ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Bachelor Project report

Cyriaque Gilles Guillaume Brousse

Opinion extraction in French texts

Laboratoire d'Intelligence Artificielle
Supervision by Dr. Claudiu-Cristian Musat and Prof. Boi Faltings

Lausanne, June 2015

Table of contents

1 Introduction

This report relates the work carried out during the Spring 2015 semester at the Artificial Intelligence lab of EPFL (LIA), continuing the previous research conducted there.

1.1 Goal of the project

Ninety percent of all the data available in the world has been generated during the last two years [1]. More and more textual data is thus available publicly on the Internet. People have also changed their habits of news consumption: they are more and more mobile, and tend to read their newspapers on the go, in an electronic format. The goal of the project conducted since several years at the LIA is to produce a structured summary of opinions about controversial subjects found on the Internet. We download articles, classify them by subject and analyze them to extract opinions. The information is passed to a visualization component, through which users can view highlighted opinions in the articles. Each opinion is defined to be positive or negative with respect to a topic. We can then present users with a list of articles sorted, for example, from the most positive to the most negative.

1.2 Existing infrastructure

Most of the work has been completed by LIA for English texts. The task was to adapt the existing back-end structure (program logic) for French texts. At first, it would seem that a mere refactoring would fulfill the requirement; however, the English module uses libraries that are not available for French, due to the very different structure of the two languages. Therefore, this project consisted of several phases of research and implementation.

2 Model

The implementation we provide is based on the following concepts and definitions.

2.1 Natural language processing concepts

A **word** is the smallest unit of natural language we can use. More formally, it is a triplet $\langle value, sentence\ id, POS\ tag\rangle$, where the string value is its textual representation, the sentence identifier is its position in the sentence, and the part-of-speech tag is its grammatical nature – for example, “verb”, “noun” or “adjective”.

A **sentence** is a series of words, structured with punctuation, that has a meaning.

Our version of a news **article** is simply a collection of sentences.

From a given article, we can extract sentences and then words, assigning to each entity its characteristics. This process will be described later on. Once we have obtained separate words, we can define some abstractions.

A **grammatical dependency**, or simply **dependency**, is a relation between two words. It is more formally a triplet $\langle relation, governor, dependent \rangle$. The governor and the dependent are the two words of the dependency, linked by a relation. An actual example of dependency we could extract in "*a beautiful house"* is `mod(house,beautiful)`, where "beautiful" is the dependent word to "house". The nature of this dependency is "modifier", since "house" is a noun, and "beautiful" is an adjective.

A **chain** of two dependencies, is an entity such that $\{reln_1(gov_1, dep_1) :: reln_2(dep_1, dep_2)\}$, where gov_1, dep_1 and dep_2 are all mutually distinct. As described,

- each dependent of the first dependency must be the governor of the second, and
- no duplicate words are allowed: if we consider that a sentence is a collection of nodes, then a chain may not go through a node more than once.

Note that the concept of a chain is not limited to two dependencies only. It is easily extensible to an arbitrary number of dependencies.

2.2 Entities

An **object** is defined in [2] as an entity that can be a product, person, event, organization, etc. It is associated with a pair $obj: (T, A)$, where T is a hierarchy of components, sub-components, and so on. A is the set of attributes of *obj*.

Such a recursive definition is more complex than necessary for our purpose. We will only consider an object and its **aspects**.

A **topic** in our model is a collection of **keys**, namely words related to the same object. A topic exists in an article or a collection of articles. We can determine the **proportion** of each article's topic, relative to each other. For instance, an article about agriculture might be composed of a topic with main key "tractors", composing 28% of the said article.

Finally, an **opinion** about an object is a positive or negative view, attitude, emotion, etc. on the object from an opinion holder.

The **opinion polarity**, or **orientation**, may be positive, negative or neutral.

We might want to extract opinions at the whole article level, declaring it positive or negative. This would however make little sense, since we are interested in extracting opinions expressed about a specific subject, not an article. Several subjects might also be dealt with in the same article. Finally, we want to pinpoint a fine-grain location of each opinion in an article. Therefore, the process that we implemented focuses on extracting opinions from specific *topics* of a subject.

