Università degli Studi di Salerno

Dipartimento di Scienze Politiche, Sociali e della Comunicazione

Dottorato in Scienze della Comunicazione, Sociologia, Teorie e Storia delle Istituzioni, Ricerca Educativa, Corporeità didattiche, Tecnologie e Inclusione

XV Ciclo

A Hybrid Framework for Text Analysis

Alessandro Maisto

Supervisor and Tutor:
Prof. Annibale Elia

A.A. 2016/2017

The machine is only a tool after all, which can help humanity progress faster by taking some of the burdens of calculations and interpretations off its back. The task of the human brain remains what it has always been; that of discovering new data to be analyzed, and of devising new concepts to be tested.

I. Asimov

Contents

List of Tables

List of Figures

Chapter 1
Introduction

In Computational Linguistics there is an essential dichotomy between Linguists and Computer Scientists. The first ones, with a strong knowledge of language structures, have not engineering skills. The second ones, contrariwise, expert in computer and mathematics skills, do not assign values to basic mechanisms and structures of language. Moreover, this discrepancy, especially in the last decades, has increased due to the growth of computational resources and to the gradual computerization of the world; the use of Machine Learning technologies in Artificial Intelligence problems solving, which allows for example the machines to "learn", starting from manually generated examples, has been more and more often used in Computational Linguistics in order to overcome the obstacle represented by language structures and its formal representation.
The dichotomy has resulted in the birth of two main approaches to Computational Linguistics that respectively prefers:

- rule-based methods, that try to imitate the way in which man uses and understands the language, reproducing syntactic structures on which the understanding process is based on, building lexical resources as electronic dictionaries, taxonomies or ontologies;
- statistic-based methods that, conversely, treat language as a group of elements, quantifying words in a mathematical way and trying to extract information without identifying syntactic structures or, in some algorithms, trying to confer to the machine the ability to learn these structures.

One of the main problems is the lack of communication between these two different approaches, due to substantial differences characterizing them: on the one hand there is a strong focus on how language works and on language characteristics, there is a tendency to analytical and manual work. From other hand, engineering perspective finds in language an obstacle, and recognizes in the algorithms the fastest way to overcome this problem.
However, the lack of communication is not an incompatibility: as it did at the beginning, i.e. with Harris, the best way to treat natural language, could result from the union between the two approaches.

At the moment, there is a large number of open-source tools that perform text analysis and Natural Language Processing. A great part of these tools are based on statistical model and consist on separated modules which could be combined in order to create a pipeline for the processing of the text. Many of these resources consist in code packages which have not a GUI (Graphical User Interface) and they result impossible to use for users without programming skills. Furthermore, the vast majority of these open-source tools support only English language and, when Italian language is included, the performances of the tools decrease significantly. On the other hand, open source tools for Italian language are very few.

In this work we want to fill this gap by present a new hybrid framework for the analysis of Italian texts. It must not be intended as a commercial tool, but the purpose for which it was built is to help linguists and other scholars to perform rapid text analysis and to produce linguistic data. The framework, that performs both statistical and rule-based analysis, is called *LG-Starship*. The framework has been built in modules and, each module performs a specific task of Natural Language Processing or text analysis. In addition, a graphical interface will include, in a first time, a tab for each module. The included modules are the followings:

- a module for charging text, which performs some basic text analysis such as tokens and letters counts and offers some preprocessing operation as text cleaning and normalization or StopWords removal.
- a module for POS Tag and Lemmatization, which includes the counts of lemmas and grammatical categories.
- a module for statistical analysis, with the computation of Term Frequency and Tf-idf by texts units, that presents line charts for selected words.
- a Semantic module that calculates semantic similarity for text units or disambiguates ambiguous words.
- a Syntactic module that analyzes syntax structures of sentences tagging verbs and its arguments with semantic labels.

The framework can be considered "hybrid" in a double sense: as explained in the previous lines, it uses both statistical and rule/based methods, by relying on standard statistical algorithms or techniques, and, at the same time, on Lexicon-Grammar syntactic theory. In addition, it has been written in both Java and Python programming languages. LG-Starship Framework has a simple Graphic User Interface but will be also released as separated modules which may be included in any NLP pipelines independently.
There are many resources of this kind, but the large majority works for English. There are very few free resources for Italian language and this work tries to cover this need by proposing a tool which can be used both by linguists or other scientist interested in language and text analysis who have no idea about programming languages, as by computer scientists, who

can use free modules in their own code or in combination with different NLP algorithms.

In our opinion, the name of the framework could effectively represent some of the main features of the project: in the Starship metaphor, suggested by the name of one of the standard algorithms we used in the project, the *Hyperspace Analogue to Language* distributional semantics algorithm, textual data can be considered as a huge galaxy or universe which is constantly growing. Moving through this universe is not possible with standard tools as existing text editors or search engines, but requires a special machine which takes into account the nature of the environment and can proceed through misleading elements and dangerous interpretation as a starship moving through an asteroid field. In addition, the continuous growth of this universe, makes necessary some tools which allow to jump from a specified point to any other point of the text universe as a spaceship that performs an hyperspace jump to reach remote stars.
As happens in science-fiction novels, there are many different kind of starships, each one using a different kind of technology; in LG-Starship, the technology on which the engine is based is the Lexicon-Grammar Theory (LG) which will be discussed in next chapters.

Chapter 2

Related Works

2.1 NLP Tools

There is a large number of open-source tools for Natural Language Processing. Some of these tools consist of a mere computational structure that must be enriched with any kind of linguistic resource such as dictionaries, grammars or algorithms. Other tools present an applicable pipeline of statistical, syntactic and semantic algorithms that allow different text analysis approaches. In this session some of most used free tools for Natural Language Processing are presented.

One of the most important NLP tool is *Nooj* [Silberztein, 2005, 2014]. Nooj is an empty linguistic framework based on the Lexicon-Grammar Theory developed by Maurice Gross in the '70 and make a great use of Finite-State Automata. Nooj is the evolution of an obsolete tool called INTEX [Silberztein, 1998] with which it shares many components. Nooj allows user to process large sets of texts in short time. Its architecture is based on .NET technology.
As we said, Nooj framework is not provided with linguistic resources by default, but there is a large international community of Nooj users that developed in last years dictionaries and grammars for more than 20 languages including French, English, Italian, Chinese, Arabic, Spanish etc.. Nooj includes three main components:

- a lexical component that allows the creation of dictionaries and inflexion grammars;
- a morphology component that allows the development of derivational and morphological grammars;
- a syntactic component for the construction of syntactic grammars for parsing or automatic tag of texts

In addition, Nooj can perform some kind of statistical analysis as frequency calculation, tf/idf, and similarity between terms. For what concerns the Italian module for Nooj, it was realized by [Vietri, 2014] of the University of Salerno. The free Italian module includes:

- a dictionary of simple words (more then 129.000 entries);
- a dictionary of compound nouns (about 127.000 entries);
- a dictionary of first and last names (about 2500 entries);
- a dictionary of toponyms that includes city and country names;
- inflexional grammars associated with dictionaries entries;

- morphological grammars for derived forms of proper nouns, pronominal forms recognition and derivation recognition;
- a syntactic grammar for the extraction of date/time patterns

Nooj application and resources are free available at `www.nooj-association.org`. Nooj is a user-friendly linguistic application that offers excellent computational performances and very good results. The Nooj language is easy to learn; dictionaries could be easily created, expanded or modified and grammars could be generated graphically or by rule editor. By contrast, Nooj presents two major drawbacks:

1. Is not possible to include Nooj into another code because there are not APIs and it is not possible to apply Nooj to dynamic corpora. A command-line application exists but it works only if Nooj main application is running into the PC.
2. Nooj POS tagging follows a precise philosophy and does not disambiguate ambiguous terms: i.e. the italian determiner *la* is recognized as a Determiner, but also as a Noun because in the dictionary *la* also exists as a musical note. In this way, syntactic grammars easily reaches a recall of 100%, but the precision must be refined with the addiction of linguistic constrains that could cause slower performances.

Another framework for NLP is GATE, General Architecture for Text Engineering [Cunningham, 2002, Cunningham et al., 2002]. GATE constitutes an infrastructural system for research and development of language processing softwares and has been implemented at the University of Sheffield. The tool provides only an infrastructure for linguistic studies.
GATE provides a Collection of REusable Object for Language Engineering (CREOLE) that could be used together with a document manager through a graphical interface. By selecting a number of CREOLE objects it is possible to define a pipeline module for text analysis and produce output as annotations or structured texts. The main components of CREOLE are:

- **tokenizer**, that splits text into simple tokens, by differentiating between words, numbers, punctuation or symbols;
- **sentence splitter**, that uses finite state transducers to segment the text into sentences;
- **tagger** is a modified version of Brill tagger and produces Part Of Speech tag on each token;

- **gazetteer**, that consists of a list of cities, organizations and other kind of proper nouns;
- **semantic tagger**, that consists of rules written in Java Annotation Pattern Engine language;
- **orthomatcher** is an optional module that have as primary objective to perform co-reference between entities and to assign annotations to unclassified names.
- **coreferencer**, that finds identity relations between entities in the text.

Once a module has been create and run, it is possible to show the produced data by selecting an available result viewer. Default viewer displays annotations on a typical tabular form, but it's possible to select different viewers as a tree viewer. It is also possible to compare sets of annotations using the Compare Annotation Tool (CAT).

The Natural Language Toolkit, better known as NLTK [Bird, 2006] is a suite of program modules, data sets and tutorials that supports research in computational linguistics and NLP. Written in Python, NLTK includes the vast majority of NLP algorithms, providing implementation for tokenizers, stemmers, taggers, chunkers, parsers, clusterers and classifiers. It also includes Corpus samples as Brown Corpus or CoNLL-2000 Chunking Corpus.

NLTK is very useful for English language and achieves good performances on English texts analysis, but it is almost impossible to apply it to other languages as Italian. Moreover, the installation of modules is not easy and the use of Corpus collections that is not included into the standard packages is a hard task.

Based on NLTK there are many python libraries. One of the most used open-source libraries is *TextBlob* [Loria, 2014], that provides API for common NLP tasks. Main features of TextBlob are:

- Noun phrase extraction
- Part-of-speech tagging
- Sentiment analysis
- Classification based on Naive Bayes classifiers or Decision Trees
- Language translation and detection using Google Translate APIs
- Tokenization

- Word and phrase frequencies
- Parsing
- N-grams
- Word inflection and lemmatization
- Spelling correction
- WordNet integration

In order to analyze a text with a TextBlob module it is needed to convert the text format into a TextBlob String, a proper format for texts. Once realized the conversion of one or more texts, it is possible to apply the other modules. For what concerns modules that could be considered Language-dependents, the package presents resources only for English Language, but in many cases, with correct lexical resources, modules include training functions that could be used in order to extend the usage of the module to other languages. Furthermore, "statistical" modules are not dependent on language and can be applied on multilingual corpora.

As the authors declare, and on how it is based on NLTK, TextBlob uses *Pattern* repositories. As NLTK, Pattern [Smedt and Daelemans, 2012] is a package for Python, and includes functionalities for web mining, NLP, machine learning and network analysis. Pattern is thought as a tool for both scientific and non-scientific audience. It is organized in separate modules:

- *pattern.web* that includes tools for web mining;
- *pattern.en* that includes regular expression-based parser for English that uses a finite-state POS tagger, a tokenizer, a lemmatizer and a chuncker, and is bundled with WordNet
- *pattern.search* is a N-gram pattern matching algorithm that uses an approach similar to regular expressions. Can be used to search a mixture of words, phrases, POS tags, control characters or taxonomy terms.
- *pattern.vector* that represents Documents or Corpus as lemmatized bag-of-words in order to calculate TF/IDF values, distance metrics or dimension reduction. The module includes clustering algorithms, Naive Bayes and SVM classifier.
- *pattern.it* includes the same features of the *pattern.en* module, but implement new functions as Gender prediction, Article specification, Verbs conjugation. In Italian module is not included the Sentiment Function.

Since NLTK and Pattern were written in Python, there is another family of libraries, developed in Java, used to perform NLP tasks. One of these libraries is *OpenNLP* [OpenNLP, Wilcock, 2009]. OpenNLP library is a machine learning based toolkit for natural language processing that supports some of most common NLP tasks as tokenization, sentence segmentation, POS tagging, named entity extraction, chuncking and parsing. A large manual and a complete documentation can be found on the library web page[1]. OpenNLP includes three kind of tokenizer, classified on the base of their complexity. A whitespace tokenizer that splits sequences by white spaces, a simple tokenizer that detects tokens by character class and learnable tokenizer that uses a maximum entropy probability model to calculate tokens. Named entity extraction also works in two different ways: the package includes a statistical Name Finder, but also offers a dictionary-based name finder implementation. OpenNLP module for POS Tagging is an implementation of Perceptron Tagger for English language, but the package includes a training module that can be used in order to apply the algorithm to other languages. Parsing is offered in two different implementations: a chunking parser and a treeinsert parser (not recommended by authors). The chuncking one is a statistical parser and must be trained with a training dataset that respects the Penn Treebank format [Marcus et al., 1994].

Stanford CoreNLP [Manning et al., 2014] is a set of Natural Language analysis tools designed to be flexible and extensible. It presents a set of modules that can be included in a text analysis pipeline. The modules includes:

- a POS tagger module that is an implementation of log-linear POS tagger [Toutanova and Manning, 2000];
- a Named entity recognizer (NER), that includes, in particular, three classes of nouns (Persons, Organizations and Locations) and is based on CRFClassifier [Finkel et al., 2005];
- a probabilistic parser.
- a sentiment analysis algorithm based on Recursive Neural Network built on grammatical structures. The module is also based on a Sentiment Treebank, a database of over 215.000 phrases automatically annotated.

Stanford CoreNLP is developed for English, Chinese, French, German and Spanish.

For Italian Language [Pianta et al., 2008] proposes a suite of tools for NLP called TextPro. The single tools that compose TextPro are offered as standalone programs, but they can be used in an integrated environment, providing

[1] `http://opennlp.apache.org/`

an extensible framework to create and add new components. The main characteristics of TextPro are simplicity (easy to install and configure), modularity and portability. TextPro consists of nine main components each one represents a classic task of NLP such as text cleaning, tokenization, morphological analysis, POS tagging, chunking, Named Entity Recognition, lemmatization and Multiword recognition.

2.2 Tokenization

Digital Texts consists in a continuous string of machine-readable codified symbols. In this continuous characters, numbers, graphical symbols, punctuation and whitespace are all represented by codes. Normally, machines do not have the ability to distinguish between them and can not isolate words, numbers, etc.

Tokenization is the task that consists in isolate word-like units from texts detecting white spaces or differences between character types. These word-like units, called “tokens”, represent the full inflected or derived form of the word and can be also called “wordforms” [Martin and Jurafsky, 2000].

Tokenization is not a trivial task [Grefenstette and Tapanainen, 1994]: to note that tokenization is more than a simple split by white spaces, we need to think to the same concept of “word”. In Graffi and Scalise [2002], the notion of word is analyzed in depth. They notice that what is word in a language, could not be a word in other languages; it is necessary to distinguish phonological words from morphological and syntactical words. A word could be a string of characters included between two white spaces, but if we take into account a sequence as the sentence in example 1 we can object to this simplistic definition.

(1) *This is a doctoral thesis, not a paper.*

- if we consider the first word, “This”, it is included between the start of the sentence and a white space. Nevertheless it must be considered a word.
- if we respect the definition of word given above, “thesis,” must be considered a word, but the symbol “,” is a punctuation and must be considered as a different “token”. In Italian language, in the majority of cases, it’s possible to previously add a whitespace between a word and a punctuation symbol, but in words like “po’ ”, a contraction of “poco”, *a bit*, or

"dell' ", a form for the prepositional article "dello/a", *of the*, the apostrophe should remain attached to the word because it contributes to the sense of the word.

- the word "doctoral thesis" could be considered a single unit because the sense of the single words is different from the sense of their sum. In fact, "doctoral thesis" is considered a compound word and must be recognized as a single token.

Traditionally, tokenization task uses hand-crafted rules with regular expressions or finite state automata [Frunza, 2008] as in Bird [2006], Di Cristo [1996], Pianta et al. [2008]. As in Grefenstette and Tapanainen [1994], regular expressions are necessary in order to recognize some special structures: ambiguous separators in numbers, for example, can be recognized by a regular expression that includes dots and commas for numbers as 12.000 or $0,234$, slash and backslash for dates or other symbols (percent symbol or monetary symbols). For what concern abbreviation, it is necessary to consult a lexicon. In the same way, in Maisto and Pelosi [2014c] an electronic dictionary of Compound words has been used to recognize multiword expressions in a corpus of reviews.
A second kind of approaches for tokenization is based on sequence labeling algorithms such as Hidden Markov Models. Grana et al. [2002] presents a POS tag algorithm that includes tokenization in order to decide if one or more words form or not the same term, and assigns the appropriate number of tags. The model uses the Viterbi algorithm [Forney, 1973] in order to evaluate streams of tokens of different lengths over the same structure. In Frunza [2008], tokenization is performed without any human expertise or dictionaries, but the only source of information is a pre-tokenized corpus on which the algorithm learns the rules for the correct tokenization.

2.3 PosTag and Lemmatization

Part-of-speech (POS) tagging is one of the most important and used preprocessing step in Natural Language Processing task, it is considered the necessary baseline for every further linguistic analysis. It consists in attributing to each wordform its grammatical category and disambiguates that terms which could be considered ambiguous in the selected language. In Italian, a Part-of-Speech tag of the sentence 2 is shown in table 2.1:

(2) *la porta è aperta.*
the door is open.

Words	**Tags**
la	DET
porta	N
è	V
aperta	A

Table 2.1: Example of Part-Of-Speech Tag

A POS tagger algorithm must be able to tag correctly each word and to decide what is the correct tag in case of ambiguity. In the example above, the word *porta* is used with the meaning of “door”, but in Italian it could corresponds to the present second person of the verb *portare*, “to bring”. Furthermore it is necessary to establish a set of tags, of varying complexity, that must be coherent with the selected idiom. In the example the tags only express the grammatical category of terms, but it is possible to set deeper tags that can express, for example the if the determiner is defined or undefined, etc.

Although there are several available resources for English language, the number of tools currently available for the Italian language is drastically reduced. The situation is even more reduced when we consider only tools freely available and open source: only TreeTagger, Pattern and OpenNLP are free. Since ‘70s automatic tag of Part-Of-Speech was considered an important application for every future researches in computational linguistics.
Greene and Rubin [1971] propose a rule-based approach in the TAGGIT program and reach the 77% of precision in disambiguated the Brown Corpus [Francis and Kucera, 1982].

Church [1988] presents a program that uses a linear time dynamic programming algorithm to find an assignment of POS to words based on lexical and contextual probability with an estimated precision of 95-99%. The stochastic method proposed, first calculate the Lexical Probability by dividing the frequency of a word for the frequency of the same word with a specific POS tag. The stochastic method proposed in the paper makes use of lexical disambiguation rules. Then, the contextual probability is calculated as

> *The probability of observing part of speech X given the following two parts of speech* Y *and* Z*, is estimated by dividing the trigram frequency* XYZ *by the bigram frequency* YZ.

the probability of observe a Verb before an article and a noun is estimated by the frequency of the sequence Verb, Adjective and Noun divided by the frequency of the sequence Adjective, Noun).

In 1992, Cutting et al. [1992] present a POS tagger based on Hidden Markov Model (HMM). He define 5 feature that a Part-Of-Speech tagger must has. A POS Tagger must be:

- **Robust** because it must work with text corpora that contains ungrammatical constructions, isolated phrases, unknown words or non-linguistic data.
- **Efficient** because it must tag the largest number of words in the shortest possible time.
- **Accurate** because it should assign the correct POS tag to every word encountered.
- **Tunable** as meaning that it should take advantage of linguistic insights.
- **Reusable** as the effort required to retarget the tagger to new corpora, tagsets or languages should be minimal.

Authors use HMM because this kind of models permits a complete flexibility in the choice of training corpora, tagsets or languages, reducing time and complexity. This model reach the 96% of precision on Brown Corpus.

An example of Rule-based POS tagger was presented by Brill [1992]. In this method, the algorithm automatically acquires the rules and reach an accuracy comparable to stochastic taggers. Once the algorithm assigns to each word the most likely tag, estimated by examining a large tagged corpus without regards to context, it improves its performances using two procedures:

a. unknown words capitalized are considered as Proper Nouns;

b. a procedure attempts to assign the tag most common for words ending in the same three letters.

In a second step, the tagger acquires patches to improve performances. Patch templates are based on context, on lexical properties and on distribution region.

The importance of context in POS tagging is also underlined by Ratnaparkhi et al. [1996], that use in parallel, a probability and some contextual features. For what concern non-rare words (a word which occurs more than 5 times in the corpus), they are tagged by using the simple probability model. Rare and unknown words, on the contrary, are tagged as words with similar prefixes or suffixes. Tested on the Wall St. Journal corpus, the model obtains a total accuracy of about 95%.
Other statistical model could be found in Brants [2000], Toutanova and

Manning [2000], Giménez and Marquez [2004], Denis et al. [2009].

Currently, Pos Tagging is essentially considered a "solved task", with state-of-the-art taggers achieving precision of 97%-98%[Toutanova et al., 2003, Shen et al., 2007].
Best algorithms are considered the Stanford Tagger (version 2.0), that use the maximum entropy cyclic dependency network [Manning, 2011]; BI-LSTM-CRF [Huang et al., 2015], and NLP4J [Choi, 2016] that use Dynamic Feature Induction. In Collins [2002] was presented a new perceptron algorithm for training tagging models as an alternative to maximum entropy models. The Averaged Perceptron Tagger uses a discriminative, feature-rich model. Features are modeled using *feature functions*: $\phi(h_i, t_i)$ when h_i represents the history and t_i the tag. History h_i is a complex object modeling different aspects of the sequence being tagged. It contains previously assigned tags and other contextual features such as the form of current word.

For a sequence of words w of length n in a model with d feature functions, the scoring function is defined as:

$$score(w,t) = \textstyle\sum_{i=1}^{n} \sum_{s=1}^{d} \alpha_s \phi_s(h_i, t_i)$$

with α_s as feature weights paired with a feature function ϕ_s. The Viterbi algorithm is used to compute the highest scoring sequence of tags.
Votrubec [2006] proposes a new implementation of Averaged Perceptron Tagger highlighting some behaviors as the aversion of the algorithm to the excess of information or to complex features, and establishing that the maximum accuracy usually came between the 4th and the 8th interaction. In this implementation, a set of morphological features are added to the standard perceptron tagger implementation in order to reflect the complexity of Czech and, in general, Slavic languages. Based on Votrubec [2006], Hajič et al. [2009] propose a different set of features for English and Czech language due to the typological difference between the two idioms.

For what concern Lemmatization task, it consists in take a token and convert it into its common base form: the lemma. In English, base form for a noun is the singular (e.g. *mouse* rather than *mice*) and the noninflected form of the verb. In Italian language, singular is used as canonical form of nouns, adjectives (masculine if the name is masculine or masculine/feminine, and feminine for feminine nouns), pronouns and determiner, and infinitive for verbs. With respect to the sentence 1, the lemmatization results are shown in table 2.2:

In some cases Lemmatization is replaced with another language modeling technique called *Stemming*. Stemming is a procedure to reduce all words with

Words	**Tags**	**Lemmas**
la	DET	il/lo
porta	N	porta
è	V	essere
aperta	A	aperto

Table 2.2: Example of Lemmatization

the same stem to a common form. An analysis of the precision of a search algorithm launched on English texts processed with the two algorithm [Balakrishnan and Lloyd-Yemoh, 2014], shows that, though the Lemmatization obtains best values of precision, the difference between these two techniques is insignificant.

In Porter [1980] a stemming algorithm was presented. The algorithm removes suffixes from a word form until the "stem" was found. The use of a stemmer instead of a lemmatizer is desirable because it reduce the size and the complexity of the data in the text. Actually, both approaches can be combined as Ingason et al. [2008] do for Islandic language, applying the lemmatization first and, subsequently, using a stemming algorithm on lemmatized text.

In Plisson et al. [2004], two word lemmatization algorithms were compared: the first one is based on *if-them* rules, the second one is based on ripple down rules induction algorithms. Ripple Down Rules [Srinivasan et al., 1991] have been developed for knowledge acquisition and maintenance of rule-based systems. The algorithm create exception to existing rules when a rule produce incorrect conclusions. Most algorithms use external lexical information, as *Morfette* [Chrupała et al., 2008], or *SEM* [Constant et al., 2011] for French texts.
In particular, Morfette uses a probabilistic system in order to joint morphological tagging and lemmatization from morphologically annotated corpora. Morfette system is composed by two learning modules, one of that for lemmatization. A decoding module searches for the best sequence of pairs of morphological tags and lemmas for an input sequence of wordforms. The class assigned to a wordform - lemma pair is the corresponding shortest edit script (SES) between the two reversed strings. SES represents the shortest sequence of instructions which transforms a string w into a string w'. The precision of Morfette on the lemmatization task is the 93% for Romanian, 96% for Spanish and 88% on Polish language.

Kanis and Müller [2005] use a training Dictionary in order to search the longest common substring of the full form and the lemma. Using the dictionary, the system generates lemmatization rules deriving the derivation rules that lead to the formation of flexed forms.

For what concern Italian resources, we can cite an Italian version of Tree-Tagger [Schmid, 1995], an Italian model for OpenNLP [Morton et al., 2005], TagPro [Pianta and Zanoli, 2007], CORISTagger [Favretti et al., 2002], Tanl POS tagger [Attardi et al., 2009], ensemble-based taggers [Dell?Orletta, 2009] and Pattern tagger [Smedt and Daelemans, 2012].

2.4 Statistic analysis of Texts

Statistical text analysis is based on counting words and extract information from text by analyzing the results of the counting process.
Consider the following example:

(3) *il cane mangia il gatto che mangiò il topo.*
the dog eat the cat that ate the rat

How many words compose the sentence 3? As we saw in 2.2, in the sentence 3 we can count 9 wordforms or tokens. We saw that exist some basic operation we can do with a list of tokens like the one presented above, as the tag of Part-Of-Speech elements or the reduction to the citation forms (Lemmatization). But if we need to know how many different words compose the sentence, we must count the “types”. Pierce (1931-58, vol. 4, p. 423) introduces the distinction between tokens and types by specifying that in a sentence as the one above, *there is only one word type ‘the”* but there are three tokens of it in the sentence. The relation between the Type and its tokens is an *instatiation* and can be quantified in terms of token occurrence.

The first and simplest operation we can perform on a text is the extraction of types and the calculation of the number of instances that appear in a text. For example:

Type	Occurrences
il	3
cane	1
mangia	1
gatto	1
che	1
mangiò	1
topo	1

Table 2.3: Types and Occurrences

in table 2.3 the first column represents the types encountered in the sentence and, the second column represents the number of tokens for each type. Obviously, articles and preposition, always will stay on the top of occurrence ranks. For this reason, in some cases the text is pre-processed in order to delete this kind of elements, also known as "stop-words" [Tokunaga and Makoto, 1994].

Simple occurrence ranks can be useful for analysis of larger texts. In the next table we present the most frequent words of the chapter 9 of the first book of the *Lord of Rings* saga [Tolkien, 1991], and the first chapter of *The hobbit* [Tolkien, 2012]. Stop-words has been removed from the list.

Type	Occurrences
said	51
frodo	42
night	34
aragorn	31
river	29
boats	28
out	26
great	25
dark	25
Sam	24

Table 2.4: Types and Occurrences extracted from *Lord of Rings, The fellowship of the ring*, Chapter 9

Type	**Occurrences**
said	76
bilbo	37
like	37
out	35
thorin	34
gandalf	33
dwarves	29
door	27
baggins	25
little	24

Table 2.5: Types and Occurrences extracted from *The Hobbit*, Chapter 1

Reading the lists in table 2.4 and in table 2.5 we can learn that the first text speaks about “Frodo”, it could be set on “night”, near a “river” with “boats”. In the same way, the second text has as its characters “Bilbo”, “Gandalf”, “Thorin” and speaks about “dwarves” and a “door”. However, the second text (the one extracted from “The hobbit”) is larger and counts 9828 tokens, while the first one count only 8243 tokens. This difference, reflected on occurrence ranking, could not reflect the real impact of a wordform on a text. In this case, the two texts have similar dimensions and then the occurrences of a type in both texts is often similar. But if we try to compare texts of different dimensions, the occurrence ranking may misled.

In order to show this difference, we need to take into account a new text, the Chapter 7 of the second book of the *Lord of Rings* saga. The chapter contains 4946 tokens (about half of the previous two) and the occurrence ranking is shown in table 2.6.

The table 2.6 shows that in the selected text, the wordform “frodo” appear 30 times, less then the same wordform in the first text as reported by tab. 2.4. It is clear that we need to take into account the proportion between the two texts and calculate a “weight” for each word in each text.

Term Frequency is the simplest measure to weight each term in a text. A term has an importance proportional to the number of times it occurs in a text [Luhn, 1957]:

Type	**Frequency**
said	34
frodo	30
gollum	28
up	28
down	21
sam	20
now	19
must	18
out	18
under	17

Table 2.6: Types and frequencies extracted from *Lord of Rings, The two towers*, Chapter 7

$$TF = \frac{n_{i,j}}{|d_j|}$$ [Baeza-Yates et al., 1999]

Applying this formula it is possible to compare text with different dimensions. Considering as "text 1" the Chapter 9 of "The fellowship of the ring", "text 2" the chapter 7 of "The two towers", table 2.7 compares word occurrences and frequencies of three different types:

Type	Text 1 occurrences	Text 2 occurrences	Text 1 frequency	Text 2 frequency
said	51	34	0,006187	0,006874
frodo	42	30	0,005095	0,006066
night	28	11	0,004125	0,002224

Table 2.7: Proportional difference between occurrence and frequency

As we can see in the table 2.7, while "said" in Text 1 occurs 17 times more that in text 2, the frequency is very similar. For what concern the wordform "frodo", it occurs 12 times more in first text but its weight is significantly lower.
However, in many cases, word frequency may be insufficient to a deep analysis or a comparison between more texts.

Using “The Hobbit” as text, we can perform other kind of analysis. In a book like “The Hobbit”, a novel, often the name of the main character recurs and has high frequency in every chapter. If we need to classify each chapter of the book by the words which appear in it, considering the Term Frequency, the word “bilbo” will always be selected in this list of words. In this case, its weight do not correspond with the amount of information the word may give us.

In order to resolve problems like this, we must introduce the *Term Frequency - Inverse Document Frequency* Weight (TF-IDF) [Salton and McGill, 1986]. In TF-IDF the Term Frequency value is calculated for each word, then, each value is multiplied by the number of document that contains that term in order to create a term-by-document matrix [Blei et al., 2003]. TF-IDF determines how relevant a given word is in a given document. Given a document collection D, a word w, and an individual document $d \in D$:

$$w_d = f_{w,d} * log(|D|/f_{w,D})$$

where $f_{w,d}$ equals the number of times w appears in d, $|D|$ is the size of the corpus, and $f_{w,D}$ equals the number of documents in which w appears in D [Salton and Buckley, 1988].

Using this formula it is possible to select high weighted words for each text and classify a text by its characterizing elements, those words that differentiate it from the other texts.
For example, the first chapter of “The Hobbit”, in which the main character, Bilbo receives the visit of Gandalf and the dwarves and must prepare them the dinner, and in which the group speaks about the map and the planned adventure, has the tf-idf ranking shown in table 2.8:

Type	**Tf-idf value**
plates	0.0598
gandalf	0.0591
map	0.0534
belladonna	0.0478
dungeons	0.0445

Table 2.8: Tf-Idf best ranked words for *The Hobbit*, Chapter 1

If this ranking is compared with the frequency rank for the same chapter, it is possible to notice that they contain different words (except “gandalf”) and, while all the words with the higher frequency value can be founded throughout the book, the words in table 2.8 return a good representation of the theme of the chapter.

It is also possible to follow the distribution of a group of words through the book, as, for example, the distribution of the words “hobbit”,“dragon”, “elves” and “battle”. In figure 2.1, a graph of the tf-idf distribution for these 4 words are presented. Is easy to notice that some words as “dragon” or “battle” increase their weight in the chapter in which the battle take places or the dragon appears. Contrariwise, the word “hobbit” which has higher values than the other words in the first 9 chapters, here has lower values.

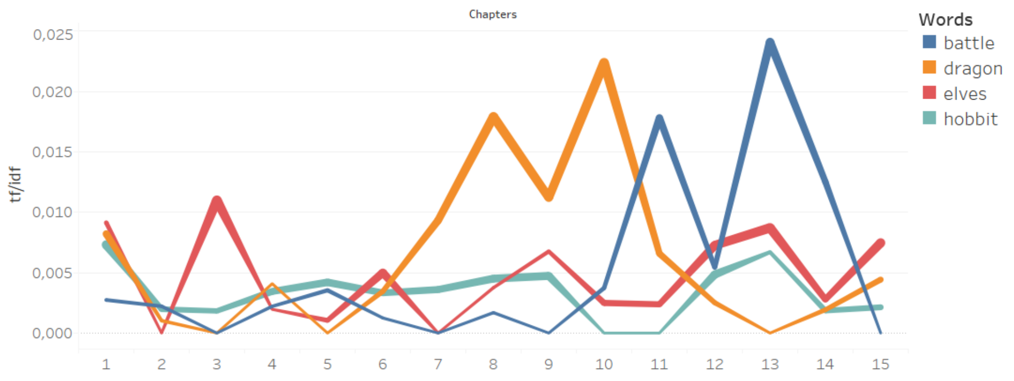


Figure 2.1: Tf-Idf distribution of 4 words through the book

Figure 2.1 represents words frequencies with different line widths in order to show how the tf-idf points out main words in a different way. The curve of “battle” (blue line), for example, emerge in tf-idf graph, in the chapter 11 and 13 where its width is smaller than the width of the red line, “elves”.

