Retrieving and Processing Information on Company Reviews

Bruno Rosendo
Faculty of Engineering of the University of Porto, Portugal
up201906334@fe.up.pt

João Mesquita
Faculty of Engineering of the University of Porto, Portugal
up201906682@fe.up.pt

Rui Alves
Faculty of Engineering of the University of Porto, Portugal
up201905853@fe.up.pt

ABSTRACT

Nowadays, enormous amounts of data are produced on a daily basis, leading to the rise of new topics such as *Big Data*. However, this seemingly exponential growth also brings great challenges associated with it. The purpose of this paper is to develop a functional search system that provides information about companies and how they are perceived by their workers, allowing users to find their dream job while ensuring the efficiency and relevance of the system results. To accomplish that, the work is split into 3 stages: **Data Preparation**, which includes a characterization of the dataset, followed by its cleaning and processing, **Information Retrieval**, going through the indexing, retrieval, and evaluation steps, and **System Improvement** focused in enhancing the product for the end user.

KEYWORDS

data, datasets, company, companies, information, retrieval, processing, analysis, solr, search, Indeed, ratings, indexing, query, boost, evaluation

1 INTRODUCTION

The present work is the accumulation of the three stages of a project whose goal is to develop a search system, powered by a rich and structured data collection.

The group chose to work with a dataset about company reviews, taken from Indeed, which was freely available on Kaggle [15]. The motivation is to develop a search system capable of providing information on a considerable number of companies and customized searches according to the user needs, such as searches by rating, happiness parameters or CEO approval.

The paper starts with the **Data Preparation** stage, where there's a description of the original dataset, followed by data characterization, the used processing pipeline, and a conceptual model representing the final collection of this part. Subsequently, onto the **Information Retrieval** stage, it starts with the documents definition, the indexing process and the details of the schema. The retrieving process is then explained, including *boosts* and other query modifiers, followed by some information needs. Afterwards, the evaluation process exposes the acquired results, precision metrics, and the conclusions drawn. Lastly, the **Search System Improvements** stage exposes the enhancements made to the retrieval system, such as faceted search and a better list of synonyms, and the developed user interface resulting in a more friendly use of the system. The paper ends with a section for conclusions and future work.

2 DATASET

The original dataset is composed of a large list of companies, accompanied by relevant information for people interested in working for them, including reviews from their employees, salary, and guidance on the interview process. This proved to be a very rich dataset with enough information for the project's goal, so there was no need to complement it with any additional data.

2.1 Company Reviews

The dataset contains more than 17 thousand companies (around 25MB), distributed across a large number of job fields and mostly located in the United States. The provided information was all taken from the Indeed website, a widely used job platform.

Each company review has around 20 fields characterizing them which can be arranged into three main groups:

Information about the company: Besides categorical fields like name and industry, we have textual fields, including a description of the company, the location of its headquarters, a revenue range, and the number of its employees. Some of these don't have the most appropriate format and would ideally be numerical fields.

Reviews: Each company has a given number of reviews associated, which are summarized by a set of textual fields. These include salary per role and ratings on different categories, like locations or CEO approval, as well as a general average rating. Most of them are stored as serialized documents with a great variety of categories, meaning that some data processing and exploration are needed in order to properly structure and understand these ratings.

Interviews: The companies also have reviews on their interview process. Here, we have two categorical fields exposing the difficulty and experience level of the interview, as well as a textual field describing its duration, which would ideally be a numeric field.

With this information, we should be able to implement a pipeline capable of cleaning the dataset and processing it into a more consistent and reliable format.

2.2 Data Source

The chosen dataset was collected from Kaggle [15] and is free to use, without any kind of authorization needed. The author stated that he built the dataset by scraping reviews from the Indeed website.

There are other people who have used this dataset for their own studies, which can be seen in the related notebooks on Kaggle's page [13] and shows that good projects can be done with this data source.

2.3 Data Characterization

In order to properly understand the dataset we're working with, it's essential to go through the process of data analysis and characterization. To start, we can see in Table 1 that most of the fields have a considerable fraction of missing values, so we cannot assume their presence when working with the dataset, especially the *happiness* field. More interestingly, around 2% of the companies do not have a name identifying them, which might not be very interesting when developing a search system. It's also important to note that the dataset doesn't contain any duplicate companies, all the names are unique.

Table 1: Number and ratio of missing values on each field of the dataset.

field	missing	field	missing
name	338 (2%)	**ceo_approval**	5828 (34%)
rating	1434 (8%)	**interview_count**	5551 (33%)
reviews	1556 (9%)	**headquarters**	1817 (11%)
description	1 (0%)	**happiness**	12463 (73%)
industry	1846 (11%)	**roles**	9922 (58%)
employees	2027 (12%)	**salary**	5427 (32%)

Furthermore, we can calculate some statistical information regarding numerical fields, which is presented in Table 2. Note that some of these fields had to be previously processed in order to retrieve their numerical values (e.g. the ratings on the companies' locations).

Table 2: Statistics for the dataset's numerical fields.

field	mean	std	min	max	median
rating (0-5)	3.52	0.61	1.0	5.0	4.0
ceo_approval (%)	69.98	14.71	6.0	100.0	72.0
happiness (%)	61.91	9. 43	28.0	97.0	62.0
salary ($/hour)	12.43	5.72	1.77	83.47	11.71
locations (0-5)	3.96	0.62	1.52	5.0	4.0
ratings (0-5)	3.29	0.54	0.0	5.0	3.28

Even though the individual statistics of each field are useful, it's an even better idea to look at how they correlate with each other. [1] For starters, let's see two bar charts of the industries with the best and worse ratings in Figure 1 and Figure 2.

It's also interesting to see what the rating distribution looks like in the histogram of Figure 3. It bears a resemblance to a normal distribution around the 3.5 value!

Besides general rating, it's possible to analyze additional ratings regarding the companies, in Figure 4.

One of the most important affairs of the dataset is understanding how the ratings are affected by the information of the companies. It's very interesting to see how well the CEO approval correlates with the respective company's rating in Figure 5's box plot. On the other hand, the revenue has almost no effect on the ratings, as we can see in Figure 6. Finally, we can use a heatmap to visualize how some of the rating factors correlate with each other, in Figure 7.

