NON-LINEAR CONSTRUCTION OF N-GRAMS IN COMPUTATIONAL LINGUISTICS

SYNTACTIC, FILTERED AND GENERALIZED N-GRAMS

Grigori Sidorov

Mexico 2013

Table of Contents

Preface

This book is about a new approach in computational linguistics related to the idea of constructing n-grams in non-linear manner, while the traditional approach consists in using the data from the surface structure of texts, i.e., the linear structure.

In this book, we propose and systematize the concept of syntactic n-grams, which allows using syntactic information within the automatic processing methods such as classification or clustering. It is a very interesting example of application of linguistic information in the automatic (computational) methods. Roughly speaking, the suggestion is to follow syntactic trees and construct n-grams based on paths in these trees. There are several types of non-linear n-grams; future work should determine which types of n-grams are more useful in which natural language processing (NLP) tasks.

The book, first and foremost, is intended for specialists in the field of computational linguistics. However, we made an effort to explain in a clear manner how to use n-grams; we provide a large number of examples, and therefore we believe that the book is also useful for graduate students who already have some previous background in the field.

We want to emphasize that no profound knowledge of computing or mathematics is required; the proposed concepts are intuitively very clear; we use very few formulas and if they appear, they explained in detail.

Grigori Sidorov

September 2013

Introduction

In this book, we discuss an idea poorly studied in the field of computational linguistics: the construction of n-grams in a non-linear manner.

First, we discuss the concept of the vector space model in detail—a conceptual framework for comparison of any type of objects, and then its applicability to the text-related tasks, i.e., its use in computational linguistics. Concepts related to word frequency (*tf-idf*) are discussed, and the latent semantic analysis that allows reducing the number of dimensions is briefly presented.

We mention important concepts concerning the design of experiments in computational linguistics and describe the typical scheme of experiment in this area.

We present the concept of traditional (linear) n-grams and compare it with the concept of n-grams obtained in a non-linear manner: syntactic, filtered and generalized n-grams.

Syntactic n-grams are n-grams constructed by following paths in syntactic trees. The great advantage of syntactic n-grams is that they allow introducing pure linguistic (syntactic) information into machine learning methods. The disadvantage is that syntactic parsing is required for their construction.

We consider both continuous and non-continuous syntactic n-grams. When constructing continuous syntactic n-grams, bifurcations (returns, interruptions) in the syntactic paths are not allowed; when removing this constraint, non-continuous syntactic n-grams are obtained: all sub-trees of length *n* of a syntax tree are considered. It is noteworthy that continuous syntactic n-grams is a special case of non-continuous syntactic n-grams.

We propose a metalanguage for the representation of non-continuous syntactic n-grams, i.e., a formal way to represent a non-continuous syntactic n-gram using brackets and commas, e.g., "a b [c [d, e], f]". In this case, brackets and commas are a part of the n-grams.

In this book, we also present several examples of continuous and non-continuous syntactic n-grams construction for syntactic trees obtained using the FreeLing and the Stanford parsers.

We show that the application of syntactic n-grams in one of the traditional computational linguistics tasks, the task of authorship attribution, gives better results than using traditional n-grams.

At the end, we present several ideas concerning the other types of non-linearly constructed n-grams: 1) filtered n-grams: a filter of words or features is built using a certain criterion before constructing n-grams, then n-grams are constructed using only the elements that passed through the filter, 2) generalized n-grams: words "are generalized" using lexical relations, especially synonymy and hypernymy, in this way, the set of elements used for constructing n-grams is reduced.

Many experimental studies are required in order to determine which construction parameters of continuous and non-continuous, filtered and generalized n-grams are the best and for which existing tasks in computational linguistics.

The book systematizes the recent author's proposals on the non-linear construction of n-grams and their use in the vector space model; thereby, some parts of the book are based on the author's previous works published in various journals and conferences with updates and necessary adjustments.

Work done under partial support of Mexican government (CONACYT, SNI), Mexico City government (ICYT-DF PICCO10-120 project), National Polytechnic Institute, Mexico (projects SIP 20120418, 20131441, COFAA), and FP7 PEOPLE-2010 IRSES: Web Information Quality—Evaluation Initiative (WIQ-EI) European Commission project 269180.

PART I.
VECTOR SPACE MODEL IN THE ANALYSIS OF SIMILARITY BETWEEN TEXTS

Chapter 1. Formalization in computational linguistics

Computational linguistics

Computational linguistics is an important area within the field of linguistics. Computational methods used in computational linguistics originate from computer science, or, to be more specific, from artificial intelligence. However, the primary object of study of computational linguistics remains the modeling of human language, and therefore it continues to be a part of the field of humanities.

Computational linguistics studies how to construct language models so as to be understandable by the computers, that means that it not only analyzes the use of language in human behavior, but also applies specific formal methods that allow the exact formulation of the hypotheses and their subsequent automatic evaluation using linguistic data (corpora) [16, 28, 35].

Computational linguistics and artificial intelligence

The formal part of computational linguistics is related to the methods of artificial intelligence [27]. In general terms, we can define the purpose of the science of artificial intelligence as formal modeling of human intelligence; i.e., the questions that artificial intelligence answers are: what is intelligent behavior? How do humans solve so many practical problems on a daily basis, in most cases without committing errors? Not all areas of artificial intelligence are related to human language, for example, vision or troubleshooting, etc. However, in artificial intelligence, several methods applicable to any type of data have been developed: methods of machine learning; precisely these methods are applied in modern computational linguistics turning it into a formalized science. To some extent, computational linguistics becomes an empirical science, where hypotheses are verified based on the experiments.

Formalization in computational linguistics

Computers are the most important modern tools that have ever been created by mankind. However, the nature of computers is a simple binary logic: zeros, ones and some logical operations upon them. How to transform such a complex phenomenon as human language into this simple logic? For this reason, computational linguistics uses both the knowledge we have about the human language and the existing tools in the area of computer science and mathematics: various types of formal models, programming languages, etc. [6].

Indeed, it is curious that many modern studies in the field of computational linguistics increasingly resemble other areas of computer science and artificial intelligence, especially machine learning related areas: automatic classification or automatic clustering. Nevertheless, we insist that without the linguistic part these methods cannot be applied in human language models. In this case, the part related to linguistics consists in selecting the features (and their values) that are introduced into the classification and clustering algorithms. So why the step towards the use of formal methods was taken precisely in recent years? From our point of view, this is related to the advances of the Internet where a large number of texts are freely available these days. These texts are an excellent source for learning automatic systems. Although it seems that modern automatic systems can do magic – to take data and perform tasks so much like a human being—in fact, they are based on the application of machine learning methods to exceedingly big data sets.

The most common and probably the only way to apply machine learning methods is to use the vector space model. Obviously, it is one of the most commonly used models in modern computational linguistics. The following chapters briefly explain this concept and discuss possible values that can be represented in vector space in case of texts.

Chapter 2. Vector space model

The main idea of the vector space model

First of all, everything that will be explained below is very simple and does not require any computational knowledge, rather only a common sense. Therefore, we suggest the reader to continue without the fear of encountering profound mathematics; we try to explain the few formulas that appear here as clearly as possible.

The vector space model is a widely used model in computer science. Its wide use is due to the simplicity of the model and its very clear conceptual basis that corresponds to the human intuition in processing information and data. The idea behind the model is very simple, and it is an answer to the question, how can we compare objects in a formal way? It seems that the only way to describe the objects is to use a representation with features (characteristics) and their values. It is a universal idea, and it even seems to be the only possible way to work with formal objects [49].

Perhaps, the other option could be the use of associative memories [27], in this case, instead of the features and their values, the relations between the objects are used; the relations are expressed through artificial neurons with corresponding connections; the connections have the corresponding weights. There is a large amount of literature on this subject; however, we will not discuss it in this book.

Well, now we know how to represent two (or more) objects by selecting the features and their values. Note that, in this case, to select the features means to build a model of our objects. Thus, the selection of the features is subjective, but the subsequent comparison is already completely objective.

It is necessary to mention that the selection of the values scale for the features also affects the model we are building, and this is a decision to be made, for example, it is not the same to measure in grams or in tons.

Example of the vector space model

To get a clear picture, let us provide an example. Suppose we want to compare two books. Which features shall we select? As mentioned above, it to a great extent depends on our needs, i.e., there is no unique and correct set of the objects features. However, some features are more common than others, for example, in the case of books, the “number of pages” would be an important feature for many purposes. There could also be such features as “cover color”, “author”, “publisher”, “sociological profile” of the people who liked this book, etc.

The value of the “number of pages” feature would be numeric, i.e., it would be represented by a number. For the “publisher”, the values would be a list of possible publishers. For the feature “sociological profiles”, the values would be a little harder to represent, for example, “undergraduate students of the first year” or “university professors between 40 and 50 years old”, etc. We stress once again that the selection of both features and their values is our own choice, i.e., we are the ones who are building the model, and it would be a matter of its practical application to see whether it is useful or not. This idea is very clear in the case of features; moreover, it is often applied to values. For example, we can measure the weight in kilograms or grams—this decision will greatly affect the subsequent comparisons between objects.

Now we have the representation of two objects in terms of features and their values. What is the next step? What else is needed to build a vector space model? Should it be something very complex? Certainly not, in fact, the model is already built. It is a space of *N* dimensions; each dimension in this space corresponds to a feature: the number of dimensions is equal to the number of features of the object in our model.

One can imagine a dimension as an axis in which we can mark the values of a feature/dimension. If the values are numeric, the interpretation is clear—it is the distance from the point with coordinates (0,0,...). If the values are naturally ordered, for example, age ranges, it is also clear how to treat them, although we have to assign them some numerical values. If the values are not related, for example, the cover color of a book, the easiest solution is to use a random order and equally assign numerical values to each symbolic value.

Note that if we want to handle this situation properly, we can introduce as many new dimensions (features) as many values we have for each feature, and their values can be "present-absent" (1 or 0). The advantage is that, in this case, we do not have to sort the values, the drawback is that the number of dimensions significantly increases.

The next step is related to the following questions: where the vectors are and why is it a vector space? As already mentioned, each object is a set of features and their values, which corresponds to exactly one point in a space of *N* dimensions (the dimensions correspond to the features). This point corresponds to a vector (an arrow in the geometric representation) of *N* dimensions (n-dimensional vector), which starts at the point with the coordinates (0,0,0,..) in this space. For example, let's consider a 100 page book with *red* cover and compare it with a 50 page book with *green* cover. This is a two-dimensional space: "number of pages" and "cover color", and each book is a point with coordinates [100, *red*] and [50, *green*]. Note that we have to choose which numerical values correspond to *red* and *green*.

The dimensions correspond to the position of the particular values in the vector, i.e., feature number 1 has the position 1 in the vector, feature number 2 has the position 2, etc. In this case, as all dimensions are equal, the ordering of the features never affects the model.

Another question is how can we formally represent the vector spaces? As already mentioned, each object is a set of values of the selected features, for example, one book is presented as X = [(*number of pages = 100*) (*cover color = red*)] and the other one as Y = [(*number of pages = 50*), (*cover color = green*)]. Two questions arise: (1) Is there a need to repeat each time the name of the feature? and (2) What can be done if an object does not have any value of a given feature?

Table 1: Example of the vector space tabular representation.

	Book *X*	**Book *Y***
Dimension 1: ***Number of pages***	*100*	*50*
Dimension 2: ***Cover color***	*red*	*green*

The answer to the first question is simple, the feature name is usually omitted, thus X = [*100, red*] and Y = [*50, green*]. Recall that the value of the same feature has the same index in the vector, i.e., the same position in the vector. Another explanation can be based on a table where the columns correspond to the objects and the rows to the features (or it can be the other way around, it does not change the model), see Table 1. In this regard, tabular representation (matrix) and vector representation are the same: the columns correspond to the objects and the rows to the features, while the value of each cell is the value of the feature of a given object.

In this sense, the mere fact of knowing the column of the value already defines to which feature this value corresponds, i.e., this kind of information is directly known from its position in a table.

The answer to the second question is equally simple, what is to be done if a feature is simply not defined for a given object: the corresponding positions are filled in with zero values. In all subsequent calculations, these values will not affect the result being equal to zero.

Note that the tabular representation is conceptually the same as the vector space model *per se*: the columns correspond to the dimensions. The only difference is that, in this case, it is not so natural to use geometric concepts that we present below, but the advantage of this representation is that later we can apply the methods of linear algebra, e.g., the latent semantic analysis (LSA). We will provide an example of application of these methods below.

Similarity of objects in the vector space model

What are the advantages of the vector space concept? We have already built it, how does it help us? It turns out that we can use the metaphor of space to calculate the similarity between objects, i.e., to compare objects based only on very simple geometrical notions, not more complicated than the Pythagorean theorem as explained below.

Now, each object is a vector in a space of N dimensions. How can we compare these vectors? The geometric principle states that the vectors in more or less the same direction resemble each other. Formally speaking, the more acute the angle between the vectors, the greater the similarity. It is very clear and intuitive in a two-dimensional space. For example, in Figure 1, the more acute the angle between each pair of arrows, the more "similar" the arrows of this pair, i.e., their directions most closely coincide.

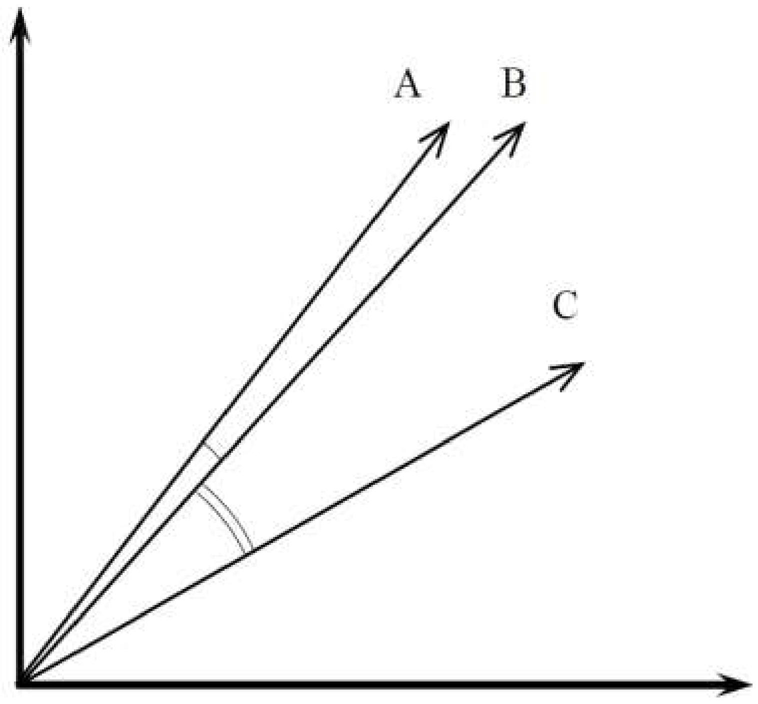

Figure 1: Example of the vector space: similarity between vectors.

