TÉCNICO LISBOA

MY BELOVED SEBENTA: Natural Language Processing in a Nut(s)shell

DO NOT DISTRIBUTE

Luísa Coheur

Contents

Acknowledgements

I'd like to express my gratitude to all the students that helped me to improve this document, namely to Fábio Alves, Renata Pacheco, Alexandre Pereira, Rui Silva, Pedro Pereira and Ricardo Rodrigues. Special thanks to Ricardo Silva, who gave me very detailed comments about the whole document and without being asked, translated two subsections that were still in Portuguese. As usual, the responsibility for any imprecision lies with me.

About this document

This document was mostly written during my sabbatical leaves (spring: 2014, 2015). Some parts were never really finished and it soon became out-of-date, namely the section dedicated to semantics. Anyway, it is mostly ok.

Greetings!

Are you sure you are ready?

Hello and welcome to the Natural Language (NL) class, where you will learn how difficult to formalize and deal with natural languages are. You have probably learned to program in C, Python and Java, but during this semester you will understand that those are languages for babies. Do you think C syntax is complicated? Do you think that a compiler for Java is difficult to write? Please... try to do it for Portuguese!

In fact, one of the major problems when dealing with NLs is that they are inherently ambiguous, contrary to formal languages that are designed to be unambiguous. Words are ambiguous, sentences are ambiguous, and even when we think they aren't... well... they are. Really. Consider for example the sentence *Read the book*. If you say it referring to a book, it means that the book is worth to be read. Nevertheless, if you say it about a movie, it is probably because the movie is bad. Another example: you enter the class in time, and you professor makes you a huge smile and says "Good Morning!", which really means "Good Morning!". However, if you arrive late, she will probably also say "Good Morning", but (notice it!) in a different tone. This time, the interpretation of that sentence is probably something as "you awful creature, don't you have a watch (or other more modern gadget) that will allow you to arrive on time to my beloved lectures?". Ugh....

The other major problem is variability. We can say the same thing in many different ways. That is, there is a panoply of ways of saying the same thing. In other words: we can verbalise something by using different words. Meaning that the same think can be expressed by using different word sequences. Or... hum... Got the idea?

Nothing in Natural Language Processing (NLP) is for free, nothing rules 100% of the time, and you will have to live with probabilities. Yes, I know, probably not your favourite class, but there is nothing I can do about it: we will have to cope with incertitude to survive. So a question arises: are you sure you are ready?

What will we be talking about?

Are you still there? If so, maybe you are brave enough to embrace this voyage through NLP. Anyway, I'm (almost) sure you will love it: there are tools for so many things, the set of applications is unlimited and you just need to have imagination and tenaciousness. You will learn how to play with the NLP cards and I'll have the pleasure to teach you how to. So, that's it: welcome to NL courses and enjoy it (a lot)! :-)

Chapter 1

What? Why? Who? Where?

In this chapter we will answer to several questions regarding NL and NLP.

1.1 What does Natural Language mean?

Let us start by making sure that you understand what NL means. You can find many definitions in the literature, but in my opinion, the concept can be clearly understood if the following semantic is given to the words "Natural" and "Language":

- The word "natural" expresses the natural evolution due to people communication.
- The word "Language" means a grammatical system (that is, with its own rules) that is used to communicate.

In fact, there are new words emerging everyday, as for instance *kunami*, *troikiano*, *stressar*, *sanita*, *quispo*. Some of these words die after some time; others are officially recognized and added to dictionaries. A good example is the word *stressar*, which is currently in the Priberam's dictionary[1]. It should also be clear that there are some rules controlling what can be said. For instance, you can say *Eu adoro as aulas de Língua Natural*, but not **aulas eu Língua as Natural adoro*.

So, Portuguese and English are NLs and C++ is not (there is "no natural evolution due to people communication" within programming languages). However, the distinction is not that simple: what about **Esperanto**? Esperanto, created by Ludwig L. Zamenhoff (a Polish physician) in 1887, is the most commonly used "natural artificial" (or "artificial natural") language. It has the advantage of having an extremely regular grammar (for instance, there are no irregular verbs) and of keeping a simple relation between written/spoken text (for each letter there is one single sound), which leads to a faster learning. Esperanto's vocabulary came mostly from Romance languages (a minor part came from Germanic languages, English and Slavic languages). It was artificially built, but it has had a natural evolution though time. Therefore, it is a NL or not? Well, some people consider Esperanto

[1] By the way, this site (`http://www.priberam.pt/dlpo/`) is a "must".

as a NL and other people don't. Google will show you many opinions on this subject and will also show you other constructed languages such as **Ido** and **Interlingua**. To conclude, although not leading to so much discussion, what about Frantuguês (`http://www.youtube.com/watch?v=_jPufypajn8`)? There is a dictionary already available!

1.2 Why is it so difficult to deal with Natural Language?

NLs are definitely difficult to deal with, some being more complicated than others. For instance, Portuguese possesses much more **specific verb tenses** than English does, and, thus, the conjugation of verbs is more complex. The existence of **agreement between words** is also a source of problems in many languages. For instance, in Portuguese, nouns and adjectives have to agree, which is not the case in English. In German, besides the *Masculine* and *Feminine*, there is also the gender *Neuter*. As we will see throughout this course, there are NL with very strange rules.

