INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




# Análise de lacunas no mercado Praxis

**RUI PEDRO DA SILVA ALMEIDA**
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

# Praxis Market Gap Analysis


## Rui Pedro da Silva Almeida


**A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Computer Systems**


**Supervisor: Dr. Nuno Escudeiro**



Porto, October 13, 2020

# Abstract

Praxis is a platform that allows students to search internship offers submitted by companies, research labs and higher education institutions. The success of the platform depends on its adoption by students that use keywords to find internships and organizations that need candidates to fill their vacancies.

The two problems detected with the platform are the high number of searches without results and the low percentage of internships with candidates. This work explores text mining techniques with the objective of creating a tool to analyse the discrepancy between supply and demand, helping to identify market gaps and possible improvements to the search engine.

The final solution allows Praxis administrators to intuitively analyse the available data on a dashboard. Additionally, from the analysis made, limitations were found on the existing search engine, so an improved one was created. Software engineering best practices were followed and the defined objectives were achieved.

**Keywords:** Business Intelligence, Data Visualization, Information Retrieval, Text Mining

# Resumo

O Praxis é uma plataforma que permite a estudantes procurar propostas de estágio submetidas por empresas, centros de investigação e instituições de ensino superior. O sucesso da plataforma depende da sua adoção por parte dos estudantes, que utilizam palavras-chave para pesquisar estágios, e das organizações que necessitam de candidatos para preencher as suas vagas.

Os dois problemas detetados na plataforma são o elevado número de pesquisas sem resultados e a baixa percentagem de estágios com candidatos. Este trabalho visa explorar técnicas de processamento automático de texto com o objetivo de criar uma ferramenta para analisar a discrepância entre a oferta e a procura, ajudando a identificar lacunas no mercado e possíveis melhorias no processo de pesquisa.

A solução final permite aos administradores do Praxis visualizar os dados disponíveis num painel de controlo, de maneira intuitiva. Além disso, a partir da análise realizada, foram encontradas limitações no motor de busca existente, motivando assim a criação de um novo. Foram seguidas boas práticas de engenharia informática e atingiu-se os objetivos definidos.

# Acknowledgements

I would like to thank my dissertation supervisor, Dr. Nuno Escudeiro, for all the help given in these last months. The continuous feedback contributed significantly to the quality of the project.

Additionally, I must express my gratitude to my closest family and friends for always supporting and motivating me.

# Contents

# List of Figures

# List of Tables

# List of Code

# List of Acronyms

| | |
|---|---|
| API | Application Programming Interface. |
| BI | Business Intelligence. |
| ETL | Extract-Transform-Load. |
| HTML | Hypertext Markup Language. |
| HTTP | Hypertext Transfer Protocol. |
| IR | Information Retrieval. |
| JSON | JavaScript Object Notation. |
| JVM | Java Virtual Machine. |
| Kstem | Krovetz Stemmer. |
| LDA | Latent Dirichlet Allocation. |
| LSA | Latent Semantic Analysis. |
| NLP | Natural Language Processing. |
| OS | Operating System. |
| POS | Part of Speech. |
| SQL | Structured Query Language. |
| SSH | Secure Shell. |
| T-SNE | T-distributed Stochastic Neighbor Embedding. |
| TF-IDF | Term Frequency–Inverse Document Frequency. |
| VM | Virtual Machine. |
| YAML | YAML Ain't Markup Language. |

# Chapter 1

# Introduction

This chapter starts by providing a brief overview of the Praxis internship platform and the addressed problem. The approach used to test the hypotheses and expected results are also presented. Finally, the document structure is described with the main topics that will be covered.

## 1.1 Context

Praxis[1] is a web-based internship platform. It allows companies, higher education institutions and research labs to publish internship offers. Students can search the available offers and apply to internships.

The project was funded with support of the European Commission, through the Erasmus programme, to address the needs of the academic and professional communities. The Praxis network has over 130 partners in more than 30 different countries [1].

## 1.2 Problem

Students use the Praxis platform to search for academic internships. One of the issues detected with the platform is the number of searches without results. About half of the searches made by students do not return any internship offers. This information is currently obtained periodically through a manual process consisting of several steps.

Another issue is the number of internship offers without candidates. Praxis administrators and partners want to promote the match between supply and demand, but they do not have automated reports about the topics of internships and search keywords.

The high number of empty results and unassigned internships are important problems that need to be addressed to encourage the use of the platform by students and organizations alike.

## 1.3 Hypothesis

Automated reports and dashboards are helpful tools for decision making. A data mining solution can be built using open source or commercial technologies. This will enable the visualization of trends and identification of gaps between available proposals and frequent

[1] http://www.praxisnetwork.eu

searches done by students. Having this information, actions can be taken to decrease the market gap.

Modern search engines and algorithms may help improve search results. A common alternative to exact keyword matching is fuzzy matching, which can handle spelling mistakes. Other query expansion methods, such as finding synonyms, semantically related words and stemming are also to be considered.

In short, it is hypothesized that the use of text mining techniques will allow to identify market gaps and possible improvements to handle query inputs, like correcting spelling mistakes or replacing certain words.

## 1.4 Approach

The software solution will be designed and built incrementally. Through several iterations, new features or technologies may be considered and evaluated. Software engineering best practices are to be followed, like the use of version control and reproducible infrastructure deployment.

The state of the art on information retrieval systems is very rich, as extensive research has already been made to optimize search results. To analyse the data collected over the years on the Praxis website, a number of text mining algorithms will be tested.

One of the objectives of this project is to reuse existing technologies and libraries as much as possible. This enables a faster development speed and eases future maintenance as there is less code to manage.

## 1.5 Expected results

With the development of this project, many Praxis users will be positively impacted as searches will retrieve more results, while keeping a high precision. Praxis administrators and partners can have access to timely and detailed information about supply and demand to better understand the dynamic internships market.

The final solution consists of various software modules:

- **Data pipeline** to extract data from the operational database, transform it and load it in another database, to be used for analytical processing.
- **Search engine** responsible for processing search queries and retrieve the most relevant internships.
- **Search Application Programming Interface (API)** to provide secure access to the search engine through the Hypertext Transfer Protocol (HTTP) protocol.
- **Web dashboard** to visualize the existing data using text mining techniques and extract valuable information.

Although this project was specifically engineered for Praxis, it could be used in other domains. The same technologies and algorithms are still valid to optimize searches and analyse unstructured text data.

## 1.6 Document structure

This document is organized as follows:

**Chapter 1** presents the project, identifying its purpose and stakeholders.

**Chapter 2** is initialized by a clear description of the business context. Afterwards, the state of the art regarding text mining, information retrieval and related technologies is studied.

**Chapter 3** presents the solution analysis and design, providing the functional and non-functional requirements, as well as technical diagrams to describe the system architecture.

**Chapter 4** contains an analysis of the historical data collected on Praxis to better understand the market of academic internships.

**Chapter 5** covers the implementation details and technical decisions made to fulfil the requirements.

**Chapter 6** describes the deployment procedures and evaluates the final solution.

Lastly, **Chapter 7** provides an overview of the work done and fulfilled objectives, while showing the solution limitations and future work.

# Chapter 2

# Context and state of the art

In this chapter, the proposed problem, its surrounding context and value proposition are explored. Additionally, a study of the state of the art is made and the technologies for the envisioned solution are presented.

## 2.1 Contextualization

Education plays an important role in society, especially as demand for specialized people increases in the job market. In 2017 there were about 20 million higher education students in Europe [2]. Internships are often the first professional experience that students have. To promote innovation in the academic internship market, the European Union funded Praxis.

The Praxis web platform helps to connect students with companies, higher education institutions and research labs. Organizations are able to disseminate national or international internship proposals while students can search and apply to them.

Between January 2014 and November 2019, approximately 151 940 searches for internships were made on Praxis and from those, 61 490 had no results, representing about 40% of the total queries. In addition, 48% of the 2163 internships did not get any candidates. These issues needs to be analysed so that actions can be taken to improve search results and increase the match between supply and demand.

The current process to analyse data is very time consuming. It involves manually connecting to the machine that contains the Praxis database, running a set of Structured Query Language (SQL) queries, exporting their results to files, transferring them to a local system and importing them to Excel for analysis.

### 2.1.1 Value proposition

The development of a tool to automatically analyse searches made by students and internship proposals made by organizations is extremely valuable. It will help Praxis administrators identify market gaps and enable them to take actions in order to reduce those gaps, such as sharing this information with companies and schools.

Analysis of existing data also provides guidance on how the search engine may be optimized. While some searches correctly do not return any results, it is hypothesized that others could return similar matches instead of none, by fixing typos or replacing words with synonyms or related terms.

The proposed work will enhance the experience of all users in the Praxis platform. Students will find more internships, and consequently, organizations will have more candidates, thereby increasing the adoption of Praxis.

The elaborated business model canvas is presented in Figure 2.1 to describe the key business concepts.

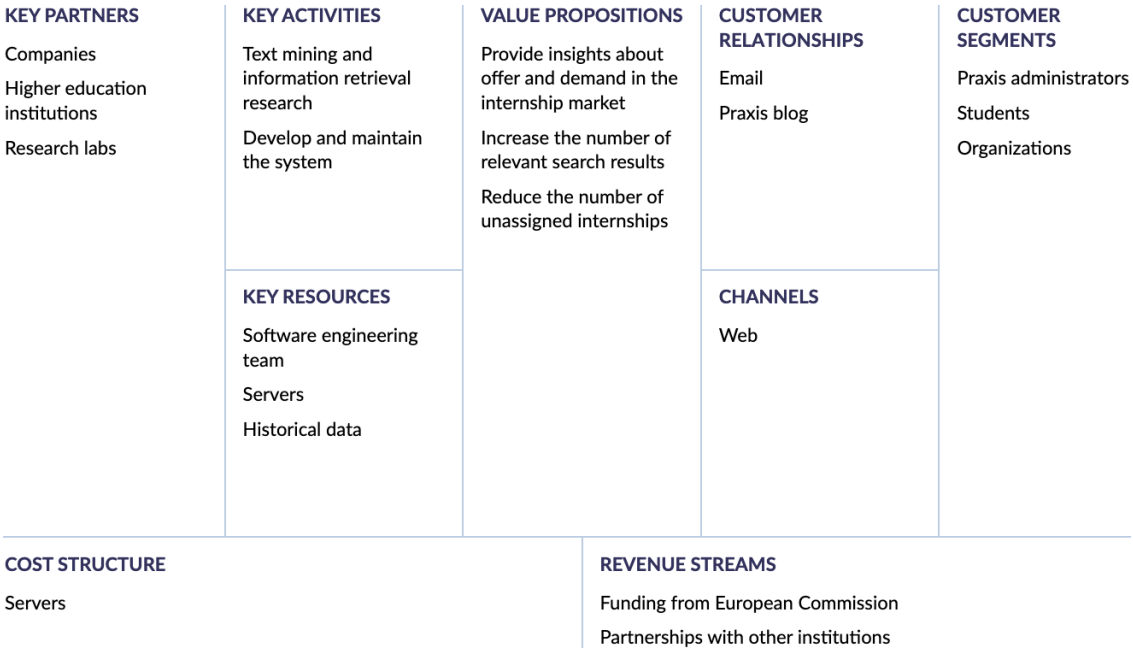


Figure 2.1: Project business model canvas

There are three customer segments that benefit from this project, Praxis administrators who want to increase the platform adoption and have effective reporting tools, students that want to find relevant offers and organizations in need of internship candidates.

Important partners include various organizations, such as companies, higher education institutions and research labs, as they are the ones that publish internships and are interested in knowing the state of the market.

Key activities involve creating the system using state of the art technology and algorithms, and then maintaining it. The most important resources are the engineering team to develop the system, servers to host the project and historical data for analysis.

The main costs come from maintaining the infrastructure since most of the work is now done by volunteers. In the past, Praxis received funding from the European Commission, but looking forward, partnerships with other institutions may be the best option to generate revenue.

### 2.1.2 Market research

The recruitment market was estimated to be worth $215.68 billion worldwide in 2017 and given the economic and technological development, this number is expected to grow to $334.28 billion by 2025 [3].

Search engines are problematic in recruitment because there are multiple different job titles for the same job [4]. This might cause delays or even missed opportunities in the recruitment process.

Although this work is designed for Praxis, it can be adapted for other companies in the recruitment market, ranging from small organizations to global enterprises like LinkedIn, Indeed or Glassdoor.

## 2.2 Text mining

Text mining is a subtopic of data mining. Data mining is the process of extracting information from data. With each different type of data, such as quantitative, categorical, text, spatial, temporal or graph-oriented, there are different techniques and algorithms that can be used to extract insights. Most data mining algorithms work with structured numeric data, so transformations must be applied to unstructured data before mining information [5]. Natural Language Processing (NLP) is a branch of artificial intelligence that helps computers to understand human language for tasks such as Part of Speech (POS) tagging, named entity recognition, sentiment analysis and others.

A text mining project includes the following phases [6]:

1. **Data extraction** - Retrieves relevant data from a database or other sources.
2. **Text preprocessing** - Makes the text more consistent with multiple techniques such as tokenization, word filtering and stemming.
3. **Text representation** - Transforms text into a format suitable for algorithms.
4. **Knowledge discovery** - To visualize data, discover patterns and create models.

### 2.2.1 Data extraction

Over 80% of business data is stored as text, for example web pages, and emails [7]. It is often not stored in the same place, but spread across multiple databases and other internal or external systems. Extract-Transform-Load (ETL) processes are used to move data from heterogeneous sources to a centralized database, while cleaning and enriching it. This can be expensive, especially when dealing with Big Data [8].

### 2.2.2 Text preprocessing

Preprocessing tasks are important as they affect text mining algorithms. The first step is tokenization, which consists on breaking text into words or phrases, called tokens, removing punctuation in the process. Lowercase transformations and accent removal are common normalization methods. Then filtering may also be helpful to remove stop words, which are words that appear too often, as well as removing words that occur too rarely, as they are usually not very relevant [9].

Another common preprocessing technique is stemming, which is the process of converting words to their stem, or root form, for example *"argued"* could become *"argu"*, depending on the stemmer aggressiveness. These algorithms can be language dependant and mostly truncate words based on rules so there are some inaccuracies. Lemmatization is an alternative, with higher computational costs, that results in valid words. It depends on dictionary

lookups and POS tagging to correctly identify the meaning of a word in a sentence, as different lemmas may be returned if a word is a verb or a noun [10].

### 2.2.3 Text representation

In text mining, a collection of documents is named "*corpus*". Each single document contains a set of words, often referred as terms. A document-term matrix is commonly used to represent a *corpus*, where each row corresponds to a document and each column, or feature, is a term. When using such a representation, the order of a sequence of words in a document is lost [5]. To capture semantic relationship between words, n-grams can be tokenized, instead of single words. A n-gram is a consecutive sequence of $n$ words.

A document-term matrix is sparse and its values may contain a one-hot encoding, word count or Term Frequency–Inverse Document Frequency (TF-IDF). Term frequency measures how often a word appears in a document and inverse document frequency measures how important a term is in the corpus. Other term-weighting schemes exist, but TF-IDF variants are the most common [11]. The weight $w_{i,j}$ of a term $i$ in a document $j$ can be calculated with the formula

$$w_{i,j} = tf_{i,j} \times \log \frac{N}{df_i} \tag{2.1}$$

where $tf_{i,j}$ represents the term frequency, $N$ is the corpus size and $df_i$ is equal to the number of documents containing the term.

A recent alternative to bag-of-words models, which keep word counts but disregard order, are word embeddings that can be obtained using unsupervised machine learning algorithms such as *word2vec* and *GloVe*. These algorithms are trained on huge corpus and are able to capture semantic similarity by representing similar words close to each other in a dense vector space. Cosine similarity between two word vectors can be used to retrieve nearest neighbours. Vector arithmetic can be used to explore word analogies, for example *"queen" - "woman" + "man"* should produce a close vector representation of *"king"* [12]. BERT is a state of the art technique to produce context sensitive word embeddings, so a word may have a different representation on different sentences [13].

### 2.2.4 Knowledge discovery

After transforming the text corpus to a numeric representation, visualization techniques may provide insights, and existing machine learning or data mining methods like classification or clustering can be used [6]. Results should be evaluated and additional iterations to the text mining process may be needed.

Two common approaches to visualize text data are word clouds and co-occurrence networks [14]. In word clouds, words are arranged in varying size, color, and position, based on their frequency or importance. Co-occurrence networks are used to show the relationship between words or documents.

Classification is a supervised machine learning problem where labelled data is given to an algorithm and it learns to label new observations based on training datasets. Creating new training datasets can be time consuming as it may require expert knowledge and the sample size should be large enough to achieve good performance [15].

Clustering is an unsupervised problem that involves grouping similar data points based on their similarity. Common algorithms include hierarchical clustering [16] or k-means [17]. Determining the ideal number of clusters depends on cohesion metrics or domain knowledge.

Topic modelling is a text mining technique to analyse collections of documents by discovering topics. A number of algorithms exist, such as Latent Dirichlet Allocation (LDA) and Latent Semantic Analysis (LSA). These techniques have been applied to massive collections of documents and even other kinds of data, like genetic data, images and social networks [18].

## 2.3 Information retrieval

Information Retrieval (IR) is the activity of finding resources that have relevant information, usually documents, from a large collection, in response to a query [19]. Web search engines like *Google* and *Bing* are examples of large-scale distributed systems for IR.

### 2.3.1 Indexing

Indexing is an operation to facilitate searches, by avoiding the need to do full scans over the entire dataset. With indexes, faster retrievals are traded for additional storage and increased update times because the index has to be maintained.

Inverted indexes are data structures used in search engines to quickly find documents containing the words in a query. As the index stores a list of words and the documents in which they appear, matches are quickly retrieved. Stemming can be useful to increase matches, as suffixes like plurals are removed.

### 2.3.2 Querying

An IR process starts when a user enters a query into the system, which is then evaluated, and a set of relevant results are shown to the user. Query expansion techniques like finding synonyms and fixing spelling errors are often used to improve performance. The results are typically ranked on how well each document matches the given query.

Boolean ranking models are the easiest to implement, as documents are only fetched if they exactly match the query. Vector space models calculate the distance between the query weight vector and the document weight vector, so partial matches can be retrieved, and documents can be ordered based on the similarity score. Probabilistic models use probability theory to estimate how likely a document is relevant to the information need and started to show very good results with the Okapi BM25 weighting scheme [10]. Feature based models can incorporate many features like another model's score, term frequencies, so that machine learning algorithms can be trained [20].

### 2.3.3 Evaluation measures

Metrics must be used to compare the performance of different IR algorithms. Unranked sets can be evaluated using precision, the fraction of retrieved documents that are relevant, or recall, the fraction of relevant documents that are retrieved.

A perfect recall of 1 can be achieved, at the cost of decreasing precision, by always retrieving all documents. As such, the F1 score (or F-measure) is often used to balance the trade-off between precision and recall [10], by calculating their harmonic mean, as represented in formula 2.2, where $P$ is the precision and $R$ is the recall.

$$\mathrm{F1} = 2 \times \frac{\mathrm{P} \times \mathrm{R}}{\mathrm{P} + \mathrm{R}} \tag{2.2}$$

On huge web search engines, it is not possible to calculate absolute recall as the number of relevant results for a given query are usually unknown [21]. On these cases, the mean precision of the top results on a set of queries can be considered.

As the number of matches increases, ordering becomes more important. To evaluate ordered sets, the mean average precision and the normalized discounted cumulative gain are widely used metrics [10].

When an IR system has already been deployed, new versions can be tested by serving them to a small random sample of users. Metrics such as the number of clicks on the first result or the first page can be evaluated to determine the best version. This kind of process is known as *AB testing* [10].

## 2.4 Technologies

There are a few key technology components that must be compared to develop a data pipeline, a search engine, a search API and a web dashboard. Due to budget constraints, only open source software is to be considered. More in-depth details of the selected technologies are available in Chapter 5.

### 2.4.1 Data pipeline

A simple data pipeline is needed to synchronize the operational and analytical database. Syncing the databases should be as frequent as possible to have fresh data while avoiding high load the operational system. The pipeline should be easy to create, monitor and maintain. With these objectives in mind, two tools stand out, Jenkins[1] and Logstash[2].

Jenkins is an open source automation server. Although Jenkins is mostly used for building, testing and deploying software, it is also a good choice for simple ETL pipelines as the execution of programs can be scheduled and failures in the pipeline are reported. The actual program responsible for the ETL operation needs to be developed or generated with an ETL tool.

Logstash is a data collection engine originally built for log collection. Logstash can ingest data from multiple sources simultaneously, transform it, and then send it to a variety of destinations [22]. Only a single configuration file is required, per pipeline, to handle scheduling and the ETL operation, which is a significant advantage for small tasks. Metrics are exposed through an API for monitoring purposes.

[1] https://jenkins.io
[2] https://www.elastic.co/logstash

With the research made, Logstash appears to be the right tool for this case. The simplicity of having to create and maintain a few small configuration files is a big advantage over other alternatives.

### 2.4.2 Search engine

Elasticsearch[3] and Solr[4] are the most popular open source search engines [23]. Both of them expose an HTTP API and are based on Lucene, a high-performance full-featured text search engine library written in Java. In terms of functionalities, they are very similar, but Elasticsearch offers more integrations, better analytical queries, easier distributed scaling [24] and is much more popular according to Google Trends, as represented in Figure 2.2.