For example, vectors A and B resemble each other more than vectors B and C. And of course vectors A and C are least similar to one another. Note that when having three vectors, we can compare three pairs of vectors in total.

In a space with a larger number of dimensions, it is harder to imagine this similarity, therefore, we always suggest considering examples in a two-dimensional space taking into account that in a space with a larger number of dimensions, the principles would be exactly the same.

Cosine similarity between vectors

In order to formally express the similarity, we use the cosine measure of the angle between vectors: the more acute the angle, the greater the cosine, i.e., the greater the similarity between vectors, and thus, the compared objects are more similar.

To calculate the cosine similarity between two vectors V and U, the inner product (dot product) of the normalized vectors is used. The **normalization** consists in dividing the result by the length of each vector or, equivalently, in multiplying their lengths. The length is called the **Euclidean norm** and is denoted, for example, by $\| V\|$ for the vector V.

We would like to remind the reader what is "to normalize" and why it is important. Often it is crucial not to compare absolute values, but relative ones. For example, is 10 greater or less than 20? The question seems to make no sense, because obviously 20 is greater than 10. However, what if we knew that 10 was in the range of 30, and 20 was in the range of 100? In this case, if we normalize these values: 10/30 is greater than 20/100. This demonstrates that if the scale is taken into account (the same scale is used) when comparing values, the results depend on the normalization (scale). In the case of two-dimensional vectors, one vector can be a very long arrow, while the other a short arrow. In order to be able to compare their similarity, we have to convert them into unit vectors using the Euclidean norm.

The dot product of two vectors is a value that is easy to obtain: multiplications of the vector values in each dimension are summed up. In two dimensions (dimension 1 and dimension 2), for vectors U and V, it would be:

$$Dot\_product = V_1 \times U_1 + V_2 \times U_2.$$

That is, the first elements of the vectors are multiplied among themselves, then the same with the second elements of the vectors, and at the end, the products are summed up.

More generally:

$$Dot\_prpduct(V, U) = \sum_{n=1}^{m} (V_n \times U_n),$$

where m is the number of dimensions in the vector space model (which is equal to the vectors' length).

As already mentioned, the normalization of the vectors is the second step in the calculation of the cosine (the first is the calculation of the dot product). To this end, the length of the vectors is to be calculated, and the Euclidean norm is to be obtained. The Euclidean norm converts each vector into a unit vector (a vector of length 1). The normalization using the Euclidean norm ‖V‖ is the division of the vector by its length. The vector length is calculated as follows:

$$\|V\| = \sqrt{\sum_{k=1}^{m} V_k^2}.$$

In the case of two dimensions, this corresponds to the application of the Pythagorean theorem:

$$\|V\| = \sqrt{{V_1}^2 + {V_2}^2},$$

where V_1 and V_2 are the values of the vector V on the axis 1 and 2. Actually, V_1 is the value of the vector in the dimension 1, i.e., the length of the first leg, and V_2 is the value of the vector in the dimension 2, i.e., the length of the second leg, see Figure 1. In this case, the vector itself corresponds to the hypotenuse, and the Pythagorean theorem is applied.

In the case of a larger number of dimensions, the corresponding elements are added to the formula in the same way.

Thus, the final formula[1] for calculating the cosine similarity consists in obtaining the dot product of two vectors and applying the Euclidean norm:

[1] Translator's remark: Note that this measure was generalized into soft cosine measure by Sidorov et al (2014)., when the

$$\mathrm{sim}(V, U) = \frac{\sum_{n=1}^{m} (V_n \times U_n)}{\|V\| \times \|U\|}.$$

In this case, the cosine similarity indicates to which extent the vectors V and U are similar. For positive values, the cosine ranges between 0 and 1.

Note that the cosine similarity is defined for exactly two objects (two vectors). The similarity of an object with itself would be equal to 1. For objects that correspond to the orthogonal vectors (i.e., the angle between them is 90°), the cosine similarity is equal to 0.

The concepts presented in this chapter are quite simple; nevertheless, they allow comparing any type of objects using (1) the vector space model, (2) the corresponding spatial metaphor, and (3) basic geometrical notions.

---

similarity of features is taken into account, see, for example, Wikipedia.

Chapter 3. Vector space model for texts and the *tf-idf* measure

Features for text represented in vector space model

Now we know what a vector space model for any type of objects is. This model consists in selection of features and assignment of values to these features, which allows to represent our objects as vectors, and then to measure their similarity applying the cosine similarity formula. Recall that for the similarity calculation, exactly two objects have to be considered; in the case of a larger number of objects, the similarity (comparisons between objects) is calculated in pairs.

Let's see how this model is applied for the comparison of documents (texts). That is, the objects that we want to compare are documents.

The need for measuring similarity is a very typical situation in automatic natural language processing and computational linguistics tasks. For instance, the most common information retrieval task is precisely the calculation of similarity. We will explain it in a little more detail.

Information retrieval is based on a collection of documents; this collection can be quite large. In the case of the Internet search engines, this collection consists of all the Internet texts previously indexed by the “spiders”, i.e., the programs that follow the links on the Internet (the World Wide Web). A user makes a query which also has a textual form. In this case, the retrieval task is to find the documents in the collection which to a larger extent “look like” the query, i.e., they are similar to the query [4]. As additional criterion, other criteria of similarity such as user profiles or the collection structure (e.g., as in the *PageRank* algorithm) are often used.

Now, if we want to compare the documents, and the suggested way is to use the vector space model, how do we select the documents features? What are the features and their values? As always with the vector space model, we have many options, and it is up to us which features we will consider important and how we will select their values. Thus, in the vector space model, the comparison is objective, but the selection of the features and their values is subjective.

The simplest way is to use words as documents features. Usually some additional procedures are implemented, such as stemming, i.e., all word forms are replaced by their lemmas. For example, the normalized forms *worked*, *working*, etc. correspond to the lemma *work*. Furthermore, the similarity calculation often excludes auxiliary words (stop words) such as prepositions or articles, since their presence in a document bears no information on the document itself, but determined by the characteristics of the language. It is common for many tasks; however, there are specific tasks which require the presence of such words, for example, authorship identification [2].

Values of text features: *tf-idf*

If words are used as features, what values may they have? Intuitively, the values should be somehow related to word frequency. In this sense, the more frequent the word, the more important is this word for a document. Not everything is that simple, but this is the main point.

The frequency of the word in a text document is called *tf* (term frequency), i.e., word (term) frequency shows how many times the word appears in a document. More specifically, it is denoted by tf_{ij}, i.e., how many times the word i appears in the document j. In this sense, it is a value that may differ for each document in the collection.

Normally the frequency of a word is combined with another measure, called *idf* (inverse document frequency). The intuition behind the *idf* is related to the following: if a word appears in all the documents of our collection, then this word cannot distinguish between these documents, and therefore is not useful. Conversely, if a word appears only in one document of our collection, it is a very useful word for similarity calculation (or, for example, for information retrieval which, as already mentioned, is a particular case of similarity calculation). The *idf* is calculated for each word in a given collection, i.e., it depends on the collection but does not depend on a specific document in the collection.

The formula for calculating the *idf* is the following:

$$idf_i = \log \frac{N}{DF_i},$$

where *N* is the total number of documents in the collection, *DF_i* is the number of documents where the word *i* appears at least once, i.e., in this case, no matter how frequent the word is within the document, if it appears only once, it is already sufficient; note that this convention is verified by practice.

It can be observed that if a word appears in all the documents, then the value of the *idf* is equal to log(N/N), which is equal to 0. The *idf* value is the highest when a word appears in only one document.

The logarithm is used to soften the influence of the high frequencies; it is a typical use of logarithms in computational linguistics. Note that *DF_i* can never be equal to zero, because we have considered only the words that are present in our collection. Instead of *tf* we may also use its logarithm: *log(tf + 1).* In this case, we have to anticipate the possibility of zero, so we use "+1".

In general case, it is recommended to combine *tf* and *idf* of a given word for each document: normally they are multiplied. The measure that combines these two metrics is called *tf-idf*, and it is often used as the value of features in the vector space model for the documents comparison. One can apply not only *tf-idf* but also pure *tf*, normalized *tf*, *idf*, etc. [28].

Term-document matrix

So, the objects we are considering are documents (texts). Words (terms) are features of these documents; each word has a *tf-idf* value for each document. That means that each document corresponds to a vector of *tf-idf* values of words.

We can also represent all this information as a matrix. This matrix is called "term-document matrix". An example is presented in Figure 2.

	Doc_1	Doc_2	Doc_3	Doc_4
$Word_1$	0.3	0	0	0.02
$Word_2$	0.7	0.01	0	0.5
$Word_3$	0	0	0	0
$Word_4$	0	0	2.2	0

$Word_5$	1.2	0	0	0
…				

Figure 2: Term-document matrix.

The documents $Doc_1 ... Doc_N$ represent the documents in our collection, the words $Word_1 ... Word_M$ represent all the words that appear in these documents; the words are ordered in a specific way. The values in the table correspond to the assumed *tf-idf* values in the collection.

Note that the table is usually highly sparse, i.e., it contains many zeros. Therefore, it is recommended to use the inverted index for its representation. The inverted index consists in converting the table into a list. For example, the inverted list for the figure above would be:

(Doc_1, $Word_1$, *0.3*), (Doc_1, $Word_2$, *0.7*), (Doc_1, $Word_5$, *1.2*), (Doc_2, $Word_2$, *0.01*), (Doc_3, $Word_4$, *2.2*), (Doc_4, $Word_1$, *0.02*), (Doc_4, $Word_2$, *0.5*),

In this case, there is no need to keep any elements for the zero values in the list. The initial matrix and the inverted index contain the same information. It is a standard procedure that should always be applied to sparse matrices.

It can be noted that this representation is equal to the vector representation, for example, the document *Doc*$_1$, as shown in Figure 2, corresponds to the vector [0.3, 0.7, 0, 0, 1.2].

We will use the matrix (tabular) representation when discussing the latent semantic analysis.

Traditional n-grams as features in vector space model

Note that when using words as features, the information about the syntactic relations between the words is lost. The words become what is called "bag of words". However, for many tasks this loss of information is acceptable. Later in the book, we will propose a possible solution to avoid this loss of syntactic information: syntactic n-grams.

What else, apart from the words, could be the features of documents? Perhaps, it is hard to come up with it fast from scratch, but the concept itself is very simple;

it refers to (traditional) n-grams. The simplicity of the described models is something that we have already mentioned when talking about the vector space model, and we believe that we are convincing the reader that it is actually so.

Traditional n-grams are sequences of elements as they appear in a document [35]. In this case, the letter *n* indicates how many elements have to be taken into account, i.e., the length of a sequence or of an n-gram. For example, there are bigrams (2-grams), trigrams (3-grams), 4-grams, 5 -grams, and so on. Thus, if we talk about unigrams, i.e., n-grams constructed of a single element, it is the same as talking about words.

There are different types of elements that form n-grams. These elements can be lemmas or words; they can also be part of speech tags (POS tags) such as nouns, verbs, etc. The tags can be more detailed, i.e., include grammatical features, for example, a label VIP1S could mean "verb, indicative, present, first person, singular". We can construct n-grams using this kind of tags.

In recent years, character n-grams (character sequences taken from a text) are being used in various different tasks. Interestingly, for some tasks, such as authorship attribution, character n-grams give fairly good results [52]. Their linguistic interpretation is not sufficiently clear and remains an open question.

Let's see an example of traditional n-grams of words. For the sentence: *John reads an interesting book,* we can obtain the following bigrams (2-grams): *John reads*, *reads an*, *an interesting*, *interesting book*. Or the following trigrams (3-grams): *John reads an*, *reads an interesting*, *an interesting book,* etc. We can replace each word for its lemma or part of speech and construct the corresponding n-grams. As we can see, the process is very simple, but it is successfully used in the computational linguistic systems.

If n-grams are used as features, what values may they have? As in the case of words (unigrams), these are the values related to their *tf-idf*. Note that the frequencies of n-grams are usually much lower than the frequencies of words, i.e., n-grams appear much less in a text. It is logical since we actually observe the appearance of sequences of two or more words together, which is a much less likely event than a single word.

Thus, in order to apply the vector space model to texts, we can use n-grams as features. These n-grams can be of various sizes, composed of elements of various types, and their values can be frequencies of *tf*, *idf* or *tf- idf*.

Chapter 4. Latent Semantic Analysis (LSA): reduction of dimensions

Idea of the latent semantic analysis

After building the vector space model, we can represent and compare any type of objects of our study. Now we can discuss the question whether we can improve the vector space we have built. The importance of this question is related to the fact that the vector space model can have thousands of features, and possibly many of these features are redundant. Is there any way to get rid of the features that are not that important?

There are several methods of analysis of the dependencies between features, for example, principal component analysis (PCA), the correlation coefficient, etc. In this chapter, we will briefly describe the method that is called latent semantic analysis (LSA) or latent semantic indexing (LSI) [10].

First of all, there is a need to clarify that although the idea of the latent semantic analysis applied to texts is to find the words that behave similarly (based on the analysis of their contexts), and, in this sense, have the same semantics; in most cases, the results have nothing to do with semantics, except for the initial intention. That is, the idea is to search for the semantic distributional similarity; however, in practice, this similarity is very hard to find applying the latent semantic analysis.

In fact, the latent semantic analysis is an application of a matrices processing technique taken from linear algebra. This technique is called singular value decomposition (SVD) and allows finding the rows with more information (large values) in the matrices, and thereby eliminates the rows with less information (small values).

In this sense, for our purposes, the latent semantic analysis is just a technique to reduce dimensions of a vector space. Well, and why are we suddenly talking about matrices? And how the matrices are related to the dimensions? We have already discussed this issue in the section related to the term-document matrix: the objects are represented as vectors—this corresponds to a multidimensional space—, but the set of vectors represents a matrix.

Examples of the application of the latent semantic analysis

In this book, we will not enter into mathematical details of the latent semantic analysis. Instead, we will provide a couple of simple examples. The intuition behind the latent semantic analysis can be represented in two ways.

Let us consider a matrix that characterizes four objects $O_1 — O_4$ and uses four features $f_1 — f_4$. It can be seen that the values of each pair of features are repeated. In this sense, one feature in each pair is redundant, see Figure 3.

