Universidad Carlos III de Madrid

Research and development of Big Data technologies in the field of Human Resources

Bachelor's degree Thesis

Author: Ignacio Martín Martínez

Supervisor: Dr. José Alberto Hernández Gutiérrez

Degree in Telematics Engineering

September 2014

Acknowledgements

First of all, I would like to thank my family, for being unconditionally supportive and helping, especially to my father, David who has been listening and attending to all my crazy ideas and projects in the fields I study. Also, I would really like to thank my close friends and relatives, who have supported or contributed to my work and motivation in different ways and which, without them, I would not have been able to end this huge work the cheerful way I have.

I would also like to thank all my university classmates, especially the ones with whom I have shared days, classes, practices, projects and more. To all those who have helped me during the degree in one or other way and have been supportive and have taken me out from more than one problem. All of them could be considered a part of this work, since I have never had such a supportive and caring partners.

Finally, I would like to thank my supervisor, José Alberto for this year of ideas, new technologies, unexpected turns and constant support and for the result of this journey which started a year ago and have given us moments of trouble, anxiety, stress and complication. Thanks to our joint effort, we have been able to develop a Bachelor's degree thesis which I am very proud of.

Additionally, I would also like to thank Telefonica as due to its "Talentum Startups" scholarship I have received additional means, formation and support to carry on with this amazing adventure which is now ending and which, I hope, will bring more new adventures and projects for me to take part in.

Abstract

The aim of this work is to describe and report the achievements reached in the research and development of Big Data technologies and frameworks. The document describes the research process and the orientation towards the field of intelligent human resources management and recruiting, retrieving information from LinkedIn, Tecnoempleo and Infojobs and its posterior analysis inside a customized framework.

The whole report covers the entire process of data selection and retrieval by means of emerging technologies such as Node.js or MongoDB, data processing using R programming language supported by Hadoop and algorithms such as TFIDF or Okapi BM25 and finally reaches conclusions and practical applications, which are considered either for future study or revision during the project.

Most of the results are extracted from simple data analyses from the collections, which show very relevant data for Internet enterprises such as the activity from users and companies in LinkedIn or the most relevant aspects of job offers in Tecnoempleo. Additionally, a framework for this data analyses is built and documented to allow further research on the subject from a very clear starting point where all information can be gathered, processed and analysed within the same framework.

Finally, to all research work a business application is added with a developed prototype which shows commercial possibilities for the data sets gathered. Such prototype is also developed by emerging technologies and is highly connected to the framework and all the Big Data ecosystem.

Table of Contents

Table of Figures

1. Motivation and Objectives

1.1 Motivation

At present, information society is gaining relevance and data production and consumption is increasing. For years, the Internet has been focused on the information and contents generated by human users, where each user shares his data with a limited number of users. Nowadays, with the advent of social networks and the relevance acquired by smartphones and tablets, the Internet is experimenting a huge development and turning into “The Internet of Things” where every device is connected and generating more data to be processed as more devices are being connected at all times.

Consequently, as new information is generated, new studies and use cases appear for this new data which entail several ways of analysis, processing and computing. Moreover, the possibilities of such a big and diverse dataset provide a huge amount of possibilities and results that tend to be highly valued by top companies in the world.

This huge amount of information require new and advanced techniques for data processing, where traditional programming has to evolve as traditional techniques are based on computer power, which result useless when data size overcome single processor and memory capacity. Due to the limitations on computer hardware, this project will be addressing software solutions to optimize data processing over a common personal computer hardware with no improvements of any kind.

In order to achieve the main goal of this project, we will target the field of human resources and recruiting as application for the technologies and methods here described since the Internet offers a great amount of information and new ways of job searching and job recruiting. Therefore, the main sources of information will be professional networks such as LinkedIn or Tecnoempleo where users can either post or apply job offers as well as publish any other relevant professional information.

Actually, the field of talent hunting and human resources in businesses has not yet developed all the power that Big Data technologies have to offer. Although there exist several solutions and web services working on this field, no one offers solutions to analyse candidate profiles or job offers in a very extensive way. In fact, these services are rather limited and underdeveloped, not exploiting all the opportunities that datasets have to offer.

1.2. Objectives

Considering all the motivations explained in the previous section, this project addresses the new situation in the Net, so new solutions based on new programming paradigms and technologies have a strong starting point that takes advantage of Big Data technologies. Moreover, the project also targets the statistical analysis of job offer and candidate analysis, which is a very useful information both for enterprises and job seekers.

Thus, the aim of the project is to study and develop technologies for processing large amounts of data at the lowest cost possible using software solutions. For this purpose, several existing solutions and environments, such as MongoDB or Hadoop will be studied and tested in order to find the proper starting point to develop new contributions in the field. Concretely, the means defined to achieve our main objective are specified below:

O1	Data capture	Design and develop a framework for heterogeneous human-resource and job hunting data capture from several sources (LinkedIn, Tecnoempleo, Infojobs…)
O2	Data analysis	Implement framework in Objective 1 using Big Data technologies, offering a powerful set of tools that allow easy and fast data analysis and processing
O3	Data Mining	Study and implement within the framework information processing and retrieval algorithms to analyse the dataset and extract concrete results and conclusions from this analysis.
O4	Data visualization and application	Extract from the dataset analysis in the framework (Objectives 1 and 2) ideas for new services oriented to the Web. Design and implement a basic prototype of any of these ideas.

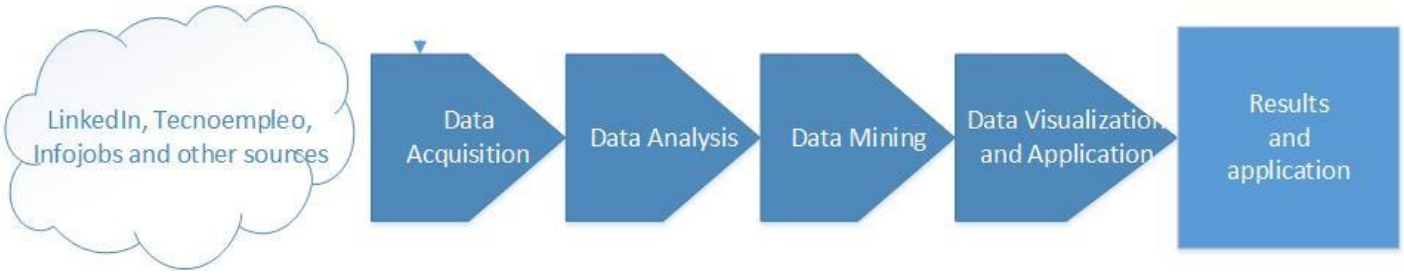


Figure 1.1 Objective schema during the project

Additionally, figure 1.1 illustrates the flow chart diagram to be followed in order to achieve the goals and expected results from the project. Taking as input the sources of information, all the steps will be performed until the final result is achieved.

1.3. Dissertation overview

This document is divided in 5 chapters where the methodology and processes of the project are presented and explained.

Chapter 2 presents the state of the art and all key technologies from where the project starts. This chapter is also divided in relevant sections which cover all the technologies and relevant solutions inside the proper framework.

Chapter 3 justifies the selection of technologies and algorithms to be used as well as developing the architecture of the full software solution developed. Additionally, it explains in detail how every phase in the project is developed and how the framework is installed and configured.

Chapter 4 presents the results obtained from the studies performed and answers the questions proposed. All the experiments undertaken are explained in detail in this section together with its results.

Chapter 5 explains and shows the prototype application derived from the whole project, including a brief description and a use case with detailed captures and descriptions of the application.

Finally, chapter 6 summarizes the conclusions extracted from the project and proposes new studies and works to continue the research towards new applications and results as well as describing the ideas which the team already though about.

Additionally, annex I includes the planning and cost of the full project from its beginning and annex II includes additional captures and tables which are referenced in previous chapters.

2. State of the art and key technologies

One of the most interesting features of Big Data processing is the huge amount of existing solutions, which offer the best starting point to develop new technologies in the field. All the technologies used in this project could be classified in four groups: Technologies for crawling information, technologies for storing information, technologies for processing information and technologies for web service design.

2.1. Data acquisition tools

At present, there exist several tools both from programmers and web services to extract information from the Internet.

2.1.1. Crawling

A crawler is a computer program that navigates through the web searching information about one or more topics by means of recursively accessing all the hyperlinks contained in the pages retrieved. Generally a crawler works by downloading the full HTML page for future parsing of tags.

There exists several libraries in various programming language, which provide functions to automatically download and even retrieve information from the selected XML tags based in XPath routes. Some of this solutions include languages such as Javascript, Java, R or Python.

Javascript offers many libraries for HTML page retrieval, such as http and https from node, JQuery or XMLHttpRequest, which perform various HTTP method requests to retrieve static web pages for future processing. In order to process HTML Javascript provides some libraries like HTML DOM API. However, Javascript HTML processing lacks some good libraries other programming languages possess.

Java instead, offers several integrated web crawling tools which perform both page retrieval and HTML parsing such as crawler4j, webSPHINX or JSpider. These crawlers allow programmers to retrieve complete websites, parsing the HTML to recursively obtain new links and information. Some of them even offer workbenchs and almost all of them are open source and supported by big enough communities.

The R language programming also provides very specific libraries for web crawling: XML and RCurl. RCurl allows to send an HTTP request to download HTML pages and XML is a tag parser for both XML and HTML which, based on XPath sentences retrieve a concrete tag from the XML tree or even convert a full HTML page into an R table.

Python, which is getting very popular in for web development and Internet related programs, offers also a lot of libraries devoted to web crawling. Some of the most popular are urllib, which opens

a connection with an URL, HTMLParser, which parses HTML allowing the user to customize the parsing functions. Other parsing modules are Beautiful Soup or lxml and for HTTP requesting, Requests module may be used.

2.1.2. REST API

Brute force crawling is usually efficiency expensive, as parsing a tree structure increases complexity and therefore, operation time. Due to this, several web services, especially social networks and user-based websites offer an especial tool for data acquisition called REST API.

An REST API (Representational State Transfer Application Programming Interface) is simply a web application which receives HTTP requests to several paths in its domain and answers with structured information. Such information may be returned in several formats, such as XML or url-encoded, although at present, JSON format is gaining most relevance. The returned information may be paginated to improve efficiency.

The REST model is more extended day after day, since it provides easily manageable information to any request and supports any platform. Thus, it is a very good alternative for web services which support, for instance, both common web page and mobile application.

In order to improve security in these services, many APIs are protected by different methods, such as OAuth. OAuth is a common three way authorization method developed to provide a user with mechanisms for controlling which applications access his/her data and grant or revoke such authorizations within an application.

At present, several Internet companies offer such tools to the open public. Generally, the information provided is constrained both by the own user authorization and by the company by means of daily query limits or strict registering of applications which use these tools. These companies are the ones such as Twitter, Facebook or LinkedIn.

2.2. Big Data Technologies

2.2.1 Hadoop ecosystem

Apache Hadoop is a software framework supporting distributed applications under an open source licence. It was inspired in Google technologies for Map Reduce and Hadoop Distributed File System. The key for Hadoop functionality is its distributed architecture: every simple piece of information stored is chunked and replicated by the Hadoop system, which keeps track of where each chunk is and coordinates efficient information management and retrieval.

Hadoop is composed by two main components: Hadoop Distributed File System and MapReduce. Both components are based on distributed programming and continuous communication. MapReduce operations usually take as input a directory of files stored in the Hadoop file system and computes the required operations in a distributed way.

In addition to this two main components, there exist several extensions based on Hadoop which enlarge its processing possibilities to specific subjects which would have to be built over plain Hadoop MapReduce otherwise.

2.2.1.1. Hadoop distributed file system

Hadoop Distributed File System (HDFS) is the storage component of the Hadoop platform. HDFS consists on a set of distributed nodes which store information and their replicas. The main components of HDFS are:

- NameNode: This is the primary node of the Hadoop File System and is in charge of keeping track of all the documents stored in the system. It stores the metadata related to all files and is in charge of starting and coordinating basic file operations such as opening, creation or modification for clients. The name node also provides the end user with a consistent file system management system and determines the mapping of information blocks to DataNodes.
- Secondary NameNode: This element is in charge of creating checkpoints of the name node contents by downloading name node contents, processing the information to merge files and other efficiency operations and uploads the information back to the name node. Secondary name node also stores a copy of the metadata for restoring in case of name node failure. However, the secondary NameNode is not a name node redundancy and will not assume its functions in case name node goes down.
- DataNode: The DataNodes are responsible for serving read and write requests to the clients. They store chunks of information under NameNode coordination. The DataNodes lack the logic and capacity for information retrieval: they only read or write their chunks at request

.

HDFS nodes are grouped in clusters. Each cluster must contain a NameNode and one or more DataNodes and may contain a Secondary NameNode. Each cluster component may be either installed in different machines communicated via Local Area Network or working in a single machine by means of a software based pseudo-distributed mode.

Additionally, it is important to remember that HDFS is not a data base; it is a file system where we can store folders and files. Therefore, in order to store information in HDFS some middle layer such as a built in database or a custom file hierarchy is required.

2.2.1.2. MapReduce

The MapReduce component of Hadoop is in charge of performing MapReduce operations from data stored into the Hadoop Distributed File System (HDFS). These MapReduce operations consist on the parallel processing of documents by means of a Map operation, which receives a set of documents, performs some operations and emits a set of key value pairs, and a Reduce operation, which receives the key value pairs obtained by the mappers and sums them according

to their key values, as well as performing possible further operations. The Hadoop MapReduce module is highly distributed and has two types of components:

- JobTracker: The JobTracker is the component responsible for distributed operation management. It is the component which separates an entire MapReduce operation into simple tasks which are sent to the TaskTrackers to be performed. The JobTracker decides whether each TaskTracker performs a map or a reduce operation and coordinates the input and output stream of TaskTrackers.
- TaskTracker: The TaskTracker component receives a chunk of the data set and performs either a Map or a Reduce operation on such data set. Once the TaskTracker ends, it emits its results and waits for more data.

Like HDFS, the Hadoop MapReduce component is separated in clusters of machines interconnected where each machine can have installed 1 or more TaskTrackers. Additionally, one machine should have installed the JobTracker and all configuration files should be edited accordingly. It is important to remind that each of these components is a separate entity that can be installed and configured independently from the rest.

2.2.1.3. Hive

Hive is a data warehouse which eases data management in an HDFS system. It provides a simple language very similar to SQL called HiveQL which allows to perform SQL like queries to a Hadoop Distributed File System transparently for the user.

Additionally, Hive allows the user to define Map Reduce operation code to be performed natively and provides full index support. It supports many file formats and is open Source.

However, since a MongoDB database was deployed and working when Hadoop was introduced and all data introduced in Hadoop was formatted using node.js programs, Hive was discarded as it did offer no new or interesting feature.

2.2.1.4. Mahout

Mahout is a software project of the Apache foundation that pursues building a scalable machine learning library. It implements the basic machine learning algorithms for clustering, classification or filtering on distributed and scalable systems.

Mahout can be thus implemented on top of the HDFS, being able to exploit all the power of Hadoop MapReduce operation into its own algorithms, making possible the application of machine learning algorithms to huge data sets in a transparent way.

Although it has been usually deployed over Hadoop, there exist many other distributed systems over which Mahaout can work, such as Spark.

2.2.1.5. Spark

Spark is an open source data analytics framework built on top of the Hadoop Distributed File System. In contrast to Hadoop MapReduce component, Spark does not compile the MapReduce classical implementation, providing efficiency improvements of a hundred times with respect to Hadoop MapReduce.

The key to Spark performance is that it provides primitives for in-memory cluster computing, so data can be loaded into a cluster memory and queried or processed repeatedly, which makes it a good candidate over which machine learning algorithms can be deployed. There exists Spark APIs for Java, Python or Scala, which combined with the power of these languages provide a very easy and powerful starting point.

Each Spark application run as an independent set of processes on a cluster, coordinated by the SparkContext object in the main program or “Driver program”. The SparkContext can connect to several type of cluster managers, such as Hadoop YARN clusters or the own Spark clusters. To allocate resources across applications.

Once connected, SparkContext acquires executors on nodes in each cluster. These executors are processes inside each node dedicated to the task performed and therefore, they store data and run computations. Once acquired, the SparkContext sends each executor the code for the application and afterwards, the tasks each executor should perform.

2.2.1.6. Pig

Pig is a platform which provides a High-Level programming language, similar to SQL, over the Hadoop MapReduce operation component in order to reduce MapReduce program development complexity as well as increasing efficiency.
The language provided by Pig is called Pig Latin. Pig interprets Pig Latin commands and translates them to Java MapReduce operation code for many basic MapReduce operations. Additionally, Pig Latin code may be augmented by means of the User defined functions written in Java, Python, Javascript or Groovy.

2.2.2. Google Ecosystem

2.2.2.1 Google File System

Google File System (GFS) is a distributed file system implemented by Google to run on commodity hardware and meet some of the specifics needed for Google search service. Many of the goals of GFS are shared with other file systems, such as scalability, performance or reliability.

GFS is meant to be deployed in a huge cluster of hundreds or thousands of machines built with inexpensive components and is expected to be accessed by a similar or even larger number of client machines. Therefore, there will be components failing at any time or even permanent faliures, so the file system must be highly fault tolerant.

Google File system provides a familiar file system interface, even though it does not implement any standard API such as POSIX. This API consists in a tree-like file hierarchy, where each directory is identified by its pathname. Common file operations are supported.

A cluster in GFS consists on a single master and several chunkservers. Each chunkserver is a Linux machine running a server process. Running a chunkserver and a client in any of this machines is possible, as long as its resources meet the requirements. Files are diveided into fixed-size chunks identified by a unique 64bit *chunk handle.* As a default, every chunk is stored three times in different chunkservers in order to assure reliability and fault tolerance.

The master maintains and handles all file system metadata, including mappings from files to chunks and viceversa. Clients communicate to the master for metadata information, but all communications involving data management is managed directly by the proper chunkservers. There is no cache in neither clients nor chunkservers in order to avoid coherence problems.

Chunk size in GFS is, by default, of 64MB. Each replica is stored as a plain Linux file on a chunkserver. Due to this size, reads and writes of a normal file do not require more than one interaction with the master, avoiding it to become a bottleneck.

2.2.2.2. MapReduce

MapReduce is, in fact, a Google creation. When the search engine started to get larger, a programming paradigm for scalable and efficient computing was needed, and thus, MapReduce appeared.

The MapReduce programming paradigm is based on taking a set of input key/value pairs and producing a set of output key/value pairs. The map function takes the input set and produces an intermediate set of key/value pairs according to the user code provided.The MapReduce system then groups together values according to their keys and sends them to the reduce function. The reduce function then accepts one or more keys and their set of values and merges them together to form a smaller set of values for each key.

The implementation of MapReduce is based on a distributed architecture. The map invocations are distributed across multiple machines which receive a chunk of the input data each. Reduce invocations are also distributed by partitioning the intermediate key values into several pieces by means of a hashing function. The number of partitions and partitioning functions are specified by the user.

After MapReduce completion, the output of the process is stored in each of the reducer output files, which can be combined or used as input for another MapReduce task or distributed application. The master keeps several data structures, such as each MapReduce state or the identity of the worker machine. Furthermore, the user coordinates the flow of information between map and reduce functions as well as storing the location of the output results.

2.2.2.3. Big Table

Big Table is Google proprietary storage system. Each Big Table is a distributed multi-dimensional persistent sorted map indexed by row and column keys and timestamp. Each value of this structure is an array of bytes.

It is important to remember that Google Big Table is a storage system prepared to index web pages and contents, and thus, its row indexes store the URL of each page in lexical order. Each column key is grouped into families, which are tiny groups of readable string identifiers which information is of the same type. Several column families may be defined in each Big Table and each column is identified by 'family:qualifier', being the qualifier any random string. Each pair row-column stores a cell with the information relevant to the indexes, that is, some feature (column index) of a web page (row index). Finally, each cell in Big Table can contain many versions of the information which are indexed by timestamp.

Google Big Table also offers an API with functions to perform basic operations to the data for client operations.

However, since Big Table is a proprietary system and it is based in another proprietary system (Google File System) it is not a good candidate for our storage system. Furthermore, Big Table is optimized for webpage metainformation storage and provided we are searching a system capable of storing as much information as possible, Google Big Table is limited for our intentions.

2.2.3. Microsoft Azure

Microsoft Azure (previously Windows Azure) is a cloud platform developed by Microsoft which provides different services for many applications, such as cloud computing or hosting sites. Microsoft Azure supports many programming languages, both from the Microsoft ecosystem and other acknowledged solutions.

It is self-described as a clod layer which works over several systems using Windows Server. Some of the supported services provided are non-relational storage, application hosting, or security backups. However, Azure constrains applications to work in Windows Server 2008, no matter the language or environment where they have been conceived.

2.2.4. Amazon Web Services

Amazon Web Services is a collection of cloud computing services available as a full cloud platform offered by Amazon. It is one of the pioneers in the field of cloud computing services and is broadly used by companies such as Dropbox or Foursquare.

AWS was launched in 2006. Its services online are available for any web service or other client-side applications. AWS provides a REST API and access through SOAP and is charged by usage. The AWS layer provides several services from scalable computing services to SQL databases, NoSQL, mailing server solutions or web servers

2.2.5. SAP Hana

SAP Hana is an in-memory database solution from SAP AG. It provides a database with several additional components, which add cloud computing and advanced analysis and modelling functionalities to the basic database solution.

Hana rests on low cost RAM memory, multicore processing capacity and solid state disk fast access to data in order to offer a high-performance solution of analytic and transactional applications. The first version in the market appeared in 2010 and has been improved until the present day.

2.2.6. Storage Systems

Data storage is another key point in Big Data since even with a really good processing system, if the database or storage system is not able to retrieve quickly and efficiently the information needed, the whole system will fail due to the bottleneck created.

2.2.6.1. MongoDB

MongoDB is the leading NoSQL database in the Big Data environment. It is a document oriented database where documents are formatted as JSON and indexes are fully supported. Mongo offers data replication, high availability and allows horizontal escalation transparently, so queries and data insertion is optimized.

MongoDB is supported by node and R among other programming languages and there exists several frameworks to model and schematize database contents. It also provides an easy tool for efficiently backing up and restoring information, so the database contents are highly portable and safe.

Thanks to the efficient horizontal replication and high availability support, queries to a mongo database are extremely fast when the collection is highly populated, which makes mongo a very good candidate for Big Data storage. Moreover, node offers with the connector a console program to access the data stored and built in functions, such as map-reduce, which will be presented later on this section.

Therefore, Mongo is a highly efficient, portable and complete solution for storage of large amounts of data with good interfaces for the technologies used within this project, which added to being an open source technology and having good documentation and community makes mongo the best alternative for data storage in many Big Data projects.

2.2.6.2. SQL

SQL stands for Structure Query Language and it is a standard language for relational databases access and management. A relational database is a database paradigm which defines tables of information in which each entry is a row and has columns, which are common to all elements. In

a relational database, rows among tables are related by one or more columns, allowing to store in a simple and ordered way thousands of entries.

Nevertheless, SQL is just a query language which allows managing a database manager. These database managers are in charge of interpreting the SQL sentences and performing the proper operations in the system for data storage. Many of these database managers (DBMS) are very popular and used in many applications. Some of the most well-known are: MySQL, PostgreSQL, Oracle, SQL lite and many others.

Relational databases follow the acronym ACID, which stands for:

- Atomic: All the steps that must be taken for the operation occur or the whole operation fails.
- Consistency: Everything, once stored, must be consistent in all queries and updates of the system. Whatever operation we perform, it must jump from a valid database state to another valid state.
- Isolation: Any number of transactions over the same information must not generate any error. Each single operation over the information must be independent from all the other ones.
- Durability: All changes performed to the information must be safely stored and definite, without the possibility of turning back.

These behaviour is very important, since it allows SQL to be a reliable, quick and robust system to manage and store information.

2.2.6.3. Cassandra

Apache project Cassandra is a NoSQL database which provides a Scalable and High available solution efficiently on top of either commodity hardware or distributed systems. Cassandra provides Linear scalability and fault-tolerance for the data stored in it.

The storage provided by Cassandra is structured in key/value pairs of data. The distributed architecture is based on a P2P communication protocol among nodes, creating a high level of redundancy.

2.3. Data mining tools

One of the key points for “Big Data” Analysis is processing information. Traditionally, data is loaded to computer memory and processed sequentially following the execution thread of the program. At present, this paradigm requires either hardware over scaling or a huge amount of time, and thus neither of them are acceptable for an efficient application.

In contrast, when referring to Big Data, parallel processing is a key concept. Any Big Data processing program takes advantage of all the parallel processing resources, such as multiple

core processors, by creating parallelizable and distributed algorithms which split data into independent chunks in which simple tasks may be easily performed.

One of the most important paradigms for data processing is Map-Reduce. The Map-reduce algorithm is a programming paradigm based on parallel processing which allows the user to define in two stages(map and reduce) the way of processing the information receiving key-value pairs and emitting new ones. This way of programming makes task performance really faster, although adding complexity to the traditional paradigm.

2.3.1. R

The R programming language is a strongly functional language broadly used for statistical analysis and information retrieval. R has a very efficient and strong base core which can be extended with libraries natively programmed, so all methods offered by these libraries are also highly optimized.

Once the information is stored, it can be loaded to R using a data structure called dataframe. A dataframe is matrix with named rows and columns which can contain any mixture of any kind of data. R provides several methods to efficiently traverse and perform operations over dataframes and many R packages accept them as input. In addition, R provides several built-in statistical and mathematical analysis libraries.

Thus, R is a more than appropriate candidate for data mining and processing. Especially in the scope of this project, there are three relevant R libraries: tm, RHadoop and RHipe. The first one is a data mining and processing library, whereas the other two are Hadoop connectors with R.

2.3.1.1. R: tm

The tm package is a text mining library which allows the user to prepare and process text in a simple manner. It is based on creation of Corpora, which are collections of documents that tm joins, process according to user commands and optimizes for term frequency matrices creation. Other operations supported are text cleaning, text trimming or stopword removal based on given lists for many languages.

2.3.1.2. R: RHadoop

RHadoop is a Hadoop connector for R developed by Revolution analytics. Hadoop is a very powerful tool, but it is restricted to writing Java programs in order to perform a map-reduce job. Although this is not a major inconvenience, it is an important drawback, since Java is not such a statistical specific programming language as R is. Thus, having a Hadoop connector to R allows to perform map reduces in an environment optimized for data mining and statistical analysis. The RHadoop program is composed by three R libraries: rmr, rhdfs and rhive.