Another kind of brute text analysis which could be performed over an non-lemmatized, unstructured text is the extraction of N-grams. N-grams are sequences of N consecutive words that appear in texts one or more times. For example, in “The Hobbit” we can extract *bi-grams* as “said bilbo” or “your magnificence” or “worked metals”, that represent more complex recursive structures of the text. We can also extract *tri-grams* as “under the mountain” or “said the wizard” or “the great goblin”. Also N-grams could be accompanied by frequency values and could be used in order to find multiword expressions or elementary patterns.

2.5 Distributional Semantics

Distributional hypothesis started in the second half of '50 with Zellig Harris [Harris, 1954]. Harris states that the distribution of an element could be defined as the sum of all the contests of that element:

> *"An environment of an element* A *is an existing array of its co-occurrents, i.e. the other elements, each in a particular position, with which* A *occurs to yield an utterance.* A*'s co-occurrents in a particular position are called its selection for that position."*

When someone speaks, he have to chose the next word from a member of those classes of words that normally occur together. Each element in a language can be grouped into classes and while the relative occurrence of a class can be stated exactly, the occurrence of a particular member of one class relative to a particular member of another class must be calculated in terms of probability. In other words, if we discover that two linguistic units w_1 and w_2 has similar distributional properties, i.e. they occur with the same other entity w_3, then we can assume that w_1 and w_2 belong to the same linguistic class.

This distributional structure that underlies the language is not a mathematical creation, but exist in reality. It exist in the speakers in the sense of reflecting their speaking habits. Details of distributional perception differs on different speakers, but the structure exists for everyone.
The distributional hypothesis was then picked up by many other authors; Schütze and Pedersen [1995] affirm that *"words with similar meanings will occur with similar neighbors if enough text material is available"* in a work that presents an algorithm for word sense disambiguation based on a vector representation of word similarity derived from lexical co-occurrence.

Landauer and Dumais [1997] present a model for the simulation of knowledge learning phenomena based on local co-occurrence data in large representative corpus, called Latent Semantic Analysis (LSA). An unsupervised methodology for propagating lexical co-occurrence vector into ontology was presented by Pantel [2005] that based the algorithm on principle of distributional similarity which states that *"words that occur in the same contexts tend to have similar meanings"*.

Sahlgren [2008] points the focus on what kind of distributional properties should be token into account. Concerning this question, he discerns between two distinct approaches: the first one, that build distributional profiles for word based on which other words surround them; the second one that build distributional profiles based on which text regions words occur. However the two approaches are based on different types of distributional raw materials, they could be considered as functionally equivalent when it represents meaning similarities.

The first approach, called Hyperspace Analogue to Language (HAL) was presented by Lund and Burgess [1996]. In the paper, the semantic similarity between two words are calculated by comparing co-occurrence vectors of the two words with an Euclidean measure of distance. The co-occurrence matrix of words per words is generated analyzing a n-words windows through a large corpus of heterogeneous texts as showed in the table 2.9.

	barn	fell	horse	past	raced	the
<PERIOD>	4	5	0	2	1	3
barn	0	0	2	4	3	6
fell	5	0	1	3	2	4
horse	0	0	0	0	0	5
past	0	0	4	0	5	3
raced	0	0	5	0	0	4
the	0	0	3	5	4	2

Table 2.9: Example Matrix for “The Horse Raced Past the Barn Fell” computed for windows width of 5 words

In Burgess [1998] a HAL vocabulary of 70.000 most frequently used symbols is used to generate a matrix with a dimension of 70.000 X 70.000 and, word vectors have been processed with a multidimensional scaling algorithm to transform it into a two-dimensional pictorial representation of each word. This procedure generate semantic knowledge by grouping semantic neighbors and grammatical knowledge. The corpus used to generate the matrix is 300 million words of English text from Usenet newsgroups.

This model could be used in order to representing semantic meaning of words and allows the characterization of a variety of aspects of lexical ambiguity [Burgess, 2001]. HAL was also used to examine word category features as in Audet et al. [1999] that extract cognitively-relevant differences between the representations for abstract and concrete words, in particular, how they differ in the availability of associated contextual information stored in memory.

In order to realize an informational inference mechanism, Song and Bruza [2001] use HAL to build a high dimensional conceptual space that offers a cognitively motivated theory (expressed in terms of vectors) on which to found inference at a symbolic level. The matrix was generated on the Reuters-21578 collection with a vocabulary of 5403 words (stop words has been removed) and a windows size set to be 6. Azzopardi et al. [2005] propose a probabilistic Hyperspace Analogue to Language (pHAL): the probabilistic term co-occurrence matrix has been defined by normalizing the count of terms. The corpus used was a collection of 40.000 documents from the Wall Street Journal using a windows size of 5. This probabilistic model produces better average precision from query expansion derived from the Conceptual Spaces.

The second model, called Latent Semantic Analysis (LSA), was presented in Landauer and Dumais [1997]. Words are also represented as vectors derived from co-occurrence in text. In LSA vectors the semantic space is selected first as a set of contexts. The first 2000 characters in each of 30.473 articles in the Grolier's Academic American Encyclopedia was considered a space: each article is assigned to a column in the matrix and the 60.000 words are assigned to rows. Each time a word appear in an article the cross value is increased. Finally, matrix entries are logarithmically transformed and subjected to a singular value decomposition (SVD). The result is the extraction of about 300-400 important dimensions and each word's values on the dimensions.

LSA could be considered as a possible theory about all human knowledge acquisition, as a homologue of an important underlying mechanism of human cognition in general [Landauer and Dumais, 1997].

LSA could be used to address some questions into the challenge of understanding the meaning of the words. For Hofmann [1999] *polysems*, words which have multiple senses in different context, and *synonymys* , i.e. words that have a similar meaning, are partially addressed by standard LSA algorithm, due to its unsatisfactory theoretical foundation. For this reason Hoffmann propose a Probabilistic Latent Semantic Analysis that achieves best results on two tasks as the perplexity minimization and automated indexing of documents.

LSA was also used for word clustering [Bellegarda et al., 1996, Dumais et al., 1988], producing a semantic cluster of domain for generic vocabulary words or for automatic indexing and retrieval [Deerwester et al., 1990], by exploding latent semantic relations between words. Foltz et al. [1998] use Latent Semantic Analysis as a technique for measuring the coherence of texts.

Other approaches to distributional properties of texts could be founded in Senaldi et al. that use co-recurrence statistics of full content semantic words (nouns, verbs, adjectives and adverbs) in order to construct a semantic space that exploit relation between words of the COLFIS Corpus [Bertinetto et al., 2005], or Pointwise Mutual Information (PMI) [Baldwin, 2008] that represents one of the association measures available in the state of arts and calculate the force of the relation between two words in terms of probability that the word b occurs if a word a occurs.

2.6 Syntactic Dependency Parsing

Syntactic Parsing is one of the main open issue of Natural Language Processing and understanding. Martin and Jurafsky [2000] define parsing as "a combination of recognizing an input string and assigning a syntactic structure to it". Generally, parsing could be used in a great number of NLP algorithms as information extraction, opinion mining, machine translation or question answering.

Usually, parsers are classified in two main approaches, with at one side the constituency parsers, based on the idea that groups of words within utterances can be shown to act as single units, and, at the other side, the dependency parsers, which consider parse as a tree where nodes stand for words in utterance and links between nodes represent dependency relation within pairs of words [Lucien, 1959].

In Dependency Parsing, as nodes represents words, it is important to select a set of label for relations which may be of two main types: Argument dependencies, which are the most important kind, labeled as, for example, subject (*subj*), direct object (*obj*), indirect object (*iobj*); Modifier dependencies, labeled as determiner (*det*), modifier (*mod*), etc.

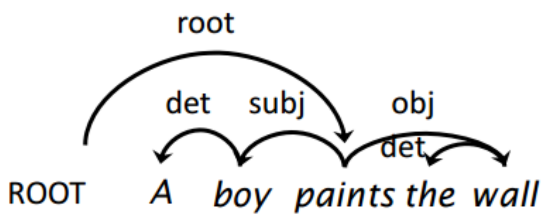


Figure 2.2: Example of dependency structure of a simple sentence

In fig. 2.2 is shown an example of dependency structure of the sentence "A boy paints the wall": the "root" of the sentence is shown as an external element connected with the main element of the sentence, the verb "paints", which is the *head* of the sentence. Two edges indicate the dependency of the arguments, subject and object, from the verb, and two edges labeled *det* indicates the dependency of the determiners from the respective name.

Deterministic dependency parsers derive a single analysis for each input string with no redundancy or backtracking, in a monotonic fashion [Nivre et al.]. Parser configuration are represented by triples $<$*S,I,A*$>$, where S is the stak, I is a list or remaining input tokens and A is the current arc relation for the dependency graph: an input string W initialize the parser with a $<$*null, W,0*$>$ state. The parser terminate when reaches a $<$*S, null,A*$>$ configuration, in other words when all tokens of the input string have been converted into nodes and arcs of a dependency graph. The way in which the parser determine the dependency structure includes a mix of bottom-up and top-down processing, constructing left-dependencies bottom-up and right dependencies top-down.

Obrębski presents a similar but non-deterministic algorithm. Non-deterministic parser may use a *guide* that can inform the parser at non-deterministic choice [Boullier, 2003]. Nivre et al. use a treebank to train a classifier that can predict the next transition given the current configuration of the parser.

Ensemble models for dependency parsing have been proposed by Nivre and McDonald [2008], that integrate Graph-Based and Transition-Based Dependency Parser, and by Attardi and Dell'Orletta [2009] that combine deterministic transition-based linear deterministic dependency parser with a Left-Right parser and a stacked right-to-left or *Reverse Revision* parser, both in learning time. The models proposed by Hall et al. [2010] and Sagae and Lavie [2006] combine independently-trained models at parsing time. In Surdeanu and Manning [2010] is demonstrated that ensemble models that combine parsers at runtime reach better results than models that combines two parsing at learning time.

Nivre and Nilsson [2004] study the influence of Multiword Units on deterministic parsing accuracy. In the model, Multiwords units have been manually annotated in the training corpus and then the parser has been tested in both, a lexicalized and a non-lexicalized version. The results shown that parsing accuracy increases not only for Multiword Units themselves, but also with respect to surrounding syntactic structures.

Collins [1997] presents a statistical parser which uses a lexicalized grammar for English language. For each non-terminals $X(x)$, X is the non-terminal label and x is a pair of $<w,t>$ where w is the associated head-word and t is the POS tag. Rules calculate the *Parents* of each head-child by considering left and right context, and head-child inherits head-word from it. Each rule has an associated probability defined as a product of term probabilities.

While much of researches on statistical parsing were focused on English, other languages pose new problems for statistical methods: Collins et al. [1999] propose a new method for Czech, a highly inflected language with relatively free word order. The first problem leads to a very large number of possible word forms and pose the question on how to parametrize a statistical parsing model in order to make good use of inflectional information.

Dependency parsing tools available are, for example, the Stanford Parser[2] which is a Probabilistic Natural Language Parser for English [Chen and Manning, 2014, Manning et al., 2014] then adapted mainly for Chinese, German, Arabic, but also Italian; MST Parser[3] which is a two-stage multilingual dependency that works on 13 different languages [McDonald et al., 2006], Bohnet's Parser [Bohnet, 2010] which is a graph-based parser included into the *Mate Tool*[4]. For Italian Language we cite *LinguA*[5], a Linguistic Annotation pipeline which combines rule-based and machine learning algorithms and includes a dependency parsing [Attardi and Dell'Orletta, 2009].

2.7 Lexicon-Grammar Framework

With *Lexicon-Grammar* (LG) we mean the method and the practice of formal description of the natural language, introduced by Maurice Gross in the second half of the 1960s, who, during the verification of some procedures from the transformational-generative grammar (TGG) [Chomsky, 1965] laid the foundations for a brand new theoretical framework.

During the description of French complement clause verbs, through a transformational-generative approach, Gross [1975] realized that the English descriptive model of Rosenbaum [1967] was not enough to take into account the huge number of irregularities he found in the French language.

LG introduced important changes in the way in which the relationship between lexicon and syntax was conceived [Gross, 1971, 1975]. It has been underlined, for the first time, the necessity to provide linguistic descriptions grounded on the systematic testing of syntactic and semantic rules along the whole lexicon, and not only on a limited set of speculative examples.

The huge linguistic datasets, produced over the years by the international

[2]`http://nlp.stanford.edu:8080/parser/`
[3]`http://www.seas.upenn.edu/`
[4]available at `http://code.google.com/p/mate-tools/`
[5]`http://www.italianlp.it/demo/linguistic-annotation-tool/`

LG community, provide fine-grained semantic and syntactic descriptions of thousands of lexical entries, also referred as "lexically exhaustive grammars" [D'Agostino, 2005, D'Agostino et al., 2007], available for reutilization in any kind of NLP task[6].

The LG classification and description of the Italian verbs[7] [Elia et al., 1981, Elia, 1984, D'Agostino, 1992] is grounded on the differentiation of three different macro-classes: transitive verbs; intransitive verbs and complement clause verbs. Every LG class has its own definitional structure, that corresponds with the syntactic structure of the nuclear sentence selected by a given number of verbs (e.g. *V* for *piovere* "to rain" and all the verbs of the class 1; *N0 V* for *bruciare* "to burn" and the other verbs of the class 3; *N0 V da N1* for *provenire* "to come from" and the verbs belonging to the class 6; etc...). All the lexical entries are, then, differentiated one another in each class, by taking into account all the transformational, distributional and structural properties accepted or rejected by every item.

The Lexicon-Grammar theory lays its foundations on the Operator-argument grammar of Zellig S. Harris, the combinatorial system that supports the generation of utterances into the natural language.
Saying that the operators store inside information regarding the sentence structures means to assume the nuclear sentence to be the minimum discourse unit endowed with meaning [Gross, 1992].
This premise is shared with the LG theory, together with the centrality of the *distributional analysis*, a method from the structural linguistics that has been formulated for the first time by Bloomfield [1933] and then has been perfected by Harris [1970].
The insight that some categories of words can somehow control the functioning of a number of *actants* through a dependency relationship called *valency*, instead, comes from Lucien [1959].
The *distribution* of an element A is defined by [Harris, 1970] as the sum of the contexts of A. Where the *context* of A is the actual disposition of its co-occurring elements. It consists in the systematic substitution of lexical items with the aim of verifying the semantic or the transformational reactions of the sentences.

All of this is governed by verisimilitude rules that involve a graduation in the acceptability of the item combination.

Although the native speakers of a language generally think that the sentence elements can be combined arbitrarily, they actually choose a set of items

[6]Italian LG descriptions are available for 4,000+ nouns that enter into verb support constructions; 7,000+ verbs; 3,000+ multiword adverbs and almost 1,000 phrasal verbs [Elia, 2014].

[7]available at `http://dsc.unisa.it/composti/tavole/combo/tavole.asp`.

along the classes that regularly appear together.

The selection depends on the likelihood that an element co-occurs with elements of one class rather than another[8].
When two sequences can appear in the same context they have an *equivalent distribution.* When they do not share any context they have a *complementary distribution.*
Obviously, the possibility to combine sentence elements is related to many different levels of acceptability. Basically, the utterances produced by the words co-occurrence can vary in terms of plausibility.
In the operator-argument grammar of Harris, the *verisimilitude of occurrence* of a word under an operator (or an argument) is an approximation of the likelihood or of the frequency of this word with respect to a fixed number of occurrence of the operator (or argument) [Harris, 1988].

Concerning *transformations* [Harris, 1964], we refer to the phenomenon in which the sentences A and B, despite a different combination of words, are equivalent to a semantic point of view. (e.g. *Floria ha letto quel romanzo = quel romanzo è stato letto da Floria* "Floria read that novel = that novel has been read by Floria" [D'Agostino, 1992]). Therefore, a *transformation* is defined as a function T that ties a set of variables representing the sentences that are in *paraphrastic equivalence* (A and B are partial or full synonym) and that follow the *morphemic invariance* (A and B possess the same lexical morphemes).

Transformational relations must not be mixed up with any derivation process of the sentence B from A and *vice versa.* A and B are just two correlated variants in which equivalent grammatical relations are realized [D'Agostino, 1992].

Because the presentation of the information must be as transparent, rigorous and formalized as possible, LG represents its classes through binary matrices like the one exemplified in Table 2.10.

They are called "binary" because of the mathematical symbols "+" and "–" respectively used to indicate the acceptance or the refuse of the properties P by each lexical entry L.
The symbol X unequivocally denotes an LG class and is always associated to a class definitional property, simultaneously accepted by all the items of the class (in the example of Table 2.10 P_1). Subclasses of X can be easily built by choosing another P as definitional property (e.g. L_2, L_3 and L_4 under the definitional substructure P_2).

[8] Among the traits that mark the *co-occurrences classes*; we mention, by way of example, human and not human nouns, abstract nouns, verbs with concrete or abstract usages, etc. [Elia et al., 1981]

$\boldsymbol{X}$	P_1	P_2	P_3	P_4	P_n
L_1	+	−	−	−	+
L_2	+	+	−	−	−
L_3	+	+	+	+	+
L_4	+	+	+	+	+
L_n	+	−	−	−	+

Table 2.10: Example of a Lexicon-Grammar Binary Matrix

The sequences of "+" and "–" attributed to each entry constitute the "lexico-syntactic profiles" of the lexical items [Elia, 2014].
The properties, on which the classification is arranged, can regard the following aspects [Elia et al., 1981]:

distributional properties, by which all the co-occurrences of the lexical items L with distributional classes and subclasses are tested, inside acceptable elementary sentence structures;

structural and transformational properties, through which all the combinatorial transformational possibilities, inside elementary sentence structures, are explored.

It is possible for a lexical entry to appear in more than one LG matrix. In this case we are not referring to a single verb that accepts divergent syntactic and semantic properties, but, instead, to two different *verb usages* attributable to the same morpho-phonological entry [Elia et al., 1981, Vietri, 2004, D'Agostino, 1992]. Verb usages possess the same syntactic and semantic autonomy of two verbs that do not present any morpho-phonological relation.

Chapter 3
The project

Since there is a lack of open-source resources of this kind for Italian language, and none of these are based on the Lexicon-Grammar Theory, the present work aims to fill this gap by illustrating a project of a new hybrid Linguistic software for the automatic treatment of text, called LG-Starship Framework, which supply both current main families of approaches of text analysis, statistics and Rule-based algorithms.

The idea is to built a modular software that includes, in the beginning, the basic algorithms to perform different kind of analysis. Modules will perform the following tasks:

- Preprocessing Module: a module with which it is possible to charge a text, normalize it or delete stop-words. As output, the module presents the list of tokens and letters which compose the texts with respective occurrences count and the processed text.
- Mr. Ling Module: a module with which POS tagging and Lemmatization are performed. The module also returns the table of lemmas with the count of occurrences and the table with the quantification of grammatical tags.
- Statistic Module: with which it is possible to calculate Term Frequency and *tf-idf* of tokens or lemmas, extract bi-grams and tri-grams units and export results as tables.
- Semantic Module: which use The Hyperspace Analogue to Language algorithm to calculate semantic similarity between words. The module returns similarity matrices of words per word which can be exported and analyzed.
- Syntactic Module: which analyze syntax structures of a selected sentence and tag the verbs and its arguments with semantic labels.

The objective of the Framework is to build an “all-in-one” platform for NLP which allows any kind of users to perform basic and advanced text analysis. With the purpose of make the Framework accessible to users who have not specific computer science and programming language skills, the modules have been provided with an intuitive GUI.

The Framework takes the start from a text or corpus written directly by the user or charged from an external resource. The LG-Starship Framework workflow is described in the flowchart shown in fig. 3.1.

As we can see, the pipeline shows that the Pre-Processing Module is applied on original imported or generated text in order to produce a clean and

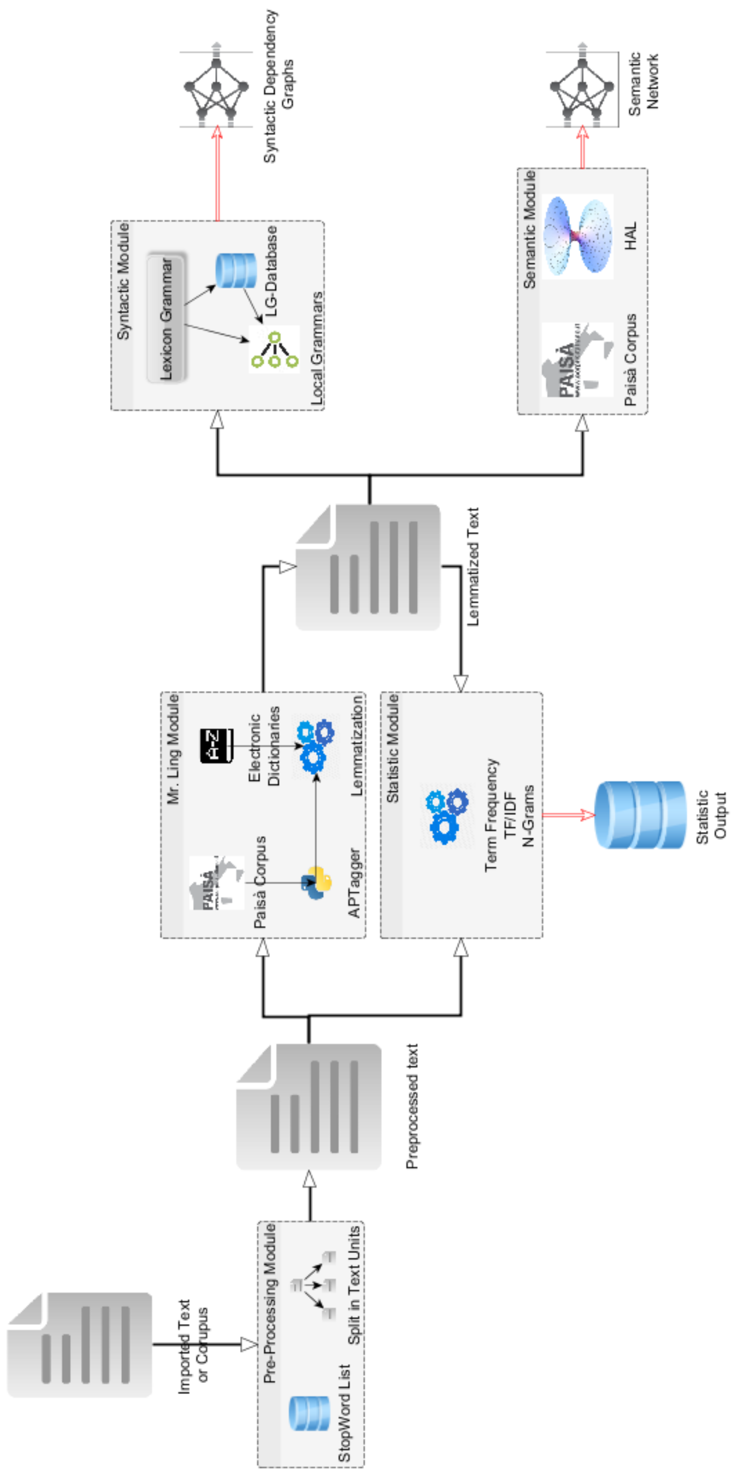


Figure 3.1: Workflow of the LG-Starship Framework.

normalized preprocessed text. On this text can be applied the Statistic Module or the Mr. Ling Module. The first one produces as output a database of lexical and numerical data which can be used to produce charts or perform more complex analysis; the second one produces a new lemmatized version of original text with information about POS tags and syntactic and semantic properties by using an adaptation of the Averaged Perceptron Tagger, in order to POS Tag the text and a series of electronic dictionaries to lemmatize it.

This lemmatized text, which can be processed with the statistic module with the purpose to produce quantitative data, is the input for two deeper level of text analysis carried out by both the Syntactic Module and the Semantic Module.
The first one lays on the Lexicon Grammar Theory and use a database of Predicates in development at the Department of Political, Social and Communication Science. Its objective is to produce a Dependency Graph of the sentences that compose the text.
The Semantic Module uses the Hyperspace Analogue to Language distributional semantics algorithm trained on the "Paisà Corpus" to produce a semantic network of the words of the text.

The project is developed in Java and Python and uses Jython, a Python interpreter for Java. While python is particularly suitable for NLP, Java offers a vast set of external packages, including packages for GUI, distributional semantics or data-visualization, and permits to interface the code with a large number of external resources through API.

In this chapter the modules which compose the Framework will be presented. In the first part we illustrate the technical foundation of the project. In the second part the modules and the algorithm which compose them will be described and analyzed. At the end we present the GUI with illustration and examples of use.

3.1 Technical Background

All project modules are written in Java Programming Language and includes some basic Java algorithms for different purposes. However, Pos Tag algorithm is built as a Python module integrated in the Java code by using a Python interpreter for Java called Jython. In this section we will briefly present the basic of the two programming languages and the external modules we implement, and we will define a meta-language in which we illustrate modules algorithms and functions.

Java is one of the most popular programming languages, with more than 9 million of developers, and it is present in the 97% of desktop computers in United States[1]. Java is a computer programming language that is:

- General-purpose: a general-purpose programming language is designed to be used for writing software in many application domains.
- Concurrent: Java uses a form of computing defined Concurrent Computing in which several computations are executed during overlapping time periods, and is opposite to sequentially [Herlihy and Shavit, 2006].
- Class-based: a style of object-oriented programming in which inheritance is achieved by defining classes of objects as opposed to the objects themselves. Objects are entities that combine *states* (i.e. data), *behavior* (i.e. methods) and *identity*. Classes define the structure and behavior of an object by representing the definition of all objects of a specific type [Wegner, 1990].
- Object-Oriented: a programming paradigm based on the concept of "objects" which may contain data, or code in the form of procedures, also knowns as Methods. Object's procedures can access and modify the data fields of the object with which they are associated [Lewis and Loftus, 2008].

Java is designed to have as few implementation dependencies as possible and a compiled Java Code can run on all platforms that support Java without the need of recompilation.
Java is particularly used for client-server web application, and derive its syntax from *C* and *C++* programming languages.

Java implementation is packaged into two distributions: the Java Runtime Environment (JRE), which contains the parts of the Java SE platform required to run Java programs and is intended for end users, and the Java Development Kit (JDK) which is intended for software developers and includes development tools as Java compiler, Javadoc, Jar and debugger. In this work we used the *Eclipse Compiler for Java*, an open source incremental compiler used by the Eclipse Project. An example of Java code is the following:

```java
class HelloWorldApp {
  public static void main(String[] args) {
```

[1]https://www.java.com/it/about/

```
    System.out.println('Hello World!'); // Prints the string code
       to the console.
  }
}
```

In the example, we first declare a *class* named “HelloWorldApp”. Blue code represents Java reserved words and code in green represents comments. Purple code represents *strings*. Source file must be named as the public class they contain, for example *HelloWorldApp.java*. This file must be compiled into bytecode producing a file named *HelloWorldApp.class* which can be executed. Each java source file may only contain one public class, but it can contain multiple classes with no public access or public inner classes. The code in blue represents keyword and “public” denotes that a method can be called from code in other classes. This access level can be modified as “private” or “protected”.
Keyword “static” indicates a *static method* which is associated only with the class. Static methods can be invoked without a reference to an object. Keyword “void” indicates that the main method does not return any value to the caller.
The “main” is the name of the method which the java launcher calls to pass control to the program. The main method must accept an array of *String* objects.
Printing is part of the Java standard library: the *System* class defines a public static field called *out* which is an instance of *PrintStream* class and provides a method for printing data to standard out, such as *println(String)* which appends a new line to the passed string.

Python [Van Rossum, 1993] is a widely used programming language which has the following features:

- High-level: Python is a programming language with a strong abstraction from the details of the computer. High-level programming languages may use natural language elements and may automate significant areas of computing systems, making the process of developing a program simpler and understandable.
- General-purpose.
- Dynamic: a class of high-level programming language which, at runtime, execute many common programming behaviors that static programming languages perform during compilation.

Python design philosophy emphasizes code readability and its syntax

allows programmers to express concepts in fewer lines of code than possible in Java or C++. Python supports multiple programming paradigms including object-oriented or functional programming. It features automatic memory management and has a large standard library.
Python uses whitespace indentation to delimit blocks rather then braces or keywords, and its statements includes "=" in a different way from other programming languages: $x = 2$ do not give to a variable x a value of 2, but may be translated to "*a generic name* x *receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2*". Other Python statement includes *if*, *for*, *print*, etc.
An example of Python code is the following:

```
# This program prints Hello World!
print 'Hello World!'
```

Instead of using five line, Python code hide structures and allows to reduce the code to one line of code. A more complex example my be the following:

```
# This program prints Hello World! in a different more complex way
a = 'Hello'
b = 'World'
c = '!'

d = a + ' ' + b

print d+c
```

In this example we start by assigning to each generic value a String specific value. The variable *d* is defined as a concatenation of the content of variable *a*, a whitespace and the content of *b*. The entire string sequence is obtained by printing the value of *d* concatenated to the value of *c*.
In the following example is shown how Python uses indentation rather then braces:

```
# This program checks if a number is positive, negative or zero
num = float(input("Enter a number: "))
if num > 0:
   print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

Taking a number as an input from the user, the program checks if a number is positive, negative or zero using an *if...elif...else* statement. the symbol ":" introduces the result to give if the condition is verified and expects an indentation in next line.

In order to present the work as simply and clearly as possible, we must define the characteristics of a *pseudocode* in which algorithms will be presented from now on. The following examples of pseudocode represents the "positive-zero-negative" script shown above as example for the Python code:

Algorithm 1 Algorithm for checking if a number is Positive, Negative or Zero

```
Input: a number n from a User
1: if n > 0 then
2:     print n is a positive number
3: else if n < 0 then
4:     print n is a negative number
5: else
6:     print n is a zero
```

With this kind of pseudocode we are able to illustrate in a schematic way the algorithms that compose programs and cite specific lines of code in order to describe specific steps or details. In 1 we start specifying the Input data of the algorithm before, and then, with a simple "if-else statement" we define condition and outputs for each possible input. Each algorithm presented in pseudocode may include:

- **Input**: the data the algorithm expect to receive before start. When a specific algorithm was called, the INPUT must be written between round brackets after the name of the called class.
- **If-Else** statements: which are used to control if a given condition is fulfilled.
- **For** loop: a cycle statement that, for example, walks through all element of a list and performs some given instructions.
- Logical connectives: which include conjunction **and**, disjunction **or**, negation **not**.

- **Return**: which specifies the results that the algorithm produces.

- Boolean conditions: **True** or **False** control conditions.

- **Print**: which specifies the printed output of the algorithm.

The project makes use of some particular types of variables. In our pseudocode, we try to simplify the description of the methods by unify some similar variable types which difference counts, specially, in terms of coding. In pseudocode, we use only the following types:

- **String**: a string of alphanumeric and symbolic characters.

- **Integer** or **int**: an integer number.

- **Double**: a floating-point number. This types includes Java and Python types *float* and *double*.

- **List**: a list of elements of a specific type or different types. Each list will declare what kind of element it accepts by specifying the type within angular brackets (i.e. List<String> or List<String,int>). It is possible to find Lists of lists. Lists includes Arrays, Arraylists, or Vectors in java or python programming languages.

- **Set**: a Set is a list in which each element appears only once.

- **Map**: a Map is a list in which we have a first element that may be treat as a "key" and a second element, connected to the key, which may contains any types of elements. They works as Java Maps or ad Dictionary in python.

In next section we will present all the algorithms that compose the modules of the project.

3.2 The LG-Starship Project

In this section, we will present the modules that compose the Framework. Each module is composed by a collection of multiples algorithms which will be described in detail.

3.2.1 The Preprocessing Module

The standard operations which are normally performed between text selection and text analysis, are carried out in the preprocessing module, which includes:

- Text Selection: a simple routine that allows users to charge new texts from files into the module;
- Text Cleaning: an optional method that prepares texts to future analysis by deleting punctuation, normalizing capital letters, etc.
- StopWords removing: an optional method which removes stop-words from a previously provided list.
- Alphabetical Letters count: a method that divides text into list of letters and count occurrences.
- Tokenization: a method that divides texts into list of words and count their occurrences.

For what concern the first operation, the text selection, the algorithm accepts a path of text files written in a specific encoding: in fact, machines treat texts as a sequence of encoded information and for they, there are not differences between an alphanumeric characters and whitespace or other kinds of symbols. For English Language the standard ASCII encoding, acronyms of American Standard Code for Information Interchange, which is a 7-8 bit character encoding based on English alphabet, may be accepted, though is considered obsolete since the W3C adopt other encoding as the “UTF-8”. For what concern Italian Language, the preferred encoding is the UTF-8, which is able to represent all Italian character as accented characters. The Text Selection module accept all encoding, but only texts encoded in UTF-8 can be read without a loss of information.

In order to select a text, the Text Selection method accept a file-path as Input information and convert it into a String. In alternative, it is always possible to give the String directly to the program, or to use other external methods to charge texts.

Once the text is selected, the module provides some cleaning methods that could be applied or not, depending on the type of analysis that will be performed.

Normally, normalization steps can be considered as baseline, such as the coding of accents and other diacritic signs, separation into articles, paragraphs

and sentences, preprocessing of digits, units, the correction of typical formatting and punctuation errors. Many of these steps, in this project are not considered as part of normalization process, but are included into other modules as essential operations of text analysis [Adda et al., 1997]. In the present project we consider the Normalization method as a preprocessing method that allows users to delete punctuation and to reduce text in lowercase.
In some cases it may be useful to delete punctuation or to reduce the entire text to lowercase. This is the case of the statistical analysis, in which the analysis on a normalized text could return better results: the count of occurrences, frequencies or tf-idf, as written in section 2.4, is more precise when we can combine the word "Hobbit" with "hobbit". However, also in a statistical analysis, could be useful to preserve punctuation, i.e. in analysis of text style, or capital letters, i.e. in the case of Named Entity Recognition.
The method use a provided list of punctuation symbols as input:

Algorithm 2 Text Cleaning method