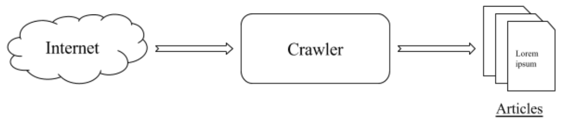
We will now present the implementation of the system, along with the challenges we faced.

3 Implementation

3.1 Presentation of the functional blocks

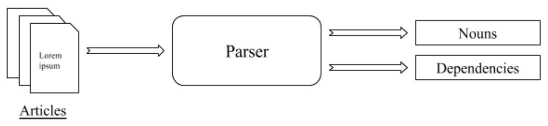
A general overview of the implemented system can be found in appendix B. In the following paragraphs, we will go over each functional block.

3.1.1 Crawler



The crawler downloads articles on a specific subject from the Internet. A subject might be “economy”, “health”, “video games”, etc. The crawler is not part of the news recommendation system back-end, but is worth mentioning since it provides us with a collection of raw articles and their content. It was the subject of a parallel project at LIA.

3.1.2 Parser

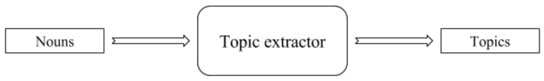


The parser takes an article as input and produces typed dependencies, based on a grammatical and statistical analysis of the sentences. The English module previously developed uses the Stanford parser, described in [3]. However, the Stanford parser for French [4] is able to produce a parse tree for French texts, but cannot output typed dependencies. We thus had to use another parser: the Malt parser, described in [5]. This parser takes a parsed sentence as input, which is defined as words along with their **POS tags**. The set of possible tags is to be found in appendix B. The Malt parser effectively outputs typed dependencies and is based on a model that was pre-trained on the French Treebank (abbreviated as “FTB” in the literature). The FTB was produced by the CNRS; it is a collection of 1 million words from *Le Monde* newspaper.

We need to somehow obtain the words’ tags from a certain analysis. The process is called **tagging** or **shallow parsing**. For this purpose, we use the Stanford tagger, described in [6].

The parser as a functional block also produces a list of nouns contained in the article, which is needed in later phases of the extraction pipeline. This list is simply obtained by filtering the pairs $\langle word_i, tag_i \rangle$ by tag, to keep only the nouns.

3.1.3 Topic extractor



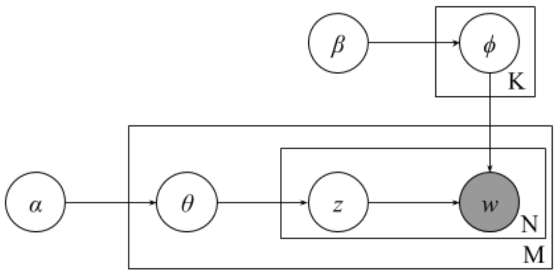
The topic extractor provides a list of topics, as defined in section 2.2. It bases the analysis on the nouns extracted by the parser. It conducts a probabilistic analysis called **latent Dirichlet allocation**, or **LDA**, presented in [7]. This processes is based on the following parameters:

- α is the per-document topic distribution
- β is the per-topic word distribution
- θ_d is the topic distribution for document d
- ϕ_k is the word distribution for topic k
- z_{dj} is the topic for the jth word in document d
- w_{dj} is a word.

The constants are:

- K, the number of topics to extract
- M, the number of documents
- N, the number of words per document.

The LDA process is presented in the next figure.

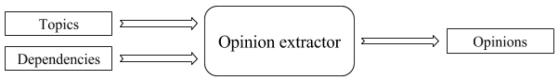


Each article is then considered as the probability distribution of its topics. Each topic is in turn viewed as the probability distribution of its nouns.

In this model, we want to estimate parameters θ and ϕ. To this extent, we use the Mallet tool, developed at the University of Massachusetts [8].

It is relevant to note that we only want to use nouns for this analysis, not any subset of the article's words, or worse, all the words. Indeed, the LDA model is unaware of the grammatical differences between two words. It does not make sense to consider verbs or adverbs in the analysis; what would for example "go" or "when" – that are not specific to any context – mean as topic keys? Nouns on their own are a good indicator of what is discussed in the article, so we will restrict the analysis to nouns, which will used as keywords in the opinion extraction phase per se.