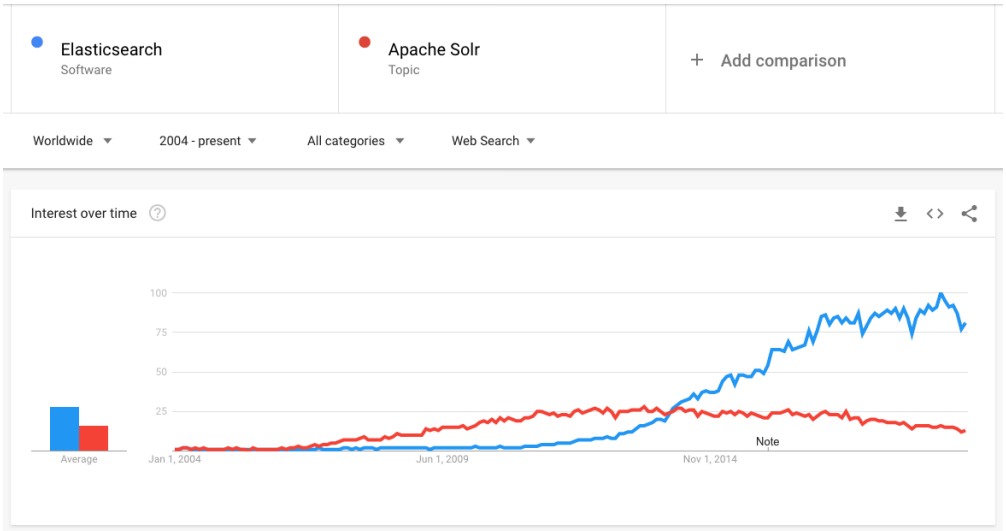


Figure 2.2: Search engines popularity [25]

Considering the popularity and additional features of Elasticsearch, especially on the analytical side for dashboard construction, it is the logical choice as it even removes the need for a separate analytical database. For security reasons, Elasticsearch should not be exposed to the public, even if authentication and traffic encryption are activated [26].

The development of a search API to act as a middleman between the website and Elasticsearch is essential, reducing exposure to security risks and enabling the addition of more features in the future.

### 2.4.3 Search API

For web development there are many competing programming languages and software libraries. It is important to choose a language that provides high performance and increases developer productivity. The ability to catch errors at compile time and enforcing type constraints are also desired, as studies show that static typing improves the maintainability of software systems [27].

A comparison of popular programming languages is presented in Table 2.1 using relevant metrics. Some of these evaluations are subjective and based on personal experience and research.

[3] https://www.elastic.co/elasticsearch
[4] https://lucene.apache.org/solr

Table 2.1: Programming languages comparison

| | C# | Go | Java | JavaScript | Python |
|---|---|---|---|---|---|
| **Compiled** | Yes (bytecode) | Yes | Yes (bytecode) | No | No |
| **Static typing** | Yes | Yes | Yes | No | Optional |
| **Performance** | Fast | Fast | Fast | Fast (I/O) | Slow |
| **Dev. speed** | Medium | Fast | Slow | Medium | Fast |
| **Ecosystem** | Medium | Medium | Big | Big | Big |

The Go programming language seems to be the strongest contender of the group. Being compiled, performance is better than interpreted alternatives and deployment is easier because a single binary can be generated and does not depend on a runtime environment being installed. Although the ecosystem is still growing, the Go standard library already includes the needed utilities for easy web development.

Go was designed at Google with the purpose of being a fast, efficient and safe statically typed, compiled language with support for networked and multi-core computing [28]. The language includes concurrency primitives as well as a garbage collector.

### 2.4.4 Web dashboard

To create dashboards there are many Business Intelligence (BI) solutions available on the market. They are useful for structured data analysis but somewhat limited when it comes to advanced data transformations and text analysis. Computational notebooks allow users to perform data analysis using programming languages like Python or R. The code is organized by paragraphs that can be executed individually or in sequence. Notebooks can be easily shared, as they are files containing all the code, outputs and some metadata. They are increasingly becoming a viable alternative to BI tools [29].

There are two notable open source alternatives, Zeppelin[5] and JupyterHub[6], that allow multiple users to collaborate on the creation of web dashboards using notebooks. They both support multiple programming languages, however only Zeppelin allows the use of different languages in the same notebook. Zeppelin has much more features, such as role-based authorization, a report mode that does not show any code, and a better user interface, with forms for interactive input. JupyterHub has other advantages like the higher Jupyter ecosystem popularity, many plugins to add the missing features, and from the tests made, it feels slightly faster while consuming less resources.

For this project, Zeppelin was selected because it provides better authorization support and more features without having to install other plugins.

## 2.5 Summary

A value proposition for this project was presented taking into consideration the problem, its context and how it can benefit the Praxis platform. An overview of the state of the art regarding text mining and information retrieval was also presented. Alternative technologies were compared to help efficiently achieve the desired objectives.

[5] https://zeppelin.apache.org
[6] https://jupyter.org/hub

The envisioned solution consists on using Logstash to build a data pipeline, Elasticsearch to handle searches, behind a custom search API made in Go, and Zeppelin for the web dashboards and interactive data analysis. This solution will improve searches and provide insights about the existing supply and demand on Praxis so that actions can be taken to reduce the market gap.

# Chapter 3

# Technical description

This chapter describes the analysis and design of the software solution, which consists on the development of various modules. Praxis administrators want a web dashboard to extract useful information about the market supply and demand, and an improved search engine to retrieve better results.

## 3.1 Analysis

To design and implement a software project, functional and non-functional requirements have to be defined. Metrics to evaluate the final result should also be defined so that objectives are clear, measurable and work can be prioritized.

The FURPS+ model, created by Robert Grady at Hewlett-Packard, is a classification system to organize requirements in different categories [30], as described in Table 3.1.

Table 3.1: FURPS+ overview

| **Category** | **Sub-categories examples** |
|---|---|
| Functionality | Features, compatibility, security |
| Usability | Aesthetics, responsiveness, documentation |
| Reliability | Availability, stability, accuracy |
| Performance | Speed, efficiency, scalability |
| Supportability | Maintainability, testability, modularity |
| Design requirements | Restrict technologies, like the type of database |
| Implementation requirements | Use a specific programming language or environment |
| Interface requirements | External interactions and data formats required |
| Physical requirements | Physical hardware characteristics like weight and size |

### 3.1.1 Requirements engineering

Functional requirements define the operations that a system must be able to perform. Most of these are described as use cases in Figure 3.1. Non-functional requirements specify quality attributes to judge how well a system is operating.

The actor present in most interactions is the Praxis administrator, as other users do not have access to the web dashboard. Any user is able to interact with the system when searching available internships.

The dashboard must show summary statistics and support filtering options like date ranges. To analyse internships, clusters can be created and compared based on their textual content.

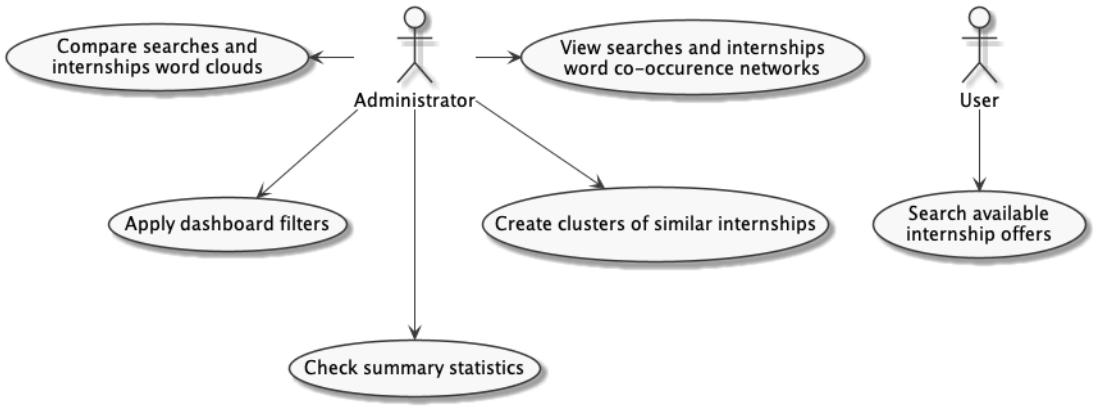


Figure 3.1: Use case diagram

Regarding visualizations, it should be possible to create bar charts, area charts and line charts to view numerical data. Word clouds and word co-occurrence networks are good options to assist interpreting textual data.

Other system-wide functional and non-functional requirements are specified in Table 3.2 for each of the software modules.

Table 3.2: Non-functional requirements

| **Module(s)** | **FURPS+ category** | **Description** |
|---|---|---|
| Dashboard | Functionality | Secure authentication |
| Dashboard | Functionality | Compatibility with modern web browsers |
| Dashboard | Usability | Intuitive user interface |
| Search API | Reliability | Better *F1 score* than current search engine |
| Search API | Performance | Response times <1 second for 99% of cases |
| All | Supportability | Easy to deploy and maintain |
| All | Design req. | Favour existing open software over development |
| All | Implementation req. | All software must be compatible with Linux |
| Data pipeline | Interface req. | Must interact with MariaDB |

### 3.1.2 System evaluation

The evaluation of the developed system must take into account the number of completed functional and non-functional requirements. Some of them are either implemented or not, while others require a more careful evaluation as they can be subjective or take more effort to evaluate correctly.

To evaluate the usability of the dashboard, a survey will be conducted. People will be able to interact with a dashboard containing test data and then answer some questions about their experience. Once enough feedback is collected, results are evaluated.

Regarding the search API, response times can be evaluated with a load test to simulate a high volume of users. This should be done in an environment identical to production to guarantee similar future metrics. To evaluate the information retrieval performance, the *F1 score* is a good metric when dealing with unordered sets. Although the internships are ordered, it is not very significant in this case as there are not many results for each query,

so they usually all appear in the first page. Comparing this metric between multiple search engines requires selecting a representative sample of queries/documents and averaging the *F1 scores* obtained for each query.

## 3.2 Design

The design is an important phase of the software development life cycle. It needs to consider the requirements and the integration with existing software and infrastructure. A good design is critical for the long-term maintainability of a system.

### 3.2.1 Praxis data model

Praxis uses a single relational database, MariaDB[1], to store all the website data and the Lucene library to provide full-text search for internship offers. Table 3.3 describes the data model for an internship offer.

Table 3.3: Structure of an internship offer in Praxis

| Field | Description | Required |
|---|---|---|
| Title | Offer title | Yes |
| Description | Description of the offer | Yes |
| Skills | List of prerequisite skills | No |
| Benefits | List of benefits for the student | No |
| Institution | Name of the institution | Yes |
| Institution type | Type of institution | Yes |
| Location | City and country of the offer | Yes |
| Study topics | Disciplines part of the internship | No |
| Study areas | Subcategories of study topics | No |
| Study degree | Required study degree | No |
| Languages | List of required languages | Yes |
| Period | Start and end date for the internship | Yes |
| Submit date | Date of internship submission | Yes |
| Valid until | Limit date to apply to the internship | Yes |
| Status | Can be available, assigned or cancelled | Yes |
| Applicants | Total number of candidate students | Yes |

Searches can match the offer's title, description or other fields. The description may contain Hypertext Markup Language (HTML) tags, which must be ignored. To collect information about supply and demand, all search queries and additional information such as the date and number of results are logged in the database. The log data model is presented in Table 3.4.

Table 3.4: Structure of search results logging

| Field | Description | Required |
|---|---|---|
| Query | Keywords entered in search box | Yes |
| Date | Date when the search was made | Yes |
| Results count | Total number of results for the query | Yes |

[1] https://mariadb.org

Unfortunately, a user session identifier field does not exist, so it is not possible to extract information such as how many searches a user makes before getting a match. The search time is also not available, which could be useful to check hourly activity. Another critical piece of information missing are the filters. In the Praxis website, users can select filters like the country or study area, but these are not logged, so many records just have an empty query.

### 3.2.2 System architecture

To provide an overview of the required components and how they communicate with each other, a component diagram is displayed in Figure 3.2. A modular architecture is proposed to enable the easy replacement of different components if new technologies achieve measurable improvements.

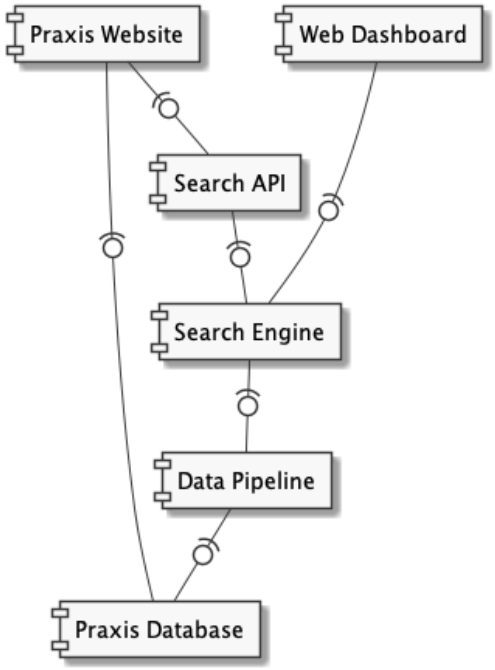


Figure 3.2: System architecture diagram

The *Praxis Website* and *Praxis Database* are already a part of the existing Praxis platform. To copy data from the operational database to other specialized systems, a batch *Data Pipeline* that executes periodically is required. The *Search Engine*, which also acts as an analytical database, is used by the *Web Dashboard* and the *Search API*. This API is a middleman between the *Praxis Website* and the *Search Engine*, to provide security and additional features if needed.

### 3.2.3 Infrastructure overview

Praxis is currently hosted in a single virtual server in a cloud provider. New software modules are to be installed in a different server so that existing workloads are not impacted. A deployment diagram in Figure 3.3 shows the hardware and software components of the entire system.

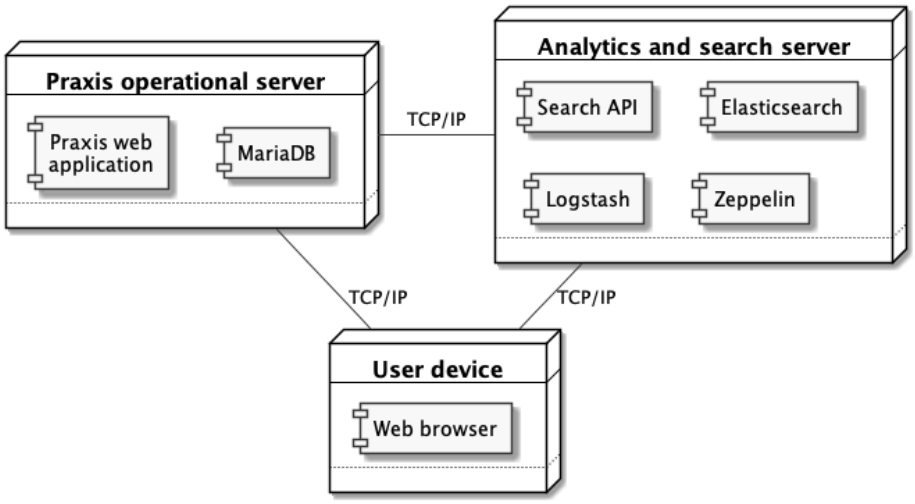


Figure 3.3: Deployment diagram

## 3.3 Summary

The functional and non-functional requirements were enumerated, with metrics being defined to evaluate the system. Based on existing constraints, a viable solution was designed. To achieve the desired objectives, a data pipeline, a web dashboard, a search engine and a search API must be implemented using open source technologies and following good development practices.

# Chapter 4

# Exploratory data analysis

This chapter provides an analysis of the historical data collected on Praxis. Both internship offers and search requests are studied to get insights on the market gap and find ways to improve the search results. In addition to finding some market gaps, a few issues with the current search engine were discovered.

A dump of the operational database was received, with data until November 2019, containing only the relevant tables to perform the analysis and setup the development environment. The visualizations were created in R, so any developed code can be reused in the dashboard implementation.

## 4.1 Internship offers

The first internship offer was created in October 2013 and since then, 2163 have been submitted. As observed in Figure 4.1, the number of internships submitted has not grown significantly in the last years. In total, 48% of internships got no applicants at all.

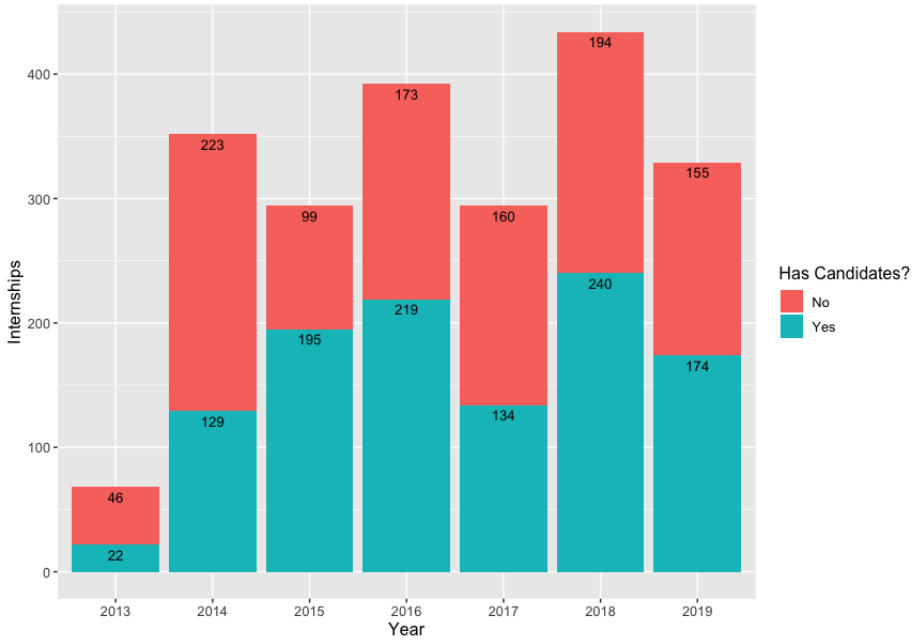


Figure 4.1: Submitted internships by year

Because internships may remain open for long periods, the number of available offers has been growing, as visible in Figure 4.2.

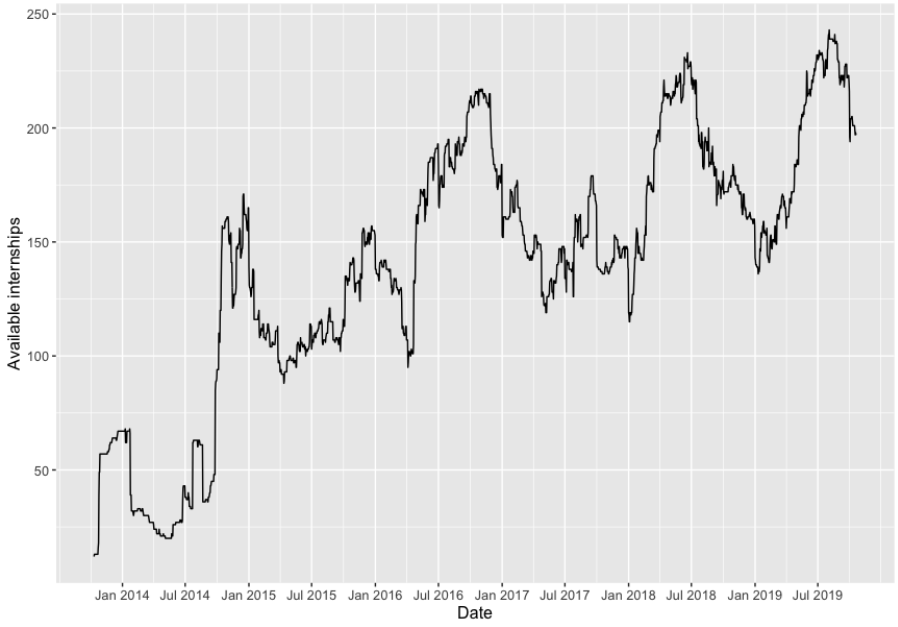


Figure 4.2: Available proposals over time

Multiple languages can be specified as a requirement for an internship. English is by far the most common, followed by Portuguese, Spanish, German and French, as displayed in Figure 4.3. Most proposals are also written in English.

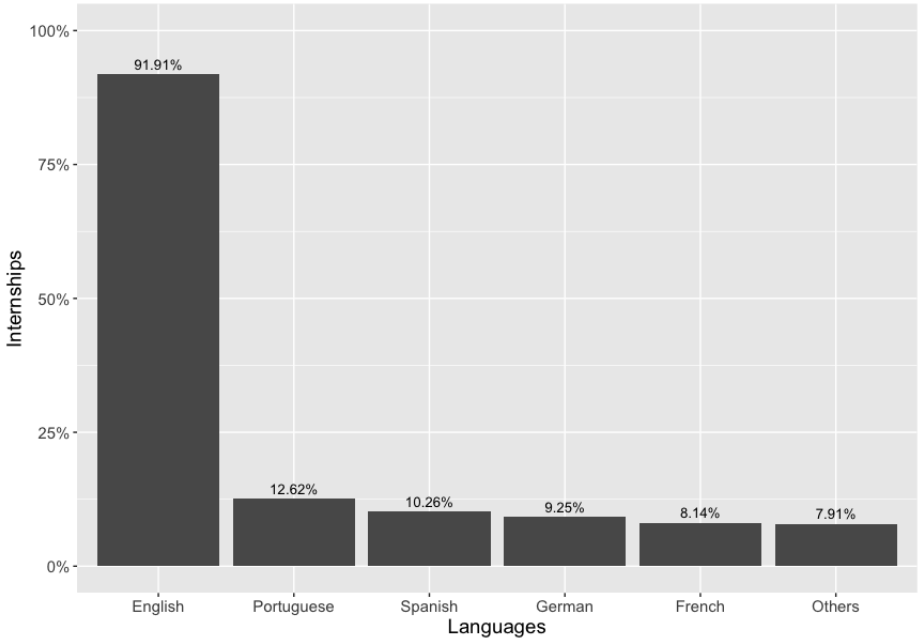


Figure 4.3: Required languages

Praxis appears to be more popular in Portugal and Spain, as there are many more internships submitted from those countries when compared to others, as shown in Figure 4.4, corresponding to 42% of all internships.