```
Input: a String text
Input: A List<String> punct
1: text = text.toLowerCase()
2: for (String element in punct) do
3:     text = text.replace(element, '')
Output: text
```

As input the algorithm needs a text, given by the user or transmitted by the Text Selection method, and a list of punctuation symbols (already included into the package). In line 1, the text is rewritten in lowercase by using the String standard embedded method *toLoweCase()* which does not require arguments (as empty round brackets indicate). In the line 2 a For Cycle is open: for each element of the list *punct*, takes this element and replace it with nothing. In the For Cycle we declare a String variable *element* that contains, alternately, each element presents in the given list. In order to replace elements we used the *replace* method, that requires two arguments, a String variable that must be replaced and a String for replacing it.

The same goes for StopWords Removing method: Stop-words can affect the retrieval effectiveness because they have a very high frequency and tend to diminish the impact of frequency differences among less common words. But the removal of Stop-Words also changes the document length and subsequently affects the weighting process [Korfhage, 2008].
In some cases, such as in statistical analysis, removing Stop-Words may be useful to increase the efficiency of a information retrieval process [El-Khair,

2006], but in a rule-based analysis, Stop words are required in order to detect syntactic structures and determine, for example, the complement type.
As for punctuation, in order to detect stop-words the module relies on a list of common Italian stop-words and replace it with an empty string into texts. Stop-words List includes all determiners and preposition, pronoun particles, possessive adjectives, conjunctions, auxiliary verbs *essere*, "to be" and *avere*, "to have".

Another method present in the module performs the count of letters included into the text. This operation, that could be useful in an extremely general analysis, simply takes the text as input and splits it in characters. Once creates a Set<String> which contains each letter, number and symbol in text, perform the count of occurrence of them. This method provides a distant view of the composition of text that can help user in detecting encoding errors or, for example, recognition of language.

The core of the module is represented by the Tokenization method. This method perform various essential tasks that could be considered the base for any subsequent kind of text analysis.
In first instance, the method corrects some errors as the excess of whitespaces and separates the punctuation symbols from words. This task is not trivial as it seems because it depends on language. In fact, in Italian, we can find word forms that include some symbols or need some symbols to return their full meaning. In particular, the Italian language is rich of words that use apostrophes to bind to the first vowel of the following word or simply to represent forgotten letters. For example, determiners *lo*, *la*, *una*, or preposition as *di*, may appear in a contract form when appear before a word that starts with vowel or "h": *un'amica*, "a friend (female)', *l'hotel*', "the hotel", *l'inverno*, "the winter" or *la macchina d'Antonio*, "the car of Antonio".
In order to avoid this problem it was necessary to separate normal punctuation from apostrophes or hyphen. The resulting algorithm is the following (texts in braces are comments):

In the first part of the algorithm the text has been treated in order to separate punctuation symbols from words. The sequence *!=* in line 2 means "not equal".
In line 4, each punctuation symbol that is not apostrophe, must be written between two whitespace in order to separate both symbols before and after words (i.e. in the case of quotation marks). After the cycle, in many cases, the algorithm produces double whitespace and in line 5 they are replaced with single whitespace. Then tokenization starts.
The first operation (line 7) is to create a list of strings that contains each word of the text. This list is a token list. It contains the elements in the exact order

Algorithm 3 Tokenization method

```
Input: a String text
Input: A List<String> punct {preparing text}
 1: for (String element in punct) do
 2:    if element != "/" then
 3:       text = text.replace(element, ' '+element+' ')
 4: text = text.replace(' ', ' ')
 5: {tokenization start}
 6: List<String> tokens = text.split(' ')
 7: Set<String> types = tokens.toSet()
 8: Map<String,Integer> tokenizedText= new Map<String,Integer>
 9: for (String element in types) do
10:    int i = count of element in tokens
11:    tokenizedText.put(element, i)
Output: tokenizedText, tokens
```

we can found them into the text and in the same quantity. In line 8 a Set of string called *types* is defined. The list of token is converted into a Set an put in *types*. In line 9 a Map of strings and integers is defined as a new, empty map. Then, for each element of types, the algorithm count how much times it appears in the tokens list and set an integer *i* with the resulting number. The couple of type and integer is then put into the map. When the for cycle terminates, the Map is returned as output.

The output of this module could be used for analysis purposes or as input of other modules of the project: tokens, that is the tokenized representation of the text, will be used for the POS Tagging phase. tokenizedText, that contains tokens and its occurrences, will serve to future statistic analysis.

3.2.2 Mr. Ling Module

As we said in 2.3, Part-of-Speech Tagging and Lemmatization represent two essential steps of text analysis. In this section, we will present the Mr. Ling Module, which is the result of the union of two independent modules called Mr. Tag and Mr. Lemmi.

The Pos Tagger module, called *Mr Tag*, is based on an implementation of an Averaged Perceptron Tagger[2] for the English language, part of "TextBlob" module. The original Tagger, written in Python 2.7, tries to predict the tag of a word computing a weight.

Starting from a training set, built on a tagger corpus, the tagger assigns each

[2]https://github.com/sloria/textblob-aptagger

known word the tag it has in the training set. If a word is unknown, or has more than one tag, the tagger tries to assign to it a random tag and, based on a given feature set, computes the weight of the tag for that word. After a given number of iteration, in each one of them the tagger tries a different tag for the word, the tag with the greatest weight wins and is assigned to the word.

We start the expansion of the set of features used by the original model, to make them more suitable for the Italian language. Then, we perform a semi-supervised training phase on a 1 million-word tagged corpus extracted from the "Paisá Corpus" [Lyding et al., 2014]. The new introduced features are focused on morphosyntactical differences between English and Italian. Our Feature Set is shown in table 3.1.

Type	**Features**
Morphology	first charachter of present word
	last 4 characters of present word
Syntax	previous word tag
	second previous word tag
	two previous word tags
	next word tag
	second next word tag
	two next word tags
	previous and next word tags
	two previous and two next word tags
Context	present word
	next word
	previous word
	two previous words
	two next words
	next and previous word
	suffix of previous word
	prefix of previous word
	second previous word
	second next word

Table 3.1: Set of Features Used for *Mr Tag* PosTagger

Since perceptron-based taggers are mainly based on a set of language-dependent features, we started looking for the features that are effective

for the Italian language. To select representative features we performed a morphological analysis and notice that, on average, Italian words are longer than English ones [Graffi and Scalise, 2002]. So we choose to take into account the last 4 characters of each words (i.e. Italian adverbs, normally end with the suffix *mente* and in verbs declinations we could have suffix as *asse* or *ebbe* in construction like *mangi-a-sse* or *dorm-i-rebbe*) instead of 3, and one character in the case of prefixation phenomena.
Syntactical and contextual features, instead, have been expanded by taking into account a bigger context for each word, including one or two words before and after the target word, this is due to the large use of determiners and distant dependencies that characterize the Italian language.

Mr. Tag module, enriched with this set of features, has been trained on the "Paisá Corpus",a large Italian Corpus (250 millions tokens) of text extracted from web pages and annotated in CoNLL[3]. However, we decided to use the "DELA" Tag Set [Elia, 1995, Elia et al., 2010] that includes:

- Names: N
- Verbs: V
- Adjectives: A
- Adverbs: AVV
- Determiners: DET
- Prepositions: PREP
- Pronouns: PRON
- Conjunctions: CONG
- Numerics: NUM
- Interjections: INTER
- Others: X

A tag for punctuation (PUNCT) has been added to the tagset because in the DELA italian Tagset it was not present.

The training phase produces as output a ".pickle" file that contains tag information about known words. The tagging algorithm is presented in 4.

Mr. Tag proceeds by creating contextual variables that are used in order to calculate the weights each feature give to the perceptron prediction algorithm.

[3]CONLL

Algorithm 4 Mr. Tag method

```
Input: a List<String> tokenizedText
Input: a PICKLE file
1: List<String,String> tagged = new List<String,String>
2: for (String word in tokenizedText) do
3:     prev, prev2 = word-1, word-2 {previous two words of the selected word
       are stored into variables}
4:     next, next2 = word+1, word+2 {next two words of the selected word
       are stored into variables}
5:     tag = tagdict.get(word) {tagdict is retrieved from the PICKLE file}
6:     if not tag then
7:         features = word.getFeatures(prev, prev1, next, next1)
8:         tag = Peceptron.predict(features)
9:     tagged.append(word,tag)
Output: tagged
```

After trying to assign the tag directly from PICKLE file, if the tag do not exists, an external method *getFeatures* uses the two previous words and the two next words to generate, for each present word a list of features for each possible tag. Then, Mr. Tag employ the Perceptron Tagger method called *predict* which uses features to assign a weight score to each tag for the present word, and, after 10 iteration, select the tag with the maximum value.

Mr. Tag Python code was subsequently added to a Java module using *Jython*[4], a Python Interpreter for Java that allows to insert python Code into Java code and vice versa. Since the Python module could be used by call the *tag* method with the text as argument directly after import the module, Java version uses a *Postagger* method which take as argument a list of tokens. A method *getTagList*() take the list of Strings produced by the Postagger and returns an array of arrays of Strings (*String[][]*) which contains tokens and tags.

The Lemmatization module, called *Mr. Lemmi*, is based on a set of dictionaries annotated in "DELA", in particular on DELAF, the Italian Electronic Dictionary of Flexed Forms. DELAF includes over 1 million of flexed Italian forms and it has been divided into 6 sub-dictionaries in order to improve performance of the algorithm.*Mr. Lemmi* uses a different dictionary for each part of speech (with the exception of Adverbs) in combination with a set of rules for decomposition of compound prepositions (i.e. *della*, that is decomposed as *di,PREP+la,DET*). As a first step, the Lemmatization module charges the

[4]`http://www.jython.org/`

three main dictionaries, Nouns, Verbs and Adjectives, converting them into variables. For each dictionary file, the method creates one variable for each alphabet letter and puts in all the dictionary entries that start with the same letter. Other dictionaries, such as the ones containing prepositions, conjunctions or pronouns, have been manually introduced into variables and inserted into the code.
Dictionary entries are composed of four parts: the token, the lemma, the POS or grammatical category and a list of syntactic and semantic properties separated by the "+" symbol. An example of dictionary entry is the following:

(4) case,casa,N+f+p

As the information after the Part-of-Speech are not classified as in a database, but for each word we can find a different number of disparate information, we had to consider all these information as one String and put them into a residual part of the dictionary entries variables.
Once the dictionaries are stored into variables, the second steps was to separates and lemmatizes Compound Prepositions; then it labels each word the corresponding Tag obtained after the POSTagging phase. After that, invariable parts of speech, like Adverbs or Conjunctions were directly added to the a list of lemmas belonging to text and lemmas retrieved from a sub-dictionary of DELAF are added to every words. In order to avoid Lemmatization errors and to correct Pos Tagging errors, if a word is not present on a variable of its specific tag, a new iteration search the word in all variables relative to other tags. In this way, if the word *perché*, "why", was wrongly tagged as noun, and the first iteration is unable to find it in the Noun variable, the second iteration search it in all other tag variables and can find it in Conjunction variable and then, correct the tag. If a word is not present in any other variable, then the algorithm assign it the original Tag from the Postagger and uses the token as Lemma. For example, if the word was *hobbit*, which is not present in the DELAF dictionary, after the second iteration the resulting Lemma will be *hobbit*.
The method *Lemmat* that has not arguments, is based on a *Lemmatize* algorithm, that uses the list of tokens and tags resulting from the Pos Tagging phase and the dictionary variables and calculates, for each token, the respective Lemma.

Both POS Tagger and the Lemmatizator have been tested on a corpus of 250.000 tokens extracted by a different portion of the "Paisá Corpus". Since "Paisá Corpus" was a recollection of over 380.000 different documents belonging to over 1.000 different web pages, the corpus resulted heterogeneous and each section presented completely different properties, so it was suitable for an efficient test phase.

To evaluate *Mr Tag* and *Mr.Lemmi*, we decided to compare their performances with other two free tools for Italian Language: Pattern and Tree-Tagger. Pattern [Smedt and Daelemans, 2012] is a Web Mining module written in Python that includes POS Tagging for English, Spanish, French, German, Duch, and Italian. Tree-Tagger [Schmid, 1995] is the most known tool for annotating text with part-of-speech and lemma information, it is available for several languages.
We compared the tools evaluating the results obtained from the analysis on the same data portion. An overview of the comparison is presented in table 3.2 and 3.3.

POS Tagging Task		
Algorithms	**Precision**	**Time**
Mr.Tag	0,931	0,213 sec.
Pattern.it	0,752	1,094 sec.
Tree-Tagger	0,858	3,409 sec.

Table 3.2: Comparison of POS Tagging task

Lemmatization Task		
Algorithms	**Precision**	**Time**
Mr. Lemma	0,969	1,638 sec.
Pattern.it	-	-
Tree-Tagger	0,927	3,409 sec.

Table 3.3: Comparison of Lemmatization task

As showed by tables 3.2 and 3.3, our modules reach very good results in terms of precision for both tasks, overcoming both Pattern and Tree-Tagger. Differently from what concerns the time, we reach a good result with the POS Tagger module, but the Lemmatizator still requires many adjustments. In fact, Tree-Tagger performs both tasks concurrently, so the time expressed by the tables must be considered as an unique time. Otherwise, our modules perform the two tasks subsequently, therefore the two values must be added to each other. In addition, Mr. Lemmi needs a time of approximatively 75 second to charge dictionary and start the algorithm, however that time is only needed when the algorithm was initialized.

At the end, we analyze in a qualitative way the results of the Lemmatization phase. The vast majority of errors derives from an incorrect part-of-speech tagging in ambiguity cases (i.e. *condotta,V*, lemmatized as *Condurre*, “to lead”, instead of *condotta,N* that could be lemmatized as *condotta,N*, “conduct”).

3.2.3 Statistic Module

While previous sections refers to modules that prepare texts to any kind of analysis, from tokenization to POS tag and Lemmatization, producing lists of tokens, tags and lemmas, this module and the followings performs the real analysis, that, as we said in 1, works in two main directions: Statistical and Rule-Based.

The Statistic Module allows us to perform a kind of text analysis which is not Language-Dependent or Topic-Dependent: with this module, it is possible to analyze text in any language (without the previous application of the POS Tagger), and for every topic, without the need to know syntactic mechanisms or to construct model of the language, without the need of a linguistic background and in a short time. The statistic module does not allow to answer specific questions about elements of the text and could not give a structure to unstructured texts, but it permits to perform rapid lexical analysis which generate a generic but useful knowledge about a text or a set of texts, a Corpus.

We presented a good example of basic statistic analysis of a text in 2.4, in which we analyze a novel, “the Hobbit”, and present some basic results. In this section we present the module that allows us to perform that kind of analysis. However, the module only provides instruments that could be used in combination with other algorithms or models for more complex analysis. The module is composed by three methods:

- Frequency computing
- Term Frequency / Inverse Document Frequency calculation
- Bigrams and Trigrams extraction

First of all, the user can select how to split the text in macro sequences and then, select the algorithm he want to apply. The user has three options to split the text:

a. divide the text in equal parts by selecting the exact number of parts the text must be divided.

b. divide the text by a symbol or a sequence of characters.

c. divide the text in paragraphs, creating a text sequence for each line present in text.

With the first option, regardless of how long is the text, the user creates equilibrate subsections by computing the total number of words in the text and dividing this number for the specified number of sequences. In this way the user does not care about editing text, but simply splits the text in equal parts.
With the second option, the user selects a pattern, a character o a sequence of character that, as they are encountered in text, cause the algorithm splits the text and create a new sequence with the text before the patter. With this option the user could decide what element cause the splitting, but it has less control over the number of resulting sequences.
The last option simply divides the text for lines. If the users needs to analyze an *ad-hoc* corpus in which each line corresponds to an element, this could be the best and faster choice. If the user needs to analyze an unstructured text in which a line could be of different lengths, depending of the analysis he must perform, this option could not be the best choice and it may be need to perform some pre-modelling operations as putting text blocks in the same line, or unify chapter lines, etc.

There is a method for splitting the text connected to each one of these options: for the first one there is a *splitEqually* method that requires as arguments a int value that represents the parts in which the text should be split, and a String value that represents the text to split. Odd values are rounded down.
The second option is performed by the *splitByPatter* method which requires two Strings as arguments, one for the pattern and one for the text. The third option is provided by the *split* method that has no arguments.

In order to calculate word frequencies, the module uses a method called *tf* (*Term Frequency*) which tokenize each sequence and calculate the frequency of each word in a sequence by dividing the occurrences of that word by the total number of words present in the sequence. The algorithm is the following:

The algorithm, takes a splitted text as input and calculates frequencies for each word in each sequence. In first part of the algorithm, lines 1-5, some necessary variables are initialized.

- In line 1, a Map<Integer,Map<String,Double» *sequenceFrequency* is initialized. The Map will contains Integer values that represent the sequences, and another Map in which the algorithm insert the word as a String and the calculated value of Term Frequency as a Double.

Algorithm 5 tf method

```
Input: a List<String> splittedText
1: Map<Integer,Map<String,Double» sequenceFrequencies = new Map<Integer,Map<String,Double»
2: Map<String,Double> frequencies = new Map<String,Double>
3: int i = 0
4: for (String sequence in splittedText) do
5:     Map<String,Integer> tokenizedSequence= tokenize(sequence)
6:     int sequenceLength = tokenizedSequence.length
7:     for (String key in tokenizedSequence) do
8:         Double freq = tokenizedSequence.get(key) / sequenceLength
9:         frequencies.put(key,freq)
10:    i = i + 1
11:    sequenceFrequencies.put(i, frequencies)
Output: sequenceFrequencies
```

- In line 2, a Map<String,Double> *frequency* is initialized. It is the Map which will contains terms

In line 3 a int with value 0 is initialized in order to enumerate the sequences in which the text is divided. Two cycles (For) start from the line 4. The outer For Cycle iterate through the list of sequences that compose the text and splits it in tokens by using the *tokenize* method we present in 3. The method creates a Map of String and Integer in which it puts the words that compose the sequence and their occurrence values. The length of this Map is assigned to the int variable.
A second For Cycle starts now and for each word in the newly created Map calculates the Double value of frequency by dividing each occurrence value for the length of the sequence. Then, the algorithm insert the word and the frequency in the variable *frequencies.*
Before closing the outer For Cycle, the value of *i*, incremented by one for each sequence, and is inserted into the *sequenceFrequencies* Map together with variable *frequency.*

The tf-idf is computed for each word in a sequence by the algorithm *tfIdf* after the applying the *tf* method. The *tfIfd* method takes into account the term frequencies and the sequence. The method, first calculates the Inverse Document Frequency value (idf), then calculates the tf-idf value by multiplying the tf value of each word for the logarithm of the idf for the given word. The algorithm is shown in 4.3.

In the First part of the algorithm 4.3 some variables have been initialized.

Algorithm 6 tfIdf method

```
Input: a Map<Integer,Map<String,Double» sequenceFrequencies
Input: a List<String> splittedText
Input: a Map<String,int> tokenizedText
 1: Map<int,Map<String,Double» sequenceTfIdf = new Map<int,Map<String,Double»
 2: Map<String,Double> tfIdf = new Map<String,Double>
 3: Map<String,int> idf = new Map<String,int>
 4: {Here starts the computing of IDF value}
 5: Set<String> types = new Set<String>
 6: for (String key in tokenizedText) do
 7:     types.put(key)
 8: int value = 0
 9: for (String word in types) do
10:     for (String sequence in splittedText) do
11:         if sequence.contains(word) then
12:             value = value + 1
13:     idf.put(word,value)
14: {Here starts the computing of TF/IDF value}
15: for (String map in sequenceFrequency) do
16:     for (String key in map) do
17:         Double value = map.get(key) * Log10(idf.get(key))
18:         tfIdf.put(key,value)
19:     sequenceTfIdf.put(sequenceFrequency.get(map), tfIdf))
Output: sequenceTfIdf
```

- In line 1, a Map<Integer,Map<String,Double» *sequenceTfIdf* is initialized. The Map will contains Integer values that represent the sequences, and another Map in which the algorithm insert the word as a String and the calculated value of tf-idf as a Double.
- In line 2, a Map<String,Double> *tfIdf* is initialized. It is the Map which will contains terms and relative tf-idf values of a single sequence.
- In line 3, a new Map<String,Double> *idf* is created in order to store idf values for each word of the whole text.

From line 5 to line 7 the text is tokenized and the types are extracted.
Subsequently (lines 8-13), we calculated the idf values: for each type a int *value* counts the number of sequences in which the type appears and put the type and the value in the *idf* variable.
At the line 15, the calculation of the tf-idf starts: iterating through Term

Frequency Map, for each sentence the method calculates the tf-idf by multiplying the term frequency of the word by the logarithm in base 10 of its idf. The Logarithm has been calculate by an external method belonging to the standard Java package *Math*, called *Log10*.

The word and the resulting tf-idf value are inserted into the *tfIdf* variable and then, each *tfIdf* variable is inserted into the variable *sequenceTfIdf* and returned as result.

In addiction to these standard measures, the Statistic Module provides a method to extract *N-grams* and calculate its occurrences. N-grams are sequence of N number of consecutive words that are repeating into texts. N-grams could help in detect recursive structures which can be useful for text categorization or topic detection. The algorithm is the following:

In the first part (lines 1-8) each N-gram is extracted and inserted in two Lists, one for bi-grams and one for tri-grams. Iterating the list of tokens, the algorithm extract the current tokens and the following (or the following two in the tri-grams).
Once filled the lists, both for bi-grams as for trigrams, the algorithm create a set of n-grams deleting repeated elements and uses this set in order to calculate the occurrences of each bi-gram and tri-gram respectively.

3.2.4 Semantic Module

In Natural Language Processing Semantic technologies have increased its popularity in last years. With the birth of the semantic web and the growth of popularity of this kind of technologies (semantic networks, ontologies, etc.) Semantics become one of most studied branches of Linguistics and text analysis. But, as we saw in 2.5, studying the relation between signifier and its meaning necessarily implies studying a theory about the human knowledge process: in order to understand how meaning is related with symbols we need to understand how human mental mechanisms process new information and create a link between the real sphere and the conceptual or symbolic sphere.

In our work we try to offer to users an instrument to perform a first, simple and automatic study of Semantics in texts. To do this we rely on the Hyperspace Analogue to Language algorithm, which theorizes that the meaning of a word is given by a list of other words which occurs in similar context than it. In order to find these words, the algorithm processes a large corpus of texts and create a co-occurrence matrix by assigning to each word a score by distance from the original word.
Applying mathematical algorithms to word's vectors it is possible to calculate semantic distance between words or construct a Cartesian plane of the words.

Algorithm 7 ngrams method

```
Input: a String text
 1: Map<String,int> bigrams= new Map<String,int>
 2: Map<String,int> trigrams= new Map<String,int>
 3: List<String> biG = new Set<String>
 4: List<String> triG = new Set<String>
 5: List<String> tokenizedText = tokenize(text)
 6: for (String token in tokenizedText) do
 7:    biG.put(token + token+1)
 8:    triG.put(token + token+1 + token+2)
 9: {Here starts the calculation of bigrams}
10: Set<String> bigramSet = biG.toSet()
11: int value = 0
12: for (String bigram in bigramSet) do
13:    for (String bi in biG) do
14:       if bigram.equals(bi) then
15:          value = value + 1
16:    bigrams.put(bigram,value)
17: {Here starts the calculation of trigrams}
18: Set<String> trigramSet = triG.toSet()
19: int value = 0
20: for (String trigram in trigramSet) do
21:    for (String tri in triG) do
22:       if trigram.equals(tri) then
23:          value = value + 1
24:    trigrams.put(trigram,value)
Output: bigrams
Output: trigrams
```

The Semantic Module is based on a large co-occurrence matrix that has been previously calculated on a large corpus. In order to calculate the Matrix we used the S-Space Package, a collection of algorithms for building Semantic Spaces written in Java, which has been developed by the Natural Language Processing Group at UCLA[5].
The corpus on which we launch the algorithm was a lemmatized version of about 45 million of words of the "Paisà Italian Corpus". As required by the algorithm we needed to set some options:

- the word window has been set on 10 context words;

[5]the entire project can be downloaded at `https://github.com/fozziethebeat/S-Space`

- as weighting function we selected the Linear Weighting Function which assigns 10 points to the next word, 9 to the following etc.;
- the maximum value for column was set on 200 elements in order to limit the memory used by the algorithm during the generation of the matrix.

The resulting file was a 128 MB .sspace file which contains the matrix.
The Semantic Module uses this file to extract from it the word's vectors and find similarity between words.

In our module there are three main components:

- Extraction of word's vectors and reduction of vectors in coordinates.
- Extraction of similarity between two words.
- Extraction of most similar word of a given word.

The first component extracts a vector for each selected word and converts the vector into a two dimensions Scaled Double Vector. Dimensions can be considered as coordinates of a point in a Cartesian Space. In this way each word is represented by a point and it is possible to graphically represent the word or to launch clustering algorithms.
In order to generate the points the *getPoints* method must be called with a String word as argument. The method returns a List of coordinates. By launching iteratively the method it is possible to create a set of points and put them into a bidimensional space.

The other two components use a pre-compiled algorithm, provided by the S-Space Package called Semantic Space Explorer, which is included into the executable jar file *sse.jar*. This algorithm permits many operation on the semantic space. Available commands are the following:

- load a file.sspace
- unload a file.sspace
- get neighbors of a given word (specifying the number of neighbors and the similarity measure)
- get similarity between two words using a specified similarity measure
- compare the vector of the same word in different semantic spaces
- get words or word's vector

- describe semantic space in terms of column and row dimensions

The Semantic Module only uses the load function, the get-neighbors function and the get-similarity function. For both functions we used the Euclidean distance instead of the standard Cosine Similarity. The number of neighbors the algorithm extract can be specified each time the method is called.
In order to extract the neighbors of a given word, the method *getNeighbors* requires two arguments, the word and the number of neighbors to calculate. The results is a List of Strings and Double which represents the neighbor word and the similarity value.
The method *getSimilarity* requires two Strings as arguments (i.e. the two words) and returns a Double value that represents the similarity between the two words.

3.2.5 Syntactic Module

The Syntactic Module belongs to a bigger project of description of the language based on the Lexicon-Grammar Theory described in 2.7. A large number of resources has been created in last years according to the LG theory and, many of these resources are not ready for an automatic usage. While dictionaries were easily converted into a machine-readable resources and they were used for the construction of the Lemmatizer and the Postagger (3.2.2), syntactic resources must be converted. The most important resource for the construction of a syntactic parser based on the Lexicon-Grammar Theory was the LG Tables of Predicates in which are described the structures of the main verbs of Italian Language.

The idea of this module is to use LG-Tables as baseline for the construction of a Lexicon-Grammar Dependency Parser able to automatically detect the structure of elementary sentences.

A dependecy parser based on Lexicon-Grammar as the one we aspire to built implies some difference compared to a dependency parser based on the Generative Grammar. In LG Theory, as in the Operator Grammar, differently from the generative grammar, sentences are not supposed to have a bipartite structure.

$$\mathrm{S} \rightarrow NP\ +\ VP$$

$$VP \rightarrow V\ +\ NP[6]$$

[6]Sentence = Noun Phrase + Verb Phrase; Verb Phrase = Verb + Noun Phrase.

but are represented as *predicative functions* in which a central element, the *operator*, influences the organization of its variables, the *arguments.* The role of operator must not be represented necessarily by verbs, but also by other lexical items that possess a predicative function, such as nous or adjectives.

(5) Maria odia il fumo
"Maria hates the smoke"

(6) Il fumo tormenta Maria
"The smoke harasses Maria"

If we observe the sentences 5 and 6, the semantic role of subject or *experiencer* is played by the noun Maria, while the semantic role of cause or *stimulus* is played by *il fumo*, "the smoke". It is intuitive that, from a syntactic point of view, these semantic roles are differently distributed into the *direct* (the experiencer is the subject, such as in 5 and the *reverse* (the experiencer is the object, as in 6) sentence structures.
With a bipartite structure represents the *experiencer* "Maria" inside 6 and outside 6 the verb phrase, and unreasonably brings it closer or further from the predicate, to which it should be attached just in the same way (Figure 3.2).
Moreover, it does not explain how the verbs of 5 and 6 can exercise distributional constraints on the noun phrase indicating the *experiecer* (that must be human, or at least animated) both in the case in which it is under the dependence of the same verb phrase (experiencer object, see 6a) and also when it is not (experiencer subject, see 5a):

(5a) $[_{Sentiment}[_{Experiencer}\begin{bmatrix} Maria \\ \textit{Il mio gatto} \\ *\textit{Il televisore} \end{bmatrix}]odia[_{Stimulus}il fumo]]$
"(Maria + My cat + *The television) hates the smoke"

(6a) $[_{Sentiment}[_{Stimulus}Il fumo]tormenta[_{Experiencer}\begin{bmatrix} Maria \\ \textit{Il mio gatto} \\ *\textit{Il televisore} \end{bmatrix}]]$
"The smoke harasses (Maria + My cat + *The television)"

According with [Gross, 1979, p. 872] the problem is that

> "rewriting rules (e.g. S $\rightarrow$ $NP\ VP$, $VP \rightarrow V\ NP$) describe only LOCAL dependencies. [...] But there are numerous syntactic situations that involve non-local constraints".

While the Lexicon-Grammar and the operator grammar representations do not change, the representations of the paraphrases of (5, 6) with support

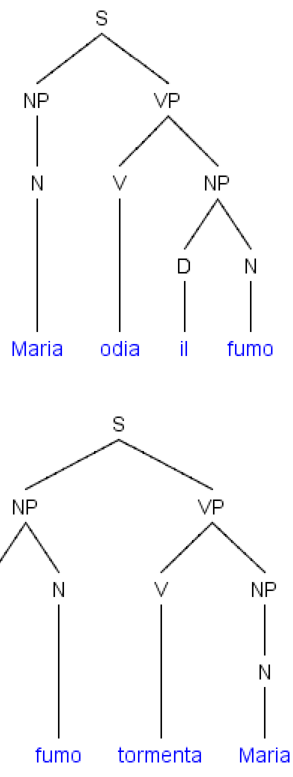


Figure 3.2: Syntactic Trees of the sentences 5 and 6

verbs (5b,6b) further complicate the generative grammar solution, that still does not correctly represent the relationships among the operators and the arguments (see Figure 3.2). It fails to determine the correct number of arguments (that are still two and not three) and does not recognize the predicative function, which is played by the nouns *odio* "hate" and *tormento* "harassment" and not by the verbs *provare* "to feel" and *dare* "to give".

(5b) $[_{Sentence}[_{Experiencer}$*Maria*$]$*prova odio per*$[_{Stimulus}$*il fumo*$]]$
"Maria feels hate for the smoke"

(6b) $[_{Sentence}[_{Stimulus}$*Il fumo*$]$*dà il tormento a*$[_{Experiencer}$*Maria*$]]$
"The smoke gives harassment to Maria"

As we reject the bipartite sentence structure proposed by Generative Grammar, the dependency parser must consider as anchor for determining the graph sentence structure the predicate which is not only identifies with the verb, but with all possible sentence paraphrases as support verbs sentences, passive sentences, etc.
To do this, a powerful resource as the LG Tables of predicates, which contains

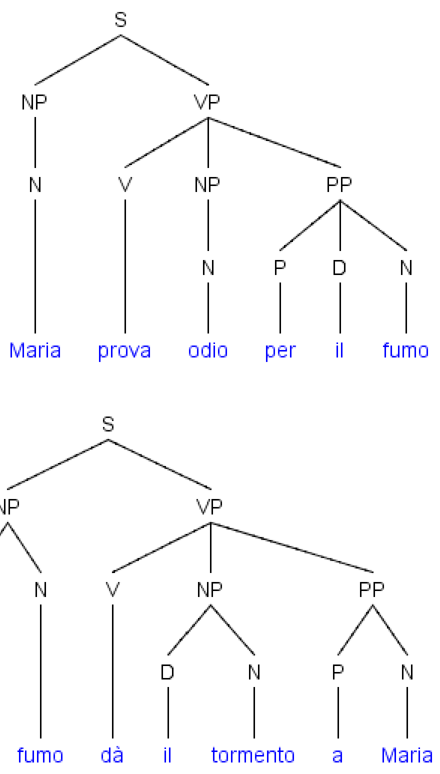


Figure 3.3: Syntactic Trees of the sentences (5b) and (6b)

all accepted structures of Italian verbs, its nominalization, adjectivalization and distributional properties, is needed.

At the moment, a machine-readable version of the tables, generically known as *Lexicon-Grammar Verbs Resource*, is currently in development at the Department of Political, Social and Communication Science of the University of Salerno. The objective is to make a powerful theoretical resource as LG-Tables, now available at the url `http://dsc.unisa.it/composti/tavole/combo/tavole.asp`, compatible with popular programming languages as Java and Python in order to make possible their use in NLP automatic application as Parsers.

In the new version of LG-Tables the tabular format is replaced by a hierarchically structured format built in Json[7] format. Json (JavaScript Object Notation) is a lightweight data-interchange format. The syntax of Json file is easy to read and write for human and allows machines to easily work with it. Json files are built on two structures: a collection of name/values pairs and

[7]`http://www.json.org/`

a ordered list of values. These universal data structures, which, one, can be associated with objects, dictionaries, maps, and the other with arrays, vectors or lists, can be supported by all modern programming languages and make it possible to interchange data with the majority of programming languages.
In Json format, an object is an unordered set of name/values pairs which begins with a *{* (left brace) and ends with a *}* (right brace). Names are followed by a *:* and pairs are separated by *,* (commas). In addition, a name can be associated with a list of pairs, embedding substructures in a single pair. An example of Json file relative to the Lexicon-Grammar Verbs Resource under construction is shown in 8.

Algorithm 8 Verb Hierarchy example