3.1.4 Opinion extractor



The opinion extractor, which is the cornerstone of the project, takes the topics and dependencies that were obtained in the previous stages. It outputs the opinions found from these elements.

In this stage, we use two external components: a **stemmer** and a **sentiment dictionary**.

Stemming, or desuffixation, is the process by which we transform any given word into its **stem**, or "root" form. Examples of such transformations for French are {"continu", "continua", "continuait", "continuité", …} $\overset{stemm}{\Longrightarrow}$ {"continu"}.

Note that, depending on the algorithm, words of different nature can be mapped to the same root. In the previous example, "continu" is an adjective, "continua" is a verbal form, and "continuité" is a noun; yet the three forms are all transformed into the same stem.

Also note that the resulting stem might not be a valid word. For example, "maintenant" can be mapped to "mainten", which does not exist in French.

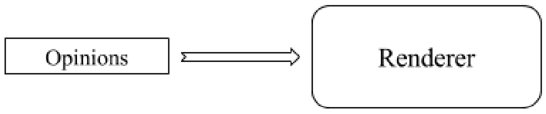
One could argue that it is best to use a **lemmatizer** instead of a stemmer. Lemmatization is the process of reducing any word to its **lemma**, or canonical form. An example for English is {"am", "are", "is", "being"} $\overset{lemm}{\Longrightarrow}$ {"be"}.

The lemmatizer does not produce invalid words, and does not change the nature of words (as discussed before), so it presents a theoretical advantage over the stemmer. However, the process is very slow, and we need to have a full dictionary for the language beforehand. For this reason, we have used a stemmer. Because of the algorithm we use, even a stemmer gives satisfactory results.

We use an external library based on the Porter stemming algorithm. This library is called "Snowball stemmer" and is available and described at [9].

The dictionary is a collection of entries defined as follows: $\langle word_i, polarity_i \rangle$, where each polarity is in the set {positive, negative, neutral}. Our dictionary is a text file containing 3981 entries.

3.1.5 Rendering component



The renderer takes the previously extracted opinions as input, and displays them in some predefined way.

The renderer is, like the crawler, not a part of the back-end structure of the balanced news recommendation system.

LIA has developed an Android application and a website, in which one can view an article where the extracted opinions are highlighted.

3.2 Parsing & Dependency extraction pipeline

The Malt parser trained on the FTB model we discussed before takes a specific format as input, namely the **CoNLL-2007** format (Conference for Natural Language Learning). This format is specified in [10].

The Stanford POS tagger is not able to produce an output in this format. It outputs a list of tokens in the form $\langle word_i, tag_i \rangle$. We thus had to write an adapter to transform these tokens into CoNLL format. Each fine-grain POS tag described in appendix A has to be present in a coarse-grain form.

We can launch the Malt parser once the input file has been written to disk. From the resulting tokens, we can reconstruct the typed dependencies needed for the opinion extraction. The transformation operates as in the following example:

```
29 quelques _ DET DET _
30 heures   _ NC  NC  _
```

⇓

```
29 quelques _ D DET _ 30 det _ _
30 heures   _ N NC  _ 26 obj _ _
```

⇓

```
det(heures-N-30,quelques-D-29)
```

3.3 Topic extraction

Recall the model presented in section 3.1.3.

The script we use to launch the Mallet tool specifies the following constants:

- 16 topics are to be extracted ($K = 16$)
- 16 keys per topic
- M and N are computed from the list of articles

We also use an optimization called “interval optimization” of 20 passes. This leads to more accurate topics.

3.4 Opinion extraction pipeline & algorithms

3.4.1 A word on chains

It is important to come back on the concept of a **chain** of dependencies. As described in section 2.1, a chain of two dependencies can be constructed when we have two dependencies that match the pattern $reln_1(a, b) :: reln_2(b, c)$. In this section, we will consider that a chain has length of exactly two dependencies.