Rmr package is in charge of connect to the Hadoop MapReduce engine and perform map reduce tasks defined by means of R code. Although working inside R, this package takes control of the

Hadoop cluster and performs all tasks using all the Hadoop daemons and resources, not just the ones reserved for R.

Rhdfs is an interface package between the Hadoop Distributed File System and the R environment. It basically allows Hadoop File System to be controlled from R console. This package provides a low level interface to HDFS with functions to perform basic and advanced operations over datasets stored in HDFS.

Rhbase is an interface for operating the Hadoop HBase data source stored at the distributed network. It provides several methods for initialization and read, write and table manipulation operations.

However, it is not necessary to install the three packages in order to run MapReduce operations in R. If we use HBase, we need rhbase, else we will need rmr for MapReduce operations and rhdfs for HDFS management interface.

2.3.1.3. R: RHipe

RHipe stands for R and Hadoop Integrated programming Environment and is a R and Hadoop connector. It allows to perform map-reduce tasks using R. The working of the RHipe library consists on an RClient program which maps and sends tasks to the Hadoop Job Tracker, passing all needed parameters such as map and reduce functions or input and output files.

One of the most important keys in the RHipe library is that it provides a low level interface over Hadoop. Any R user with RHipe can perform operations in R over a large dataset stored in HDFS, which allows the user to process Hadoop stored data with the advanced tools available on R.

2.3.1.4. R: Wordcloud

The Wordcloud package is a simple library that allows the R programmer to create a wordcloud by passing a set of words and their weight in order to create a weighted size cloud of words.

2.3.1.5. R: stats

The R stats library is a R library which offers many statistical, text mining and clustering functions. Clustering algorithms for words based on relevance numerical measures allow the user to separate words in groups (clusters) according to the measures given.

The stats function also provides statistical distribution generator functions and basic statistical measures computation.

2.3.2. Python

Python is an interpreted multiparadigm programming language originally created as a successor of the ABC programming language with a huge scientific orientation background in the Centrum Wiskunde and Informatica in the Netherlands.

Nevertheless, in the past years, Python has become a very popular programming language, which has provided an exceptional framework for the development of a lot of tools and libraries for many subjects. As a matter of fact, Big Data and data analysis have been some of these subjects, and thus, there are many tools in python, from MapReduce framework to natural language processing and data analysis tools developed and supported for their use.

2.3.2.1. Octo

Octo.py is a simple MapReduce framework which provides a good approach for processing fair big data sets by separation into parallel tasks. The system is based on a server-client approach where the server coordinates all clients connected which will execute the map and reduce functions defined by the user with the data provided by the server. Once each client is finished, it will return the results to the server which will integrate all received data and finalise the execution. Once they end, the clients wait for new tasks from the server.

2.3.2.2. Hadoop MapReduce

Hadoop ecosystem provides an API to run MapReduce operations which takes as input file texts stored in HDFS and outputs files in another directory of HDFS. The python code to develop will use Hadoop Streaming API to move information through MapReduce phases.

Each mapper or reducer function will be coded in a separated python file (mapper.py and reducer.py respectively). Once everything is set, the user can start the Hadoop MapReduce task by using the Hadoop Streaming API command passing by parameters the location of the map and reducer functions written in Python.

2.3.2.3. Natural Language Toolkit

Natural Language ToolKit (NLTK) is a platform to support the creation of programs that manage human language data. It provides simple and easy interfaces to over 50 corpora and lexical resources besides a good deal of text processing libraries which ease tasks such as classification, tokenization, stemmeing or parsing.

Thanks to its good documentation on programming fundamentals as well as on computational linguistics, it has become an easy and suitable tool for specialists in many subjects, such as education, engineering or even linguists.

2.3.2.4. TextBlob

TextBlob library is a tool for textual data processing by means of a simple API which complements tools such as NLTK or pattern. TextBlob provides functions for classification, parsing sentiment analysis tokenization or even word and phrase frequency computation.

2.3.2.5. Pandas

Pandas provides high-performance and simple data structures and tools for data processing in Python. Pandas helps to fill the lack of Python built-in tools for data analysis and modelling. It

provides the functions and resources to perform the whole data analysis process in python, avoiding the user the need of changing to a more specific programming language.

2.3.3. Java over Hadoop

Although there exists many other solutions to analyse data in Java, the more relevant tool written in Java for Big Data analysis is Hadoop. Due to this, the most basic Hadoop MapReduce operation is easily performed in Java by defining the map and reduce operations inside Java.

Setting up a MapReduce operation in Java is as simple as programming a Java class extending the Mapper class for mapper, another class extending the Reducer class and a Driver class which will coordinate the Hadoop Streaming API during the operation. Once everything is developed and tested, a Jar file should be created which, when executed, will trigger the MapReduce operation.

2.3.4. Matlab

Matlab is a software mathematical tool which provides an Integrated Development Environment (IDE) to develop mathematical computations with matrices, functions and other mathematical elements by means of a proprietary programming language called M.

In spite of being a proprietary software solution, Matlab offers several libraries loaded with solutions for many engineering problems, such as telecommunications, mathematical transforms, photography and sound mathematical processing or even basical statistical distribution creation and meassurements.

However, Matlab computing power decays with very large data sets, which turns to be together with the lack of advanced data processing and statistical tools a huge disadvantage for Big Data processing and management.

2.3.5. Scala

Scala is an acronym for “Scalable Language” and it is a multi-paradigm programming language built over Java with a clear and easy syntax. It is also apt for mission critical server systems, as precise typing forces the detection of many problems beforehand.

The main feature of Scala is its scalability as a result of integration of object-oriented and function paradigms. The object orientation allows the creation of advanced component architectures through classes and traits while its functional design provides support for evolving from a “java without semicolons” state to functional composition patterns, always supporting the preferences of the programmer.

Additionally, Scala can interoperate with Java sin a seamless manner. Classes may be mixed and cross-referred, as the Scala compiler includes a subset of the Java compiler to allow such mixtures. Furthermore, everything in Scala is an object, which provides the flexibility required to

make the language evolve and create scalable server software using concurrent and synchronous processing, parallel processing or distributed processing on a cloud environment.

2.4. Web technologies

When developing web services, there exists a great deal of alternatives of several programing languages and web frameworks which make web programming really easy. In the case of this project, only a light and simple prototype was to be made, so many powerful web frameworks were out of question due to complexity. Despite of this, we find several alternatives in order to design a web service prototype.

2.4.1. Node.js

As stated before, Node.js is local adaptation of Javascript which was conceived for light backend programming. In addition to npm, which allows package management easily, the node environment offers several packages which make web development simple.

The Express framework is one of these packages. Express is a really easy framework in node for backend programming. Using it jointly with npm, it provides the basic code for a server running in host which can be easily modified to develop a powerful web application very quickly. Furthermore, it provides Jade, which is an easy and powerful html template engine based on Javascript.

Other relevant framework provided by the node community is sails.js, which provides ORM, RESTful API auto generation, basic security functions implementation and static file auto linking. Another examples are RhapsodyJS, total.js or Locomotive.

2.4.2. PHP

PHP is a backend object oriented interpreted language very developed for web applications which also provides several frameworks to develop simple and light web services. It runs on an Apache Server and is open source.

PHP is introduced inside an html file and interpreted in the server before making the response to the user. Furthermore, PHP provides a very complete documentation and a great user community, as well as many third party sites devoted to the PHP programing.

2.4.2. Django and Python

Python is an interpreted multi-paradigm programming language which philosophy is writing code easily readable for humans. Python is commonly used in scientific environments due to its easy syntax and quick development and execution.

Django is a very easy web framework for Python which provides a very powerful set of functions that ease hugely the labour of the programmers. Among Django facilities are included ORM, REST API auto generation or user management.

Additionally, Django offers a simple template engine for html coding as well as an easy way of linking static files and external source files.

2.4.3. HTML5

HTML5 is the fifth version of the HyperText Markup Language, which is a frontend markup language which helps to define and state the layout and elements of a web page for a browser. In addition to being the web standard, HTML5 includes several improvements with respect to older versions that allow programmers to easily implement several modern functions, such as drag and drop, form validation, special form fields creation and many others.

HTML allows the programmer to include Javascript code and style commands both in the same html file or linking an external file. Additionally, many web frameworks provide template engines which allow creating several html pages based on a few files with some additional code.

2.4.4. CSS

CSS stands for Cascading Style Sheets and is a standard for frontend style definition. CSS allows the programmer to actually design and decorate web pages. CSS is really important when developing a professional web service, since it allows element collocation and colour and styles definition. In summary, CSS provides the mechanisms to turn an ugly HTML page into a more readable and good looking page.

2.4.5. R packages for web communication

In order to establish a connection with the project framework, it is necessary to offer an API which provides functions for data processing when required in any application. For this purpose, R provides several packages which help developing a full HTTP server in R, which allows performing computations and operations over our dataset.

2.4.5.1. Rook

Rook is an R package developed by Jeffrey Horner which creates an interface for R HTTP server functions. This package abstracts a web server to several objects and methods over these objects which are called in the same way as any other R object or method. Consequently, creating an R server with Rook only requires to write an R script to perform the required functionality with some calls to Rook functions.

Additionally, Rook allows writing HTTP responses to each request and customizing headers, which offers more alternatives when writing server side web services.

2.4.5.2. Shiny

Shiny is a light RStudio Package prepared to deploy an HTTP web server for R. It provides functions to develop a web page which allows the user to perform operations in R and deploy the results, with graphical elements and in a more user-friendly interface.

Nevertheless, Shiny is a package aimed to web page development, and therefore, is not prepared to deploy a REST API. There is no HTML code, only shiny functions, which handle the presentation of the web underneath.

2.4.5.3. OpenCPU

OpenCPU is an R library that provides tools for developing a cloud server with statistical computing capabilities and full compatibility to create HTTP interfaces. The HTTP API offers support for the main HTTP methods and a good path creation system for API functions. Additionally, it provides Javascript seamless integration and easy input and output methods and data types.

However, it does not allow writing responses directly, which leaves the programmer constrained to the existing methods, with little possibilities when none of them matches the project interests.

2.5. Text mining and information retrieval algorithms

Text mining consists in deriving high quality information from plain text. It is also known as text analytics, since it performs analyses on the text passed as input. Text mining includes several tasks such as text categorization, text clustering or stemming. Information retrieval refers to the activity of obtaining information which satisfies the user needs from a collection of resources. Such retrieval is based on either metadata or a user based query.

At present, there exist several algorithms both for data mining and information retrieval which generally offer good results for different situations.

2.5.1 TF-IDF

The Term Frequency - Inverse Document Frequency algorithm is a “bag of words” algorithm which scores all words appearing inside a document based on the number of times the word appears in the document and the number of documents where the words appear within the collection.

The algorithm starts from a collection of documents, such as a set of texts from a book or a collection of web pages in plain text. After text cleaning and some other pre-processing, the words inside each document are counted, which results in a set of “term frequencies” of each term for each document in the collection. To obtain the inverse document frequency, it is necessary to first obtain the document frequency, which is the count of documents where each term appears at

least once. Then, the inverse document frequency is just the quotient between the total number of documents in the collection and the number of documents containing a particular term.

The TF-IDF algorithm implements the following formula

$$TFIDF_{t,D} = TF_{t,D} \cdot log \frac{N}{N_t}$$

where $TFIDF_{t,D}$is the TFIDF value for term t and Document D, $TF_{t,D}$The term frequency of term t in document D, N the total number of document in the collection and $N_t$the number of documents containing term t

The result of the algorithm is a list of ranked terms per document. The numeric output is called relevance of a term and it is the importance it has within the document. Therefore, the term with the greatest relevance will be the one to consider, as it represents the keyword in the document, whereas the term with the least relevance is a word which has no value at all.

One of the most important aspects of this algorithm is to know how does it discriminate between relevant and not relevant terms. For that, we have to consider the two main parts of the formula:

- The term frequency makes a word which is mentioned several times in a document more relevant
- The inverse document frequency gives importance to words that appear in a very limited subset of the document collection with respect to words that appear in most documents.

2.5.2. Okapi BM25

BM25 algorithm is a word ranking algorithm which behaves in a very similar way to TFIDF, since it discriminates terms by their numeric score of relevance. However, in contrast to TFIDF which is a very simple algorithm, Okapi algorithm improves performance and results by considering more aspects of the text and text collection.

The essential mechanism of Okapi is very similar to TFIDF: it contrasts the number of times a word appears inside a document to the number of documents that contain that term, but instead of just multiplying both terms, Okapi shapes the result introducing document lengths, as the term frequency will tend to be greater in a document with many words. The Okapi BM25 algorithm is based, among others, on the following formula:

$$score(t,D) = \frac{f(t,D)\,(K+1)}{f(t\,,D)\;+\;k\cdot(1-b+b\cdot\frac{|D|}{avgdl})}\;\cdot\;log(\frac{N}{N_t})$$

Where t is a term in the document, D a document from the collection, f(t,D) the term frequency of query term t in document D, |D| the document word length, avgdl the average document length from the documents in the collection, N is the number of documents in the collection and N_t is the number of documents where t appears. K and b are free parameters that may be modified to refine the algorithm results.

From the formula, it can be appreciated the effect of the length on the result. An average length of the document will not affect very much to the TFIDF result, whereas a greater length will reduce the contribution of the TF and a smaller length will increase it.

2.5.3. Clustering algorithms

Clustering consists in separating documents in subsets that are coherent among them, but different from others. Clustering is a very good example of unsupervised learning: there is no human assigning documents to classes, but is the machine itself the one deciding which document belongs to which class.

It is very important to consider the classification of clustering algorithms in soft and hard clustering: hard clustering implies a strict assignment where each document belongs exactly to one single cluster whereas in soft clustering any document may belong to more than one cluster at the same time.

2.5.3.1. K-means clustering

K-means clustering tries to create k groups of documents which present similar characteristics in terms of Euclidean distance. When given a numeric value, such as relevance, K-means algorithm can compute clusters based on the distance among such values.

This algorithm is based on the computation of centroids for each of the K clusters based on the values given as input. Initially, the centroids are randomly assigned. Each value is assigned its closest centroid according to Euclidean distance, conforming each of the K clusters. Once the assignment is completed, the centroids are recalculated according to the points conforming the cluster. This process is repeated until the recomputation of centroids results in almost the same point.

K-means clustering is one of the most common clustering algorithms at present. Furthermore, it has been broadly developed and possible to implement using MapReduce, which makes it a very useful candidate for Big Data analysis.

2.5.3.2. Canopy clustering

Canopy clustering is commonly used as a pre-clustering algorithm to speed up the clustering, especially in large data sets. The primary target of this algorithm is to partition data in overlapping subsets or canopies and afterwards perform more refined clustering techniques.

The algorithm consists on setting two threshold values (T_1 and T_2) so the first is greater than the second, pick any data point to be the centroid for cluster X, compute the distance of all points to such centroid and compare them to thresholds. If the distance is smaller than T_1, then the point is added to the cluster X and if it is also smaller than T_2 then the point is removed. The computation and assignment of points to X is repeated until there are no points left and X is not empty.

2.5.3.3. Hierarchical aggregative clustering

Flat clustering is efficient and relatively simple, but outputs an unstructured set of clusters which requires a specific number of clusters as input. Hierarchical clustering instead outputs a structured result, a hierarchy, which gives a priori more information than the flat clustering approach.

One of the most common types of hierarchical clustering algorithms are the agglomerative algorithms. A bottom-up approach to agglomerative clustering considers each document inside a collection as a single cluster and successively agglomerates pairs of clusters until all clusters are together below the same hierarchical structure. A top-down approach splits clusters recursively in a hierarchical structure until single documents are reached.

Hierarchical agglomerative clustering results are usually represented as dendograms, which are tree-like structures which leaves are documents from the collection which are merged one another as branches go up to the root. Once a dendogram is created, the number of clusters created depends on the point of cut desired, which allows the user to define the sharpness and proximity among clusters.

However, Hierarchical clustering algorithms offer better results at the cost of increasing complexity and therefore reducing efficiency, so the selection of a clustering algorithm must be conducted by the selection between very good results or simplicity and speed.

2.5.4. Vector space scoring

The vector space scoring algorithm derives document vectors with a term for each word that appears in the document. By default, each term weight is computed using TFIDF algorithm. Therefore, the collection of documents can be considered a set of vectors in a vector space where each term is a numerical value.

Consequently, the way to find the score of a document given a query, also in vector representation, would be to compute the dot product of the document vector and the query vector. Nevertheless, as document lengths may vary significantly, a query could result being closer to a document just because that document is larger than others. To solve this problem, this dot product

should be normalized, so instead of computing the dot product, we will compute the cosine similarity, which given two document vectors d_1 and d_2 is defined as:

$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{\left|\vec{V}(d_1)\right| \cdot \left|\vec{V}(d_2)\right|}$$

Where $\vec{V}(x)$ denotes the document vector of x and $\left|\vec{V}(x)\right|$ is the modulus of such document vector. The formula shows the computation of cosine similarity between two documents, but any of them could be a query document represented in our vector form. Additionally, it could be appreciated that apart from comparing a document to a query, vector space scoring may be used to contrast documents.

It is very important to understand the results from this algorithm. Since the equation computes similarity, a greater resulting value is a better match, so when applying this algorithm with a query to find and order results, decreasing similarity ordering would output the best results.

3. System architecture and design

Within this project, several existing technologies have been selected, installed and developed to achieve objectives. One of the main criteria taken into account when selecting technologies has been to choose open source solutions as far as possible. This is due to the fact that open source licenses are easier to obtain and use, documentation and user communities tend to be bigger and more developed and open source programs usually are more flexible and allow changes in the main program. Another key point has been documentation: if no documentation or not a very good documentation was found, the technology was discarded, since without documentation, working with a technology usually turns difficult and tedious.

Additionally, emerging technologies have been seriously considered, since we understand that it is very important to create solutions adapted to the future tendencies and standards. It is also very important to bear in mind that this project covers several aspects of data processing and data mining and thus there are several requirements for technologies and programs which are currently on research and development.

All the decisions and steps taken during the project have been undertaken in order to achieve the primary goal of the project, and therefore, each of the steps in the development of the project follows the consecution of one of the objectives established in chapter 2. Such steps, shown in figure 3.1, will guide the development and structure of this chapter in order to present the technologies selected, the methodology applied and the tasks undertaken during the project.

Figure 3.1 Objective schema during the project

3.1. System architecture overview

Such a big and complex framework requires a very defined and structured design in order to be manageable, simple and extensible. The whole system is divided in smaller modules which can be treated separately despite interacting among them with no problems. Each module is then separated in smaller components which conform their functionalities.

The modules created are: the crawling module (O1), which is in charge of retrieving and storing structured information, the data processing module (O2), which manages retrieving data from the

database and processing it as needed and the service module (O4), and it is in charge of deploying and serving all web components, including the necessary database queries.

In addition, the database has to be a central element which serves to all modules with persistent and updated information. For this purpose, there will only exist a central database system based on MongoDB, although each module can implement and develop its own storage information systems for other purposes.

All modules in the application will follow a Model View Controller (MVC) design pattern which can be supported by different frameworks and all of them will work as independent entities of the main framework. The architecture schema of the framework is illustrated on figure 3.2.

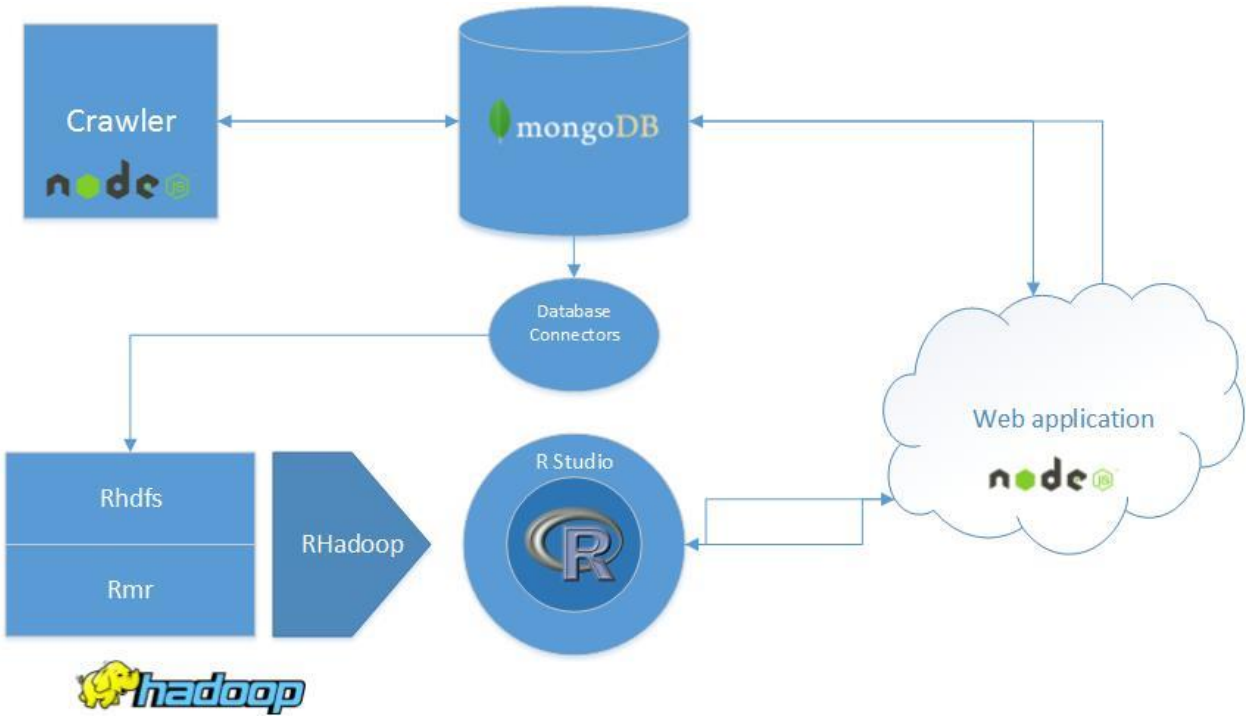


Figure 3.2 Project architecture schema

3.2. Data acquisition

3.2.1. Technologies involved

Node.js (node from now) was a technology emerging at the beginning of the project and showed very good perspectives for crawling and as basic programming language in web development. Node is a Javascript based programming language which is very close to the web environment, very easy to understand and learn and highly compatible with many of the technologies and data structures used in the project.

Furthermore, node provides several libraries to connect with MongoDB, which is the main candidate for data storage, in contrast to many other programming languages which offer less

options. Moreover, node is a light environment, highly portable, easy to install on any operative system and the language is Javascript, which is a well-known programming language with the same code syntax on any platform.

In addition, node provides the “node package manager” (npm) which allows the user to easily install all needed packages with a line or two and keep everything up to date automatically. Furthermore, node was created to ease the development of web services and, therefore, there exist a good deal of web frameworks and template engines. Although this may not seem useful for web crawling, it could be a very good candidate for prototype development, so acquiring the knowledge from the beginning will save time in the future.

LinkedIn offers a full REST API to retrieve diverse information, from public user profiles to job offers and company information. This API is fully supported by the company and only requires registering to develop applications interacting with it.

Once the application is registered, Oauth credentials and programmer user tokens are provided. The Oauth credentials are needed for any request and allow the programmer to implement user authorization of his application to the public. User tokens allow the programmer to access to all the information of a user as well as other API methods which must be called on behalf of an authorized user.

There also exists a node package to send OAuth authenticated requests to the LinkedIn API. Additionally, some node code is used to adapt the other crawler outputs to the database model and the backend of the prototype obtained is also programmed in node using the Express framework.

For this project, the most relevant methods on the LinkedIn API are the search methods (people-search, company-search and job-search) which allow any user to retrieve user profiles, job offers or company profiles according to some key parameters (name and surname in profiles or keywords otherwise).

However, the API has a limit of 500 queries per API method and day, which results in a severe limitation for data retrieval and the need of executing the crawler once a day. The requirement of some key parameters in the search API methods also requires additional efforts finding the proper parameters to query the API.

For the other source pages (Tecnoempleo and Infojobs) we used the R XML and RCurl libraries since, in this case, no API was provided and XML library provides functions for easy and quick html parsing and RCurl allows to easily download all the html pages needed. In any case, it must be taken into account that this crawling method is much less efficient, since HTML parsing is usually slow and many web pages understand several repeated HTTP request to them as harmful and perform techniques to avoid such connections.

As it was the most well-known and handy solution for data storage, the selection for long term data storage was MongoDB. MongoDB provides a NoSQL vary fast solution for JSON-like document storage with a highly efficient index management system and several data processing tools embedded, such as MapReduce. Additionally, there exist many node libraries for the connection with MongoDB, such as Mongoose.

Mongoose, which is a node package for MongoDB and node communication, provides a really easy way of connecting to MongoDB from node, modelling the raw data to be inserted into user predefined schemas and also gives the possibility of performing built in operations over a whole MongoDB collection.

3.2.2. Development process

The very first step in this project was to start obtaining any kind of Human Resources and Job Hunting data from the web as soon as possible. The initial source of information was LinkedIn. After some research on the subject and all the considerations stated in previous section, node was the programming language preferred for LinkedIn application management.

Once the design decision was taken and after trying how all these APIs together work, some design considerations had to be taken. The main design decision was that an MVC (Model-View-Controller) architecture was to be used.

Initially, the design of the crawler proposed creating three different and separated crawlers, each one containing its own controller, model and querying engine (according to the MVC programming model) and managing just one kind of data. Thus, there was a crawler for person profiles, another crawler for job offers and a last one for companies. Each crawler stores information following the data structures specified in annex II (AII.1).

Initially, all controllers execute an init function which reads configuration files and opens a connection to the MongoDB database. These configuration files mainly contain a set of search parameters to query the LinkedIn API (Names and surnames for person profiles and keywords for the rest of collections). Additionally, the init function calls the init functions of the rest of components (the querying engine and the database model). Figure 3.3 offers a snippet of the init code from one of the controllers.

After initialization, the program calls the querying engine to start making queries to the API. The querying engine is a layered architecture with two levels:

1. The first level offers a common querying engine which performs any type of query agnostic to the specific contents of the query.
2. The second one adds the logic to specify the type of data and parameters of each query over a common lower-layer interface.

