Search engine for *Politics* Lovers

David Dias[*]
IST, Technical University of Lisbon
Av. Prof. Doutor Aníbal Cavaco Silva
2744-016 Porto Salvo, Portugal
david.dias@ist.utl.pt

Rui Francisco[†]
Institute for Clarity in Documentation
Av. Prof. Doutor Aníbal Cavaco Silva
2744-016 Porto Salvo, Portugal
rui.miguel.guerreiro.francisco@ist.utl.pt

ABSTRACT

This project was developed in the Data Mining Course of the Communication Networks Master in Engineering, the goal is to develop a robust and reliable search engine and data crawler for information related with politics and their participants. It was development mechanisms of evaluating popularity and relationships as well as a quick search engine for quality results

Keywords

TD-IDF, BM25, Search, Crawler, Politics, Personalities, Data-Mining, Big Data, Feeds, News, Sentiment, Natural Language Processing

1. INTRODUCTION

Super Search Engine is a Political news crawler, analyser and indexer that enables it's users to find with excellent performance and result quality news related with political personalities, know their popularity and understand how this personalities relate to each other. This document describes the implementation realised , the challenges, the algorithms, the design decisions and the results achieved by the authors of the system. In the following parts of the documents, the architecture will be described as well as the data collecting and searching mechanisms, how each individual component is built and finally the conclusions achieved by building a search engine about a category of news.

2. ARQUITECTURE

Super Search Engine Architecture is divided in modules loosely coupled, they are interchangeable as long they respect the *I/O* expected, the goal with this type of implementation is enabling enhancing each individual component without breaking the system in itself. There is a module for collection news, one for entity extraction, news search, sentiment analysis, do the search and one extra to measure the results. All data produced/collected is saved in collections of a *MongoDB* instance and indexed in *Whoosh*. In Figure 1 we can see the overview of the architecture, section 2.1 and 2.2 will explain the Mining and Searching Scheme respectively and each component will be introduced with proper reasoning in the algorithm choice

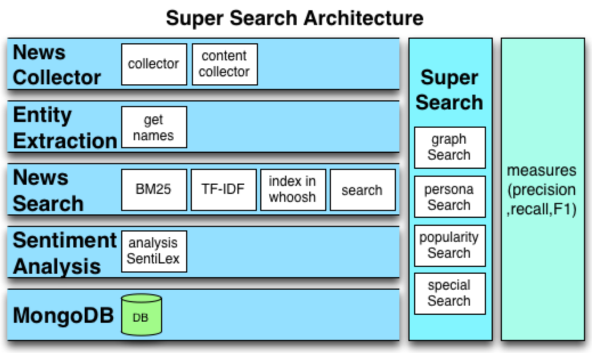

Figure 1: Super Search Architecture

2.1 Mining scheme

Our Data-Mining scheme is composed by 4 different operations done asynchronously to offer operation interchangeability, the sequence is done as follows:

1. Collect the news using a feedparser plus a content collector
2. Index the news in Whoosh by Title and Body so they are both searchable
3. Extract the entities found in each news if they are political personalities
4. Analyse the sentiment present in each news collected and attribute it to the personality present

When this process is finished, we've enough information prepared for our Searching Scheme, described in section 2.2. The Mining Scheme is described in Figure 2. We use *MongoDB* which is a *Document Oriented Database* for the data store for it's developer friendly functionality and fast data indexing, Super Search Engine takes advantage of *MongoDB*, for example in upserts (update, if not exists, insert), saving

[*]Master Student
[†]Master Student

time in news comparisons and verifications, it's scalability led us also to very quick deployment since it's schema less. We use *Whoosh* for indexing our data collection for easy sorting of documents base on *BM25* and *TF-IDF*, giving strong measurements of relevance for documents in function of a query.