For a given set of dependencies D and specific dependency, called the starting dependency, noted $first$, it is conceptually easy to construct the set of chains C. We use the following algorithm for this purpose:

```
procedure allChainsFromFirst(first: Dependency, D: Set[Dependency]) {
  if (first ∈ $D$) {
      $D = D \setminus$ first
  }

  candidates: Set[Dependency]

  D.filter{second ⇒
         first.dep = second.gov
      && first ≠ second}
   .foreach{$d_i$ ⇒ (candidates =
            candidates ∪ $d_i$)}

  $C$: Set[Chain]
  candidates.foreach{second ⇒
   $C = C$ ∪ (new Chain(first,second))}

  return $C$
}
```

3.4.2 *Stemming algorithm*

When performing a dictionary lookup, we want to know what the polarity of a word is, if it exists. However, we might have a developed form of the word (i.e. a conjugated verb) that is not present in such a form in the dictionary.

We will then use the process of **stemming** described in section 3.1.4: we stem the word to look up. We ask the dictionary for a polarity for the stemmed form. If it exists, we return it. Otherwise, we return the polarity of the unstemmed word, if any.

3.4.3 *Principle of opinion extraction*

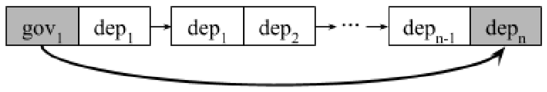
We will now present the algorithm developed for extracting opinions. The general idea was kept from previous work conducted at LIA.

The general idea is the following. Given a parsed article and a collection of topics:

- For a parsed article, split the extraction by parsed sentence
- For a parsed sentence, process each typed dependency for every topic in the following way:
  - First determine the word that is related to the topic by iterating over the topic keys. This word is called the **topic word**.
  - To obtain an opinion, the other word of the dependency must be polar. We stem the word and do a dictionary lookup in the fashion described earlier. If a polarity is returned, we have found that the word is the **polar word**. An opinion was found, but we need to carefully determine the **global polarity** for the opinion. This determination is described in section 3.4.4. Now we have everything we need to add the found opinion to the set of extracted opinions for the current article.
  - Try to find chains that start with this dependency. For each found

chain, if any, analyze the chain. If the analysis returned an opinion, add it to the set of extracted opinions.

One may wonder how to analyze a chain. It turns out that a chain can be viewed as a single dependency, where the governor of the acting dependency is the governor of the first dependency in the chain. Similarly, the dependent word of the acting dependency is the dependent of the last word in the chain. We illustrate this concept in the following schema:



This way of proceeding takes on its full meaning, since the concept of chain allows us to take into consideration words gov_1 that present an indirect modifier dep_n.

3.4.4 *Concept of global polarity*

By default, the polarity of an opinion is the polarity of the polar word (i.e. the word whose lookup in the sentiment dictionary yielded a polarity).

This way of assigning the global polarity for the opinion might not work when the topic word (i.e. the word that is a topic key) is also polar. Let us consider the example of "cancer rapide". A dependency could be `mod(cancer,rapide)`. The polarity word in this case is "rapide", and it is positive. We have a problem since the topic word, "cancer", possesses a negative polarity. Therefore, we would like this opinion to be of negative polarity. A simple model for determining the global polarity would be to compute the sign product of the two polarities. The following computation would ensue:

$$\begin{aligned} &polarity(\text{cancer, rapide}) \\ &= polarity(\text{cancer}) \times polarity(\text{rapide}) \\ &= negative \times positive \\ &= negative \end{aligned}$$

We do not want, however, dependencies such as `mod(cancer,mauvais)` to be regarded as positive, even if the polarity of both words is negative. The rule is then to invert a positive global polarity to a negative one only if the polarity of the two words is different.

Another discrepancy in the global polarity arises when we are faced with a dependency like `obj(éviter,accident)`. Both "éviter" and "accident" are negative; however, the global polarity should be positive. We notice in this example that one of the two words is a verb. This is a special case of the rule we formulated before.