Once the results of a query are received, the program checks if there exists any error, computes whether the following query changes its offset or the query parameters and sends the results to the modelling module to be processed.

```
function init()
{

 readPersonFiles();

 eventEmitter.on('personInfoRead', function(){

   persondb= new Db(DBname, new Server('localhost', 27017), {w:'majority'});
   persondb.open(function(err, db)
   {
      if(err!=null)
      {
          console.log("ERROR");
          console.log(err);
       }
       else
       {
           personCol= require('../model/personDb');
           personCol.init(DBname, DBcollection);
       }
   });

 });
}
```

Figure 3.3 Crawler init function

The model module performs modelling and saving operations to the raw data obtained from the API. Using Mongoose, the data is modelled according to the database schema and saved. To this point, no data validation or duplicate removal is performed: everything downloaded is modelled and stored in the database as it is.

After saving results, the program then decides whether the next step is a new page from the current query or the beginning of a new query. When the proper action is undertaken, the querying process is repeated until there are no more queries pending or the daily limit of downloads is reached. If the later occurs, the program terminates its execution.

Nevertheless, once the prototypes of the three programs were developed, a “super controller” managing the execution of each of the programs sequentially was proposed. The fundamental advantage provided with this solution was the reduction of execution complexity, since the super controller manages the order of the queries and the change of program when any of them reaches its daily limit, leaving the user the lone task of starting the program.

The basic working of the “super controller” is to coordinate the work flow of controllers so they are executed sequentially. The programming required for this element was really simple, as it only needs to add events and their listeners at the beginning of each controller as well as triggers at the right moment.

In fact, one of the most complicated elements to manage in a node program is the workflow, as Javascript is a programming language hardly based on callback functions and does not provide wait methods neither any blocking system to partially stop execution. Consequently, the crawler program is strongly related to events and event listeners, as they are the most adequate way to control workflow when callback functions appear.

Figure 3.4 shows a schema of the full crawler program working with all its modules and dependencies. Additional code captures of the query engine as well as the rest of components may be found in annex II (AII.2)

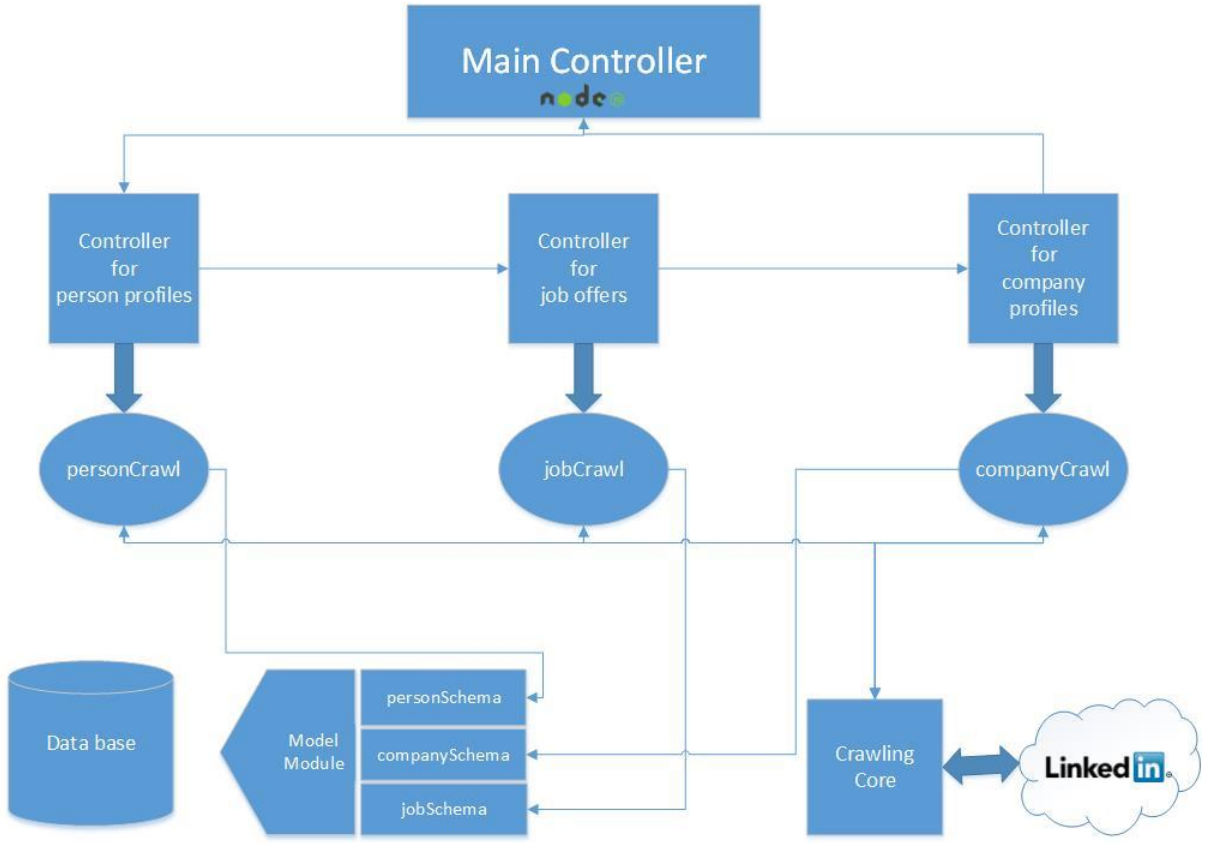


Figure 3.4 Crawler structure

Once a sufficiently large dataset had been downloaded, data processing started and it was appreciated that there existed other sources that could enlarge and enrich the dataset. Consequently, Tecnoempleo and Infojobs were added to the source list and crawlers for both were developed.

The design and development of these crawlers was easier and faster. Using XML and RCurl libraries a simple R script could be written that requested each web all the job offers and then

processed the results with the Html parsing functions of XML library. Once the information is processed and prepared is stored in a structured dataframe which is saved in the computer hard drive as an Rda file.[1]

The basic working of these crawlers consist on three steps starting from the search results page of each web:

1. First, using XML library we retrieve the content from the HTML tag corresponding to each job offer Universal Resource Locator (URL) and store them in a list using XML method "xpathSApply". The sentence in R used is shown in figure 3.5. This step can be repeated until every search result page is traversed.
2. Once all the job offer URLs are stored in a list, RCurl sends GET request to each URL and the HTML page is stored to be completely parsed and exported to an R list using XML.
3. Finally, the proper element from the mentioned list is extracted to obtain all job offer fields, create a dataframe row for that offer and add the row to the resulting dataframe.

```
41    html <- htmlParse(script,encoding = "UTF-8")
42    urls <-xpathSApply(html, "//div[@class='search-results-list ']/ul/li/div[@class='description']/h2/a",xmlGetAttr,"href")
```

Figure 3.5 Code for HTML parsing

However, this new sources are not as populated as LinkedIn and, therefore, should not be considered as primary sources but as a complement to the main one. Despite this fact, both are a good enough source to take into consideration when performing different data processing analysis separately. Additional captures on these crawlers code may be found in annex II (AII.2).

3.3. Data analysis

3.3.1. Technologies involved

Data analysis requires the setting up and configuration of a framework of great capabilities. Due to the size of the data stored, traditional data analysis is out of question, as it is impossible to load the full dataset to computer memory. Furthermore, single-threaded analysis fails to provide a fast enough solution for this project to offer a feasible way of analysing data efficiently. Consequently, data structuring and advanced efficient parallel operations are needed in the construction of a Big Data framework.

In a first approach, the Hadoop ecosystem would be a very good alternative, since it offers really nice applications to manage Big Data, is open-source and provides a really good supporting community.

However, for data processing, the best alternative would be the R programming language, as it offers the largest collection of data analysis and statistical tools, a good deal of mechanisms to manage large amounts of data and several libraries for graphical representation of results.

1 Rda file: R data format to store dataframes in the hard disk

Moreover, it is an open-source alternative which is gaining a lot of relevance, increasing its community and the enterprises interested in it.
Consequently, the most desirable combination for the framework would be to join both Hadoop and R to get the higher efficiency in Big Data management from Hadoop and the best analysis tools from R.

For this purpose, we found in RHadoop a real handy tool, since it allows to perform MapReduce operations over R, which provides to mapper and reducer function development all the advanced tools mentioned above and allows exporting results to a dataframe for further analysis. In contrast to RHipe, we considered that RHadoop was more complete and had a more active community, as well as a more defined architecture which helped to understand easily the package.

Therefore, rmr on top of Hadoop MapReduce engine and rhdfs on top of HDFS are the basic components of this framework. Additionally, since the whole project is stored on a MongoDB database, a MongoDB and R connector needed to be used to extract data necessary for data analysis. For this purpose, two different alternatives were considered and included in the project:

1. Package rmongodb from R provides all kind of mechanisms to query information from MongoDB with a relatively easy syntax. It deals with cursors and MongoDB BSON objects, so efficiency is guaranteed.
2. Node.js Mongoose provides all needed tools for data extraction from a MongoDB database and, with a little further processing, allows any data to be written to files which can be easily stored in HDFS.

More precisely, the Hadoop implementation selected to be the core of the framework, was Cloudera Hadoop (CDH4) [1] as it offered the best configuration documentation and an auto configurable pseudo distributed cluster installation based on Ubuntu repositories.

The server machine available is a Dell OptiPlex 360 containing an Intel Core 2 Duo processor at 2.6 GHz, 160 GB hard drive disk and 4GB RAM memory running Ubuntu 12.04 LTS (Precise Pangolin) with no graphical interface. This machine will host the entire framework as well as all the add-ons the project shall develop. The machine is in the Carlos III University of Madrid in the Telematics department and is connected to the Internet behind a firewall which only allows connections to ports needed for framework interaction.

During the project, a smaller framework based on built-in MongoDB MapReduce engine and node Mongoose was considered, as it would avoid setting up the Hadoop ecosystem and thus reduce the configuration workload. Nevertheless, the alternative was very quickly discarded as the processing needs outran easily the solution capacity.

3.3.2. Development process

The first step in framework configuration was to physically install the computer in the provided facilities and turn it on. Then, the Ubuntu 12.04 OS was installed and the machine was connected

to the Internet. In order to act as stable server, a static IP address from the university was assigned to the machine and configured on start.

When the machine contained the operative system and the basic control and monitoring programs, such as an SSH server and file sharing mechanisms, the Node.js and MongoDB programs were installed from the Ubuntu official repositories. Both programs are included in official repositories, so a simple apt-get install command was required for each.

Then, both R and Hadoop were installed. There was no necessary order of installation. The R installation can be performed adding a repository to the Unbuntu repository list which can be found in the Cran project official web page (reference 5). Once the repository entry is added and authenticated, R may be installed with an apt-get install command.

However, the plain R installation is a console program as shown in figure 3.6, which is not very desirable when the processing algorithms and variables become bigger. Consequently, Rstudio server was installed. RStudio server is a tool which provides a web based IDE for R, which keeps track of variables, functions, allows to develop code files among many other functionalities. This IDE is accessible in port 8787 of the machine hosting it, so the framework is accessible from

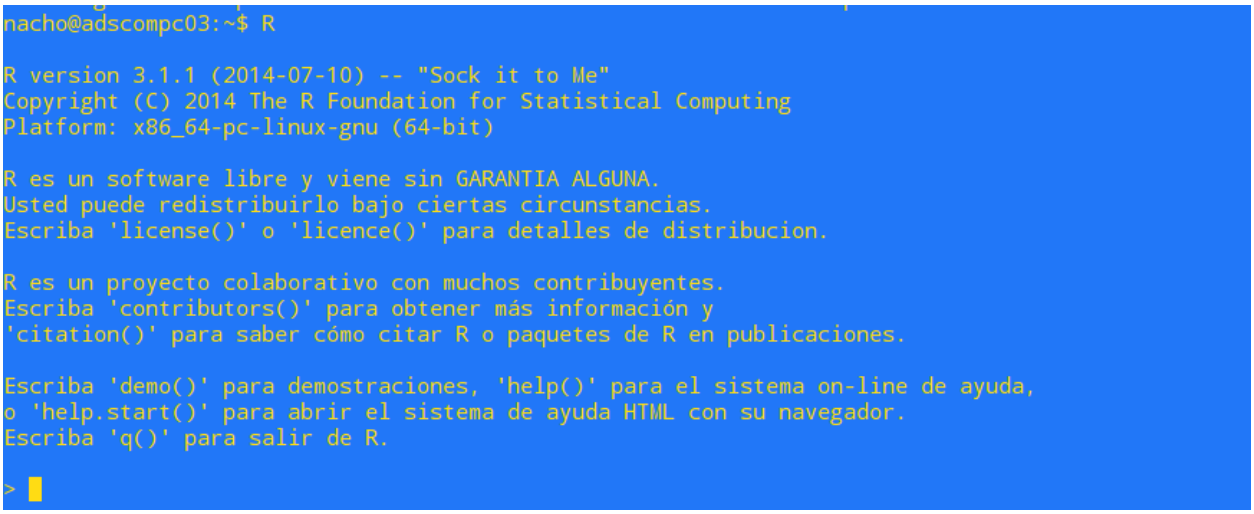


Figure 3.6 R base console program

anywhere and therefore working on the project becomes really flexible.

Cloudera Hadoop installation and configuration was a bit more difficult, although Cloudera provides a vert good userguide through the process which can be found in its webpage (reference 9). The installation process is based on a repository solution installed by a Debian page downloadable from the Cloudera webpage and once installed allows the user to install everything from repository.

Due to the resources available for the project, the pseudo cluster was developed. The Hadoop pseudo cluster is an emulation of a distributed Hadoop cluster installed in a single machine which

provides all components from the Hadoop ecosystem and works exactly as an actually distributed cluster. Although this solution is less efficient, it has less costs and requires less maintenance.

Since all Hadoop components are written in Java, it is necessary to have installed in the computer the Java Runtime Environment (JRE).

Once CDH4 is installed, Hadoop start running its processes and is completely functional, so files may be stored into the HDFS and MapReduce jobs may be started by any authorized user. The Hadoop distributed File System can be accessed at any time as if it was a typical file system and data storage may be performed in a similar manner. The file structure is exactly the same as a traditional file system structure, as shown in figure 3.7. In fact, the most notable change with respect to any other file system is the sentence to access the HDFS.

```
nacho@adscompc03:~$ hadoop fs -ls
Found 17 items
drwxr-xr-x   - nacho supergroup          0 2014-08-14 13:26 Lifespan
drwxr-xr-x   - nacho supergroup          0 2014-03-28 10:53 RHadoop
drwxr-xr-x   - nacho supergroup          0 2014-07-17 13:09 companyIds
drwxr-xr-x   - nacho supergroup          0 2014-04-24 12:52 identifyKeywords
drwxr-xr-x   - nacho supergroup          0 2014-04-29 10:41 identifyKeywordsJobs
drwxr-xr-x   - nacho supergroup          0 2014-03-27 19:42 input
drwxr-xr-x   - nacho supergroup          0 2014-04-09 10:41 java
drwxr-xr-x   - nacho supergroup          0 2014-06-29 17:25 jobDescription
drwxr-xr-x   - nacho supergroup          0 2014-05-29 13:33 jobIds
drwxr-xr-x   - nacho supergroup          0 2014-08-07 20:22 jobcount
drwxr-xr-x   - nacho supergroup          0 2014-06-24 13:04 newSpecialties
drwxr-xr-x   - nacho supergroup          0 2014-03-28 10:51 output
drwxr-xr-x   - nacho supergroup          0 2014-05-21 12:55 personSpecialties
drwxr-xr-x   - nacho supergroup          0 2014-05-21 12:54 personSpecialtiesName
drwxr-xr-x   - nacho supergroup          0 2014-05-21 12:55 personSpecialtiesName2
drwxr-xr-x   - nacho supergroup          0 2014-06-27 16:21 skillsAndExperience
drwxr-xr-x   - nacho supergroup          0 2014-06-17 11:28 tfidf
nacho@adscompc03:~$
```

Figure 3.7 Hadoop file system root directory

Finally, once both Hadoop and R are installed, the final step is to install RHadoop so they integrate each other. For that purpose, the R packages needed for the installation may be downloaded from Revolution Analytics GitHub account. In addition to the RHadoop R packages, some additional packages have to be installed to solve compatibility issues, such as RJava, RJSONIO or others which provides RHadoop package with advanced tools not installed in the basic R distribution. These packages must be installed before installing RHadoop and should be installed in the system space.

By default, R packages are installed in the personal folder of the user who logged in. In the case of RHadoop and its dependencies, this is not possible, as the user starting mapreduce operations and HDFS operations would vary with respect to the R user. Consequently, all RHadoop

packages and dependencies must be installed by the root user account. This way, all packages get installed inside the system files and permission issues are avoided.

After all these configurations, the basic framework is completed with the exception of MongoDB integration for data analysis. Although direct connection to MongoDB from R is also possible, most of the data analysis inputs rely on a variable node script which connects to MongoDB via Mongoose and extracts the fields desired for each data analysis to store them in a set of plain text files which will be introduced in HDFS for processing. An example script of this connectors may be found in annex II (AII.3).

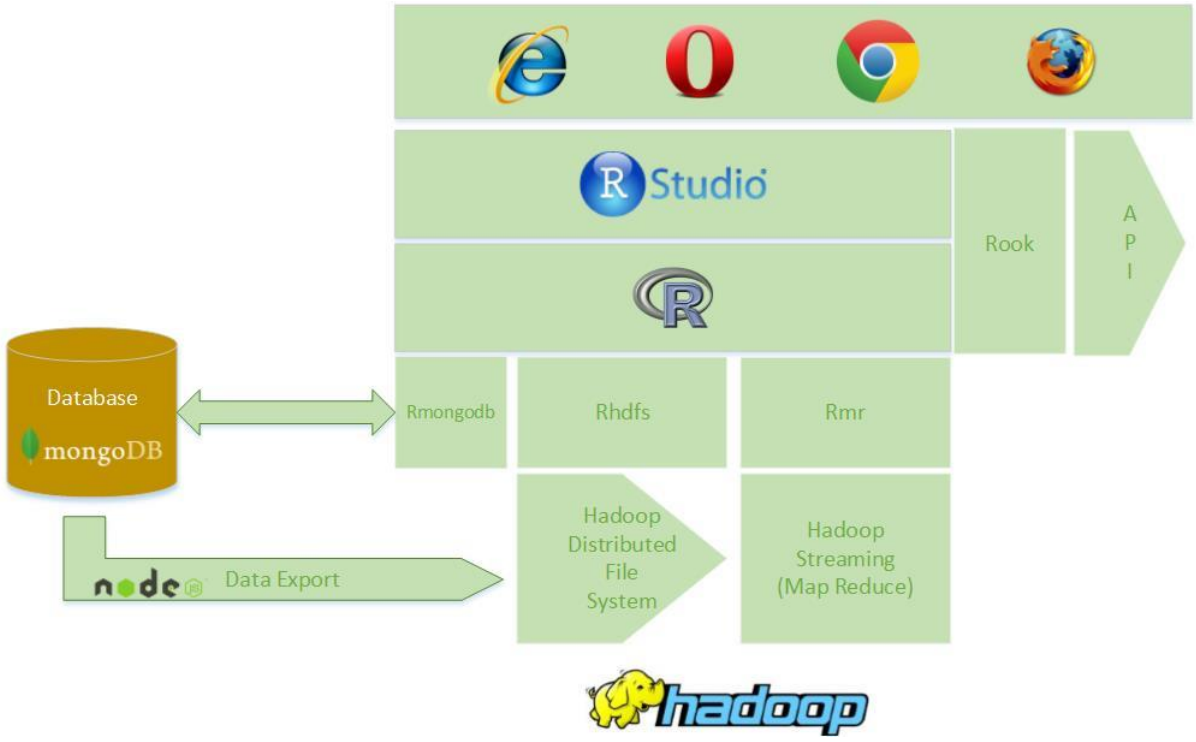


Figure 3.8 Framework schema

The last part of the complete framework is the REST API to provide data processing functionalities to external applications which do not manage R structures nor code. This REST API opens a set of HTTP GET requests on server port 8080 (which is currently protected by firewall from external connections) which allow external users to manage the framework for their information processing needs. The implementation of this component is specified in section 3.4 with the rest of the web application developments and is based on the Rook package. Figure 3.8 illustrates a diagram of the full framework and their components.

3.4. Data mining

3.4.1. Technologies involved

The basic technologies involved in the consecution of this objective are mostly algorithms used to classify, score or separate words of the data set. These algorithms are both well-known algorithms with a clear and developed projection and new or improvements of other algorithms which try to address the new problems here encountered.

Obviously, the main technological support for this objective is the framework developed for the project, which will provide the starting point for data analytics. Additionally, many other R libraries will be added to the framework ecosystem in order to improve and augment the capabilities and functionalities of the framework. For instance, stats, plyr or tm libraries are added to the framework.

Stats library provides a bunch of advanced statistical analysis tools as well as other data analytic advanced tools, as the k-mean algorithm implementation. Plyr exploits many possibilities of parallel computing and provides highly efficient functions to be applied to larger data structures. Tm provides basic structures and algorithms optimized for data mining, as well as the functions to manage and create them. For presenting the results, the libraries wordcloud and gplots provide several advanced functions for graphical representation.

One set of algorithms highly involved in the project nature is the family of the word ranking algorithms. Such algorithms include various mathematical models which try to discriminate and punctuate words to extract the most relevant ones from a great collection of texts depending on various factors. Thus, they allow the quantification of the relevance a term has inside one or more documents.

Word ranking algorithms are strongly related to term frequency and document frequency values, which we have seen can be obtained by means of a MapReduce program. Once these frequencies are obtained, the computation of word scores is fairly simpler and requires less computer resources.

The most relevant algorithms studied for this project are Term Frequency Inverse Document Frequency (TF-IDF) and Okapi BM25. Both algorithms are strongly related to term frequencies and inverse document frequency. Inside the framework, both can be implemented fairly easily from a collection of terms and their frequencies obtained using the MapReduce word count.

The TF-IDF algorithm implementation is used mainly to extract keywords from huge text collections. The actual TF-IDF algorithm computes the TF-IDF score for each term per document, but in this project, it was adapted to compute a unique score for the whole document collection. For that purpose, once we get the term and document frequencies for each word from the MapReduce word count, we can add them up according to the following equation in order to compute this custom TF-IDF measurement.

$$TF - IDF_{full} = TF \cdot log \frac{N}{N_w}$$

Where TF is the full term frequency, that is, the number of times a word appears in the collection, N the number of documents inside the collection and $N_w$is the number of documents containing the current word.

Okapi algorithm is a more complex algorithm that takes into account other aspects of the collected texts, such as text lengths and other free parameters. Here, starting again from a MapReduce word count containing the document term frequency of each term, the term itself and the identifier of the text where it appears, we can compute text lengths and document frequency of each term by means of the count R library 'plyr' function and then compute the okapi score, which follows the following equation:

$$score(Q, D) = \sum_{i=1}^{n} \frac{f(q_i, D)\ (K + 1)}{f(q_i\ , D)\ +\ k \cdot (1 - b + b \cdot \frac{|D|}{avgdl})} \cdot log(\frac{N}{N_{q_i}})$$

Where Q is a query of n words, D a document from the collection, $f(q_i, D)$the term frequency of query term q_i in document D, |D| the document word length, avgdl the average document length from the documents in the collection, N is the number of documents in the collection and $N_{q_i}$is the number of documents where $q_i$appears**.** K and b are free parameters that may be modified to refine the algorithm results and in this project are fixed to 1.2 and 0.75 respectively.

3.4.2. Development process

As stated before, there exist several programs in Node.js which open a collection, and write to a set of files all the fields required for the data analysis to be performed. In order to assure compatibility, each file of the set is structured separating fields of a same entity using the character chain: "///" and entities one another by means of the character chain: "asdfghjkl".

The selection of such trivial chains is hardly conditioned by the nature of the documents processed. Several texts retrieved contain typical separation characters such as the new line character or the tab or space characters. In Addition, the export program is also responsible for text cleaning, removing all unwanted characters, such as punctuation characters, characters containing tildes or even numerical digits.

After the execution of any of these connectors, the data is stored inside a set of plain text files compiling to the above format which are ready for further development. These files are then sent to a HDFS directory which will be specified afterwards as the input path for a MapReduce task.

Then, the rmr package can be invoked. When Map and reduce tasks are programmed and a MapReduce job starts, Hadoop opens in parallel all the files in the collections and sends their content to various mappers that perform the programmed task by the user and emit key-value pairs for the reducers, which perform on their pairs the task written by the user. A code skeleton for MapReduce operations is shown in figure 3.9

```
####################Required Libs#######################
#additional libraries go here
##########################################################

#####hdfs initialization#####
hdfs.init()
############################



wordcountTFIDF = function(input, output = NULL, pattern = " ")
{
  wc.map = function(., lines)
  {
    ##Mapper code goes here
  }
  wc.reduce = function(word, counts )
  {
    ##Reducer code goes here
  }
  #MapReduce defined here
  mapreduce(input = input ,output = output,input.format = "text",map = wc.map,reduce = wc.reduce)
}
##Operation starts, job started
wordcountTFIDF('/user/nacho/personSpecialties')
```

Figure 3.9 MapReduce skeleton code

It is important to remember that, even though Hadoop MapReduce is able to process huge amounts of data more efficiently and faster, it is limited, and thus helps to reduce the information to process for R to continue processing. When a MapReduce operation concludes, it can be loaded into an R data frame containing two columns: one for keys and one for values. Then, this dataframe can be processed very efficiently by R.

In this line, the first word ranking algorithm developed was the custom implementation of TF-IDF described above. Using MapReduce, it is possible to count the number of appearances of any word within a collection. For that purpose, we just need a mapper which cleans and separate words in a text by whitespaces and a reducer that sums all the occurrences of each word.

Alternatively, it is possible obtain document frequency of each word, which is the number of entities from the collection where a word appears, by extending the mapper to separate texts using the separating chains and emitting for each word the term frequency of the word and an additional instance of the word (with added characters, for example “%%”) which has a value of one, so when the reducer sums values it will emit for each word the term frequency, identified by the word itself, and the document frequency, which will be identified by the word itself followed by a restricted chain of characters.

When this MapReduce task is finished, the output dataframe contains both global TF and IDF of all words in the collection of documents examined, so with a little more processing consisting in

joining all terms together and applying the formula, it is possible to obtain a dataframe with each single term appearing on the collection and the TF-IDF relevance value. Then, after ordering the results, the dataframe contains a decreasing ordered list with the terms appearing in the text collection, ready to be displayed with a histogram, a pie or a wordcloud.

Then, based on this TFIDF implementation, a point is reached when the implementation of a job offer search engine seems feasible. Consequently, the first step is deciding which algorithm will be used for this purpose. Okapi BM25 seemed a better candidate, as it considers text lengths and the collection available contains a very length-variable collection.