	O_1	**O_2**	**O_3**	**O_4**
f_1	1	1	0	0
f_2	1	1	0	0
f_3	0	0	1	1

f4	0	0	1	1

$\downarrow$ LSA

	O1	**O2**	**O3**	**O4**
f1′	-1.4	-1.4	0	0
f2′	0	0	-1.4	-1.4

Figure 3: Example of the application of the LSA.

Suppose we apply the latent semantic analysis to these data. We have to indicate the desired number of dimensions at the output; in our case, we know that the number of dimensions is two (two features).

Note that the number of objects remains unchanged, i.e., equal to four. The values corresponding to each feature and each object were changed; however, the four objects we have are well described by the two new features, and the elimination of the two other features did not affect their descriptive capacity.

In another example, we consider two dimensions, in each of which a value is marked. We can map these two dimensions to another dimension, as shown in Figure 4. The projections in the new dimension describe our objects just the same.

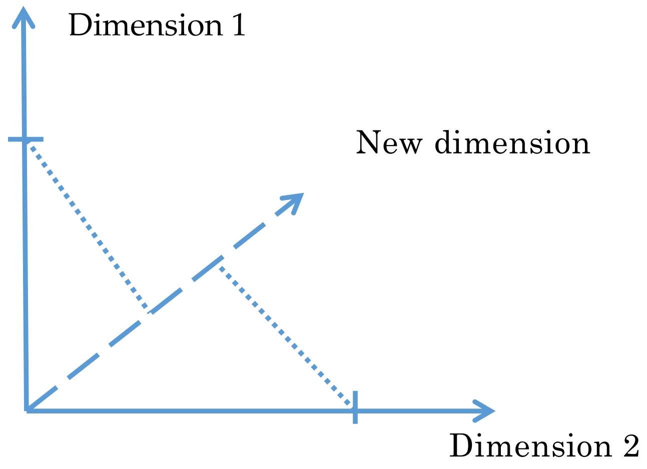

Figure 4: Projections to a new dimension.

We can use another metaphor to describe the latent semantic analysis. It consists in the rotation of our multidimensional space to keep our data with minor changes as possible, and, at the same time, get rid of some dimensions.

Usage of the latent semantic analysis

As mentioned above, the latent semantic analysis is a way to reduce dimensions in a vector space model. It is considered that the latent semantic analysis reflects the distributional properties of its elements, i.e., allows capturing the contexts similarity of words or n-grams; however, in practice, it is hard to present clear cases.

When applying the latent semantic analysis, it is important to indicate the following parameter: how many dimensions should be in the new vector space, i.e., how many dimensions should be reduced. It is recommended to try values between 100 and 500 dimensions, although it is to be verified experimentally for each task.

There are libraries that implement the latent semantic analysis; therefore, we recommend using a freely available code (C, Python, etc.). The latent semantic analysis is a procedure that consumes some computational processing time; however, this procedure is not excessively long.

Chapter 5. Design of experiments in computational linguistics

Machine learning in computational linguistics

As we mentioned earlier in the book, in the automatic analysis of natural language (natural language processing, NLP) and in computational linguistics, machine learning methods are becoming more and more popular. Applying these methods increasingly gives better results [14, 19, 21, 23, 28, 36, 37, 42, 43, 45].

The main purpose of applying the machine learning methods is to try to model the hypotheses formulated by linguists and its further evaluation. In this case, human intuition is replaced by large amounts of textual data—possibly with additional labels made manually—, and by sophisticated learning methods based on mathematics and statistics. The intension is not to replace the linguists in the research process, but to develop tools that can be useful for them.

Furthermore, the application of the machine learning methods allows an accurate evaluation of the language hypotheses, the reproduction of the results, and, to some extent, makes computational linguistics a more exact science. In this sense, some branches of computational linguistics already require empirical procedures when the formulated hypothesis is verified using computational experiments based on the data, and not only on the intuition of native speakers or the experimenter himself.

In the modern computational linguistics, the supervised machine learning methods are the most commonly used, i.e., manually labelled data is used for training. Another option is related to the use of the unsupervised methods, when the system itself has to learn directly from the data. Of course, it is much more complicated to use the unsupervised methods, because in this case, the computer itself has to analyze the data without any human intervention. Normally a huge amount of data is needed to be able to apply these methods.

In computational linguistics, the most used specific machine learning methods are naive Bayes (NB), support vector machines (SVM), classifier based on decision trees (J48).

Basic concepts in the design of experiments

As mentioned above, in computational linguistics, common concepts of information retrieval can be applied [4]— precision and recall—, which measure the viability of a hypothesis in a formal way. The harmonic mean of these two measures is called F1 measure. We always recommend using the latter for the comparison of methods. These are relatively simple concepts. Assume that we have a collection of documents, a query, and a system that we are evaluating. The system generates an output, i.e., presents some retrieved documents which the system considers relevant for the query; we will call them "all retrieved".

Among these documents, some are retrieved correctly, i.e., these are relevant documents: "relevant, retrieved". While others are retrieval errors committed by the system—these are "not relevant, retrieved" documents. That means that the set we call "all retrieved" consists of "relevant, retrieved" and "not relevant, retrieved" documents.

Hence the concept of precision emerges: how good is the answer with respect to itself? How many documents in the output are correctly retrieved? Precision (P) is the ratio of the "relevant, retrieved" documents to all relevant documents ("all relevant").

$$P = \frac{relevant, retrieved}{all\ retrieved}.$$

For example, the precision is 1 when all the documents in the output are correctly retrieved.

There can be, however, other documents relevant to the query, which the system did not manage to retrieve, let's call them "relevant, not retrieved". Hence the concept of recall (R) emerges: how good (specific) was the output of the system with respect to the collection? Were we able to retrieve most of the relevant documents or only a few? Thus, recall is the ratio of the "relevant, retrieved" to all relevant documents ("all relevant"); the latter set includes "relevant, retrieved" and "relevant, not retrieved" documents.

$$R = \frac{relevant, retrieved}{all\ relevant}.$$

Recall equals 1 when the system retrieves all the relevant documents. There is always a relation between precision and recall: if we try to increase one of those values, the other decreases.

Finally, the formula for the F1 measure that combines precision P and recall R is:

$$F1 = \frac{2 \times P \times R}{P + R}.$$

The measure is called F1 because the same weights are assigned to the precision and recall, which are equal to 1. In the case of assigning different weights, the value of F1 varies.

Another concept applied in the design of experiments is the so-called "baseline". This concept corresponds to a commonly accepted method of the state of the art to solve the same problem and has to be overcome by the proposed hypothesis. Normally, the baseline is not a very sophisticated method. It is also advisable to make a comparison of the proposed method with the more complex methods of the state of the art.

Since we are talking about the data manually annotated by humans (annotators), which is used for the evaluation of the method, the concept of "gold standard" is introduced. As the data is manually annotated, it is not supposed to contain errors (this is the idea of "gold"), and if the system can reach this standard, it works really well.

An interesting question is related to the agreement between human annotators, i.e., whether humans themselves annotate some linguistic phenomenon differently. If so, we cannot expect a machine to resolve properly this same phenomenon. To measure the agreement among the annotators "kappa statistic" [9] is used.

There is also the concept of the top line, that is, the maximum value that can be obtained by a program, given a mismatch between annotators. In general, it is advisable to ask several annotators to do the job, and not just one, for better reasoned and less biased judgments.

To carry out the experiments the k-fold cross validation technique is commonly used, where *k* is a numerical value, which normally equals to 10. The technique is to divide all the data sample into 10 (or *k*) subsamples. First, the subsample 1 is used for the evaluation of the system performance, and the subsamples 2-10 are used for training; then subsample

2 is selected for the evaluation, and the other nine subsamples for training, and so on.

Thus, the system is evaluated 10 times on different data and trained 10 times on slightly different data; then the average of the 10 evaluations is taken as the final value. In this way, the effect of fluctuations in the data can be neutralized.

Note that it is very important not to evaluate the performance of the system using the same data on which the system was trained; this is why it is divided into k subsamples. If the same data is used, the learning algorithm can detect the specific characteristics of this data, rather than generalize. This is called overfitting. The overfitting problem is always present in every learning situation, and a number of steps should be taken to prevent it.

To perform all the procedures described, we have to represent the problem formally. Let's briefly overview some of the ideas discussed above. The most commonly used way of representing the objects we are investigating is the vector space model that was discussed in the previous chapters. The problem is represented as a problem of automatic classification in a space, more precisely in a vector space. The objects are represented as sets of features and their values, i.e., each object corresponds to a vector of these values (hence the term “vector space”). This means that each object is a point in the multidimensional space of features. This model is very easy to imagine in the case of two features (two dimensions). For a larger number of dimensions, let’s just assume that it is similar.

After constructing the space, a metric is defined in this space. Typically it is the metric of objects similarity defined by the cosine similarity. The idea behind this similarity is the following: the more two objects resemble each other, the more acute the angle between the corresponding vectors in the defined space, and therefore, the greater is the cosine of this angle.

Now, the next question is how to select the features to define the vector space? At this point of our process of designing the experiment, linguistic considerations begin to prevail: it is precisely the linguistic part that determines the features to be selected.

For example, the simplest idea applied in information retrieval is to use all the words in various documents as their features, and then to compare these documents: the more “similar” the words in a pair of documents, the more these documents resemble each other. This is the way of constructing the vector space for the information retrieval task. For measuring "similarity" of words, their *tf–idf* are to be used.

Obviously, in this case, the type of linguistic information that can be used is restricted by the formal requirement of using the vector space model, i.e., it is our obligation to represent objects as sets of features and their values.

The next option commonly used in practice that already has some linguistic justification, is the idea of using n-grams as features in the vector space model. As noted above, the concept of traditional n-gram—words (or other elements) sequences as they appear in a text—has a linguistic justification, that is to introduce the syntagmatic information of words (follow or precede other words).

However, it would have been much more helpful to use an even more "linguistic" knowledge, i.e., that encloses more proper linguistic information. As a path in this direction, in our previous works [46, 47, 48, 49, 50, 51], we have proposed a new concept of n-grams, which contains more information of a linguistic nature than traditional n-grams: syntactic n-grams. The idea of syntactic n-grams is to construct those by following paths in syntactic trees. That way, syntactic n-grams remain n-grams, yet allow introducing syntactic information into machine learning methods.

In the next part of the book, we discuss syntactic n-grams as well as some other possibilities of non-linear construction of n-grams: filtered and generalized n-grams.

Design of experiments

After considering all the notions mentioned above, we can describe the design of experiments in modern computational linguistics which includes the following steps:

1. Define the task (for example, automatic summarization, authorship attribution, information retrieval, etc.). Often defining a task which differs from the standard tasks contains an interesting scientific contribution.
2. Select the texts for the design of experiments, which is equivalent to the corpus preparation. Several criteria for texts selection can be used. It is

always better to use existing corpora for a given task—that allows a more objective interpretation of the results and comparison. However, it is not mandatory—one can always develop the corpus himself. In this case, we recommend making it public so that others could use it as well.

3. Prepare the gold standard. To do this, we have to annotate the corpus (or a part of it) manually. The type of annotation depends on the problem we are solving. It is recommended to base on judgments of several annotators and to calculate the agreement between them to determine the top line.
4. Build the vector space model selecting the features and their values. It is advisable to try several types of features: unigrams, n-grams, syntactic n-grams as well as various elements they can be composed of (words, lemmas, POS tags, etc.); and several types of values: *tf*, *tf-idf*, etc. Furthermore, we can apply a specific method to make some changes in the standard vector space model. To a large extent, that would be a scientific contribution.
5. Define and implement one or more baseline methods, which are very simple; and one or more methods of the state of the art, which are more

complex ones. Of course, all these methods have to tackle the same problem.

6. Select and implement one or more supervised machine learning methods. It is recommended to use the methods already developed, for example, WEKA [25] implements dozens of machine learning methods. At the same time, we has to analyze the parameters of these methods and their ranges in order to subsequently test different combinations of these parameters. As already mentioned, in computational linguistics, the most used methods are naive Bayes (NB), support vector machines (SVM), and classifier based on decision trees (J48). Nevertheless, we recommend trying as many methods as possible. Moreover, at this point, we have to consider the option of applying the latent semantic analysis (or a similar way of the vector space transformation) to reduce the dimensionality of the problem. LSA is already implemented in WEKA.
7. Convert the textual data into a format accepted by the machine learning methods based on the vector space model built. In the case of WEKA, these are ARFF (attribute relation file format) files.
8. Conduct supervised machine learning experiments: these are the procedures that can be

performed by the machine learning methods—they are able to automatically distinguish between several classes defined in the corpus based on the vector space model built. We recommend using the cross validation procedure based on 10 folds (subsamples). Experiments are carried out for the proposed model and compared with the results of the baseline algorithms and with the algorithms of the state of the art.

9. Calculate the values of precision, recall, and especially of F1 measure for all of the methods mentioned above and perform a comparison between the methods. If the proposed method gives better results, this method is justified.

At the current stage, the scientific contribution to a considerable extent consists in construction of the vector space model, and some additional procedures that allow the transformation of this model (a specific method proposed by a researcher). To some extent, the scientific contribution may also include the problem definition as well as the analysis that shows which machine learning methods, dimension reduction, and parameters are the best for the selected problem.

This is the current research paradigm. We hope that in the future, more attention will be paid to the development of linguistic features—manual, automatic or semiautomatic—, as it is already being realized in the generative models based on local linguistic features, such as Conditional Random Fields, given that for many tasks they produce better results than the traditional methods.

PART II.
NON-LINEAR CONSTRUCTION OF N-GRAMS

Chapter 6. Syntactic n-grams: the concept

The idea of syntactic n-grams

As we have already mentioned, the main idea of the formal features applicable in computational linguistics is related to the vector space model and the use of n-grams as features in this space, which also includes unigrams, i.e., words.

Recall that traditional n-grams are sequences of textual elements (words, lemmas, POS tags, etc.) in the order of their appearance in a text. Traditional n-grams represent syntagmatic information, and they are widely and successfully used in various computational linguistics tasks. Traditional n-grams ignore syntactic knowledge, and they are based solely on syntagmatic information; the established relation is "follow another word". The next question is how can we keep on using the n-grams technique, which is known to give good results, and, at the same time, introduce syntactic information? The solution proposed in this book is the special manner of obtaining n-grams – non-linear manner. Our general proposal is to construct n-grams following paths in syntax trees.

Intuitively, we still deal with n-grams but avoid the noise introduced by the surface structure of the language. This type of noise may occur, because syntactically unrelated words may appear together on the surface level. We can tackle this phenomenon if we follow the actual syntactic relations that link the words, even though those words are not immediate neighbors.