```
“Class 1”: {
  “@id”: 1,
  “description”:”...”
  “marcato”: “true|false”,
  “pron”: true,
  “aux”: “avere”,
  “verbi”:{
    “@id”: “rompere”,
    “struttura base”: “N0 V N1”,
    “struttura2”: “N1 si V”,
    “struttura3”: “N0 V N1 Prep N2”,
    “struttura4”: “N0 Vsup V-n Prep N1”,
    “N0”: “Num”,
    “N1”: “Npc”,
    “V-n”: “rottura”,
    ...
  }
}
```

The structure first defines the class of verbs it describes, showing the numerical ID, a short description (i.e. semantic description as Psychological Verbs) and some features as auxiliary verbs or the pronominal form, then lists all the verbs belonging to the class: each verb is described with an ID which corresponds to the lemma, the base structure and all accepted structures, the semantic restriction of the Nominal groups.

Structures could be described as a list of elements: nominal groups are listed as N followed by a number starting from 0; the verb is identified by the V; *Prep* identify the preposition. Other structures can includes the impersonal sentence (i.e. the impersonal verb form is represented by the

label *si V*), the presence of a non-essential complement (*Prep N2*), the *support verb construction*, which includes a support verb *Vsup* followed by a nominalization (i.e. *V-n* that in the example represents the noun *rottura*, "breaking", derived from the verb *rompere*, "to break".
In some cases, the structure can specify a semantic feature for each N, as for example *Num*, which indicate a Human noun or *Npc* which indicate a body part.
For the complete list of features and labels see Elia et al. [1981], Elia [1984], the list of noun tagset is shown in table 3.5.
Verbs structures also includes the definition of the V-n or the V-a.
In fig 3.4 is shown the graphical representation of the structure hierarchy of the Json. In the figure is represented the structure of the same verb of the example in 8, *rompere*.

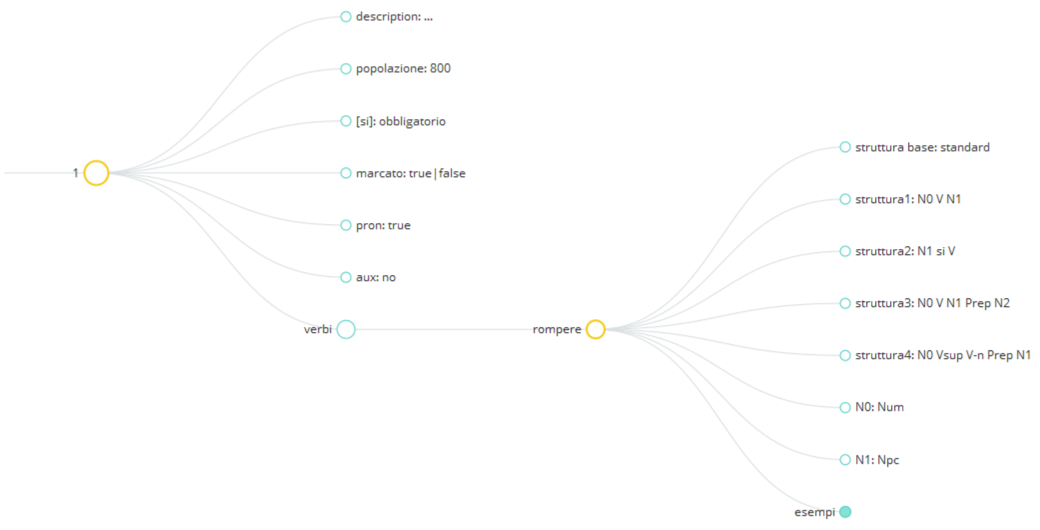


Figure 3.4: Graphical representation of the Json Hierarchy of Verbs

With the database of verbs in phase of development, the module only works with the verbs already inside the resource. At the moment, the verbs included into the Lexicon-Grammar Verbs Resource belong to the class of Psychological Predicates which includes Sentiment Verbs among the classes 41, 42, 43 and 43B [Elia, 2014, 2013] as shown in table 3.4.

The verbs from LG class 41 have as definitional structure *N0 V N1*, in which:

- N0 is generally an infinitive or completive clause (direct or introduced

Psych Verb	**Translation**	**LG Class**
angosciare	“to anguish"	41
piacere	“to like"	42
amare	“to love"	43
biasimare	“to blame"	43B

Table 3.4: Examples from the Psych Predicates dictionary

by the phrase *il fatto (Ch F + di)* “the fact (that S + of)”; but, with few exceptions, it can be also a human, a concrete or an abstract noun;

- N1 is a human noun (*Num*).

The verbs from the class 42 enter into an intransitive reverse psych-verb structure and are defined by the LG formula *N0 V Prep N1*, where:

- N0 is generally an infinitive or completive clause (direct or introduced by the phrase *il fatto (Ch F + di)* “the fact (that S + of)”) but in few cases can be also a human (*Num*) or a non restricted noun (*Nnr*);
- N1 is a human noun (*Num*), but can be also a body part or an abstract noun of the kind *discorso* “discourse”, *mente* “mind”, *memoria* “memory” [Elia, 1984].

The sentence structure that defines the class 43 and 43B is *N0 V N1*

- N0, in both the classes 43 and 43B, is generally a human noun (*Num*)
- N1 is an infinitive clause or a completive one, which is direct for the 43 and introduced by *il fatto (Ch F + di)* “the fact (that S + of)”) for the 43B.

The algorithm of the method which perform the syntactic analysis takes as input the database of verbs and the Lemmatized text. The required format of the text is a double list of strings with the following format:

Algorithm 9 Format of String[][] dictionary of the sentence *la casa è bella*, “the house is nice”

```
String[][] dictionary = new String {
    {“la”, “il”, “DET”, “f+s”},
    {“case”, “casa”, “N”, “f+s”},
    {“è”, “essere”, “V”, “aux+3+s+PR”},
    {“bella”, “bello”, “A”, “f+s”}
}
```

In this *String[][]*, each element of the list is a list in which:

- the element 0 represents the Token;
- the element 1 represents the Lemma;
- the element 2 represents the POS in a format corresponding to the tagset presented in 3.2.2;
- and the element 3 contains a series of syntactic and semantic information about word.

The element 3 contains information about the genre and the number of the token (*f* for female, *m* for male, *s* for singular, *p* for plural, etc.) and information about the verbe tense: a number from 1 to 3 which indicate the person of the verb, and a symbol for the tense as indicate in the following list:

- Infinitive = INF
- Gerund = G
- Present Participle = PPR
- Past Participle = PP
- Present Simple = PR
- Imperfect = IM
- Past = PA
- Future = F
- Imperative = IMP
- Present Subjunctive = S
- Past Subjunctive = IS
- Conditional = C

In addition, the last part can contain other syntactical information such as *aux* for the auxiliary verbs *essere* “to be” or *avere* “to have”, or semantic information as *Astr* for abstract nouns or *Conc* for concrete nouns. Concrete nouns are semantically sub-classified as showed in table 3.5.

Tag Set	**Description**	**Examples**
Npc	body parts	harm, hand, leg
Npcorg	organism parts	lung, neuron, cell, heart, liver
Ntesti	texts	book, but also softwares,
		declarations or oral texts, paintings etc.
Nindu	clothes	jacket, jeans, pants, trousers, shirt
Ncos	cosmetics	fard, eyeliner
Ncibo	food	bread, meat, chicken, ham
Nliq	liquids	gasoline, diesel, alcohol
NliqBev	drinkable liquids	water, cocacola, juice
Nmon	money	cents, Euro, Dollar
Nedi	building	house, building, palace, street,
		avenue, column, window, door
Nloc	location	mountain, street, place, square,
		region, county, country
Nmat	materials	water, wood, iron, wool, silk,
		proton, electron, quark, snow
Nbot	botanics	tree, leaf, plant, sequoia, willow
Nfarm	pharmacy	aspirin, ibuprofen
Ndrugs	drugs	cigarettes, cocaine, heroine
Nchim	chemistry	oxygen, hydrogen, carbon, citric acid
Ndisp	electronic devices	cellular phone, pc, computer, tablet,
		television, vacuum cleaner, washing machine
Nstrum	instruments	fork, spoon, knife, guitar, stick, brush
Nvei	vehicles	car, camion, train, airplane,
		and parts of vehicles as motor, silencer, wheel, cloche
Narr	furnishings	wardrobe, drawers, table, chair
Anl	animals	lion, cat, dog
Um	human	policeman, engineer, lawyer, medic
UmColl	human collective	school, industry, institute
ConcColl	concrete collective	peach orchard, orangery, myriad

Table 3.5: Tag Set for Concrete and Human Nouns

The tagset of Concrete nouns can interact with the information about arguments of predicates included into the LG Verbs Resource. As shown in 8, line 13 and 14, is specified that the arguments *N0* and *N1* can belong to the class *Num* Human nouns and *Npc* Body Parts nouns of the Concrete Nouns. Other tags, for example, can restrict the field of the object of the verb *mangiare*, "to eat" at the class of concrete nouns of foot. In this way, inside the predicate structure which specify the dependency structure of the parser, this restriction about the semantic class of complements allows to distinguish arguments from unnecessary complements (i.e. *mangiare una pizza* "to eat a pizza", and *mangiare al ristorante* to eat at the restaurant") or from complements which depend from the noun (i.e. *mangiare una pizza coi funghi* "to eat a pizza with mushrooms").

As we already said, actually the Syntactic Module works only with the predicates which belong to the class of Psychological Verbs. In the first stage, the parser found a verb of this class, listed into a *Map<String,String>* in which the key is the verb's lemma and the object is a Strings that contain all the information derived from the Json separated by a *;* (semicolon).
A Java method called *shareVerbStructures(Map<String,String> verb)* takes the information collected into the Map and convert it into instructions about how to create the structure. For each feature of the verb, the method define how to extract from the text and tag the syntactic dependencies.

If, for example, the parser found the verb *desiderare*, "to desire", the method *shareVerbStructures(Map<String,String> verb)* convert the String in which are listed the features of the verb into a list. For the verb *desiderare*, belonging to the class 43, as described in Elia [1984], the list is the following:

- N0 =: Num (N0 must be a Human Noun)
  (Maria + il ragazzo) desidera una pizza
  "(Maria + the boy) desires a pizza"

- N1 =: V0 Ω (N1 could be an infinitive with its complements)
  Maria desidera mangiare una pizza
  "Maria desires to eat a pizza"

- N1 =: Che Fcong (N1 could be a completive with subjunctive verb introduced by *che*, "that")
  Maria desidera che tu mangi una pizza
  "Maria desires that you eat a pizza"

- N1 =: Fcong (N1 could be a completive with subjunctive verb)
  Maria desidera tu mangi una pizza
  "Maria desires you eat a pizza"

- ciò (N1 could be a Pronoun)
  Maria desidera ciò
  "Maria desires this"

- Ppv=: lo (N1 could be a preverbal particle)
  Maria lo desidera
  "Maria desires it"

- N1 =: Num (N1 could be a Human Noun)
  Maria desidera Antonio
  "Maria desires Antonio"

- N1 =: N-um (N1 couldn't be a Human Noun)
  Maria desidera una macchina
  "Maria desires a car"

- N1 di Num (structure accept a human N2 introduced by a Prep *di*, "by')
  Maria desidera di Antonio che abbia un buon comportamento
  "Maria desires by Antonio that he behaves well"

- N1 da N2 (structure accept a generic N2 introduced by a Prep *da*, "by")
  Maria desidera da Antonio che si comporti bene
  "Maria desires by Antonio to behave well"

- Passiva (the passive structure is accepted)
  questa soluzione è desiderata da tutti
  "this solution is desired by all"

In addition to the features listed in the LG table, the Json also includes the following informations which complete the description of the verb:

- aux=: avere

- Vsup=: avere V-n

- Vsup=: essere V-a

- V-n=: desiderio

- V-a=: desideroso

- Str1=: N0 V N1

Once the features have been listed into the method, the algorithm proceeds in this way:

Algorithm 10 shareVerbStructures

```
Input: Map<String,String> verb
Input: String[][] dictionary
 1: Map<String,String[]> sentenceStructures = new Map<String,String[]>()
 2: String[] structure = new String[]
 3: String[] features = verb.split(";")
 4: for (String element in features) do
 5:    if (element.equals("N0=:Num") then
 6:       for (int i from 0 to dictionary.length()) do
 7:          if (dictionary[i][0].equals(verb)) then
 8:             if (dictionary[i-1][2].equals("N")) then
 9:                if dictionary[i-1][3].contains("Num") then
10:                   N0 = dictionary[i-1][0]
11:                if (dictionary[i-1][2].equals("DET")) then
12:                   DET0 = dictionary[i-1][0]
13: ...
14: structure = DET0,N0,V,DET1,N1,...
15: sentenceStructures.put(originalText, structure)
Output: sentenceStructures
```

The algorithm iterate over the feature list (line 4): if an element corresponds to some pre-defined feature of the algorithm (line 5), the construction of that structure begins.
The algorithm starts from the anchor word (in this case the verb *desiderare*) and proceeds to the right or to the left depending which element we are found for. In the current example we are searching for N0, which normally is on the left.
If the verb is at element *i* of the dictionary, the element which proceeds the verb is the element *i-1*.
If this element is a Noun and contains the Num Tag (lines 8 and 9), then the algorithm write it into the N0 String variable (line 10).
The algorithm continue with the same procedure with the next element which may be a *DET* (lines 11-12).

When all features have been processed and the entire structure is supposed to be defined, the algorithm put all elements into a List of Strings (line 14) and then into a Map together with the original text of the sentence (line 15). This Map with the original texts of the sentences as keys and their structure as a List of Strings is the output of the algorithm.

The output is processed by other method called *extractDependency(Map<String,String[]>)* which rewrite the sentences in the following format:

(7) *Maria desidera una pizza*
"Maria desires a pizza"

Listing 3.1: Structure of the sentence "Maria desidera una pizza"

```
(F
   (PRED
      (desidera)
      (ARG
         (N0
            (Maria)))
      (ARG
         (N1
            (pizza)
            (DET
               (una))))));
```

In this way the Syntactic Module construct a dependency structure based only on the arguments of the predicate, on its essential structure of complements, but which not includes other not-necessary complements:

(8) *Maria desidera una pizza coi funghi*
"Maria desires a pizza with mushrooms"

Listing 3.2: Structure of the sentence "Maria desidera una pizza coi funghi"

```
(F
   (PRED
      (desidera)
      (ARG
         (N0
            (Maria)))
      (ARG
         (N1
            (pizza)
               (COMP
                  (N2
                     (funghi)
                     (DET
                        (i)))
            (PREP
               (con)))
      (DET
         (una))))));
```

In 8 the predicate is *desidera*, "desires", with two arguments, the subject, identified as N0, and the object identified as N1. In the predicate structures there is no mention of other complements because they are not part of the essential sentence structure selected by the verb. The complement *con i funghi* "with mushrooms", which consist on a preposition and N2 is not included into the arguments of the verb, but it neither is a not essential complement of the verb, because it depends directly by the N1, *pizza*.

Distinguish between verb complements and noun complements is not a trivial task. If we take as exemple the following sentence:

(9) Maria eats a pizza with Max

(10) Maria eats a pizza with mushrooms

"with Max" and "with mushrooms" are both introduced by the same preposition and the only distinction between the two name is that "Max" is a Human Noun and "mushrooms" is a Foot Noun. The Syntactic Module will not be able to distinguish between the complement that depends by the verb (9) and the complement that depends by the noun (10) and will tag both as unnecessary verb complements.

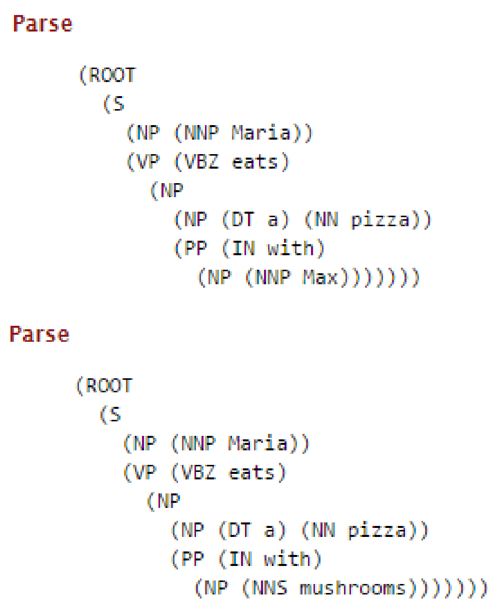
Many free on-line parser for Italian or English languages, fail on the task of distinguish between verb's complements and noun complements. We decided to parse these two sentences using the popular on-line Stanford Parser[8].

As shown in the fig. 3.5, in which are presented the results of the application of the Stanford Parser on the sentences (9) and (10), sentence (9) dependency structure treat the NP "with Max", as it depends directly on the NP "a pizza". While the same structure is correct for the sentence (10) in which the NP "mushrooms" depends on "a pizza" because it represents a specification of what is on the pizza, in (9) "Max" is not an ingredient, but depends on the verb "eat".

The Italian parser *LingA*[9] make the opposite error on the following Italian sentences:

[8]`http://nlp.stanford.edu:8080/parser/index.jsp`

[9]`http://linguistic-annotation-tool.italianlp.it/`



```
Parse

    (ROOT
      (S
        (NP (NNP Maria))
        (VP (VBZ eats)
          (NP
            (NP (DT a) (NN pizza))
            (PP (IN with)
              (NP (NNP Max)))))))

Parse

    (ROOT
      (S
        (NP (NNP Maria))
        (VP (VBZ eats)
          (NP
            (NP (DT a) (NN pizza))
            (PP (IN with)
              (NP (NNS mushrooms)))))))
```

Figure 3.5: Depenency Structure of the sentence (9) and (10) performed by Stanford Parser

(11) Maria mangia un gelato al bar
"Maria eats an ice cream at the bar"

(12) Maria mangia un gelato al limone
"Maria eats a lemon ice cream"

LinguA parser, as shown in fig 3.6, correctly analyze the sentence (11), in which the complement *al bar*, "at the bar" is a Locative Complement and depends on the verb *mangia*, "eats", but fails the parsing of the sentence (12) in which the complement *al limone*, literally "with lemon taste", depends on the noun *gelato* because specify the taste of the ice cream. The same error is done by LinguA also on the italian translation of sentence (10), but only if the preposition is written as a prepositional article *coi* "with the" and not in the reverse case.

In order to avoid this problem we have to add to the features of the predicate some additional features about not-essential complements: returning to the example (7), it is easy to imagine that in Italian, a complement introduced by the preposition "con" is not accepted by the verb *desidera* as complement if the noun is a Num:

(13) *Maria desidera una pizza con Antonio
"Maria desires a pizza with Antonio"

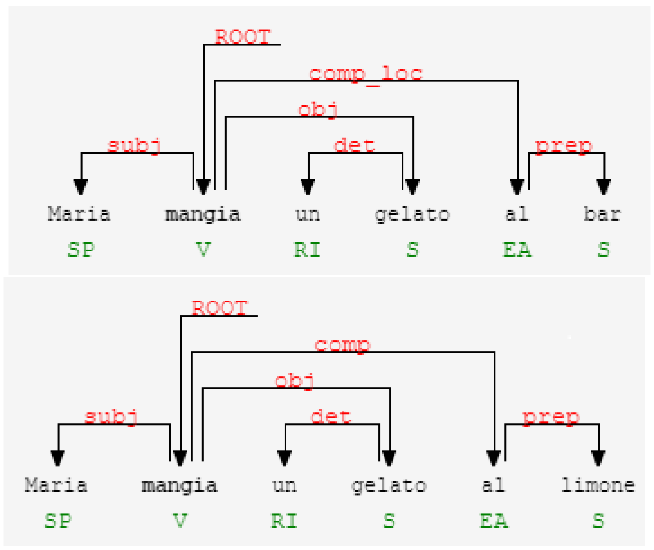


Figure 3.6: Depenency Structure of the sentence (11) and (12) performed by LinguA

in 13 the complement *con Antonio* is not accepted as in sentence (14):

(14) Maria desidera mangiare una pizza con Antonio
“Maria desires to eat a pizza with Antonio”

Obviously, the error is always lurking, and it is possible that a verb accept complements introduced by a preposition which also introduce a noun complement as happens with 11 and 12. The solution could be an hybrid model which take into account statistical distribution of words sharing the probability that a complement depends from the Noun or from the Verb directly from a repository of annotated sentences as the “Paisà Corpus”.

3.3 LG-Starship Interface

In this section we present a version of the modules presented in the previous section provided with a graphical interfaces, which permits to user not familiarized with programming languages to operate with the methods of the LG-Starship Framework.

The interface has been built following the guidelines in Shneiderman [2010]. In particular, in the book, eight golden rules of interface design, derived from experience and refined over two decades are presented:

1. Strive of consistency: Consistent sequences of actions should be required in similar situation, identical terminology should be used in prompts, menus, and help screens, and consistent color, layout, fonts should be employed through-out.

2. Cater to universal usability: recognize the needs of diverse users and design for plasticity, facilitating transformation of content. Adding features for novices, such as explanations, and features for experts, such as shortcuts and faster pacing, can enrich the interface design and improve perceived system quality.

3. Offer informative feedback: For every user action, there should be system feedback.

4. Design dialogs to yield closure: sequences of actions should be organized into groups with a beginning, middle and and.

5. Prevent errors: As much as possible, design the system such that users cannot make serious errors. If an user make an error, the interface should detect the error and offer simple, and specific instructions for recovery.

6. Permit easy reversal of actions: as much as possible, actions should be reversible.

7. Support internal locus of control: Experienced operators strongly desire the sense that they are in charge of the interface and that the interface responds to their actions.

8. Reduce short-term memory load: the limitations of human information processing in short-term memory requires that displays be kept simple, multiple-page displays be consolidated, window-motion frequency be reduced and sufficient training time be allotted for codes, mnemonics and sequences of actions.

These underlying principles provide a good starting point for desktop designers and are focused on increasing the productivity of users by providing simplified data-entry procedures. In our project, we try to present the commands in a simple and linear way, with a clear pipeline of instruction, determined, in part, by the sequence of the modules that compose the interface.

Currently, the LG-Starship Framework Interface is built in a modular way and includes six tabs, each one with a set of specific commands and information panels.

The start tab is showed in fig. 3.7. With this panel the user can write a text or charge an external one. The “calculate” button perform a simple tokenization and the computing of the types and letters presents into the text.

Two radio-buttons allow the user to select how to split the original text in text units and other two radio-buttons permit to select cleaning options as the simple cleaning of punctuation or the removal of stopwords. A text-panel on the right presents some basic information about the length of the text, its composition and its path; under this panel there are two tables in which the statistic about types and letters are presented when the calculate button is clicked.

The second tab contains the Mr.Ling module allows users to perform POS Tag and Lemmatization of the text treated in the first tab. The *Pos Tag Button* and the two radio button connected with the Pos Tag task are activated only once from the *File Menu* the user charge the dictionary resources by clicking the *Carica Dizionari* button.
In the fig. 3.8 is showed a detail of the tab 2 in which is present the *Pos Tag Button* on the left, followed by two radio button by which the user can select how to see the results of the Pos Tag task (as a concatenation of $<Token, Tag>$ or as a simple concatenation of Tags separated by whitespace). Below there is the *Lemmatization Button* followed by three radio button corresponding with the three visualization option. The *Lemmatization Button* will be blocked until the Pos Tag has not been launched and has terminated the process. On the right, the *Export Text Button* permits to export the result of the Pos Tag or the Lemmatization showed in a text panel above, as a .txt file in order to use it with other external resources.

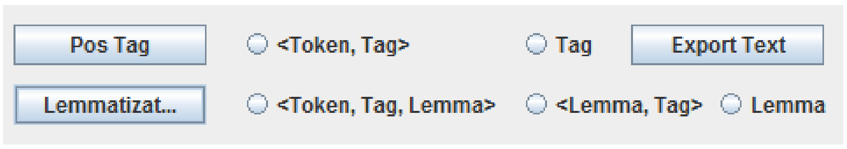


Figure 3.8: Pos Tag & Lemmatization tab of LG-S Interface

On the right of the Pos Tag and Lemmatization tab there are two tables in which, once the Pos Tag process and the Lemmatization process has finished their execution, statistics about Lemmas occurrences or Tags occurrences are presented and could be exported as .csv files for external analysis.

As an example of use of this two starting panels, we charge into the *Original Text tab* the electronic Italian version of the book “The Hobbit”, from J.R.R.

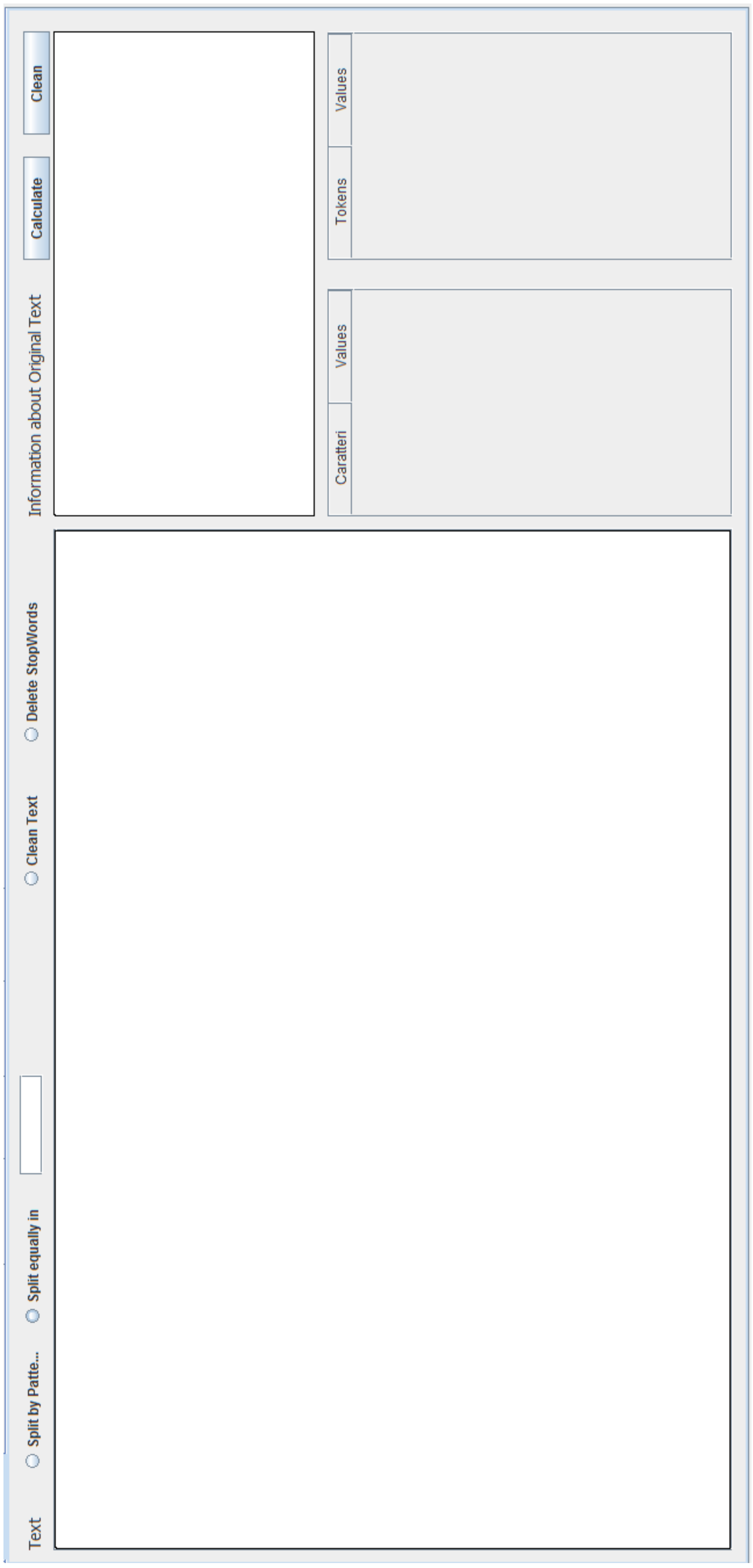
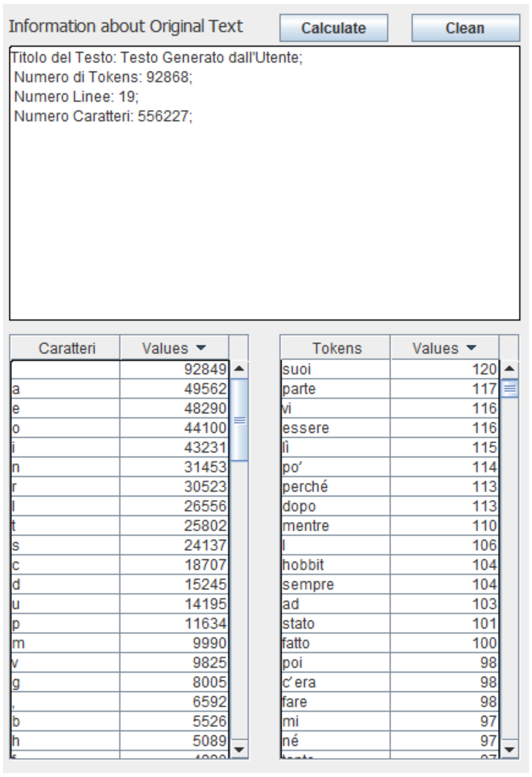


Figure 3.7: Original Text tab of LG-S Interface

Tolkien, modified so that each chapter represents one line of text.
The first operation is to select the Split option: a user can select a pattern writing it into the text panel at the side of the two appropriate buttons, or to split the text in an exact number of text units by writing this number into the same text panel.
In the text box on the right, after clicking on *Calculate*, the software displays the number of tokens found, the number of lines and the total number of characters. Below, the two tables of number of occurrences of characters and tokens are compiled as shown in fig. 3.9.



Titolo del Testo: Testo Generato dall'Utente;
Numero di Tokens: 92868;
Numero Linee: 19;
Numero Caratteri: 556227;

Caratteri	Values
	92849
a	49562
e	48290
o	44100
i	43231
n	31453
r	30523
l	26556
t	25802
s	24137
c	18707
d	15245
u	14195
p	11634
m	9990
v	9825
g	8005
,	6592
b	5526
h	5089

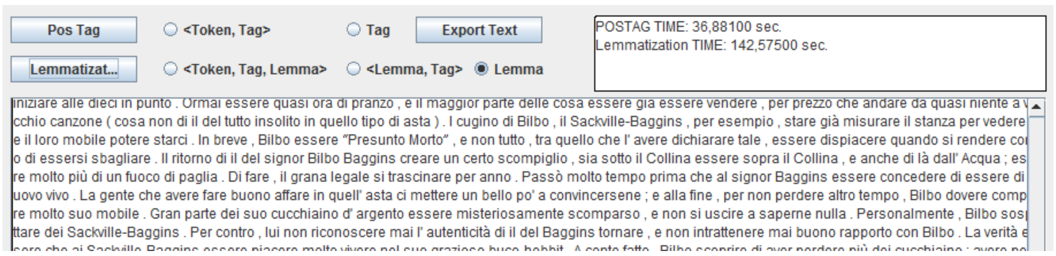
Tokens	Values
suoi	120
parte	117
vi	116
essere	116
lì	115
po'	114
perché	113
dopo	113
mentre	110
l	106
hobbit	104
sempre	104
ad	103
stato	101
fatto	100
poi	98
c' era	98
fare	98
mi	97
né	97

Figure 3.9: An example of results of *Calculate* button

Once calculated the statistics of text in terms of occurrences, performed pre-processing phase by cleaning punctuation or stop-words, we activated the *Pos Tag & Lemmatization* tab by charging lexical resources.

When the *Pos Tag Button* has been clicked, the software processes the text and, when terminates, the Pos Tagged text appears in the text panel below in the fashion the user have selected before start the Pos Tagging. On the text panel on the right, the software illustrates the time the PosTagger employs to perform the task.
At the same time, on the right of the second panel, the first of two tables,

which reports two columns, one for the tags and one for the values, is automatically compiled with each tag present in the text and its value of occurrence.
Subsequently, the *Lemmatization Button* become active. The same procedure give the software to show the lemmatized text and a message regarding the time it spends for the process. In fig. 3.10 the results of the Lemmatization task is showed.



