better, but that was an expected result. By adding synonyms to the word **mecha** to also redirect to words like **robot**, results will be worse overall, this is because the word **mecha** can target specifically a common genre on

Method	Result	P@10	R@10	AP
Default	R R R R R R R R R R	1.00	0.90	1.00
Synonyms	R R N R N R N R N R	0.60	0.85	0.77

TABLE XIII
SEARCH NEED 7 RESULTS

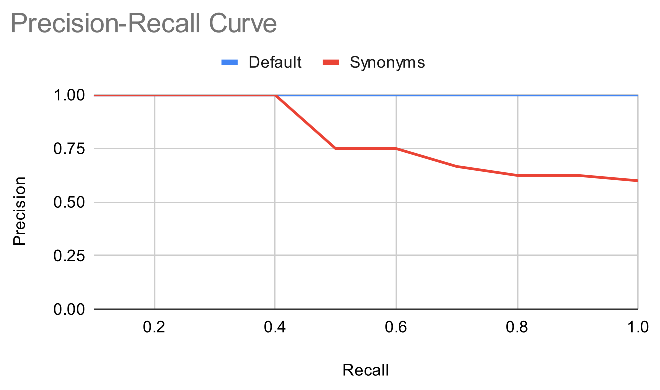

Fig. 17. Precision-Recall Curve of Search Need 7

Japanese animation shows, easily retrieving relevant results by exact match. With the synonyms filter, these results will start to include western movies with robots and may overshadow the exact match on other results.

D. Average precision

The Mean Average Precision will once again be used to compare the Retrieval Methods, the default one and the system with synonyms filter.

	Default	Synonyms
Mean AP	0.84	0.71

TABLE XIV
MEAN AVERAGE PRECISION – LANGUAGE IDENTIFICATION

While the default one shows better results on these queries, a system with better list of synonyms, such as the ones provided by Wordnet, would on show better results on queries.

E. Overall conclusions

Synonym filter presents as a possible improvement to the query results if provided with an accurate and extensive list of synonyms. However, with a limited and manually inputted synonym list, it may not be an improvement to be considered.

XIV. ELASTICSEARCH IMPRESSIONS

While ElasticSearch is a powerful tool for querying a large amount of data and creating a search engine capable of finding adequate results to the user, its steep learning curve for beginners was proven to be a challenge that slowed down the progress of the project, even when using the associated tool for data visualization, Kibana, whose interface helped to obtain direct results that were easy to be read but with the downside of having a counterintuitive interface overall for beginners. ElasticSearch's documentation proved difficult to navigate at first, but as time went on and more experience was acquired, it became a little easier.

XV. REVISIONS INTRODUCED

After the delivery of Milestone 2, some errors were detected. In Section X, all average precision values were miscalculated. The source of the error was an off by one index in the Python script used to make the calculations.

On top of that, a misunderstanding of how to calculate the recall for the precision-recall graphs meant all those were wrong as well. Before, the recall was calculated by taking into account the first 20 results. The correct way is to take into account the first 10 results. This doesn't affect the method to calculate the Recall@10 value of each search need, which is still calculated with the universe of the first 20 results.

XVI. CONCLUSION

Throughout this report, it was explained how the datasets were cleaned, processed, and analyzed, as well as the conceptualization of the database model. Through our analysis of the data, we were able to achieve a thorough understanding of the problem domain, allowing us to accurately utilize the data available to develop the search engine on later milestones. Using ElasticSearch for indexing and querying the data, it was possible to create and satisfy a set of search needs. To better understand the performance of the search engine, several metrics were used, such as precision, recall, and average precision. In the last sections, both language identification and synonyms were used to try to improve the system. Both of them had advantages and drawbacks but produced decent results. All processes were described and evaluated with the metrics used previously. All the objectives for this milestone were achieved. For possible future work, translating genres for multilanguage queries and building a search user interface, appear to be good starting points.

REFERENCES

[1] Dean, B., 2021. Netflix Subscriber and Growth Statistics: How Many People Watch Netflix in 2021? <https://backlinko.com/netflix-users#how-many-subscribers-does-netflix-have>[Accessed 11 December 2021].
[2] Bansal, S., 2021. Netflix Movies and TV Shows. [online] Kaggle.com. Available at: <https://www.kaggle.com/shivamb/netflix-shows>[Accessed 13 November 2021].
[3] Biswas, E., 2021. IMDb Largest Review Dataset. [online] Kaggle.com. Available at: <https://www.kaggle.com/ebiswas/IMDb-review-dataset>[Accessed 13 November 2021].
[4] Information Retrieval - Wikipedia. Available at: <https://en.wikipedia.org/wiki/Information_retrieval>[Accessed 11 December 2021].
[5] ElasticSearch Website. Available at: <https://www.elastic.co/elasticsearch/>[Accessed 11 December 2021].
[6] Apache Solr Website. Available at: <https://solr.apache.org/>[Accessed 11 December 2021].
[7] elasticsearch-py Website. Available at: <https://elasticsearch-py.readthedocs.io/en/v7.16.0/>[Accessed 11 December 2021].
[8] Query String Term Boosting. Available at: <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html#_boosting>[Accessed 12 December 2021].
[9] Solr vs. ElasticSearch. Available at: <https://dattell.com/data-architecture-blog/solr-vs-elasticsearch/>[Accessed 14 December 2021].

[10] Elastic Blog - Language Identification. Available at: <https://www.elastic.co/blog/multilingual-search-using-language-identification-in-elasticsearch>[Accessed 15 January 2022].

[11] Wordnet. Available at: <https://wordnet.princeton.edu/>[Accessed 17 January 2022].