Therefore, the implementation of Okapi BM25 started. It was discovered that, if given the proper data input to the algorithm, the execution of a MapReduce task could be avoided. So initially, a MapReduce task was developed to create such structure. The basic idea of the structure consists on a dataframe which relates each word in the collection to its document identifier. For this, the input data should contain the text identifier and the text contents adequately separated which the map function would separate and process to emit as key the document identifier and as value each term in each document.

This mapper function, as shown in figure 3.10, separates all lines received in ids and term chains to continue cleaning and splitting the later into single words identified by their document id which will be sent to the reducer function that will do nothing in this case, just emit the values as they enter each reducer.

```
###MAP################################
wc.map = function(., lines)
{
  docs<-strsplit(x=lines, split='asdfghjkl')
  whole<-unlist(docs)
  a<-as.data.frame(x=whole)
  for(i in 1:nrow(a))
  {
      a[i,"ids"]<-as.vector(unlist(strsplit(as.character(a$whole[i]), split="///")))[1]
      a[i,"words"]<-as.vector(unlist(strsplit(as.character(a$whole[i]), split="///")))[2]
  }
  lines<-a$words
  lines<- gsub('[[:punct:]]',' ',lines)
  lines<- gsub('[[:digit:]]',' ', lines)
  lines<- gsub('\t', ' ', lines)
  lines<-tolower(lines)
  lines<-gsub('Ã¡', 'a', lines)
  lines<-gsub('Ã©', 'e', lines)
  lines<-gsub('Ã', 'i', lines)
  lines<-gsub('Ã³', 'o', lines)
  lines<-gsub('Ãº','u', lines)
  lines<-gsub('\n', '', lines)
  values<-data.frame(lines, row.names=a$ids)
  listKeys<-unlist(x=as.character(row.names(values)))
  listValues<-unlist(x=as.character(values$lines))
  keyval(listKeys,listValues)
}
```

Figure 3.10 identifier-term mapper function code

Once this structure is created, R can handle it easily to perform further operations using functions from libraries like plyr and count term and document frequencies and even word lengths. Then, the Okapi algorithm implementation turns to be as simple as finding in this dataframe the values

included in the user query, separating them from the rest, computing the Okapi value for each document and presenting the ordered results as an output.

The score computation is made in parallel using the functions and libraries built in R, all needed values are computed from the input dataframe and Okapi algorithm is performed in parallel to all entries in the table. Once Okapi values are computed, they are re-joined. The Okapi implementation returns a dataframe containing the identifier and the okapi score of each document ordered by the later in decreasing order.

These results may be used for further analysis or even to consult which job offers are more relevant with respect a given query. When this algorithm worked, it was clear that it had many useful applications, and so the idea of creating a job offer search engine became clear.

In addition to this two main processing development lines, several experiments on data were performed in order to extract new conclusions or confirm some theories. To implement such studies, specific data was exported to the framework for analysis and further research. Many algorithms and methods have been tested and used. The specific details of each experiment as well as the result data and interpretation may be found in chapter 4. Most of the resulting illustrations and tables have been obtained using the framework tools.

3.5. Data visualization and application

3.5.1. Technologies involved

Once the data is obtained and processed, any resulting application should be tested and experiment, and for this, a prototype would be one of the best alternatives. For this purpose, it is desirable to define a common collection of web development technologies to be able to quickly create prototype applications of the results.

Consequently, back-end and front-end solutions must be chosen to set a good environment. Due to the development of the project, the best candidate for back-end programming is Node.js, since it has been very used and the team has practise in programming it. Besides, node offers the Express framework, which allows the developer to easily set up a web service based on HTTP requests.

In fact, Express manages transparently most of the issues of web server development, is highly connected to the node package manager, assuring a very fast and agile deployment, and provides a very simple Html template engine: Jade. Therefore, the front-end scheme is based on the Html code generated by Jade and Express, with static CSS and Javascript linked files which add style and LinkedIn connection features.

Since we are dealing with LinkedIn data, the LinkedIn API may be useful, both for data consulting or even user logging to the application developed. For the later, LinkedIn provides a Javascript library which helps to connect and retrieve all needed user information from the client.

Additionally, in order to access to the Big Data sets and perform operations over them using our framework, it needs to have a connector which allows compatibility of node server-side processing with the R framework. For this, an REST API would be one of the best alternatives, as it would ease data exchange over a standard protocol unaware of the technologies below and also will grant any web application in the world to easily connect to the framework. Rook library provides functionalities to easily deploy any kind of web server based on the R language, allowing the connection of the data and the data processor with any web service over a common interface.

3.5.2. Development process

For data visualization and application validation, a prototype methodology was proposed as a way of testing the results of such application at a moderate effort. Such web service had to be easy to implement and deploy. In order to create such a prototype based on the application idea, node was used.

The web page developed has to be fully functional and intuitive, granting the basic functionalities of the application as well as a friendly design. Node and express provided the starting point to create and improve a light web server with the three following views:

- Index view: The page displays a welcome message and invites the users to login the application using LinkedIn
- Search view: Once the search task is complete, the service displays the results in a new page, ordered decreasingly by relevance and paginated.
- Error view: Whenever there is an error in the web or in the API, the server displays an error page with its corresponding error.

Additionally, the web server is in charge making requests to the API with the proper queries based on the data extracted from the user LinkedIn account. For this purpose, the API provides a method which given a query in text format returns the ten most relevant job offers in JSON format. Once this JSON is received, the server displays its contents on the search view in different boxes containing a brief description, the source and the link of each job offer.

The development of a web server based on npm and express is fairly easy. The node package manager provides tools to create a functional skeleton which sets connections and bindings to port 3000 as well as the full file system, which contains the auto generated code for a “hello, world” program. These file structure is shown in figure 3.11.

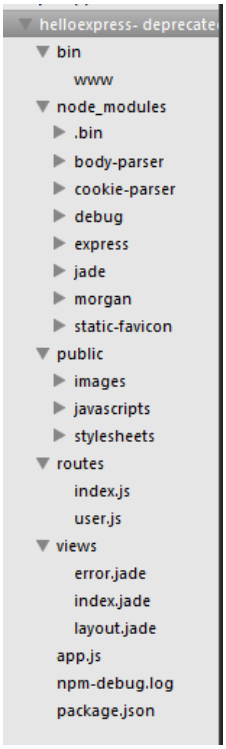


Figure 3.11

The structure shows various folders:

- bin folder contains an executable file (www) for starting up the web server
- node_modules folder contains all the additional node packages included in the project
- public folder contains static files, Javascript program files and CSS style sheets
- routes folder contains node files which define the actions to be taken when a certain path of the server file system is reached by a client. The bindings of each path to a specific node file and function is undertaken in the file app.js
- views folder contains all jade templates developed which will be rendered when the programmer calls the function render over the template name
- app.js is the main controller file of the application. Bindings of paths and methods to functions is performed there as well as many other server settings and values.
- package.json is a file containing a JSON structure which defines all dependencies of packages of the project, so when it is built, npm manages the download and installation of such modules.

Once this structure is created and all dependencies fixed, the only things to do are developing the functionalities of the application and creating the templates used.

The main functionality of the application is the search engine. When any user logs with its LinkedIn account, a Javascript routine retrieves the data relevant for analysis and sends it as a form to the server. The server then receives the information from the user and sends it to the framework API, which will return a collection of job offers in JSON format prepared for visualization. When every job offer is received, the server calls the result page rendering and sends it the results and thus, the result page is shown in the application.

Whenever there is an error, the application will show a generic error page in order to inform the user that the processing should wait. If the user wants to log out from LinkedIn from the result page, there is a button that calls the LinkedIn Javascript API and unlogs the user. During the waiting period for the results, the application displays a processing gif

.

The REST API is an important part of the framework, as it provides external communication for the processing capabilities of the framework. This API is built using the web library Rook in R, which provides methods and variables to manage external connections via HTTP. The API developed is just in charge of receiving queries from the web server, calling the okapi function implemented passing as a parameter the queries and the collection dataframe and returning the results via an HTTP response to the web server in JSON format. Figure 3.12 shows a snippet of the code of the API.

```
queryAPI<-function(env)
{
  print("Connected")
  req <- Request$new(env)
  res <- Response$new()
  res$header('"Content-Type": "application/json"')
  if (!is.null(req$POST()))
  {

    data <- req$POST()[["query"]]
    jobInfo<-req$POST()[["jobs"]]
    offset<-req$POST()[["start"]]
    print(req$POST())
    print(offset)
    print(paste("Received: ", data, sep=""))
    #received<-fromJSON(data)
    st<-toString(data)
    st2<-toString(jobInfo)
    offSet<-toString(offset)
    st<-gsub('"query', "", st)
    st<-gsub("[[:punct:]]", "", st)
    st<-gsub("[[:digit:]]", "", st)
    print(st)
    st<-gsub('\\s*$', '', st)
    st2<-gsub('"jobQuery', "", st2)
    st2<-gsub("[[:punct:]]", "", st2)
    st2<-gsub("[[:digit:]]", "", st2)
    st2<-gsub('\\s*$', '', st2)
    print(st2)
    offSet<-gsub('"start', "", offSet)
    offSet<-gsub("[[:punct:]]", "", offSet)
    print(offSet)
    offSet<-as.numeric(offSet)
    print("let's go querying")
    q<-paste(st, st2)
    q<-gsub("\\s+", " ", q)
    print(q)
    results<-query(q, new.val)
    index1<-1+offSet
    index2<-10+offSet
    results<-row.names(results[index1:index2,])
    results<-toJSON(results)
    print(paste("Results:", results, sep=" "))
    #Mongo
    id<-urls<-description<-requirements<-source<-NULL
    crawl <- mongo.create()
    buf <- mongo.bson.buffer.create()
    query<-paste('{"jobID":{ "$in":', results,"}}", sep="")
    cur <- mongo.find(crawl, 'CrawlerDB.job_offer_dbs', query = query)
```

.

Figure 3.12 REST API code snippet

The API is also connected to the MongoDB database of the framework, to a collection which merges job offers from all the sources of the project in order to retrieve all the relevant information. As a security measure, this API runs on a local port on the framework machine and cannot be accessed from the outside, so only a local web server may access it. The MongoDB collection used for this web service contains job offers from all sources in a common format, so information about any job offer can be retrieved at any time. Further details on the collection parameters may be found in annex II (AII.4)

4. Experiments and results

4.1. Data set description

As a result of all the crawling efforts during the project, several Gigabytes of information have been retrieved and different types of structured data have been prepared and processed for data analysis, as shown in the following table.

Identifier	Source	Number of Structured entities	Size in memory(KB)
LIP	LinkedIn: Personal Profiles	67071	13858
LIC	LinkedIn: Companies	515000	866284
LIJ	LinkedIn: Job Offers	170536	633478
TE	Tecnoempleo: Job Offers	2691	4562
IJ	Infojobs: Job Offers	1223	957.4

All these collections contains structured information as described in section 3. 2. In order to be processed, any of these collections must be exported to the framework as described in section 3.3. Additional information on the data stored may be found in annex II (AII.1) and captures of the collections may be also found in annex II (AII.4).

Consequently, once the crawlers retrieved information enough, the information could be exported and processed in the framework. Most of the experiments undertaken try to address the following issues:

1. Are job offers in LinkedIn a good collection for analysis? (4.2.1)
2. How do companies take advantage of social networks as LinkedIn (4.2.2)
3. What is the average active duration of a job offer in LinkedIn (4.2.3)
4. Which are the most demanded skills in LinkedIn? (4.2.4)
5. Do candidates in LinkedIn know how to meet employers' requirements? (4.2.4)
6. What are the basic parameter from offers in Tecnoempleo? (4.3.1)
7. Which skills are demanded in Tecnoempleo?(4.3.2)
8. What do employers seek: experience or formation?(4.3.3)
9. Which profiles can be found in networks as Tecnoempleo? (4.3.4)

4.2. LinkedIn job collections analysis

LinkedIn analyses in this project focus on person profiles and job offers collections in order to obtain information related to the job recruiting process as well as the existing offer in LinkedIn. The collections studied offer several fields such as job offer description or candidate skills lists which are very interesting for word scoring and clustering algorithms.

4.2.1. Job offer analysis: offer descriptive field selection.

In order to analyse the collection of job offers extracted from LinkedIn, we need to consider two relevant fields: “skillsAndExperience” and “description. Both these fields contain descriptions of the job offer; the first more oriented to job requisites and the other more oriented to job description.

Although both may seem perfect candidates to use in offer analysis, we will study both and extract a conclusion from both. In order to perform this analysis, we will find the lengths in number of words of each fields and the percentage of “stopwords”, which are common use words that result meaningless for our analysis.

SkillsAndExperience field

Initially, we consider the field skillsAndExperience, which is supposed to contain requirements for the offer candidate. By means of R functions we count the field lengths in each job offer after and before removing “stopwords”. Once this is achieved, we can plot the histograms containing the frequencies of each length.

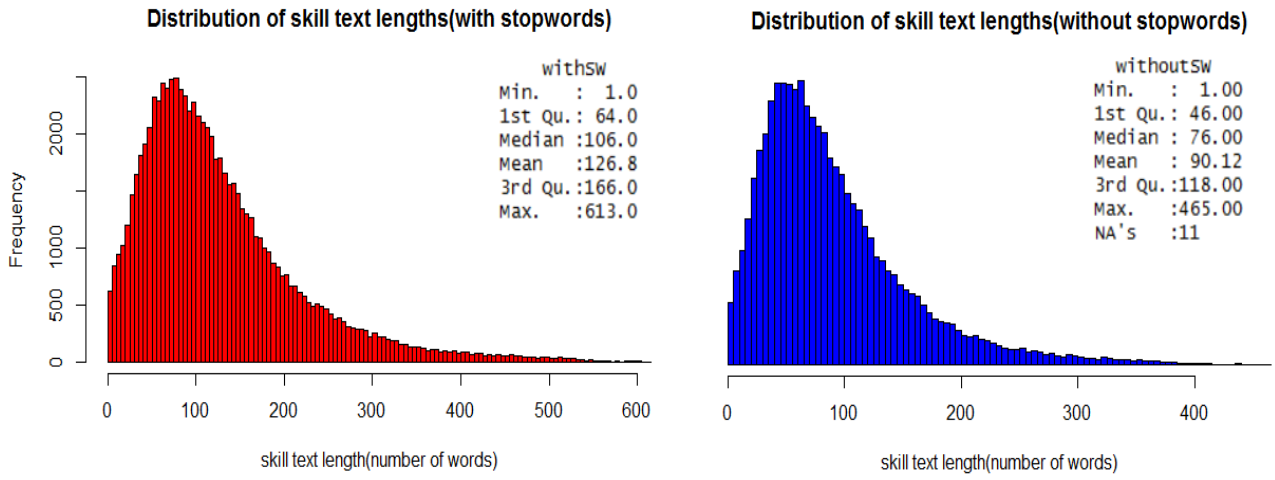


Figure 4.1 Skill text length distributions with and without stopwords

Figure 4.1 shows the skill text with and without stopwords lengths. It can be appreciated that the skillsAndExperience field is a moderate length text, which contains between 90 and 126 words in average, which is a brief description. Besides, the noise introduced by the stopwords is not very relevant in terms of text length.
However, the computation of sparsity in figure 4.2 shows that the noise produced by stopwords in the skillsAndExperience field is small indeed. The sparsity is the percentage of words from the field which give no information at all, that is, stopwords. A mean spartisty of 27.54% makes this field a certainly reliable field which contains a good deal of meaningful words which, being contained in such short texts, suggest that the field will contain a sufficiently specific terminology to provide notable results after text analysis.

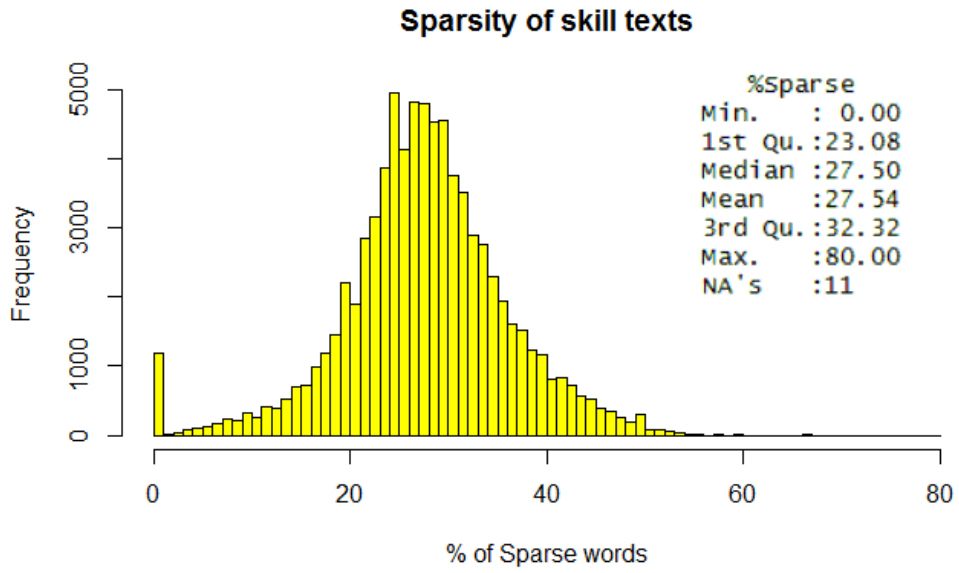


Figure 4.2 Distribution of skill text sparsity

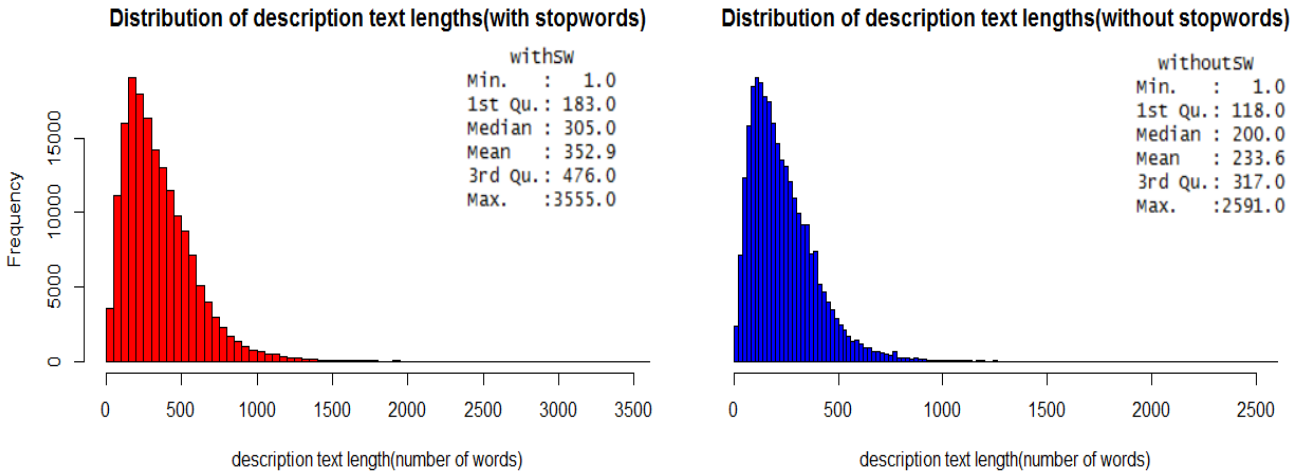


Figure 4.3 Description text length distributions with and without stopwords

Description field

The description field is longer and contains more common use terms than the skillsAndExperience field. Therefore, it contains less specific information as it is a description of the job offer and may contain whatever the employer considers relevant instead of a detailed list of requisites. We repeat the text length analysis to the description field and obtain figure 4.3 which shows the lengths of the text with and without stopwords.

Here, the mean length of texts goes from 234 and 355 terms per text on average, although reaching thousands of words in some cases. Although is a bigger difference with respect to the skillsAndExperience field, this is an expected behaviour as we are dealing with longer text fields which will necessarily contain more stopwords.

In contrast to the skills and experience though, the percentage of meaningless words increases, which suggests that even though texts are longer, they provide less meaningful information, as empty words are more common. In fact, as texts are longer, a more general description of job offers could be expected and thus, a less relevant result would appear from such analysis. Figure 4.4 illustrates this sparsity increase.

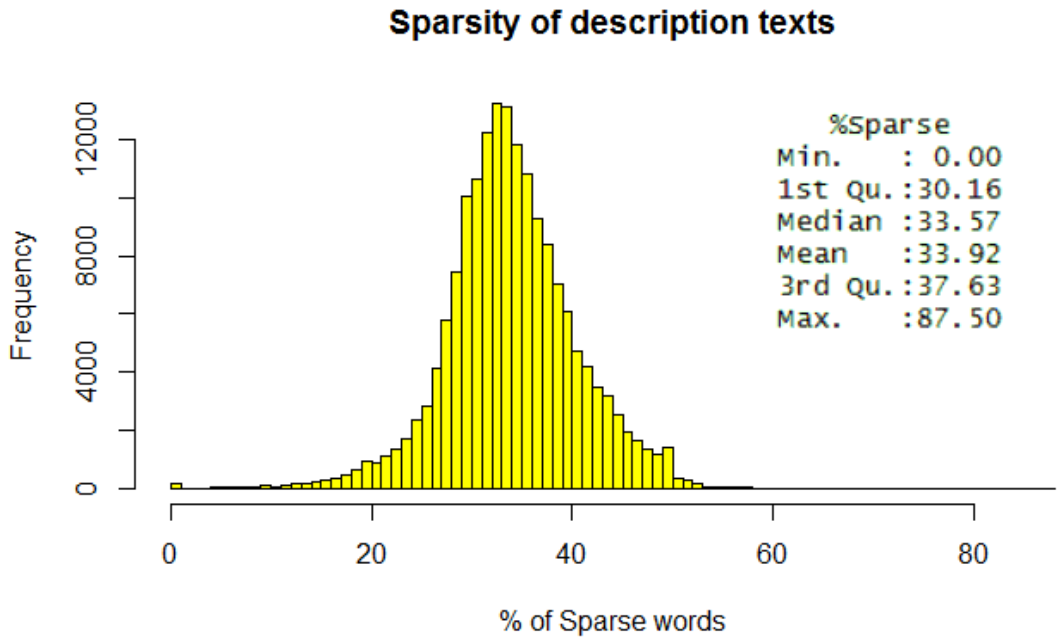


Figure 4.4 Distribution of description text sparsity

4.2.2. Companies offering jobs in LinkedIn

4.2.2.1. Experiment overview

Each job offer in LinkedIn is related to a company present in the Social Network. Thus we can obtain the average number of job offers per company as well as the company more active and offering more job offers.

Performing a simple counting Map Reduce operation with RHadoop we obtain a dataframe containing the company identifier and the number of job offers recorded for that company. Once the results are gathered, the output structure is ordered in decreasing order and by means of the summary function, the basic analyses are performed.

4.2.2.2. Experiment outcome

Each company in LinkedIn posts on average 6.65 jobs, although most companies had post no more than one or two job offers. Figure 4.5 shows the graphical representation of the distributed outcome. The result has the shape of a Pareto distribution, which is a very well-known distribution which usually occurs when resources are distributed.

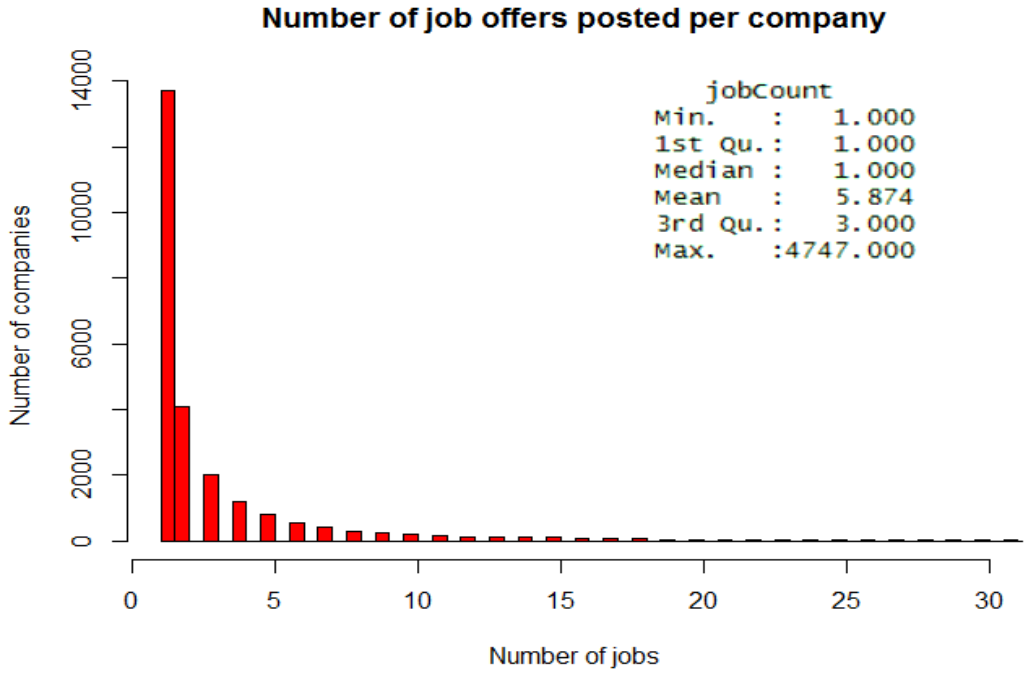


Figure 4.5 Number of jobs posted per company

The basic idea of such distribution is to support the theory that when it comes to share resources, there is always a small fraction of the users of resources who are in control of most resources, leaving the rest of the population the smaller share of resources.

Additionally, the company that has posted more job offers (4747 offers) is Hays (www.hays.com). It is important to remind that this analysis considers both currently active job offers and past job offers and, therefore, this results do not necessarily mean that Hays is currently offering 4747 jobs, but that it has been a very active company in LinkedIn posting job offers and recruiting people.

It can be thus proved that apart from compiling with the Pareto distribution, most of companies in LinkedIn are not very active in job hunting using the tools provided by the social network, as there are several well-known companies with resources showing no interest or giving no importance to job offer posting.

In any case, these results show that even though many companies do not consider seriously LinkedIn recruiting mechanisms, there are other companies very active at recruiting through LinkedIn.

4.2.3. Lifetime of job offers in LinkedIn

4.2.3.1. Experiment overview

Thanks to the posting and expiration timestamps in LinkedIn, the mean lifetime of a job offer can be inferred with a little computing effort. Consequently, we can compute it and obtain the lifespan of each job offer as well as mean statistical parameters.