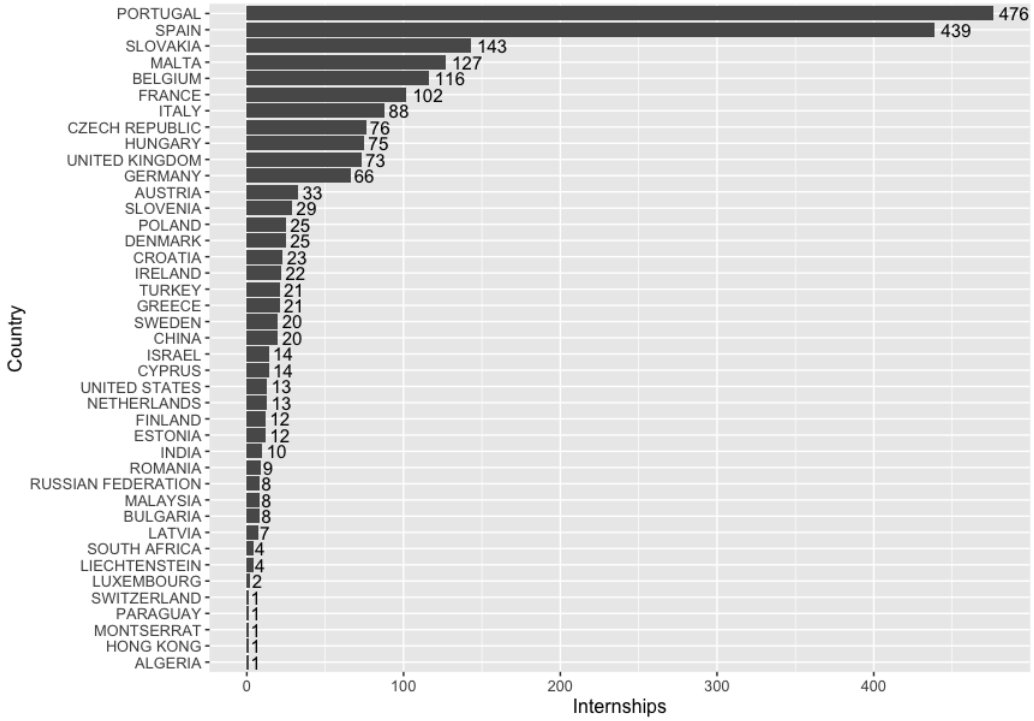


Figure 4.4: Internships by country

From 2014, up to nine topics can be assigned per internship. Figure 4.5 shows the evolution of those topics, with the most notable changes being the increase of the Business and Economics topic, accompanied by the decrease of Applied Sciences, Professions and Arts topic. Environmental Sciences, Law and Medicine topics had very few internships submitted.

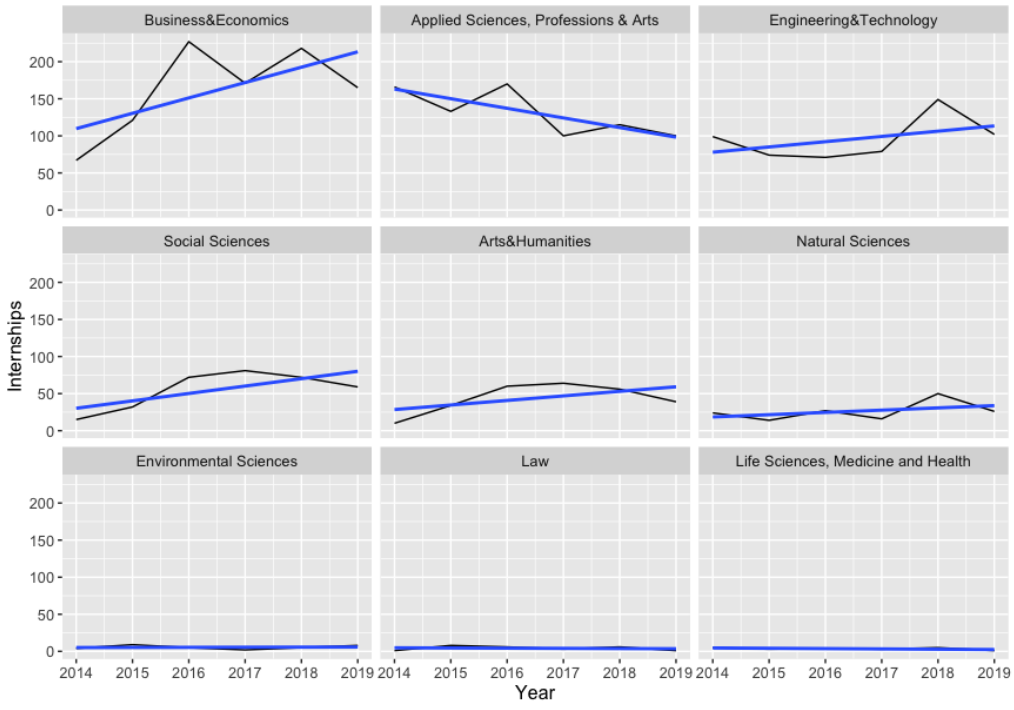


Figure 4.5: Study topics relative frequencies by year

Figure 4.6 shows the top study areas for each topic. Computer related areas dominate the Engineering and Natural Sciences topics. The areas with most supply are Marketing, Business Administration, Computer Science and Education.

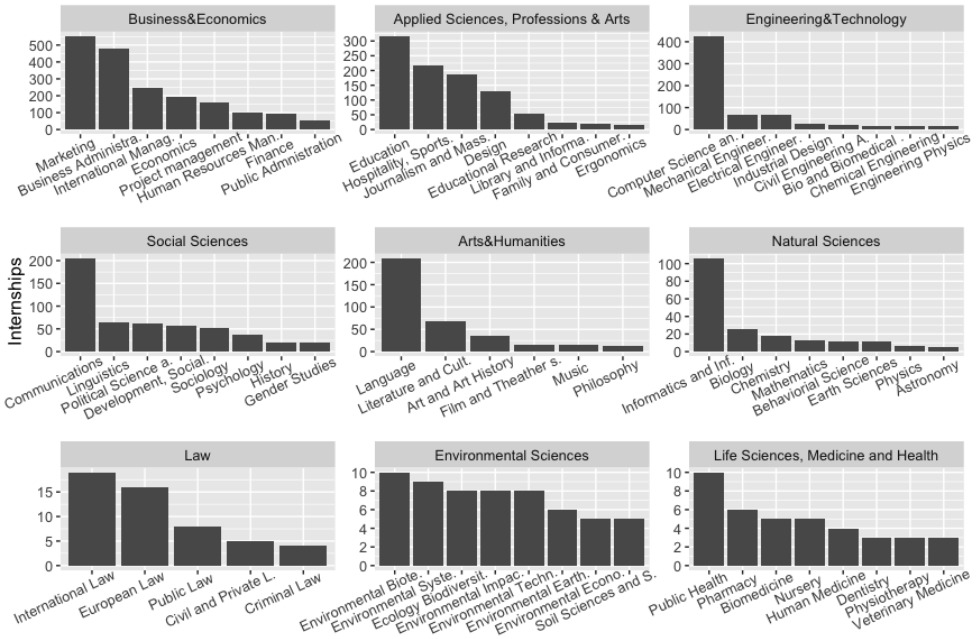


Figure 4.6: Top study areas per topic

The ability to select multiple areas per internship can be misleading when reporting these categories. For example, in Figure 4.7 it is possible to observe that Marketing internships are also often associated with areas like Journalism, Language and Communications, which belong to three other topics, causing their count to appear higher.


Figure 4.7: Top study areas combinations

The analysis of the structured data already provides some information about the market, but more insights can be extracted from the textual data. Figure 4.8 shows the most common words used in internships, without English or domain stop words like "the" and "internship".


Figure 4.8: Internships word cloud

Word relationships can be visualized with co-occurrence networks, like the one in Figure 4.9, where words that occur together with a maximum distance of 1 term are connected. In this case, instead of removing stop words, only nouns, verbs and adjectives were kept, as these word classes are more relevant for analytical purposes. For this NLP task, the UDPipe[1] POS tagger annotated the text, using an English pre-trained model built on the *UD English EWT treebank*, whose corpus contains 254 830 words and 16 622 sentences [31].

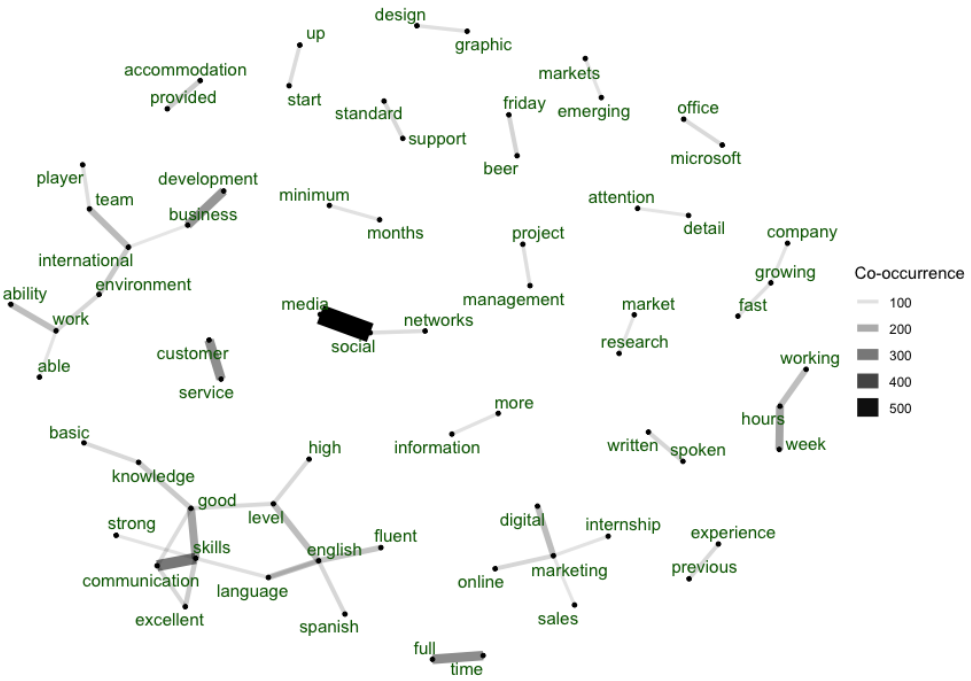


Figure 4.9: Internships co-occurrence network

[1] https://cran.r-project.org/web/packages/udpipe

From the network, we can observe many candidate stop words and some topics start to emerge like social media, customer service, business development, digital marketing, project management, graphic design and others.

### 4.1.1 Cluster analysis

To uncover more patterns from the internships, a few clustering and topic modelling algorithms were compared. Topics are more flexible than hard cluster assignments because one document may be assigned to multiple topics, as is the case with LDA, a probabilistic model where each document is represented as a mixture of topics and every topic is represented by a mixture of words [32].

Hierarchical clustering and k-means variants are often used to cluster documents [33]. Distance based algorithms require a similarity measure between documents to perform clustering. The cosine similarity is widely used for text analysis, as metrics like the Euclidean distance are not appropriate for high dimensional, sparse data [34]. Using LSA or other dimensionality reducing algorithms can improve the performance of clustering and classification algorithms [35].

Hierarchical clustering has the advantage of producing a dendrogram, a tree-based representation of the clusters, and the number of clusters does not have to be pre-specified. A limitation of hierarchical algorithms is the time complexity, which is at least quadratic, since the similarity between all documents has to be calculated. The dendrogram in Figure 4.10, constructed using Ward's criterion, shows that reducing the number of dimensions using LSA provides better results, as the original vector space causes one cluster to be dominant and the others to be outliers.

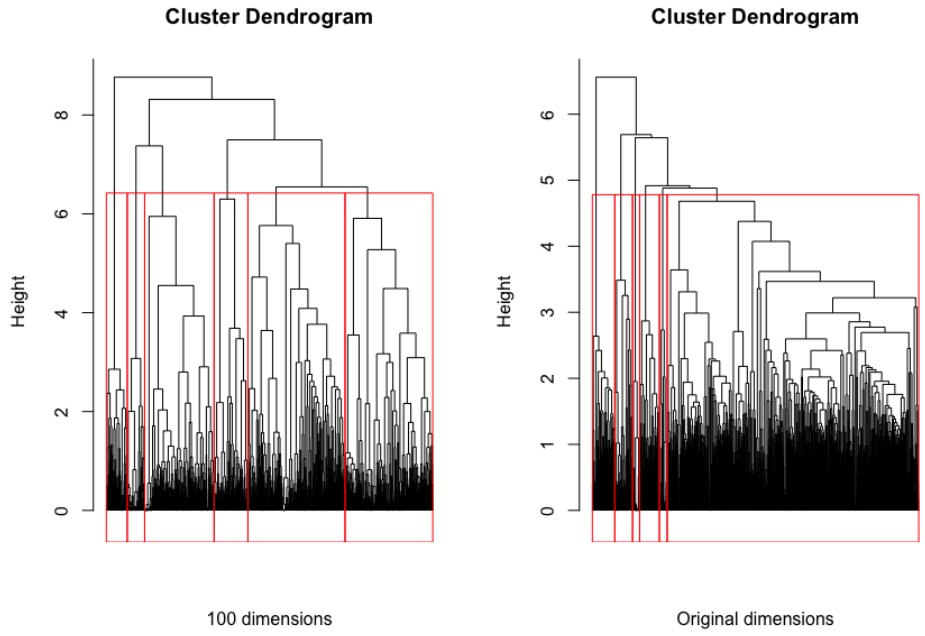


Figure 4.10: Dendrograms with a cut of 6 clusters for different dimensions

A particular variant of k-means often used for text data is spherical k-means, which uses the cosine similarity instead of the Euclidean distance [36]. K-means has the advantage

of scaling linearly with the number of documents, however it is not deterministic and the number of clusters must be pre-specified. K-means is sensitive to the initial seed selection, so different techniques have been used to improve performance on large data sets, such as using hierarchical clustering on a sample of data to select the initial centroids [36].

Determining the optimal number of clusters is a difficult task. Possible strategies include the manual inspection of the top keywords in the clusters or calculating metrics like the mean silhouette. The silhouette of a document measures how similar it is to other documents in its own cluster, when compared to other clusters, ranging from -1 to +1 [35].

The following steps were taken to cluster the internships:

1. Concatenate the internship title and description.
2. Perform tokenization to split each word.
3. Reduce dimensions by removing stop words.
4. Compute feature vectors:
    - Term frequencies for LDA topic modelling.
    - TF-IDF and LSA for hierarchical clustering and spherical k-means.
5. Apply clustering algorithms.
6. Evaluate the results.

Figure 4.11 compares the mean silhouette values for different combinations of algorithms, LSA dimensions and number of clusters. Spherical k-means has slightly better results than other algorithms. It was also observed that 100 dimensions are enough to obtain good results.

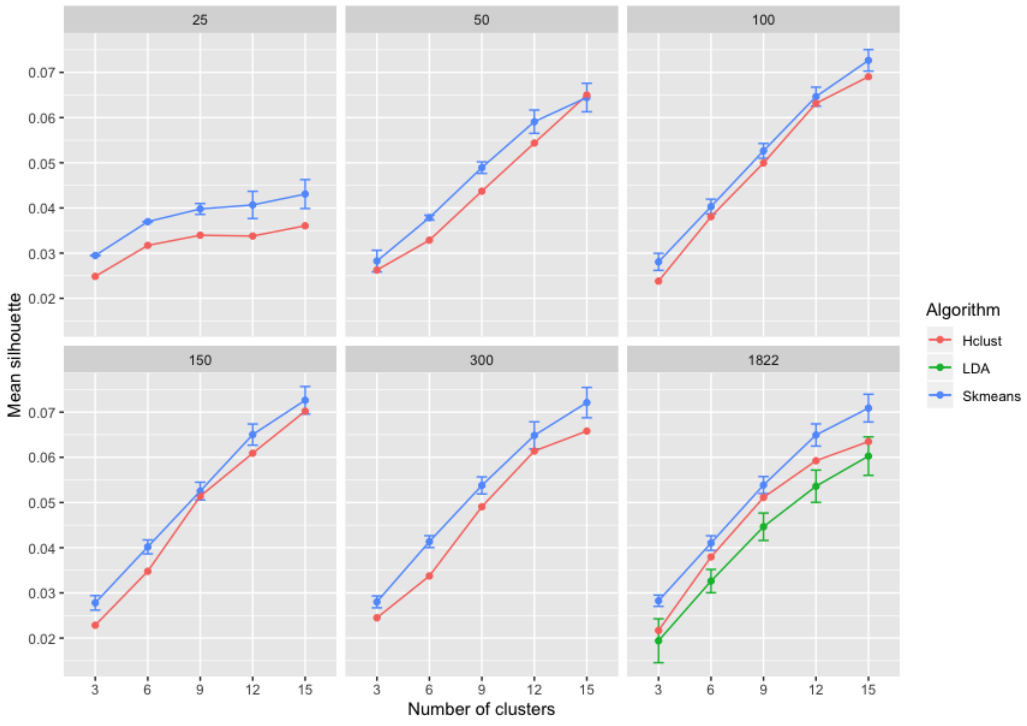


Figure 4.11: Mean silhouette comparison from 30 iterations

Other techniques like n-grams, stemming, lemmatization and filtering certain word classes did not show significant improvements and were more computationally expensive.

To better understand the generated clusters, it is useful to look at the most relevant terms in each cluster, based on the original TF-IDF matrix. By visually inspecting these, a good number of clusters appears to be around 6, as higher numbers cause some very small clusters to be created, since a few companies submit many internships with similar contents, which get grouped together.

Figure 4.12 presents the most relevant terms for each cluster when using spherical k-means with 6 clusters and the dimensions reduced to 100 through LSA. The following clusters can be identified:

- **Cluster A** - Sales, customer service and hospitality. Many of these are submitted in Barcelona.
- **Cluster B** - Erasmus, university research and management.
- **Cluster C** - Education, mostly related with teaching other languages.
- **Cluster D** - Marketing.
- **Cluster E** - Engineering, software development and design.
- **Cluster F** - Internships written in Portuguese.

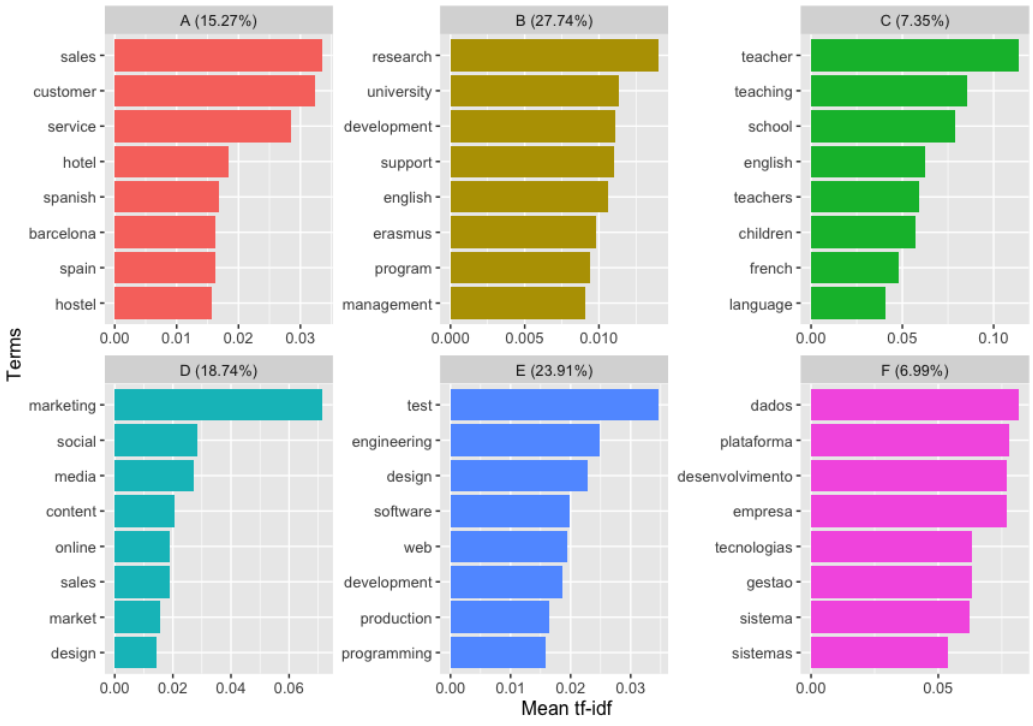


Figure 4.12: Most important terms per cluster

Looking at the evolution of these topics over the years in Figure 4.13, we can see a significant decrease in the education internships when compared to other areas. Engineering internships also appear to become less dominant, but most internships written in Portuguese are related with software engineering. There is also an increase on the internships related with sales, customer service, hospitality and marketing.