[1] All the plots created for this work are available in Google Drive, including some not present in the report.

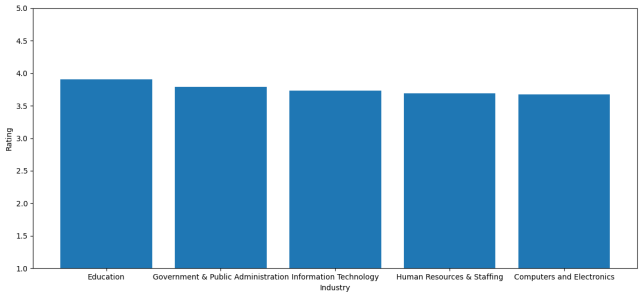

Figure 1: Industries with the best ratings.

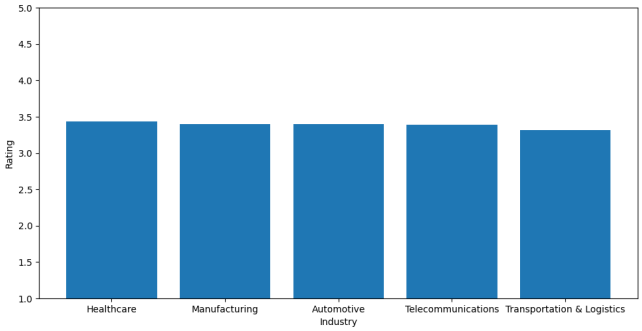

Figure 2: Industries with the worst ratings.

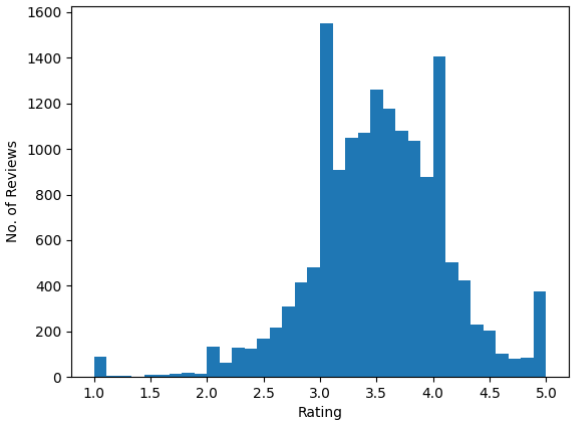

Figure 3: Distribution of the company ratings.

3 DATA PROCESSING PIPELINE

In order to obtain a ready-to-use dataset with relevant information and a well-defined structure, it is necessary to go through various steps that clean and format the original data, some of which depend on each other and should be executed subsequently. For this reason, it's important to define a pipeline where we can easily

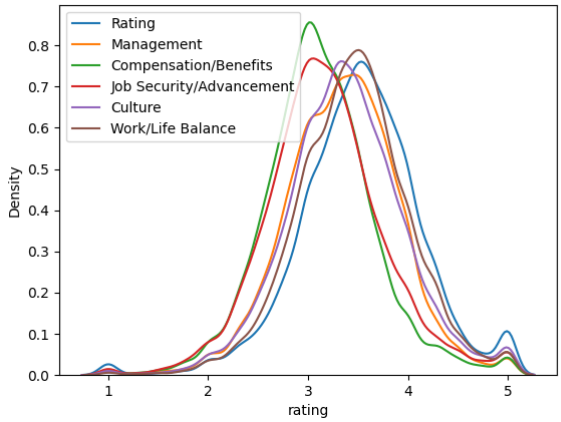

Figure 4: Distribution of different ratings about the company.

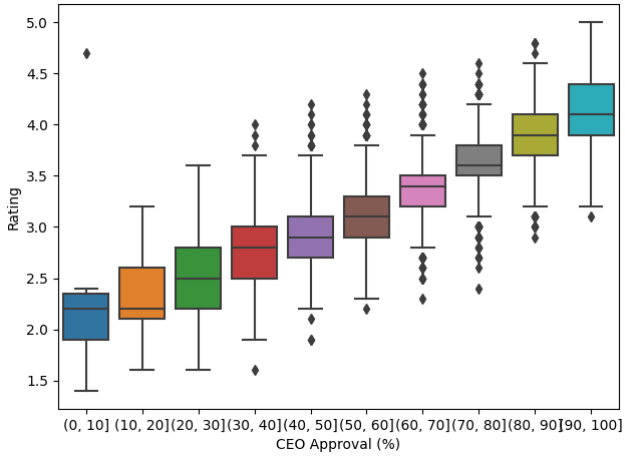

Figure 5: Correlation between the company ratings and the respective CEO approval.

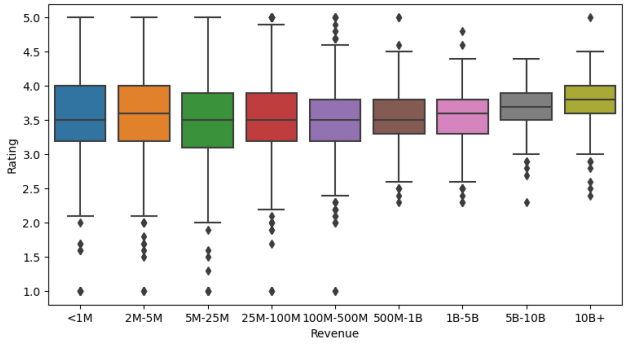

Figure 6: Correlation between the company ratings and the respective revenue.

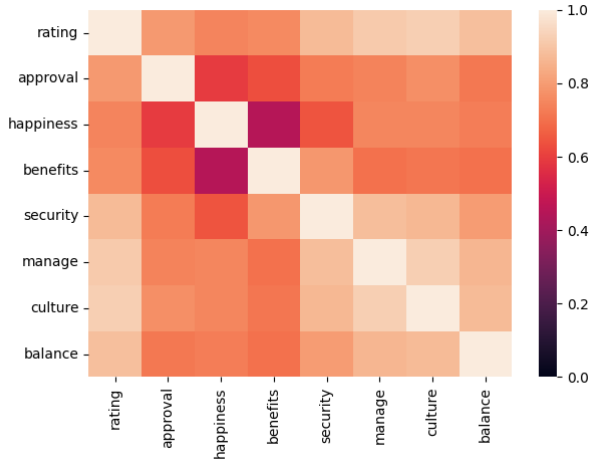

Figure 7: Correlation between the different rating factors.

show and declare all the steps needed for the construction of the final dataset.