4.2.3.2. Experiment outcome

The mean job offer duration is 47 days and 22 hours approximately, the median and the minimum job offer duration is 30 days, and the longest lifespan job offer is 12300 days long. Figure 4.6 depicts the lifespan of job offers in months posted in LinkedIn.

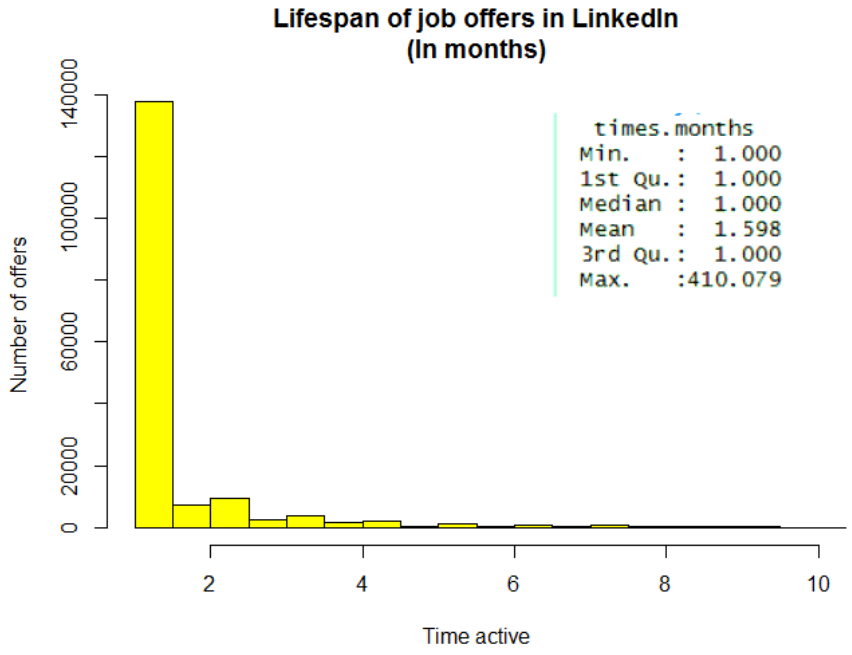


Figure 4.6 Lifespan of Job offers in months

Once again, figure 4.7 shows a Pareto distribution, which shows that most companies in LinkedIn expect to have a suitable candidate within a month whereas a very small amount of them accepts increasing the period. Consequently, we can see that candidate hunting in LinkedIn has an average duration of 1.598 months with a median of 1 month duration. This also proves that LinkedIn is a social network immerse in change and fast development and thus, any company looking for success inside it must be prepared to deal with very fast changes.

This result also gives LinkedIn candidates a good piece of advice: It is crucial to keep up the pace of the network and to apply for the interesting jobs at once, since at any time any offer can be covered or its time expired.

4.2.4. More relevant requirements and keywords in job Offers in LinkedIn

4.2.4.1. Experiment overview

From the field “skills and experience” from the retrieved LinkedIn job offers, we can extract the most demanded skills in the market. Applying our custom variant of the TF-IDF algorithm, where the term frequency is global and the inverse document frequency is computed based on each different job offer, we can obtain a collection of key terms for businesses in the LinkedIn environment.

In the same line, using the field “description” from the user public profiles retrieved from LinkedIn, we can analyse the offered skills by means of the same variant of TF-IDF algorithm to extract another collection of keywords offered by the users of the social network.

In order to count word and document frequency, a MapReduce algorithm is used. This algorithm cleans the text from punctuation signs and common “empty” words and emits values of term frequencies and document counts into an R data structure called data frame. Once we have this data frame, we apply a R script which sorts and performs TF-IDF computation to return another data frame containing each word and its relevance.

4.2.4.2. Experiment outcome

Once the analysis is performed, two dataframes containing lists of key words for each collection can be illustrated using the wordcloud R library. This wordclouds will show at a first sight the key terms of each collection in a very simple way.

Figure 4.7 shows both collections. The left figure shows the result of the person profile collection wordcloud and the right figure shows the result of the job collection wordcloud.

It can be appreciated that the job offer collection analysis results in a greater set of skills which is also less specific than the person profile collection. The skills most valued in job offers are those, which although seeming more generic, are basic requirements for any job. In contrast, the skills offered by candidates are more specific and focused in more professional tasks.

Figure 4.8 illustrates two pies with the 25 most relevant skills as ordered by TFIDF relevance of the skills offered by LinkedIn users (left) and the skills required by employers (right). It can be observed that although there are some matches, most of required skills are different to the ones offered by possible candidates.

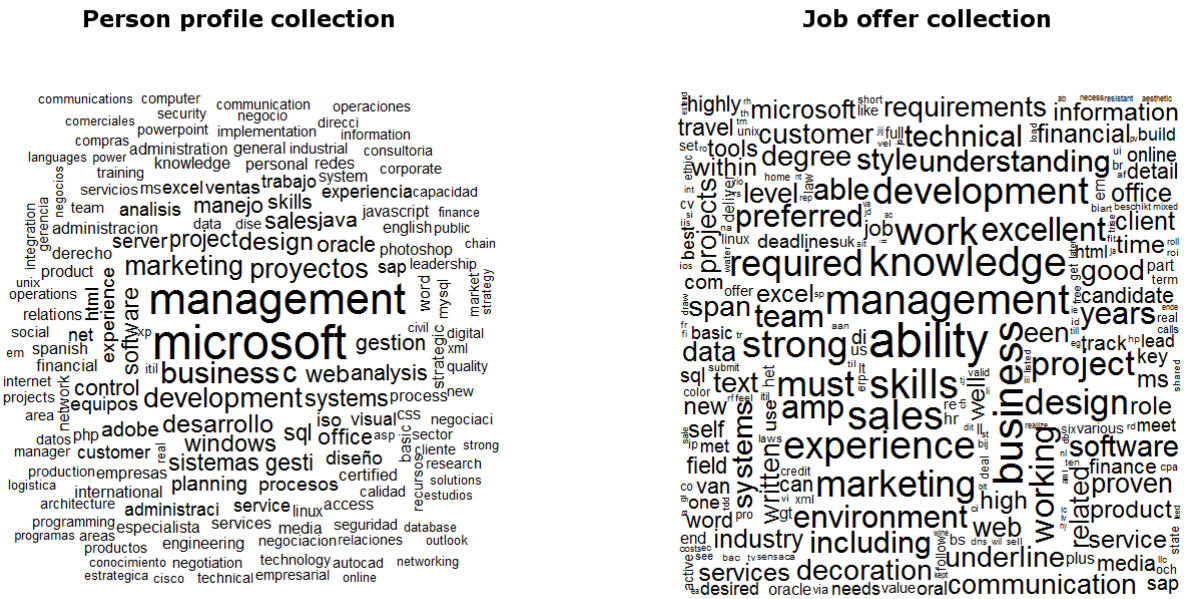


Figure 4.7 Wordclouds from person profiles and job offers

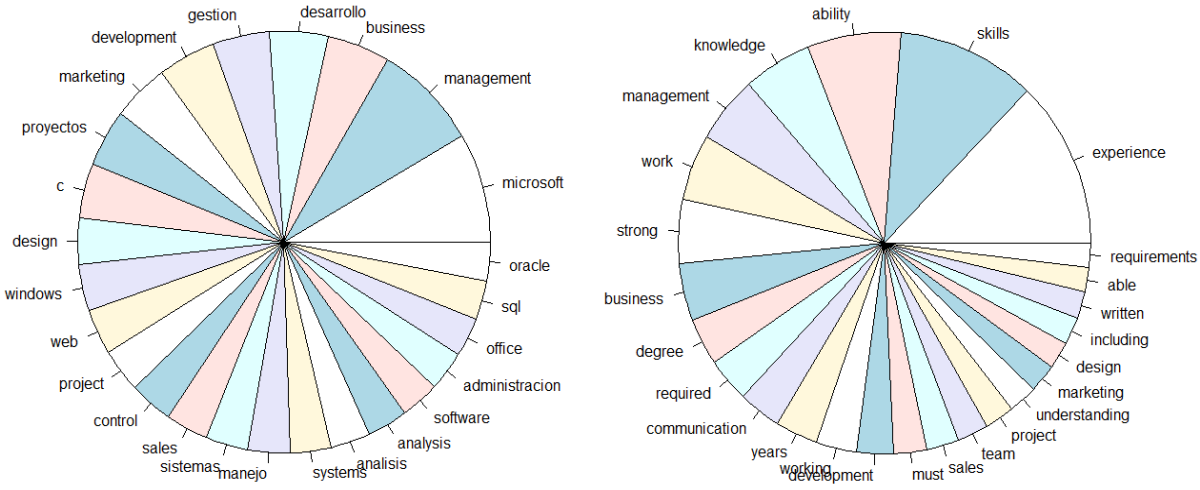


Figure 4.8 Most relevant keywords from profiles and job offers

One of the possible reasons for this result is that both enterprises and LinkedIn users write their profiles or offers with a different focus. On the one hand, Companies tend to focus on teamwork and personality skills, leaving professional skills behind. On the other hand, LinkedIn users, and by induction most of job seekers, focus on what they know, the professional skills they have acquired, forgetting the personal skills they reached.

Therefore, it could be stated that in a Big Data context, job seekers fail to offer in their profiles the abilities companies search. Consequently, the application of matching algorithms will not be totally accurate, although good enough to provide a collection of the better subjects for each job or vice versa.

It is important to remember that we are working over a limited collection of candidate profiles and job offers. Besides, not all candidates work on their LinkedIn profile in the same way. Thus, although the study is reliable, some details may change when the subjected population changes.

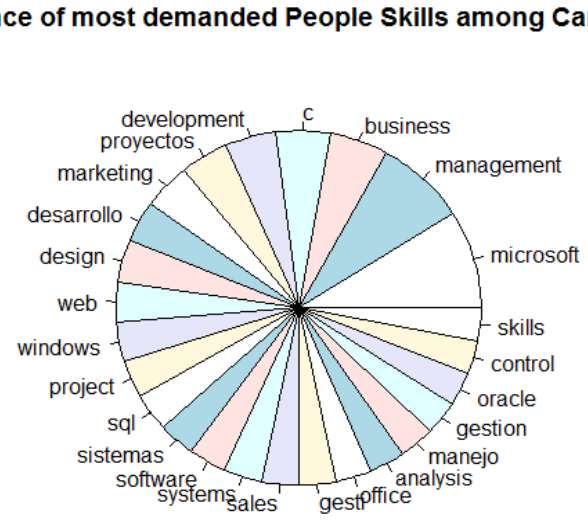


Figure 4.9 Relevance of the 25 most offered skills among candidates in job offers

In addition, figure 4.9 shows the relevance given in job offers to the 25 most relevant skills offered by candidates. As stated, the match is not perfect, but is able to satisfy many of the needs of the job recruiting side, allowing matching algorithms to provide a good result. As an example of the TFIDF algorithm and results, the table below shows the information and computation of the TFIDF of the 25 most relevant skills among candidates.

	Skills	Term Frequency	Number of Candidates	Relevance	Inverse Document Frequency
1	microsoft	605	245	869.5258	1.437233
2	management	816	606	851.8437	1.043926
3	business	394	335	512.7335	1.301354
4	desarrollo	349	299	471.4039	1.350728
5	gestion	346	304	464.8597	1.343525
6	development	338	285	463.5852	1.371554
7	marketing	336	284	461.3550	1.373080
8	proyectos	337	296	456.6711	1.355107
9	c	286	202	435.0196	1.521047
10	design	235	183	367.5277	1.563948
11	windows	224	155	366.4790	1.636067
12	web	242	219	359.6010	1.485955
13	project	241	225	355.2861	1.474216
14	control	237	222	350.7709	1.480046
15	sales	225	192	347.1969	1.543098
16	sistemas	219	185	341.4707	1.559227
17	manejo	222	197	340.0890	1.531933
18	systems	202	155	330.4856	1.636067
19	analisis	216	200	329.4797	1.525369
20	analysis	199	162	321.7599	1.616884
21	software	204	180	320.5098	1.571126
22	administracion	199	181	312.1753	1.568720
23	office	197	179	309.9885	1.573546
24	sql	191	161	309.3384	1.619573
25	oracle	180	130	308.2420	1.712455

4.3. Tecnoempleo job offer analysis

4.3.1. Tecnoempleo basic analysis

4.3.1.1. Experiment overview

Once the Tecnoempleo search page is crawled, there exists a collection 2691 job offers to analyse. From this collection, it is very easy to extract the basic raw data and present it in a more descriptive way.

Additionally, we compute the number of jobs offered by each company and take a look at the distribution they follow.

4.3.1.2. Experiment outcome

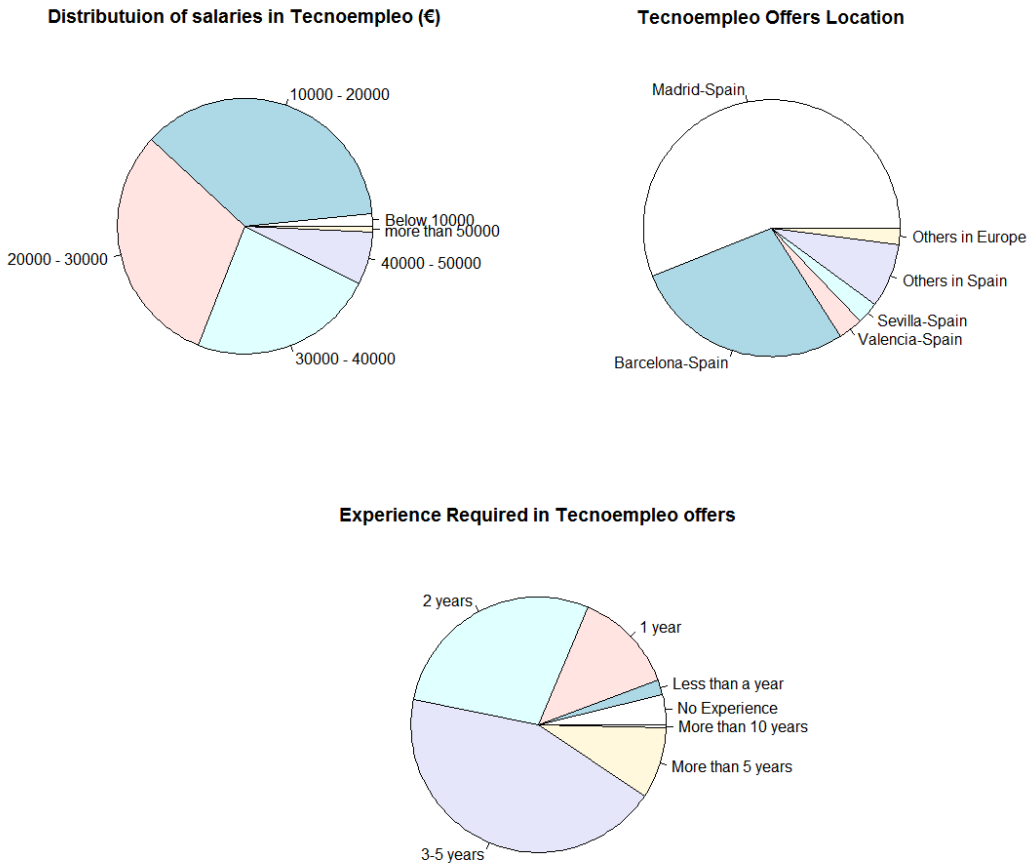


Figure 4.10 Tecnoempleo offers basic parameters

Figure 4.10 shows the basic parameters which can be extracted from Tecnoempleo. As expected, most of job offers are located in either Madrid or Barcelona, which are the biggest cities in Spain. Abundant salaries rest between the 10000€ and 40000€ range and the mean experience required is 3.06 years.

From these results, we can appreciate that almost everything expected applies. Since Madrid and Barcelona are the biggest and more developed cities in Spain, most of the jobs should be located there, followed by Sevilla and Valencia, which are two other important cities. The more common salaries are the ones which offer an acceptable amount of money and the experience required do not surpass 5 years, being very uncommon not requiring any experience at all.

Besides, there exists 390 companies offering positions in Tecnoempleo. Such enterprises have published on the web page at least one active job offer. Figure 4.11 illustrates how many enterprises offer how many jobs.

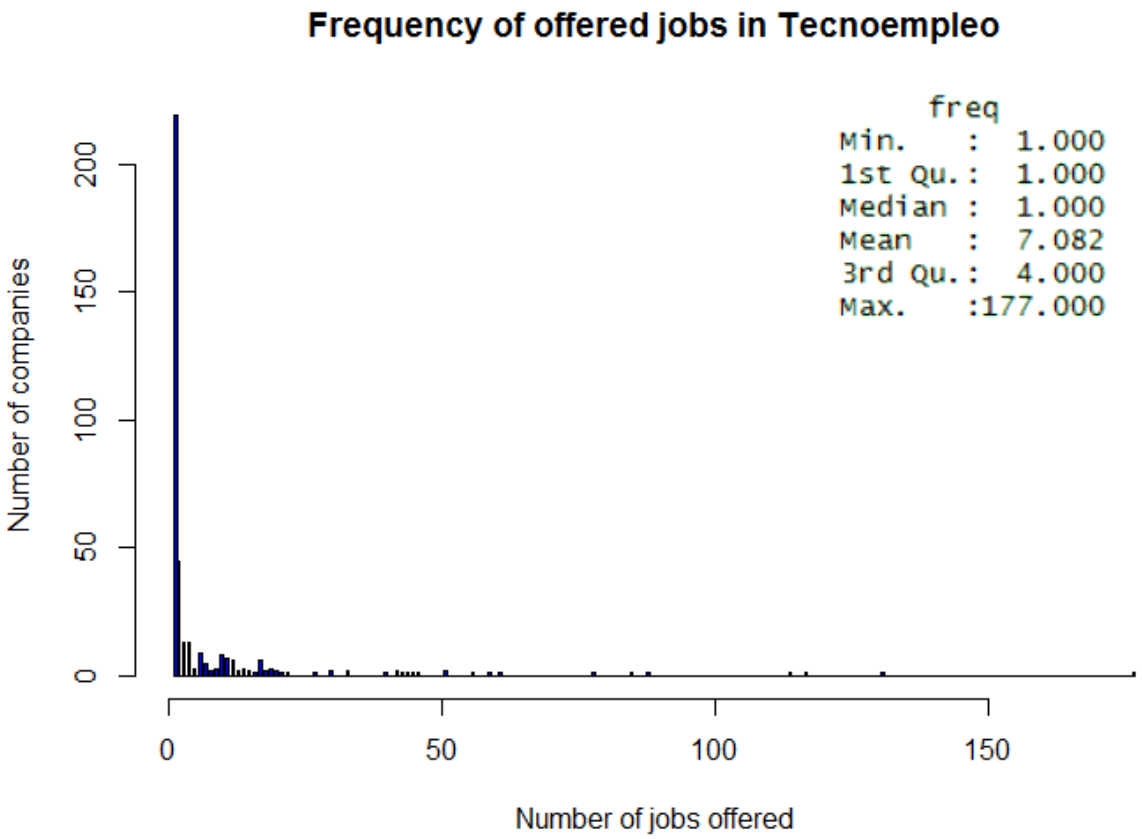


Figure 4.11 Number of jobs posted per company

Once again, Pareto distribution appears again, and shows that most enterprises offer, at most, 5 active job offers whereas some few companies surpass the 50 active job offers at the time. In contrast to LinkedIn analysis, here only active offers are shown, as the Tecnoempleo page only displays interesting offers for its users.

The fact that companies in Tecnoempleo have the same behaviour than the ones in LinkedIn suggest that this is a common behaviour in recruiting web services. In fact, it seems very reasonable that many enterprises here provide a modest amount of offers, since most of the companies belong to the group known as PYMES, which refers to small and medium size enterprises in Spain. Consequently, these PYMES are unable to perform big job offers, as they have no resources nor such a big employee need.

4.3.2. More relevant technological and profile terms

4.3.2.1. Experiment overview

Using packages wordcloud and tm from R and the collection of required technologies in each job offer from Tecnoempleo, it is easy to infer the collection of technologies most demanded in the site. Additionally the profile requirements collection can be also inferred and compare it to see whether the most demanded technologies match the profile requirements.

4.3.2.2. Experiment outcome

Technological requirements **Profile requirements**

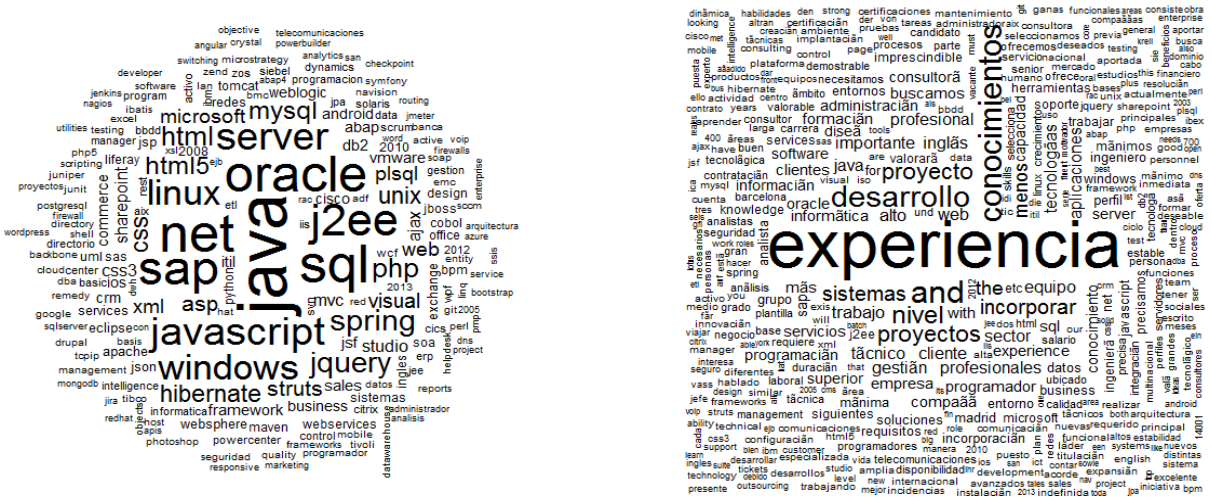


Figure 4.12 technological and profile requirement wordclouds

Figure 4.12 depicts both wordclouds, the one for technological requirements on the left and the one for profile requirements on the right. This result suggests, as in LinkedIn, that companies seek common personal skills and capabilities more than professional skills. Although Tecnoempleo offers an especial section for professional skills, the profile, the expected behaviour and capabilities of any candidate is hardly conditioned on keywords which are not strictly related to any field.

Nevertheless, the technologies collection of keywords shows clearly that Tecnoempleo is a highly-oriented professional environment which focuses on technological knowledge about web development. Thus it would be the perfect job finder for any candidate involved in programming, database management or even system administration.

4.3.3. Comparison of experience against formation

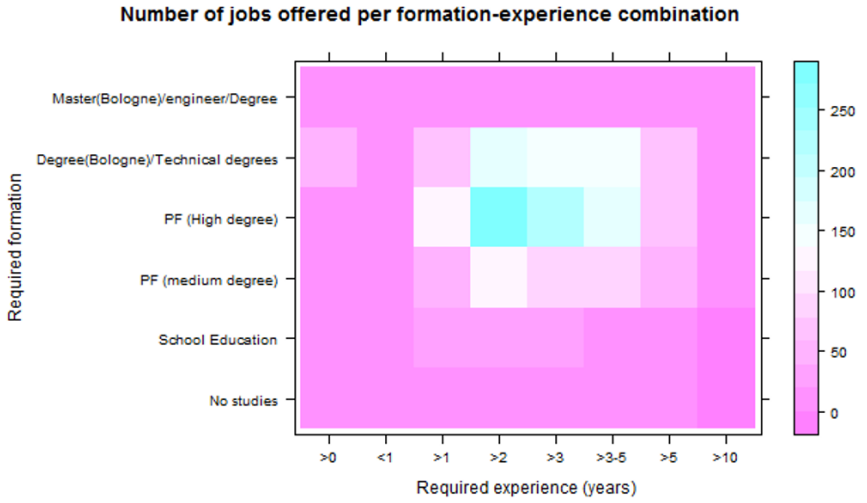
4.3.3.1. Experiment overview

One of the key aspects to be taken into account when either recruiting new workers or applying for any job is the importance given to experience and formation. Both are usually considered for any candidate purposed, but it is hardly ever clear which of both receives more importance by employers.

The aim of this experiment is to unveil which of the later is better considered and helps when achieving better positions inside a company. For that purpose, we will analyse Tecnoempleo collection to cross-correlate experience and formation in order to obtain quantitative measures like the number of jobs offered per formation and experience combination as well as the salaries offered for them.

4.3.3.2. Experiment outcome

Once processed, we obtain an experience-formation matrix which can be easily depicted as a color map using the package “lattice” for R inside our framework. Consequently we can show results for each formation-experience pair of values as the number of jobs offered(figure 4.13) and the mean salary rewarded in positions requiring such formation and experience (figure 4.14).




```
  No studies     School Education PF (medium degree)
Min.   : 0.000   Min.   : 0       Min.   :  2.00
1st Qu.: 3.250   1st Qu.: 4       1st Qu.:  8.75
Median : 5.500   Median : 7       Median : 56.00
Mean   : 6.125   Mean   :14       Mean   : 54.38
3rd Qu.:10.250   3rd Qu.:26       3rd Qu.: 85.75
Max.   :12.000   Max.   :34       Max.   :129.00

PF (High degree) Degree(Bologne)/Technical degrees
Min.   :  4.00   Min.   :  4.00
1st Qu.: 15.75   1st Qu.: 36.00
Median : 95.50   Median : 63.50
Mean   :109.12   Mean   : 78.25
3rd Qu.:173.00   3rd Qu.:138.00
Max.   :271.00   Max.   :161.00

Master(Bologne)/engineer/Degree
Min.   : 1.00
1st Qu.: 7.25
Median :11.00
Mean   : 9.75
3rd Qu.:13.00
Max.   :16.00
```

```
      >0              <1              >1              >2              >3              >3-5            >5              >10
Min.   : 4.00   Min.   : 1.000   Min.   :  4.00   Min.   : 12.0   Min.   : 10.00   Min.   : 10.00   Min.   : 4.00   Min.   :0.000
1st Qu.: 5.75   1st Qu.: 4.250   1st Qu.: 17.50   1st Qu.: 20.0   1st Qu.: 13.50   1st Qu.: 11.50   1st Qu.: 7.75   1st Qu.:0.250
Median : 8.00   Median : 7.000   Median : 44.50   Median : 80.5   Median : 57.50   Median : 48.50   Median :35.00   Median :1.500
Mean   :14.17   Mean   : 7.333   Mean   : 48.33   Mean   :103.5   Mean   : 82.83   Mean   : 68.67   Mean   :35.50   Mean   :1.833
3rd Qu.:13.25   3rd Qu.: 9.000   3rd Qu.: 61.75   3rd Qu.:153.0   3rd Qu.:130.75   3rd Qu.:123.00   3rd Qu.:61.50   3rd Qu.:3.500
Max.   :45.00   Max.   :16.000   Max.   :121.00   Max.   :271.0   Max.   :218.00   Max.   :158.00   Max.   :70.00   Max.   :4.000
```

Figure 4.13 jobs offered according to experience and formation

PF means Professional Formation and it is the Spanish name for non-university after school education for non-academic disciplines. All titles which are related to Bologne refer to the new European plan for University education.