Note that in this chapter, we propose to obtain a syntactic n-gram as a fragment of a continuous path; we do not consider bifurcations (returns, interruptions) in the path—the examples are provided below. In the following chapters, we present the concept of non-continuous syntactic n-grams (the general concept), where following the path in a syntax tree allows entering the bifurcations and going back.

So, for now we continue with the description of syntactic n-grams we call "continuous" in order to illustrate the concept of a syntactic n-gram as such. N-grams are "continuous" in the sense that from any node on the path it is possible to move to a next node.

One can see that it is precisely a non-linear manner of n-grams construction, as the elements are not taken in accordance with their linear order. In this case, the idea of linear order refers to the surface level of a text, where the elements necessarily have to follow each other—recall F. de Saussure, the linear nature of the signifier.

Often the term "syntactic n-gram" implies that an n-gram is composed of POS tags, for example, in [1]. We believe that it is a misuse of the term, since POS tags represent not syntactic but morphological information. Syntactic information is used for the POS disambiguation; however, it does not justify the use of the term in this way.

An idea somewhat similar to ours has been proposed by [7]: to use the concept of *skip*-grams (jump-grams), i.e., to randomly form sequences of elements skipping some of them. It is clearly a non-linear construction; however, the n-grams constructed this way contain more noise than the traditional n-grams, and furthermore, their number becomes too large.

A modification of this idea is to use not all *skip*-grams but only those with higher frequencies [26], called "maximal frequent sequences". However, very sophisticated algorithms are required for constructing these sequences, and there is still a problem of their interpretation—the linguistic reality that corresponds to them does not go beyond finding certain combinations of words.

The others ideas presented in this book on how to build n-grams in a non-linear manner are related to the concepts of filtered n-grams – e.g., using the *tf-idf* of n-grams as a filter BEFORE the construction of n-grams – and generalized n-grams. Say, for the generalized n-grams we can always use the first word in the list of synonyms (synset) or to promote the words in an ontology to more general concepts, then we can replace the words for these concepts and construct the n-grams out of these more general concepts.

It is noteworthy that the idea of using structural information concerning relations between words in specific tasks has been already presented in [3, 24, 32]; however, none of these works has been widespread nor associated with the idea of n-grams.

The work [41] proposes a similar idea in the field of semantic analysis, where the utility of syntactic information is shown for very specific tasks of:

1. semantic priming, i.e., psycholinguistic experiments on words similarity,
2. synonymy detection in TOEFL tests, and
3. ordering word senses according to their importance.

In our opinion, this work does not have much response in other NLP tasks, precisely because the authors do not relate syntactic information to n-grams, which is the main tool in the vast majority of tasks, nor show its utility in other tasks that are not so specifically semantics oriented.

For the illustration of the concept of syntactic n-grams let us consider, as an example, two sentences taken from a book by Jules Verne. The first example is in the Spanish language, and the second one is in the English language. For the both examples, the syntax tree is built in advance, i.e., we have syntactic information represented in terms of dependency grammars.

Example of continuous syntactic n-grams in Spanish

The parser of the FreeLing system [5, 39, 40] is used to parse the Spanish example. The syntactic dependencies are shown through the indented analysis result, i.e., the blocks with the same indentation have the same main word (if another possible main word does not appear among those words). The sentence in Spanish is the following:

El doctor Ferguson se ocupaba desde hacía mucho tiempo de todos los pormenores de su expedición. (lit: *The Dr. Ferguson has been* engaged upon *all the details of his expedition since a long time ago.*) *'Dr. Ferguson had long been engaged upon the details of his expedition.'*

The FreeLing parser generates the following output in terms of dependency grammars:

```
grup-verb/top/(ocupaba ocupar VMII1S0 -) [
  morfema-verbal/es/(se se P0000000 -)
  sn/subj/(doctor doctor NCMS000 -) [
    espec-ms/espec/(El el DA0MS0 -)
    w-ms/sn-mod/(Ferguson ferguson NP00000 -)
  ]
  prep/modnomatch/(desde desde SPS00 -)
   grup-verb/modnomatch/(hacía hacer VMII1S0 -)
[
    sn/cc/(tiempo tiempo NCMS000 -) [
      espec-ms/espec/(mucho mucho DI0MS0 -)
      sp-de/sp-mod/(de de SPS00 - ) [
        sn/obj-prep/(pormenores pormenor NCMP000
-) [
          espec-mp/espec/(todos todo DI0MP0 -)
[
            j-mp/espec/(los el DA0MP0 -)
          ]
        ]
      ]
      sp-de/sp-mod/(de de SPS00 - ) [
            sn/obj-prep/(expedición  expedición
NCFS000 -) [
          espec-fs/espec/(su su DP3CS0 -)
        ]
      ]
    ]
    F-term/term/(..Fp -)
  ]
]
```

The corresponding tree is illustrated in Figures 5 and 6. It can be seen that the parser committed several errors. For example, it placed the groups of words *de su expedición* 'of his expedition' and *de todos los pormenores* 'upon all the details' as dependents of the word *tiempo* 'time' instead of the verb *ocuparse* 'engage'. Parsers can commit such errors due to various types of syntactic ambiguity, which is difficult to solve automatically. Nevertheless, in many cases, this does not significantly affect the system performance, since the vast majority of dependencies are established correctly, and these errors are not severe enough to ruin the tree structure. In the tree below, we indicate the parser errors with dotted lines. Moreover, for the automatic processing, the output format of this parser is not very handy.

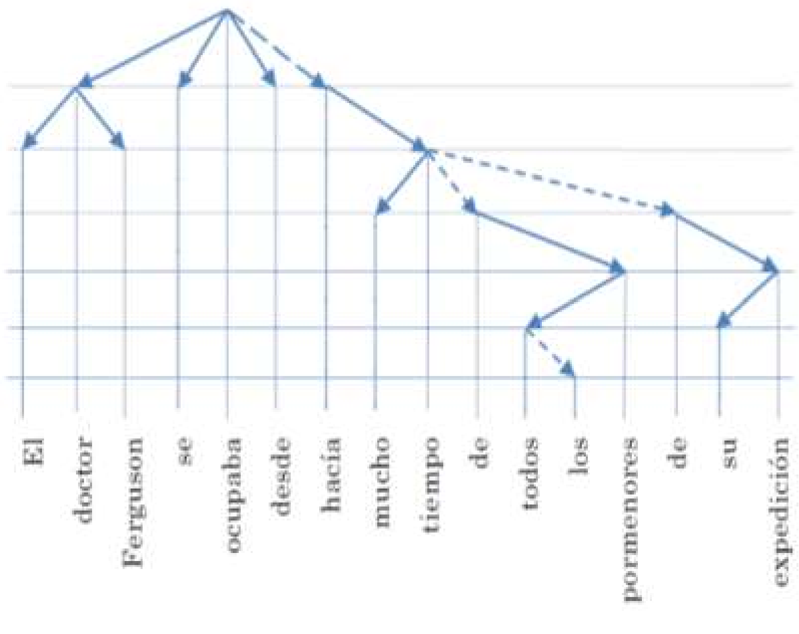

Figure 5: Example of a syntax tree in Spanish.

We have developed a software that converts the FreeLing output format into another format, which is similar to the one of the Stanford parser. The software is freely available on the author's personal web page.

We present the result of the format conversion below. Note that in this case, the words numbers correspond to the lines in the output generated by the FreeLing parser and not to the actual numbers of their positions in the sentence.

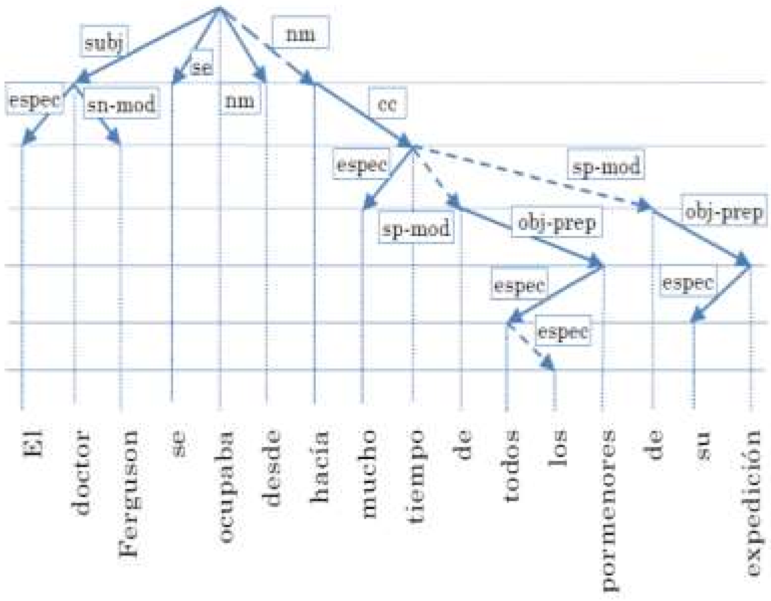

Figure 6: Example of a syntax tree with tags in Spanish.

```
top(root-0, ocupaba-1)
es(ocupaba-1, se-2)
subj(ocupaba-1, doctor-3)
espec(doctor-3, el-4)
sn-mod(doctor-3, Ferguson-5)
modnomatch(ocupaba-1, desde-6)
modnomatch(ocupaba-1, hacía-7)
cc(hacía-7, tiempo-8)
espec(tiempo-8, mucho-9)
sp-mod(tiempo-8, de-10)
obj-prep(de-10, pormenores-11)
espec(pormenores-11, todos-12)
espec(todos-12, los-13)
sp-mod(tiempo-8, de-14)
obj-prep(de-14, expedición-15)
espec(expedición-15, su-16)
```

Using the tree, we can obtain the following continuous syntactic n-grams.

The bigrams are the following:

ocupaba se 'engaged'
ocupaba doctor 'engaged dr.'
doctor el 'dr. the'
doctor ferguson 'dr. ferguson'
ocupaba desde 'engaged since'
ocupaba hacía 'engaged ago'
hacía tiempo 'ago time'
tiempo mucho 'time long'
tiempo de 'time upon'
de pormenores 'upon details'
pormenores todos 'details all'
todos los 'all the'
tiempo de 'time of'
de expedición 'of expedition'

expedición su ‘expedition his’

The obtained trigrams are the following:

ocupaba doctor el ‘engaged dr. the’
ocupaba doctor ferguson ‘engaged dr. ferguson’
ocupaba hacía tiempo ‘engaged ago time’
hacía tiempo mucho ‘ago time long’
hacía tiempo de ‘ago time upon’
hacía tiempo de ‘ago time of’
tiempo de pormenores ‘time upon details’
de pormenores todos ‘upon details all’
pormenores todos los ‘detail all the’
tiempo de expedición ‘time of expedition’
de expedición su ‘of expedition his’

The number of the 4-grams is a little less:

ocupaba hacía tiempo mucho ‘engaged ago time long ’
ocupaba hacía tiempo de ‘engaged ago time upon’[2]
ocupaba hacía tiempo de ‘engaged ago time of’

[2] This n-gram is repeated because there are two different words *de* (‘of’ and ‘upon’) in the sentence. This means that the frequency of this n-gram is equal to two in our example.

hacía tiempo de pormenores 'ago time upon details'
hacía tiempo de expedición 'ago time of expedition'
tiempo de pormenores todos 'time upon details all'
de pormenores todos los 'upon details all the'
tiempo de expedición su 'time of expedition his'

There are several 5-grams:

ocupaba hacía tiempo de pormenores 'engaged ago time upon details'
ocupaba hacía tiempo de expedición 'engaged ago time of expedition'
hacía tiempo de pormenores todos 'ago time upon details all'
hacía tiempo de expedición su 'ago time of expedition his'
tiempo de pormenores todos los 'time upon details all the'

In this case, three 6-grams can also be obtained:

ocupaba hacía tiempo de expedición su 'engaged ago time of expedition his'
ocupaba hacía tiempo de pormenores todos 'engaged ago time upon details all'
hacía tiempo de pormenores todos los 'ago time upon details all the '

And finally, there is only one 7-gram:

ocupaba hacía tiempo de pormenores todos los ‘engaged ago time upon details all the’

The methodology for obtaining continuous syntactic n-grams is the same for all languages. As we can see, it is true in the case of English and Spanish; moreover, dependency-based trees have the same structure for all languages.

Example of continuous syntactic n-grams in English

For the example in English we use the Stanford parser[3] [8]. First, we present the output of the parser as generated by the program itself, and then, in Figures 7 and 8, we illustrate the dependency-based tree using multilevel arrows. The depth of the syntax tree is an important concept: starting from the root of a sentence and moving down while following the

[3] Parser is a program that generates syntactic trees. The trees are usually based on formal grammars of various types.

syntactic path. The sentence in English is the following:

The wildest cheering resounded on all sides; the name of Ferguson was in every mouth

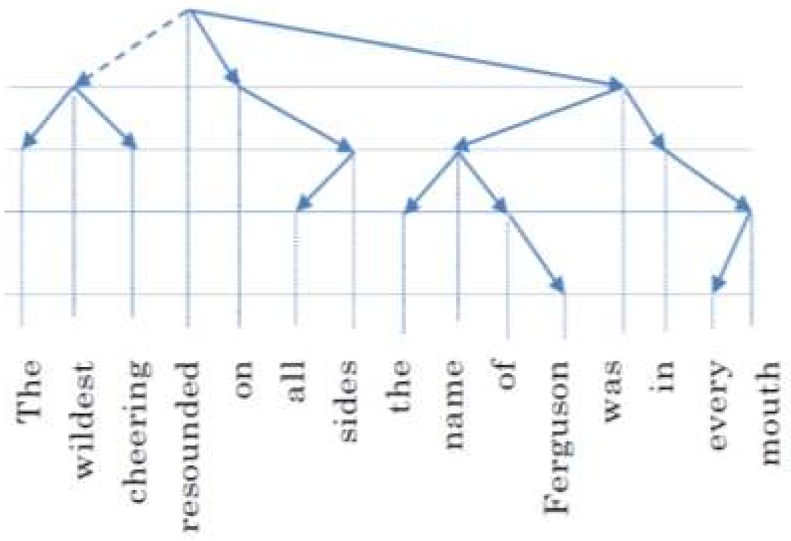

Figure 7: Example of a syntax tree.

The direct output of the parser consists of two parts. In the first part, the information is presented in terms of constituency grammars, where a greater indent of a line corresponds to a greater depth of each element. However, for our purposes, the second part of the output, where the information is presented in terms of dependency grammars, is of more interest; in this part, we can see the pairs of words and syntactic dependencies between them. From this second part we can directly build a syntax tree, as shown in Figures 7 and 8.