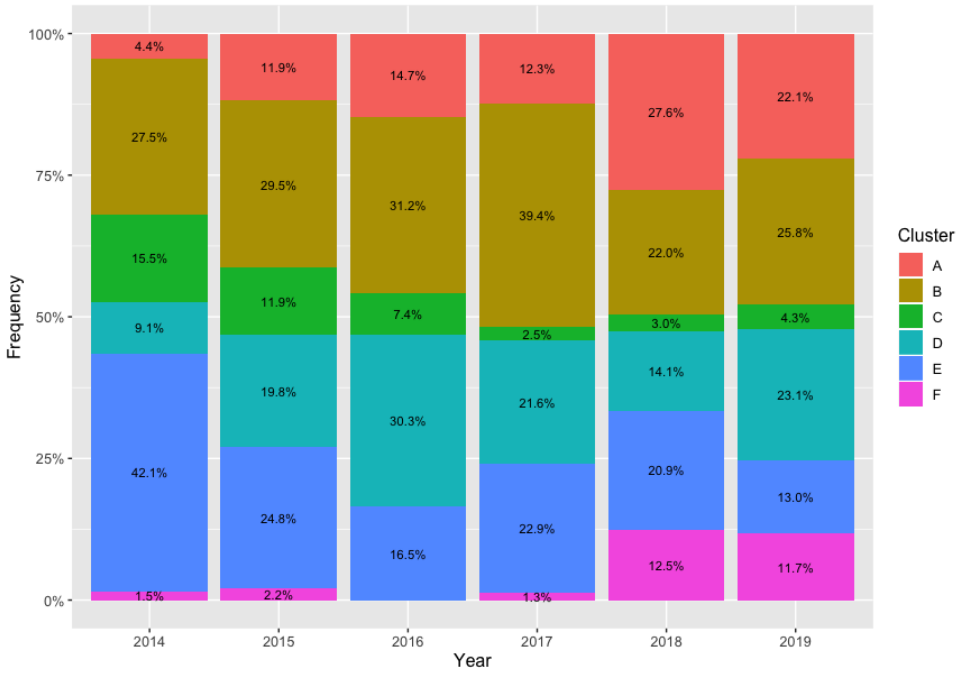


Figure 4.13: Clusters evolution

An effective way to visualize the differences in cluster assignments between different clustering algorithms, is to do a two-dimensional projection of the original TF-IDF matrix. A state of the art nonlinear dimensionality reduction algorithm for embedding high-dimensional data in two or three dimensions, for data visualization, is T-distributed Stochastic Neighbor Embedding (T-SNE) [37]. A comparison of the cluster assignments made by the tested algorithms, on the original dimensions, is displayed in Figure 4.14.

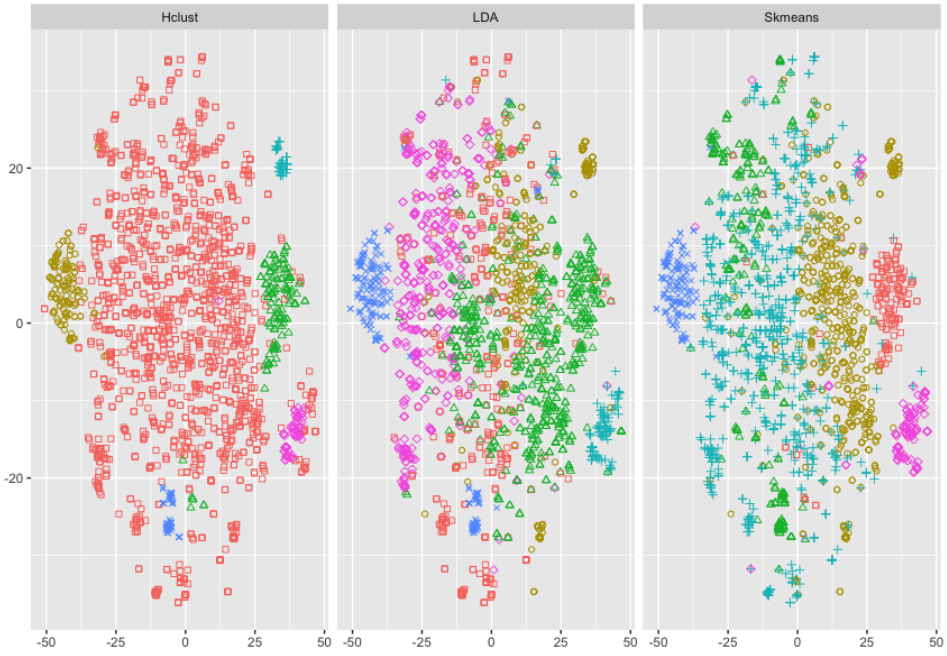


Figure 4.14: t-SNE projection of clustering algorithms

As previously observed in the dendrogram, without dimensionality reduction techniques, hierarchical clustering tends to create small clusters with outliers. Between spherical k-means and LDA, some clusters share similarities and others do not, as variations in the topics are identified. For example, with LDA, education and university research were part of a single topic, at 6 dimensions.

## 4.2 Search requests

Since searches started being logged in 2014, a total of 151 940 queries were made in Praxis. About 83% of searches contain a single term, 16% have two terms and only 1% three or more. As observed in Figure 4.15, the rate of searches has been decreasing, and in the summer, searches are less frequent. Between 2016 and 2017 the number of searches without any results increased from 29% to 52%, which warrants further investigation.

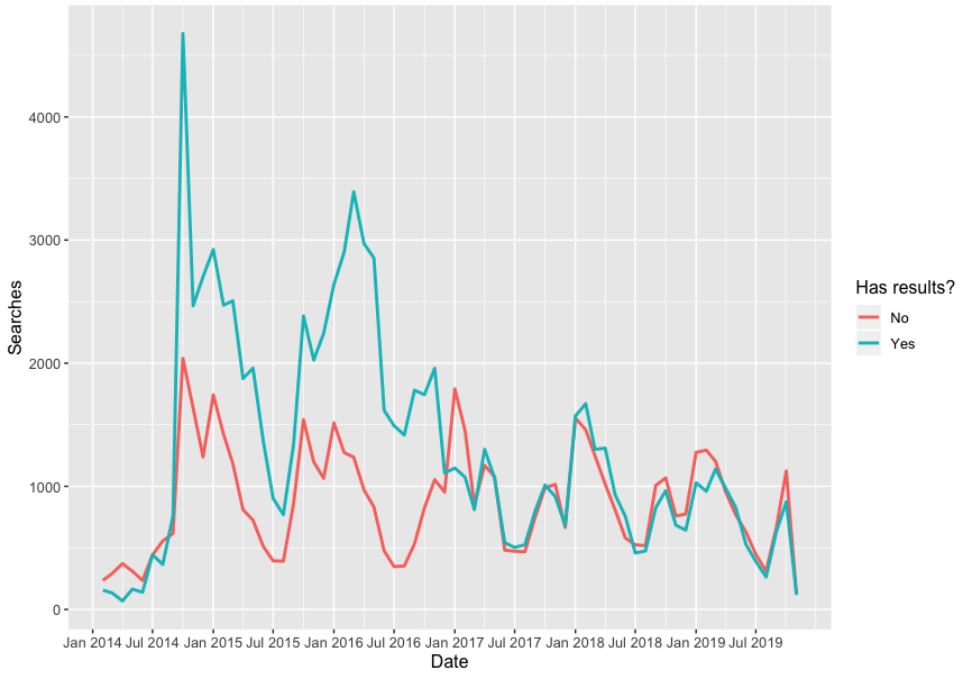


Figure 4.15: Total searches made by month

Some anomalies are detected in Figure 4.16, reaching 600 results for some searches, which should not be possible given that the maximum available internships in a single day was less than 250. In 2016 and earlier, expired internships were sometimes being returned. After this year, it appears that some queries are returning all available internships for the respective day.

The issue lies with the current Praxis search engine. Text queries are being parsed by Lucene, when they should only be matched. Queries with special keywords and symbols like "and", "or", "not", "to", "-", among others, which are part of the Lucene query syntax, are returning all available internships. When queries contain more than 2 consecutive whitespaces or trailing whitespaces, they return too many results. This behaviour also happens when the query contains non-alphanumeric characters and any filter is applied. As the search engine is indexing fields like the organization type and internship duration, terms like "company" and "month" return nearly everything.

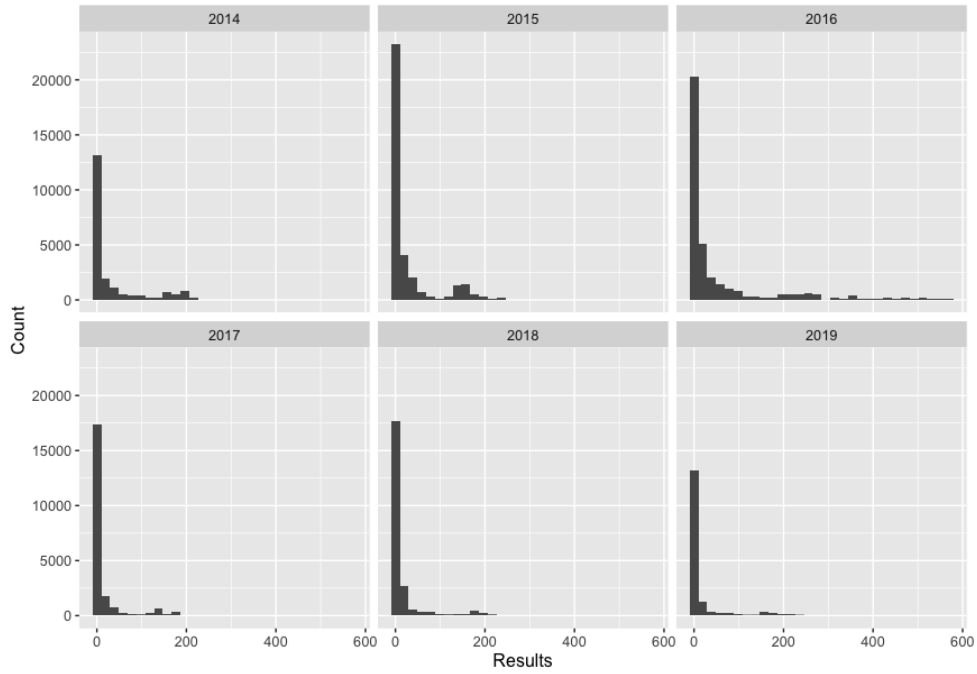


Figure 4.16: Distribution of search results by year

As such, to clean the data, queries that contain excess whitespace, non-alphanumeric characters, stop words, Lucene keywords, or return roughly the number of available proposals on the day they were made, are removed. Figure 4.17 still shows a big difference between 2016 and 2017, suggesting data quality issues associated with the collection of these metrics. In the last 3 years, 55% of searches retrieve no results.

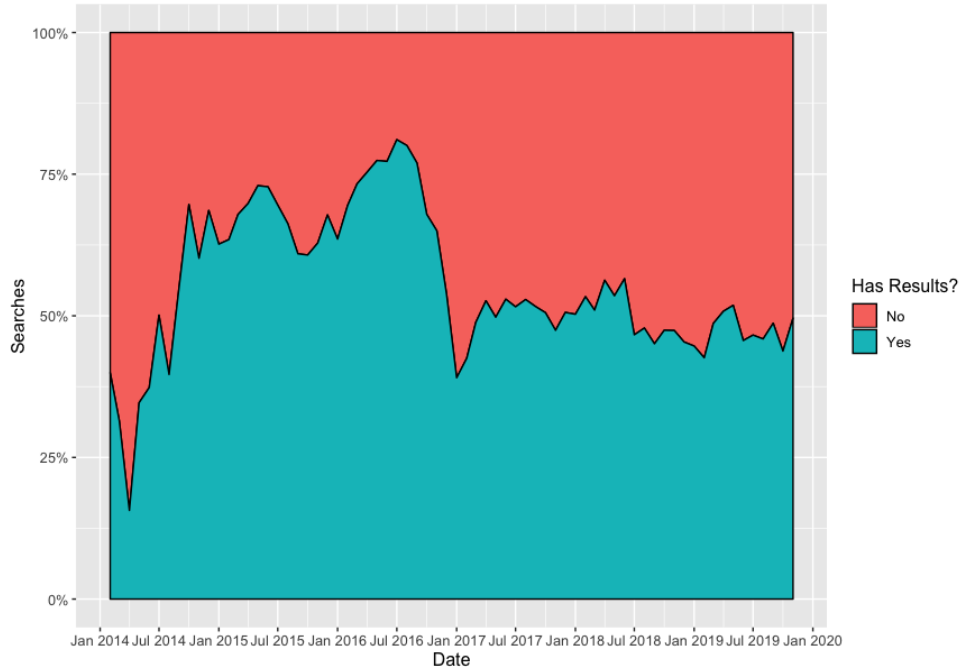


Figure 4.17: Search results over time

By observing the search terms word cloud in Figure 4.18, it is clear that engineering is the most frequent term. Students also often look for design, architecture, marketing and some sciences. Countries and cities are also frequently searched.


Figure 4.18: Search terms word cloud

The co-occurrence network in Figure 4.19 helps identify clusters of related words. Many fields of engineering are popular, but civil engineering is the most searched by students. Other topics like machine learning, renewable energy, therapy are now visible.

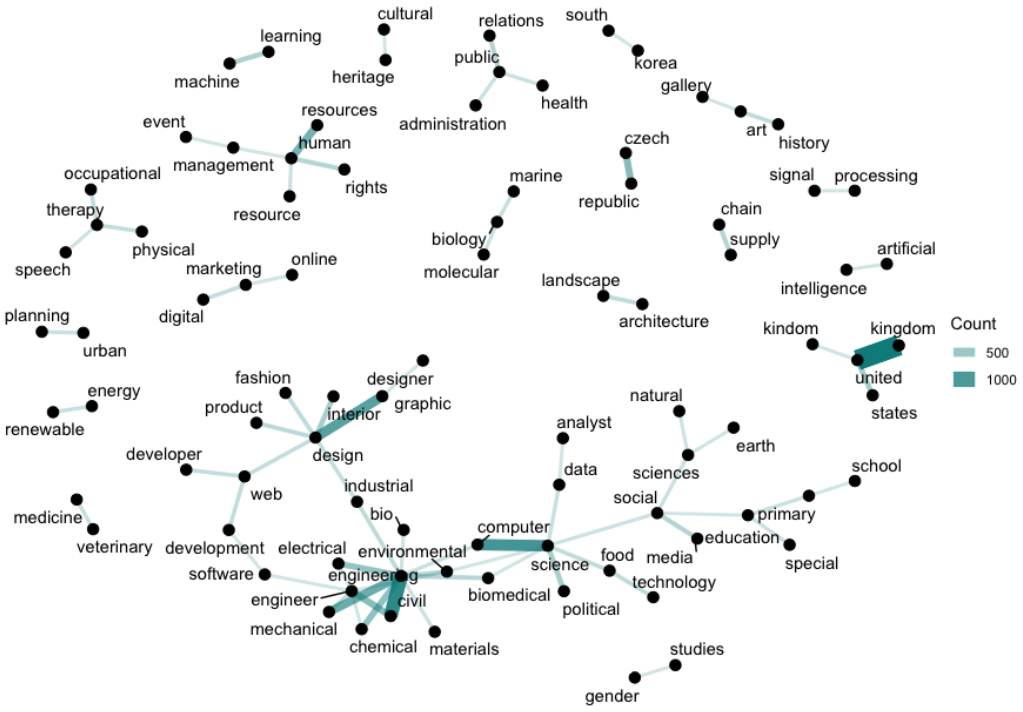


Figure 4.19: Search terms co-occurrence network

## 4.3 Market gap

To help identify market gaps, the distribution of candidates to different internship topics was plotted in Figure 4.20. Medicine and Law topics do not have many internships submitted, but they have higher median number of candidates.

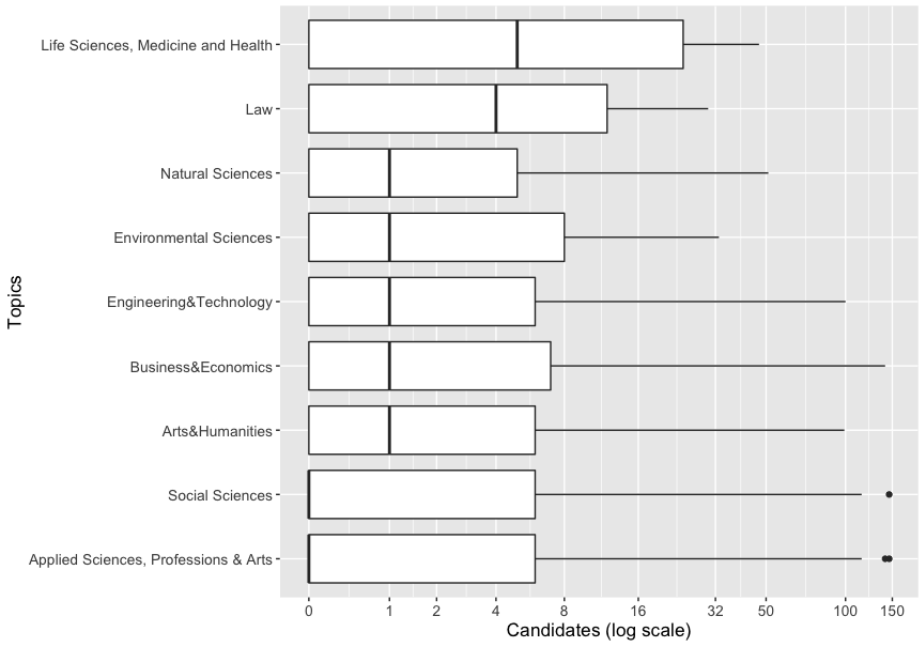


Figure 4.20: Distribution of candidates by study topic

For the comparison of word frequencies between internships and searches in Figure 4.21, countries and cities were added to the stop words. It indicates that engineering, architecture, law, design and psychology are some terms that are not matching many internships.

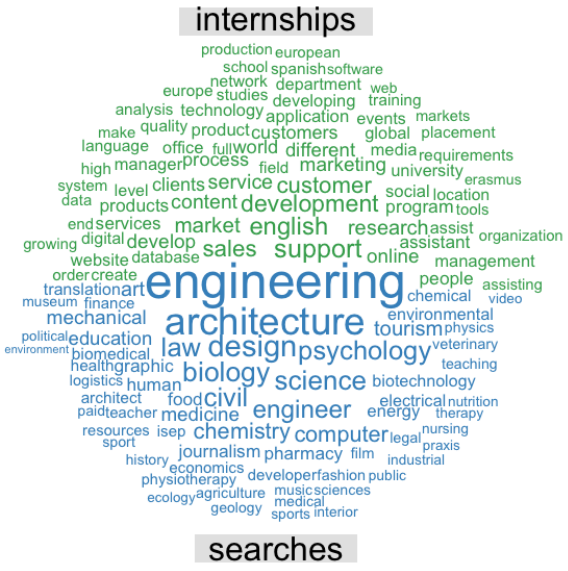


Figure 4.21: Internships and searches comparison word cloud

Among the top 1% most searched queries, some areas stand out in Figure 4.22, with many matches, like marketing, management, computer science, education and design.

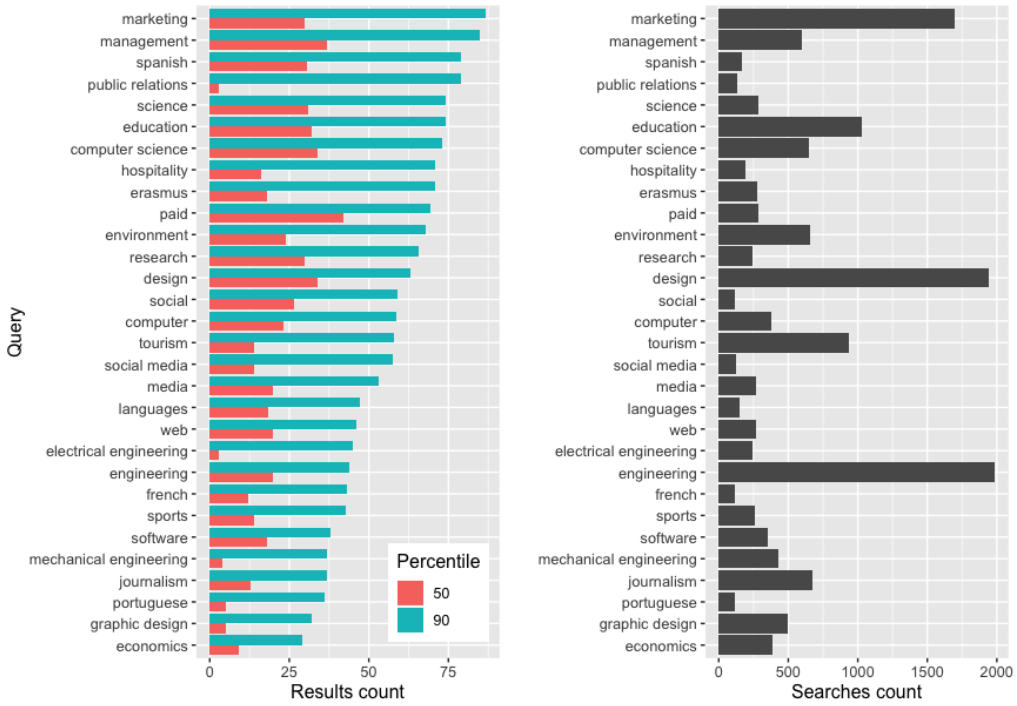


Figure 4.22: Top searched queries with most results

According to Figure 4.23, areas like psychology, chemistry, physics, photography, pharmacy, medicine, civil engineering, nursing and other sciences do not have many matches.