```
POSTAG TIME: 36,88100 sec.
Lemmatization TIME: 142,57500 sec.
```

```
iniziare alle dieci in punto . Ormai essere quasi ora di pranzo , e il maggior parte delle cosa essere già essere vendere , per prezzo che andare da quasi niente a v
cchio canzone ( cosa non di il del tutto insolito in quello tipo di asta ) . I cugino di Bilbo , il Sackville-Baggins , per esempio , stare già misurare il stanza per vedere
e il loro mobile potere starci . In breve , Bilbo essere "Presunto Morto" , e non tutto , tra quello che l' avere dichiarare tale , essere dispiacere quando si rendere co
o di essersi sbagliare . Il ritorno di il del signor Bilbo Baggins creare un certo scompiglio , sia sotto il Collina essere sopra il Collina , e anche di là dall' Acqua ; es
re molto più di un fuoco di paglia . Di fare , il grana legale si trascinare per anno . Passò molto tempo prima che al signor Baggins essere concedere di essere di
uovo vivo . La gente che avere fare buono affare in quell' asta ci mettere un bello po' a convincersene ; e alla fine , per non perdere altro tempo , Bilbo dovere comp
re molto suo mobile . Gran parte dei suo cucchiaino d' argento essere misteriosamente scomparso , e non si uscire a saperne nulla . Personalmente , Bilbo sos
ttare dei Sackville-Baggins . Per contro , lui non riconoscere mai l' autenticità di il del Baggins tornare , e non intrattenere mai buono rapporto con Bilbo . La verità e
```

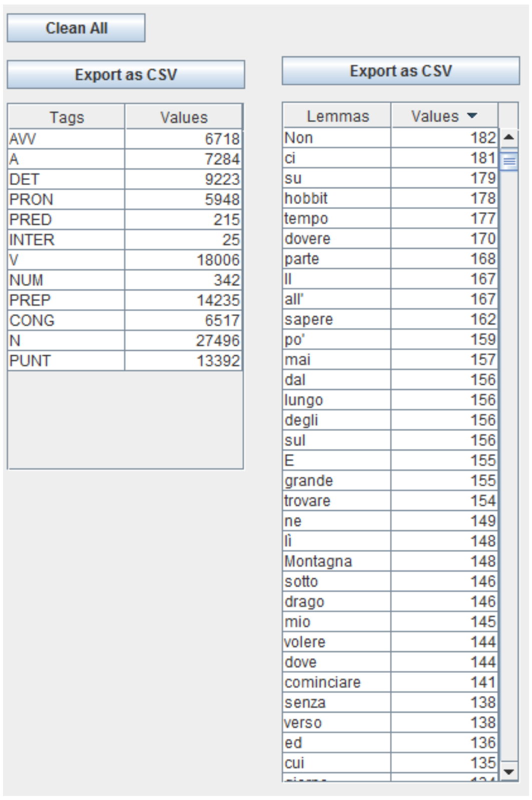
Figure 3.10: An example of results of *Lemmatization* button

As happens with the *Pos Tag Button*, when the Lemmatization stops its process, the second table on the right of the panel is automatically compiled with each lemma and its value of occurrence as shown in fig. 3.11.

The *Frequencies* tab is internally divided in two parts: on the left of the tab there are a selection menu and two tables, each one connected with a process button:

- the *Term frequency* button, for the computing of the Term frequency
- the *TF/IDF* button, for the TF/IDF calculation.

A selection menu, on the top, can be used in order to select the starting data for statistics: the user can calculate the Frequency and tf-idf starting from tokens occurrence or from lemmas occurrences.
Each one of the two tables is connected with a small text panel in which it is possible to insert the number of the text unit of which the user want to circumscribe the information of the table. For example, the user can write the number of the text unit he want and click the *show* button of the relative table. The table selects the text unit and show only the information about this unit on the table. This option could facilitate the analysis of the data by restricting the table view to a single portion of text.
On the right of the tab there are two panels. They generate two graphs about word distribution in text which permit to compare frequency values



Tags	Values
AVV	6718
A	7284
DET	9223
PRON	5948
PRED	215
INTER	25
V	18006
NUM	342
PREP	14235
CONG	6517
N	27496
PUNT	13392

Lemmas	Values
Non	182
ci	181
su	179
hobbit	178
tempo	177
dovere	170
parte	168
Il	167
all'	167
sapere	162
po'	159
mai	157
dal	156
lungo	156
degli	156
sul	156
E	155
grande	155
trovare	154
ne	149
lì	148
Montagna	148
sotto	146
drago	146
mio	145
volere	144
dove	144
cominciare	141
senza	138
verso	138
ed	136
cui	135

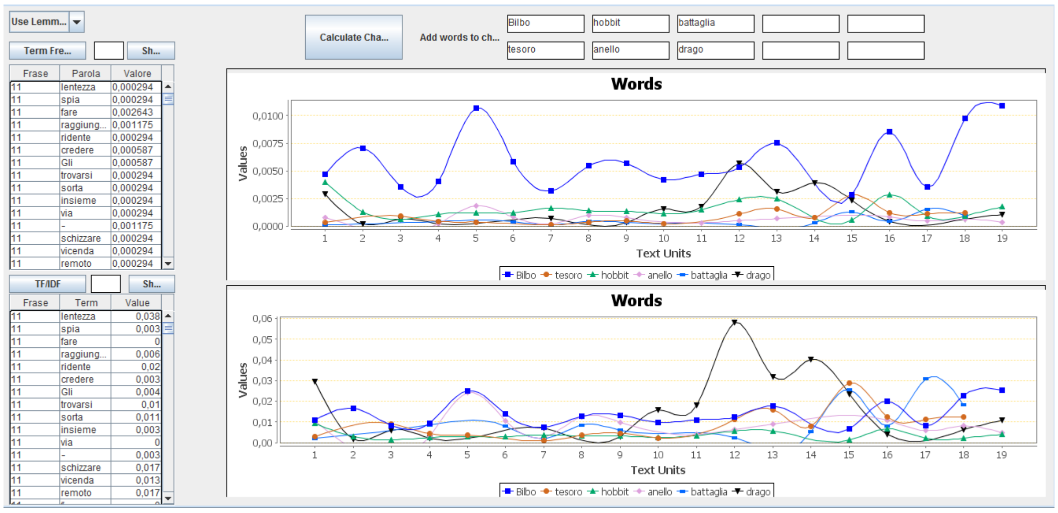
Figure 3.11: The two auto-compiled tables on the right of the *Pos Tag & Lemmatization* panel after the Lemmatization process ends

and tf-idf values. Above this two spaces the users can add up to ten words of which he want to show the distribution. The *Calculate Charts* button creates the two visualization3.13: the first one shows the distribution of the selected terms Frequency (above); the second one shows a chart with the distribution of the tf-idf of the same terms.

The charts are provided by a free Java library called JFreeChart[10]. The JFreeChart library provides a complete set of charts completely navigable, with a useful menu which permits to zoom in and zoom out the panel, save as PNG or change colors and visual options.

The *N-grams* tab only contains two tables in which, clicking on corresponding buttons, the bi-grams and tri-grams encountered into text and its statistics are showed.

[10]available at the `http://www.jfree.org/jfreechart/`



Frase	Parola	Valore
11	lentezza	0,000294
11	spia	0,000294
11	fare	0,002643
11	raggiung...	0,001175
11	ridente	0,000294
11	credere	0,000587
11	Gli	0,000587
11	trovarsi	0,000294
11	sorta	0,000294
11	insieme	0,000294
11	via	0,000294
11	-	0,001175
11	schizzare	0,000294
11	vicenda	0,000294
11	remoto	0,000294

Frase	Term	Value
11	lentezza	0,038
11	spia	0,003
11	fare	0
11	raggiung...	0,006
11	ridente	0,02
11	credere	0,003
11	Gli	0,004
11	trovarsi	0,01
11	sorta	0,011
11	insieme	0,003
11	via	0
11	-	0,003
11	schizzare	0,017
11	vicenda	0,013
11	remoto	0,017

Figure 3.12: The *Frequencies* panel after process the Term Frequency, the TF/IDF and clicking the *Calculate Charts* button.

The *Semantics* tab can be used to show the results of the application of the Semantic Module.
There are two main buttons above on the panel, the *Generate Sentence Network* button and the *Show Semantic Space*, both producing a different graph on the panel. The first button is connected to a filter panel, in which a user can introduce a Double value under which similarity values are excluded by the chart; both buttons are also connected with two text panel which allow to filter the text units the user want to select and with a select menu with 4 options (0, 5, 10 and 20) which permits to select the number of similar word that the Semantic Module can extract from the Semantic Space for each word.

While the *Show Semantic Space* button generates a Cartesian Space in which each word is represented as a coordinate, which are graphically represented into a Cartesian space, as shown in the fig. 3.13, the *Generate Sentence Network* uses another external technology to represent the semantic network of similarity of two text units.

We decided to use a powerful application for graph generation and data management and visualization called *Gephi*. Gephi is an open graph viz platform that provides a Java toolkit for integrate static graphs into user software. Gephi is released under the dual license CDDL 1.0 and GNU General Public License v3.
Gephi[11] is a leader technology in visualization and exploration of graphs and

[11]available at `https://gephi.org`

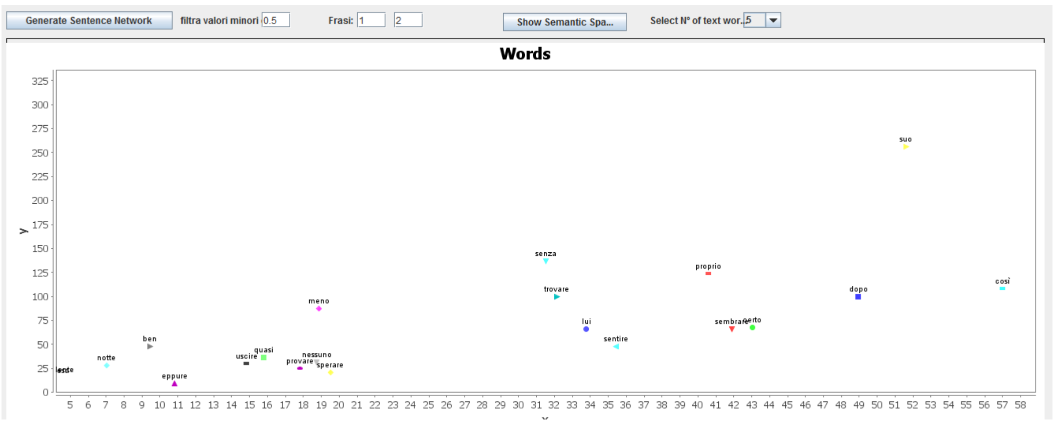


Figure 3.13: The *Semantics* panel after clicking the *Show Semantic Space* button.

networks and provides a clear and ergonomic interface for exploratory data analysis, link analysis, social network analysis etc. Gephi reaches high performance and it's completely customizable by offering a great number of plugins for layout, metrics, data sources, manipulation etc.

In addition, Gephi offers a complete set of API, integrated in its toolkit, completely written in Java. The toolkit is contained into a single JAR file that could be reused in any kind of Java application and achieves tasks that can be done in Gephi automatically.

In order to include Gephi Toolkit in the LG-Starship Framework, it was necessary to import the toolkit Jar into the project Built Path. Subsequently, we created a class Graph in which Gephi Project, Appearance, Import and Preview Controller have been initialized as Java objects. In order to create a new graph, we initialize a workspace and import the external *.dot* graph which has been create by the *HAL* algorithm embedded in the software and launched by clicking the *Generate Sentence Network* button. The file, which is automatically exported for external use, contains the generated graph in the following format:

Into the .dot file the Graph is declared as a list of nodes, followed by a list of edges and its related weights. The nodes are the words of the selected sentence or text unit and, eventually, the similar words extracted by the Hyperspace Analogue To Language Matrix.
Weights are the similarity values calculated as Euclidean Distances between the vectors of the two words. Values of 0 have been deleted from the graph because they generate entropy by forming visible connection with weight 0 between words.

Algorithm 11 The .dot Graph File generated by the HAL method