If we consider figure 4.13, we can appreciate that job offers highly depend on the formation required, since the jobs offered for a certain position are more or less proportional for the experience period increase, whereas the increase of experience do not entail any increase in jobs offered. However, figure maximum is located in the middle area of the graph, which suggests that formation may not be so relevant as it seems, since better formation does not necessarily guarantees better positions.

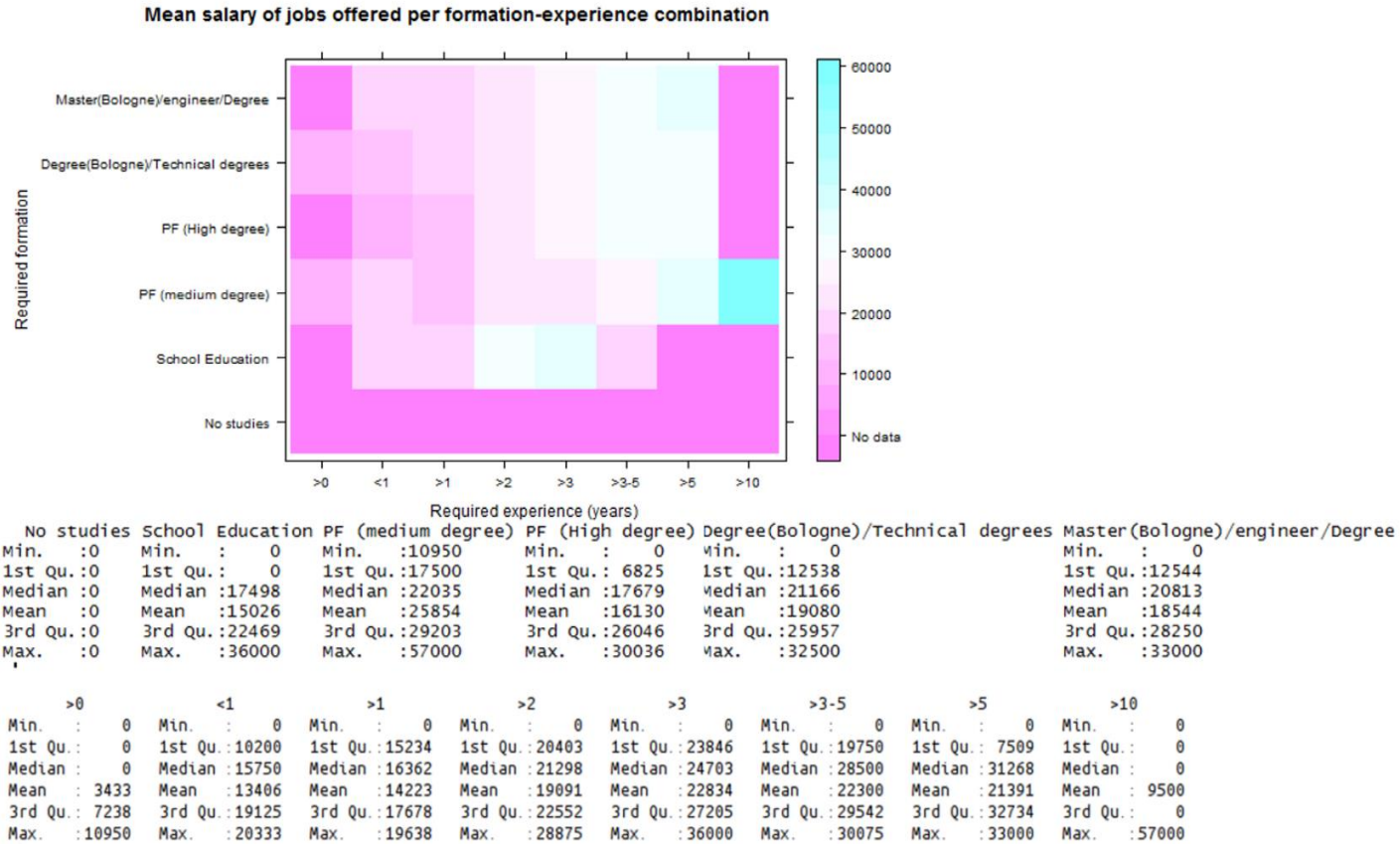


Figure 4.14 salaries given to positions requiring certain experience and formation

Figure 4.14 depicts a more complex result, as the salaries earned by all jobs offered seem to increase according to experience more than formation. The salary count increases according to the years of experience of the candidate equally for all kinds of formation. In fact, the maximum medium salary offered is one with a lot of years of experience but not a so high formation level.

In contrast to figure 4.13, figure 4.14 offers a good appreciation of how experience is better considered than formation. Additionally, mean salary is a better measure of relevance than jobs offered, as the collection in Tecnoempleo may lack job offers for certain sectors of experience or formation whereas average salary gives a more fair result even if there is only one occurrence of each experience and formation pair.

Therefore, experience is more valuated when it comes to remuneration, although there do not exist such as many job offers for positions of those characteristics. In fact, one of the most surprising results is that the most valued formation, at least in Tecnoempleo, are the Professional formation titles, both in salary and number of jobs. This is a bit strange as there are available other titles which traditionally provide greater remunerations.

4.3.4. Tecnoempleo offers clustering

4.3.4.1. Experiment overview

Using hierarchical clustering, any collection of documents can be clustered according to the existent relation among its words with the words of other documents. Hierarchical clustering separates each document to one single cluster, so a differentiation among documents from different clusters should be appreciated.

Thus, once the documents are separated by clusters, a wordcloud of each cluster can easily be invoked in order to have the possibility of visually analyse each of the clusters and see whether the separation is relevant or it does not differentiate well.

4.3.4.2. Experiment outcome

Using tm library in R and with the Tecnoempleo collection as input data, all the technology fields in Tecnoempleo are processed, cleaned and stripped of stopwords and additional spaces.
In order to decide the number of clusters to create, we need to consider the within groups sum of squares, which is a mathematical measure for the variance of the distance to each document to

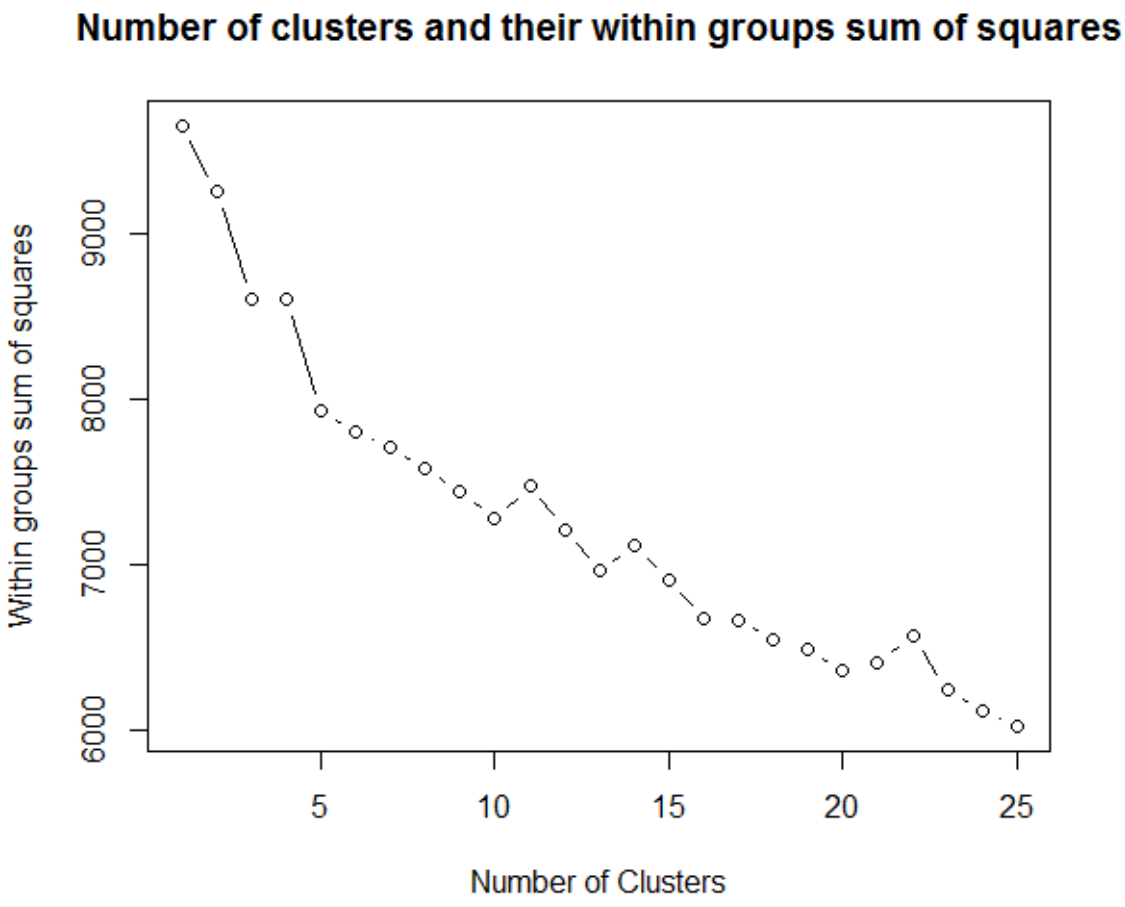


Figure 4.15 Within groups sum of squares for the Tecnoempleo technologies collection

its cluster centroid. In order to decide, we want our elements inside a cluster to be as close to the centroid as possible, but on the other hand, we do not want to raise too much the total number of clusters, as given that case, the clusters would provide very little information.

The computation of the within groups sum of squares is very simple and may be performed with a simple R algorithm. The formula for such calculus is the following.

$$wss = (N - 1) * \sum_{1}^{T} Var[t]$$

where N is the total number of documents in the collection, T is the total number of terms in the document-term matrix and Var[t] is the variance of each term value over the whole set of documents. Once the computation is finished, we can plot the results in a graph such as figure 4.15.

From figure 4.15 we easily observe that at a first glance, the most clusters created, the least the within groups sum of squares is. Although this is true, a greater number of clusters implies a bigger degree of separation and thus makes our trial of grouping documents together worthless.

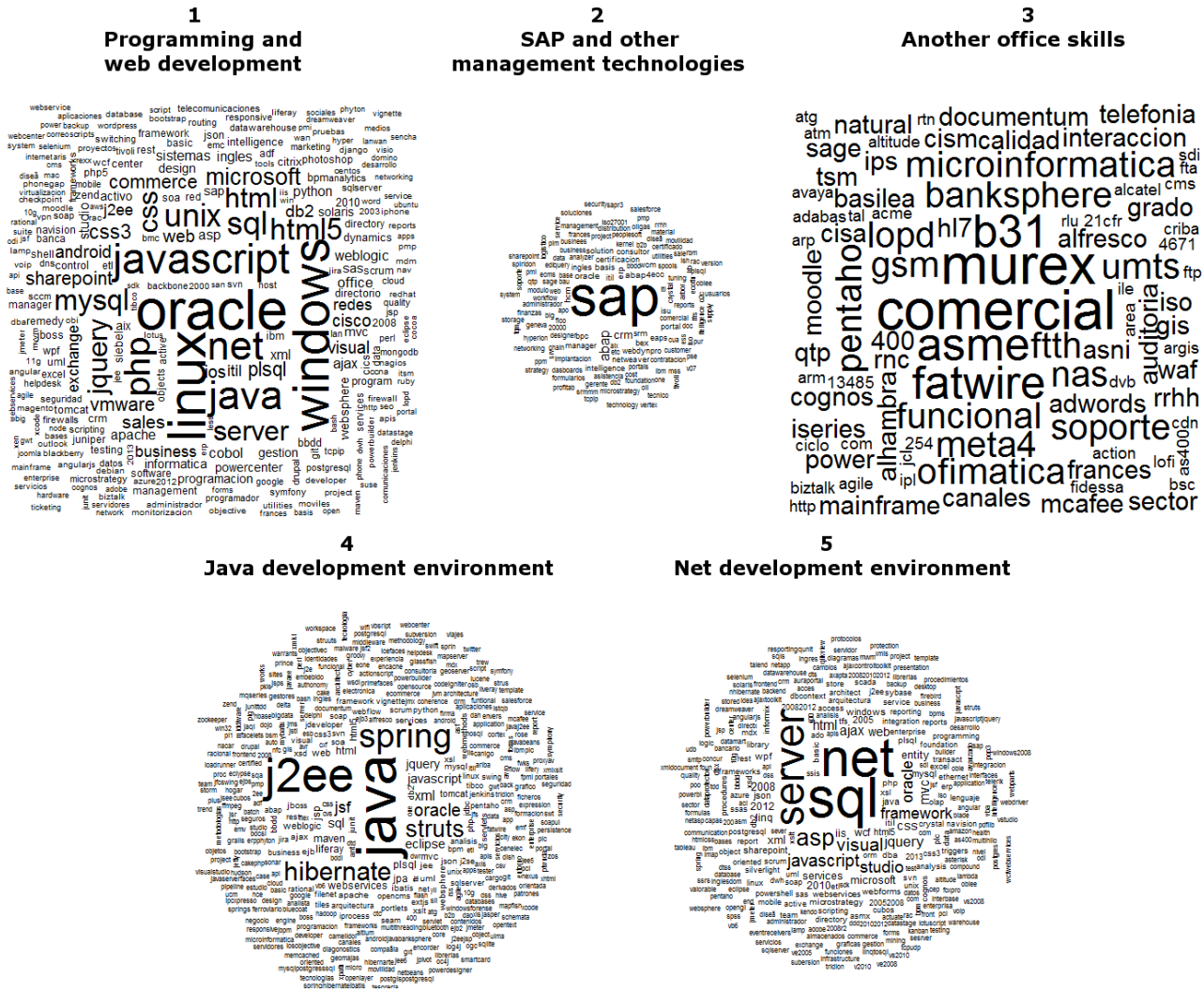


Figure 4.16 Wordclouds of the technology clusters in Tecnoempleo

For that reason, selecting between five and ten clusters seems to be a good choice, since less than five clusters increase too much the distance of each document to its centroid and more than 10 clusters do not entail such a distance decrease.

For this experiment, five clusters where chosen, as the result is easily recognizable and separable. Once the clustering, based on tm library, is performed, wordclouds from each cluster may be plotted, grouped and named, as shown in figure 4.16.

Each of the wordclouds represent one cluster. In spite of existing at least 3 clusters very similar (1, 4 and 5) they represent to different development environments, which makes them relevant for the analysis.

Cluster 1 is devoted to job offers which focus on programming and general web development, which includes all basic web development technologies, systems and environments. Cluster 2 is much related to SAP and the business management ecosystem. Cluster 3 contains other office skills, which are related to office suites, computer support, telephony or languages.

Clusters 4 and 5 are devoted to two specific technologies for web development: Java and Net respectively. Both ecosystems have acquired such relevance that are two of the most deployed and considered solutions in many companies, so it is natural their appearance as their own clusters. In fact, they have become so relevant that they take a 23.78% of the Tecnoempleo job offer collection.

Finally, once the clusters are defined, it is possible to separate job offers according to their clusters and compute the average expected salaries for each professional profile. Figure 4.17 provides an overview of the salaries offered for some of the offers in each cluster collection.

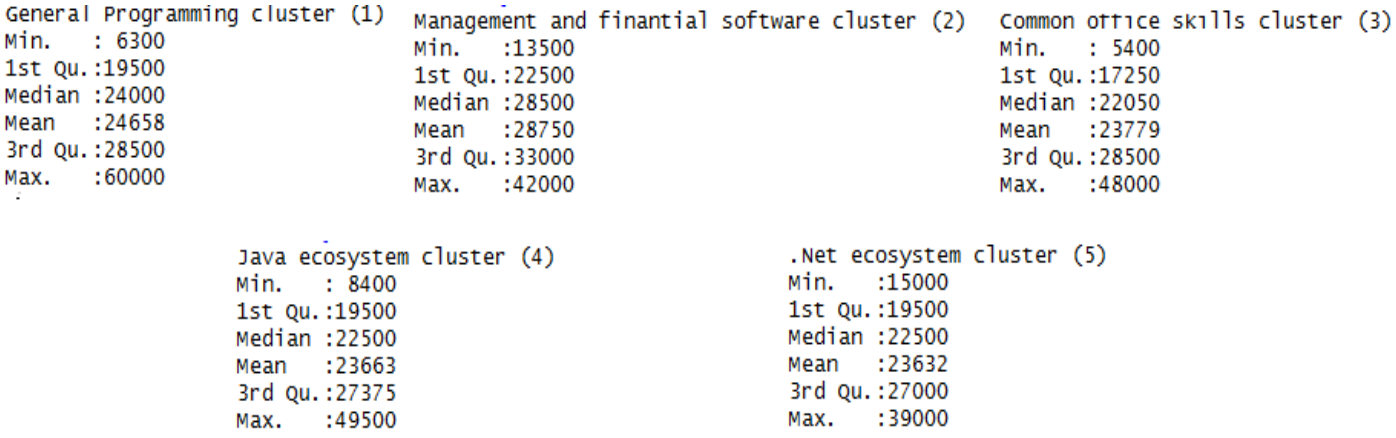


Figure 4.17 Salary summary of each of the professional groups inferred by clustering

From the image we can see that salary ranges are very similar. In case of cluster two, there are many high salaries (as the third quartile overpasses 33000 euros), which suggests that, in general, SAP technologies and environment offers better salaries than other professional profiles.

However, the greater maximum salary is in professional profile one, which suggest that cluster one offers jobs to a more general profile of employer and so it includes a bigger range of positions which require different qualifications and skills.

In order to provide a broader view on this clustering and to give the reader all the facts involved in the experiment, the following table includes additional data on the clusters.

Cluster	Number of offers	Percentage	Salary offers
General programming cluster	1616	60.05%	318
Management cluster	243	9.03%	30
Common OS cluster	192	7.13%	34
Java ecosystem cluster	427	15.87%	76
.Net ecosystem cluster	213	7.92%	53

5. Prototype

After achieving the previous results and implementing the Okapi BM25 scoring algorithm, the possibility of developing a web service based on a keyword ranking from the job offers stored seemed viable. Therefore, we created a very simple web service capable of searching for the most apt job offers for the profile of the studied candidate.

However, creating a simple web search engine was not the objective, since there exist several sites devoted to search and extraction of job offers based on user queries. Due to this, the service offers a job finder based on LinkedIn user profile. In order to obtain a suitable result for each individual, the user must log in the LinkedIn application with his credentials and authorise the service. Once correctly authorized, the service performs an Okapi BM25 word ranking based on different fields of the user profile.

5.1. Prototype design and implementation

Based on the joint collection of LinkedIn and Tecnoempleo job offers, we can provide data analysis to find the most appropriate job offers according to the qualifications and experience of a given candidate. For this purpose, the only requirement is to create a query containing the most relevant terms for the candidate and have a term-scoring algorithm prepared to score all terms in job offers.

In order to score terms, it was decided that Okapi BM25 algorithm was the best option, as performs a TFIDF-like scoring considering text lengths, which is a crucial element in such a diverse collection. Thus, Okapi algorithm was implemented and tested inside the framework and an R function which receives as parameters the collection term structure and a query and returns a dataframe containing the documents ordered by Okapi relevance becomes available for this purpose. Figure 4.13 contains a code snippet of such function.

Finally, instead of creating queries from a given user search, we found very useful the LinkedIn user profile API, which provides functions to retrieve all the information needed from any user who gives authorization. Consequently, a query based in several fields from LinkedIn profiles is created each time any user uses the application. The algorithm for query creation is still very simple and basically merges some fields, but improvements based on TFIDF ranking of such fields or stopword removal are considered and listed for future improvements. The drawback of such model is that a LinkedIn account is required in order to use the application, even though LinkedIn is a very relevant and implanted subject in many professional environments.

Once this is achieved, it is only necessary to wrap everything into a user-friendly environment, such as a web service and create the proper interfaces and connections to make everything work. For this project, we have implemented a web service prototype for this functionality.

The interface and design of the service must be prepared for simple use and should not implement anything apart from the basic requirements for the service. These services include:

LinkedIn authentication and authorization, query development and job offer retrieval. It is important to bear in mind that this service is a prototype which has to be improved and extended before being deployed for production.

```
query<-function(query, values.df)
{
  #free parameters
  k1=1.2
  b=0.75
  #basic needs
  docCount.df<-count(values.df$val)
  word.length<-count(values.df$key)
  avg.docLength<-mean(word.length$freq)
  word.length$freq=1-b+b*word.length$freq/avg.docLength
  #process query
  query<-gsub('[[:punct:]]','', query)
  query<-tolower(query)
  query<-gsub(' $', '', query)
  query<-gsub('^ ', '', query)
  val<-unlist(strsplit(query, split=" "))
  query.terms<-as.data.frame(val)
  query.terms<-query.terms[!query.terms$val %in% myStopwords,]
  query.df<-values.df[values.df$val %in% query.terms,]

  #rownames<-as.vector(count(query.df$key)$x)
  docCount.df["idf"]<-log(nrow(word.length)/docCount.df$freq)
  colnames(query.df)[3]<-"tf"
  query.df<-merge(query.df, word.length, by.x="key", by.y="x", all.x=TRUE)
  colnames(query.df)[4]<-"wl"

  #Actually perform okapi BM25 algorithm
  query.df["okapi"]<-docCount.df[query.df$val, "idf"]*(query.df$tf*(k1+1))/(query.df$tf+k1*query.df$wl)

  #To split according to values, create factors in query.df$val
  query.df$val<-factor(query.df$val)
  splitted<-split(query.df, query.df$val)
  #split + lapply to arrange column names
  splitted<-lapply(splitted, function(x)
  {
    x[as.character(x$val[1])]<-x$okapi
    #we do not want these rows any longer
    x$okapi<-NULL
    x$val<-NULL
    x$tf<-NULL
    x$wl<-NULL
    return(x)
  })
  #merge everything (use Reduce with merge to merge n dataframes)
  okapi = Reduce(function(...) merge(..., by="key",all=T), splitted)
  #now prepare the result matrix presentation
  row.names(okapi)<-okapi$key
  okapi$key<-NULL
  okapi[is.na(okapi)]<-0
  okapi["total"]<-rowSums(okapi)/length(query.terms)
  okapi<-okapi[order(okapi$total, decreasing=TRUE),]
  return(okapi)
```

Figure 5.1 Okapi implementation code

The service currently includes job offers from LinkedIn and Tecnoempleo which are stored in a MongoDB collection optimized for this service. Additional sources can be added easily by storing the required information in this collection. Detail captures on the collection may be found in annex II (AII.4).

In order to illustrate the architecture, integration and technologies involved in the prototype design, figure 7.2 shows a schema of the whole system.

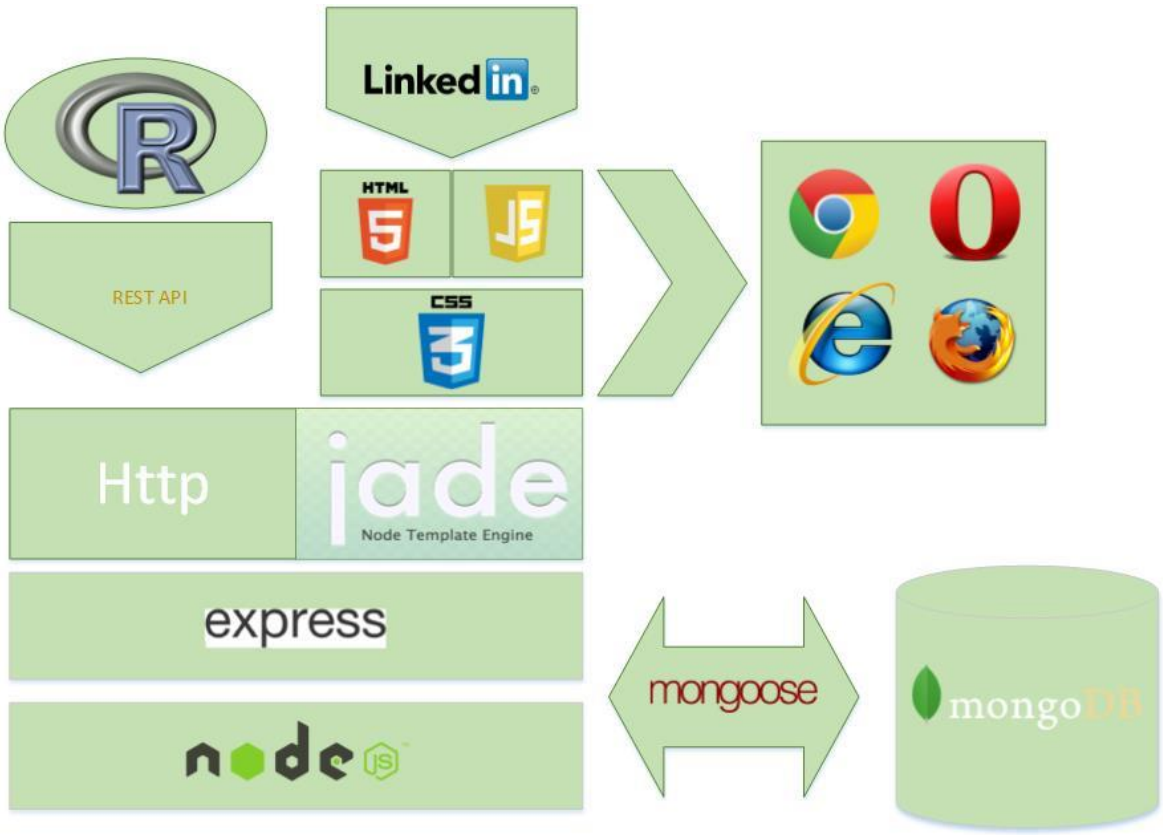


Figure 5.2 Prototype architecture schema

5.2. Use case

The first view to appear when we browse the web page is a very brief description of the service and an authentication button (Figure 7.3).