All these directions are taken into consideration in the final algorithm for determining the global polarity of a dependency (or chain):

```
procedure determineGlobalPolarity
(polarWord: Word, polarWordPolarity:
Polarity, topicWord: Word) {
  topicWordPolarity =
    dictionary.lookup(topicWord)

  if (∃topicWordPolarity &&
    topicWordPolarity ≠ neutral) {

    if (polarWordPolarity = negative
      && polarWord.isVerb) {
      /* polar word is a verb, its
         polarity is negative */
      return (topicWordPolarity =
       negative) ? positive : negative
    } else if (topicWordPolarity =
      negative && topicWord.isVerb) {
      /* topic word is a verb, its
         polarity is negative */
      return (polarWordPolarity =
       negative) ? positive : negative
    }

    if (polarWordPolarity ≠
        topicWordPolarity
      && polarWordPolarity ≠ neutral
      && topicWordPolarity ≠ neutral)
    {
      /* regular +/- case */
      return negative
    }
  } else { /* default case */
    return polarWordPolarity
  }
}
```

We have now presented all the relevant algorithms and implementation challenges. In the next part, we will show the results we obtained with our implementation of the project initial requirement.

4 Results

4.1 Corpus

The corpus we used is a subset of 150 articles from a corpus of video games reviews. Each article contains on average 1076 words. The median of words per article is 1023. The word count range is [636, 1937].

4.2 Evaluation method

We assign a **score** to each opinion as follows:

- –1 for an opinion of negative polarity,
- 0 for a neutral one, and
- 1 for a positive one.

Using these scores, we sum all the opinion scores over a given article to get the article **global score**. This enables us to tell its polarity as a whole.

We assess each extracted opinion as either negative, neutral or positive without knowledge of the actual polarity. We then compare that expected value with the actual result computed by our implementation.

4.3 Number and quality of the extracted opinions

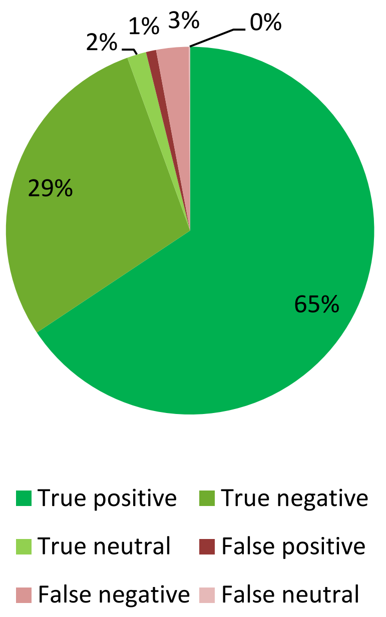
On our corpus, we were able to extract on average **7.37 opinions per article** (average over 11 runs of the full program). The range of opinions per article the system was able to extract over these 11 runs was [6.75, 8.11].

We now present the quality of the opinions, classified by error type:

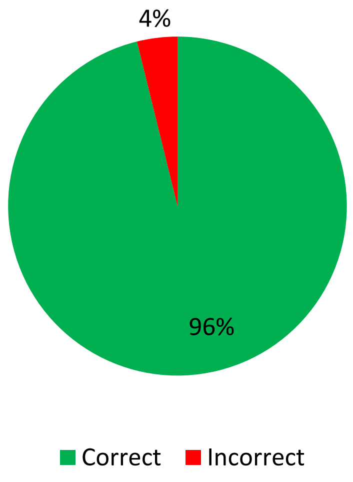
· True positives:	665
· True negatives:	292

· True neutrals:	17
· False positives (type I error):	9
· False negatives (type II error):	29
· False neutrals:	1

Over a total of 1013 opinions, 974 were classified correctly, and 39 were classified in the wrong category. The result is presented in the chart below.



The next graph compares the proportion of correctly classified opinions against the proportion of wrongly classified ones.



5 Conclusion

The implementation required vast refactoring and modularization of the back-end logic, as well as development of new algorithms. We were able to provide a functional system for French, that gives good results with respect to the pre-established requirements and model.