In Figure 8, we can see what the data pipeline looks like for this project. Each step is explained in the following sections.

3.1 Data Collection and Cleaning

As seen at the start of the pipeline and explained in Section 2, the dataset taken from Kaggle [15] was enough for the objectives of this work. Therefore, that is the only data source given as the pipeline's input.

After loading the dataset, the first step is to clean the data, which can be divided into incremental small tasks. First of all, the rows containing unidentified companies are removed. By unidentified, we mean companies missing the *name* field. At this time, the useless column *website* is also removed, since it only provides information about which websites the company possesses without giving any URLs.

After removing unwanted data, the next task is regarding the company's different ratings, given by the fields: *ratings*, *happiness*, *locations* and *roles*. As mentioned in Section 2, they are stored as documents whose values represent the rating of a given category, role or location. Although they resemble a JSON format, their keys are wrapped with single quotes which are invalid JSON syntax. Additionally, all their values contain numerical information stored as strings (text). Therefore, all of these fields are fixed, by replacing the single quotes with double quotes and converting the values to their respective types (either floating point or integer).

3.2 Data Processing

The next tasks of the pipeline involve processing the data. Contrary to data cleaning, they don't simply remove or quickly fix irrelevant or wrongly formatted data. Instead, they transform the data into a more structured and reliable format.

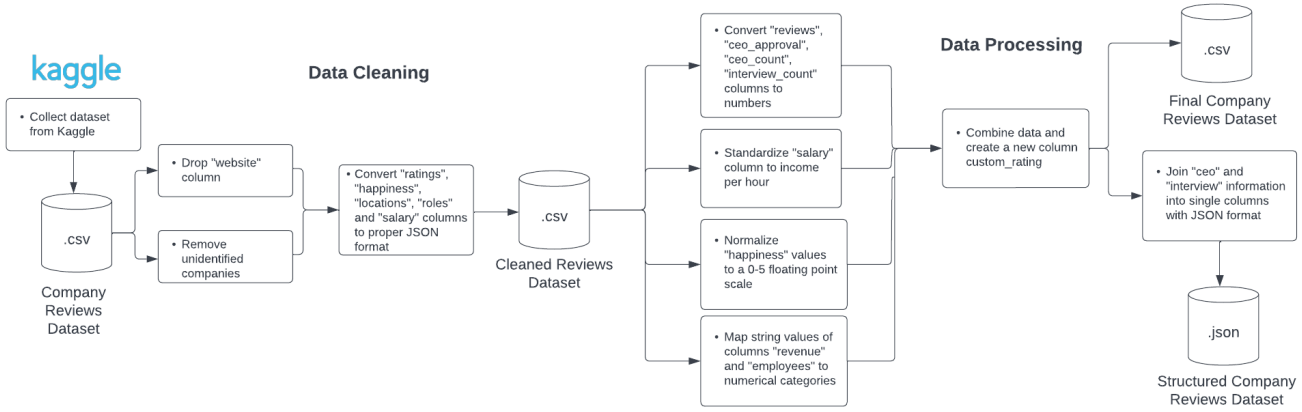

Figure 8: Data Processing Pipeline.

One of the first processing tasks is to transform the fields containing a numerical range that is described in a textual manner. The *revenue*, a monetary range indicating the revenue of the company in US dollars, and the *employees*, a range of the number of employees in the company, are the fields falling into this category. In these cases, the textual fields are replaced with categorical values, discriminating the range in which the company is described. For a more concrete example, the *revenue* field can be categorized by companies with less than a million USD, between 1 and 5 million, and so on, ending with a category for a revenue of more than 10 billion dollars.

Another important step is to fix some incorrectly typed fields. In the original dataset, the *reviews*, *ceo_count*, *ceo_approval* and *interview_count* fields hold numerical information. However, they are described in textual fields, so it's helpful to parse and convert them into simple numerical fields.

The *salary* field undergoes a similar treatment. The difference here is that the salary values (which are specified by role) are given by different time frames (i.e. salary per hour, month, year, etc.). To have a consistent and comparable salary rate, they're all converted into *dollars per hour*. In the case of the *happiness* field, a document of ratings related to personal topics, its percentual values are converted to a 0-5 score, similar to the other ratings.

After reformatting all of these fields, the pipeline creates a new field called *custom_rating*, a heuristic that tries to summarize the overall company rating, calculated with a weighted mean of all the other different ratings. The result of this step is then saved as a new CSV file.

Finally, the current CSV dataset is converted and saved in the JSON format, a document-based and well-structured format that is easy for humans to read and, more importantly, easy for machines to parse and generate. To improve the structure, the fields related to CEO and interview information are joined into their respective documents [2].

[2]The processed JSON dataset can be consulted in Google Drive.

3.3 Final Dataset

The result of the data pipeline is a JSON dataset stored in a *.json* file. It contains many different documents, each representing a company, all of its information and respective reviews and interview information. In order to visualize how these entities are related, the conceptual model in Figure 9 was built.

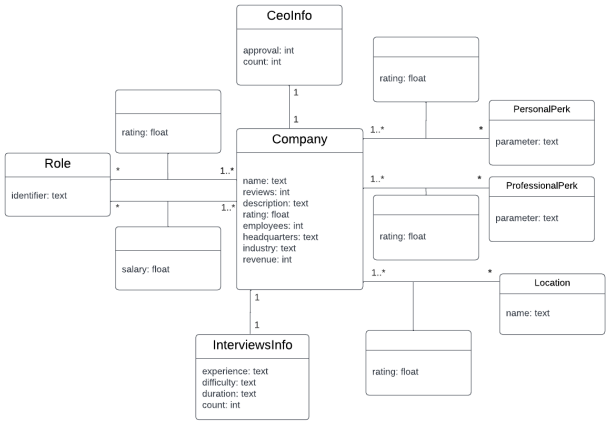

Figure 9: Conceptual Model of the Final Dataset.