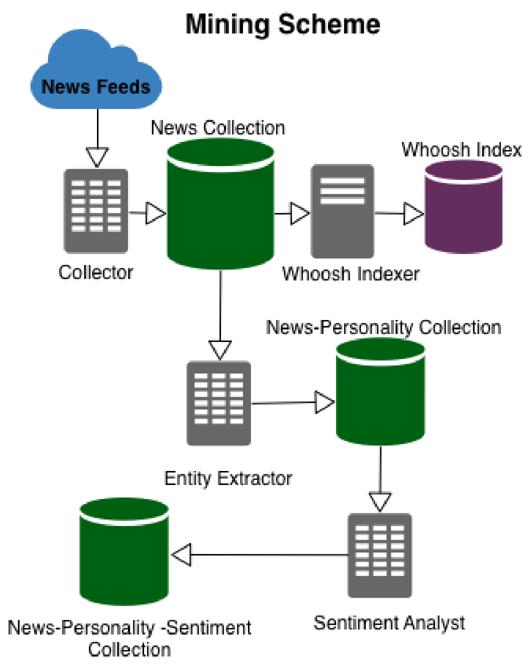

Figure 2: Data Mining Scheme of Super Search Engine

2.2 Searching scheme

Our Search Scheme was developed on top of all the data we collected, matching different signals to have best results in our search. We can see how it progresses in Figure 3.

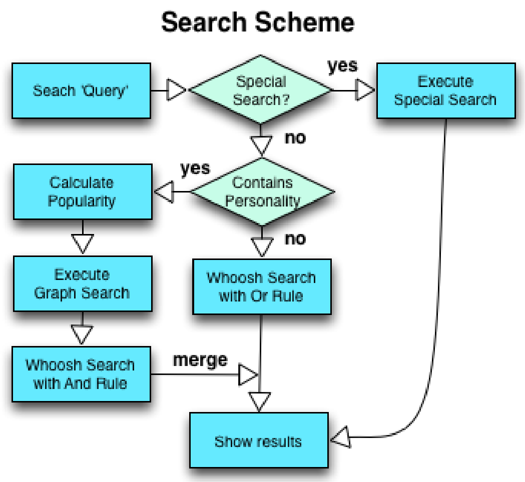

Figure 3: Seach Scheme of Super Search Engine

After receiving a query, the first question comes if it is a Special Search, which can be:

- "Who is more popular" - Returns which is the person most popular
- "Who is more hated" - Returns which person is less popular

If one of this options is selected, it is showed immediately the answer to the user.

The next step is seeing if the query introduced by the user contains a Personality, this is one of the key ingredients of super search, if in fact the query has a Personality we will execute the following operations

1. Calculate the popularity of the personality (or personalities in the query), regarding the data set available, giving details of how many news that person shows. By popularity we mean the accumulated sentiment from all the news that person appears.
2. Execute a "graph search", a graph search is calculating how strong is the relation between the searched personality and the others in the data set, the more times they appear together, the stronger is the relation
3. Search in whoosh index with "And Rule", what we mean by this is that if a personality name is "Miguel Francisco" we are going to look for news when that both names appear together and not alone, since Miguel and Francisco are both common names.

With this we are going to merge this results with a "Or Rule", so we can also find the news that relate to the rest of the query and find the best match possible, so we can create the best results.

2.3 Components

Instead of doing a massive, monolithic application, Super Search is divided into 5 individual interchangeable components, so it's easy to enhance the results by tuning each part individually, if I/O for each of them is respected.

2.3.1 *NewsCollection and Storage*

For collecting the news we use feedparser and Beautiful Soup Modules for python, this is a systematic and sequential process, after each feed is fetched(in "newsCollector/collector.py") with feedparser, we make a fetch of the Article ID of the post using Beautiful Soup(in "newsCollector/contentCollector.py"), so we can get the full news content, creating the information that will fuel our search more rich.