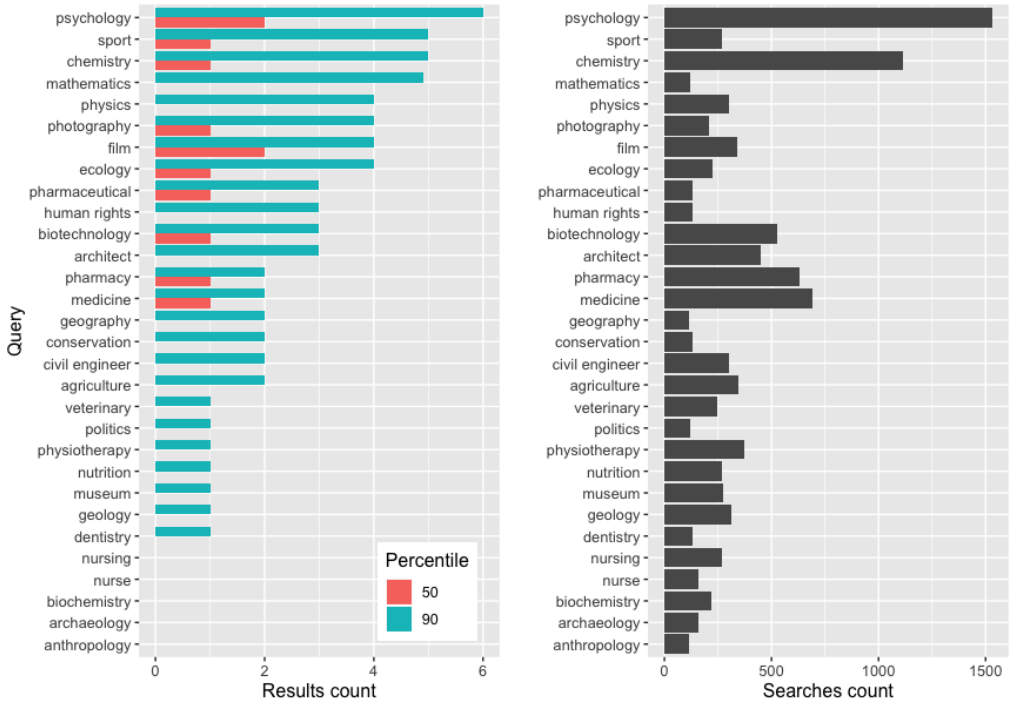


Figure 4.23: Top searched queries with least results

Search queries that never retrieved any results in Figure 4.24 indicate some possible improvements that can be made to the search engine. Search terms may contain typographical errors, which can be corrected, and some terms can be replaced with synonyms, or semantically similar words, to increase the number of matched internships.

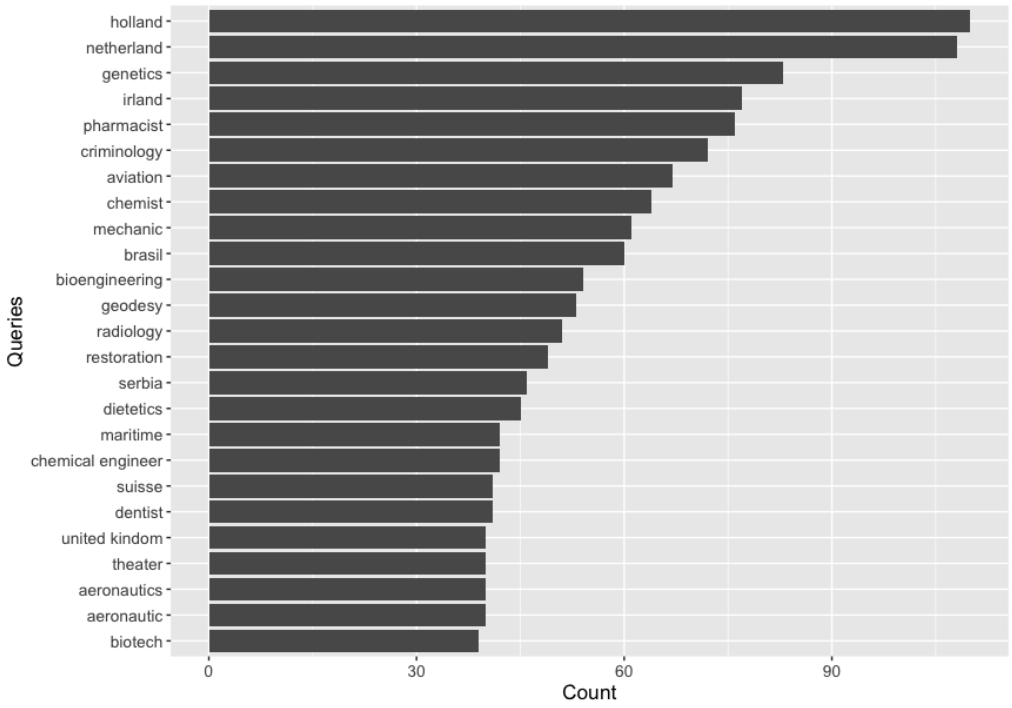


Figure 4.24: Top searched queries that never had any result

## 4.4 Summary

The analysis of the structured and non-structured data in the Praxis historical dataset gave insights into the internship market. Text mining techniques provided valuable information that would otherwise remain hidden. Technology, Marketing, Business and Accommodation related internships are common on Praxis, while areas like Architecture, Law, Medicine, Design, Engineering and Sciences are not very abundant.

Some improvement suggestions for the search engine were identified. The most critical update is preventing users from doing arbitrary Lucene queries. Additional tuning must be done to prevent certain queries from returning too many results. Correcting typographical errors and using related words can increase the number of relevant results.

# Chapter 5

# Solution development

This chapter presents the implementation of the software modules that compose the project. Before detailing the different components of the solution, the development environment is described. Then, the configuration of the data pipeline using Logstash is shown, followed by the setup of Elasticsearch as a search and analytical engine. Finally, Zeppelin is used to create web dashboards that provide insights into the market gap on Praxis.

## 5.1 Development environment

The use of version control is a best practice for software development and has become ubiquitous [38]. Given the advantages it provides, all developed code is pushed to a private Git repository on GitHub[1]. Since there is only a single contributor on this project, a simple branching strategy was used. All code is pushed to the master branch and tags are used to mark certain commits as release points.

Setting up a repeatable development environment is important to ease collaboration and guarantee consistent results across different environments or machines. A study on repeatability of computer systems research shows that a out of a sample containing 601 papers from ACM conferences and journals, 402 papers were backed by code and about 46% of these are not repeatable because the code could not be built [39].

### 5.1.1 Docker

As the issue of reproducibility gets more attention from the research community and companies, the emergence of technologies like Docker[2] can reduce this issue[40]. Docker allows packaging software applications and all their dependencies like system libraries into a single container image. This image can be published and then downloaded on another machine to be run as a container instance, even with a different operating system, as long as Docker is installed.

Containers are much more lightweight than VMs because they do not require a separate Operating System (OS), as they all share the host OS kernel, thus reducing the memory and disk footprint. Docker uses resource isolation features of the Linux kernel, such as cgroups and namespaces [41], so Docker currently needs to run inside a VM on Mac and Windows hosts. Figure 5.1 compares the architecture of a Virtual Machine (VM) and a container.

[1] https://github.com/ruial/praxis
[2] https://www.docker.com

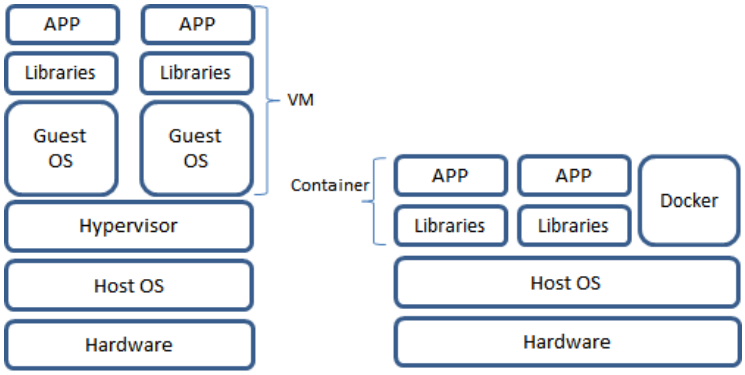


Figure 5.1: VM vs container architecture [42]

A container image is immutable, but container instances are not. Volumes should be used to persist data generated or used by containers, such as configuration files and database storage directories, otherwise data is lost when the container is rebuilt or destroyed. Containers run isolated from each other but are able to communicate when they join the same Docker network.

When changes need to be done to an image, a special file called *Dockerfile*, must be created and contain all the instructions to build the image. An image may have many gigabytes but thanks to intermediate image caches, the build process is accelerated as changes can be incremental.

### 5.1.2 Docker Compose

Having multiple containers running on the same host requires multiple steps, which may be error prone if done manually. Volumes have to be mounted to persist data, networks must be setup to allow communication and various configurations for each container often need to be made.

Docker Compose is a tool for defining and running multi-container applications through a single configuration file using YAML Ain't Markup Language (YAML) format, which by default already creates a network for the containers. A partial example of the configuration file is available on Listing 5.1. Each service has a container image, ports that are exposed, environment variables to be set and volumes to persist data when the container goes down. Building and starting all the containers is done with a single command.

```
version: '3.4'
services:
  mysql:
    image: mariadb:10.4.10
    container_name: mysql
    ports:
      - 3306:3306
    environment:
      - MYSQL_ROOT_PASSWORD=example
    volumes:
      - ./volumes/mysql/init:/docker-entrypoint-initdb.d
      - ./volumes/mysql-data:/var/lib/mysql
  elasticsearch:
    image: elasticsearch:7.4.2
    container_name: elasticsearch
```

```
16      ports:
17        - 9200:9200
18      environment:
19        - discovery.type=single-node
20        - ELASTIC_PASSWORD=some-password
21        - xpack.security.enabled=true
22      volumes:
23        - ./volumes/es-data:/usr/share/elasticsearch/data
24    logstash:
25      image: logstash:7.4.2
26      container_name: logstash
27      depends_on:
28        - elasticsearch
29      ports:
30        - 9600:9600
31      environment:
32        - JDBC_URL=jdbc:mariadb://mysql:3306/db_name
33        - JDBC_USERNAME=root
34        - JDBC_PASSWORD=example
35        - ELASTIC_URL=elasticsearch:9200
36        - ELASTIC_USERNAME=elastic
37        - ELASTIC_PASSWORD=some-password
38      volumes:
39        - ./volumes/logstash/config/pipelines.yml:/usr/share/logstash/
        config/pipelines.yml
40        - ./volumes/logstash/pipeline:/usr/share/logstash/pipeline
41        - ./volumes/logstash/connectors/mariadb-java-client-2.5.2.jar:/usr
       /share/logstash/logstash-core/lib/jars/mariadb-java-client-2.5.2.jar
```

Listing 5.1: Partial docker-compose.yml

Docker Hub[3] is a public registry that has many container images created by the individuals and organizations. When an image with the desired software installed is not publicly available, it must be created, as will be done in section 5.4 for the Zeppelin web dashboard.

The ability to simulate the entire infrastructure on a single machine is excellent for development purposes, as the complete environment is reproducible even on different systems. This was especially helpful in this project to simulate the operational database with a sample of real data.

For production workloads, Docker Compose is less used because containers typically run on different machines due to performance reasons. Therefore, alternatives are often used for deployment and configuration management, as will be discussed in Chapter 6.

## 5.2 Data pipeline

As analytical queries can be computationally expensive and the performance of the operational system should not be degraded, it was decided that another database is to be used for analytical purposes. For this reason, a data pipeline is required to synchronize data from the operational database to the analytical database, which can also be used to enable full-text searches.

[3] https://hub.docker.com

The first step of building a data pipeline is identifying the data sources and destinations. In this case there is a single source, the MariaDB Praxis operational database, and a single destination, Elasticsearch as it can be used for searching and data analysis purposes.

Afterwards, the destination data schema must be defined based on the requirements and taking into account the existing schema of the data source. Then transformations often have to be made. Examples of data transformations include filtering records, performing aggregations, dropping unnecessary record fields and adding new fields derived from existing fields or external sources.

Lastly, since minimal changes are wanted on the operational system and delays in the order of minutes are acceptable, a batching approach with a configurable polling mechanism was chosen. Alternatively, a near real time approach could be achieved by changing existing code to send data directly to new destinations or by using a change data capture platform such as Debezium in conjunction with a streaming platform like Kafka, to stream the database transaction logs as events [43].

### 5.2.1 Logstash

Logstash is the chosen technology to create the pipeline. Despite its name, it has uses beyond collecting and processing logs, thanks to a number of supported plugins. Logstash provides many types of plugins out of the box and additional ones can be developed using Ruby or Java, as JRuby is the default runtime [44]. There are four types of plugins:

- **Input** to extract data as events from a source to Logstash
- **Codec** as part of an input or output to parse the event data format, such as JavaScript Object Notation (JSON)
- **Filter** to apply event transformations
- **Output** to send events from Logstash to a destination

A Logstash pipeline consists of three stages, as presented in Figure 5.2. To start, an input plugin with the correct connection information is required to generate events. Then filter plugins can be combined to modify each event. Finally, each event is sent according to the output plugin configuration. If no codec is defined, the default is used for the specific input or output plugin. Multiple plugins can be used in parallel.

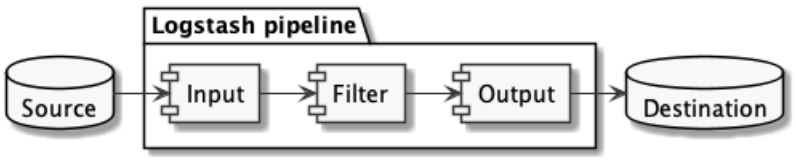


Figure 5.2: Logstash pipeline

A major strength of Logstash is that no programming is required. Through the combination of existing plugins, data pipelines can be created just by editing configuration files. Another advantage is that some plugins do optimizations like bulk insertions for better performance and handle failures with a retry mechanism. When the amount of data that needs to go through a single Logstash instance is too big, it is even possible to scale horizontally

and distribute the load among different Logstash workers, with many possible alternative architecture patterns [45].

### 5.2.2 Pipeline configuration

To create the data pipeline, two Logstash pipelines are required, one to extract the internship proposals and other for the search results, as different SQL queries have to be made. To create these pipelines, the file */usr/share/logstash/config/pipelines.yml* must include the name of the pipelines and their configuration path.

```
- pipeline.id: searches-pipeline
  path.config: "/usr/share/logstash/pipeline/searches.conf"

- pipeline.id: proposals-pipeline
  path.config: "/usr/share/logstash/pipeline/proposals.conf"
  pipeline.workers: 1
```

Listing 5.2: Pipelines definition

In addition, the number of worker processes can be changed. The default value is equal to the number of CPU cores. As the proposals pipeline contains many to many joins, multiple events are generated for the same proposal, so the number of worker threads needs to be limited to one because of a limitation in the aggregation filter[4].

The configuration for the searches pipeline is available in code Listing 5.3. The JDBC input filter is required to establish the connection to the source database, which is MariaDB. To send the data to the final destination, the Elasticsearch output plugin is set, including the index name.

No filter plugins are needed since no transformations are done and environment variables are used for dynamic configuration that change in development and production, like the database connection details. As the search request reports have daily granularity, this was scheduled to run every day at 3AM.

It is necessary to specify a column that is used to keep track of the last queried record, along with a file to store the latest value. A special variable can then be used in the SQL statement to resume the pipeline.

```
input {
  jdbc {
    jdbc_driver_class => "org.mariadb.jdbc.Driver"
    jdbc_connection_string => "${JDBC_URL}"
    jdbc_user => "${JDBC_USERNAME}"
    jdbc_password => "${JDBC_PASSWORD}"
    schedule => "0 3 * * *" # Every day at 3:00
    lowercase_column_names => false
    statement => "SELECT id, `date`, nResults, `query`
      FROM SearchRequest
      WHERE `query` <> '' # Exclude empty queries
      AND id > :sql_last_value"
    use_column_value => true
    tracking_column_type => "numeric"
    tracking_column => "id"
    last_run_metadata_path => "${HOME}/.logstash_jdbc_last_run_searches"
  }
```

[4] https://discuss.elastic.co/t/how-to-use-aggregate-filter-with-multiple-workers/157621

```
}
output {
  elasticsearch {
    hosts => ["${ELASTIC_URL}"]
    index => "searches"
    document_id => "%{id}"
    user => "${ELASTIC_USERNAME}"
    password => "${ELASTIC_PASSWORD}"
  }
}
```

Listing 5.3: Searches pipeline

The proposals pipeline is similar, but an aggregation filter is needed to combine multiple rows in lists, caused by many to many joins. Additionally, a mutate filter creates an *_all* field by concatenating the text of other fields to improve full-text searches over multiple fields. The scheduling period was set to every minute, as new internships must be available without much delay and no performance impact was observed.

## 5.3 Search engine

To improve search results on Praxis, a search engine was built taking into consideration the findings in the exploratory data analysis of the Praxis dataset. Elasticsearch is the selected search engine because of its popularity and features. In addition, it can be used as an analytical database. As Elasticsearch should not be exposed to the internet for security reasons, a custom web API with the single purpose of routing search queries to Elasticsearch was developed in Go.

### 5.3.1 Elasticsearch

Elasticsearch is a distributed open source search and analytics engine developed in Java on top of Lucene[5]. It provides a JSON web API to manage the distributed system, index documents and execute queries. When a document is stored, it is indexed and searchable in near real-time. Unlike relational databases, Elasticsearch is a document-oriented database, so data should be denormalized and there are no relations, constraints and transactions. Every document has a unique identifier which will be automatically generated if not specified.

Each index has a name and mappings, which defines how documents, and their fields, are stored and indexed. Additional settings like the number of primary and replica shards can be set, which by default is one for both. Elasticsearch can automatically update an index mapping when documents with new fields are added. In some cases, this dynamic mapping is not able to infer the type of each field correctly. For these situations, the mapping has to be explicitly defined before a document is indexed with an incorrect field type, as updates to existing fields are not supported, requiring the creation of a new index and re-indexing all data [46].

By default, strings are indexed as a text field for full-text search and as a keyword field for sorting or aggregations. Text analysis is performed by an analyzer when indexing or searching text fields to return not only exact matches, but also relevant results [47]. Therefore, the configuration of analyzers, which convert unstructured text into a structured format

[5] https://lucene.apache.org

optimized for search, is important to tune the search engine. An analyzer contains three steps: character filters, tokenizers and token filters, as demonstrated in Figure 5.3. Additional analysis plugins[6] with support for lemmatization, phonetic similarity or optimized for different languages and purposes can be installed.

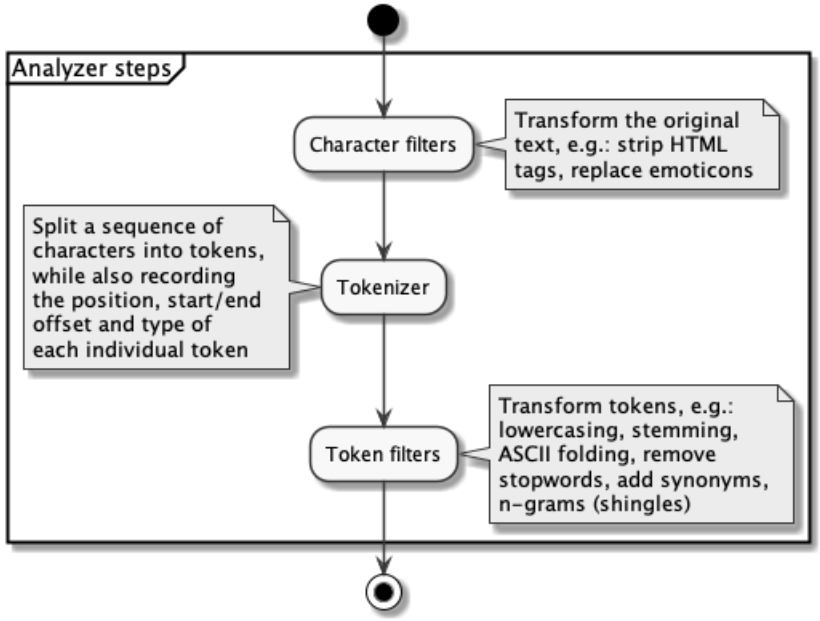


Figure 5.3: Elasticsearch analyzer steps [48]

When a full-text search query is made with multiple terms, an analyzer splits the string into multiple tokens. The search query is then converted into a boolean query to retrieve matching documents and a similarity model ranks them according to relevance. In recent years, Lucene and Elasticsearch switched their default similarity model from a TF-IDF model, to a BM25 model, as studies show improvements on the relevance of search results using the probabilistic technique [49].

Besides searches, an advantage of the powerful Elasticsearch query domain-specific language is the ability to do various kinds of analytical queries using the aggregations framework[7]. It is possible to calculate percentiles, create date histograms, get the most significant terms, among others. Multiple statistics can be obtained with a single HTTP request. Certain queries may have a big performance impact, so caching and sampling is supported by Elasticsearch.

To achieve scalability and high availability, Elasticsearch indexes are divided into two types of shards, primaries and replicas, which are split among the different nodes in the Elasticsearch cluster. Only primary shards accept document indexing requests, so each document belongs to a primary shard, being then replicated to replica shards, and unlike the latter, the number of primary shards is fixed when the index is created [50]. On a cluster with sufficient nodes, improving search speeds for an index can be done by increasing the number of replicas, however, for write-heavy environments, more primary shards are needed. Shard balancing is done automatically by the cluster when nodes are added or removed and every node can

[6] https://www.elastic.co/guide/en/elasticsearch/plugins/current/analysis.html
[7] https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html

forward client requests to the appropriate node. By default, each index is created with one replica shard, so at least two nodes are needed to keep the cluster in a healthy state.