As shown in the figure, it is composed of the following entities:

- **Company**: a company has a few informative fields about itself, such as name, industry and revenue. Additionally, this entity contains its number of reviews and the average rating.
- **Role**: each role is associated with one or more companies and a given company can have as many roles as needed. Each of these relations has a rating and salary associated with it.
- **Location**: location of the company's different offices, associated with the respective rating. A location can be shared across different companies.

- **PersonalPerk**: a personal perk is a characteristic of the company that might increase or decrease the life quality of their staff, such as *Purpose* or *Flexibility*. Each relation between a company and a personal perk has an associated rating.
- **ProfessionalPerk**: a professional perk is a characteristic of the company that impacts the professional career of the staff, such as *Compensation* or *Job Security*. Each relation between a company and a professional perk has an associated rating.
- **InterviewsInfo**: information about the company's interviews, including the difficulty, expected experience and duration.
- **CeoInfo**: information about the approval of the company's CEO based on staff reviews.

4 COLLECTION & INDEXING

At the end of the data processing phase, the company reviews dataset was processed into a single JSON file, with 16712 entries. All the documents were placed into a single core, from which all the information is retrieved.

The tool selected to perform the information retrieval task was Solr [6]. Although there were alternatives with similar functionalities, such as ElasticSearch or Lucene, this software proved to be capable of handling these use cases and the group already has knowledge of this tool. Solr has features which are very useful for this project, like full-text search and real-time indexing.

4.1 Document Definition

The documents stored in Solr [6] have a *key-value format*, which directly matches the JSON dataset obtained in the processing phase. This dataset is composed solely of a collection of documents (company reviews), so all of them were imported into Solr in a single collection (core). Most of their attributes have their type statically defined in a schema. In the case of nested objects, such as the *happiness* and *locations* attributes, it was necessary to declare dynamic fields to support their existence in the collection, since they contain consistent attributes (same type) with unpredictable names.

4.2 Indexing Process

The first step of this stage was to outline which fields [1] should not be indexed. The *indexed* attribute is used to define which fields can be used in queries to return matching documents. Hence, it was decided that the *reviews count*, *ceo count*, and *interview count* fields do not require indexing, since they do not represent information useful for the user to query by. Solr also supports index and query analyzers that add flexibility to the search system.

4.3 Schema Definition

The collection's fields are characterized by their type and whether they are *indexed* and *stored*. Without defining a schema [3], Solr tries to infer the type of the fields it's given. However, this often results in an undesired format and makes it infeasible to utilize the many tokenizers [11] and filters [2] present in the tool, such as stemming or removal of stop words.

With that in mind, the following schema types were created, which were then assigned to the corresponding fields present in the dataset:

- **regularText**: This type was created for fields containing regular written text, usually carrying a considerable amount of words. For this purpose, the *StandardTokenizerFactory* was used to split the text into tokens, treating whitespace and punctuation as delimiters, followed by several filters. *ASCIIFoldingFilterFactory* and *LowerCaseFilterFactory* were used to transform the text into its ASCII and lowercase equivalent. *PorterStemFilterFactory* was then used to convert them into tokens independent of conjugation and affixes. To improve the versatility of the search system, *StopFilterFactory* and *SynonymGraphFilterFactory* were also added, thus removing English stop words and supporting synonyms. Finally, *RemoveDuplicatesTokenFilterFactory* was used to remove possible redundancy introduced in previous filters.
- **categoricText**: This type was created for textual fields representing categories, such as names of entities or genres. For this reason, besides *StandardTokenizerFactory*, it only uses the *ASCIIFoldingFilterFactory* and *LowerCaseFilterFactory* filters. Additional processing would not make sense when handling specific categories (e.g. Netflix should be distinct from Netchex).
- **ratingValue** and **percentage**: These two types are both simply specified as *doubles* and are used for all the different ratings and percentages present in the dataset.
- **countable** and **enumerable**: These two types are both specified as *integers* but they hold different meanings. Fields whose values are whole numbers with real meaning (e.g. number of ratings) are classified as countable, while enumerables are fields holding categorical information in the form of integers.

The schema fields are specified according to their type in Table 3. Note that some of them are dynamic fields, given the unpredictability of their subfields (nested objects). In order to configure Solr to use this schema [3], it was written in the JSON format and uploaded to the system using Solr's API [3].

Finally, The *stored* attribute is used to define which fields can be retrieved by queries. Since every field from the dataset has relevant information for the user, this attribute is enabled for all of them.

5 INFORMATION RETRIEVAL

Solr offers a rich and flexible set of features for search [4] that we can take advantage of. With their help, it's possible to identify and execute several information needs useful for potential users of the system.

5.1 Retrieval Process

It is possible to perform queries through Solr's interface or directly through its API. For the sake of automation, the best choice was to use the former for the examples shown.

[3]The complete schema can be consulted on GitHub

Table 3: Schema fields and their respective types.

field	type	field	type	field	type
name	categoricText	**industry**	categoricText	**interview.experience**	regularText
rating	ratingValue	**employees**	enumerable	**interview.difficulty**	regularText
custom_rating	ratingValue	**revenue**	enumerable	**interview.duration**	regularText
reviews	countable	**ceo.count**	countable	**interview.count**	countable
description	regularText	**ceo.approval**	percentage	**happiness.***	ratingValue
headquarters	categoricText	**roles.***	ratingValue	**locations.***	ratingValue
ratings.*	ratingValue	**salary.***	ratingValue		

5.2 Boosts and Modifiers

Solr calculates the relevance of matching documents when performing queries. By using boosts, different weights can be assigned to specific fields or terms, thus controlling the relevance of documents. Besides boosting, there are other similar modifiers [10] useful for searching, which is explained below.

5.2.1 Boosting Fields and Terms. Field boosting was used to grant more importance to certain fields when performing queries. Take the first query that will be used for evaluation as an example. This query aims to fetch companies related to telecommunications. The search for "Telecommunications" in the *name*, *industry*, and *description* fields found 25 companies. However, the company with the second highest relevance wasn't even classified as belonging to the telecommunications industry, which is not ideal because the industry field should be the strongest indicator that a company belongs to that industry. Therefore, by applying a boost factor we can get a much more consistent result.

Similarly, term boosting is applied when it is necessary to give more relevance to specific terms in a search. For example, we can search for companies with "Financial Technology" while applying a boost factor to the "Technology" term, expecting to return companies more focused on technology.