2.3.2 *News Search*

News Search is implemented in the folder "NewsSearch" but it's divided into 2 phases, the first one dedicated to indexing for the Data-Mining Scheme, implemented in the file "indexInWhoosh.py", the second phase is under "NewsSearch.py", "BM25" and "TFIDF", the last two give us a search on the Whoosh index using BM25 and TFIDF respectively, using equal weights for both algorithms. The "NewsSearch.py" makes an API available for Searching using "OR Rule" or "AND Rule" and mixing two searches to be used by Super Search

2.3.3 *Extraction of Personalities*

Extraction of Personalities is done on the "entityExtraction" Module, the purpose of this Module is only to identify which of the names are present in each news and then compare it with our list of political names to see if it's a match, to make this happen we use Natural Language Processing (made available by the nltk module) to find which names are words in first hand. The process follows the algorithm:

1. Analyse next news in news collection at the MongoDB instance
2. Tokenize with each sentence with "nltk.sent_tokenize"
3. Classify each word with "nltk.pos_tag"
4. Creates chunks using the chunker function for name entity nltk.ne_chunck
5. Verify each PERSON node to see if match to our list of political personalities, if yes, then we consider that news has some comment on that personallity and insert it on the NewsPersonality Collection on the MongoDB instance

2.3.4 *Sentiment Analysis*

A classe que implementa esta funcionalidade sentimentAnalisys / sentimentFlex.py. A função sentiLexFlexToDict começa por obter toda a informação que se encontra no ficheiro SentiLex-flex-PT02.txt. E em seguida construir uma estrutura de dados mais fácil de aceder e com apenas a informação pretendida. A função polarity recebe a estrutura de dados e a notícia que vai ser analisada. Esta função divide a notícia por frases. Depois em cada uma das frases é verificada a existência de expressões e palavras que se encontram na estrutura de dados recebida. Caso exista executa se uma operação de soma ou subtração de acordo com o valor da polaridade. Assim consegue se saber se uma certa notícia é positiva ou negativa.

2.3.5 *Super Search*

Esta funcionalidade trata três tipos de pesquisa: personalidades pesquisa de conteúdos e qual a personalidade mais ou menos popular Quando se realize uma pesquisa de uma personalidade o resultado esperado é o nome da personalidade, as personalidades que se relacionam com a personalidade pesquisada e sua popularidade, a popularidade da personalidade pesquisada e as noticias onde a personalidade e referida. A função que trata a descoberta de personalidades relacionadas é graphSearch.A função primeiro verifica se a personalidade introduzida para pesquisa se encontra no ficheiro personalities.txt. Caso exista vai a base de dados procurar as noticias onde a personalidade e referida e obtém as outras personalidades que também foram referidas nessa noticia. A função que indica a popularidade e personalityPopularity. A função primeiro verifica se a personalidade introduzida para pesquisa se encontra no ficheiro personalities.txt. Caso exista vai a base de dados procurar as noticias onde a personalidade e referida e verifica de que forma a personalidade é referida na notícia de forma positiva ou de forma negativa. No fim de percorrer todas as notícias alcançasse a popularidade da personalidade pesquisada.

Quando se realiza uma pesquisa por qual é a personalidade mais popular ou menos popular executa se ou a função whoIsMorePopular ou WhoIsMoreHated. Esta duas funções são executadas dependo da query que é introduzida caso seja introduzida "who is more popular" é executada a função whoIsMorePopular, caso seja introduzida "who is more hated" é executada a outra função. Estas duas funções recebem uma estrutura de dados que contem todas as personalidades e sua popularidade. Esta estrutura é criada pela função orderedByPopularity que obtém os nomes das personalidades e sua popularidade a partir da base de dados introduzindo as numa estrutura de dados que em seguida será ordenada usando a função sorted do python.

3. RESULT ANALYSIS

3.1 Experimental Results

3.2 Measures of the results

4. FUTURE WORK

[1].

5. CONCLUSIONS

6. REFERENCES

7. REFERENCES

[1] P. C. Mário J. Silva and L. Sarmento, "Building a sentiment lexicon for social judgement mining," *International Conference on Computational Processing of the Portuguese Language (PROPOR)*, pp. 218–228.