```
1: Graph G {
2: word1;
3: word2;
4: word3;
5: word1 – word2 [weight = "0.8"];
6: word2 – word3 [weight = "0.66"];
7: word1 – word3 [weight = "0.4"];
8: }
```

Once the graph has been imported, the software generate an Undirected Graph. The choice of this kind of graphs was an obligatory choice because each value of similarity is a bidirectional value and a similarity $a \rightarrow b$ has the same value of a similarity $a \leftarrow b$. After import the graph, a Modularity Algorithm, embedded into the Gephi Toolkit, has been applied.

Modularity [Blondel et al., 2008] is a simple method to extract community structures from large networks. It is an heuristic method that is based on modularity optimization [Clauset et al., 2004], that consist in recurrently merging communities that optimize the production of modularity. Modularity of a partition is a scalar value between -1 and 1 that measures the density of links inside communities as compared to links between communities [Girvan and Newman, 2002, Newman, 2006b]. Encountered communities are shown into the graph with different random colors.

At the last, we apply another layout algorithm called "ForceAtlas2" which is a force vector algorithm [Mathieu and Tommaso, 2011]. It simulates a physical system in which nodes repulse each other like magnets while edges attract the nodes they connect. These forces create a movement that converges to a balanced state. The final state helps to interpret the data.
The resulting network is similar to the one presented in fig. 3.14.

The last tab is the *Syntactics* tab in which the Syntactic Module of the Framework take a visual form. The tab, in fact, permits to launch the Syntactic Parser on the text and visualize the results for each verb encountered and analyzed. Actually, as we said in the previous section, the list of verbs and structure on which it is possible to launch the parser is short and incomplete. In the next mouths, this list will be enriched and completed in order to offer a real and complete syntactic parser to the user of the Framework.

The panel has two main sectors: the left sector contains the *Start Syntactic Parsing* button, which starts the analysis, and the *Download Tagged Text* button, that produces a .txt file in which the semantically tagged text is

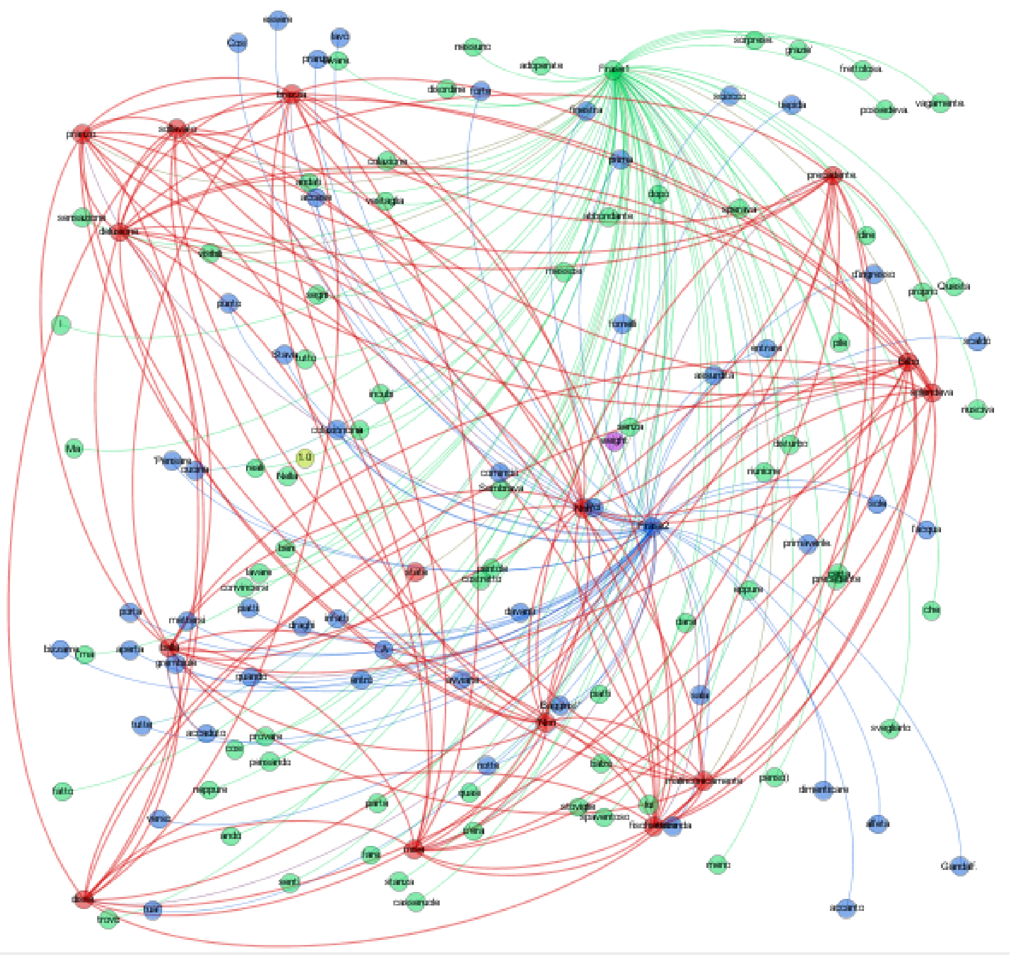
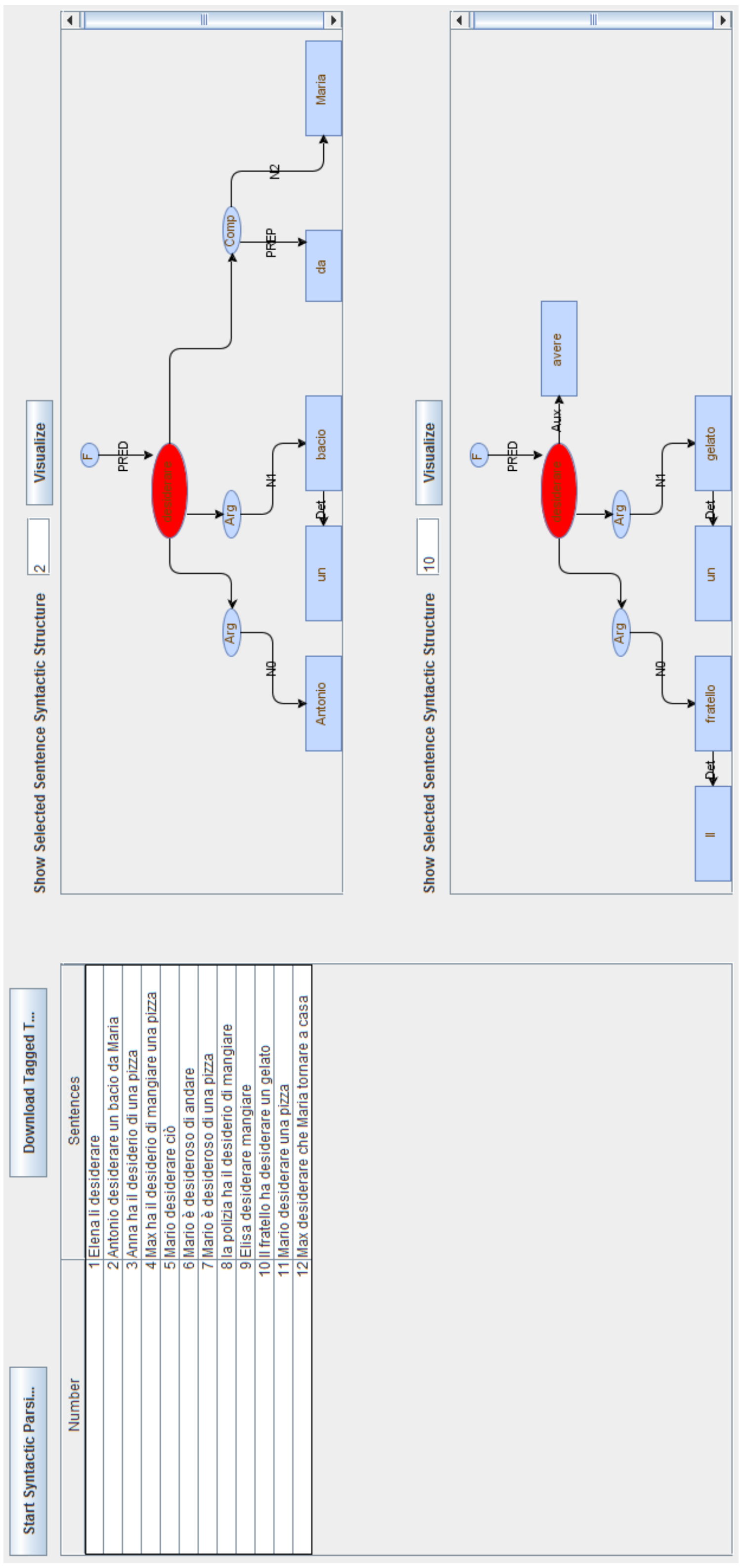


Figure 3.14: Example of network generated by Gephi Toolkit.

included. Below to these buttons, there is a large Table in which, once the analysis is completed, each sentence contained into the text will be enumerated and showed.

On the right side, there are two panels surrounded by two buttons and two text panels: writing the number of the sentences as indicated by the table and clicking on the button *visualize*, the window below will shows the graphic representation of the dependences of the selected sentence.



Number	Sentences
1	Elena li desiderare
2	Antonio desiderare un bacio da Maria
3	Anna ha il desiderio di una pizza
4	Max ha il desiderio di mangiare una pizza
5	Mario desiderare ciò
6	Mario è desideroso di andare
7	Mario è desideroso di una pizza
8	la polizia ha il desiderio di mangiare
9	Elisa desiderare mangiare
10	Il fratello ha desiderare un gelato
11	Mario desiderare una pizza
12	Max desiderare che Maria tornare a casa

Figure 3.15: The *Syntactics* panel.

In the fig. 3.15 the *Syntactics* tab is shown. Into the table on the left there are the sentences that the Syntactic Module extract from the text which, in this case, is a list of sentences which contains the Psychological Predicate *desiderare* and some other sentences. On the right, two selected sentences are shown in a graph format.

In the above panel the selected sentence was *Antonio desidera una bacio da Maria*, "Antonio desires a kiss from Maria". The module recognize the predicate *desiderare*, shown as a red ellipse, and connect it to the root (F).

From the Predicate node two arrows point the two arguments of the verb: N0, the subject, is the Noun "Antonio", N1 is *bacio*, "kiss" from which depends the determiner *un*, "a". The other complement couldn't be an argument because is not provided by the verb description. The algorithm classifies *da Maria*, "from Maria" as a simple complement in which "Maria" is the N2 element and *da*, "from" is the preposition.

In the below panel, the sentence represented is *Il fratello aveva desiderato un gelato*, "The brother had desired an ice cream". In this case, the verb is in a compound form. The predicate is connected by an arc labeled *Aux* with the auxiliary verb *avere*, "to have". The arguments are two: N0 represents the noun *fratello*, "brother", and N1 contains the noun *gelato*, "ice cream".
In order to show how the parser treat other kind of sentence structures, different kind of structures of *desiderare* have been inserted into the text.

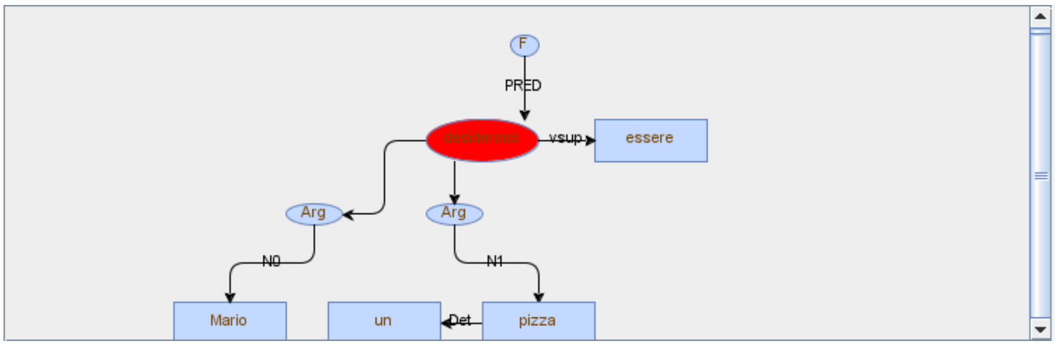


Figure 3.16: The generated dependency graph of the sentence *Mario è desideroso di una pizza.*

In fig 3.16, *Mario è desideroso di una pizza*, "Mario is desirous of a pizza", the predicate is not expressed directly by the verb *desiderare*, but by the support verb *essere*, "to be" and the deverbal adjective *desideroso*, "desirous". In the visualized structure the red eclipse which represent the predicate contains the adjective *desideroso* and is connected with its support verb by an arc labeled “Vsup”. The rest of the structure is represented exactly as it would be represented the sentence *Mario desidera una pizza*, “Mario desires a pizza”.

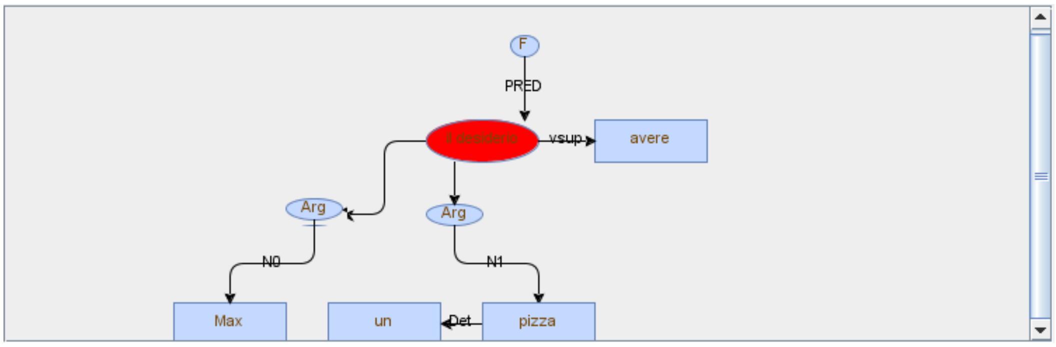


Figure 3.17: The generated dependency graph of the sentence *Mario ha il desiderio di una pizza.*

In fig. 3.17, the graph contains the structure of a support verb sentence with *avere*, “to have”, and the deverbal noun *desiderio*, “desire”. The sentence is *Mario ha il desiderio di una pizza*, “Mario has desire for a pizza” and, in this case, the predicate is the noun *desiderio*.

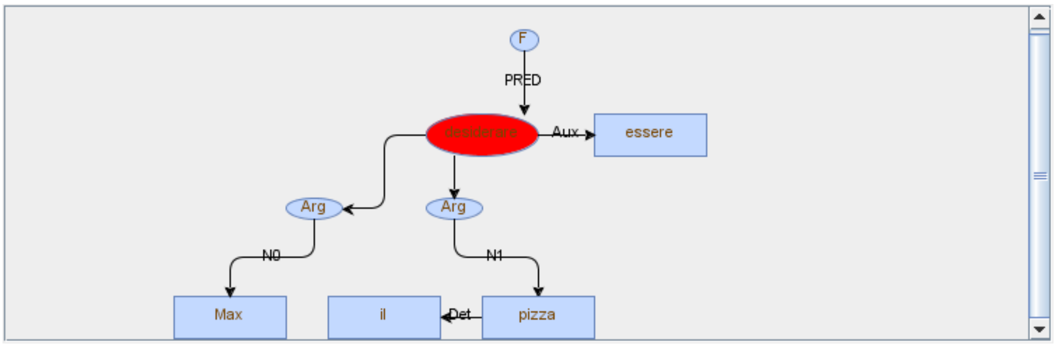


Figure 3.18: The generated dependency graph of the sentence *la pizza è desiderata da Max.*

In the example in fig. 3.18, is shown a sentence in passive form: *la pizza è desiderata da Max*, “the pizza is desired by Max”. The Syntactic Module process the sentence and represent it exactly as it was in active form: the red eclipse in which is represented the predicate contains the verb *desiderare*, and is connected with the auxiliary verb *essere*, "to be" by the "Aux" arc. As the algorithm found the form *N V desiderato da N*, it immediately assign the value of the first N to the variable N1, and conversely, the value of the second N to the variable N0, preserving the same graph structure of the active form of the same sentence.

In order to show how the LG-Starship Framework could be useful for different kind of text analysis, in the next chapter we present two experiment in which the Framework took an important role and many of its modules have been involved into the experiments.

Chapter 4

Experiments

In this chapter we try to illustrate how the framework has been utilized in two different practical projects which have been developed and published during the creation steps of the LG-Starship Framework and represented an inspiration for the design of the same framework.
For the two experiments we select two different kind of Corpora, belonging to the same macro-area of the User Generated Content:

- The first experiment represent a statistical study of the language of Rap Music in Italy through the analysis of a great corpus of Rap Song lyrics downloaded from on line databases of user generated lyrics.
- The second experiment is a Feature-Based Sentiment Analysis project performed on user product reviews. For this project we integrated a large domain database of linguistic resources for Sentiment Analysis, developed in the past years by the Department of Political, Social and Communication Science of the University of Salerno, which consists of polarized dictionaries of Verbs, Adjectives, Adverbs and Nouns.

In next sections we present this two works in detail, by illustrating motivation of corpus selection, state of the art of similar works, adopted methodology and external resources we have used in order to complete the projects.

4.1 Rapscape, the construction of a tool for the analysis of the Italian Rap Phenomenon

This section describes the steps which led to the realization of the first domain lexicon of Italian Rap Music. We applied a hybrid approach that benefits of the large amount of web available data to create a multi-level resource, combining linguistic, statistics and semantic information. We combined user generated contents, web mining and text analysis techniques, Natural Language Processing and data visualization tools to develop an interactive tool to describe, explore and query the semantic universe of Rap in Italian. The tool overcomes the lack of domain resources and provides a "distant reading" of one of the most significant cultural and linguistic phenomenon in recent years, taking advantage of Information Design techniques to make accessible and understandable a large amount of extremely sparse data.
The project make use of some of the LG-Starship Framework Modules in order to perform a rapid and precise analysis of the textual Rap sphere, integrating other powerful resources as Gephi and Tableau softwares for data visualization, Open Refine for data management and processing.

4.1.1 Initial guidelines

The rapid growth of the Web has led to increased availability of a huge amount of linguistic data. Linguistic resources are the baseline to build high-level applications and perform tasks in Natural Language Processing (NLP). The vast majority of NLP tasks, ranging from POS Tagging and Named Entity Recognition [Bunescu and Pasca, 2006] to Question Answering [Harabagiu et al., 2000], Text Classification [Nastase and Strube, 2008] and Summarization [Gabrilovich and Markovitch, 2006] are heavily dependent on rich and well built linguistic resources.

In this project we dealt with the musical domain, a field not really explored in the literature. In particular, we focus on creating a domain resource concerning the language of Italian Rap music. As emerged from recent studies [Perna et al., 2016], music genres can provide a representative overview of many social and linguistic phenomena.
Rap music, in particular offer many interesting cues for a textual analysis [Cutler, 2007, Terkourafi, 2010, Attolino, 2012], since it is one of the most vital phenomena and with greater socio-cultural impact in the music scene and in youth subcultures of the last years [Lena, 2006, Toop, 1984, Forman and Neal, 2004].
Furthermore, Rap music has become a global phenomenon extended well be-

yond the original North American borders [Androutsopoulos and Scholz, 2003, Osumare, 2007, Alim et al., 2008] and it presents a rich textual production and a high rate of innovation and experimentation of linguistic forms [Cutler, 2007, Bradley, 2009, Terkourafi, 2010].

The idea behind this project is to obtain a "mapping" of Italian Rap language, which allows to observe and analyze an extremely popular field of the contemporary cultural production in Italy [Pacoda, 1996, Filippone and Papini, 2002, Attolino, 2003, Scholz, 2005]. We focus mainly on textual aspects of lyrics, rather than musical ones. Although, Rap is characterized by a very close between word and rhythm [Bradley, 2009], the textual component occupies the central role [Attolino, 2012].

4.1.2 Methodology

Instead of dwell on a thorough analysis of a small number of texts, we decided to adopt a multidisciplinary approach - in which converge web data mining, linguistics and information design - aiming to build a very large text database to be analyzed using text-mining and computational linguistics tools and make browsable through a series of interactive views designed ad-hoc. The proposed method is composed by 4 steps:

- **Rapscape**: Building the resource extracting information from web repositories using web mining techniques;
- **cleaning and enrichment**: normalization, data cleaning and enrichment of Rapscape mixing different web resources;
- **text analysis**: linguistic and statistical analysis using LG-Starship Framework;
- **data visualization**: creation of an interactive tool to visualize, explore and query the data;

In the first phase we identified web repositories containing rap lyrics in Italian language. Official sources (official artist and record label websites) appear extremely poor in content or archived in difficulty usable formats. We chose to use user generated contents, exploiting text-sharing fan sites where users voluntarily provide their personal transcriptions or rap lyrics. This choice proved to be effective on both sides. From one side it allows to operate on a more representative dataset of the analyzed musical genre, since lyrics published in independent record labels - not only the official ones - are included. So features not otherwise identified can be seen; such as expressive forms not conventionally accepted by the majors and bad language. On the other side using user generated contents can give a sociolinguistics perspective, highlighting changes of linguistic forms of expression of artists when moving from an official context to an underground one.

In the second phase these mined data were cleaned, then crossed and enriched with other resources available online. In particular APIs providing highly accurate "technical" information about authors and lyrics have been used, since information provided by user generated contents are often inaccurate in some ways.

The third phase consists of "classical" Natural Language Processing analysis that make use of some of the modules of LG-Starship Framework presented in the previous chapter: data were tokenized (Preprocessing Module), pos-tagged and lemmatized (Mr.Ling Module).

At this stage we have taken account of previous studies conducted in the field of Music Information Retrieval (MIR), in particular those aimed to automatic analysis of lyrics [Mahedero et al., 2005, Kleedorfer et al., 2008, Hu et al., 2009] and rap texts [Hirjee and Brown, 2009, 2010, Malmi et al., 2015].

Using the Statistic Module, a series of statistical textual analysis were carried out, in order to obtain absolute term frequencies, author relative frequencies, collocations, bigrams, trigrams and words association strength. In addition, the distinctiveness of words has been calculated. This value is expressed using the Tf-idf measure.

After that, we started building the network, the proper semantic space of Rap language.Using the Semantic Module we created a semantic network of each author and allow their classification using machine learning algorithms (cluttering, topic modeling).

The fourth step concerns the visualization of collected data. In particular we develop an interactive tool able to explore and query the rap corpus. The tool allows to benefit from a "distant reading" of the large collection of data and at the same time to explore data in detail using a set of filter and zoom functions. In addition to classic Information Visualization approaches, the tool design takes account of the approach developed by Communication Design in the field of Digital Humanities [Uboldi and Caviglia, 2015], in particular regarding the definition of the user experience. The Visualization tool is composed by a set of tabs with different views focusing on different aspects and providing different possibility of interaction.

4.1.3 Building the resource

We start creating the Rapscape corpus, a POS-tagged and lemmatized lexical resource containing 16,000 Italian Rap songs grouped by artist. This corpus is composed by user generated contents, extracted mining different fansite websites using a web-scaping algorithm.

For this approach has been necessary to overcome the lack of official resources about Italian rap music. Although Rapscape is the only resource providing information about rap lyrics in Italian, it shows some problems. As a weakly-supervised resource based only on user generated contents its data can be wrong or inaccurate (e.g. typos in song titles or songs attributed to the erroneous author), furthermore although Rapscape contains a set of core linguistic information about each song and author, such as title, text, featuring, the information provided is too poor to allow for deeper analysis (e.g. statistic, semantic). Therefore, the first step in our work was to extend Rapscape giving it more robustness. To perform this task, we adopted a well-known semi-supervised approach in the literature [Navigli and Ponzetto, 2012, Bond and Foster, 2013], based on merging together data from different kinds of resources. So we took advantage of the huge amount of data available on the web to progressively enrich the core resource in semi-automatic way.

Before adding new information it was necessary to clean up original data. As mentioned above, Rapscape is built automatically, so its data need a clean up. Therefore the first problem we decided to deal with was to achieve cleaned data which could be processed effectively to obtain normalized texts. This normalization step covered the following aspects:

- **stopwords**: we detected and eliminated URLs and domain-specific stop words (e.g. *strofa*, *ritornello*, *rit* etc.);
- **featuring** rap music is characterized by several author featuring, which creates a problem in handling multiple authors. We have chosen to keep the names of other singers in the text (not credited as authors or collaborators) as these are considered "citations". These "citations" can be relevant in a future development of a citations network among the various authors;
- **punctuation** in this step punctuation has been left.

At the end of this preprocessing step a textual resource that counts more than 2,000 texts grouped by author was obtained. After that, we started the extension of the resource, taking into account different information from different sources on the web. The combination of various resources allows us to characterize artists and songs to perform further analysis and to compare the data and correct any errors. To increase the effectiveness of song data we need first to add some technical information, so we select a small set of features mined from two of the largest commercial resources freely available:

- **Discogs API**[1]: the largest on-line crowd-sourced database of music releases. For each song in our dataset we searched for its artist in Discogs API. Using artist name, we enriched our data with the following features: `alias` (alias of the artist) and `ID` (Discogs ID of the artist). We subsequently used the id to extract some information about songs: `master` (album), `year` (year of the release) and `labels` (release label).

- **Spotify API**[2]: the official Spotify API provides for each artist a complete list of its tracks, and for each track information about specific aspects of a songs (e.g. `duration_ms` or `track_number`). We used `artist name` value to retrieve the following features in the API: `URL` (the URL to artist's image), `popularity` (a value between 0 and 100) and Spotify `ID` for the artist; We then used artist `id` to retrieve: `album` (personal, compilations, featuring).

For each artist we looked for titles of its songs, and merged together this information in a JSON object.[3]

Currently our resource contains 2,465 rap songs grouped by 290 artists. Both artists and songs are accompanied with features about labels, year of publication, title, co-authorship and ID.

4.1.4 Text Analysis

After building up the resource, we started to carry out some basic analysis of basic statistics by using the LG-Starship modules.
At first glance, we note that Rap language presents some some distinctive features that may adversely affect the analysis:

- the large number of neologisms and colloquial terms;

- the extreme fragmentation of texts: usually a line is syntactically disconnected by the next one.

- the large number of dialect words.

[1]`https://www.discogs.com/developers/`

[2]`https://developer.spotify.com/web-api/`

[3]To query APIs we used python clients:
`discogs_client` (`https://github.com/discogs/discogs_client`) and `spotipy` (`https://github.com/plamere/spotipy`).

Since these features makes a syntactic analysis impossible, we decided to perform some classical statistical NLP analysis on the data: after a Pos tagging and a Lemmatization phase, we calculate Term Frequency, Tf-idf and perform N-grams extraction.

Concerning POS tagging and Lemmatization, we use the Mr.Ling Module. The module produces a lemmatized and pos-tagged text (960.349 tokens, 60.000 types) as output.

As we said, the great influence of colloquial language and neologisms, including the large use of *bad words* and dialect words, and the presence of typing errors, make the Pos Tag and the Lemmatization more difficult.
The calculated precision of the module for these two task with a regular text, decreases drastically with the actual corpus: while the Pos Tagger, which is based on a probabilistic algorithm, can overcome these class of errors, though the fragmentation of a text characterized by short sentences results in a decrease of the precision due to the lack of context, the Lemmatization task suffers these problems in a more pronounced way: in tab. 4.1.4 an example of error for each of the cited phenomena are showed.

phenomenon	**example**	**POS Tag**	**Lemmatization**
Colloquial language	*smamma* “shove off”	N	smamma
Neologisms	*reppo* “to rap”	N	rappo
Bad words	*troia* “bitch”	N	troio
Dialect words	*uocchie* “eyes”	A - N - V	oucchie
Foreign words	that	N	that
Typing errors	*accezzione* “meaning”	N	accezzione
Abbreviations	*bro* “brother”	N	bro
Distortions	*zzziooo* “uncle”	N	zzziooo

Table 4.1: Examples of words that produce errors in Pos Tagging and Lemmatization

We uses the resulting annotated text to perform some statistical analysis using some classical data mining measures: word absolute and relative frequency, Tf-idf, Bigrams and Trigrams frequency. The results of this step, performed by the Statistic Module of the LG-Starship Framework, is a large Tf-idf matrix Author per Terms, an Author-word frequency matrix and two Author-Ngrams matrices. In order to limit the dimension of the Tf-Idf matrix, that originally includes 290 rows (one for each singer) and a number of words ranged between 300 and 1000, we select the 30 words with highest value of Tf-Idf per author. The same operation has was carried out with Term Frequency values and N-grams Values.

In addition to basic statistics analysis, we focused on identifying the most interesting words of Italian rap language. A useful approach might be to compare how the relative frequency of words in rap songs differs from Italian language in general. To make this analysis we compared our corpus with *ItWaC*[4] [Baroni et al., 2009] , a huge word corpus constructed from the Web and using words from *La Repubblica Corpus* [Baroni et al., 2004]. To make the comparison we defined an arbitrary index that measures the *"Rapness"*, R_w for each word w:

$$R_w = \log \frac{N^w_{rap}}{N^w_{ItWaC}} \tag{4.1}$$

where R is the frequency of occurrences of word type w in our corpus of rap lyrics and is the frequency of occurrences of word type w in the *ItWaC* corpus. The Rapness has been calculated both on the total corpus and for individual artists. We considered only words with a full semantic value, filtering by part of speech only nouns, verbs and adjectives. To prevent rare words, we took only the top 3000 most frequent words in the corpus and the most frequent 50 for each part of speech for authors. The top 10 rap words are shown in the following table.

word	**rapness**	**POS**
sveglia	5.07	N
sguardo	4.32	N
maria	3.57	N
felicitá	2.96	N
fake	2.94	N
rapper	2.91	N
avvelenato	2.61	A
maledire	2.54	V
rappare	2.24	V
volere	1.43	V

Table 4.2: Top 10 most rap words.

Notice that this is a slightly unbalanced comparison, because we compare domain texts with general ones. A better measure of what constitutes *Rapness* should provide a comparison with lyrics of other genres, but currently this kind of resource for the Italian language does not exist. Another method to

[4]`http://wacky.sslmit.unibo.it/doku.php?id=corpora`

measure linguistic difference between rap language and the Italian language in general is measuring how the distribution of parts of speech vary between them. Table 4.1.4 shows the comparison between rap parts of speech with *ItWaC* ones.

POS	ItWaC	Rap
N	941.990 (40,4%)	1.823.423 (54,5%)
ADJ	706.330 (30,3%)	383.369 (11,4%)
V	679.758 (29,2%)	1.141.949 (34%)

Table 4.3: POS distribution.

Although the corpora differ much in size and are not directly comparable, the percentage of distributions gives us an interesting overview of the linguistic characterization of rap songs. We can see that Rap lyrics focus much more on nouns and verbs, whereas adjectives seem to be much less relevant than.

As mention above, it is highly difficult to carry out a syntactic analysis due the features of texts. In order to analyze the Rapscape corpus from a semantic point of view, we decided to perform two different types of semantic analysis that don't need syntactic parsing.

First, we chose to calculate the similarity between authors using a state of the art semantic measure as the Cosine Similarity [Huang, 2008]. To calculate similarity between authors, we apply the Cosine Similarity to a matrix of Authors per Tf-idf.
After that, we apply the Semantic Module of LG-Starship Framework. We extend the semantic space of each author with other similar words that could not appear in the corpus and calculate the semantic space of the Corpus. The results of this approach is a table in which each row represent a term, a distance value and a second term.

4.1.5 Data Visualization

The goal of the Visualization step is the development of an interactive web-based (html, css, js) tool which make the dataset browsable and queryable. In order to build the tool we examine the specific problems posed by the visualization of large text corpora [Wise et al., 1995, Fortuna et al., 2005, Alencar et al., 2012, Sinclair et al., 2015, Kucher and Kerren, 2014, Brath and Banissi, 2015] and consider some suggestions proposed by previous works on lyrics databases [Labrecque, 2009, Baur et al., 2010, Oh, 2010, Sasaki et al., 2014].

To achieve this goal we chose to use some techniques grounded in Information Design and Data Visualization field. In recent years several theoretical approaches, methodologies and tools for the interactive exploration of large collection of data have been developed, with particular focus to web-oriented technologies. In Information Design field, the textual dataset visualization deals several issues, ranging from words readability to spatial distribution of large collection of texts. To overcome these known issues we have chosen to adopt an "explorative" approach, following the well known paradigm in the literature [Shneiderman, 1996] based on overview first, zoom and filter, then details-on-demand.

This approach allows us to efficiently map data using visual dimensions in order to create an interactive graphic representation, allowing to overview the whole corpus and several filtered restricted views. For the first development of the prototype we used *Tableau Public*[5], *Gephi* and *Sigma.js*[6], tools freely available on the web.

The visualization tool is composed by a set of "views" and navigation filters that allow to observe data from multiple angles through different levels of detail following canonical Information Design pattern. A series of visualization layers is combined together in some views providing an overall look about various aspects of the dataset: basic statistics such as frequencies and terms distribution, range of vocabulary, word rankings, bipartite networks between authors and terms; networks between words and its most representative semantic cluster.

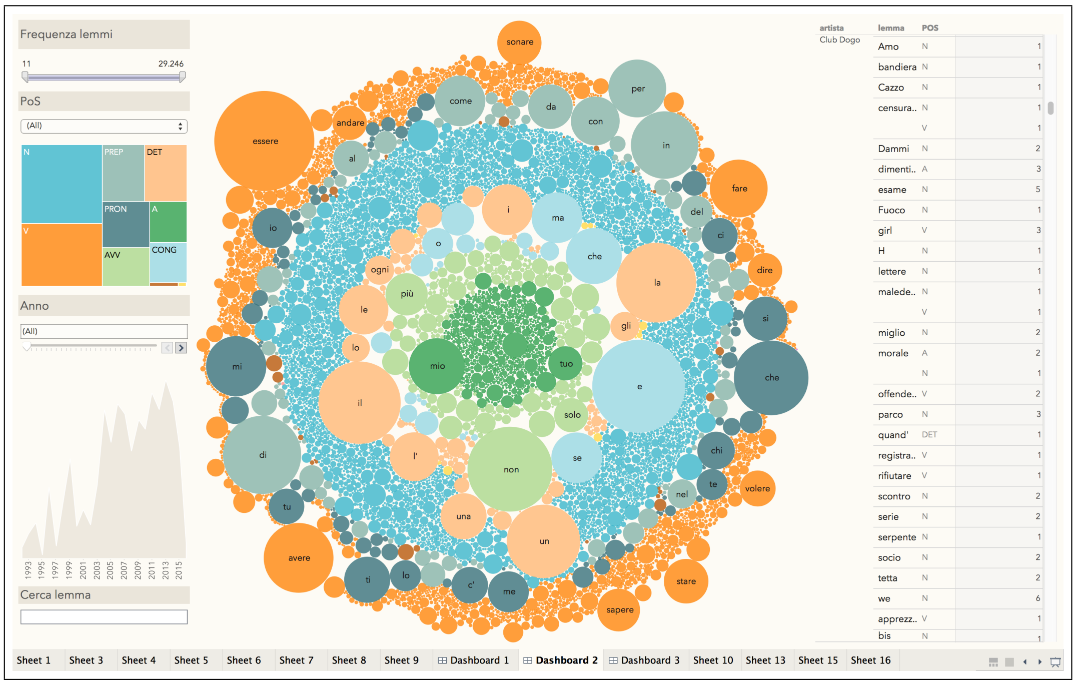
Filters and secondary views are designed instead to move quickly between different levels and perspectives on the dataset and go into the details to analyze data relating to the individual author (most frequent terms, dominant topics, etc) or individual lyric. It is also possible to draw comparisons between the authors (or groups of authors), or between lyrics (or groups of lyrics). Therefore the visualization is designed primarily as an exploratory tool that makes possible the analysis of Italian Rap textual universe at different levels of depth and granularity. Following figures present some relevant views and filters of the tool.

The first visualization shows in figure 4.1 shows the overall word frequencies in the corpus calculated by the application of the Preprocessing Module and Mr.Ling Module. The visualization technique adopted is the Circle Packing: each word is a labeled circle. Dimension and color of the circles are related to the number of simple word occurrences in the dataset.

[5]`www.tableau.com/`

[6]`sigmajs.org/`

This first kind of visualization provides an overall view of the entire dataset (Overview first). The right bar shows lemmas, part-of-speech and absolute frequency grouped by author. It is possible to explore data, filtering by author, frequency, year or part of speech (Zoom and filter) using the left bar. In addition, clicking on a single circle a set of specific information about lemma can be obtained (Details on demand). Circles' sizes depend to tf-idf value.



artista	lemma	POS	
Club Dogo	Amo	N	1
	bandiera	N	1
	Cazzo	N	1
	censura..	N	1
		V	1
	Dammi	N	2
	dimenti..	A	3
	esame	N	5
	Fuoco	N	1
	girl	V	3
	H	N	1
	lettere	N	1
	malede..	N	1
		V	1
	miglio	N	2
	morale	A	2
		N	1
	offende..	V	2
	parco	N	3
	quand'	DET	1
	registra..	V	1
	rifiutare	V	1
	scontro	N	2
	serie	N	2
	serpente	N	1
	socio	N	2
	tetta	N	2
	we	N	6
	apprezz..	V	1
	bis	N	1

Figure 4.1: Overall word's frequencies

In the example, we choose not to filter the dataset. As we expected, bigger circles represent determiners, preposition or verbs. It is interesting to note that, before the circle *essere*, "to be", and *avere*, "to have", the bigger verb circle is *sonare*, a dialectal term for *suonare*, "to play". This highlight the importance of dialect languages into the Italian Rap Language.

The data derived by the application of the Statistic Module is shown in various new tabs: the second visualization (figure 4.2) shows the changes in tf-idf rank for each word on a time scale. Using the left bar data can be filtered by Part-Of-Speech, tf-idf value or year.

In the example presented in the Fig. 4.2, we set the "artist" on *Fabri Fibra* and reduce the dataset to words with high tf-idf value (greater than 0.1610). Time line only shows years in which the selected artist published an

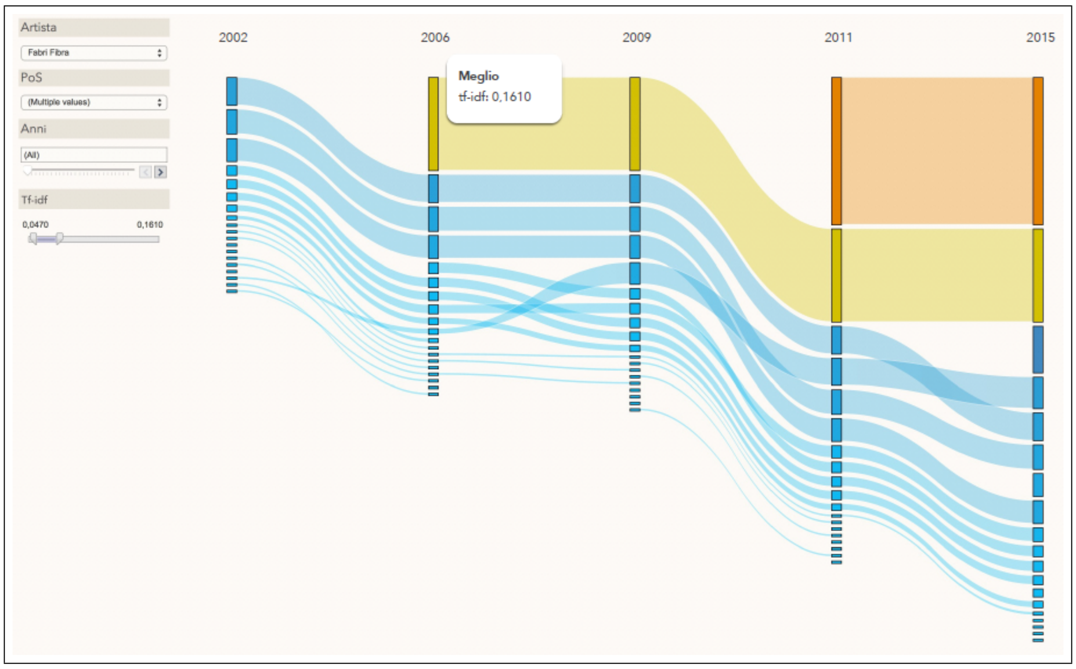

Artista
Fabri Fibra
PoS
(Multiple values)
Anni
(All)
Tf-idf
0,0470 0,1610

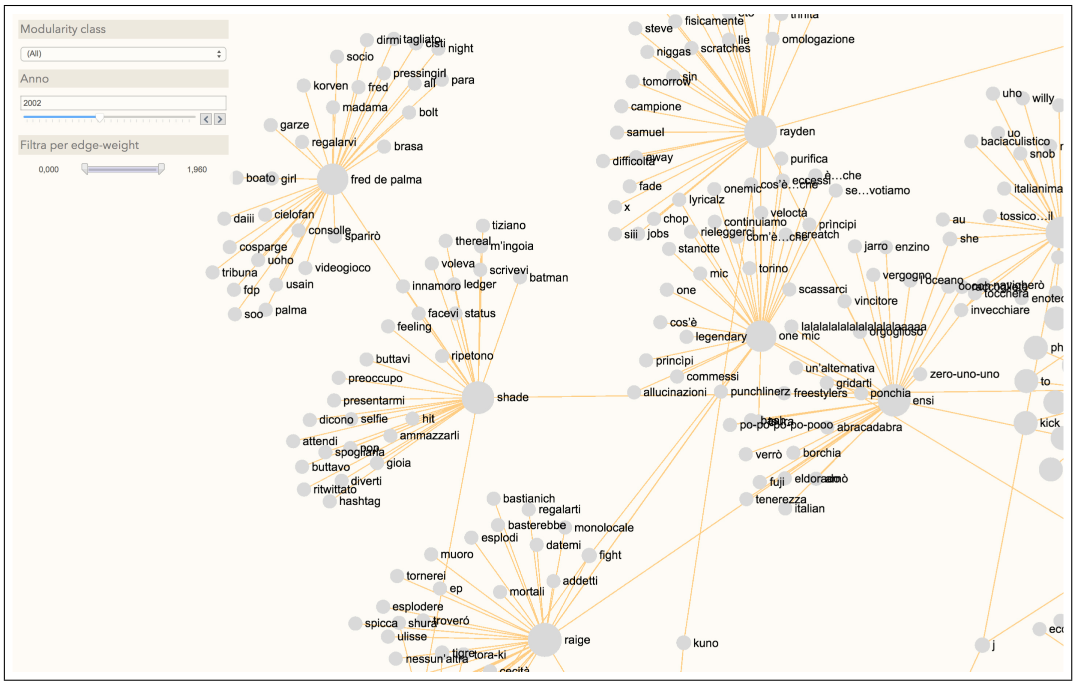
Figure 4.2: Changes of word's rank for author in relation with the year

album.

The Figure 4.3 shows a detail of the tf-idf bipartite network, representing co-citation between authors. In the left bar it is possible to filter nodes by year or by value. Bigger nodes represent authors, on the contrary, smaller represent terms.

This kind of visualization is useful to show the evolution of impact of single words over time. The original dataset contains the first 30 words with the higher tf-idf value for author. Different authors nodes could connect with the same term and, in case of co-citation, we can find a direct connection between two authors nodes. In the example we decided to show values after year 2002: the author "Shade", for example, is connected to word like *batman,feeling*, *status*, *pop* or *hashtag*, but also with the author node "Fred De Palma".

As mentioned above, co-authorship (featuring) is a common phenomenon in Rap music: the vast majority of lyrics have two or more authors. In order to extract information about co-authorship, we collect every author's names (and their aliases) and perform a pattern matching to find them in lyrics (usually the author's names is in brackets, but there may be many several



Modularity class
(All)
Anno
2002
Filtra per edge-weight
0,000 1,960

Figure 4.3: Detail of the Graph Author-Word in which each author was connected with the words with higher tf-idf value

other possibilities). The results are shown in 4.4, a co-author network in which nodes are authors and weighted arcs represent the number of shared songs.

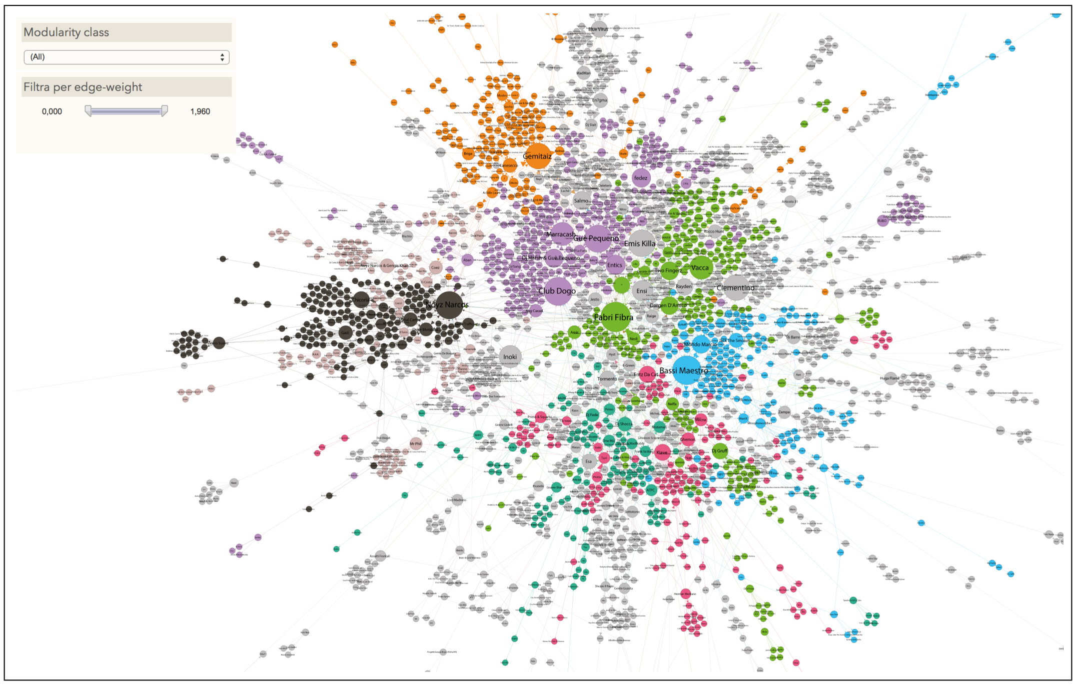


Figure 4.4: The Authors Collaboration Network

Co-authorship network, also called Collaboration Network, as shown in fig. 4.4, has been calculated applying a built-in Gephi algorithm called "Modularity Class" that measure the strength of division of a network into groups or communities [Newman, 2006a]. This graph shows how mainstream authors are strongly interlinked, but there are many small groups of underground authors or local communities.

The 4.5 shows the Cosine Similarity Network, in which we represent similarity between authors using Cosine Similarity algorithm applied to the Author's vector of tf-idf.
Resulting clusters are coherent with real relations between authors. In fig. 4.5 is shown a blue cluster that represent the Naples Area, a series of author belonging to the geographical, cultural and linguistic area of Naples. In the same way, the brown group in the top right represent a community of authors that worked together in the past.

The figure 4.6 represents the Semantic Network of words, in which the

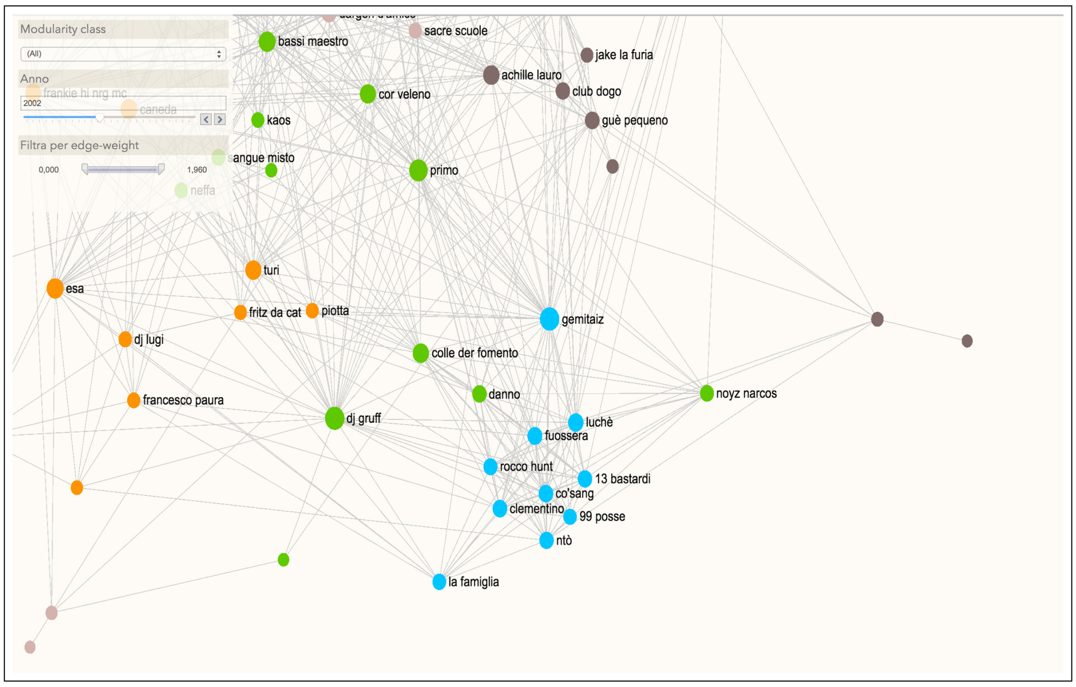
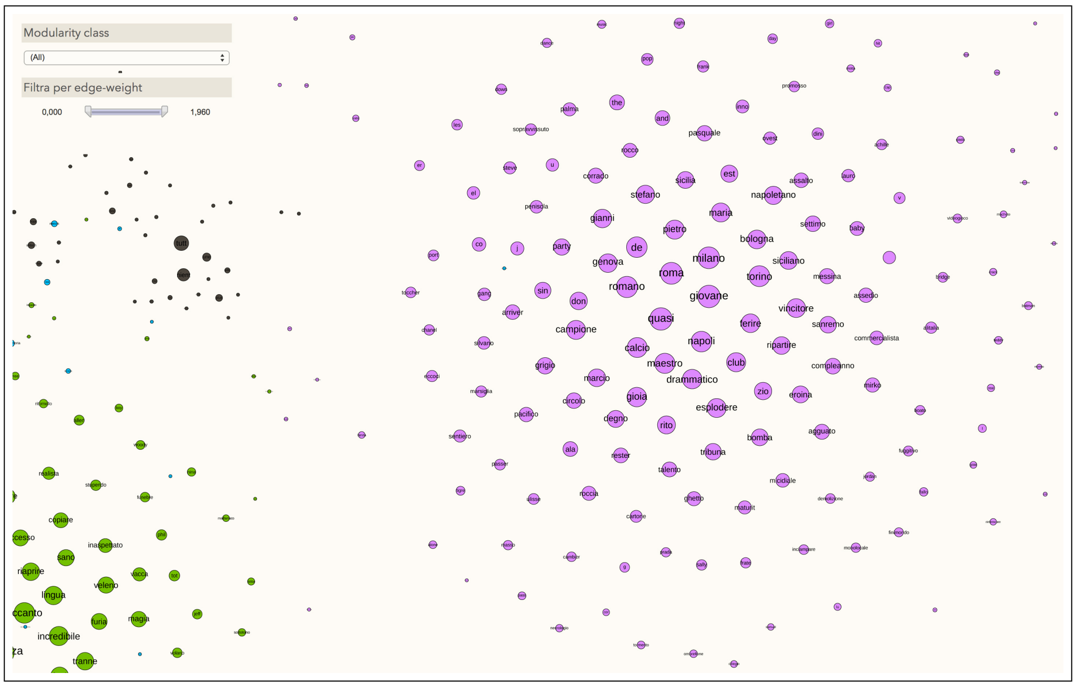


Figure 4.5: A detail from the Cosine Similarity Network of Authors

distance between words are extracted from the HAL matrix generated by the Semantic Module; some semantic clusters are visible.

In the example, we focused on a single cluster that includes geographical words such as *roma*, *milano*, *bologna*, but also *napoletano*,*siciliano*, *ovest ghetto* or *penisola*, and proper names as *stefano*, *pasquale*, *alitalia* or *commercialista*. This is an example of how the Semantic Module can produce knowledge about a text or a corpus: downloading the vector of the words with an high weight value, it is possible to find different clusters of terms and identify thematic areas included into the analyzed text. It could be possible to select different groups of authors and compare the emerged semantic network in order to compare how change the semantics of each group.

This project underlines how a linguistic tool as the LG-Starship Framework is able to rapidly construct different kind of linguistic databases about large and disparate corpora of texts: in fact, a linguistic study of a musical phenomenon as Rap or other kind of cultural textual phenomenon (i.e. film scripts, novels, etc.) could be carried out by users who are not familiar with linguistics theories or techniques. Thanks to the LG-Starship Framework, this kind of analysis, including a comparative study on different cultural



Modularity class

(All)

Filtra per edge-weight

0,000 1,960

Figure 4.6: A detail of Hal Network

areas will be developed in short time and in a simple and direct way.

4.2 A rule-based method for product features Opinion Mining and Sentiment Analysis

In this section we present an experiment of feature-based Opinion Mining and Sentiment Analysis of hotel, smartphones and videogames users review. The experiment, which involves many of the modules presented in the previous chapter (3.3), includes the following steps:

- Collection of a user-generated review Corpus;
- Postagging and Lemmatization of the Corpus grounded on the Mr.Ling Module;
- Feature-Based Opinion Mining based on the Syntactic Module;
- Semantic Expansion based on the Semantic Module applied to the extracted opinions.