5.2.2 Independent Boosts. Independent boosting can be achieved in Solr by the means of boost functions [12]. Instead of a query, these are arbitrary functions that execute after each user query and boost documents in some way. A typical example is boosting documents that are more recent than others.

5.2.3 Fuzzy Search. Fuzziness allows us to discover terms that are similar to the specified query without being an exact match, by using an edit distance algorithm. An optional parameter can be used to specify the maximum number of edits allowed (default to 2). This modifier can get similar results to the use of stemming in regular text but that filter is not used in all textual fields.

For example, if a user searches for companies working in the "Telecommunication" industry, they would get zero results since the dataset stores this industry as "Telecommunications". By using fuzziness and changing the query to "Telecommunication~", we're able to improve the expected results.

5.2.4 Wildcards. Solr also supports single or multiple-character wildcard searches within single terms. This can be especially useful in cases where stemming is not available and we still want to match terms that may vary in spelling. For example, one could search for "Organi?ation" to match company names independently of the word's spelling (American or British).

5.2.5 Range Search. Searching by range is particularly useful in the dataset used in this project, given the amount of numerical ratings present. By using this modifier, a user is able to search for ratings in a specified range of values. For example, one could search for companies with an overall rating of at least 4 and a work-life balance of at least 3.5 (using the maximum value of 5).

5.2.6 Proximity Search. Proximity searches can be used to look for terms that are within a specific distance from one another, specified in a maximum number of words between them. For example, one could use the query "Financial Technology"~10, on the description field to search for a broader set of companies with work related to this industry.

5.2.7 Phrase Matching w/ Slop. Slop refers to the number of positions one token needs to be moved in relation to another in order to match a phrase specified in a query. For example, a user could search for "Health and Technology" on the description field with a slop of 2. This way, the system would also match companies with "Technology and Health" in its description.

5.3 Information Needs

To have an evaluation basis for the developed system, five information needs were identified and subsequently used to calculate a few manual and objective metrics. In the current section, each information need is described and accompanied by its respective query, whose results are evaluated by analyzing the first ten companies retrieved (with the greatest score). The metrics used were average precision, precision at 10 (P@10), and recall at 10 (R@10). To calculate these metrics, it was necessary to define the 10 most relevant companies for each query and label them as the relevant results. Therefore, the metrics express not only if the results are relevant, but if they are the most relevant.

From the query parsers provided by Solr, the *Standard*, *DisMax*, and *Extended DisMax* (*EDisMax* in short) were tested. It was concluded that *EDisMax* was the best one for this use case, given its powerful improvements over the remaining parsers (e.g. powerful stop word handling and improved boost functions) [9].

To evaluate the effectiveness of the schema, boosts, and modifiers, three different configurations were tested: schemaless and no modifiers, schema and no modifiers, and schema with modifiers. This is shown in the following tabled, where 'Y' means a result is

relevant, 'N' means it is not relevant, and '-' means there were no more results in that given query and system.

5.3.1 Telecommunication companies. A user wants to find companies working in the industry of telecommunications (Table 4).

Query: Telecommunications

Table 4: Results for the telecommunications information need.

Rank / Metric	Schemaless	Schema	W/ modifiers
1	Y	Y	Y
2	Y	Y	Y
3	-	N	Y
4	-	N	N
5	-	Y	N
6	-	N	Y
7	-	N	N
8	-	N	N
9	-	Y	Y
10	-	N	N
Avg. Precision	0.49	0.59	0.71
P@10	0.20	0.40	0.50

5.3.2 Companies related with sports. A user wants to find sport-related companies. For this, he searches for companies with "Sport" in their name. (Table 5).

Query: name: Sport

Table 5: Results for the companies related with sports.

Rank / Metric	Schemaless	Schema	W/ modifiers
1	-	Y	Y
2	-	-	Y
3	-	-	Y
4	-	-	N
5	-	-	N
6	-	-	-
7	-	-	-
8	-	-	-
9	-	-	-
10	-	-	-
Avg. Precision	-	0.29	0.63
P@5	-	0.20	0.60

5.3.3 Health manufacturing companies. A user wants to find companies with work related to the health industry and manufacturing (Table 6).

Query: Health manufacturing

5.3.4 Fintech companies with many employees. A user wants to find companies working in the financial technology industry that have more than 5000 employees (categorical value of 8 or more) (Table 7).

Query: Financial Technology (employees: [8 TO *])

Table 6: Results for the health manufacturing information need.

Rank / Metric	Schemaless	Schema	W/ modifiers
1	-	Y	Y
2	-	Y	Y
3	-	Y	Y
4	-	Y	Y
5	-	Y	Y
6	-	Y	Y
7	-	N	N
8	-	N	N
9	-	Y	Y
10	-	N	N
Avg. Precision	-	0.91	0.91
P@10	0.00	0.70	0.70

Table 7: Results for the fintech with many employees information need.

Rank / Metric	Schemaless	Schema	W/ modifiers
1	Y	N	Y
2	Y	Y	N
3	N	Y	Y
4	N	Y	Y
5	N	Y	Y
6	N	Y	Y
7	N	Y	Y
8	N	N	N
9	-	N	Y
10	-	Y	N
Avg. Precision	0.49	0.65	0.76
P@10	0.20	0.70	0.70

5.3.5 Companies in the sector of Health Technology. A user wants to find companies that are involved with health technology (Table 8).

Query: "Health Technology"

Table 8: Results for the health technology information need.

Rank / Metric	Schemaless	Schema	W/ modifiers
1	N	Y	N
2	-	-	Y
3	-	-	N
4	-	-	Y
5	-	-	N
6	-	-	Y
7	-	-	N
8	-	-	Y
9	-	-	N
10	-	-	N
Avg. Precision	0.00	0.29	0.40
P@10	0.00	0.10	0.40

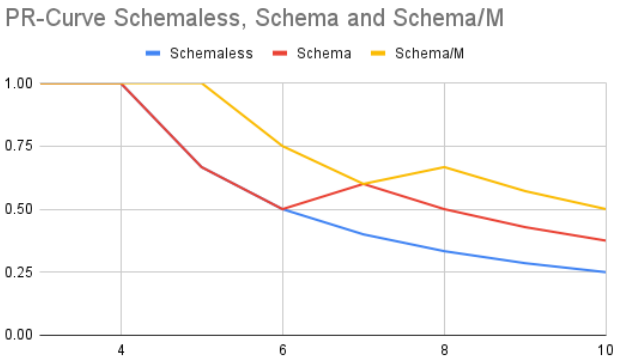

Figure 10: Precision-Recall curve of Information Retrieval 1.