A single Elasticsearch node can have many roles assigned: master-eligible, data, ingest or coordinating, and on large clusters it is recommended to separate each node's role [51]. The master node is responsible for coordinating the cluster and three or five master nodes is a common configuration to maintain a quorum and be able to elect a new leader in case of failure [52]. Data nodes contain the shards, so they handle searches, aggregations, insertions, updates and deletions. Coordinating nodes act as load balancers, routing requests to the correct nodes. Lastly, ingest nodes deal with document pre-processing operations.

### 5.3.2 Index configuration

As there are two different types of documents that need to be indexed, searches and internships, two indexes need to be created. The searches index is simple, so the default index settings and implicit mappings created by Elasticsearch do not have to be changed, but an explicit mapping is required for the internships index.

To optimize search results, the internships index configuration took into account the findings from the exploratory data analysis and the options available on Elasticsearch. An HTTP request must be made to create the index correctly, with different analyzer settings for indexing and searching. The implemented analyzer configuration is defined in Listing 5.4.

```
{
  "analysis" : {
    "filter": {
      "praxis_synonyms": {
        "type": "synonym_graph", "synonyms_path": "synonyms.txt"
      },
      "praxis_stopwords": {
        "type": "stop", "stopwords_path": "stopwords.txt"
      },
      "plural_stemmer" : {
        "type" : "stemmer", "name" : "minimal_english"
      },
      "no_stem": {
        "type": "keyword_marker", "keywords_path": "no_stem.txt"
      }
    },
    "analyzer": {
      "default": {
        "type": "custom", "tokenizer": "standard",
        "char_filter": ["html_strip"],
        "filter": ["lowercase", "apostrophe", "asciifolding",
          "no_stem", "kstem", "plural_stemmer", "praxis_stopwords"]
      },
      "default_search": {
        "type": "custom", "tokenizer": "standard",
        "filter": ["lowercase", "apostrophe", "asciifolding",
          "no_stem", "kstem", "plural_stemmer", "praxis_stopwords",
          "praxis_synonyms"]
      }
    }
  }
}
```

Listing 5.4: Internships index configuration

As internships may contain HTML tags, they are stripped in the index analyzer. The standard tokenizer works well with many languages, so it is applied on both analyzers. In addition, all terms are converted to lower case, everything after apostrophes, including the apostrophe itself is removed and diacritics are replaced with ASCII equivalents. Afterwards, stemming is applied, however some words are excluded as they should not be stemmed. Stop words are also removed. A list of synonyms is maintained and used at query time to improve results.

A light stemming approach was taken with the Krovetz Stemmer (Kstem) and the minimal plural stemmer, as not all plurals were being removed with the first. Kstem is a hybrid algorithm that combines morphological rules and dictionary lookups to avoid incorrect stemming [53]. More aggressive stemmers like the Porter2 stemmer tend to increase recall at the cost of precision, which may result in a worst *F1* score.

### 5.3.3 Search API

Boolean search queries are typically used on Elasticsearch to retrieve documents that have to match more than one condition, by combining multiple clauses. The *filter* and *must_not* clauses are used to select matching documents and do not affect the relevance score. To order the documents by relevance, the *must* or *should* clauses are required, with the former excluding any document that does not match the specified conditions. Each clause can have multiple conditions.

By default, on full-text searches, the search terms are converted to a boolean query with the *should* clause, where each term is separated by an *OR* condition. To control the search engine precision, a *minimum_should_match* condition is used to specify the absolute number, or percentage, of minimum terms that should be present to retrieve a document. It is possible to specify different minimum values depending on the size of the query.

Typographical errors cause queries to retrieve no results, which is often undesired. To deal with this, similar words can be suggested to the user using the Elasticsearch's term suggester, or matched automatically using the *fuzziness* attribute. A metric commonly used to match similar words is the *Damerau–Levenshtein edit distance*, which is calculated by the minimum number of single-character changes, or transpositions of two adjacent characters, to transform a word into another. Damerau stated that over 80% of spelling errors are within an edit distance of one [54].

A full-text search can be matched against one or more fields and relevance can be boosted for each field using *multi_match* conditions. The creation of an additional field, at index time, that contains the text of other fields is also a valid and documented approach [55]. It is useful when combined with the *minimum_should_match* condition, which can only be used on a single field. A *must* clause can be used to match a query on the big field, which contains all the text, and a *should* clause can then tune relevancy based on the other fields, so that a term appearing in a title is more relevant than a description.

Taking into consideration the existing data and analysis made, the format of search queries is specified in Listing 5.5, which contains a boolean query with multiple terms to retrieve valid internships at a certain date, ordered by relevance. Some fields are more relevant than others, so boosting is applied. An additional *_all* field was required, which has the text of the other fields, in a bigger field. If the query has three terms, all terms should be in the *_all* field, otherwise 75% of terms must be present. A maximum edit distance of one is allowed after the third character of each term, as it fixes most typographical errors and has good performance.

```
POST http://localhost:9200/proposals/_search
Content-Type: application/json
{
  "query": {
    "bool": {
      "must": [{
        "match": {
          "_all": {
            "query": "uk software engineer",
            "fuzziness": "1", "prefix_length": 3,
            "minimum_should_match": "3<75%"
          }
        }
      }],
      "should": [{
        "multi_match": {
            "query": "uk software engineer",
            "fields": ["title^8", "description^2", "studies.*",
              "country^16", "city^16", "languages^16", "orgName"],
            "fuzziness": "1", "prefix_length": 2,
            "tie_breaker": 0.3
        }
      }],
      "must_not": [
        { "term": { "deleted": true } },
        { "term": { "visible": false } },
        { "term": { "status": "assigned" } }
      ],
      "filter": [
        {"range" : { "submitDate" : { "lte" : "2019-01-01" } } },
        {"range" : { "validTo" : { "gte" : "2019-01-01" } } }
      ]
    }
  }
}
```

Listing 5.5: Search query example

The values defined in the query, to control precision and relevancy, were set based on the observation of results for common search queries. To properly tune relevance over time, instrumentation should be in place to monitor metrics such as the number of clicks on the top result(s), how often users they click a result and go back, how many queries are made per user, and so on [56]. Recent internships could rank higher by adding a condition to the *should* clause using a *boost* parameter.

There is a lot of flexibility when it comes to relevance tuning as different applications have different requirements, so even custom plugins with ranking scripts can be developed. As the number of matches increases, their order becomes more important, especially on the first ones.

Recent developments in Elasticsearch allow semantic search at scale with the introduction of the *dense_vector* field, which can be used to store text embeddings, and the ability to calculate the cosine similarity between these vectors [57]. Text embeddings for documents and queries can be generated from pre-trained models like BERT and then compared. This is still an area of active research, but *Google* is already implementing such techniques in *Google Search* with good results [58].

A web API was developed to provide a secure interface to the Elasticsearch cluster. Figure 5.4 shows the flow of a search request through the system, which results in one or two HTTP requests. If the search query does not retrieve any internships, alternative terms are suggested using Elasticsearch's term suggester and the significant text aggregation.

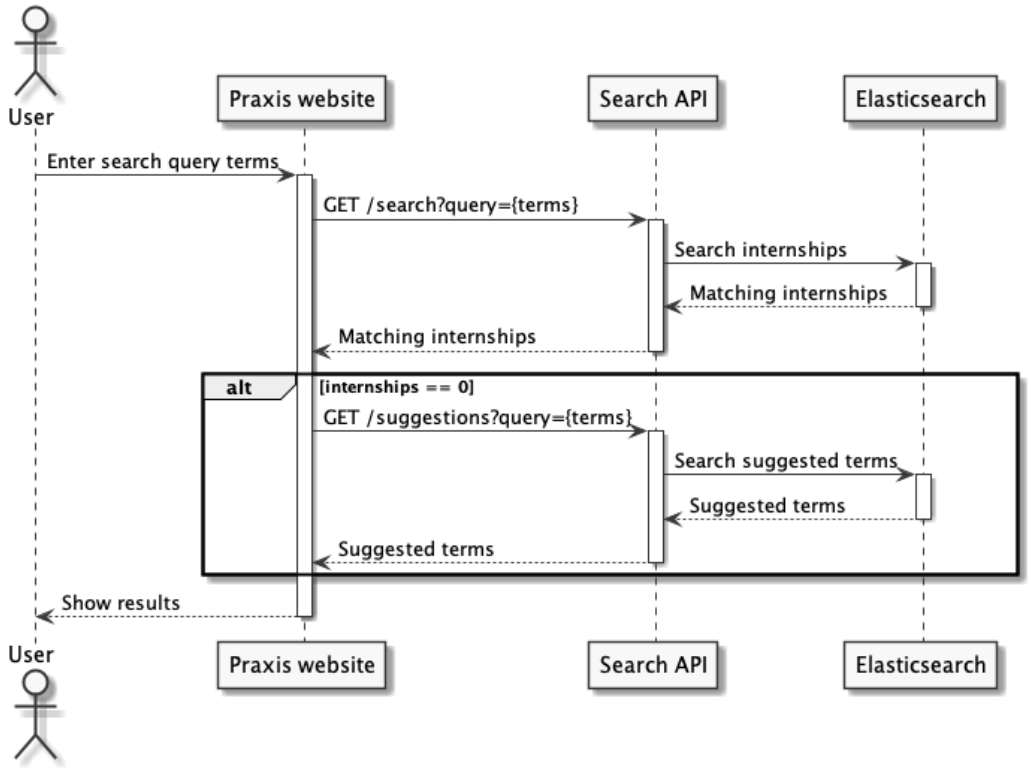


Figure 5.4: Search sequence diagram

The selection of Go as a programming language to develop this web API proved to be a good decision. Out of the box, the standard library included everything required to create HTTP clients and servers, without too much code and with intuitive abstractions, so no external libraries were required. Additionally, static typing helped ensure the code correctness during compile time and lightweight binaries could be generated for multiple operating systems, without any dependencies. These binaries are small, have fast start-up times and good runtime performance without consuming a lot of memory or CPU resources.

## 5.4 Web dashboard

The creation of a web dashboard is valuable for Praxis administrators, as they need a solution to visualize the structured and non-structured data on Praxis, to extract useful information about the market and the usage of the website. The main objective of the dashboard is to automate the analysis made with the R language in the previous chapter, for any given time frame. R was chosen due to having many available text mining libraries and for being a productive language to conduct exploratory data analysis.

### 5.4.1 Zeppelin

To enable easy data exploration on the web, the selected technology was Zeppelin, an open source web-based notebook for interactive data analytics. Each dashboard is a notebook, or note, that contains blocks of code called paragraphs, which can be executed individually. Text mining techniques can be applied, as multiple programming environments are supported, and software libraries can be installed. In addition, it allows the collaboration of multiple users, with configurable environment isolation.

An overview of the Zeppelin architecture is presented in Figure 5.5. The frontend web application communicates with the backend through a REST API and WebSockets, allowing the connection to stay open, so updates from the server are sent in real time to the frontend, giving users instant feedback on the execution of notes. Interpreters for various environments, such as R and Python, can be installed on the server. Zeppelin is developed in Java and is able to interact with other applications associated with the Big Data ecosystem, such as Spark[8], an open source cluster-computing framework, which is often used to scale heavy computational tasks horizontally [59].

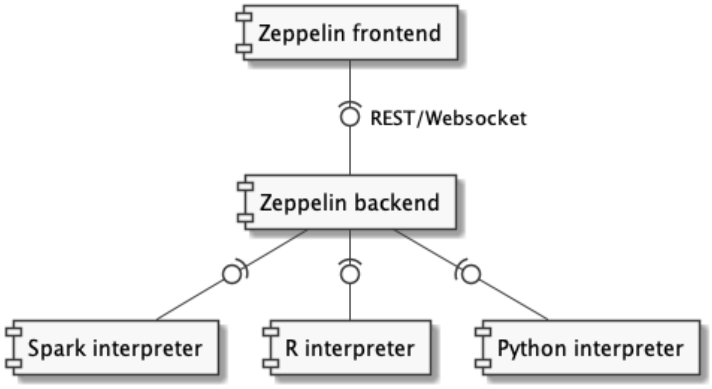


Figure 5.5: Zeppelin architecture

One or more Java Virtual Machine (JVM) processes are responsible for the communication with interpreters, and different ones may be used in the same notebook. Interpreter initialization is lazy, so they are not started until code execution is requested, and after a specified amount of idle time, the interpreter is stopped. Variables defined in one notebook can be accessed in another if using the same session. There are three modes to run interpreter processes:

- **Shared** - a single JVM process and session
- **Scoped** - a single JVM process and one session per notebook
- **Isolated** - a separate JVM process per notebook

The default *shared mode* achieves higher performance, but variables are shared across all notebooks and if one notebook causes the interpreter process to die, all notebooks are affected. The *scoped mode* does not share variables between notebooks, as a session is created for each. Lastly, the *isolated mode* is more computationally expensive, as it creates a separate JVM process per notebook, with the corresponding interpreter, but sessions are not shared, and other notebooks do not affect each other if an interpreter dies [60].

[8] https://spark.apache.org

When running Zeppelin in a multi-user environment, interpreter isolation becomes more important, as users do not want to overwrite each other's variables, and a bad query can cause the interpreter to become unresponsive. By default, notebooks are executed by the same user as the Zeppelin process, which may be a security risk, since users could read sensitive files owned by Zeppelin. As such, Zeppelin also provides environment isolation per user with the ability to run the interpreter as the user executing the notebook.

Authentication and authorization is based on the Shiro[9] security framework. Supported mechanisms include the usage of a local file, LDAP, Active Directory or JDBC. It can also handle role assignment and authorization based on URL rules. There are four types of notebook permissions assignable to users or roles:

- **Owners** - can change permissions of the notebook
- **Writers** - can update code in the notebook
- **Runners** - can execute the notebook
- **Readers** - can view the notebook

A Zeppelin notebook is composed of one or more paragraphs. Each paragraph contains a code section, with the analysis source code, and a result section, with the analysis output. A JSON file is generated for each notebook, containing the code, last run results and some metadata. When a paragraph result contains images, they are converted to Base64 format and the output is converted to HTML. A report mode, which only shows results and dynamic forms is available to allow less technical users to interact with the system. By printing tables, Zeppelin is capable of plotting bar charts, line charts, area charts and scatter plots.

Other notable Zeppelin features include a cron scheduler to run notebooks at specific time intervals, visualization plugins using Helium[10] packages, and the ability to use local Git repositories to version control the notebooks. Other storage options such as GitHub and MongoDB are available.

Although Zeppelin is a very useful tool, some issues were identified. On high load, the interpreters can crash, and as a result, the entire notebook needs to run again, since variables are lost, making it not a very good solution for intensive data pipelines, as also reported by other researchers [61]. There are also limitations related with authorization, like the inability to share dashboards or visualizations with unauthenticated users, and it is not possible to create read-only users, who cannot create notebooks. An alternative to share reports is to use the scheduler to periodically execute a notebook that generates a static webpage or pdf and uploads it to a destination.

### 5.4.2 Libraries and configurations

A Zeppelin Docker image is distributed on DockerHub, however it does not contain all the required software libraries used in the analysis. These libraries included algorithms for text vectorization, clustering and visualization. A new image was created with the Dockerfile in Listing 5.6 and uploaded to the registry, as additional configurations and R packages were required.

```
1 FROM apache/zeppelin:0.9.0
2
```

[9] https://shiro.apache.org
[10] https://zeppelin.apache.org/helium_packages.html

```
3  USER root
4
5  # update libraries, like R to 3.6, add sudo to run isolated interpreters
6  RUN sed -i "s/deb http:\/\/cran.rstudio.com\/bin\/linux\/ubuntu xenial
       \//deb https:\/\/cloud.r-project.org\/bin\/linux\/ubuntu xenial-
       cran35\//" /etc/apt/sources.list && \
7    apt-get update && \
8    apt-get upgrade -y --allow-unauthenticated r-base r-base-dev && \
9    apt-get remove -y 'r-cran-*' && \
10   apt-get install -y --no-install-recommends libgsl-dev sudo && \
11   echo "zeppelin ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers && \
12   rm -rf /var/lib/apt/lists/*
13
14 # additional required R packages
15 RUN R -e "update.packages(checkBuilt=TRUE, ask=FALSE)" && \
16   R -e "install.packages('elastic')" && \
17   R -e "install.packages('tidyverse')" && \
18   R -e "install.packages('udpipe')" && \
19   R -e "install.packages('text2vec')" && \
20   R -e "install.packages('tidytext')" && \
21   R -e "install.packages('wordcloud')" && \
22   R -e "install.packages('ggraph')" && \
23   R -e "install.packages('skmeans')" && \
24   R -e "install.packages('Rtsne')" && \
25   R -e "install.packages('IRkernel');IRkernel::installspec(user=FALSE)"
26
27 # create a python symbolic link to allow R to run isolated
28 RUN ln -sf /opt/conda/bin/python /usr/bin/python
29
30 # temporary file must be writable by all users
31 RUN mkdir -p /zeppelin/figure && \
32   touch /zeppelin/figure/unnamed-chunk-1-1.png && \
33   chmod 666 /zeppelin/figure/unnamed-chunk-1-1.png
34
35 USER zeppelin
```

Listing 5.6: Zeppelin Dockerfile

Other configurations were not set in the Docker image, but through the use of environment variables and file system volumes, which can be dynamically changed as needed. Regarding authorization, authenticated users can have two roles assigned, admin and analyst. Admins are able to change system settings through the Zeppelin web administration interface and analysts have permissions to edit notebooks and run or restart interpreters.

Since only a few people need access to the system, instead of having to rely on an external authentication service, a local file with hashed passwords was the selected mechanism to handle user accounts. To prevent other users from reading this file, its read permissions were restricted to the *zeppelin* user. Additionally, isolation was setup per user and note so that interpreter processes are owned by the respective user and a new one is started for every note. Despite having a performance cost, this increases security, stability and reduces variable conflicts in different dashboards.

### 5.4.3 Visualizations

Most visualizations from the exploratory data analysis were replicated in the final dashboard, with a few additional features. As visible in Figure 5.6, global filters were added to the top

of the dashboard, to allow users to control the time range. When possible, the native charts were favoured over images generated in the interpreter as they are interactive and it is easier to change their chart type.

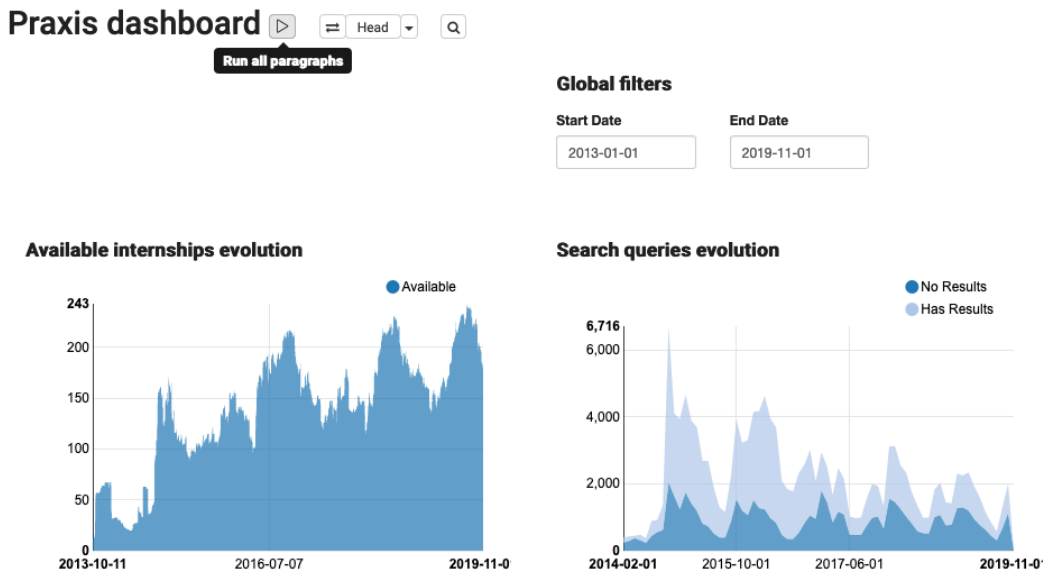


Figure 5.6: Dashboard global filters and report mode

Additional local filters were added to the dashboard as needed, with sensible defaults, so that manual input is not required, as demonstrated in Figure 5.7. Unfortunately, Zeppelin does not recognize dependencies between paragraphs, so they must be executed in order, otherwise variables may be undefined.

Figures of the full dashboard are available in appendix A.

## 5.5 Summary

The implementation technical details were described and the decisions made were justified in this chapter. With the selected technologies, Docker, Logstash, Elasticsearch, Go and Zeppelin, in a short amount of time, a viable solution was developed to improve search results on Praxis and another to help administrators analyse the available data on the platform and extract actionable information.


Figure 5.7: Topic modelling paragraphs with local filters and code edit mode

# Chapter 6

# Deployment and Evaluation

This chapter reports the deployment procedures, which aimed to be reproducible, with as minimal human interaction as possible, following the current best practices. Additionally, the system is evaluated to verify if the original goals were achieved.

## 6.1 Configuration management

In recent years, virtualization and cloud computing led to an increase in the number of servers managed by system administrators [62]. To help manage this complexity, and ensure the environment is reproducible, infrastructure and related procedures can be defined as code and placed into version control. Many alternative technologies are available for configuration management, such as Ansible[1] or SaltStack[2].