Figure 5.3 Application index

If we press this button, an auto generated LinkedIn Authorization window appears and guides us through the process of logging in to the application. Figure 7.4 shows a capture of such logging in.

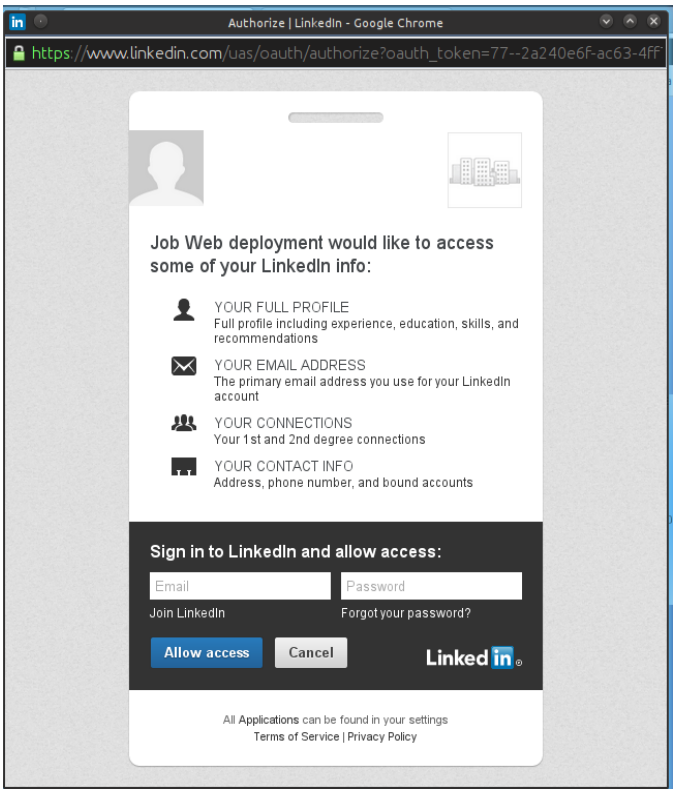


Figure 5.4 Application logging with LinkedIn

Then, after correct authorization, the profile will be analysed in order to create a suitable query for Okapi BM25 ranking. A waiting icon will appear in the webpage until results are ready to be shown and they will be displayed, as shown in figure 7.5.

In order to show whether a user is connected as well as the user whose profile is considered, a very simple detail is displayed on the right side of the view. Additionally, a logout button allows the user to exit his account and repeat the search or try with another account.

The results are paginated. Each page will show 10 results in decreasing order of relevance (Oakpi score) starting in page 1. There is a limit for the prototype of 5 pages of results, although the program support the retrieval of more than 50 job offers.

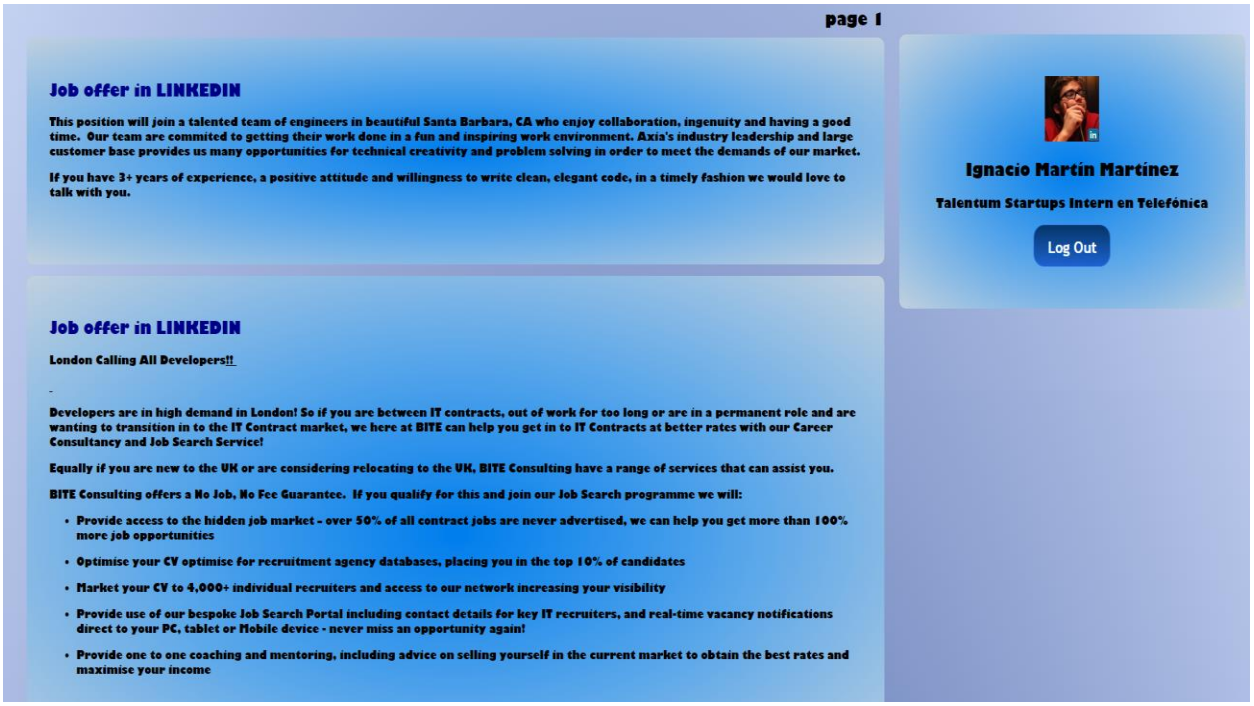


Figure 5.5 Search result view

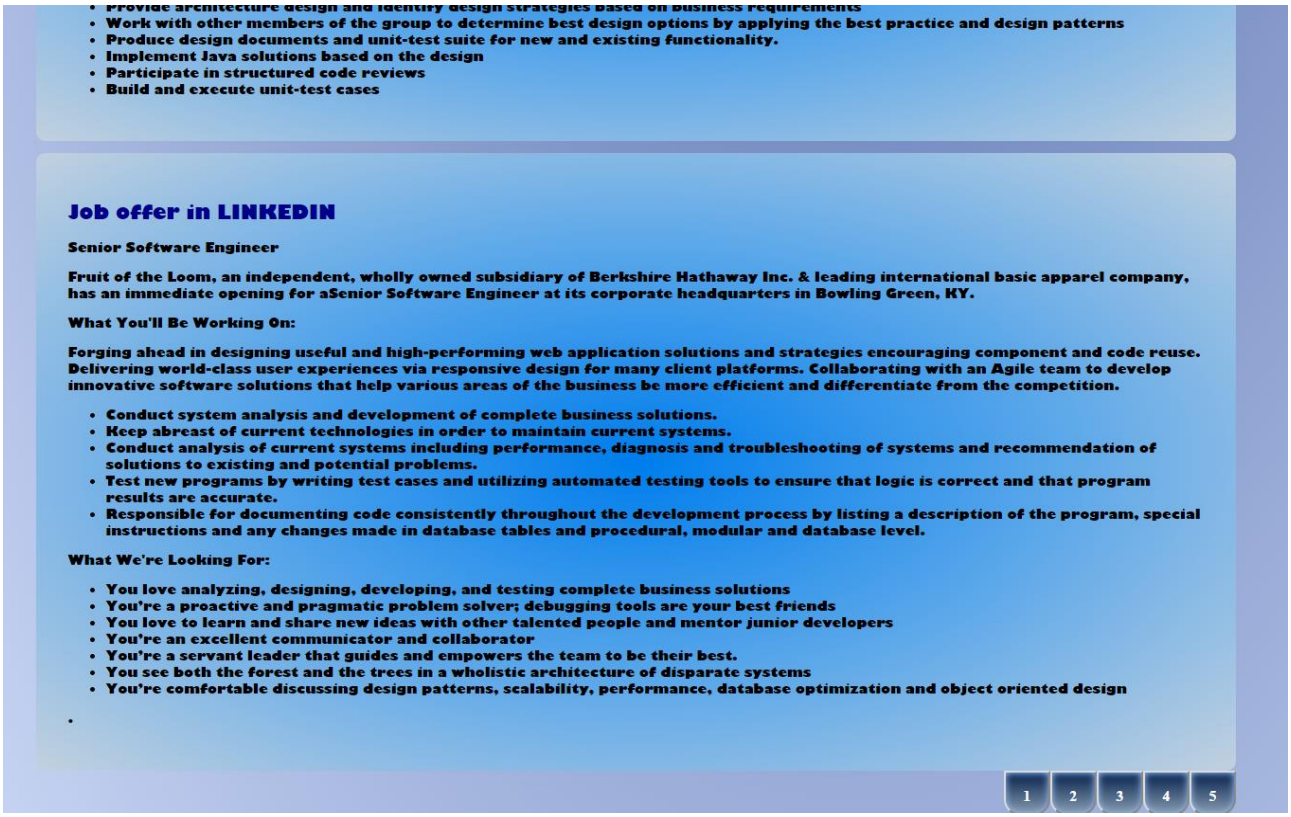


Figure 5.6 Pagination options detail

Figure 7.6 shows the pagination detail at the end of the search results page. The number of the current page on display is on the top of the view, as shown in figure 7.5. Whenever a new page is selected, the okapi algorithm is recomputed.

5.3. Example

Once the application is settled and deployed into the server, we can use any LinkedIn account to provide it with data to create a query and obtain the best results. As a demonstration, we will make and document a sample query using the fake LinkedIn account created for the project. Such account was created to interact with the LinkedIn API and its profile may be altered to our convenience.

For this example, we will develop a profile of project team leader with telecommunications and team working background. Figure 5.7 shows a capture of the LinkedIn profile of such candidate. It is important to bear in mind that this is an emulation of a user, and thus, the profile does not contain a full candidate profile but instead a proof of concept prepared profile.

In figure 5.7 can be appreciated that the candidate has written an extract talking about its background and experience, at least one job position and some skills sent in the skill field.

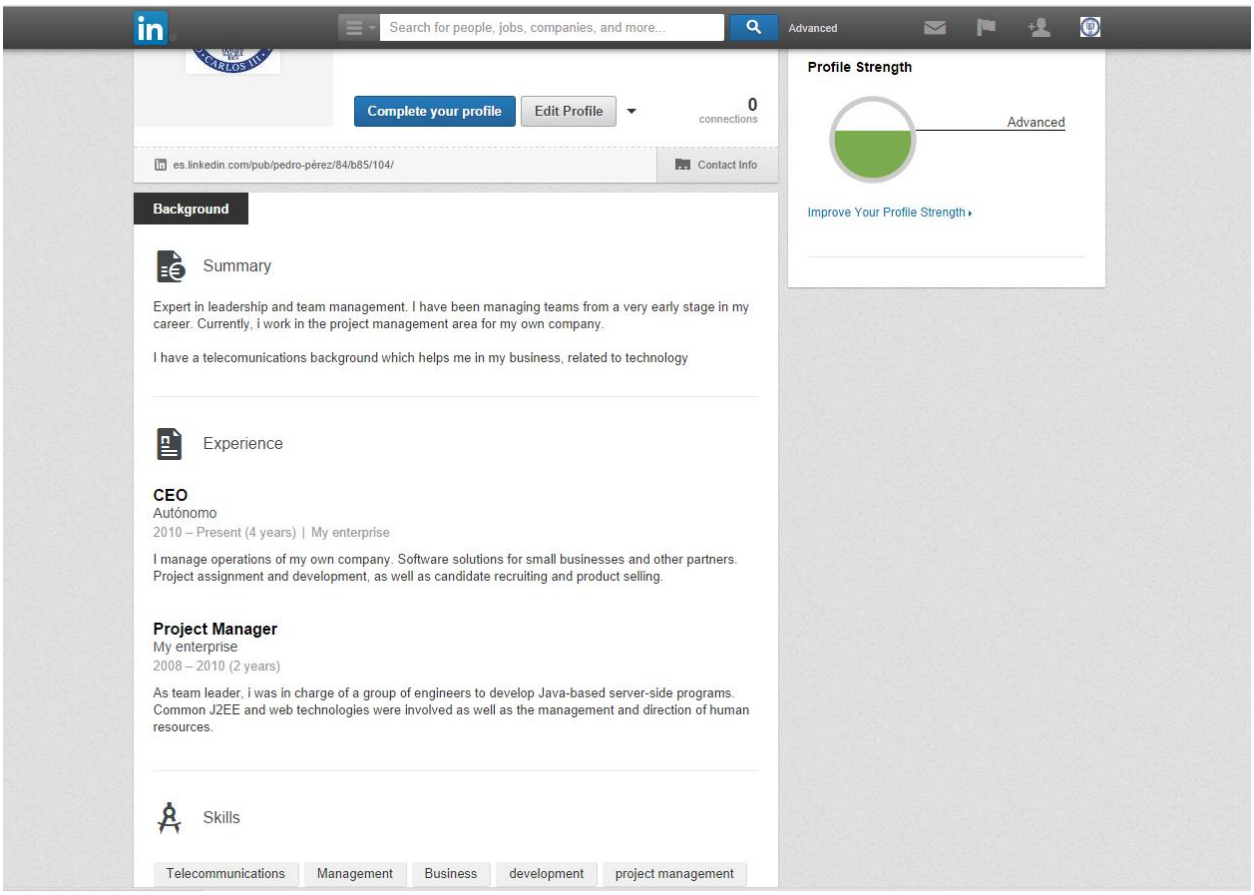


Figure 5.7 Capture of the LinkedIn Management profile for application testing

Once the query is executed the results are shown as in figure 5.8. Additionally, and in order to also try results obtained in experiment 4.3.4, we will feed the application with a profile in LinkedIn containing the basic technologies and experiences from cluster four, the Java ecosystem cluster.

As expected, results shown in figures 5.8 and 5.9 are quite different, as they address different profiles written in a different way and that for almost sure belong to candidates with different job hunting interests and careers

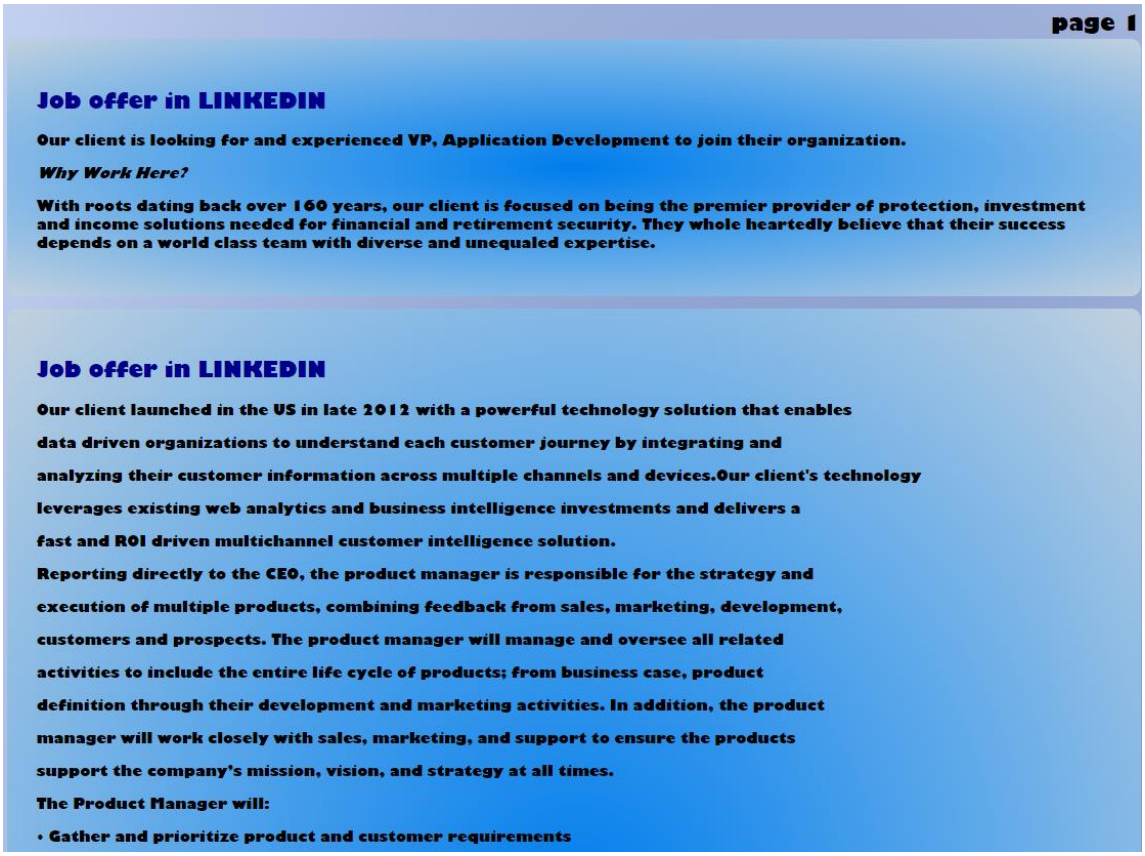


Figure 5.8 Application output for a management profile

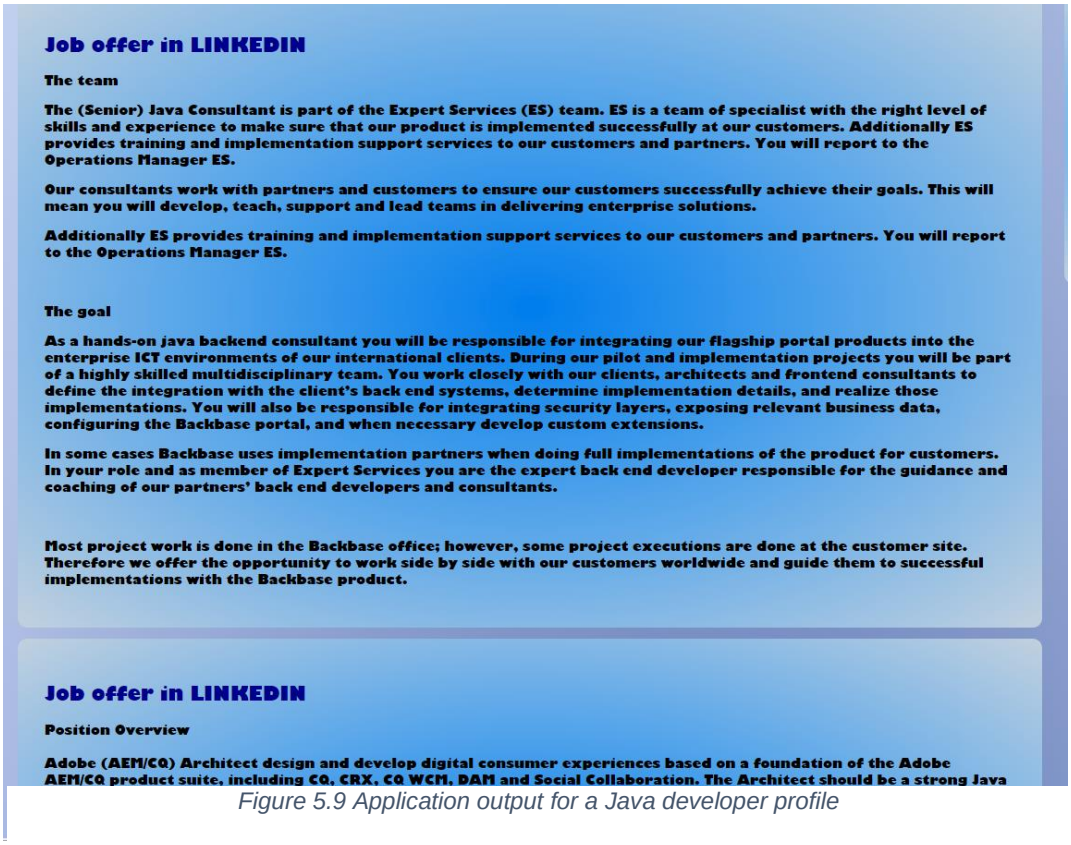


Figure 5.9 Application output for a Java developer profile

6. Conclusions and Future Work

To conclude this report, it is important to add some global considerations and conclusions to be considered. The project started as a research project seeking for new applications and developments based on Big Data, and after all the research and work undertaken, it is ending with satisfying results. All objectives have been achieved and many unexpected results have been reached, as well as many others which were expected.

Certainly, the main conclusion that could be extracted at the end is that Big Data technologies together with a good data set offer almost unlimited opportunities for new developments and applications. Furthermore, such applications can be also applied to other data sets to contrast and complement them and extract new conclusions.

Moreover, Big Data enhance data processing techniques to the point of increasing their efficiency and thus contributing to enlarge companies' profits, since data processing is becoming a more important subject in all modern companies. Besides, there is a huge environment behind Big Data techniques with tons of different solutions and implementations and many more appearing each day, so Big Data processing is also becoming easier and more common.

Concretely, in the scope of this project, we can state that Big Data is a powerful tool for job hunting and human resources applications, as it allows to compare and contrast much more job offers or candidates in much less time. In fact, the applications described in this report are just a small subset of all the possibilities open for study and development.

Actually, all the studies and applications described are the most interesting and the ones adjusting to our planning of all the ones we had in mind. Nevertheless, further work and ideas should be considered, since the data set contains much more information.

Essentially, the data set retrieved contains information used for this project that can be correlated in a different way or with other data to obtain more results. Additionally, there is a good collection of company information stored which has not been processed due to the lack of time and resources.

Furthermore, the application prototype should be extended and improved in order to offer new functionalities, such as candidate finder, enterprise finder, new opportunities or better performance in the service. Besides, other applications and web services could be developed, since the prototype developed covers one of the opportunities that arise in this context.

The table below shows a more specific and precise list of the applications and studies that are considered for future work

Identifier	Name	Description
FT1	Hierarchical Clustering in LinkedIn	The experiment consists on performing a hierarchical clustering algorithm to a collection of documents from LinkedIn in order to find keywords which appear together within the collection.
FT2	Studies on Companies	Due to our limited resources, the company collection could not be processed in order to reach new conclusions, so studies inside this collection could result in more interesting results
FT3	Candidate Matching Application	Create an application that given a set of requirements for an specific job offer returns the most appropriate candidates subscribed to the service
FT4	Relation of candidates to companies	Using the positions field from LinkedIn user profiles and the companies collection find the companies each user has been working in and define the skills and characteristics of candidates expected by each company.
FT5	Search new information sources	Search new information sources and design the crawler to retrieve more data and define the structure they will have inside the framework
FT6	Develop the prototype application	Finish the testing of the prototype application here showed and start its production as market product. This development includes: New functionalities, refine the current functionalities and revise design.
FT7	Application for candidate profile improvement	Study and develop an application capable to suggest the user with the best knowledge, professional work or course to be taken in order to improve its possibilities for job offer selection. Such application shall be based on the user professional profile and the skill information extracted from job offers and companies.
FT8	Development of a candidate guide for job seeking	From the studies performed during the project and adding new results, a tip guide for job selections could be written in order to provide help for candidates through selection processes.

Additionally, the experiments undertaken have proved that all the information retrieved and processed through the whole project provides the basis for a broad study on the field of human resources which will provide very valuable information to both employers with job hunting and candidates through selection processes.

In fact, most of studies have shown that there are some issues to be considered at present which, if resolved, would improve the process of job recruit and job finding. Probably, the most important

of them would be the results of the experiment described in section 4.2.4, as the inability to highlight the proper key terms could make any potential candidate seem inappropriate for a given job offer even though he has the required personal profile due to a failure of such candidate to highlight the correct skills.

Not only have the experiments served to detect such different points of view, but also to find preferences when selecting candidates such as the discovering on experiment from section 4.3.3 where we found that experience is more appreciated than the formation required or the one in section 4.3.2 where the key terms in profiles and technological knowledge fields give some pieces of advice when deciding which skills improve or what new technologies learn.

Moreover, we have been able to try the potential of clustering algorithms, which efficiently create subsets of documents with an astonishing relation among them. It is certainly an advanced technique which requires time and understanding but which outputs a satisfactory result. In this project, clustering has helped to separate job offers according to the technological profiles they require, which allows us to find professional profiles and estimate job parameters as well as applying the key terms of each cluster to other collections to study the outputs.

7. References

[1] node.js.2009. node.js. [ONLINE] Available at:http://nodejs.org. [Accessed 31 July 14].
[2] npm.2010. npm. [ONLINE] Available at:https://www.npmjs.org. [Accessed 1 August 14].
[3] LinkedIn.2002. LinkedIn Developer Network. [ONLINE] Available at: https://developer.linkedin.com. [Accessed 1 August 14].
[4] Linkedin-js. 2007. masylum/linkedin-js. [ONLINE] Available at:https://github.com/masylum/linkedin-js . [Accessed 14 August 14].
[5] Cran Project. 1999.The Comprehensive R Archive Network. [ONLINE] Available at:http://cran.r-project.org/. [Accessed 1 August 14]
[6] 2008. MongoDB. [ONLINE] Available at:http://www.mongodb.org. [Accessed 14 August 14].
[7] Express framework. 2010. Express-node.js web application framework. [ONLINE] Available at:http://expressjs.com. [Accessed 14 August 14].
[8] Apache Hadoop. 1995. Welcome to Apache Hadoop. [ONLINE] Available at:https://hadoop.apache.org. [Accessed 14 August 14].
[9] Cloudera. 2006. Cloudera. [ONLINE] Available at:http://www.cloudera.com. [Accessed 14 August 14].
[10] E. Dede, M. Govindaraju, D. Gunter, R. Canon, L. Ramakrishnan. 2013. Performance Evaluation of a MongoDB and Hadoop Platform for Scientific Data Analysis. [ONLINE] Available at:http://datasys.cs.iit.edu/events/ScienceCloud2013/p02.pdf. [Accessed 14 August 14].
[11] 2013. MongoDB MapReduce with Hadoop. [ONLINE] Available at:https://engineering.groupon.com/2013/big-data/mongodb-mapreduce-with-hadoop. [Accessed 14 August 14].
[12] 2007. mongo-hadoop connector. [ONLINE] Available at:https://github.com/mongodb/mongo-hadoop/blob/master/README.md. [Accessed 14 August 14].
[14] rud.is. 2012. Get an R Data Frame from a MongoDB query. [ONLINE] Available at:http://rud.is/b/2012/10/22/get-an-r-data-frame-from-a-mongodb-query/. [Accessed 19 August 14].
[15] Manning, C. D., 2009. *An Introduction to Information Retrieval*. 1st ed. Cambridge: Cambridge University Press.
[16] E. Dede, M. Govindaraju, D. Gunter, R. Canon, L. Ramakrishnan. 2013. Performance Evaluation of a MongoDB and Hadoop Platform for Scientific Data Analysis. [ONLINE] Available at:http://datasys.cs.iit.edu/events/ScienceCloud2013/p02.pdf. [Accessed 14 August 14].