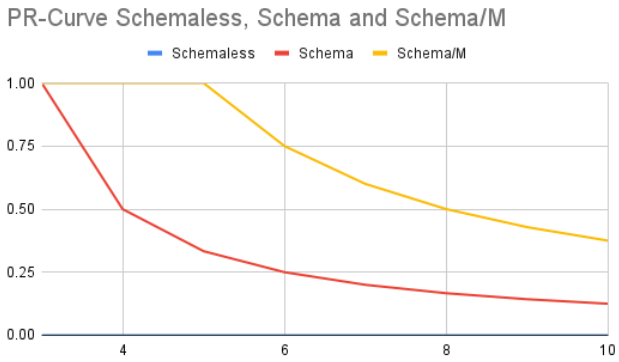

Figure 11: Precision-Recall curve of Information Retrieval 2.

6 SEARCH RESULTS EVALUATION

The performed manual evaluation presents an overview of the quality of the different configurations being tested and how they perform for each information need. Using the acquired results, it's possible to calculate the *Mean Average Precision* of each configuration (Table 8). This metric provides a measure of quality across recall levels for different queries.

Table 9: Mean Average Precision for each configuration.

Configuration	Mean Avg. Precision
Schemaless	0.196
Schema	0.546
Schema w/ Modifiers	0.682

Moreover, the Precision-Recall graphs of each query are shown in figures 10, 11, 12, 13, and 14.

Through analysis of the results, it is possible to verify that the system without schema is not very flexible since the precision of the information needs is either 1 or 0 and the recall is low. This is because the *tokenizer* and the filters are not applied to the fields. On the other hand, the other configurations are usually capable of finding more results. The system using modifiers behaves particularly well, achieving a mean average precision of 68.2%. Although some pr curves seem to be missing, such as in Figure 12, this occurs when the query doesn't return any result or when the PR curve with schema overlaps with the curve with modifiers. In these cases, they can be differentiated by the respective metrics table.

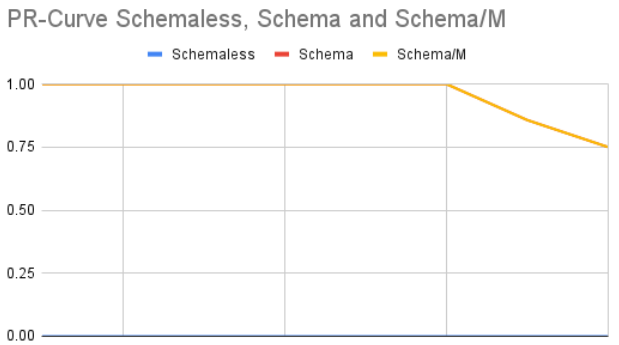

Figure 12: Precision-Recall curve of Information Retrieval 3.

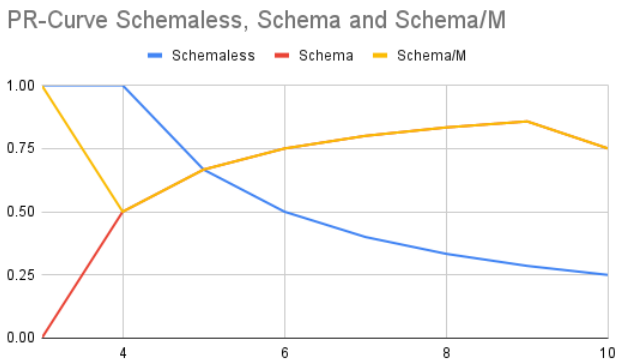

Figure 13: Precision-Recall curve of Information Retrieval 4.

7 SEARCH SYSTEM IMPROVEMENTS

The goal of this stage was to improve the search system described in the previous sections. To do so, the enhancements were grouped into the following two categories:

- Information Retrieval: enhancements made to the system described in Section 5, including an improved list of synonyms, faceted search capable of providing the user with a set of filters for certain fields, a spell-checking functionality, and the possibility of automatically suggesting words to the user as he writes queries.
- User Experience: improvements focused on giving a friendly experience to the end user. This segment explains how a user interface was developed, what was used to build it and what the final result looks like.

Other than that, the final version of this report also makes a few revisions to the previous iterations. Namely, the insertion of an

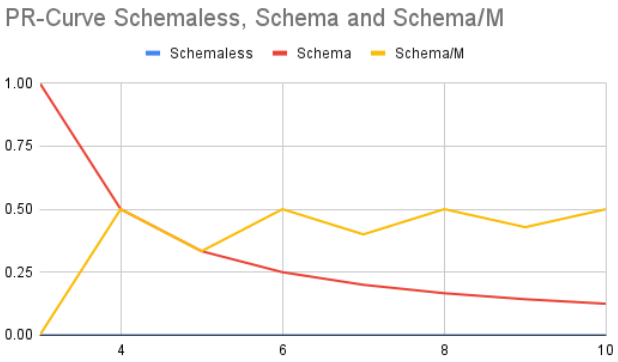

Figure 14: Precision-Recall curve of Information Retrieval 5.

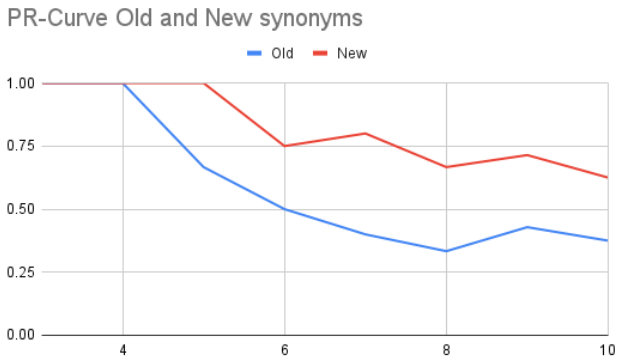

Figure 15: Precision-Recall curve between systems with different synonyms.

extra graph in Section 2, the addition of prase matching w/ slop and more explicit tables of query results in Section 5, and a recalculation of the graphs presented in Section 6.