### 6.1.1 Ansible

Ansible is an open source agentless automation engine for cloud provisioning, configuration management and application deployment, which executes code over Secure Shell (SSH) by default [63]. Most popular competing alternatives require software agents to be installed on the controlled servers, which leads to better performance, but a more complex architecture, as a centralized server is required, so that agents can pull configurations [64].

The main reason to choose Ansible was its simplicity. Being agentless, additional software does not need to be installed on the servers, as long as Python is already present. By using SSH, no additional ports have to be opened and no infrastructure changes have to be made, as updates can be pushed as needed, from other machines, if a network connection is established. The desired state of the system is written in easy and descriptive YAML files called playbooks.

To understand Ansible, the following concepts are helpful:

- **Control node** is a machine with Ansible installed to run ad-hoc commands or playbooks on the target hosts.
- **Managed nodes** are the servers managed with Ansible, also referred to as hosts.
- **Inventory** is a list of managed nodes. It contains information such as IP addresses, groups and optional variables for each host. This list can be dynamic.

[1] https://www.ansible.com
[2] https://www.saltstack.com

- **Modules** are the code blocks executed by Ansible to perform operations on the hosts. Custom modules can be developed to add new features.
- **Tasks** are calls to modules, including the associated arguments. A task can invoke a single module and an ad-hoc command can execute a single task.
- **Handlers** are tasks that are executed when notified of changes by other tasks.
- **Roles** are reusable components consisting of tasks, variables, handlers, or even modules, which can be included in playbooks. Ansible Galaxy[3] is a popular website where roles are shared by the community.
- **Playbooks** are ordered lists of tasks, which are assigned to hosts. Variables and Jinja templates can be used. They should be idempotent, so that running the same playbook multiple times, leaves the system in the same state as running it just once.

A local Docker container with Ansible was used to deploy the applications. The configuration files and environment variables were stored in files that were mapped to the container through volumes.

### 6.1.2 Deployment playbooks

Four playbooks were created to install the Elasticsearch, Logstash, the Search API and Zeppelin applications as Docker containers. They all share similarities like the use of the Docker module to manage the life cycle of the containers, as demonstrated in Listing 6.1. When the configuration files or environment variables are updated, the container must be restarted. In some cases, setup scripts are required to guarantee a good initial state. After a container starts, the application can take some time to process HTTP requests, so a few retries are needed.

```
---
- hosts: elasticsearch-hosts
  vars:
    analysis_ansible: /home/ansible/elasticsearch/analysis/
    analysis_elastic: /opt/praxis/volumes/elasticsearch/analysis/

  tasks:
    - import_tasks: tasks/docker_install.yml
      become: yes

    - name: Create volumes directory structure
      file:
        path: "{{ analysis_elastic}}"
        state: directory
        recurse: yes

    - name: Copy elasticsearch analysis files
      copy:
        src: "{{ analysis_ansible }}"
        dest: "{{ analysis_elastic }}"
      notify:
        - Restart elasticsearch
        - Create index

    - name: Make setup script executable
```

[3] https://galaxy.ansible.com

```
26        file:
27          path: "{{ analysis_elastic }}/index-setup.sh"
28          mode: "0700"
29
30      - name: Copy elasticsearch env file
31        copy:
32          src: templates/elasticsearch.env
33          dest: /opt/praxis/elasticsearch.env
34          mode: "0600"
35
36      - name: Start elasticsearch container
37        docker_container:
38          name: elasticsearch
39          image: elasticsearch:7.4.2
40          memory: 2048M
41          state: started
42          restart_policy: unless-stopped
43          env_file: /opt/praxis/elasticsearch.env
44          ports:
45            - "9200:9200"
46          volumes:
47            - /opt/praxis/volumes/elasticsearch/data:/usr/share/
    elasticsearch/data
48            - /opt/praxis/volumes/elasticsearch/analysis:/usr/share/
    elasticsearch/config/analysis
49        register: elasticsearch_container_started
50
51    handlers:
52      # container is already restarted when env file changes
53      - name: Restart elasticsearch
54        docker_container:
55          name: elasticsearch
56          restart: true
57        when: elasticsearch_container_started.changed == False
58
59      - name: Create index
60        shell: "{{ analysis_elastic }}/index-setup.sh"
61        register: result
62        until: result is not failed
63        retries: 10
64        delay: 3
```

Listing 6.1: Elasticsearch deployment playbook

The developed playbooks are idempotent. Running them multiple times will always leave the system in the final desired state, as it detects that no changes are needed. When there is an update, only the required tasks are executed.

To prevent high resource usage on the host, the maximum RAM was set for each container, but CPU or I/O usage limitations could also be defined. In the future, if one host is not enough, the containers can be deployed to different machines. Kubernetes[4] is an emerging technology to deploy, scale and manage containerized applications.

[4] https://kubernetes.io

## 6.2 System evaluation

All the proposed functional requirements were implemented. To evaluate how well the developed system performs, metrics were obtained to check if the non-functional requirements were achieved.

### 6.2.1 Search engine performance

The goals of the new search engine were to improve the search results and keep the response times under 1 second for most users, in normal load and network conditions. The *F1 score* is the chosen metric to be optimized to achieve a good trade-off between precision and recall.

To improve the *F1 score*, alternative English stemmers were tested, as most internships are written in English. The existing search engine does not use stemming. It was found that less aggressive stemmers like Kstem improve this metric more than the Porter2 stemmer or no stemming, as demonstrated in Table 6.1. Only recall was slightly better with Porter2, at the cost of precision. A set of 30 representative queries were made on different indexes, containing the internships available on the date with the highest count.

Table 6.1: Search engine stemming comparison

| Stemmer | Mean results | Mean precision | Mean recall | Mean F1 score |
|---|---|---|---|---|
| None | 5.87 | 0.77 | 0.55 | 0.61 |
| Kstem | 12.10 | **0.84** | 0.86 | **0.80** |
| Porter2 | 18.93 | 0.68 | **0.90** | 0.70 |

When no matches were found, the precision was considered to be zero, as the queries always had possible matches. The mean *F1 score*, often referred to as macro *F1*, was calculated via arithmetic mean of the individual *F1 score* of each query, instead of calculating the harmonic mean of the mean precision and recall, as suggested by recent literature for being more robust [65].

By combining a light stemmer, such as Kstem, with a list of synonyms and a list of words that should not be stemmed, these metrics can be further improved. In many cases, removing suffixes like plurals or *"ing"* improves results, but some words may need to be excluded, such as *"accounting"*. With lighter stemmers, smaller dictionaries and stemming exclusion lists have to be maintained.

Regarding fuzzy searches, they never decrease recall but may decrease overall precision. Words like *medic* are only 1 transposition away from *media*, resulting in false positives, although exact matches are ranked higher than fuzzy matches. Unlike stemming, fuzziness can be controlled at query time, so Elasticsearch's term suggester can be used to suggest corrections when there are no matches.

To check the response times of the Search API, load tests were made using JMeter[5], an open source Java-based load testing desktop application, simulating normal and peak traffic. During a period of 30 seconds, distinct numbers of concurrent users were simulated by sending different search queries from a local machine to a virtual server hosted in the cloud, with an Intel Xeon CPU with 2 virtual cores, 8GB of RAM and 32GB SSD. The results are presented in Table 6.2, which shows that all requests take less than a second.

[5] https://jmeter.apache.org

Under normal conditions there are less than 10 searches per second in Praxis, so in 99% of the cases, the Search API is expected to take less than 148 milliseconds.

Table 6.2: Load testing results in periods of 30 seconds

| Users/s | Samples | Mean | Std. dev. | Median | P90 | P99 | Max |
|---|---|---|---|---|---|---|---|
| 1 | 487 | 60 ms | 15.41 ms | 58 ms | 67 ms | 124 ms | 173 ms |
| 10 | 4289 | 67 ms | 19.24 ms | 62 ms | 89 ms | **148 ms** | 204 ms |
| 50 | 6864 | 214 ms | 58.79 ms | 207 ms | 288 ms | 401 ms | 666 ms |
| 100 | 6409 | 461 ms | 94.55 ms | 452 ms | 583 ms | 631 ms | 886 ms |

Throughput was reduced when going from 50 to 100 concurrent requests, as there is more load on Elasticsearch, and it takes longer to finish the requests. If the traffic on Praxis increases significantly, response times can be improved with an in-memory cache as many searches are repeated or by adding more replicas to the Elasticsearch cluster.

### 6.2.2 Web dashboard usability survey

Usability is defined in ISO 9241 Part 11 as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [66]. The users of the implemented system are expected to have some experience with data analysis and their main goal is to extract information about internships and searches made on Praxis to identify market gaps.

To evaluate the dashboard usability, a survey was distributed to 15 colleagues with engineering and business backgrounds. A guide was also included, with instructions to connect to the dashboard and an introduction to Zeppelin so they could use the system more effectively. The survey contained the following statements:

1. The dashboard is aesthetically pleasing.
2. The dashboard is easy to use.
3. The system feels fast and responsive.
4. The available information provides a good overview of the internships market supply and demand.
5. Changing visualizations and doing different analyses does not require much effort.

Participants could evaluate these statements on a five-point Likert scale, widely used in surveys [67], to specify their agreement level. Survey results are presented on Table 6.3.

Table 6.3: Survey results

| Question | Strongly disagree | Disagree | Neutral | Agree | Strongly agree |
|---|---|---|---|---|---|
| **1** | 0 | 2 | 4 | 8 | 1 |
| **2** | 0 | 0 | 3 | 9 | 3 |
| **3** | 0 | 0 | 2 | 8 | 5 |
| **4** | 0 | 0 | 3 | 8 | 4 |
| **5** | 0 | 3 | 2 | 8 | 2 |

Regarding the first and second statements, not many users selected "Strongly agree". They suggested alternative BI tools, such as Kibana[6] and Power BI[7], that were easier to use and provided better looking visualizations than Zeppelin. However, they are mostly used to analyse structured data, so text mining use cases like document clustering or topic modelling are not supported.

Most users agreed or strongly agreed that the system was fast. When doing a full run with an interval of years, it takes a few seconds to load the data and create clusters of internships based on their text, as it is computationally intensive. Users considered the system was generally responsive with nearly instant feedback on most tasks.

A large percentage of users confirmed that the information presented allowed them to have a good overview of the market supply and demand by observing the most searched keywords and comparing the different internships clusters. The last question shows that some people need help changing visualizations and the reason given was their unfamiliarity with programming or the R language, so they were not comfortable editing the code to create their own ad-hoc reports.

Although the sample of users is small, the majority agrees with all the statements. As such, the usability goals are met. Specific user interface improvements were suggested, like changes to the home page and keeping the toolbar always visible in the report mode, which would have to be implemented in the Zeppelin source code

## 6.3 Summary

By using Ansible as a deployment automation tool, the entire process of deploying the project to a production environment is faster and less error-prone.

The deployed system was evaluated to assess if the functional and non-functional requirements were met. There was an improvement on the F1 score of the new search engine and responses were always under 1 second even under load. Users demonstrated satisfaction with the web dashboard through a usability survey.

[6] https://www.elastic.co/kibana
[7] https://powerbi.microsoft.com

# Chapter 7

# Conclusion

This chapter summarizes the accomplished goals, limitations of the developed solution and future improvements.

## 7.1 Accomplished goals and contributions

The developed project achieved the defined goals set initially. The creation of a web dashboard that is simple to use and easy to extend allows Praxis administrators to quickly have access to the available data and identify market trends. The studied text mining techniques achieved good results in extracting useful information from the description of internships.

Through the analysis made, some market gaps and issues with the existing search engine were identified. A new search API was developed to fix those issues and increase the number of correct matches between searches and internships. The use of stemming, spelling correction and synonyms were important to improve results.

Many alternative solutions were compared for each component of the project. By using existing technologies instead of developing everything from scratch, development and maintenance effort is reduced. With the automation of the deployment process and the use of containers to ensure reproducibility, future changes are easier to make, and problematic updates can be debugged or reverted faster.

Furthermore, during the development phase, a page with incorrect documentation was identified in the Elasticsearch documentation. A pull request[1] was submitted and merged to the master branch, thus contributing to the open source ecosystem.

## 7.2 Limitations and future work

The technology chosen to create dashboards, Zeppelin, has some limitations. The most notable one is the inability to share a dashboard or a visualization with unauthenticated users. Usability improvements like additional form input controls and updating the entire dashboard when a global filter is changed without explicitly having to click another button would be appreciated. In the future, the dashboard authentication mechanism should use an external identity provider so that when a user needs to be added or removed, the Zeppelin process does not need to be restarted.

A task out of scope for this project was the integration of the improved search engine with the Praxis website. As a simple JSON-based web API is exposed, no significant development

[1] https://github.com/elastic/elasticsearch/pull/59834

effort is expected. By continuously monitoring the searches and results, adjustments can be made to the search engine over time like adding new synonyms and improving the relevance of results. Logging additional online metrics, like the number of clicks on the first page, or the number of successive searches without selecting a result, would be helpful for further research, as they indicate how relevant search results are.

Lastly, it is important to follow the evolution of the state of the art in NLP and text mining algorithms, as they may improve the quality of the information extracted from the available textual data.

# Bibliography


[1] Praxis consortium. *The PRAXIS Network*. url: http://www.praxisnetwork.eu/information/praxisnetwork (visited on 12/27/2019).

[2] Eurostat. *Tertiary education statistics - Statistics Explained*. url: https://ec.europa.eu/eurostat/statistics-explained/index.php/Tertiary_education_statistics (visited on 02/04/2020).

[3] Q. Y. Research. *Recruitment Market: Global Industry to Attain Huge Value of US$ 334.28 Bn by 2025 - QY Research*. url: https://www.prnewswire.com/news-releases/recruitment-market-global-industry-to-attain-huge-value-of-us-334-28-bn-by-2025-qy-research-893326038.html (visited on 02/05/2020).

[4] Josh Bersin. *Google For Jobs: Potential To Disrupt The $200 Billion Recruiting Industry*. url: https://www.forbes.com/sites/joshbersin/2017/05/26/google-for-jobs-potential-to-disrupt-the-200-billion-recruiting-industry/ (visited on 02/05/2020).

[5] Charu C. Aggarwal. *Data Mining: The Textbook*. English. 2015 edition. New York, NY: Springer, Apr. 2015. isbn: 978-3-319-14141-1.

[6] Charu C. Aggarwal and ChengXiang Zhai, eds. *Mining Text Data*. English. 2012 edition. Springer, Feb. 2012.

[7] Vishal Gupta and Gurpreet S. Lehal. "A Survey of Text Mining Techniques and Applications". In: *Journal of Emerging Technologies in Web Intelligence* 1.1 (Aug. 2009), pp. 60–76. issn: 1798-0461. doi: 10.4304/jetwi.1.1.60-76.

[8] Tim Furche et al. "Data Wrangling for Big Data: Challenges and Opportunities". In: *Advances in Database Technology — EDBT 2016*. Advances in Database Technology. University of Konstanz, 2016, pp. 473–478. isbn: 978-3-89318-070-7. doi: 10.5441/002/edbt.2016.44.

[9] Mehdi Allahyari et al. "A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques". In: *arXiv:1707.02919 [cs]* (July 2017). url: http://arxiv.org/abs/1707.02919.

[10] Christopher Manning, Prabhakar Raghavan, and Hinrich Schuetze. *Introduction to Information Retrieval*. 2009. url: https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf.

[11] Osman A. S. Ibrahim and Dario Landa-Silva. "A new weighting scheme and discriminative approach for information retrieval in static and dynamic document collections". In: *2014 14th UK Workshop on Computational Intelligence (UKCI)*. Bradford, UK: IEEE, Sept. 2014, pp. 1–8. isbn: 978-1-4799-5538-1. doi: 10.1109/UKCI.2014.6930160.

[12] Jeffrey Pennington. *GloVe: Global Vectors for Word Representation*. url: https://nlp.stanford.edu/projects/glove/ (visited on 01/21/2020).

[13] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv:1810.04805 [cs]* (May 2019). url: http://arxiv.org/abs/1810.04805.

[14] Ted Kwartler. "Common Text Mining Visualizations". In: *Text Mining in Practice with R*. John Wiley & Sons, Ltd, 2017, pp. 51–83. isbn: 978-1-119-28210-5.

[15] Rosa L. Figueroa et al. "Predicting sample size required for classification performance". In: *BMC Medical Informatics and Decision Making* 12.1 (Feb. 2012), p. 8. issn: 1472-6947. doi: 10.1186/1472-6947-12-8.

[16] Ying Zhao, George Karypis, and Usama Fayyad. "Hierarchical Clustering Algorithms for Document Datasets". In: *Data Mining and Knowledge Discovery* 10.2 (Mar. 2005), pp. 141–168. issn: 1573-756X. doi: 10.1007/s10618-005-0361-3.

[17] D. Sculley. "Web-scale k-means clustering". In: *Proceedings of the 19th international conference on World wide web*. WWW '10. New York, NY, USA: Association for Computing Machinery, Apr. 2010, pp. 1177–1178. isbn: 978-1-60558-799-8. doi: 10.1145/1772690.1772862.

[18] David M. Blei. "Probabilistic topic models". In: *Communications of the ACM* 55.4 (Apr. 2012), pp. 77–84. issn: 0001-0782, 1557-7317. doi: 10.1145/2133806.2133826.

[19] Sholom M. Weiss, Nitin Indurkhya, and Tong Zhang. *Fundamentals of Predictive Text Mining*. English. 2nd ed. 2015 edition. New York, NY: Springer, Oct. 2015. isbn: 978-1-4471-6749-5.

[20] Liu Tie-Yan. *Learning to Rank for Information Retrieval*. London: Springer, 2011. isbn: 978-3-642-14266-6.

[21] S. M. Shafi and Rafiq A. Rather. *Precision and Recall of Five Search Engines for Retrieval of Scholarly Information in the Field of Biotechnology*. url: http://www.webology.org/2005/v2n2/a12.html (visited on 02/16/2020).

[22] Elastic. *Logstash Introduction - Logstash Reference*. url: https://www.elastic.co/guide/en/logstash/current/introduction.html (visited on 02/02/2020).

[23] Solid IT. *DB-Engines Ranking*. url: https://db-engines.com/en/ranking/search+engine (visited on 02/02/2020).

[24] Rafal Kuć. *Solr vs Elasticsearch: Performance Differences & More [2019]*. July 2019. url: https://sematext.com/blog/solr-vs-elasticsearch-differences/ (visited on 02/02/2020).

[25] Google. *Google Trends*. url: https://trends.google.com/trends/explore?date=all&q=%2Fm%2F0h64sgb,%2Fm%2F02qd9s1 (visited on 02/01/2020).

[26] Josh Bressers. *Tips to secure Elasticsearch clusters for free with encryption, users, and more*. June 2019. url: https://www.elastic.co/blog/tips-to-secure-elasticsearch-clusters-for-free-with-encryption-users-and-more (visited on 02/15/2020).

[27] Stefan Hanenberg et al. "An empirical study on the impact of static typing on software maintainability". In: *Empirical Software Engineering* 19.5 (Oct. 2014), pp. 1335–1382. issn: 1382-3256, 1573-7616. doi: 10.1007/s10664-013-9289-1.

[28] The Go Authors. *Frequently Asked Questions (FAQ) - The Go Programming Language*. url: https://tip.golang.org/doc/faq#What_is_the_purpose_of_the_project (visited on 02/06/2020).

[29] Dan Osipov. *The Rise of Data Science Notebooks*. May 2016. url: https://www.datanami.com/2016/05/04/rise-data-science-notebooks/ (visited on 07/18/2020).

[30] Peter Eeles. *Capturing Architectural Requirements*. Nov. 2005. url: http://www.ibm.com/developerworks/rational/library/4706.html (visited on 07/16/2020).

[31] Universal Dependencies. *English Treebank*. url: https://universaldependencies.org/treebanks/en_ewt/ (visited on 05/11/2020).

[32] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation". In: *The Journal of Machine Learning Research* 3 (Mar. 2003), pp. 993–1022. issn: 1532-4435.

[33] Michael Steinbach, George Karypis, and Vipin Kumar. "A Comparison of Document Clustering Techniques". In: (2000), p. 2.

[34] Alexander Strehl, Joydeep Ghosh, and Raymond Mooney. "Impact of Similarity Measures on Web-Page Clustering". In: *Workshop on artificial intelligence for web search (AAAI 2000)* 58 (July 2000), p. 7.

[35] Frizo Janssens, Wolfgang Glänzel, and Bart De Moor. "Dynamic hybrid clustering of bioinformatics by incorporating text mining and citation analysis". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07*. San Jose, California, USA: ACM Press, 2007, p. 360. isbn: 978-1-59593-609-7. doi: 10.1145/1281192.1281233.

[36] A. Zanasi. *Text Mining and its Applications to Intelligence, CRM and Knowledge Management*. WIT Press, Sept. 2007. isbn: 978-1-84564-131-3.

[37] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

[38] Shaun Phillips, Jonathan Sillito, and Rob Walker. "Branching and merging: an investigation into current version control practices". In: *Proceeding of the 4th international workshop on Cooperative and human aspects of software engineering - CHASE '11*. Waikiki, Honolulu, HI, USA: ACM Press, 2011, p. 9. isbn: 978-1-4503-0576-1. doi: 10.1145/1984642.1984645.

[39] Christian Collberg, Todd Proebsting, and Alex M Warren. "Repeatability and Benefaction in Computer Systems Research — A Study and a Modest Proposal". In: (Feb. 2015), p. 68.