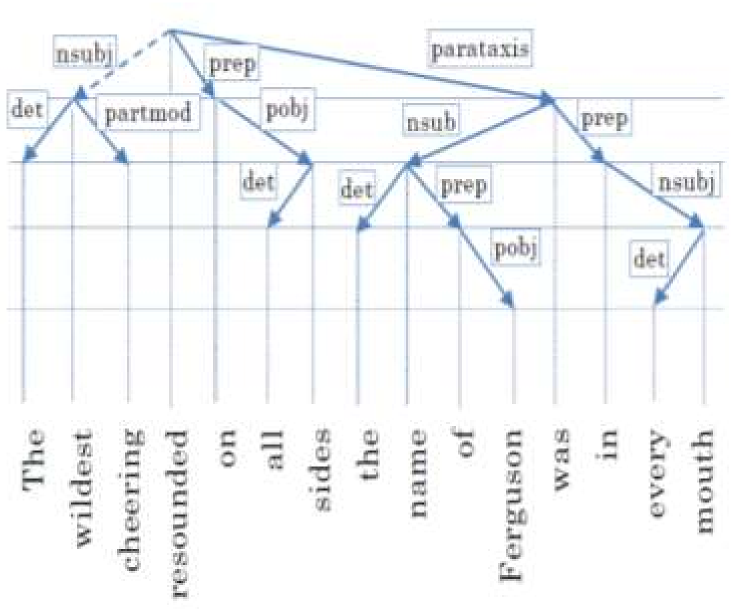

Figure 8: Example of a syntax tree with tags.

Note that in the automatic parser, various errors may occur. In the figures below, one type of such errors is marked with the dotted line, where the word *wildest* was parsed as a noun and the word *cheering* as a participle; when the correct way would be to parse *cheering* as a noun and *wildest* as an adjective.

The first part of the output corresponds to the constituency formalism:

```
(ROOT
 (S
  (S
   (NP
    (NP (DT The) (NN wildest))
    (VP (VBG cheering)))
   (VP (VBD resounded)
    (PP (IN on)
     (NP (DT all) (NNS sides)))))
  (: ;)
  (S
   (S
    (NP
     (NP (DT the) (NN name))
     (PP (IN of)
      (NP (NNP Ferguson))))
    (VP (VBD was)
     (PP (IN in)
      (NP (DT every) (NN mouth)))))

(. .)))
```

The second part of the output is presented in terms of dependency grammars:

```
det(wildest-2, The-1)
nsubj(resounded-4, wildest-2)
partmod(wildest-2, cheering-3)
root(ROOT-0, resounded-4)
```

```
prep(resounded-4, on-5)
det(sides-7, all-6)
pobj(on-5, sides-7)
det(name-10, the-9)
nsubj(was-13, name-10)
prep(name-10, of-11)
pobj(of-11, Ferguson-12)
parataxis(resounded-4, was-13)
prep(was-13, in-14)
det(mouth-16, every-15)
pobj(in-14, mouth-16)
```

One can see that for each pair of words the parser indicates the type of the syntactic relation between them and the number of the word in the sentence. This information is important, because if a word is repeated in a sentence, there would be no way to determine to which of the two instances of that word the given pair refers.

The same syntax trees are shown in Figures 7 and 8, the only difference is that in Figure 8, the name of the corresponding syntactic dependency is shown above each arrow.

After the explanation above, it is quite obvious which n-grams can be obtained from this sentence.

For example, the following bigrams can be obtained:

resounded wildest
wildest cheering

wildest the

resounded on

on sides

resounded was

was name

name of

of Ferguson

was in

in mouth

mouth every

Also the following trigrams can be extracted:

resounded wildest the

resounded wildest cheering

resounded on sides

on sides all

resounded was name

resounded was in

was name the

was name of

name of ferguson

was in mouth

in mouth every

There are six 4-grams:

resounded on sides all
resounded was name the
resounded was name of
resounded was in mouth
was name of ferguson
was in mouth every

And finally, there are only two 5-grams:

resounded was name of ferguson
resounded was in mouth every

Chapter 7. Types of syntactic n-grams according to their components

N -grams of lexical elements

So, we have already learned how to obtain syntactic n-grams (although, at the moment, we are considering only continuous syntactic n-grams). Now let's discuss what types of syntactic n-grams exist depending on the elements they are formed of, i.e., what kind of elements (components) can be parts of syntactic n-grams. In fact, the considerations to be discussed are the same for any type of n-grams.

It is clear that the use of words is the most obvious option, as in all the examples mentioned above. It is also clear that instead of words we can use their normalized forms—lemmas—obtained by morphological normalization. Another similar option is to use the stems of words (the process of obtaining the stems is called stemming). In this sense, the lemma and the stem have the same function: they represent the entire set of grammatical forms that correspond to a word. The advantage of using morphological normalization is that the number of elements that can compose n-grams and, therefore, the number of n-grams are reduced, i.e., there are fewer dimensions in the vector space model.

N -grams of POS tags

Similarly, instead of words we can use the grammatical information of each word (POS tag), for example, in the case of Spanish, we can use the tags that are produced by the parsers or morphological analyzers, as in the FreeLing system: NCFS000, VMII1S0, etc. The tags used in FreeLing are a *de facto* standard for encoding morphological information in Spanish; the standard is called EAGLES. In this case, the first letter corresponds to the grammatical class: "N" stands for noun, "V" for verb, etc. The second letter reflects a number of lexical properties, for example, in the case of nouns, "C" stands for common name (another value of this feature could be "P" - proper noun), etc. The subsequent letters reflect grammatical features, e.g., in the case of nouns, "F" corresponds to female gender, "M" to male gender, etc.

N -grams of syntactic relations tags

In order to form syntactic n-grams, there is also an additional option of using new types of elements as compared to traditional n-grams: tags of syntactic relations (SR-tags) in a syntax tree, for example, `nsubj` or `pobj` in the case of English (Figure 8), or `espec` or `obj-prep` in the case of Spanish (Figure 6). In this case, an n-gram would be a sequence of SR-tags, for example, `sp-mod obj-prep espec` (in the presented example, it is a trigram).

N -grams of characters

The last option that exists for traditional n-grams construction is the use of characters as elements of n-grams. For example, in the phrase *John reads*, there are the following bigrams: *"jo"*, *"oh"*, *"hn"*, *"n "*, *" r"*, *"nr"*, *"re"*, *"ea"*, *"ad"*, *"ds"*. Here the space between the words is used as an element of the n-grams; one can also use punctuation marks. However, for some tasks, it is better not to consider auxiliary characters.

In the same way as in the case of traditional n-grams, one can also use characters as elements of syntactic n-grams; however, there is a need to obtain syntactic bigrams or trigrams of words in advance, and then to consider the sequence of words in a syntactic n-gram as a source for constructing n-grams of characters. It is a matter of future research to determine whether this type of syntactic n-grams is useful. It turns out that the application of traditional n-grams of characters gives the best results in certain tasks, for example, in the task of authorship attribution [52]. However, in our point of view, the application of n-grams of characters is somewhat counter-intuitive, and it is necessary to analyze the reasons of its good performance (see the section concerning filtered n-grams of characters below).

Mixed n-grams

Finally, mixed n-grams may exist, which means that some elements of an n-gram are of a certain type, while other elements of the same n-gram are of another type. It seems that characters cannot be used in the mixed n-grams construction, as they are of a different nature: characters represent parts of words, while other types of elements represent words.

Concerning the mixed n-grams, it should be analyzed in the future, which combinations of elements (words, POS tags, SR-tags) in which positions (in the beginning, in the middle, or in the end of an n-gram) give better results.

Classification of n-grams according to their components

To sum up, we can say that there are syntactic n-grams of:

- lexical elements (words, lemmas, or stems);
- part of speech tags (POS tags);
- tags of syntactic relations (SR-tags);
- characters;
- mixed syntactic n-grams (combinations of the above).

In [41], the idea of weighting the relations between elements of a syntactic n-gram is mentioned. This idea does not seem directly applicable within the context of the vector space model, where n-grams are the features (dimensions). However, this idea can prove useful when calculating the weights of syntactic n-grams, apart from the traditional values of *tf-idf* measure.

Chapter 8. Continuous and non-continuous syntactic n-grams

Continuous syntactic n-grams

In the previous chapters, we introduced the new concept of syntactic n-grams, i.e., n-grams obtained following paths in syntax trees.

As we show in the following chapters, syntactic n-grams can give better results than traditional n-grams in various NLP tasks. Note that syntactic n-grams can be applied in any tasks where traditional n-grams are used, because they allow the construction of the vector space. We will analyze their applicability for the task of authorship attribution.

The disadvantage of syntactic n-grams consists in the fact that previous syntactic processing is required for their construction, which takes some processing time; however, it is not a serious limitation. One more limitation is that for some languages, there are no existing automatic parsers; nevertheless, the parsers do exist for the more widely spoken languages such as Spanish or English.

The discussion that follows in this chapter addresses the comparison of continuous syntactic n-grams with non-continuous syntactic n-grams.

The sentence to be analyzed is the following[4]:

Tomé el pingajo en mis manos y le di un par de vueltas de mala gana (lit: *I took the scrap in my hands and gave it a pair of turns without enthusiasm*) *'I took the scrap in my hands and turned it a couple of times unwillingly'*.

The syntax tree of the sample sentence is shown in Figures 9 and 10, using the dependency and constituency formalisms [15, 22, 51]. Note that the expression *de_mala_gana* (lit: *without enthusiasm*) is considered as one word.

[4] A sentence from one of the books by A. Conan-Doyle.

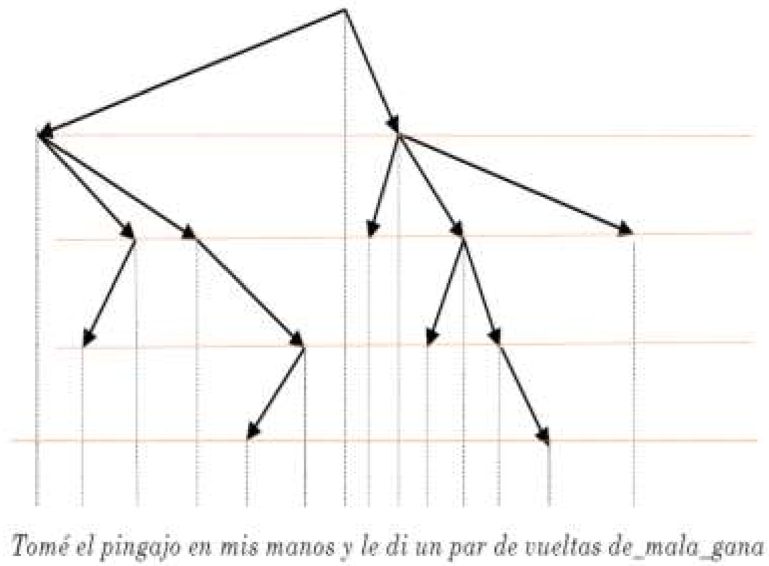
Tomé el pingajo en mis manos y le di un par de vueltas de_mala_gana

Figure 9: Dependency-based tree (generated by the FreeLing parser).

All the syntactic n-grams considered in the previous chapters are continuous, regardless of the type of elements they are formed of. That means that syntactic path we are following is never bifurcated. For example, in Figure 11, the path marked with the bold arrows corresponds to the continuous syntactic 5-gram, *y di par de vueltas* (lit: and gave pair of turns).

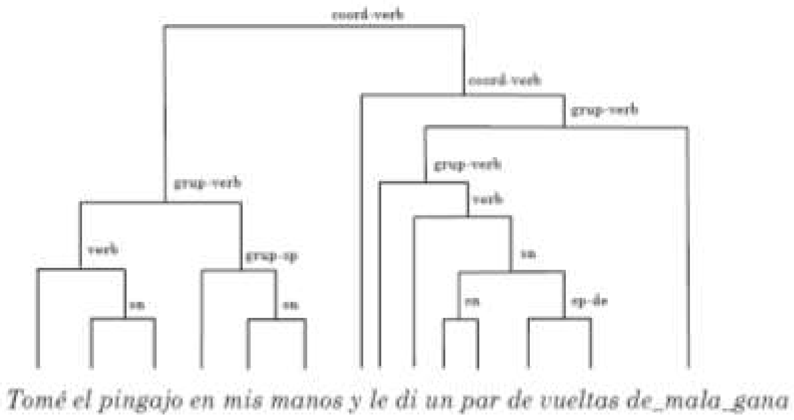

Figure 10: Constituency-based tree (generated by the FreeLing parser).

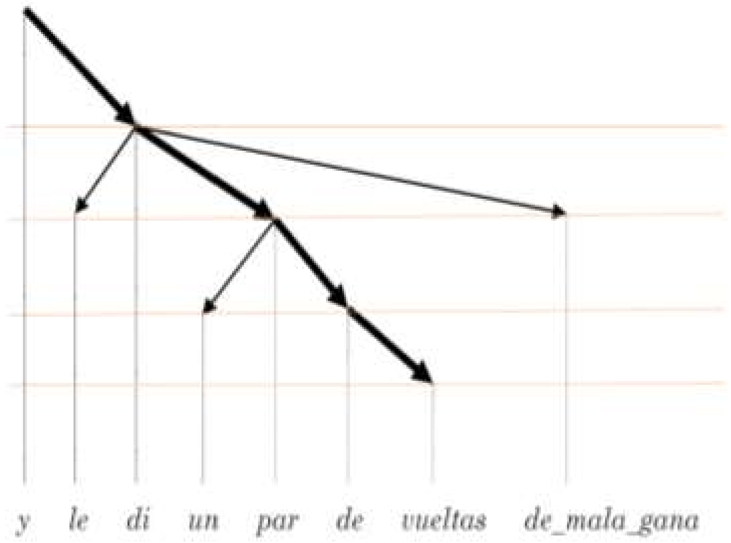

Figure 11: Continuous syntactic n-grams from the syntax tree fragment, a 5-gram.

We present another type of syntactic n-grams below, for which bifurcations are allowed.

Non-continuous syntactic n-grams

In this section, we present a generalization of the concept of continuous syntactic n-grams: non-continuous syntactic n-grams [50, 51].

As shown in the previous section, the intuition behind the concept of continuous syntactic n-grams is mainly related to the fact that a sequence of related words can be considered as such, as a whole.

However, there are other interesting linguistic concepts that do not fit into the model of a one-dimensional sequence, for example, verb valency patterns.

For instance, the verb *to buy* has the following actants: *who*, *what*, *from who*, *for how much money*. It would be interesting to have them presented in an n-gram at the same time.