As shown in the previous list, the core of this project lays entirely on three modules of the LG-Starship Framework: it uses the dependency parser contained into the Syntactic Module in order to extract sentences which contains psychological verbs and, in addition, we implement a new series of grammars which found support verb construction with evaluative adjectives or verbles sentences with evaluative adjectives. New grammars are based on the *SentIta* dictionaries which is a database of electronic dictionaries of sentiment nouns, adjectives, adverbs and verbs accompanied by polarity values.

The entire project will be presented in the following sections.

4.2.1 Initial guidelines

Consumers, as Internet users, can freely share their thoughts with huge and geographically dispersed groups of people, competing, this way, with the traditional power of marketing and advertising channels.
The expansion of the e-commerce, and the growth of the user generated contents, that can constitute large scale databases for the machine learning algorithms training, facilitated the growth of interest around the automatic treatment of opinion and emotions.
The same preconditions caused a parallel raising of attention also in economics and marketing literature studies. Basically, through user generated contents, consumers share positive or negative information that can influence, in many different ways, the purchase decisions and can model the buyer expectations,

above all with regard to *experience goods* [Nakayama et al., 2010]; such as hotels [Ye et al., 2011, Nelson, 1970], restaurants [Zhang et al., 2010], movies [Duan et al., 2008, Reinstein and Snyder, 2005], books [Chevalier and Mayzlin, 2006] or videogames [Zhu and Zhang, 2006, Bounie et al., 2005].

The rapid growth of the Internet drew the managers and business academics attention to the possible influences that this medium can exert on customers information search behaviors and acquisition processes. In summary, the growth of the user generated contents and the eWOM (electronic Word of Mouth) can truly reduce the information search costs. On the other hand, the distance increased by e-commerce, the content explosion and the information overload typical of the BigData age, can seriously hinder the achievement of a symmetrical distribution of the information, affecting not only the market of experience goods, but also that of search goods. An appropriate management of online corporate reputation requires a careful monitoring of the new digital environments that strengthen the stakeholders' influence and independence and give support during the decision making process.

It would be difficult, for example, for humans to read and summarize the huge volume of data about costumer opinions. However, in other respects, to introduce machines to the semantic dimension of human language remains an ongoing challenge.
In this context, business and intelligence applications, would play a crucial role in the ability to automatically analyze and summarize, not only databases, but also raw data in real time.

The largest amount of on-line data is semistructured or unstructured and, as a result, its monitoring requires sophisticated Natural Language Processing (NLP) tools, that must be able to pre-process them from their linguistic point of view and, then, automatically access their semantic content.

In any case, it is of crucial importance for both customers and companies to dispose of automatically extracted, analyzed and summarized data, which do not include only factual information, but also opinions regarding any kind of good they offer.

Companies could take advantage of concise and comprehensive customer opinion overviews that automatically summarize the strengths and the weaknesses of their products and services, with evident benefits in term of reputation management and customer relationship management.
Customer information search costs could be decreased trough the same overviews, which offer the opportunity to evaluate and compare the positive and negative experiences of other consumers who have already tested the same products and services.

4.2.2 The Product Feature Extraction

Differently from what happens with objectivity language it is impossible to directly observe or verify subjective language. *Opinions* are defined as positive or negative views, attitudes, emotions or appraisals about a topic, expressed by an opinion holder in a given time. They are represented by Liu [2010] as the following quintuple:

$$o_j, f_{jk}, oo_{ijkl}, h_i, t_l$$

Where o_j represents the *object* of the opinion; f_{jk} its *features*;oo_{ijkl}, the positive or negative opinion *semantic orientation*; h_i the *opinion holder* and t_l the *time* in which the opinion is expressed.
Because the *time* can almost alway be deducted from structured data, we focused our work on the automatic detection and annotation of the other elements of the quintuple.
The *Semantic Orientation* (SO) gives a measure to opinions, by weighing their *polarity* (positive/negative/neutral) and *strength* (intense/weak) [Taboada et al., 2011, Liu, 2010]. The polarity can be assigned to words and phrases inside or outside of a sentence or discourse context. In the first case it is called *prior polarity* [Osgood, 1952]; in the second case *contextual* or *posterior polarity* [Gatti and Guerini, 2012].
We can refer to both opinion *objects* and *features* with the term *target* [Liu, 2010], represented by the following function:

$$T{=}O(f)$$

Where the *object* can take the shape of products, services, individuals, organizations, events, topics, etc., and the *features* are components or attributes of the object.
Each object O, represented as a "special feature", which is defined by a subset of features, is formalized in the following way:

$$F = \{f1,\ f2,\ \ldots,\ fn\}$$

Targets can be automatically discovered in texts through both synonym words and phrases W_i or indicators I_i [Liu, 2010]:

$$W_i = \{w_i1, w_i2, \ldots, w_im\}$$
$$I_i = \{i_i1, i_i2, \ldots, i_iq\}$$

In order to provide a formal definition of sentiments, we refer to the function of [Gross, 1995]:

P(sent,h) Caus(s, sent, h)

In the first one, the *experiencer* of the emotion (h) is function of a *sentiment* (*Sent*), through a predicative relationship; the latter expresses a *sentiment* that is caused by a *stimulus* (s) on a person (h).

Pioneer works on feature-based opinion summarization are Hu and Liu [2004, 2006], Carenini et al. [2005], Riloff et al. [2006] and Popescu and Etzioni [2007]. Both Popescu and Etzioni [2007] and Hu and Liu [2004] firstly identified the product features on the base of their frequency and, then, calculated the Semantic Orientation of the opinions expressed on these features. Recent works are Khan et al., Wei et al. [2010] and Zhang and Liu [2011] who selected candidate product features by employing noun phrases that appear in texts close to subjective adjectives, Gutiérrez et al. [2011] exploited Relevant Semantic Trees (RST).
Wei et al. [2010] proposed a semantic-based method that made uses of a list of positive and negative adjectives defined in the General Inquirer to recognize opinion words and, then, extracted the related product features in consumer reviews.

4.2.3 Automatic Feature-based Opinion Mining

The first step of the Project consisted in the Corpus Collection: the corpus dataset has been built using Italian opinionated texts in the form of users' reviews and comment found on e-commerce and opinion websites. It contains 600 text units (50 positive and 50 negative for each product class) and refers to three different domains, hotels, cellulars and videogames, for all of which different websites have been exploited. The Internet portals used are: *ciao.it*, *amazon.it* and *tripadvisor.it* and *booking.it* for hotel reviews. Each single review has been stored with a tag structured as follow:

$$C\#\#P\#$$

C indicates the category: H for hotels, V for videogames, C for cellulars; the category is followed by a numerical identity ranging from 00 to 50. The polarity of the opinion is expressed by a P for positive and N for negative followed by a number indicating the value of the opinion given by the user.

Category	**Entries**	**Example**	**Translation**
Adjectives	5,381	*allegro*	cheerful
Adverbs	3,693	*tristemente*	sadly
Compound Adv	774	*a gonfie vele*	full steam ahead
Idioms	577	*essere in difetto*	to be in fault
Nouns	3,215	*eccellenza*	excellence
Psych Verbs	604	*amare*	to love
LG Verbs	651	*prendersela*	to feel offended
Bad words	182	*leccaculo*	arse licker
Tot	15,077	-	-

Table 4.4: Composition of SentIta

As second step, the corpus collected has been processed with the LG-Starship Framework: after a first pre-processing phase which has been carreid out by the Preprocessing Module, the corpus has been divided in 18 text units and Postagged and lemmatized with the Mr. Ling module.

The third step regards the opinion extraction. Our contribution to the resolution of the opinion feature extraction and selection tasks, goes through the exploitation of the *SentIta* database [Pelosi, 2015] that consists of a set of dictionaries that describe their lemmas with appropriate semantic properties; they have been used to define and summarize the features on which the opinion holder expressed his opinions.

SentIta contains the dictionaries shown in table 4.4. Great part of these dictionaries have been manually tagged, but some of their (adverbs and derived nouns) have been automatically built Pelosi [2015], Maisto and Pelosi [2014a,b]. In *SentIta* each lemma is tagged with polarity values (*POS* for positive values and *NEG* for negative values) and/or strength values (*FORTE* when the value increase and *DEB* when the value decrease).

Other Electronic dictionaries included into the project are the Concrete Noun dictionary that has been manually built and checked by seven linguists. It counts almost 22,000 nouns described from a semantic point of view by the property Hyper that specify every lemma's hyperonym. The tags it includes have been described in the tab. 3.5 of the section 3.2.5.

In the Feature Extraction phase the sentences or the noun phrases expressing the opinions on specific features are found in free texts. Annotations regarding their polarity (BENEFIT/ DRAWBACK) and their intensity (SCORE=[-3;+3]) are attributed to each sentence on the base of the opinion words on which they are anchored.

In order to complete the analysis, the Syntactic Module has been enriched with the SentIta resources: in fact, the original module only extract sentences in which appear Psychological Predicates, but ignore verbles sentences and sopport verb sentences in which the adjective which follow the support verb is not included into the databases of deverbal adjectives connected to the Psychological Predicates. Furthermore, the module share from SentIta the polarity values of each lexical entry.

We added to the original module a dictionary of evaluative adjectives that go to represent the alternative anchors for the parser.

The sentiment expressions in which we inserted the adjectives from SentIta are of the kind *N0 essere Agg val.* Where *Agg val* represents an adjective that expresses an evaluation [Elia et al., 1981] and the support verb for these predicates is *essere* "to be".
This verb gives its support also to the expression *N0 essere un N1-class* (e.g. *Questo film è una porcheria* "This movie is a mess"), that without it, together with the adjectives, would not posses any mark of tense [Elia et al., 1981].
A great part of the compound adverbs possess also an adjectival function (e.g. *a fin di bene* "for good", *tutto rose e fiori* "all peace and light", so they have been included in this support verb construction as well.

The support verbs' equivalents included in this case are the following [Gross, 1996].

- aspectual equivalents: *stare* "to stay", *diventare* "to become", *rimanere*, *restare* "to remain";
- causative equivalents: *rendere* "to make";
- stylistic equivalents: *sembrare* "to seem", *apparire* "to appear", *risultare* "to result", *rivelarsi* "to reveal to be", dimostrarsi, mostrarsi "to show oneself to be".

Among the Italian LG structures that include adjectives[7] we selected the following, in which polar and intensive adjectives can occur with the support verb *essere* [Vietri, 2004, Meunier, 1984]:

- Sentences with polar adjectives:

[7]For the LG study of adjectives in French see Picabia [1978], Meunier [1999, 1984].

  - *N0 Vsup Agg Val*

    L'idea iniziale era accettabile[+1]

    "The initial idea was acceptable"

  - *V-inf Vsup Agg Val*

    Vedere questo film è demoralizzante[–2]

    "Watching this movie is demoralizing"

  - *N0 Vsup Agg Val di V-Inf*

    La polizia sembra incapace[–2] di fare indagini

    "The police seems unable to do investigate"

  - *N0 Vsup Agg Val a N1*

    La giocabilità è inferiore[–2] alla serie precedente

    "The playability is worse than the preceding series"

  - *N0 Vsup Agg Val Per N1*

    Per me questo film è stato noioso[–2]

    "In my opinion this movie was boring"

- Sentences with adjectives as nouns intensifiers and downtoners:

  - *N0 Vsup Agg Int di N1*

    Una trama piena[+] di falsità[–2]

    "A plot filled with mendacity"

Predicativity is not a property necessarily possessed by a particular class of morpho-syntagmatic structures (e.g. verbs, that carry information concerning person, tense, mood, aspect); instead, it is determined by the connection between elements [Giordano and Voghera, 2008, De Mauro and Thornton, 1985].

Also on the base of their frequency in written and spoken corpora and in informal and formal speech, together with [Giordano and Voghera, 2008], we consider verbless expressions syntactically and semantically autonomous sentences, which can be coordinated, juxtaposed and that can introduce subordinate clauses, just like verbal sentences.

Negation Operator	**Sentiment Word**	**Word Polarity**	**Shifted Polarity**
non	*fantastico*	+3	-1
	bello	+2	-2
	carino	+1	-2
	scialbo	-1	+1
	brutto	-2	+1
	orribile	-3	-1

Table 4.5: Negation rules.

Among the verbless sentences available in the Italian language, we are interested here on those involving adjectives indicating appreciation (*Agg val*), e.g. *Bella questa!* "Good one!" [Meillet, 1906, De Mauro and Thornton, 1985].
Below we report a selection of customer reviews that can give an idea of the diffusion of the verbless constructions in user generated contents.

- Hotel Reviews: [*Posizione fantastica*] si raggiungono a piedi molti luoghi strategici di Londra, [*molto curato nel servizio e nel soddisfare le nostre richieste*].....[*ottimo servizio in camera*].[8]

- Videogame Reviews: [*Provato una volta*] e [*subito scartato*], [*odioso il modo in cui viene recensito*].[9]

- Smartphone Reviews: [*Utilissimo il sistema di spegnimento e riaccensione ad orari programmati*]. [*Telefonia e sensibilità OK*].[10]

In addition to these structures, a system able to recognize the negation has been introduced into the parser: negation indicators do not always change a sentence polarity in its positive or negative counterparts; they often have the effect of increasing or decreasing the sentence score. That is why we prefer to talk about valence "shifting" rather than "switching".

The general rules, concerning negation operators are summarized in the Tables 4.5.

[8]`http://www.tripadvisor.it/ShowUserReviews-g186338-d651511-r120264867-Haymarket_Hotel-London_England.html`

[9]`http://www.amazon.it/review/RK6CGST4C9LXI`

[10]`http://www.amazon.it/Alcatel-Touch-Smartphone-Dual-Italia/product-reviews/B00F621PPG?pageNumber=3`

Feature 1	Feature 2	Similarity
colazione “breakfast”	ristorante “restaurant”	0.907
colazione “breakfast”	arredamento “forniture”	0.828
colazione “breakfast”	vista “view”	0.751

Table 4.6: Example of similarity between extracted features.

Once the dependency parser has been enriched with the described structures, the Syntactic Module process the corpus and tag evaluative adjectives (A-val) and Nouns (N0) respectively as “predicates” and “feature”. Polarity values shared from SentIta by the module, are computed for each review taking into account the negation rules. The output of the parser is a list of this kind:
La colazione era fantastica!
“the breakfast was fantastic!”
[Review:H03P4,Sentiment:“fantastica”,Feature:“colazione”,Aux:essere,Polarity:“+3”]

4.2.4 Automatic Classification of Features and Reviews

Once the opinions have been extracted, the framework creates a database of evaluative adjectives and related features in which each N0 and A-val has been stored with the opinion from which has been extracted. This database has been used as input for the next step of the project which concerns the semantic expansion of the concordance. In this phase, two operation are carried out: feature categorization and review classification.

In the first step, the module calculate the semantic similarity between all features extracted from the entire corpus.

In table 4.6 is shown the value of similarity between the feature *colazione*, “feature” and other features. As shown, the similarity value is higher with the feature “restaurant” (0.9). The idea is to extract the feature with a higher value of similarity for each feature and create a graph: in this graph, each feature represent a node and is connected by a weighted arc to its most similar feature. So, a node could has several in-arcs but only one out-arc. In addition to the extracted features, the similarity is calculated also with six *Original Features*, which have been inspired by the features that *booking.it* propose in its reviews pages. These features are:

- *pulizia*, “cleaning”;

- *comodità*, “confort”;
- *ambiente*, “location”;
- *stanza*, “room”;
- *personale*, “employees”;
- *prezzo*, “price”;

If the value of each in-arc is higher than the value of out-arc, then this feature is considered as a category and features pointing on it belonging to this category. When an in-arc and an out-arc have the same values, if one of the two nodes is an original feature or has a direct connection with an original feature, then it is considered the category. For example, if the feature *pranzo*, “lunch” has a out-arc with a value of 0.94 with the word *ristorante* and an in-arc coming the same word *ristorante* with the same value, but also is connected to the original feature *comodità* with value of 0.8, also if the value is smaller than its in-arc, *comodità* will be considered the feature and the two nodes are considered classes of that category.

In the second operation we perform a review classification based on the expanded semantic network of each sentence. The first step is to collect features (N0) and sentiments (Pred) and expand the sentence semantic networks by extracting their 50 more similar words. The algorithm create a matrix of similarity values in which each row represent a sentence and each column a word.
The matrix is used in order to calculate semantic similarity between sentences. Two algorithms have been tested with this purpose: we applied a Cosine Similarity to matrix’s vectors and generate a graph in which each node corresponds to a sentence and each arc corresponds to the calculated similarity. Then, a Modularity Class algorithm has been applied to the graph in order to highlight any groups of sentences.

4.2.5 Experimental Results

As we said in section 4.2.3, the Corpus is composed by user generated reviews of hotels, videogames and smartphones and has been manually collected from the portals *ciao.it*, *amazon.it*, *tripadvisor.it* and *booking.it* that provide a user punctuation which could be adopted as control value in a evaluation task. An extract of the corpus is reported below:

Review:

Sono stato in questo albergo quasi per caso, dopo aver subito un incidente e devo ammettere che se non fosse stato per la disponibilità, la cortesia e la professionalità del personale non avrei potuto risolvere una serie di cose. L'hotel è fantastico: la colazione ottima ed il responsabile della sala colazione gentilissimo. La pulizia è impeccabile. La vista sulla città era incredibile ma la notte c'era un silenzio di tomba. Per quanto riguarda le stanze, il letto era comodissimo e la doccia fantastica!

"I happened to be in this hotel by accident and I must admit that if it hadn't been for the willingness, the kindness and the competence of the employees I couldn't have solved a number of things. The hotel is fantastic: great the breakfast and very kind the manager of the breakfast room. Cleaning was impeccable. The city view was incredible but there was a deadly silence. For what concern the rooms, the bed was very comfortable and the shower fantastic!".

After the application of the parser, the extracted features has been processed by the Semantic Module in order to find the feature categories. The Semantic Module calculate the feature similarity crossing each extracted feature and then, by selecting the high value for each feature, define a network. An example of table of similarity is shown in table 4.7.

As shown in figure 4.7 and in the relative table 4.7, the features *hotel*, *letto*, "bed" and *vista*, "view" are nodes which don't have any leaf and the algorithm can't consider there as categories. For what concern intermediate nodes as *silenzio*, "silence", *pulizia*, "cleaning" or *cortesia*, "kindness", the algorithm starts an evaluation phase as described in section4.2.4: *pulizia*, belonging to the original six features, has to be consider a feature. The feature *silenzio*, "silence", has two out-arcs, one with weight 0.970 with the node *ambiente* and the other with weight 0.954 with the node *personale* which are two original features. Ignoring the in-arc which have a lower value, its category has to be considered *ambiente*.
The words *doccia*, "shower" and *colazione* has a mutual connection with weight 0.939. Both has a connection with an original features, but the stronger connection is the one between *colazione* and *comodità*, so, *doccia* like *colazione* belong to the category *comodità*.
The final classification of the features is shown in table 4.8.

These results are far from being acceptable. While the majority of the

Source	**Target**	**Weight**
disponibilità	responsabile	0.936
hotel	silenzio	0.899
pulizia	cortesia	0.939
cortesia	personale	0.942
professionalità	personale	0.972
personale	professionalità	0.972
colazione	doccia	0.939
doccia	colazione	0.939
vista	pulizia	0.894
silenzio	personale	0.954
letto	silenzio	0.916
responsabile	disponibilità	0.936
pulizia	pulizia	1.0
silenzio	ambiente	0.970
vista	pulizia	0.894
personale	personale	1.0
doccia	personale	0.907
letto	personale	0.905
disponibilità	prezzo	0.959

Table 4.7: Similarity Graph in table format.

Category	**Features**
comodità	cortesia
	doccia
	colazione
personale	personale
	professionalità
ambiente	hotel
	silenzio
	letto
prezzo	responsabile
	disponibilità
pulizia	pulizia
	vista

Table 4.8: Extracted categories and relative features.

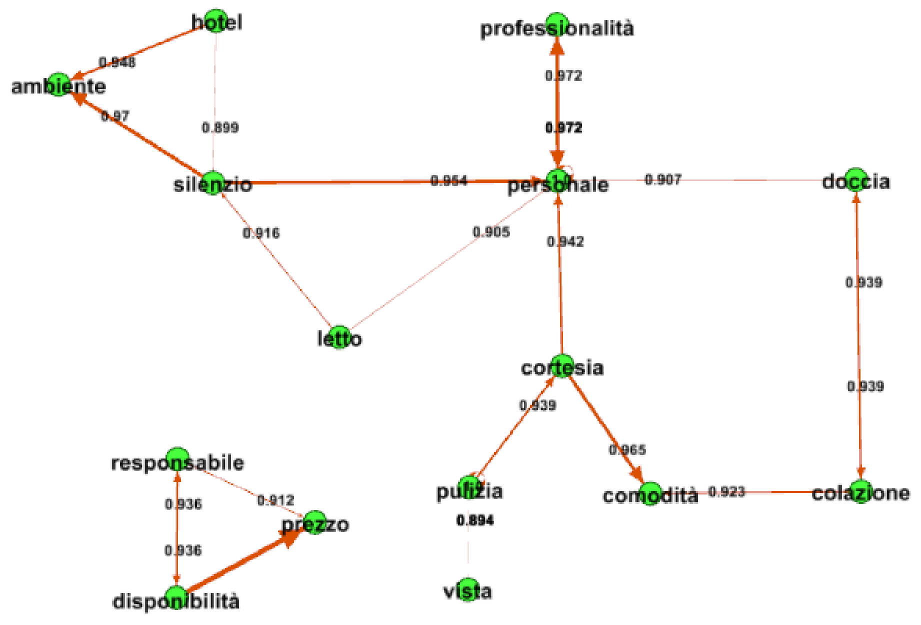


Figure 4.7: Resulting Similarity Graph

features have been correctly classified, the feature *letto*, "bed", *responsabile* "manager", *disponibilità*, "willingness" and *vista*, "view" have been wrongly classified. For what concern the feature *responsabile*, the problem could be caused by the ambiguity of the word. In the other cases find the cause of the problem is more complex.

The overall precision of this classification method reach a precision of 60% and has been calculated on a sample of 60 reviews equally distributed between videogames, hotels and smartphones reviews and positive and negative reviews.
For what concern the precision of the simple feature extraction, we obtain a precision on the entire corpus of about 90% with a recall of 72% manually calculated on a sample of 100 document.

Once the features have been classified, the parser proceeds to the tag of semantic roles, automatically producing an annotated text as the following:

Annotation:

Sono stato in questo albergo quasi per caso, dopo aver subito un incidente e devo ammettere che se non fosse stato per <BENEFIT TYPE="PREZZO" SCORE="2"> *la disponibilità* </BENEFIT>, <BENEFIT TYPE="COMODITÀ" SCORE="2"> *la cortesia* </BENEFIT> e <BENEFIT TYPE="PERSONALE" SCORE="2"> *la professionalità* </BENEFIT> del <BENEFIT TYPE="PERSONALE"> personale </BENEFIT> non avrei potuto risolvere una serie di cose. <BENEFIT TYPE="AMBIENTE" SCORE="3"> *L'hotel è fantastico* </BENEFIT>: <BENEFIT TYPE="COMODITÀ" SCORE="3"> *La colazione ottima* </BENEFIT> ed <BENEFIT TYPE="PREZZO" SCORE="3"> *il responsabile della sala colazione gentilissimo* </BENEFIT>. <BENEFIT SCORE="3" TYPE="PULIZIA"> *La pulizia è impeccabile* </BENEFIT>.<BENEFIT SCORE="3" TYPE="PULIZIA"> *La vista sulla città era incredibile* </BENEFIT> ma <BENEFIT SCORE="2" TYPE="AMBIENTE"> *la notte c'era un silenzio di tomba* </BENEFIT>. Per quanto riguarda le stanze, <BENEFIT SCORE="3" TYPE="AMBIENTE"> *il letto era comodissimo* </BENEFIT> e <BENEFIT SCORE="3" TYPE="COMODITÀ"> *la doccia fantastica!* </BENEFIT>

The dependency parser included into the LG-Starship Framework also includes a visualization tool which show the graph structure of the analyzed sentence in a fashion similar to the one presented in the figure 4.8.

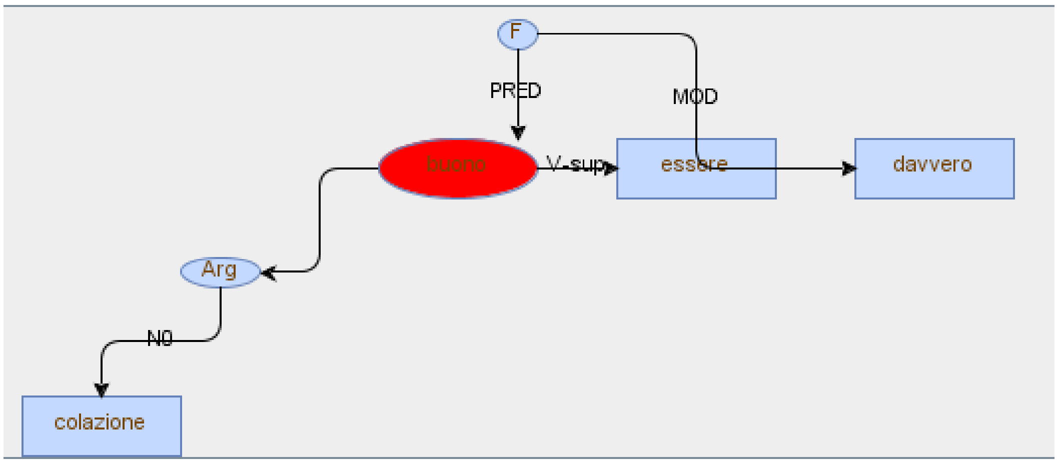


Figure 4.8: Visualization of the dependency graph of the sentence "la colazione era davvero buona"

Differently from what happens with the standard parser, in the example is shown a dependency graph that represent a support verb sentence which anchor is the evaluative adjective. In the sentence 15

(15) *La colazione era davvero buona.*
"the breakfast was very good"

the predicative element was the adjective *buono*. The verb *essere*, "to be", is tagged as support verb (*V-sup*) and the adverb *davvero*, "very" is considered a modifier of the predicate. The argument of the predicate (N0) is *colazione*, "breakfast", and is considered a feature.

Once the parser tag the text and the framework create a database of features, we perform a semantic expansion of each review in order to create a semantic network. This semantic network has two main objectives: the first output of this semantic expansion is a big word semantic graph which contains the terms related with a specific text. The graph, which is shown in the figure 4.9, emphasizes nodes with a high weighted degree which, in the majority of cases represent adjectives.

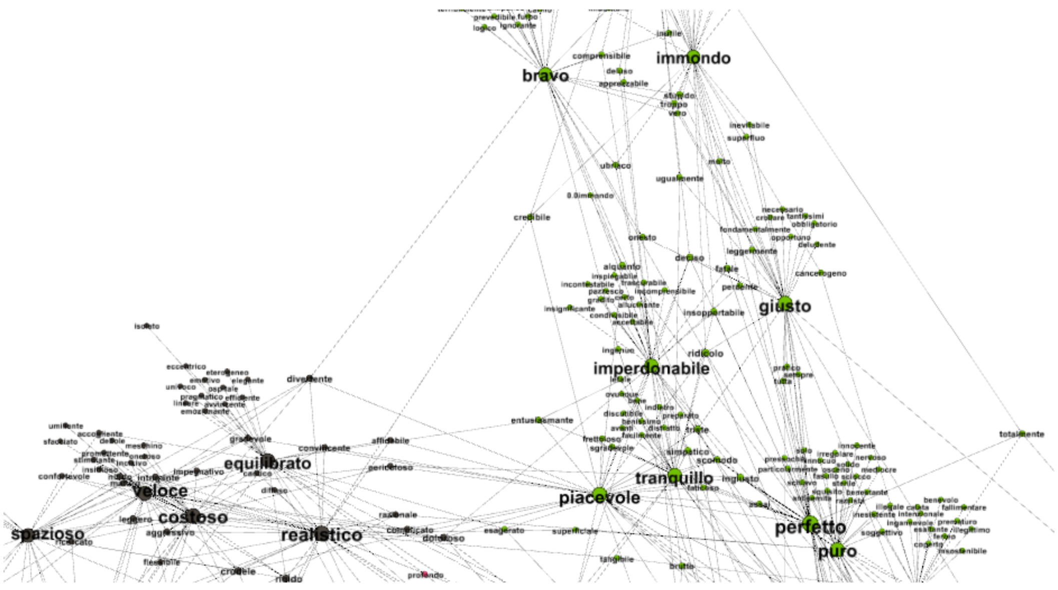


Figure 4.9: Extract of the word semantic Graph of Hotel Reviews

The graph shown in the fig. 4.9 has been calculated expanded the features extracted by five hotel review and five videogames review, with their 50 more similar words.

In a second time, the same semantic expansion algorithm has been applied to a bigger corpus of 150 reviews of hotels, videogames and smartphones divided into 15 groups: for each feature of each group we extracted the more similar 50 words and we created a large similarity matrix in which rows represent groups and columns represent the extracted words.

From the Similarity Matrix, applying a Cosine Similarity Algorithm, we calculate a distance between group vectors. A graph has been generated using *Gephi* and processed with the Modularity Class algorithm which partitioned the graph in base of similarity weights and found internal communities. Nodes are labeled with the number of the file the initial of the product ("H" for hotels, "V" for videogames and "C" for cellulars or smartphones), and the initial of the general polarity ("N" for negative and "P" for positive). The result is shown in the figure 4.10.

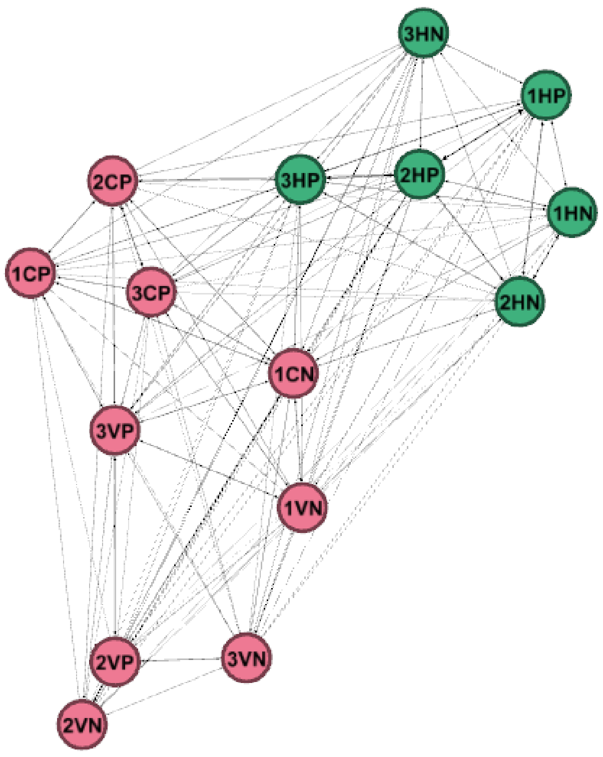


Figure 4.10: Network communities calculated by the Modularity Class Algorithm on the Similarity Matrix

Looking at the figure, there was a big community colored in red, and a second community in green. For what concern the green community, it all texts that contain hotel reviews. On the other hand, the red community includes all smartphone and videogames texts.

Considering that videogames and smartphones belonging to similar domains, both related with technology or computer science, the results, in particular with hotel texts were encouraging.

In table 4.9) the evaluation of the results of the Sentiment analysis at a Document-level is shown. We calculated the precision twice by considering in a first case as true positive the reviews correctly classified by the Semantic Module on the base of their polarity and in a second case by considering as

true positive the documents that received by the Framework exactly the same stars specified by the Opinion Holder.
In detail, in the *Polarity only* row the True Positives are the documents that have been correctly classified by the Module, with a polarity attribution that corresponds to the one specified by the Opinion Holder.
In the *Intensity also* row the True Positives are the document that received by the module exactly the same stars specified by the Opinion Holder.

Precision	**Smartphones**	**Hotels**	**Videogames**	**Average**
Polarity only	72.0	63.0	72.0	74.0
Intensity also	45.0	25.0	34.0	36.3

Table 4.9: Precision measure in document-level classification

As we can see, the latter seem to have a very low precision, but upon a deeper analysis we discovered that is really common for the Opinion Holders to write in their reviews texts that do not perfectly correspond to the stars they specified. That increases the importance of a this kind of analysis, that does not stop on the structured data, but enters the semantic dimension of texts written in natural language.

Recall	**Smartphones**	**Hotels**	**Videogames**	**Average**
	98.6	98.9	91.2	97.5

Table 4.10: Recall in both the sentence-level and the document level classifications

The document-level Recall has been automatically checked with the LG-Framework, by considering as true positive all the opinionated documents which contained at least one appropriate sentiment indicator. So, the documents in which the parser did not annotate any pattern were considered false negatives. Because we assumed that all the texts of our corpus were opinionated, we considered as false negatives also the cases in which the value of the document was 0.

Taking the F-measure into account, the best results were achieved with the smartphone's domain (77.0%) in the sentence-level task and with the hotel's dataset (94.8%) into the document-level performance.

As future work, the idea is to improve the feature classification precision by applying parallel alternative strategies in order to correct the errors of the experimental method based on semantic similarity. In addition, since Sentiment Analysis is strongly domain-dependent, in future will be possible to includes in the project new domains such as politics, cars, books or movies. Once the method will reach a satisfactory precision level, it will be possible to add a new Panel to the interface of the LG-Starship Framework which will be entirely dedicated to the Sentiment Analysis.

Chapter 5

Conclusion and Future Work

In this thesis we presented a project of an hybrid linguistic framework called LG-Starship Framework which is based on the idea that hybrid models could offer better performances and more variated results than statistical or rule-based model. The Framework comes from the lack of this kind of free resources for the Italian language since, presently, the vast majority of linguistic softwares are optimized for English language or they are not free.
The LG-Starship Framework is also designed to take advantage from the large amount of electronic linguistic resources developed at the Department of Political, Social and Communication Science of the University of Salerno in the last years. Resources that consist on several electronic dictionaries, verb tables, finite state grammars. A unique set of resources that could has a strong impact on the linguistic community because of its completeness and depth. This set of resources contains:

- a vast dictionary (over 1 million lemmas) of simple flexed words, tagged with syntactic and semantic informations of disparate kinds.
- a complete dictionary of Compound Nouns of Italian, classified both in semantic domains and syntactic structure.
- Predicate Tables (over 4000 italian verbs) classified and complete of selected structures.
- several sub-dictionaries (Proper Nouns, Concrete Nouns, etc.)
- a list of Finite State Automata for recognition of temporal expressions, proper nouns, movement verbs etc.
- morphological dictionaries of medical-domain confixes correlated with combinatorial rules for recognition and classification of medical terms.

At the moment not all these resources have encountered a use on the presented Framework, but we made a large use of electronic dictionaries and predicate tables.
As theoretical background, the project is founded on the Lexicon-Grammar Theory, presented by Mourice Gross in the '70. The theory which take distance from the Chomsky's Trasformational-Generative Grammar (TGG), is largely based upon the work of Zellig Harris inheriting the idea that the focus of a sentence is the predicate, intended as a verb or a sequence of word which have the same function.
Gross starts a vast work of compilation of linguistic data for French language which has been extended for Italian Language by the group leaded by Elia at the University of Salerno. The results are the resources mentioned above from

which, they have a particular importance the LG Tables, a set of Predicate Tables which includes over 4000 Italian Verbal Uses. LG-Tables could be considered a powerful tool, specially because of their tabular format and of their content: in fact, they contain a list of feature and properties for each predicate which define exactly every structure it could select. The tables have a binary format and each verb, classified with a criteria of syntactic similarity, is included into a class of verbs that share some properties.
The LG-Starship Framework take advantage from these tables which can be automatically converted into a database of verbs, in order to construct a rule-based dependency parser which can represent a sentence as a graph structured as the LG theory illustrate.
The Framework contains the following modules:

- A Preprocessing Module that perform preprocessing task as text normalization, cleaning and stop-words remove.
- the Mr. Ling Module which is based on the electronic dictionaries developed at the University of Salerno and perform the Part-Of-Speech tagging and Lemmatization tasks.
- The Statistic Module which perform basic and advanced statistic measures on the text as Term Frequency, TF/IDF or N-grams extraction.
- The Semantic Module that is based on a distributional semantics algorithm called Hyperspace Analogue to Language, permits to extract similar words, calculate similarity between two words and generate semantic networks.
- The Syntactic Module which contains a dependency parser entirely based on the LG-Tables.

The Framework can be considered hybrid also for other reasons. Its modules will be released as a Java .jar package which can be used in external and more complex projects, but also it will be distributed as an Interface software which can be useful for those users who have not programming skills but are interested to Natural Language Processing or Text Analysis. The Interface version will includes all the modules and some visualization tools in order to give to the users the opportunity to generate charts or simply produce data for external uses.

For what concern the obtained results, the Framework, which is only at a Beta Version, obtain discrete results:

- The Preprocessing Module obtains good results in terms of time of processing and precision, in particular with short and medium texts. The time of processing, which exponentially grows with the grow of the texts, can be improved, but is not so different from other tools.
- The Mr. Ling Module reaches good results in terms of precision (about 92% for the POS Tagging and Lemmatization) and time, although it employs more time that *TreeTagger* which obtains similar values of accuracy.
- The Statistic Module calculate Term Frequency and TF/IDF in a very short time (less than 1 second for a 10.000 tokens' text), but the results of the N-Grams extractor may be improved.