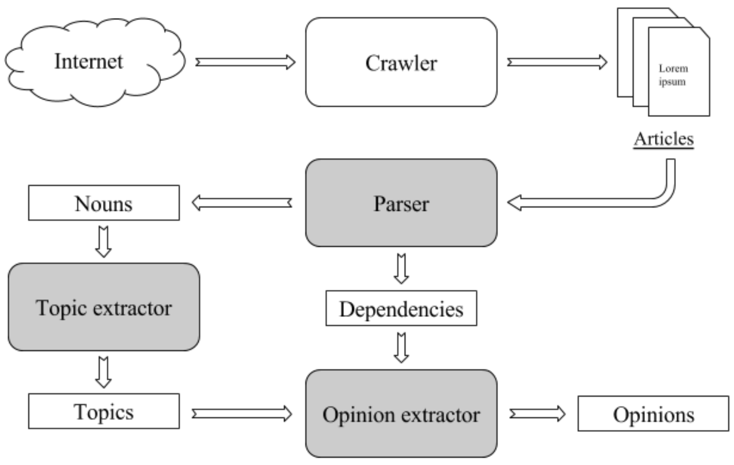
A next step could be to consider the type of the dependencies that exist between words when extracting opinions. This was done in a previous project with a different parser. In our implementation, we use the Malt parser on a pre-trained model that does not output very varied types of dependencies (i.e. most of the time “mod”, “subj” or “obj”). The final results are still satisfying, since all extracted opinions were constructed through one or two dictionary lookups. The dictionary acts as a filter and ensures that opinions are indeed polar.

Appendices

A Part of speech tags used by the Malt parser trained on the FTB

Fine tag	Coarse tag	Category
ADJ	A	Adjective
ADJWH	A	Interrogative adjective
ADV	ADV	Adverb
ADVWH	ADV	Interrogative adverb
DET	D	Determiner
DETWH	D	Interrogative determiner
NC	N	Common nouns
NPP	N	Proper noun
CC	C	Coordination conjunction
CS	C	Subordination conjunction
V	V	Indicative or conditional verb
VINF	V	Infinitive verb
VPR	V	Present participle
VPP	V	Past participle
VIMP	V	Imperative form
VS	V	Subjunctive form
PRO	PRO	Pronoun
PROREL	PRO	Relative pronoun
CLS	CL	Subject-clitic pronoun
CLO	CL	Object-clitic pronoun
CLR	CL	Reflexive-clitic pronoun
P	P	Preposition
P+*	P	Preposition + * (amalgam)
PUNC	PUNC	Punctuation
I	I	Interjection
ET	ET	Foreign word

B General overview of the back-end system



The blocks we worked on in this project are highlighted in gray.

Bibliography

[1] «Big Data, for better or worse: 90% of world's data generated over last two years,» *Science Daily,* 22 May 2013.

[2] L. Bing, «Sentiment analysis and Subjectivity,» *Handbook of Natural Language Processing, Second edition,* 2010.

[3] D. Klein et C. D. Manning, «Fast Exact Inference with a Factored Model for Natural Language Parsing,» *Advances in Neural Information Processing Systems 15 (NIPS 2002),* pp. 3-10, 2003.

[4] S. Green, M.-C. de Marneffe, J. Bauer et C. D. Manning, «Multiword Expression Identification with Tree Substitution Grammars: A Parsing tour de force with French,» *EMNLP 2011*, 2010.

[5] J. Nivre, «An Efficient Algorithm for Projective Dependency Parsing,» *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03),* pp. 149-160, April 2003.

[6] K. Toutanova, D. Klein, C. D. Manning et Y. Singer, «Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network,» *Proceedings of HLT-NAACL 2003,* pp. 252-259, 2003.

[7] D. M. Blei, A. Y. Ng et M. I. Jordan, «Latent Dirichlet Allocation,» *Journal of Machine Learning Research,* vol. 3, pp. 993-1022, 2003.

[8] A. K. McCallum, «MALLET: A Machine Learning For Language Toolkit,» 2002. [En ligne]. Available: http://mallet.cs.umass.edu.

[9] M. Porter, «Snowball: suffix stripper grammar (Stemmer),» 2002. [En ligne]. Available: http://snowball.tartarus.org/algorithms/french/stemmer.html.

[10] J. Nivre, «The CoNLL 2007 Shared Task on Dependency Parsing,» 2007.