However, both in the case of traditional n-grams and continuous syntactic n-grams, all these components would be separated into different n-grams. Thus, the intuition behind the concept of continuous syntactic n-grams is precisely the intention to merge semantically related words, even though they do not have a continuous path, but a path that connects them.

It is very easy to give a formal definition of non-continuous syntactic n-grams: these are all the sub-trees of length *n* of a syntax tree.

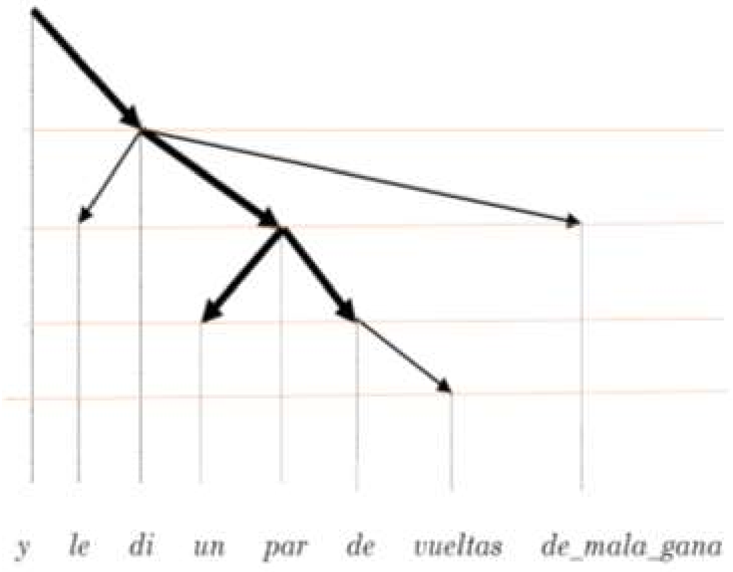

Figure 12: Non-continuous syntactic n-grams from the syntax tree fragment, a 5-gram.

Two examples of non-continuous syntactic n-grams that are fragments of the sentence considered above are shown in Figures 12 and 13. The n-grams are marked with the bold arrows.

In the first case (Figure 12), the 5-gram is *y di par [un, de]* 'and gave pair [a, of]'. Note that it is necessary to introduce a metalanguage for the representation of non-continuous syntactic n-grams in order to resolve the ambiguity. The metalanguage is discussed in the next chapter.

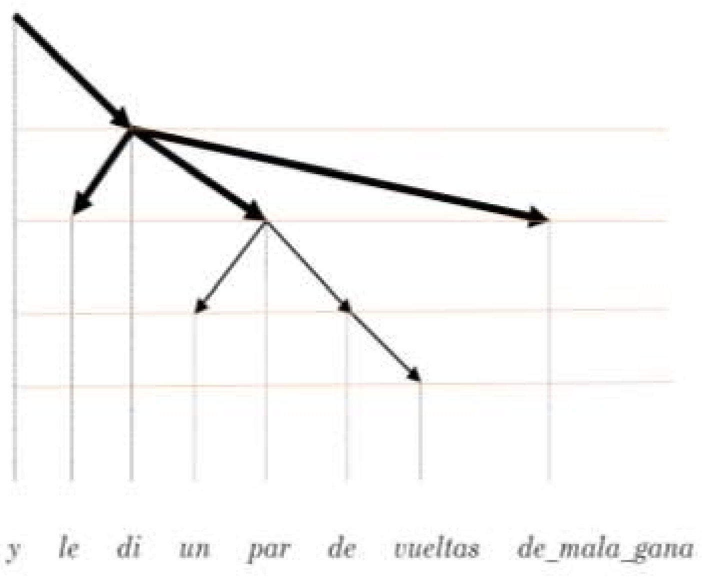

Figure 13: Non-continuous syntactic n-grams from the syntax tree fragment, another 5-gram.

In the second case (Figure 13), the 5-gram is *y di [le, par, de_mala_gana]* ‘and gave [it, pair, without_enthusiasm]’.

Thus, in a formal way, continuous syntactic n-grams are defined as all the sub-trees of length *n* without bifurcations of a syntax tree. Another way to put it formally is that each node of the path is connected to a single node.

Applying the proposed definitions, it follows that continuous syntactic n-grams are a particular case of non-continuous syntactic n-grams. The length of a tree is the number of branches in this tree, which corresponds to the value of *n* (in the case of n-grams).

Another term that we propose in order to denote the non-continuous syntactic n-grams is t-n-grams (tree n-grams), i.e., n-grams of trees. There is also an option to use the term “arboreal n-gram”. A. Gelbukh suggestion is to use the term “tree grams, t-grams”; however, in our opinion, the term “tree n-grams” seems more justified, since, in this way, there is a relation established between the proposed term and the traditional concept of n-grams. A consideration in favor of the term “t-gram” is its simple form;

however, we prefer to adhere to the term "syntactic n-gram".

It is a matter of future work to determine the application of which type of n-grams (continuous or non-continuous) is more suitable for which computational linguistics tasks. It is possible that one type of n-grams is better for some tasks, while the other is better for other tasks.

It is noteworthy that the number of non-continuous syntactic n-grams is greater than the number of continuous syntactic n-grams, since the latter is a particular case of the former.

The construction algorithm (or the algorithm for obtaining) of continuous syntactic n-grams is relatively simple. For the root node, all the possible combinations of its children, whose size is not greater than *n*, are to be considered; this procedure should be repeated recursively for each child node. In this way, we are to go successively through all the nodes of a syntax tree.

Chapter 9. Metalanguage of syntactic n-grams representation

The question arises, how to represent non-continuous syntactic n-grams without resorting to their graphic form? Recall that continuous syntactic n-grams are simply sequences of words (obtained by following paths in a syntactic tree), but the case of the non-continuous syntactic n-gram is rather different.

We propose to use the following conventions. Note that these are conventions, so they can be modified in the future. Within each non-continuous syntactic n-gram, there may be continuous parts and one or several bifurcations. Let's separate the continuous elements of n-grams with whitespaces and put commas in the bifurcation parts; we will also use parentheses to mark the bifurcation parts in order to avoid the structural ambiguity in the future.

Note that we have always considered the highest level element to appear on the left side (the main word of the relation), and the component of the lower level on the right side (the word dependent on the syntactic relation). It is the most natural way; nevertheless, it is necessary to mention it explicitly.

Two examples of the non-continuous syntactic 5-grams are shown in Figures 12 and 13, *y di par [un, de]* 'and gave pair [a, of]'; *y di [le, par, de_mala_gana]* 'and gave [it, pair, without_enthusiasm]'.

Note that we cannot avoid using the brackets or commas in our metalanguage, as otherwise it becomes ambiguous. In our examples, it seems that we can avoid using the brackets if the comma indicates that the previous word is a part of a bifurcation. It is only possible when the elements in the bifurcation do not contain a path. For example, in the 5-gram, *y di [par un, de_mala_gana]* 'and gave [pair a, without_enthusiasm]', the word *un* 'a' is a dependent of *par* 'pair', which is expressed with the whitespace; however, in this case, it is clear that we cannot avoid the brackets.

It is noteworthy that the brackets and the commas are parts of the n-grams now; however, this in no way precludes the possibility to identify the similarity between syntactic n-grams. Although they now contain some additional symbols and not only words, they can still be compared without any complications.

Here is another example of a possible ambiguity. Let's consider the case in which an n-gram has two bifurcations and multiple contiguous fragments. For example, the n-grams "*a [b, c [d, e, f]]*" and "*a [b, c [d, e], f]*" have a node *f* as the third node below the node *c*, or as the third node below the node *a*, see Figure 14.

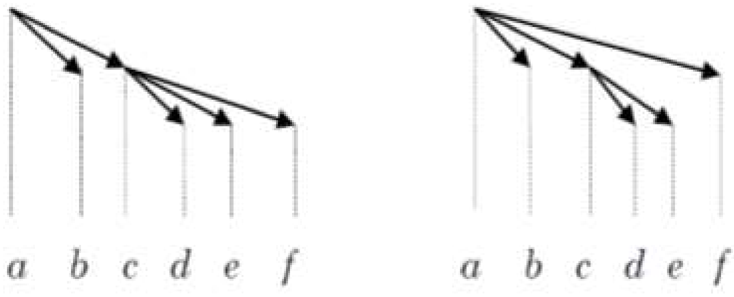

Figure 14: Example of a possible ambiguity in bifurcations.

Now, there are two ways to tackle the bifurcation parts, i.e., the parts separated by commas:

1. as they appear in the text, which is the most natural way, or
2. to sort then in a certain way, for example, alphabetically, which decreases the number of n-grams.

The latter option allows us to take into account the changes related to the word order. However, further research is required in order to determine which of the two options is better and for which NLP tasks.

Another possibility that we would like to mention is to mark the depth directly in the non-continuous syntactic n-grams. The intuition behind this idea is that, for some types of non-continuous syntactic n-grams, the position in the syntax tree of a sentence can be important. In this case, the notation would be: *y*$_1$ *di*$_2$ *par*$_3$ *[un*$_4$*, de*$_4$*]* 'and$_1$ gave$_2$ pair$_3$ [of$_4$, a$_4$]', y$_1$ *di*$_2$ *[le*$_3$*, par*$_3$*, de_mala_gana*$_3$*]* 'and$_1$ gave$_2$ [it$_3$, pair$_3$, without_enthusiasm$_3$]'. Technically, it would be sufficient to mark the level of the first word only; we can mark the other levels as well, but it is not strictly necessary.

Note that this notation can also be used to denote the structure of an n-gram by replacing commas and brackets. It has to be taken into account that all the n-grams should originate from the level "1"—regardless of their actual level in a sentence—, since in the opposite case, it will not be possible to identify the similar n-grams that belong to different levels.

Considering this complication in the n-grams construction, we tend to use brackets and commas.

We consider two examples of non-continuous syntactic n-grams construction below, one for Spanish and another for English, and compare them with continuous syntactic n-grams.

Chapter 10. Examples of non-continuous syntactic n-grams construction

Example for Spanish

In this section, we provide examples of the continuous and non-continuous syntactic n-grams construction for Spanish. We analyze the sample sentence provided in the previous chapter:

Tomé el pingajo en mis manos y le di un par de vueltas de mala gana (lit: *I took the scrap in my hands and gave it a pair of turns without enthusiasm*) ‘*I took the scrap in my hands and turned it a couple of times unwillingly*’.

In order to construct syntactic n-grams automatically, it is necessary to parse the text beforehand using a parser. For the Spanish language we use the FreeLing parser [5, 39, 40], which is freely available online.

The parser can generate syntax trees in terms of two formats: constituency and dependency. The dependency-based tree is shown in Figure 9, and the constituency-based in Figure 10. Both formats have essentially the same information on words relations. For the syntactic n-grams construction, it seems better to use the dependency format, since this representation is more transparent. However, one can use the constituency-based tree in the same way.

It is noteworthy that the parser performs morphological analysis and lemmatization in the first place. As can be seen, a lemma and grammatical information correspond to each word in the sentence, e.g., *Tomé tomar* VMIS1S0 'Took take VMIS1S0'. First comes the word, then the lemma, and finally the grammatical information.

We have already mentioned that in order to represent the grammatical information, the EAGLES coding scheme is applied, which is a *de facto* standard for the automatic morphological analysis of Spanish. For example, considering the VMIS1S0 tag, the first letter "V" stands for verb ("N" for noun, "A" for adjective, etc.), "I" stands for indicative, "S" for past, "1" for first person, and the other letter "S" for singular. As can be seen, each position encodes a specific type of grammatical information, and each tag contains at most seven positions, some of which may not be used in certain cases, for example, in the case of nouns.

First, we present the results of the parsing of the sentence above using the constituency formalism (Figure 10).

```
+ coor-VB_[
  grup-verb_[
    +verb_[
      +(Tomé tomar VMIS1S0 -)
    ]
    sn_[
      espec-ms_[
        +j-ms_[
          +(el el DA0MS0 -)
        ]
      ]
      + grup-nom-ms_[
        +n-ms_[
          +(pingajo pingajo NCMS000 -)
        ]
      ]
    ]
    grup-sp_[
      +prep_[
        +(en en SPS00-)
      ]
      sn_[
        espec-fp_[
          +pos-fp_[
            +(mis mi DP1CPS -)
          ]
        ]
        +grup-nom-fp_[
          +n-fp_[
            +(manos mano NCFP000 -)
          ]
        ]
      ]
    ]
  ]
  +(y y CC -)
  grup-verb_[
    patons_[
      +paton-s_[
        +(le le PP3CSD00 -)
      ]
    ]
    +grup-verb_[
      +verb_[
        +(di dar VMIS1S0 -)
      ]
    ]
    sn_[
      espec-ms_[
        +indef-ms_[
          +(un uno DI0MS0 -)
        ]
      ]
      +grup-nom-ms_[
        +n-ms_[
          +(par par NCMS000 -)
        ]
      ]
      sp-de_[
        +(de de SPS00 -)
        sn_[
          +grup-nom-fp_[
            + n-fp_[
              +(vueltas vuelta NCFP000 -)
            ]
          ]
        ]
      ]
    ]
    sadv_[
      +(de_mala_gana de_mala_gana RG -)
    ]
  ]
  F-term_[
    +(..Fp -)
  ]
]
```

Similar information is presented using the dependency formalism (Figure 9).

```
coor-vb/top/(y y CC -) [
  grup-verb/co-v/(Tomé tomar VMIS1S0 -) [
    sn/dobj/(pingajo pingajo NCMS000 -) [
      espec-ms/espec/(el el DA0MS0 -)
    ]
    grup-sp/sp-obj/(en en SPS00 -) [
      sn/obj-prep/(manos mano NCFP000 -) [
        espec-fp/espec/(mis mi DP1CPS -)
      ]
    ]
  ]
  grup-verb/co-v/(di dar VMIS1S0 -) [
    patons/iobj/(le le PP3CSD00 -)
    sn/dobj/(par par NCMS000 -) [
      espec-ms/espec/(un uno DI0MS0 -)
      sp-de/sp-mod/(de de SPS00 - ) [
        sn/obj-prep/(vueltas vuelta NCFP000 -)
      ]
    ]
    sadv/cc/(de_mala_gana de_mala_gana RG -)
  ]
  F-term/modnomatch/(..Fp -)
]
```

As mentioned above, it is easier to use dependencies, because in this case, they practically contain syntactic n-grams.

It can be seen that the three words *de_mala_gana* (lit: without enthusiasm) actually represent a single adverb.

Now, let's present the extracted syntactic n-grams. First, we present the continuous syntactic n-grams.