- The Semantic Module precision is closely linked to the Matrix on which is based. Due to hardware limitations, the used Matrix has been generated from a text of 45 millions of tokens and reduced to the first 200 columns.
- The Syntactic Module received a partial evaluation because it is incomplete at the moment. The completion of the module is subject to the completion of the LG Verb Resource, a machine-readable version of LG-Tables which is under construction at the University of Salerno. The results obtained so far are goods in terms of time and precision, but at the moment the parser suffer the same problems than other existing resources.

In the chapter 15 we presented two projects in which the Framework has been largely tested: in the first experiment has been performed a statistical study of the language of Rap Music in Italy through the analysis of a great corpus of Rap Song lyrics downloaded from on line databases of user generated lyrics. The project make a large use of Preprocessing Module, Mr.Ling Module, Statistic Module and Semantic Module. It demonstrated how the framework treat a large corpus producing data resources which can be exported and adapted to additional external analysis. In fact, the data produced by the framework has been processed and combined in order to create a web resources which permits to make a "distant reading" of the Italian Rap Music by exploiting different visualization techniques and tools.
The second experiment is a Feature-Based Sentiment Analysis and Opinion Mining project of user reviews. The integration of a large domain database of linguistic resources in Italian of Sentiment Analysis developed in the past years by the Department of Politic, Social and Communication Science of the University of Salerno, which consists of polarized dictionaries of Verbs,

Adjectives, Adverbs and Nouns, demonstrates how the Framework can be integrated with external resources in order to broaden its field of action. In this project, except for the Statistic Module, every module of the Framework was involved.

Even though the results are encouraging, the Framework is far from being considered complete and for a “1.0” version a hard work is expected. While the Preprocessing Module and the Mr.Ling Module could be released in the present state, other modules need improvements: the N-grams extraction of Statistic Module must be improved and tested and more functions as the implementation of a Pointwise Mutual Information algorithm can be added; the Semantic Module needs a reorganization of its methods with the implementation of more complex forms of querying, but, primarily, it needs a more complete matrix, built on a larger and more heterogeneous corpus.

For what concern the Syntactic Module, the parser is actually waiting for the completion of the LG-Tables conversion, but it needs the implementation of more complex strategies in order to avoid errors on noun or verb complements recognition. The idea is, once the LG Verb Resource is complete, to implement an hybrid system which start with the essential sentence structure given by the predicates as the LG theory indicate, and continue the analysis by exploiting a State Of the Art statistic or probabilistic method.
In conclusion, the Graphical interface of the program will be improved and made more attractive and modern so that it can be released as a free software. In addition, some of the modules can be inserted in a web-interface and published for free on-line utilization.

Bibliography

G. Adda, M. Adda-Decker, J.-L. Gauvain, and L. Lamel. Text normalization and speech recognition in french. *training*, 3:4–0, 1997. (Cited on page 43.)

A. B. Alencar, M. C. F. de Oliveira, and F. V. Paulovich. Seeing beyond reading: a survey on visual text analytics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(6):476–492, 2012. (Cited on page 98.)

H. S. Alim, A. Ibrahim, and A. Pennycook. *Global linguistic flows: Hip hop cultures, youth identities, and the politics of language.* Routledge, 2008. (Cited on page 92.)

J. Androutsopoulos and A. Scholz. Spaghetti funk: appropriations of hip-hop culture and rap music in europe. *Popular Music and Society*, 26(4):463–479, 2003. (Cited on page 92.)

G. Attardi and F. Dell'Orletta. Reverse revision and linear tree combination for dependency parsing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 261–264. Association for Computational Linguistics, 2009. (Cited on pages 27 and 28.)

G. Attardi, A. Fuschetto, F. Tamberi, M. Simi, and E. M. Vecchi. Experiments in tagger combination: arbitrating, guessing, correcting, suggesting. In *Proc. of Workshop Evalita*, page 10, 2009. (Cited on page 17.)

P. Attolino. Stile ostile. rap e politica, 2003. (Cited on page 92.)

P. Attolino. Iconicity in rap music the challenge of an anti-language. 2012. (Cited on pages 91 and 92.)

C. Audet, C. Burgess, et al. Using a high-dimensional memory model to evaluate the properties of abstract and concrete words. In *Proceedings of the cognitive science society*, pages 37–42. Citeseer, 1999. (Cited on page 24.)

L. Azzopardi, M. Girolami, and M. Crowe. Probabilistic hyperspace analogue to language. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 575–576. ACM, 2005. (Cited on page 25.)

R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999. (Cited on page 20.)

V. Balakrishnan and E. Lloyd-Yemoh. Stemming and lemmatization: a comparison of retrieval performances. *Lecture Notes on Software Engineering*, 2(3):262, 2014. (Cited on page 16.)

T. Baldwin. A resource for evaluating the deep lexical acquisition of english verb-particle constructions. In *Proceedings of the LREC Workshop Towards a Shared Task for Multiword Expressions (MWE 2008)*, pages 1–2, 2008. (Cited on page 25.)

M. Baroni, S. Bernardini, F. Comastri, L. Piccioni, A. Volpi, G. Aston, and M. Mazzoleni. Introducing the la repubblica corpus: A large, annotated, tei (xml)-compliant corpus of newspaper italian. *issues*, 2:5–163, 2004. (Cited on page 97.)

M. Baroni, S. Bernardini, A. Ferraresi, and E. Zanchetta. The wacky wide web: a collection of very large linguistically processed web-crawled corpora. *Language resources and evaluation*, 43(3):209–226, 2009. (Cited on page 97.)

D. Baur, B. Steinmayr, and A. Butz. Songwords: Exploring music collections through lyrics. In *ISMIR*, pages 531–536, 2010. (Cited on page 98.)

J. R. Bellegarda, J. W. Butzberger, Y.-L. Chow, N. B. Coccaro, and D. Naik. A novel word clustering algorithm based on latent semantic analysis. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 172–175. IEEE, 1996. (Cited on page 25.)

P. M. Bertinetto, C. Burani, A. Laudanna, L. Marconi, D. Ratti, C. Rolando, and A. M. Thornton. Corpus e lessico di frequenza dell'italiano scritto (colfis). *Scuola Normale Superiore di Pisa*, 2005. (Cited on page 25.)

S. Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics, 2006. (Cited on pages 8 and 12.)

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003. (Cited on page 21.)

V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008. (Cited on page 82.)

L. Bloomfield. Language, holt, new york. In *Edizione italiana (1974) Il linguaggio, Il Saggiatore, Milano*. 1933. (Cited on page 29.)

B. Bohnet. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd international conference on computational linguistics*, pages 89–97. Association for Computational Linguistics, 2010. (Cited on page 28.)

F. Bond and R. Foster. Linking and extending an open multilingual wordnet. In *ACL (1)*, pages 1352–1362, 2013. (Cited on page 94.)

P. Boullier. Guided earley parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT'03)*, pages 43–54, 2003. (Cited on page 27.)

D. Bounie, M. Bourreau, M. Gensollen, and P. Waelbroeck. The effect of online customer reviews on purchasing decisions: The case of video games. In *Retrieved July*, volume 8, page 2009. Citeseer, 2005. (Cited on page 107.)

A. Bradley. *Book of rhymes: The poetics of hip hop*. Basic Books, 2009. (Cited on page 92.)

T. Brants. Tnt: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231. Association for Computational Linguistics, 2000. (Cited on page 14.)

R. Brath and E. Banissi. Using text in visualizations for micro/macro readings. In *IUI Workshop on Visual Text Analytics*, 2015. (Cited on page 98.)

E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the workshop on Speech and Natural Language*, pages 112–116. Association for Computational Linguistics, 1992. (Cited on page 14.)

R. C. Bunescu and M. Pasca. Using encyclopedic knowledge for named entity disambiguation. In *EACL*, volume 6, pages 9–16, 2006. (Cited on page 91.)

C. Burgess. From simple associations to the building blocks of language: Modeling meaning in memory with the hal model. *Behavior Research Methods, Instruments, & Computers*, 30(2):188–198, 1998. (Cited on page 24.)

C. Burgess. Representing and resolving semantic ambiguity: A contribution from high-dimensional memory modeling. 2001. (Cited on page 24.)

G. Carenini, R. T. Ng, and E. Zwart. Extracting knowledge from evaluative text. In *Proceedings of the 3rd international conference on Knowledge capture*, pages 11–18. ACM, 2005. (Cited on page 109.)

D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750, 2014. (Cited on page 28.)

J. A. Chevalier and D. Mayzlin. The effect of word of mouth on sales: Online book reviews. In *Journal of marketing research*, volume 43, pages 345–354. American Marketing Association, 2006. (Cited on page 107.)

J. D. Choi. Dynamic feature induction: The last gist to the state-of-the-art. In *Proceedings of NAACL-HLT*, pages 271–281, 2016. (Cited on page 15.)

N. Chomsky. *Aspects of the Theory of Syntax*. Number 11. MIT press, 1965. (Cited on page 28.)

G. Chrupała, G. Dinu, and J. Van Genabith. Learning morphology with morfette. 2008. (Cited on page 16.)

K. W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the second conference on Applied natural language processing*, pages 136–143. Association for Computational Linguistics, 1988. (Cited on page 13.)

A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004. (Cited on page 82.)

M. Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pages 16–23. Association for Computational Linguistics, 1997. (Cited on page 27.)

M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002. (Cited on page 15.)

M. Collins, L. Ramshaw, J. Hajič, and C. Tillmann. A statistical parser for czech. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 505–512. Association for Computational Linguistics, 1999. (Cited on page 28.)

M. Constant, I. Tellier, D. Duchier, Y. Dupont, A. Sigogne, and S. Billot. Intégrer des connaissances linguistiques dans un crf: application à l'apprentissage d'un segmenteur-étiqueteur du français. In *TALN*, volume 1, page 321, 2011. (Cited on page 16.)

H. Cunningham. Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002. (Cited on page 7.)

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. Gate: an architecture for development of robust hlt applications. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 168–175. Association for Computational Linguistics, 2002. (Cited on page 7.)

C. Cutler. Hip-hop language in sociolinguistics and beyond. *Language and linguistics compass*, 1(5):519–538, 2007. (Cited on pages 91 and 92.)

D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 133–140. Association for Computational Linguistics, 1992. (Cited on page 14.)

E. D'Agostino. *Analisi del discorso: metodi descrittivi dell'italiano d'uso.* Loffredo, 1992. (Cited on pages 29, 30 and 31.)

E. D'Agostino. Grammatiche lessicalmente esaustive delle passioni il caso dell'io collerico. le forme nominali. In *Quaderns d'Italià*, pages 149–169, 2005. (Cited on page 29.)

E. D'Agostino, G. De Bueriis, A. Cicalese, M. Monteleone, D. Vellutino, S. Messina, A. Langella, S. Santonicola, F. Longobardi, and D. Guglielmo. Lexicon-grammar classifications. or better: to get rid of anguish. In *26th International Conference on Lexis and Grammar (LGC'07)*, 2007. (Cited on page 29.)

T. De Mauro and A. M. Thornton. La predicazione: teoria e applicazione all'italiano. In *Sintassi e morfologia della lingua italiana d'uso: teorie ed applicazioni descrittive*, pages 487–519. 1985. (Cited on pages 112 and 113.)

S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990. (Cited on page 25.)

F. Dell?Orletta. Ensemble system for part-of-speech tagging. *Proceedings of EVALITA*, 9:1–8, 2009. (Cited on page 17.)

P. Denis, B. Sagot, et al. Coupling an annotated corpus and a morphosyntactic lexicon for state-of-the-art pos tagging with less human effort. In *PACLIC*, pages 110–119, 2009. (Cited on page 15.)

P. Di Cristo. Mtseg: The multext multilingual segmenter tools, multext deliverable msg 1, version 1.3. 1, cnrs, aix-en-provence, 1996. (Cited on page 12.)

W. Duan, B. Gu, and A. B. Whinston. The dynamics of online word-of-mouth and product sales—an empirical investigation of the movie industry. In *Journal of retailing*, volume 84, pages 233–242. Elsevier, 2008. (Cited on page 107.)

S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and R. Harshman. Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–285. ACM, 1988. (Cited on page 25.)

I. A. El-Khair. Effects of stop words elimination for arabic information retrieval: a comparative study. *International Journal of Computing & Information Sciences*, 4(3):119–133, 2006. (Cited on page 43.)

A. Elia. *Le verbe italien. Les complétives dans les phrases à un complément.* Fasano di Puglia: Schena-Nizet, 1984. (Cited on pages 29, 63, 64 and 67.)

A. Elia. Dizionari elettronici e applicazioni informatiche. In *JADT*, 1995. (Cited on page 47.)

A. Elia. On lexical, semantic and syntactic granularity of italian verbs. In *Penser le Lexique Grammaire: Perspectives Actuelles, Honoré Champion, Paris*, pages 277–286. 2013. (Cited on page 63.)

A. Elia. Lessico e sintassi tra tempo e massa parlante. In *Marchese M.P., Nocentini A., Il lessico nella teoria e nella storia linguistica*, pages 15–47. Edizioni il Calamo, 2014. (Cited on pages 29, 31 and 63.)

A. Elia, M. Martinelli, and E. D'Agostino. *Lessico e Strutture sintattiche. Introduzione alla sintassi del verbo italiano.* Napoli: Liguori, 1981. (Cited on pages 29, 30, 31, 63 and 111.)

A. Elia, F. Marano, M. Monteleone, S. Sabatino, and D. Vellutino. Strutture lessicali delle informazioni comunitarie all'interno di domini specialistici. In *Statistical Analysis of Textual Data, Proceedings of 10th International Conferences" Journées D'Analyse Statistique des Données Textuelles" Roma, Università" La Sapienza*, pages 9–11, 2010. (Cited on page 47.)

R. R. Favretti, F. Tamburini, and C. De Santis. Coris/codis: A corpus of written italian based on a defined and a dynamic model. *A Rainbow of Corpora: Corpus Linguistics and the Languages of the World. Munich: Lincom-Europa*, 2002. (Cited on page 17.)

A. Filippone and L. Papini. La parola e il suo potere: Il linguaggio del rap italiano. *Rassegna italiana di linguistica applicata*, 33(3):71–86, 2002. (Cited on page 92.)

J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005. (Cited on page 10.)

P. W. Foltz, W. Kintsch, and T. K. Landauer. The measurement of textual coherence with latent semantic analysis. *Discourse processes*, 25(2-3):285–307, 1998. (Cited on page 25.)

M. Forman and M. A. Neal. *That's the joint!: the hip-hop studies reader.* Psychology Press, 2004. (Cited on page 91.)

G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973. (Cited on page 12.)

B. Fortuna, M. Grobelnik, and D. Mladenic. Visualization of text document corpus. *Informatica*, 29(4), 2005. (Cited on page 98.)

W. Francis and H. Kucera. Frequency analysis of english usage. 1982. (Cited on page 13.)

O. Frunza. A trainable tokenizer, solution for multilingual texts and compound expression tokenization. In *LREC*, 2008. (Cited on page 12.)

E. Gabrilovich and S. Markovitch. Overcoming the brittleness bottleneck using wikipedia: Enhancing text categorization with encyclopedic knowledge. In *AAAI*, volume 6, pages 1301–1306, 2006. (Cited on page 91.)

L. Gatti and M. Guerini. Assessing sentiment strength in words prior polarities. In *arXiv preprint arXiv:1212.4315*. 2012. (Cited on page 108.)

J. Giménez and L. Marquez. Svmtool: A general pos tagger generator based on support vector machines. In *In Proceedings of the 4th International Conference on Language Resources and Evaluation*. Citeseer, 2004. (Cited on page 15.)

R. Giordano and M. Voghera. Frasi senza verbo: il contributo della prosodia. In *Sintassi storica e sincronica dell'italiano, Atti del Conv. Intern. SILFI, Basilea*. 2008. (Cited on page 112.)

M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002. (Cited on page 82.)

G. Graffi and S. Scalise. Le lingue e il linguaggio. *Il Mulino, Bologna*, 2002. (Cited on pages 11 and 47.)

J. Grana, M. A. Alonso, and M. Vilares. A common solution for tokenization and part-of-speech tagging. In *International Conference on Text, Speech and Dialogue*, pages 3–10. Springer, 2002. (Cited on page 12.)

B. B. Greene and G. M. Rubin. *Automatic grammatical tagging of English.* Department of Linguistics, Brown University, 1971. (Cited on page 13.)

G. Grefenstette and P. Tapanainen. What is a word, what is a sentence?: problems of tokenisation. 1994. (Cited on pages 11 and 12.)

M. Gross. *Transformational Analysis of French Verbal Constructions.* University of Pennsylvania, 1971. (Cited on page 28.)

M. Gross. *Méthodes en syntaxe.* Hermann, 1975. (Cited on page 28.)

M. Gross. On the failure of generative grammar. In *Language*, pages 859–885. JSTOR, 1979. (Cited on page 59.)

M. Gross. The argument structure of elementary sentences. In *Language Research*, volume 28, pages 699–716. 1992. (Cited on page 29.)

M. Gross. Une grammaire locale de l'expression des sentiments. In *Langue française*, pages 70–87. JSTOR, 1995. (Cited on page 109.)

M. Gross. Les verbes supports d'adjectifs et le passif. In *Langages*, volume 30, pages 8–18. Armand Colin, 1996. (Cited on page 111.)

Y. Gutiérrez, S. Vázquez, and A. Montoyo. Sentiment classification using semantic features extracted from WordNet-based resources. In *Proceedings of the 2nd Workshop on Computational Approaches to Subjectivity and Sentiment Analysis*, pages 139–145. Association for Computational Linguistics, 2011. (Cited on page 109.)

J. Hajič, J. Raab, M. Spousta, et al. Semi-supervised training for the averaged perceptron pos tagger. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 763–771. Association for Computational Linguistics, 2009. (Cited on page 15.)

J. Hall, J. Nilsson, and J. Nivre. Single malt or blended? a study in multilingual parser optimization. In *Trends in Parsing Technology*, pages 19–33. Springer, 2010. (Cited on page 27.)

S. M. Harabagiu, M. A. Paşca, and S. J. Maiorano. Experiments with open-domain textual question answering. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 292–298. Association for Computational Linguistics, 2000. (Cited on page 91.)

Z. Harris. *Language and information.* Columbia University Press, 1988. (Cited on page 30.)

Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954. (Cited on page 23.)

Z. S. Harris. Transformations in linguistic structure. In *Proceedings of the American Philosophical Society*, pages 418–422, 1964. (Cited on page 30.)

Z. S. Harris. Discourse analysis. In *Papers in structural and transformational linguistics*, pages 313–347. 1970. (Cited on page 29.)

M. Herlihy and N. Shavit. The art of multiprocessor programming. In *PODC*, volume 6, pages 1–2, 2006. (Cited on page 37.)

H. Hirjee and D. G. Brown. Automatic detection of internal and imperfect rhymes in rap lyrics. In *ISMIR*, pages 711–716, 2009. (Cited on page 93.)

H. Hirjee and D. G. Brown. Rhyme analyzer: An analysis tool for rap lyrics. In *Proceedings of the 11th International Society for Music Information Retrieval Conference.* Citeseer, 2010. (Cited on page 93.)

T. Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 289–296. Morgan Kaufmann Publishers Inc., 1999. (Cited on page 25.)

M. Hu and B. Liu. Mining opinion features in customer reviews. In *AAAI*, volume 4, pages 755–760, 2004. (Cited on page 109.)

M. Hu and B. Liu. Opinion feature extraction using class sequential rules. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, pages 61–66, 2006. (Cited on page 109.)

X. Hu, J. S. Downie, and A. F. Ehmann. Lyric text mining in music mood classification. *American music*, 183(5,049):2–209, 2009. (Cited on page 93.)

A. Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56, 2008. (Cited on page 98.)

Z. Huang, W. Xu, and K. Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015. (Cited on page 15.)

A. K. Ingason, S. Helgadóttir, H. Loftsson, and E. Rögnvaldsson. A mixed method lemmatization algorithm using a hierarchy of linguistic identities (holi). In *Advances in Natural Language Processing*, pages 205–216. Springer, 2008. (Cited on page 16.)

J. Kanis and L. Müller. Automatic lemmatizer construction with focus on oov words lemmatization. In *International Conference on Text, Speech and Dialogue*, pages 132–139. Springer, 2005. (Cited on page 16.)

K. Khan, B. Baharudin, and A. Khan. Identifying product features from customer reviews using hybrid dependency patterns. Citeseer. (Cited on page 109.)

F. Kleedorfer, P. Knees, and T. Pohle. Oh oh oh whoah! towards automatic topic detection in song lyrics. In *ISMIR*, pages 287–292, 2008. (Cited on page 93.)

R. R. Korfhage. Information storage and retrieval. 2008. (Cited on page 43.)

K. Kucher and A. Kerren. Text visualization browser: A visual survey of text visualization techniques. *Poster Abstracts of IEEE VIS*, 2014, 2014. (Cited on page 98.)

A. Labrecque. *Computer Visualization of Song Lyrics.* PhD thesis, WORCESTER POLYTECHNIC INSTITUTE, 2009. (Cited on page 98.)

T. K. Landauer and S. T. Dumais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2):211, 1997. (Cited on pages 23 and 25.)

J. C. Lena. Social context and musical content of rap music, 1979–1995. *Social Forces*, 85(1):479–495, 2006. (Cited on page 91.)

J. Lewis and W. Loftus. Java software solutions foundations of programming design. *Pearson Education Inc*, 2008. (Cited on page 37.)

B. Liu. Sentiment analysis and subjectivity. In *Handbook of natural language processing*, volume 2, pages 627–666. Chapman & Hall Goshen, CT, 2010. (Cited on page 108.)

S. Loria. Textblob: simplified text processing. *Secondary TextBlob: Simplified Text Processing*, 2014. (Cited on page 8.)

T. Lucien. Eléments de syntaxe structurale. In *Kliencksieck, Paris.* 1959. (Cited on pages 26 and 29.)

H. P. Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4): 309–317, 1957. (Cited on page 19.)

K. Lund and C. Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208, 1996. (Cited on page 24.)

V. Lyding, E. Stemle, C. Borghetti, M. Brunello, S. Castagnoli, F. Dell'Orletta, H. Dittmann, A. Lenci, and V. Pirrelli. The paisa corpus of italian web texts. In *Proceedings of the 9th Web as Corpus Workshop (WaC-9)*, pages 36–43, 2014. (Cited on page 46.)

J. P. Mahedero, Á. MartÍnez, P. Cano, M. Koppenberger, and F. Gouyon. Natural language processing of lyrics. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 475–478. ACM, 2005. (Cited on page 93.)

A. Maisto and S. Pelosi. Feature-based customer review summarization. In *On the Move to Meaningful Internet Systems: OTM 2014 Workshops*, pages 299–308. Springer, 2014a. (Cited on page 110.)

A. Maisto and S. Pelosi. A lexicon-based approach to sentiment analysis. the italian module for nooj. In *Proceedings of the International Nooj 2014 Conference, University of Sassari, Italy.* Cambridge Scholar Publishing, 2014b. (Cited on page 110.)

A. Maisto and S. Pelosi. A lexicon-based approach to sentiment analysis. the italian module for nooj. In *Proceedings of the International Nooj 2014 Conference, University of Sassari, Italy*, 2014c. (Cited on page 12.)

E. Malmi, P. Takala, H. Toivonen, T. Raiko, and A. Gionis. Dopelearning: A computational approach to rap lyrics generation. *arXiv preprint arXiv:1505.04771*, 2015. (Cited on page 93.)

C. D. Manning. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 171–189. Springer, 2011. (Cited on page 15.)

C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60, 2014. (Cited on pages 10 and 28.)

M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119. Association for Computational Linguistics, 1994. (Cited on page 10.)

J. H. Martin and D. Jurafsky. Speech and language processing. *International Edition*, 710, 2000. (Cited on pages 11 and 26.)

J. Mathieu and V. Tommaso. Forceatlas2, a graph layout algorithm for handy network visualization. *Webatlas. fr*, 29, 2011. (Cited on page 82.)

R. McDonald, K. Lerman, and F. Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 216–220. Association for Computational Linguistics, 2006. (Cited on page 28.)

A. Meillet. *La Phrase nominale en indoeuropéen.* Société de linguistique, 1906. (Cited on page 113.)

A. Meunier. La sémantique locative de certaines structures: N0 être adj. In *Revue québécoise de linguistique*, volume 13, pages 95–121. Université du Québec à Montréal, 1984. (Cited on page 111.)

A. Meunier. Une construction complexe *N0hum être Adj de V0-inf W* caracteéristique de certains adjectifs á sujet humain. In *Langages*, pages 12–44. JSTOR, 1999. (Cited on page 111.)

T. Morton, J. Kottmann, J. Baldridge, and G. Bierner. Opennlp: A java-based nlp toolkit, 2005. (Cited on page 17.)

M. Nakayama, N. Sutcliffe, and Y. Wan. Has the web transformed experience goods into search goods? In *Electronic Markets*, volume 20, pages 251–262. Springer, 2010. (Cited on page 107.)

V. Nastase and M. Strube. Decoding wikipedia categories for knowledge acquisition. In *AAAI*, volume 8, pages 1219–1224, 2008. (Cited on page 91.)

R. Navigli and S. P. Ponzetto. Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012. (Cited on page 94.)

P. Nelson. Information and consumer behavior. In *The Journal of Political Economy*, pages 311–329. JSTOR, 1970. (Cited on page 107.)

M. E. Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006a. (Cited on page 103.)

M. E. Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006b. (Cited on page 82.)

J. Nivre and R. T. McDonald. Integrating graph-based and transition-based dependency parsers. In *ACL*, pages 950–958, 2008. (Cited on page 27.)

J. Nivre and J. Nilsson. Multiword units in syntactic parsing. *Proceedings of Methodologies and Evaluation of Multiword Units in Real-World Applications (MEMURA)*, 2004. (Cited on page 27.)

J. Nivre, J. Hall, and J. Nilsson. Memory-based dependency parsing. (Cited on page 27.)

T. Obrębski. Dependency parsing using dependency graph. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03).(2003) 217–218*. (Cited on page 27.)

J. Oh. Text visualization of song lyrics. *Center for Computer Research in Music and Acoustics, Stanford University*, 2010. (Cited on page 98.)

A. OpenNLP. a machine learning based toolkit for the processing of natural language text. *URL http://opennlp. apache. org (Last accessed: 2013-06-18)*. (Cited on page 10.)

C. E. Osgood. The nature and measurement of meaning. In *Psychological bulletin*, volume 49, page 197. American Psychological Association, 1952. (Cited on page 108.)

H. Osumare. *The Africanist aesthetic in global hip-hop: Power moves.* Palgrave Macmillan New York, 2007. (Cited on page 92.)

P. Pacoda. *Potere alla parola: Antologia del rap italiano*, volume 1397. Feltrinelli, 1996. (Cited on page 92.)

P. Pantel. Inducing ontological co-occurrence vectors. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 125–132. Association for Computational Linguistics, 2005. (Cited on page 23.)

S. Pelosi. Sentita and doxa: Italian databases and tools for sentiment analysis purposes. In *Proceedings of the Second Italian Conference on Computational Linguistics CLiC-it 2015*, pages 226–231. Accademia University Press, 2015. (Cited on page 110.)

S. Perna, A. Maisto, P. Vitale, and R. Guarasci. Il linguaggio del rap. possibilitaá di unánalisi multidisciplinare. In *Aidainformazioni*, volume 1-2, pages 209–218, 2016. (Cited on page 91.)

E. Pianta and R. Zanoli. Tagpro: A system for italian pos tagging based on svm. *Intelligenza Artificiale*, 4(2):8–9, 2007. (Cited on page 17.)

E. Pianta, C. Girardi, and R. Zanoli. The textpro tool suite. In *LREC*. Citeseer, 2008. (Cited on pages 10 and 12.)

L. Picabia. *Les constructions adjectivales en français: systématique transformationnelle*, volume 11. Librairie Droz, 1978. (Cited on page 111.)

J. Plisson, N. Lavrac, D. Mladenic, et al. A rule based approach to word lemmatization. In *Proceedings C of the 7th International Multi-Conference Information Society IS 2004*, volume 1, pages 83–86. Citeseer, 2004. (Cited on page 16.)

A.-M. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *Natural language processing and text mining*, pages 9–28. Springer, 2007. (Cited on page 109.)

M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980. (Cited on page 16.)

A. Ratnaparkhi et al. A maximum entropy model for part-of-speech tagging. In *Proceedings of the conference on empirical methods in natural language processing*, volume 1, pages 133–142. Philadelphia, USA, 1996. (Cited on page 14.)

D. A. Reinstein and C. M. Snyder. The influence of expert reviews on consumer demand for experience goods: A case study of movie critics*. In *The journal of industrial economics*, volume 53, pages 27–51. Wiley Online Library, 2005. (Cited on page 107.)

E. Riloff, S. Patwardhan, and J. Wiebe. Feature subsumption for opinion analysis. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 440–448. Association for Computational Linguistics, 2006. (Cited on page 109.)

P. S. Rosenbaum. *The grammar of English predicate complement constructions. Research monograph 47.* Cambridge, MA: MIT Press, 1967. (Cited on page 28.)

K. Sagae and A. Lavie. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132. Association for Computational Linguistics, 2006. (Cited on page 27.)

M. Sahlgren. The distributional hypothesis. *Italian Journal of Linguistics*, 20 (1):33–54, 2008. (Cited on page 23.)

G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988. (Cited on page 21.)

G. Salton and M. J. McGill. Introduction to modern information retrieval. 1986. (Cited on page 21.)

S. Sasaki, K. Yoshii, T. Nakano, M. Goto, and S. Morishima. Lyricsradar: A lyrics retrieval system based on latent topics of lyrics. In *ISMIR*, pages 585–590, 2014. (Cited on page 98.)

H. Schmid. Treetagger| a language independent part-of-speech tagger. *Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart*, 43:28, 1995. (Cited on pages 17 and 50.)

A. Scholz. Subcultura e lingua giovanile in italia: hip-hop e dintorni. Aracne, 2005. (Cited on page 92.)

H. Schütze and J. O. Pedersen. Information retrieval based on word senses. 1995. (Cited on page 23.)

M. S. Senaldi, G. E. Lebani, L. Passaro, and A. Lenci. Semant-it. (Cited on page 25.)

L. Shen, G. Satta, and A. Joshi. Guided learning for bidirectional sequence classification. In *ACL*, volume 7, pages 760–767. Citeseer, 2007. (Cited on page 15.)

B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996. (Cited on page 99.)

B. Shneiderman. *Designing the user interface: strategies for effective human-computer interaction.* Pearson Education India, 2010. (Cited on page 73.)

M. Silberztein. Intex: a finite state transducer toolbox. *Theoretical Computer Science*, 231(1), 1998. (Cited on page 6.)

M. Silberztein. Nooj: a linguistic annotation system for corpus processing. In *Proceedings of HLT/EMNLP on Interactive Demonstrations*, pages 10–11. Association for Computational Linguistics, 2005. (Cited on page 6.)

M. Silberztein. Nooj manual [electronic resource]. mode of access (2014), 2014. (Cited on page 6.)

S. Sinclair, S. Ruecker, and M. Radzikowska. Information visualization for humanities scholars. *MLA Commons https://dlsanthology. commons. mla. org/information-visualization-forhumanities-scholars/. accessed July*, 2015. (Cited on page 98.)

T. D. Smedt and W. Daelemans. Pattern for python. *Journal of Machine Learning Research*, 13(Jun):2063–2067, 2012. (Cited on pages 9, 17 and 50.)

D. Song and P. Bruza. Discovering information flow suing high dimensional conceptual space. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 327–333. ACM, 2001. (Cited on page 24.)

A. Srinivasan, P. Compton, R. Malor, G. Edwards, and L. Lazarus. Knowledge acquisition in context for a complex domain. *Pre-print of Proceedings of the Fifth EKAW91*, 1991. (Cited on page 16.)

M. Surdeanu and C. D. Manning. Ensemble models for dependency parsing: cheap and good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 649–652. Association for Computational Linguistics, 2010. (Cited on page 27.)

M. Taboada, J. Brooke, M. Tofiloski, K. Voll, and M. Stede. Lexicon-based methods for sentiment analysis. In *Computational linguistics*, volume 37, pages 267–307. MIT Press, 2011. (Cited on page 108.)

M. Terkourafi. *The languages of global hip hop.* A&C Black, 2010. (Cited on pages 91 and 92.)

T. Tokunaga and I. Makoto. Text categorization based on weighted inverse document frequency. In *Special Interest Groups and Information Process Society of Japan (SIG-IPSJ.* Citeseer, 1994. (Cited on page 18.)

J. R. R. Tolkien. The lord ofthe rings. *London: Grafton*, 1991. (Cited on page 18.)

J. R. R. Tolkien. *The hobbit.* Houghton Mifflin Harcourt, 2012. (Cited on page 18.)

D. Toop. *The rap attack: African jive to New York hip hop.* South End Press, 1984. (Cited on page 91.)

K. Toutanova and C. D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 63–70. Association for Computational Linguistics, 2000. (Cited on pages 10 and 14.)

K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003. (Cited on page 15.)

G. Uboldi and G. Caviglia. Information visualizations and interfaces in the humanities. In *New Challenges for Data Design*, pages 207–218. Springer, 2015. (Cited on page 93.)

G. Van Rossum. An introduction to python for unix/c programmers. *Proc. of the NLUUG najaarsconferentie. Dutch UNIX users group*, 1993. (Cited on page 38.)

S. Vietri. *Lessico-grammatica dell'italiano. Metodi, descrizioni e applicazioni.* UTET Università, 2004. (Cited on pages 31 and 111.)

S. Vietri. The italian module for nooj. In *In Proceedings of the First Italian Conference on Computational Linguistics, CLiC-it 2014*, 2014. (Cited on page 6.)

J. Votrubec. Morphological tagging based on averaged perceptron. *WDS'06 proceedings of contributed papers*, pages 191–195, 2006. (Cited on page 15.)

P. Wegner. Concepts and paradigms of object-oriented programming. *ACM SIGPLAN OOPS Messenger*, 1(1):7–87, 1990. (Cited on page 37.)

C.-P. Wei, Y.-M. Chen, C.-S. Yang, and C. C. Yang. Understanding what concerns consumers: a semantic approach to product feature extraction from consumer reviews. In *Information Systems and E-Business Management*, volume 8, pages 149–167. Springer, 2010. (Cited on page 109.)

G. Wilcock. Text annotation with opennlp and uima. 2009. (Cited on page 10.)

J. A. Wise, J. J. Thomas, K. Pennock, D. Lantrip, M. Pottier, A. Schur, and V. Crow. Visualizing the non-visual: spatial analysis and interaction with information from text documents. In *Information Visualization, 1995. Proceedings.*, pages 51–58. IEEE, 1995. (Cited on page 98.)

Q. Ye, R. Law, B. Gu, and W. Chen. The influence of user-generated content on traveler behavior: An empirical investigation on the effects of e-word-of-mouth to hotel online bookings. In *Computers in Human Behavior*, volume 27, pages 634–639. Elsevier, 2011. (Cited on page 107.)

L. Zhang and B. Liu. Identifying noun product features that imply opinions. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 575–580. Association for Computational Linguistics, 2011. (Cited on page 109.)

L. Zhang, B. Liu, S. H. Lim, and E. O'Brien-Strain. Extracting and ranking product features in opinion documents. In *Proceedings of the 23rd international conference on computational linguistics: Posters*, pages 1462–1470. Association for Computational Linguistics, 2010. (Cited on page 107.)

F. Zhu and X. Zhang. The influence of online consumer reviews on the demand for experience goods: The case of video games. In *ICIS 2006 Proceedings*, page 25. 2006. (Cited on page 107.)