[40] Carl Boettiger. "An introduction to Docker for reproducible research, with examples from the R environment". In: *ACM SIGOPS Operating Systems Review* 49.1 (Jan. 2015), pp. 71–79. issn: 0163-5980. doi: 10.1145/2723872.2723882. url: http://arxiv.org/abs/1410.0846.

[41] Docker. *Docker frequently asked questions (FAQ)*. Apr. 2020. url: https://docs.docker.com/engine/faq/ (visited on 04/28/2020).

[42] Antonio Silva, Ana García Hernando, and Mary Luz Mouronte. "Testing the Feasibility of Residential Wireless Interfaces Virtualization". In: *International Journal on Advances in Software* 11 (2018), pp. 65–77.

[43] Robin Moffatt. *No More Silos: How to Integrate your Databases with Apache Kafka and CDC*. Nov. 2019. url: https://www.confluent.io/blog/no-more-silos-how-to-integrate-your-databases-with-apache-kafka-and-cdc (visited on 04/26/2020).

[44] João Duarte. *Logstash 7.0.0 released*. Apr. 2019. url: https://www.elastic.co/blog/logstash-7-0-0-released (visited on 04/26/2020).

[45] Elastic. *Deploying and Scaling Logstash - Logstash Reference*. url: https://www.elastic.co/guide/en/logstash/current/deploying-and-scaling.html (visited on 04/26/2020).

[46] Elastic. *Mapping - Elasticsearch Reference*. url: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html (visited on 05/06/2020).

[47] Elastic. *Text analysis overview - Elasticsearch Reference*. url: https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-overview.html (visited on 05/06/2020).

[48] Elastic. *Anatomy of an analyzer - Elasticsearch Reference*. url: https://www.elastic.co/guide/en/elasticsearch/reference/current/analyzer-anatomy.html (visited on 05/06/2020).

[49] Bharvi Dixit. *Mastering Elasticsearch 5.x*. Packt Publishing Ltd, Feb. 2017. isbn: 978-1-78646-887-1.

[50] Elastic. *Scalability and resilience: clusters, nodes, and shards - Elasticsearch Reference*. url: https://www.elastic.co/guide/en/elasticsearch/reference/current/scalability.html (visited on 05/05/2020).

[51] Elastic. *Node - Elasticsearch Reference*. url: https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-node.html (visited on 05/07/2020).

[52] Amazon. *Dedicated Master Nodes - Amazon Elasticsearch Service*. url: https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/es-managedomains-dedicatedmasternodes.html (visited on 05/07/2020).

[53] Robert Krovetz. "Viewing morphology as an inference process". In: *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '93*. Pittsburgh, Pennsylvania, United States: ACM Press, 1993, pp. 191–202. isbn: 978-0-89791-605-9. doi: 10.1145/160688.160718.

[54] Fred J. Damerau. "A technique for computer detection and correction of spelling errors". In: *Communications of the ACM* 7.3 (Mar. 1964), pp. 171–176. issn: 00010782. doi: 10.1145/363958.363994.

[55] Elastic. *Multi-match query - Elasticsearch Reference*. url: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-multi-match-query.html (visited on 05/28/2020).

[56] Zachary Tong and Clinton Gormley. *Elasticsearch: The Definitive Guide*. O'Reilly Media, Jan. 2015. isbn: 978-1-4493-5854-9.

[57] Ashok Chilakapati. *Semantics at Scale: BERT + Elasticsearch*. Oct. 2019. url: https://xplordat.com/2019/10/28/semantics-at-scale-bert-elasticsearch/ (visited on 05/31/2020).

[58] Pandu Nayak. *Understanding searches better than ever before*. Oct. 2019. url: https://blog.google/products/search/search-language-understanding-bert/ (visited on 05/31/2020).

[59] Apache Software Foundation. *Apache Spark Interpreter for Apache Zeppelin*. url: https://zeppelin.apache.org/docs/latest/interpreter/spark.html (visited on 06/11/2020).

[60] Apache Software Foundation. *Zeppelin Interpreter Binding Mode*. url: https://zeppelin.apache.org/docs/latest/usage/interpreter/interpreter_binding_mode.html#which-mode-should-i-use (visited on 06/11/2020).

[61] Yanzhe Cheng et al. "Building Big Data Processing and Visualization Pipeline through Apache Zeppelin". In: *Proceedings of the Practice and Experience on Advanced Research Computing*. Pittsburgh PA USA: ACM, July 2018, pp. 1–7. isbn: 978-1-4503-6446-1. doi: 10.1145/3219104.3229288.

[62] Sanjeev Thakur et al. "Mitigating and Patching System Vulnerabilities Using Ansible: A Comparative Study of Various Configuration Management Tools for IAAS Cloud". In: *Information Systems Design and Intelligent Applications*. Ed. by Suresh Chandra Satapathy et al. New Delhi: Springer India, 2016, pp. 21–29. isbn: 978-81-322-2755-7.

[63] Red Hat. *How Ansible Works*. Library Catalog: www.ansible.com. url: https://www.ansible.com/overview/how-ansible-works (visited on 06/24/2020).

[64] James Benson, John Prevost, and Paul Rad. "Survey of automated software deployment for computational and engineering research". In: 2016, pp. 1–6. doi: 10.1109/SYSCON.2016.7490666.

[65] Juri Opitz and Sebastian Burst. "Macro F1 and Macro F1". In: *arXiv:1911.03347 [cs, stat]* (Nov. 2019). url: http://arxiv.org/abs/1911.03347.

[66] International Organization For Standardization. *ISO 9241-11: Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs): Part 11: Guidance on Usability*. ISO, 1998.

[67] Fábio Petrillo et al. "Interactive analysis of Likert scale data using a multichart visualization tool". In: *IHC+CLIHC*. 2011.

# Appendix A

# Dashboard figures

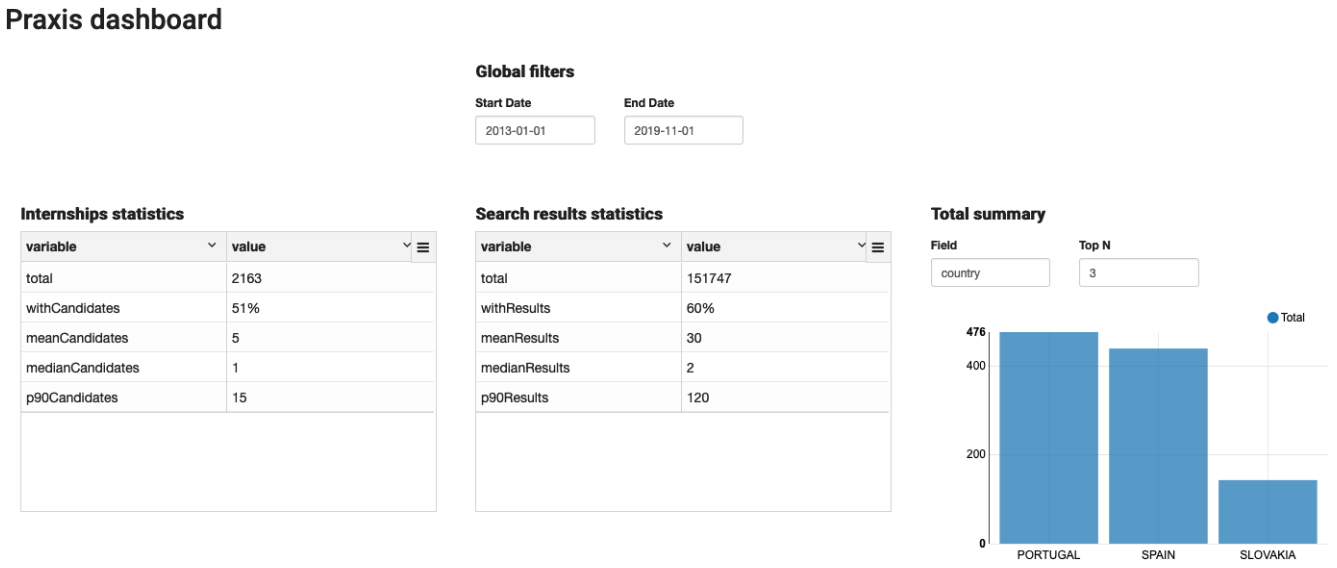


Figure A.1: Dashboard summary statistics

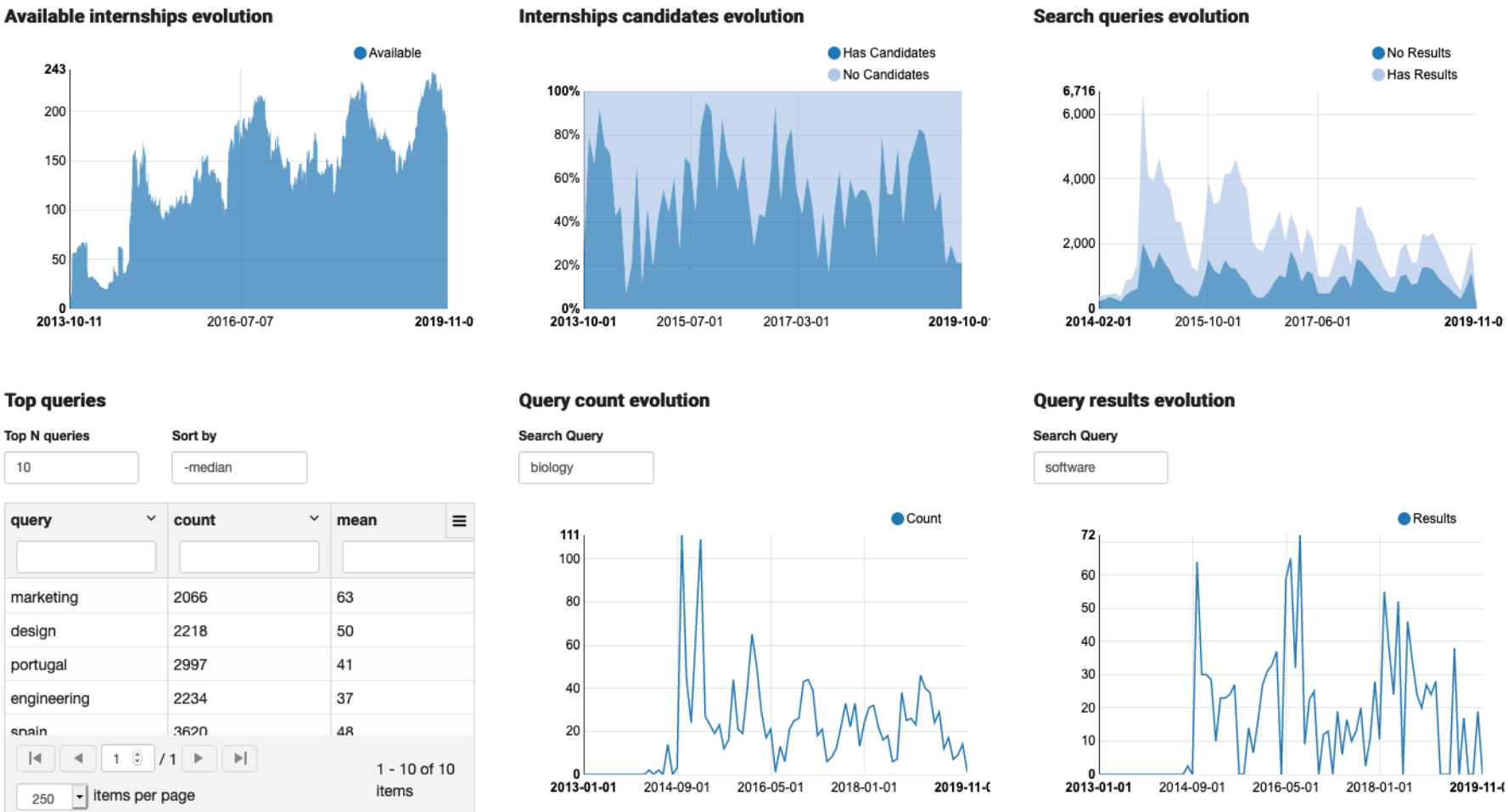


Figure A.2: Internships and searches evolution

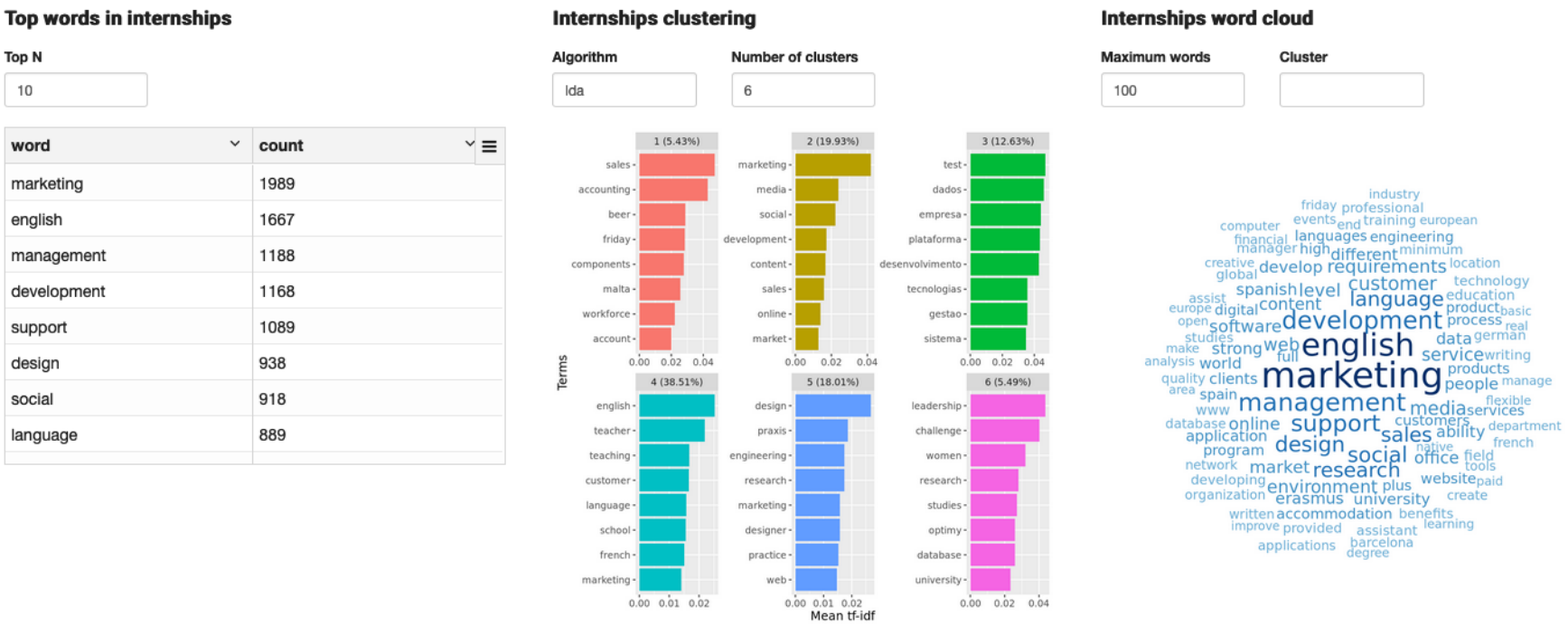


Figure A.3: Internships clustering and word cloud

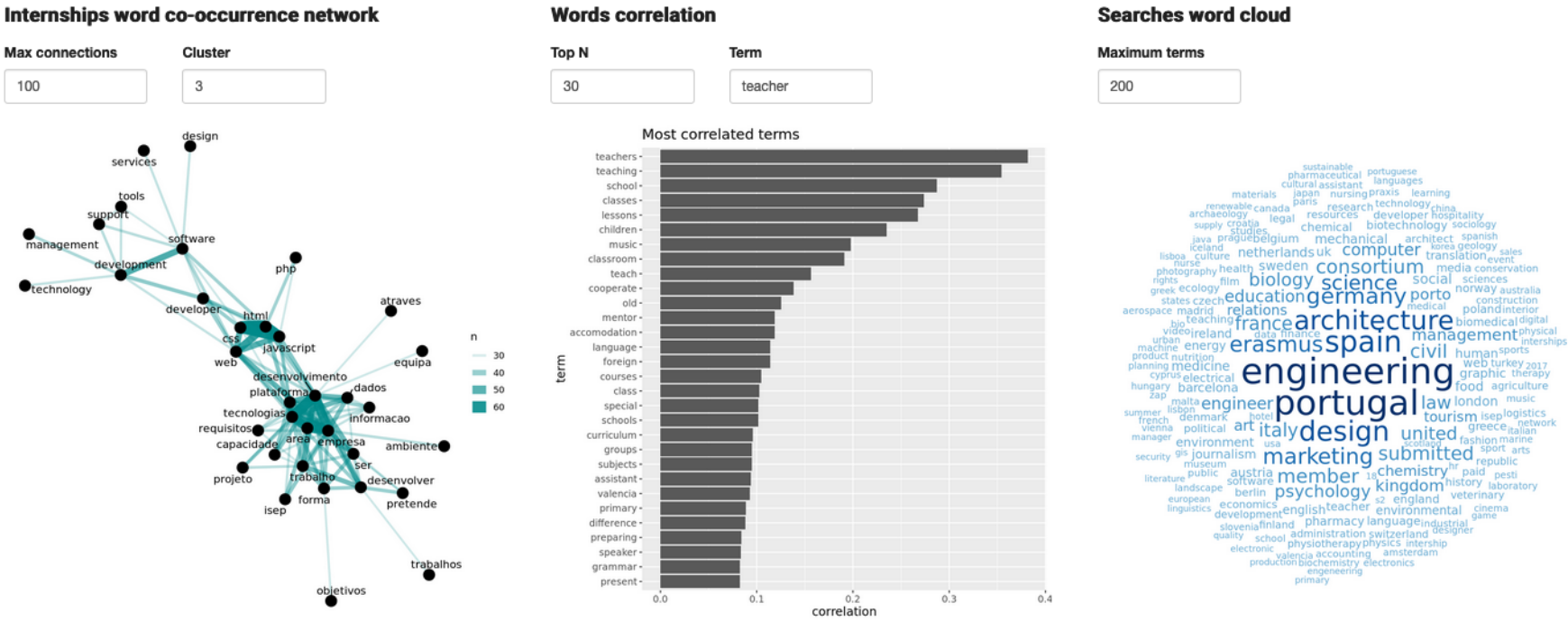


Figure A.4: Internships word relations and searches word cloud

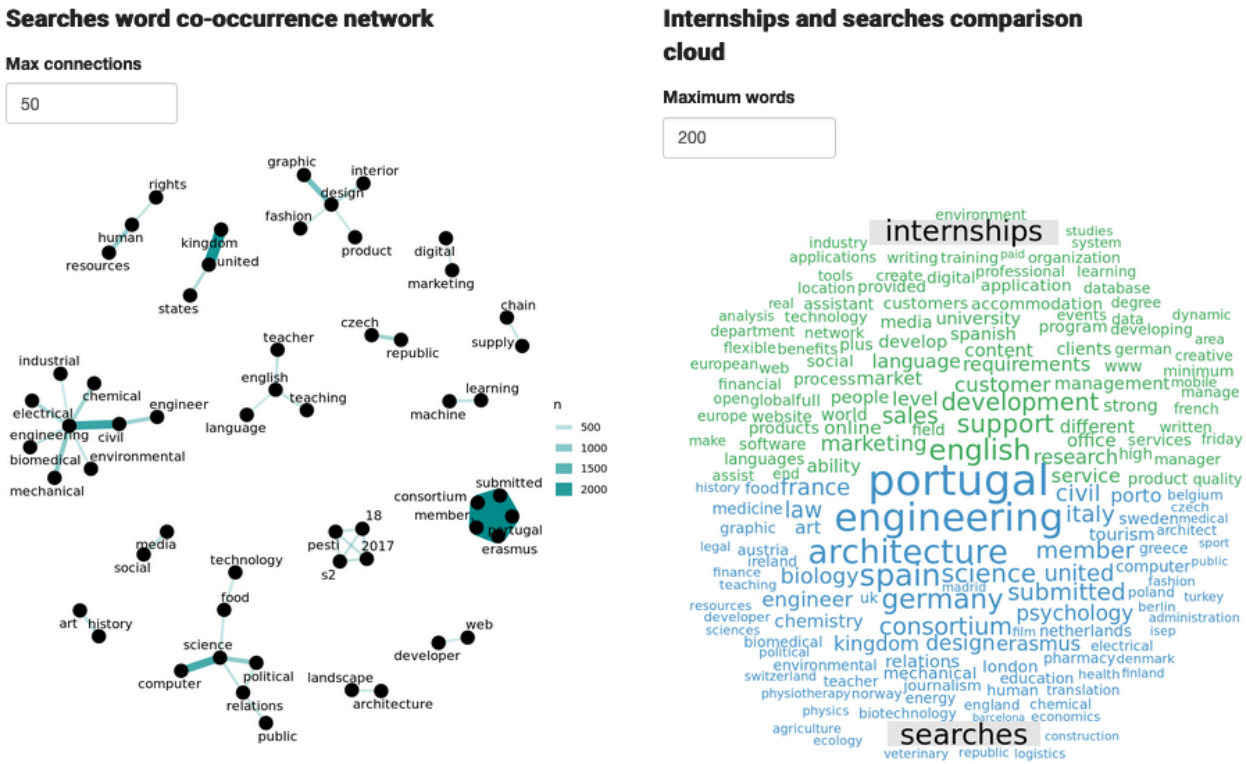


Figure A.5: Searches co-occurrence network and comparison cloud