7.1 Information Retrieval Improvements

The information retrieval tool being used, Solr, provides various components [5] meant to improve the search results seen by users, such as spell checker, suggester, faceting, *more like this*, learning to rank, result clustering and synonym support. From the ones mentioned, some of them were considered not relevant and discarded as an improvement for our system:

- More like this: this component didn't seem very interesting because our system focuses on finding relevant results using a search bar. Any result seen by the user should already be accompanied by similar ones returned by the search query, meaning that finding more like a specific result would be redundant.
- Learning to rank: this feature allows us to run machine learning models in the system, improving the rank returned by search results. However, a good model implies a large dataset of subjective ranking of search results, which simply isn't realistic for the current system with only a few users.
- Result clustering: this plugin would try to automatically discover groups of related search hits and assign them labels. This isn't very useful because, in the current system, the users already have a lot of fields to search for, so it could become overwhelming to them.

Thus, synonyms, spell checker, suggester, and faceting were the components identified as most relevant for the system and further explored in this work.

7.1.1 Synonyms. In the previous iteration of the search system, it used an already well-formatted list of synonyms that was freely available online. However, this list gave a worse performance than what was expected. For that reason, a custom list of synonyms was built using the WordNet [17] lexical database.

For example, in the previous version of the system, a query using the term "globe" would get matched with unexpected synonyms like "horn" and "pill". In the improved version, the query will only match with correct synonyms, such as "earth" and "world".

Let's test this new system with another information need where a user wants to find companies involved with panic support and management (Table 10 and Figure 15). This query previously return more than a thousand results, most of them completely irrelevant.

Query: "panic"

Table 10: Results for the new synonyms list.

Rank / Metric	Old Synonyms	New Synonyms
1	Y	Y
2	Y	Y
3	N	Y
4	N	N
5	N	Y
6	N	N
7	Y	Y
8	N	N
9	N	N
10	N	-
Avg. Precision	0.53	0.76
P@10	0.30	0.50

7.1.2 Spell Checker. The spell-checking feature of Solr allows the users to be informed of possible typos they might write in their queries. This way, they can understand what went wrong or even have their mistakes fixed automatically.

To implement this mechanism, Solr's API was used to add a search component that uses an IndexBasedSpellChecker [7] in the company's description, the field with the most text density. Afterwards, a request handler is added to expose this functionality to a particular endpoint. An example of a query to this handler is shown below:

```
[Request] /solr/reviews/spellcheck?q=tecknology
[Response] "suggestions":[
      "tecknology",{
        "numFound":1,
        "startOffset":0,
        "endOffset":10,
```

```
        "suggestion":["technology"]}
        ]
```

7.1.3 Suggester. Solr also provides a component to suggest queries based on incomplete phrases or terms. This is useful to display suggestions while the user is typing their query, identically to what happens in a Google search. In this case, a suggester was used as an auto-complete that recommends words when the user starts writing them.

The setup process is very similar to the one used for spell-checking. A new search component is added to the configuration using the FreeTextLookupFactory [8] implementation, followed by the corresponding request handler. An example of a query to this handler is shown below:

```
[Request] /solr/reviews/suggest?q=tech
[Response] "suggest":{"reviewsSuggester":{
      "tech":{
        "numFound":10,
        "suggestions":[{
            "term":"technology",
            "weight":10745044845232370,
            "payload":""},
          {
            "term":"technologies",
            "weight":2696173614301849,
            "payload":""},
            ...
          {
            "term":"technologically",
            "weight":237897671850163,
            "payload":""},
          {
            "term":"techlogix",
            "weight":118948835925081,
            "payload":""}]}
        }}
```

7.1.4 Faceted Search. Faceting is the arrangement of search results into categories based on indexed terms. This allows the user to easily filter fields by their available values, avoiding meaningless searches that would never match results. In this system, many categorical values benefit from this functionality: industry, employees, revenue, interview experience and interview difficulty.

Discriminated values, ready to be used in faceted searches, can be discovered by querying a Solr's core using the *facet* parameter set to true. There are additional parameters which help tune the search, such as *facet.field*, to choose the fields used by the faceted search, *facet.limit*, to limit the number of values returned in each field, and *facet.sort*, to choose whether the values are sorted by the number of documents or lexicographically. The following example shows how this query can be done:

```
[Request] /solr/reviews/select?q=*:*&facet=true&
facet.field=industry&facet.limit=10
[Response] "facet_counts":{
    "facet_fields":{
      "industry":[
        "healthcare",439,
        "services",281,
        "manufacturing",126,
        "education",118,
        "construction",96,
        "facilities",95,
        "retail",84,
        "wholesale",83,
        "logistics",74
        "transportation",74]},
    "facet_heatmaps":{}}
    }
```

7.2 User Experience Improvements

The final focus of this work was on enhancing the experience for the end user so that anybody could use the search system without any technical background. Therefore, a user interface (frontend) was developed using React [14] and MaterialUI [16]. Solr's API served perfectly as the backend of this interface. Some of the features implemented in this application were:

- Free text search bar, allowing users to search whatever terms they wish.
- A scrollable search results page that displays summarized information about the most relevant companies according to the chosen query.
- Specialized view for company details, accessible by clicking on the corresponding search result.
- Filters for certain fields (rating, employees, etc.). TODO: complete
- Auto-complete feature that suggests words as the user is writing in the search bar.
- Appealing design. A clean and intuitive interface is key to the user experience, giving a much more natural feeling to the ones using the system. The UI is straightforward so that users can start finding companies right away.
- An "About" page with summarized information on how the application works so that users can have support in case they have difficulties.

Some screenshots of the developed user interface can be found in Figure 16, Figure 17, and Figure 18.

8 CONCLUSIONS AND FUTURE WORK

This paper presented the process of developing a functional search system that provides information about companies and how they are perceived by their workers, with special care to ensure the efficiency and relevance of the search results.

In the first stage of the project, the used dataset was analyzed and characterized, allowing the construction of a data processing pipeline resulting in a new structured dataset. With it, the second stage of information retrieval was developed, including the indexing process with a defined schema, the retrieving process using Solr's query modifiers, and the evaluation of the search results obtained. The third stage of this work focuses on making improvements to the previous system and on enhancing the experience for the end users of the product. Some of the implemented improvements were a better list of synonyms, spell-checking, query suggestions, faceted searches, and the development of a user interface.