[17] S. Ghemawat, H. Gobioff, S. Leung. 2003. *Google Research: The Google File System*. [ONLINE] Available at: http://research.google.com/archive/gfs.html. [Accessed 09 September 14].
[18] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh. 2006. *Google Research: BigTable*. [ONLINE] Available at:http://research.google.com/archive/bigtable.html. [Accessed 09 September 14].
[19] J. Dean, S. Ghemawat. 2004. *Google Research: MapReduce*. [ONLINE] Available at:http://research.google.com/archive/mapreduce.html. [Accessed 09 September 14].
[20] Lantz, B., 2013. *Machine Learning with R*. 1st ed. Liverly Place: Packt Publishing Ltd.
[21] Williams, G. 2014. *Data Science with R - Text Mining*. [ONLINE] Available at:http://handsondatascience.com/TextMiningO.pdf. [Accessed 18 September 14].

Annex I

Project planning

The expected duration for the project is of 12 months starting in October of 2013. In order to develop the project and achieve a correct use and management of time and resources, different work packages (WP) containing various tasks (T) each have been defined as follows:

- WP1. Study and research on Big Data Technologies and possible data sets:
    - T1.1: Study of the main Big Data concepts and Technologies
    - T1.2: Search of a data set and exploration of possibilities
    - T1.3: Preparation of a Crawling Strategy on the selected data set
- WP2. Crawling of Data:
    - T2.1: Design and prototype of a Crawler programme.
    - T2.2: Development of the main Crawler.
    - T2.3: Testing of the Web Crawler.
    - T2.4: Installation and preparation of storage systems.
    - T2.5: Data Crawling.
- WP3. Configuration and installation of the data processing framework:
    - T3.1: Selection of framework components.
    - T3.2: Installation of framework components.
    - T3.3: Testing of framework with independent data sets.
    - T.3.4: Testing of framework with project data sets.
- WP4. Data Processing:
    - T4.1: Data manual study to extract possible analyses
    - T4.2: Data processing inside framework
    - T4.3: Result Checking
- WP5. Application prototype development:
    - T5.1: Selection of technologies involved in the application
    - T5.2: Development of a connector for the project framework.
    - T5.3: Development of the application prototype.
    - T5.4: Testing of the application prototype.
- WP6. Report and documentation.
    - T6.1: Partial documentation of components during development.
    - T6.2: Report structure and milestones.
    - T6.3: Report writing.
    - T6.4: Report revision and corrections.
- WP7. Presentation preparation:
    - T7.1: Elaboration of presentation slides.
    - T7.2: Presentation preparation.

Additionally, we will define points when the project will reach a milestone to help the work plan and the project development control. Such milestones (M) are defined in the table below:

Identifier	Name	Description
M1	Project orientation	After the initial research, the team finds a proper orientation in the subject to work with.
M2	First Crawler	The first crawler version is functional and tested.
M3	Good data set	The information collected is big and good enough to start performing reliable studies.
M4	Application	An application for the results and data obtained has arisen and the team has approved it.
M5	Prototype developed	The prototype of the application is apt for demonstration.
M6	Report written	The report is written and delivered for corrections.
M7	Presentation	The presentation is prepared.

In order to have an illustration of this plan, we show the Gantt and network diagrams in the pages below:

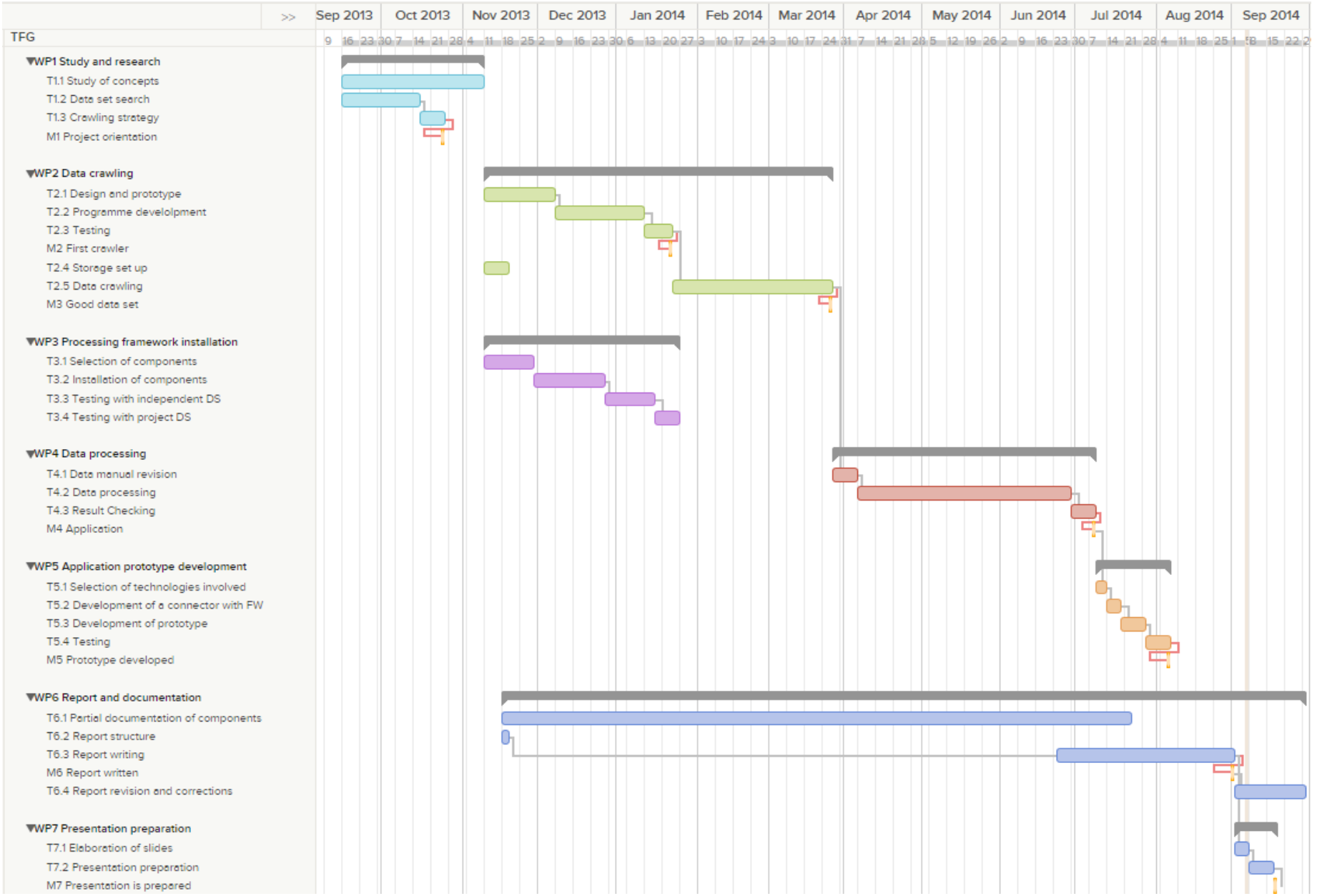


Figure AI.1 Gantt diagram

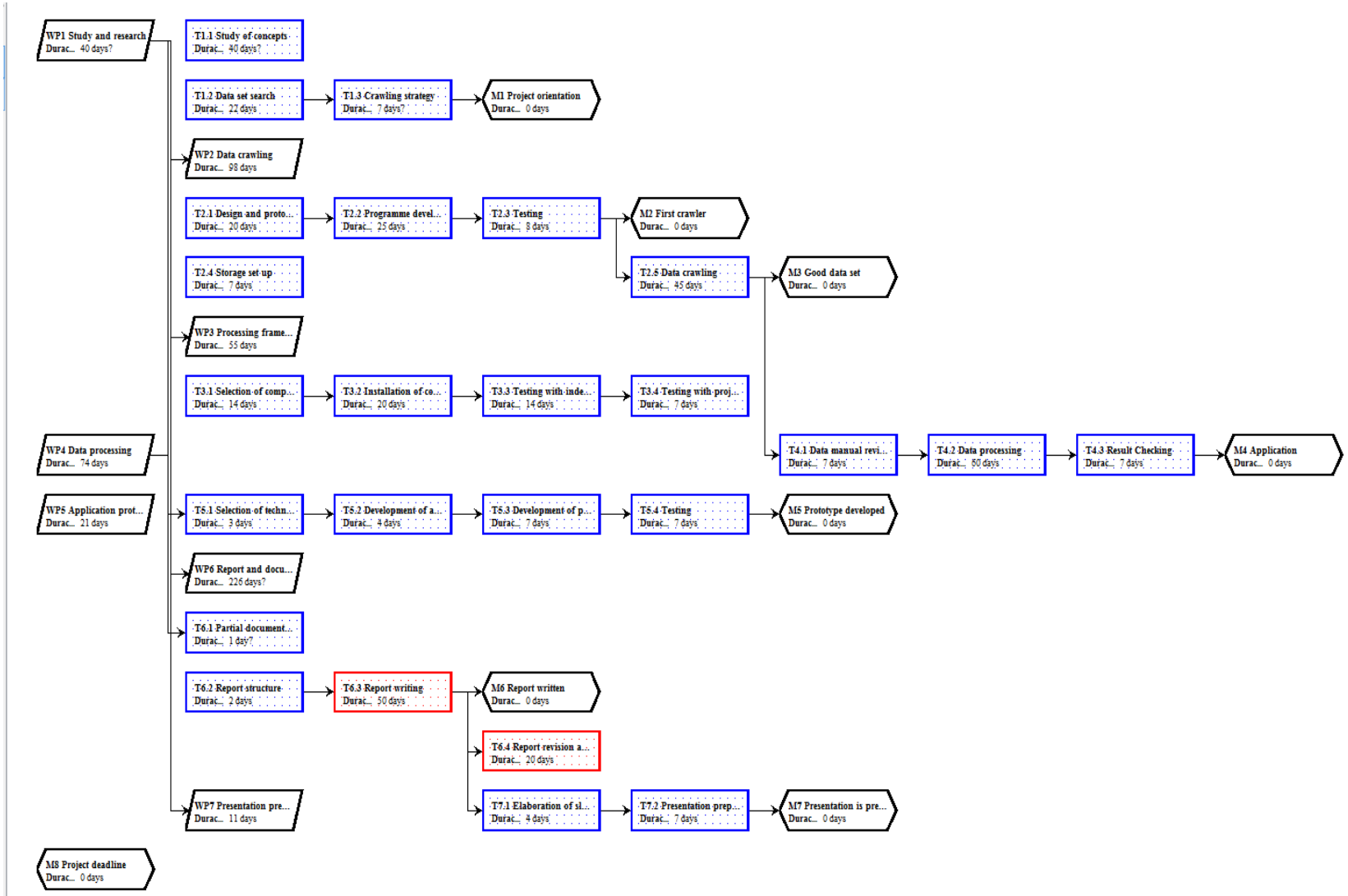


Figure AI.2 Network diagram

Budget

A research project like the one being described requires a very detailed and exhaustive budget in order to take the maximum possible efficiency out of the resources provided. Due to this, the budget in the project must contain all costs, including possible unexpected expenses, in order to achieve the best optimization possible.

The main cost in the project is human resources, as the working materials are computers and other goods that are very common in any ordinary company. Concretely, in this project, there are two team components: The Thesis tutor and the student. The student has devoted 4 hours per day during 5 ordinary week days for 12 months, which is a total of 960 hours in total. The tutor has not devoted as much time, since he has more functions in his job. Consequently, as an estimation, we will consider that he has devoted the 10% of the time the student has: 96 hours.

In order to estimate both salaries, we will consider Person Month measure (PM). PM is a measure of time where the employee is devoted to a job. Typically, the PM for any worker is stablished at 160 hours, although in the case of university professors it is fixed at 130 hours for research, as they must impart classes. Consequently, considering an income of 24000€/ year for the student and 48000€ for the professor, we can obtain the salary per PM by dividing brute salaries by the months worked. As a result we obtain 2000€/PM for the student and 4000€/PM for the professor. Then, we multiply the PMs devoted to the project of each and obtain a reward of 12000€ (6 PM) for the student and of 2400 € (0.6 PM) for the professor. The detailed cost report is reflected in the following table.

Other relevant expenses are all the computers used during the project. For all hardware used during the development a lifetime of 3 years has been considered and consequently the prize of such equipment is adjusted proportionally to the duration of the project. Software licenses should not be considered since the totality of the project has been undertaken using Open Source software.

Identifier	Name	Description	Total Cost
C1	Professor's Expenses	Salary and costs of maintenance of the tutor	2400€
C2	Student's Expenses	Salary and costs of maintenance of the student	1200€
C3	Computer	Computer used as server during the project. Cost of a year considering a lifetime of three years.	400€
C4	Student's Laptop	Laptop computer used by the student to access server and programming operations. The cost is considered for a year of use considering a lifetime of three years.	300€

C5	Professor's Laptop	Laptop computer used by the professor to supervise and contribute to the student's work. Cost of a year considering a lifetime of three years.	400€
Total Costs			**15400€**

Annex II

AII.1. MongoDB collections data schemas

Identifier	Source	Scheme defined

LIP	LinkedIn: Personal Profiles	personSchema
		• name
		• surname
		• headline
		• linkedinId
		• industry
		• location
		• specialties
		• profileURL,
		• positions
		  ○ company:
		  ▪ id
		  ▪ pointerToMongo
		  ○ current
		  ○ startDate
		  ○ positionTitle

LIC	LinkedIn: Companies	companySchema
		• blogRssUrl
		• description: String,
		• employeeCountRange
		  ○ code
		  ○ name
		• foundedYear
		• id
		• industries
		• locations
		• logoUrl
		• name
		• numFollowers
		• specialties
		• status
		  ○ code
		  ○ name
		• twitterID
		• universalName
		• websiteUrl

Identifier	Source	Scheme defined

LIJ	LinkedIn: Job Offers	jobSchema
		• active
		• company
		  ○ id
		  ○ pointerToMongo
		• description
		• descriptionSnipet: String,
		• expirationDate
		  ○ day
		  ○ month
		  ○ year
		• expirationTimestamp
		• jobID: String,
		• jobPoster
		• location
		• position:
		  ○ experienceLevel
		  ▪ code
		  ▪ name
		  ○ industries
		  ▪ values
		  ○ jobFunctions
		  ▪ values
		  ○ jobType
		  ▪ code
		  ▪ description
		  ○ locationName
		  ○ title
		• postingDate:
		  ○ day
		  ○ month
		  ○ year
		• postingTimestamp
		• siteJobURL
		• skillsAndExperience

TE	Tecnoempleo: Job Offers	• Company • State • Candidate Profile • Minimum requirements • Experience • Description • Professional Level • Contract Type

Identifier	Source	Scheme defined
TE	Tecnoempleo: Job Offers	• Dedication • Functions • Technologies Involved • Url • Salary • Incentives • Needed Stay • Info about other States • Freelance • Dependent People
IJ	Infojobs: Job Offers	• Categories • Number of Jobs offered • Description • Url • Job level • People organized • Department

AII.2. Additional code lines from crawler programs

```
var params={
        token:{
            oauth_token_secret: usersecret,
            oauth_token: usertoken
        }
    };
    console.log("La query es: " + query);
    linkedin.apiCall('GET', query, params, function(error, result){
        if (result!=null)
        {

            console.log(result);
            if(result.numResults<1)
                console.log("No hay resultados");
            else
            {
                if(query_type=="person")
                {
                    console.log("person")
                    print_people(result);
                }
                else if(query_type=="company")
                    print_companies(result);
                else if(query_type=="job")
                    print_jobs(result);
                else if(query_type=="group")
                    print_groups(result);
                else
                    console.log("There was an error with the received data");
            }

        }
        else
            {
                console.log(error);
                return null;
            }
```

Figure AII.1 Base crawler detail: LinkedIn crawling engine code snippet

```
db=mongoose.connection;
db.on('error', console.error.bind(console, 'connection error'));

db.once('open', function callback()
{
    console.log("Connected to "+ dbName);

    //We define here the person schema
    personSchema= mongoose.Schema(
    {
        name: String,
        surname: String,
        headline: String,
        linkedinId: String,
        industry: String,
        location: String, //just store the name
        specialties: String,
        profileURL: String,
        positions: [{company: {id: String, pointerToMongo:String}, current:Boolean, startDate: Date, positionTitle: String}],
    });
    //And we model it to create/open the collection


    person= mongoose.model(dbCollection, personSchema);
```

Figure AII.2 Mongoose person profile schema

```
jobSchema= mongoose.Schema(
{
    active: Boolean,
    company: {id: String, pointerToMongo:String},
    description: String,
    descriptionSnipet: String,
    expirationDate: {day: String, month:String, year:String},
    expirationTimestamp: String,
    jobID: String,
    jobPoster: String, //person ID
    location: String,
    position: {experienceLevel:{code:String, name: String}, industries: {values:[]},
        jobFunctions: {values:[]}, jobType:{code: String, description: String},
        locationName: String, title: String},
    postingDate: {day: String, month:String, year:String},
    postingTimestamp: String,
    siteJobURL: String,
    skillsAndExperience: String
```

Figure AII.3 Mongoose job offer schema

```
extract <- function(urlNews)
{
  script <- tryCatch(getURL(urlNews),HTTPError = function(e)
    {
      cat("HTTP error: ", e$message, "\n")
    })
  html <- htmlParse(script,encoding = "windows-1252")
  table<-readHTMLTable(html, encoding='windows-1252')
  names<-as.data.frame(table[[4]][1], stringsAsFactors=FALSE)
  values<-as.data.frame(table[[4]][2], stringsAsFactors=FALSE)
  info<-names
  info["values"]<-values
  info<-na.omit(info)
  values<-info$values
  data<-as.data.frame(values)
  data<-data.frame(t(data))
  colnames(data)<-(info$oferta)
  return(data)

}
```

Figure AII.4 Tecnoempleo Crawler. This code snippet shows the extraction of a given URL

AII.3. Framework additional captures

```
console.log("Connected to "+ DBname);
{
  if(err==null)
  {
    console.log("Something not null");
    eventEmitter.on("newResult", function(index)
    {
      if(docs[index].skillsAndExperience!=undefined)
      {
        console.log("something")
        console.log(docs[index].skillsAndExperience);
        var toExport=docs[index].skillsAndExperience.replace(/\n|\t/gi, " ");
          toExport=toExport.replace(/( [^rc] )/gi, "");
          toExport=toExport.replace("á", "a");
          toExport=toExport.replace("é", "e");
          toExport=toExport.replace("í", "i");
          toExport=toExport.replace("ó", "o");
          toExport=toExport.replace("ú", "u");
          toExport=toExport.replace("ñ", "nnnn");
          toExport=toExport.replace("(https?:\/\/[^\s]*)/gi", "");
          toExport=toExport.replace(/(<\/?.[a-zA-Z0-9\s|=|"|'|;]*>)/gi, " ");
          //toExport= toExport.replace(/[^a-zA-Z\s]/gi, " ");
          toExport=toExport.replace(/(\s+)/gi, " ");
          toExport=toExport.replace("nnnn", "ñ");
        buffer+=docs[index].jobID+"///"+toExport+"asdfghikl"
        index++;

          if(index>=docs.length)
            eventEmitter.emit("endProgram");
          else
          {
            if(index%5000==0)
              eventEmitter.emit("checkpoint", index);
            else
              eventEmitter.emit("newResult", index);
          }
      }
```

Figure AII.5 Mongo-framework connector based on Node.js solutions

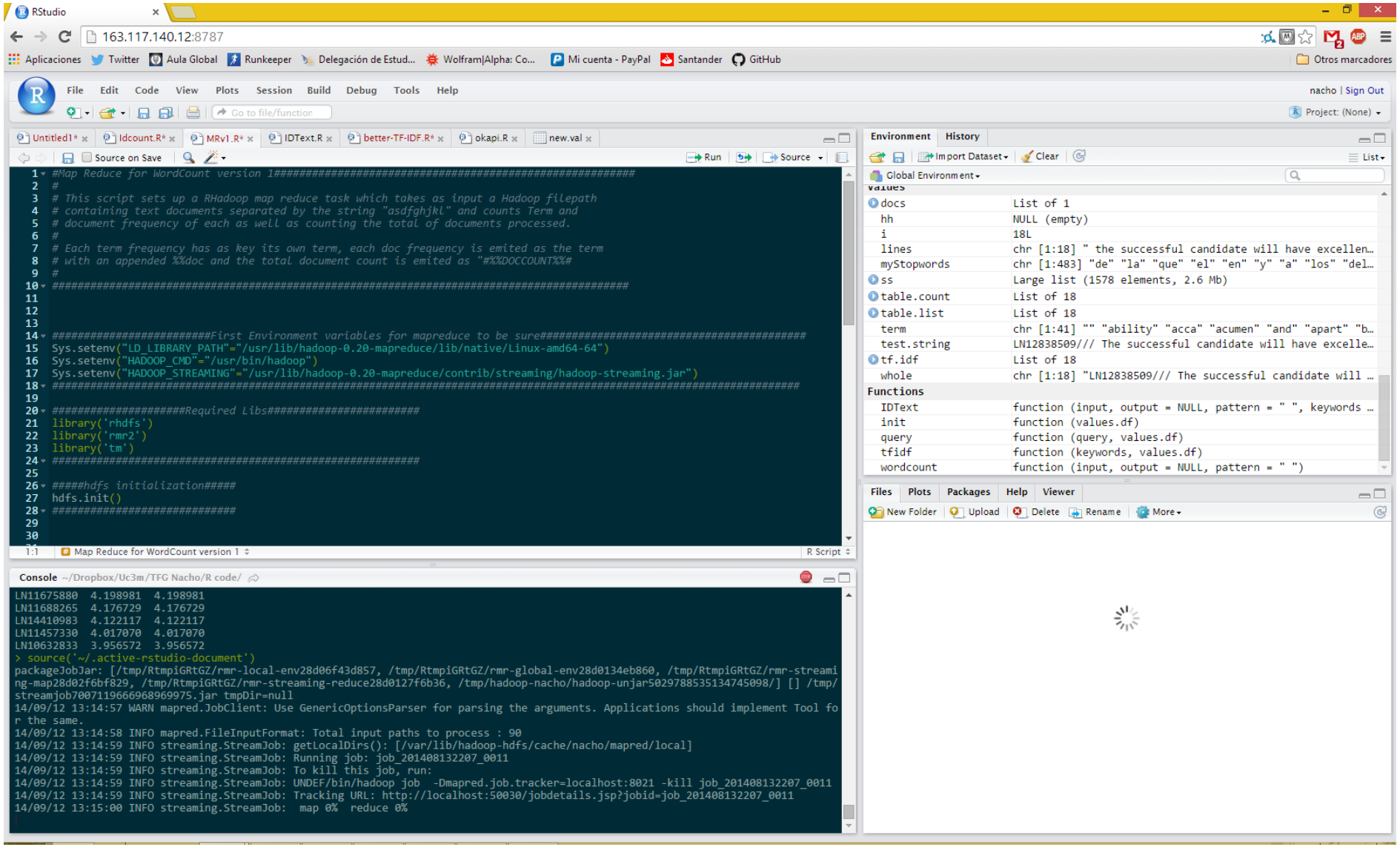


Figure AII.6 RStudio server capture during a Hadoop MapReduce job

AII.4. MongoDB Collections stats images

```
> db.persons_processeds.stats()
{
        "ns" : "CrawlerDB.persons_processeds",
        "count" : 67061,
        "size" : 13858460,
        "avgObjSize" : 206.65453840533246,
        "storageSize" : 16773120,
        "numExtents" : 6,
        "nindexes" : 1,
        "lastExtentSize" : 12582912,
        "paddingFactor" : 1,
        "flags" : 1,
        "totalIndexSize" : 2575440,
        "indexSizes" : {
                "_id_" : 2575440
        },
        "ok" : 1
}
```

Figure AII.7 LinkedIn person profile collection (LIP)

```
> db.jobs_processed2.stats()
{
        "ns" : "CrawlerDB.jobs_processed2",
        "count" : 170536,
        "size" : 626573612,
        "avgObjSize" : 3674.142773373364,
        "storageSize" : 696696832,
        "numExtents" : 20,
        "nindexes" : 1,
        "lastExtentSize" : 121233408,
        "paddingFactor" : 1,
        "flags" : 1,
        "totalIndexSize" : 7799904,
        "indexSizes" : {
                "_id_" : 7799904
        },
        "ok" : 1
}
```

Figure AII.8 LinkedIn job offer collection (LIJ)

```
> db.companies.stats()
{
        "ns" : "CrawlerDB.companies",
        "count" : 521824,
        "size" : 866284252,
        "avgObjSize" : 1660.1081054148526,
        "storageSize" : 920875008,
        "numExtents" : 20,
        "nindexes" : 1,
        "lastExtentSize" : 161656832,
        "paddingFactor" : 1,
        "flags" : 1,
        "totalIndexSize" : 16948848,
        "indexSizes" : {
                "_id_" : 16948848
        },
        "ok" : 1
}
>
```

Figure AII.9 LinkedIn company collection (LIC)

```
db.tecnoEmpleo.stats()
        "ns" : "CrawlerDB.tecnoEmpleo",
        "count" : 2596,
        "size" : 4562224,
        "avgObjSize" : 1757.4052388289676,
        "storageSize" : 9777152,
        "numExtents" : 5,
        "nindexes" : 1,
        "lastExtentSize" : 7340032,
        "paddingFactor" : 1,
        "flags" : 1,
        "totalIndexSize" : 98112,
        "indexSizes" : {
                "_id_" : 98112
        },
        "ok" : 1
```

Figure AII.10 Tecnoempleo job offer collection (TE)

```
> db.job_offer_dbs.stats()
{
        "ns" : "CrawlerDB.job_offer_dbs",
        "count" : 173128,
        "size" : 580275736,
        "avgObjSize" : 3351.7151240700523,
        "storageSize" : 641757184,
        "numExtents" : 20,
        "nindexes" : 1,
        "lastExtentSize" : 111681536,
        "paddingFactor" : 1,
        "flags" : 1,
        "totalIndexSize" : 5633264,
        "indexSizes" : {
                "_id_" : 5633264
        },
        "ok" : 1
}
```

Figure AII.11 Tecnoempleo and LinkedIn mixed job offer collection