The syntactic bigrams (basically, there is no difference between continuous and non-continuous bigrams) are:

y tomé 'and took'
tomé pingajo 'took scrap'
pingajo el 'scrap the'
tomé en 'took in'
en manos 'in hands'
manos mis 'hands my'
y di 'and gave'
di le 'gave it'
di par 'gave pair'
par un 'pair a'
par de 'pair of'
de vueltas 'of turns'

di de_mala_gana 'gave without_enthusiasm'

The continuous trigrams are:

y tomé pingajo 'and took scrap'
y tomé en 'and took in'
tomé pingajo el 'took scrap the'
tomé en manos 'took in hands'
en manos mis 'en hands my'
y di le 'and gave it'
y di par 'and gave pair'
y di de_mala_gana 'and gave without_enthusiasm'
di par un 'gave pair a'
di par de 'gave pair of'
par de vueltas 'pair of turns'

The continuous 4-grams are:

y tomé pingajo el 'and took scrap the'
y tomé en manos 'and took in hands'
tomé en manos mis 'took in hands my'
y di par un 'and gave pair a'
y di par de 'and gave pair of'
di par de vueltas 'gave pair of turns'

We present the non-continuous syntactic n-grams without repeating the same elements (continuous n-grams), although they are also a part of the non-continuous syntactic n-grams. Note that in this case, we have to use the proposed notation for the non-continuous n-grams in order to be able to distinguish them from other possible configurations. The notation forms a part of the n-grams; it is the n-gram itself (not something additional). Thus, the new non-continuous trigrams (compared to the continuous n-grams) are:

tomé [pingajo en] 'took [scrap in]'
di [le par] 'gave [it pair]'
di [le de_mala_gana] 'gave [it without_enthusiasm]'
di [par de_mala_gana] 'gave [pair without_enthusiasm]'
par [un de] 'pair [a of]'

The new non-continuous 4-grams are:

tomé [pingajo el, en] 'took [scrap the, in]'
tomé [pingajo, en manos] 'took [scrap, in hands]'
di [le, par un] 'gave [it, pair a]'
di [le, par de] 'gave [it, pair of]'
di [le, par, de_mala_gana] 'gave [it, pair, without_enthusiasm]'
di [par un, de_mala_gana] 'gave [pair a, without_enthusiasm]'

di [par de, de_mala_gana] ‘gave [pair of, without_enthusiasm]’
par [un, de vueltas] ‘pair [a, of turns]’

Example for English

In this section, we discuss the construction of syntactic n-grams for the English language. To simplify the comparison with Spanish, we consider the translation of the sentence provided in the previous section. Note that in this case, the figure that corresponds to the tree has been generated automatically; the code, which allows it, is freely available on the author's personal web page[5].

I took the scrap in my hands and turned it a couple of times unwillingly

One can use the same parser as in the previous examples, i.e., FreeLing; however, let's try to use another parser for English that we have already

[5] http://www.cic.ipn.mx/~sidorov

mentioned before – the Stanford parser [15]. The constituency-based tree is shown in Figure 15.

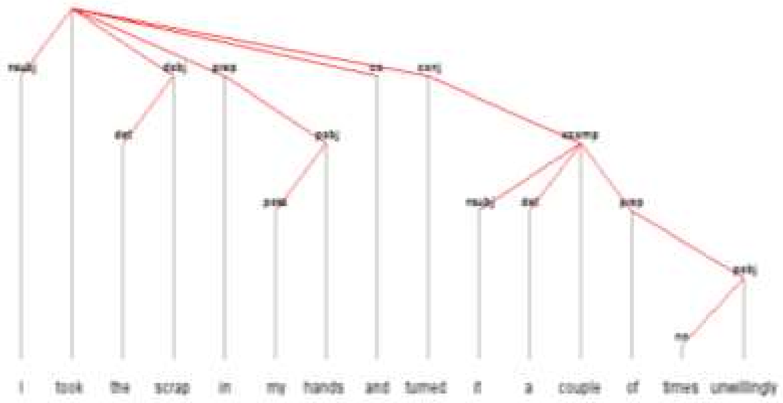

Figure 15: Constituency-based tree for the example in English.

As mentioned above, most parsers generate their output in both dependency and constituency formalisms. The output in terms of constituency grammars is the following:

```
(ROOT
  (S
    (NP (PRP I))
    (VP
      (VP (VBD took)
        (NP (DT the) (NN scrap))
        (PP (IN in)
          (NP (PRP $ my) (NNS hands))))
      (CC and)
      (VP (VBD turned)
        (S
          (NP (PRP it))
          (NP
            (NP (DT a) (NN couple))
            (PP (IN of)
                          (NP  (NNS  times)  (NN
unwillingly)))))))
    (..)))
```

As we have already discussed, in the Stanford parser, a very simple but expressive representation of a dependency-based tree is used: relation name and two words—or their POS tags or lemmas, depending on the type of elements we want to consider—along with the corresponding numbers of their positions in a sentence. The main word is mentioned first and then the dependent word, i.e., the order of the words is important. This information allows building the syntax tree in a unique way. The following is the output of the parser in the format mentioned above.

```
nsubj(took-2, I-1)
root(ROOT-0, took-2)
det(scrap-4, the-3)
dobj(took-2, scrap-4)
prep(took-2, in-5)
poss(hands-7, my-6)
pobj(in-5, hands-7)
cc(took-2, and-8)
```

```
conj(took-2, turned-9)
nsubj(couple-12, it-10)
det(couple-12, a-11)
xcomp(turned-9, 12-couple)
prep(couple-12, of-13)
nn(unwillingly-15, times-14)
pobj(of-13, unwillingly-15)
```

It can be seen that although the sentence is very similar to the one above, the other parser applied different rules, to be more precise, handled the conjunction in a different way and also committed some errors: for example, the word *unwillingly* was incorrectly related to *of* instead of *turned*; the word *it* was related to *couple* and not to *turned*. However, the parser errors do not conceptually affect our discussion, since our task is not to improve the parser but to apply existing tools. It is also worth mentioning that eventually the parsers are improving. Another interesting consideration that has been already mentioned is that parser errors are usually related to various types of syntactic ambiguity.

Now we proceed to the construction of continuous and non-continuous syntactic n-grams. Syntactic bigrams—recall that in principle, there is no difference between continuous and non-continuous syntactic bigrams—are the follows:

took I
took scrap
scrap the
took in
in hands
hands my
took and
took turned
turned couple
couple it
couple a
couple of
of unwillingly
unwillingly times

The continuous syntactic trigrams are:

took scrap the

took in hands

in hands my

took turned couple
turned couple it

couple turned a
turned couple of
couple of unwillingly
of unwillingly times

The continuous syntactic 4-grams are:

took in hands my
took turned couple it
took turned couple a
took turned couple of
turned couple of unwillingly
couple of unwillingly times

As in the previous example, we do not repeat the same elements, although the continuous syntactic n-grams are also a part of the non-continuous syntactic n-grams.

The non-continuous syntactic trigrams—the new ones (some of them may be parser errors; however, it does not affect the proposed idea, as these are errors of another type and can be corrected by improving the parser itself)—are:

took [I, scrap]
took [I, in]

took [I, and]

took [I, turned]

took [scrap, in]

took [scrap, and]

took [scrap, turned]

took [in, and]

took [in, turned]

took [and, turned]

couple [it, a]

couple [it, of]

couple [a, of]

The non-continuous 4-grams (the new ones) are:

took [I, scrap the]

took [in, scrap the]

took [and, scrap the]

took [turned, scrap the]

took [I, in hands]

took [scrap, in hands]

took [and, in hands]

took [turned, in hands]

took [I, scrap, in]

took [I, scrap, and]

took [I, scrap, turned]

took [scrap, in, and]

took [scrap, in, turned]
took [in, and, turned]
couple [it, a, of]
couple [it, of unwillingly]
couple [a, of unwillingly]

Note that in this case, we took the elements of the non-continuous syntactic n-grams in the order of their appearance in the text. As mentioned above, another option is to sort them in some way, for example, alphabetically.

Chapter 11. Automatic analysis of authorship using syntactic n-grams

Corpus preparation for the automatic authorship attribution task

We have conducted various experiments [46] in order to test the usefulness of the concept of syntactic n-grams. Essentially, we consider the task of authorship attribution, i.e., there are texts for which the authors are known and a text for which we have to determine the author (among the considered authors only). In our case, we use a corpus composed of texts written by three different authors.

The task of authorship attribution is clearly a classification task: the authors correspond to the names of the classes, the classes are the texts for which the authors are known, and the task is based on the decision to which class a text belongs [2, 29, 33, 52]. The features that we use are traditional n-grams and syntactic n-grams ranging in size from two to five. We also employ a tool that makes it easy to apply various available classification algorithms: the WEKA system [25].

We use a corpus composed of works by the following authors: Booth Tarkington, George Vaizey, Louis Tracy (the authors wrote in English in the nineteenth century). The composed corpus contains five novels by each author for training, in total 11 MB, and three works by each author for classification, in total 6 MB [46, 47, 48]. The Stanford parser is used to obtain the syntactic n-grams.

Evaluation of the authorship attribution task using syntactic n-grams

For the design of experiments, we use profiles of various sizes. The term "profile" means that we use the corresponding number of the most frequent n-grams, for example, for the profile of 400, we use 400 n-grams with greater frequency in the training corpus, etc.

We apply a standard classification algorithm called "support vector machine" (SVM). It is known that for many tasks the support vector machine algorithm outperforms other classification algorithms.

Classification results (for the authorship attribution task) for bigrams are shown in Table 2, and for trigrams in Table 3. To compare the results, we apply other ways of features selection: n-grams of words, n-grams of POS tags, and n-grams of characters.

Table 2: Results of the authorship attribution task for bigrams.

Profile size	**Syntactic n-grams of SR-tags**	**N-grams of POS tags**	**N-grams of characters**	**N-grams of words**
400	<u>100%</u>	90%	90%	86%
1,000	<u>100%</u>	95%	95%	86%
4,000	<u>100%</u>	ND	90%	86%

7,000	100%	ND	ND	86%
11,000	100%	ND	ND	89%

Table 3: Results of the authorship attribution task for trigrams.

Profile size	**Syntactic n-grams of SR-tags**	**N-grams of POS tags**	**N-grams of characters**	**N-grams of words**
400	100%	90%	76%	81%
1,000	100%	90%	86%	71%
4,000	100%	100%	95%	95%
7,000	100%	100%	90%	90%

11,000	<u>100%</u>	95%	<u>100%</u>	90%

"ND" means that not many n-grams were found for the specific profile, i.e., the number of n-grams was relatively small.

As can be seen, the method based on syntactic n-grams gives the best results. However, it is noteworthy that the top line (facility of obtaining the results) for the considered problem is quite high, as we have a great amount of training data and use only three classes (authors). The detailed information on the experiments and the experimental data for a larger number of authors can be found in our previous studies [46, 47, 48], so we do not present the detailed description in this book.

Chapter 12. Filtered n-grams

Idea of filtered n-grams

In this and the following chapters, we present two ideas related to the non-linear construction of n-grams. Recall that the non-linear construction consists in taking the elements which form n-grams in a different order than the surface (textual) representation, i.e., in a different way than words (lemmas, POS tags, etc.) appear in a text.

In the previous chapters, we discussed the concept of syntactic n-grams, where the order in which the words are taken is defined by a syntax tree.

Another option to obtain n-grams, in a different from taking the elements as they appear in a text way, is to filter out certain elements in a text. In this way, the words that are not neighbors can be considered as such.

In fact, in its simplest version, this idea can be widely applied—stop words are filtered out during the construction of n-grams.

What we propose in this chapter is to apply this idea consistently and to filter out words in texts using some criteria. The most obvious criterion is to use the *tf-idf* measure, which is discussed in detail in the first part of this book. However, one can apply other measures. Somewhat similar happens with the stop words whose *tf-idf* values are very low due to their low *idf*.

The next step is to choose a threshold in order to filter out the words. It is a matter of future experiments to determine the optimal thresholds. Note that we recommend considering not only one threshold to discard the upper or the lower part, but a combination of two thresholds in order to discard the upper part and the lower parts simultaneously. This corresponds to the intuition that the most important values are in the middle. We can generalize this idea and use not only two thresholds but a series of pairs of thresholds, although it seems unlikely that there are specific optimal ranges for the values of the features.

Note that this idea can be easily combined with the idea of syntactic n-grams: we just skip the filtered words according to the selected thresholds in our path in a syntax tree. In this case, an interesting question arises, what is to be done in the case of bifurcations? We believe that we should follow all the possible paths that originate from a bifurcation, although we should not consider the word itself in this bifurcation.

Example of filtered n-grams

Let's consider a simple example of the filtered n-grams construction. Let's assume that we are to analyze the sentence that has been already used as an example in the previous chapters.

Tomé el pingajo en mis manos y le di un par de vueltas de mala gana (lit: *I took the scrap in my hands and gave it a pair of turns without enthusiasm*) *'I took the scrap in my hands and turned it a couple of times unwillingly'*.

Suppose that we want to obtain traditional n-grams constructed out of the filtered words. We will present a table with the *tf-idf* values of each word in an imaginary collection, see Table 4.

Table 4: Possible values of *tf-idf* of words

Word	***tf-idf***
manos 'hands'	1.5
de_mala_gana 'without_enthusiasm'	1.46
upper threshold, < 1.3	
***vueltas* 'turns'**	**1.23**
***tomé* 'took'**	**1.2**
***par* 'pair'**	**0.9**
***pingajo* 'scrap'**	**0.7**
lower threshold, > 0.1	
di 'gave'	0.003

Word	***tf-idf***
el ‘the’	0
en ‘in’	0
mis ‘my’	0
y ‘and’	0
le ‘it’	0
un ‘a’	0
de ‘of’	0

Now, let’s assume that we filter out the words with very high or very low *tf-idf* values. For example, we select the thresholds: 1) “> 0.1” and 2) “< 1.3”. Only the words in bold *tomé* ‘took’, *pingajo* ‘scrap’, *par* ‘pair’, and *vueltas* ‘turns’ are in the considered range.

Note that the comparison signs of the thresholds can differ from our example, i.e., for the upper threshold we use "less-than, <" sign; one can try "greater-than, >" sign and use the words above the threshold, and not the ones below, as in the example.

Using the words that are still under consideration, we can construct the n-grams. For example, the traditional bigrams would be:

tomé pingajo 'took scrap'
pingajo manos 'scrap hands'
manos par 'hands pair'
par vueltas 'pair turns'

The continuous syntactic bigrams would be:

tomé pingajo 'took scrap'
tomé manos 'took hands'
par vueltas 'pair turns'

If we also consider non-continuous syntactic bigrams, the following bigram will be added:

[tomé, pingajo] '[took, scrap]'

It is quite curious, because as we have stated above, continuous and non-continuous bigrams coincide.