Figure 16: Homepage with Filters and Fuzziness.

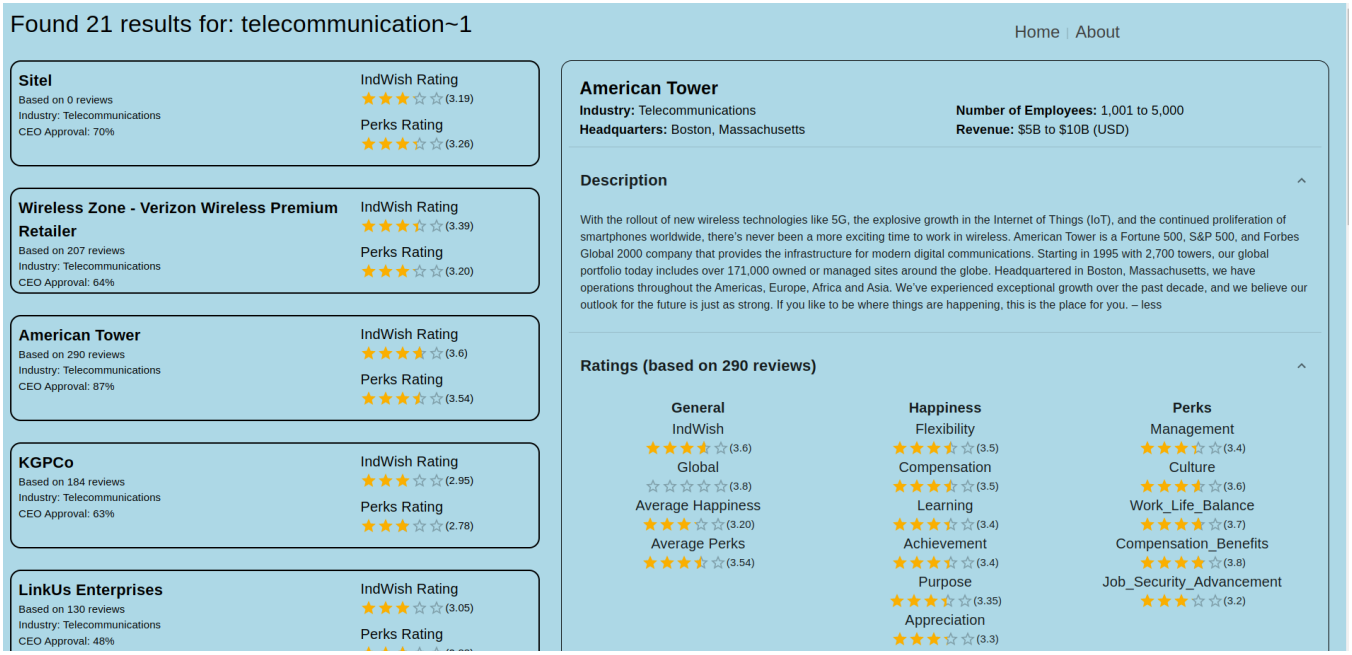

Figure 17: Search Results Page for the User Interface.

The results obtained with the use of the search system proved the effectiveness of using a defined schema and applying boosts and other modifiers to the queries applied. This resulted in a mean average precision of 68.2%, showing that the search system offers results with some quality.

The following steps for the project are the improvement of the current user interface, which is still missing features like the integration of faceted searches and spell checking. The system's backend (Solr) can also still be improved, for example, by means of machine learning models that train with the data from the system's users.

Figure 18: Homepage search suggestions.

Another possible enhancement is the creation of user accounts, allowing them to save their favourite company reviews (or review them) and have personalized search results according to their preferences.

REFERENCES

[1] Apache. 2017. Field type definitions and properties. Retrieved December 10, 2022 from https://solr.apache.org/guide/6_6/field-type-definitions-and-properties.html.

[2] Apache. 2017. Filter descriptions. Retrieved November 12, 2022 from https://solr.apache.org/guide/6_6/filter-descriptions.html.

[3] Apache. 2017. Schema api. Retrieved November 12, 2022 from https://solr.apache.org/guide/6_6/schema-api.html.

[4] Apache. 2017. Search overview. Retrieved November 12, 2022 from https://solr.apache.org/guide/6_6/overview-of-searching-in-solr.html.

[5] Apache. 2017. Search overview. Retrieved December 10, 2022 from https://solr.apache.org/guide/6_6/searching.html.

[6] Apache. 2022. Solr. Retrieved December 9, 2022 from https://solr.apache.org/.

[7] Apache. 2017. Spell checking. Retrieved December 10, 2022 from https://solr.apache.org/guide/solr/latest/query-guide/spell-checking.html.

[8] Apache. 2017. Suggester. Retrieved December 10, 2022 from https://solr.apache.org/guide/6_6/suggester.html.

[9] Apache. 2019. The extended dismax (edismax) query parser. Retrieved November 13, 2022 from https://solr.apache.org/guide/7_2/the-standard-query-parser.html.

[10] Apache. 2017. The standard query parser. Retrieved November 13, 2022 from https://solr.apache.org/guide/7_2/the-standard-query-parser.html.

[11] Apache. 2017. Tokenizers. Retrieved November 12, 2022 from https://solr.apache.org/guide/6_6/tokenizers.html.

[12] Lucidworks. 2011. How to tune document's relevance in solr. Retrieved December 11, 2022 from https://lucidworks.com/post/use-solr-boost-function/.

[13] Yael Man. 2021. Company review rating factors. Retrieved October 10, 2022 from https://www.kaggle.com/code/yaelman/company-review-rating-factors.

[14] Meta. 2022. React - a javascript library for building user interfaces. Retrieved December 11, 2022 from https://reactjs.org/.

[15] Reza. 2020. Company reviews. Retrieved November 10, 2022 from https://www.kaggle.com/datasets/vaghefi/company-reviews.

[16] Material UI SAS. 2022. Mui: the react component library you always wanted. Retrieved December 11, 2022 from https://mui.com/.

[17] WordNet. 2022. Wordnet. Retrieved December 10, 2022 from https://wordnet.princeton.edu/.