This situation changes when we start filtering out the elements of a syntax tree without deleting paths. To check how this latter bigram is constructed, please refer to Figure 9. We start with the root, which is filtered out based on the thresholds. Since we are in a bifurcation, we mark it with brackets, and then look for two unfiltered elements on each side of the bifurcation, which we separate by commas. It is noteworthy that a situation like this can only occur while constructing filtered n-grams.

Filtered n-grams of characters

Another idea that we would like to present in this book is related to the non-linear construction of n-grams of characters. In this case, one option is to filter out the words in the first place—for example, using *tf-idf*—and then to construct the n-grams of characters out of the remaining words. This idea is similar to the one already discussed in this chapter.

However, there is another option, which consists in filtering out the characters before we start constructing n-grams out of them. Basically, the idea is to filter out the characters that occupy certain positions in words or have certain features.

More specifically, regarding the features, we can filter, for example, the vowels and construct n-grams out of the remaining characters. One can try several types of characters features for various tasks.

When it comes to the positions of characters, we can consider, for example, only the first three or the last three characters of each word and ignore (filter out) the remaining characters of the word. When dealing with the Spanish language, this filtering strategy should take into account affixes (suffixes, prefixes, inflexions) and assign less weight to the n-grams that represent the stems of words. It is worth mentioning that one can do the opposite: consider the n-grams of characters that correspond to the stems and try to discard the n-grams that are related to the grammatical elements.

It is known that n-grams of characters give good results in the task of authorship attribution, i.e., reflect the personal style. The reason for this is not entirely clear, and we believe that experiments with various strategies of the non-linear construction of n-grams of characters can clarify which lexical or grammatical phenomenon is behind this type of n-grams. It is necessary to perform experiments with various parameters and with various n-grams construction strategies.

Chapter 13. Generalized n-grams

Idea of generalized n-grams

Another idea related to the non-linear construction of n-grams, i.e., using distinct elements or distinct order of their appearance in a text, is the idea of replacing words by their synonyms or by the generalized concepts that correspond to the words according to a certain ontology.

We will call this type of n-grams "generalized n-grams". Their construction is non-linear, because the n-grams are not constructed out of words according to their appearance in a text.

When replacing words by their synonyms, we are at the same ontology level, and when using their hypernyms, we are moving to a higher ontology level. In both situations, it is highly desirable to perform the word sense disambiguation task beforehand, since the selection of correct synonyms and hypernyms greatly depends on this task.

The general idea behind generalized n-grams is to reduce the lexical variety of texts, since in this way, the number of n-grams is substantially decreased. It is quite clear that this idea can be easily combined with the ideas of syntactic and filtered n-grams.

The application of information concerning synonyms is rather simple: for each word we compose the list of synonyms, e.g., we can use WordNet synsets or any thesaurus and replace the word by the first synonym in the list, and then proceed to the n-grams construction.

There are several strategies for using hypernyms available in ontologies. We can always use the current level of each word plus a constant.

$$hypernym\_level = word\_level + c$$

In this case, we have to move c levels higher in the ontology we are using.

Another possibility is to set a reasonably low ontology level and always move up from this level. If we come across a word of a very high level, it is advisable to leave it alone and not to move down to the corresponding level.

In this way, we replace the words by their synonyms or hypernyms and then construct the n-grams out of these new elements.

Example of generalized n-grams

The case of synonyms seems quite obvious. Let's consider an example of hypernyms.

Let's use the same sample sentence as in the previous chapter. Assume that we use the filtering strategy and remain with the following filtered words: *tomé* ‘took’, *pingajo* ‘scrap’, *manos* ‘hands’, *par* ‘pair’, *vueltas* ‘turns’.

As an example we use the strategy of moving only one level up in the ontology.

Assume that we have the following hypernyms: *manos* ‘hands’ → *brazos* ‘arms’ (strictly speaking, it is a holonym; however, it serves our purpose), *tomé* ‘took’ → *actué* ‘acted’, *pingajo* ‘scrap’ → *herramienta* ‘tool’, *vuelta* ‘turn’ → *movimiento* ‘movement’, *par* ‘pair’ → *número* ‘number’. Obviously, this information depends on the specific ontology, where the words and correspondences are to be found.

Now we can construct the n-grams using generalized concepts and not words as such, e.g., the bigram *tomé pingajo* ‘took scrap’ is replaced by *actué herramienta* ‘acted tool’ or *actuar herramienta* ‘act tool’. The usefulness of this replacement depends on our purposes. The advantage is that the words *martillo* ‘hammer’, *cuchillo* ‘knife’, etc. would have the same hypernym *herramienta* ‘tool’; in the same way, the words tomé ‘took’, clavé ‘stabbed', corté ‘cut’, etc. would have the hypernym *actúe* ‘acted’ or *actuar* ‘to act'.

It is a matter of future studies to determine the usefulness of generalized n-grams, and the tasks for which their application would give better results than using the other types of n-grams.

Bibliography

1. Agarwal, A., Biads, F., Mckeown, K.R.: Contextual Phrase-Level Polarity Analysis using Lexical Affect Scoring and Syntactic N-grams. Proceedings of the 12th Conference of the European Chapter of the ACL (EACL), pp. 24–32 (2009)

2. Argamon, S., Juola, P.: Overview of the international authorship identification competition at PAN-2011. In: Proc. of 5th Int. Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (2011)

3. Baayen, H., Tweedie, F. and Halteren, H.: Outside The Cave of Shadows: Using Syntactic Annotation to Enhance Authorship Attribution. Literary and Linguistic Computing, pp. 121–131 (1996)

4. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley (1999)

5. Carreras, X., Chao, I., Padró, L., Padró, M.: FreeLing: An Open-Source Suite of Language

Analyzers. In: Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04) (2004)

6. Cazés Menache, D., del Castillo, N., Mansilla, R., Pineda, L.A., Sidorov, G., Sierra Martínez, G.: El dominio de la lingüística: más allá de las ciencias exactas y naturales. UNAM, 213 p. (2009)

7. Cheng, W., Greaves, C., Warren, M.: From n-gram to skipgram to concgram. International Journal of Corpus Linguistics 11, no. 4, pp 411-433 (2006)

8. de Marneffe, M.C., MacCartney, B., Manning, C.D.: Generating Typed Dependency Parses from Phrase Structure Parses. In: Proc. of LREC (2006)

9. Díaz Rangel, I., Sidorov, G., Suárez-Guerra, S.: Creación y evaluación de un diccionario marcado con emociones y ponderado para el español. Onomazein, 29 (2014)

10. Dumais, S.T.: Latent Semantic Analysis. Annual Review of Information Science and Technology 38: 188 (2005)

11. Fillmore, Ch., Langendoen, T. (eds): Studies in Linguistic Semantics. (1971)

12. Firth, J.R.: A Synopsis of Linguistic Theory. In: Palmer, F.R. (ed), (1968) Selected Papers of J.R. Firth 1952-59. London/Harlow: Longmans (1957)

13. Gale, W.A., Church, K.W.: A program for Aligning Sentences in Bilingual Corpora. In: Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics, Berkeley, California (1991)

14. Gelbukh, A., Alexandrov, M., Han, SangYong: Detecting Inflection Patterns in Natural Language by Minimization of Morphological Model. In: A. Sanfeliu, J.F. Martínez Trinidad, J.A. Carrasco Ochoa (Eds.) Lecture Notes in Computer Science N 3287, Springer-Verlag, pp. 432–438 (2004)

15. Gelbukh, A., Calvo, H., Torres, S.: Transforming a Constituency Treebank into a Dependency Treebank. Procesamiento de Lenguaje Natural, No 35. Sociedad Española para el Procesamiento de Lenguaje Natural (SEPLN) (2005)

16. Gelbukh, A., Sidorov, G.: Procesamiento automático del español con enfoque en recursos léxicos grandes. IPN, 307 p. (2010)

17. Gelbukh, A., Sidorov, G.: Approach to construction of automatic morphological analysis systems for inflective languages with little effort. Lecture Notes in Computer Science, N 2588, Springer-Verlag, pp. 215–220 (2003)

18. Gelbukh, A., Sidorov, G.: Zipf and Heaps Laws' Coefficients Depend on Language. Lecture Notes in Computer Science N 2004, Sringer-Verlag, pp. 330–333 (2001)

19. Gelbukh A., Sidorov, G.: Alignment of Paragraphs in Bilingual Texts using Bilingual Dictionaries and Dynamic Programming. Lecture Notes in Computer Science, N 4225, Springer-Verlag, pp 824-833 (2006)

20. Gelbukh, A., Sidorov, G., Guzman-Arenas, A.: Use of a weighted topic hierarchy for text retrieval and classification. Lecture Notes in Artificial Intelligence, No. 1692, Springer, pp. 130–135 (1999)

21. Gelbukh, A., Sidorov, G., Han, SangYong: On Some Optimization Heuristics for Lesk-Like WSD Algorithms. Lecture Notes in Computer Science, N 3513, Springer-Verlag, pp. 402–405 (2005)

22. Gelbukh, A., Sidorov, G., Han, SangYong, Hernández-Rubio, E.: Automatic Enrichment of Very Large Dictionary of Word Combinations on the Basis of Dependency Formalism. Lecture Notes in Artificial Intelligence N 2972, Springer-Verlag, pp 430-437 (2004)

23. Goldsmith, J.: Unsupervised Learning of the Morphology of a Natural Language. Computational Linguistics 27:2, 153-198 (2001)

24. Habash, N.: The Use of a Structural N-gram Language Model in Generation-Heavy Hybrid Machine Translation. LNCS, 3123, pp. 61–69 (2004)

25. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update; SIGKDD Explorations, 11(1) (2009)

26. Hernández-Reyes, E., Martínez-Trinidad, J. Fco., Carrasco-Ochoa, J.A., García-Hernández, R.A.: Document Representation Based on Maximal Frequent Sequence Sets. LNCS 4225, pp. 854–863 (2006)

27. Inteligencia Artificial. G. Sidorov (Ed.), Alfa Omega, (2014)

28. Jiménez-Salazar, H., Pinto, D., Rosso, P.: Uso del punto de transición en la selección de términos índice para agrupamiento de textos cortos. Procesamiento del Lenguaje Natural, 35, pp. 383-390 (2005)

29. Juola, P.: Authorship Attribution. Foundations and Trends in Information Retrieval. 1(3):233–334 (2006)

30. Jurafsky, D., Martin, J.: Speech and Language Processing. Prentice Hall (2009)

31. Kay, M., Roscheisen, M.: Text-translation alignment. Computational Linguistics, 19(1):121–142 (1993)

32. Khalilov, M., Fonollosa, J.A.R.: N-gram-based Statistical Machine Translation versus Syntax Augmented Machine Translation: comparison and system combination. In: Proceedings of the 12th Conference of the European Chapter of the ACL, pp. 424–432 (2009)

33. Koppel, M., Schler, J., Argamon, S.: Authorship attribution in the wild. Language Resources and Evaluation 45(1):83–94 (2011)

34. Lesk, M. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. Proc. of ACM SIGDOC Conference, Toronto, Canada, pp. 24-26 (1986)

35. Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, MA (1999)

36. Medina Urrea, A.: Automatic Discovery of Affixes by means of a Corpus: A Catalog of Spanish Affixes. Journal of Quantitative Linguistics 7(2), pp. 97–114 (2000)

37. Miranda-Jimenez, S., Gelbukh, A., Sidorov, G: Generación de resúmenes por medio de síntesis de grafos conceptuales. Revista "SIGNOS. Estudios de Lingüística", 47(86) (2014)

38. Montes y Gómez, M., Gelbukh, A., López López, A., Baeza-Yates, R.: Flexible Comparison of Conceptual Graphs. Lecture Notes in Computer

Science N 2113, Springer-Verlag, pp.102-111 (2001)

39. Padró, L., Collado, M., Reese, S., Lloberes, M., Castellón, I.: FreeLing 2.1: Five Years of Open-Source Language Processing Tools. In: Proceedings of 7th Language Resources and Evaluation Conference (LREC 2010), ELRA La Valletta, Malta (2010)

40. Padró, L., Stanilovsky, E.: FreeLing 3.0: Towards Wider Multilinguality. In: Proceedings of the Language Resources and Evaluation Conference (LREC 2012), ELRA, Turkey (2012)

41. Pado, S., Lapata, M.: Dependency-based construction of semantic space models. Computational Linguistics, 33(2): 161–199 (2007)

42. Pichardo-Lagunas, O., Sidorov, G., Cruz-Cortés, N., Gelbukh, A.: Detección automática de primitivas semánticas en diccionarios explicativos con algoritmos bioinspirados. Onomazein, 28 (2013)

43. Reyes, J.A., Montes, A., González, J.G., Pinto, D.E.: Clasificación de roles semánticos usando

características sintácticas, semánticas y contextuales. Computación y sistemas, 17(2): 263–272 (2013)

44. Sierra, G., Alarcón, R.: Recurrent patterns in definitory context. In: Proc. CICLing-2002, Computational Linguistics and Intelligent Text Processing. Lecture Notes in Computer Science N 2276, Springer-Verlag, pp. 438–440 (2002)

45. Sierra, G., McNaught, J.: Natural Language System for Terminological Information Retrieval. Lecture Notes in Computer Science, N 2588, Springer, pp. 543–554 (2003)

46. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., Chanona-Hernández, L.: Syntactic Dependency-based N-grams as Classification Features. LNAI, 7630, pp. 1–11 (2012)

47. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., Chanona-Hernández, L.: Syntactic Dependency-Based N-grams: More Evidence of Usefulness in Classification. LNCS, 7816 (Proc. of CICLing), pp. 13–24 (2013)

48. Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., Chanona-Hernández, L.: Syntactic N-grams as Machine Learning Features for Natural Language Processing. Expert Systems with Applications, 41(3): 853–860 (2014)

49. Sidorov, G.: N-gramas sintácticos y su uso en la lingüística computacional. Vectores de investigación, 6(6): 1–15 (2013)

50. Sidorov, G.: Non-continuous syntactic n-grams. Polibits, 48: 67–75 (2013)

51. Sidorov, G.: Syntactic Dependency Based N-grams in Rule Based Automatic English as Second Language Grammar Correction. International Journal of Computational Linguistics and Applications, 4(2): 169–188 (2013)

52. Stamatatos, E.: A survey of modern authorship attribution methods. Journal of the American Society for information Science and Technology 60(3): 538–556 (2009)