Nevertheless, despite the fact that some NLs are particularly complex, all of them share the same two main problems, which are the ones that makes them so difficult to deal with from a computational point of view:

- **Linguistic variability**: the possibility of expressing the same thing in many, many, many, many, many, many, many, many, many, many different ways.
- **Ambiguity**: the fact that words/expressions/sentences have several meanings (which leads to the biggest mess in the world).

Exercise 1: Linguistic variability 1

Try to find 10 ways of saying that you adore your NL courses. You will see that it is not that difficult to find 10 alternative sentences, but many more exist (by the way: instead of *alternative sentences*, use from now on the word **paraphrase**. I'm sure your friends will be quite impressed).

Exercise 2: Linguistic variability 2

Imagine that you are building an agent that answers questions about IST. Write a list of FAQs that the agent should be able to answer and then write all the paraphrases of each question in the FAQs list that you can imagine. Abort the process eventually.

Considering ambiguity, this is the main characteristic that makes NL and artificial languages so different. Just to give you a brief idea how ambiguous is your own language (in case you haven't noticed yet), consider the following sentences:

1. *Encontramo-nos no banco ao lado do quiosque.*
2. *Cuidado com o Degrau!*
3. *O Pedro viu o homem na montanha com o telescópio.*

4. *Quero um hotel com piscina que tenha água quente.*
5. *Quero um hotel com piscina que tenha suite nupcial.*
6. *O general bateu a bota.*

The word *banco* is ambiguous: it can be that thing where you can sit or the place where you can put your money. The word *Degrau* is not ambiguous, but if somebody tells you that *Degrau* is the name of his/her dog, the meaning of the sentence completely changes. We call this **lexical ambiguity** (notice that the fact that it starts with a capital letter can give you a clue that it is not being used in the usual way).

The third situation is a classic example of what we call **syntactic ambiguity**, because the main problem is the relation between the syntactic constituents and how to connect them. Who had the telescope? Pedro or the man in the mountain? The man was in the mountain or it was Pedro who was in the mountain? This case also illustrates a complicated phenomena, known in NLP as the **PP-attachment** problem (PP, stands for "Prepositional Phrase"), where the machine needs to identify to which phrase the PP is attached. However, other constituents can also pose a similar problem. This is the case of examples 4 and 5. In the first case the relative sentence *que tenha água quente* is almost sure related with *piscina* and, in the second case, *que tenha suite nupcial* is related with *hotel*. You should understand that the main problem is that we, humans, are able to decide that some interpretations are possible and others aren't; however, it is very difficult to make the computer understand which are the correct interpretations.

Exercise 3: Syntactic Ambiguity
Try to find all the interpretations of the sentence "*O Pedro viu o homem na montanha com o telescópio*".

Finally, consider the sentence *O general bateu a bota*. *Bateu a bota* is an **idiomatic expression** that means *to die*. Thus, from that sentence we are not able to understand what happened with the general, as this sentence can be understood both ways – literally and figuratively.

The main problem with ambiguity is that sometimes even information sources that should be 100% clear, are ambiguous. For instance, consider the sentence (taken from a Portuguese newspaper):

Espanha: Portugal será o último país da UE a recorrer a ajuda externa.

The first time I read it, I thought that Spanish people were saying that Portugal would never ask for help, that is, we would be the last country in the world that would ask for help (ya, I'm so naive). Then, I understood that they were saying that we would really be the ***last*** country asking for help, because they would never do it.

In fact, if during one single day, you keep close attention to what people tell you (even professors), you will see the incredible doses of ambiguity that we produce everyday. However, as you will see, the majority of situations can be disambiguated, mainly due to the **context** in which they occur, as you can see in the following examples. Notice

that, once again, while it may be easy for you to understand what is now the correct meaning of each one of the previous examples, it is extremely difficult to teach a computer to do it.

1. *Encontramo-nos no banco ao lado do quiosque. Aquele que tem um multibanco na entrada.*
2. *Cuidado com o Degrau! Morde que se farta.*
3. *O Pedro viu o homem na montanha com o telescópio. O homem estava a nadar costas no lago.*[2]
4. *O general bateu a bota. Conseguiu assim sacudir a pedra que o estava a magoar.*

Nevertheless, it is not only the textual context that helps to disambiguate a sentence. Consider the sentence *O professor não corrigiu um exame!* and try to say it loud with the two following meanings: in the first, the professor corrected all the exams, except one; in the second, the professor did not correct a single exam. Strange, isn't it? An innocent sentence with two different meanings that depend on how you said it (like the "Good morning" case). In fact, this is not so unusual if we think about, for instance, irony. This opens many possibilities to the meaning of almost any sentence.

1.3 Who are the brave people that work in NLP?

Figure 1.1: Brave people working in NLP

In Portugal there are many research institutes working in NLP, including the one from INESC-ID (L2F). In fact there are research institutes at Porto, Aveiro, Minho, Algarve, Évora, and in companies such as Porto Editora, Priberam and Unbabel. In the rest of the world there are many and many groups working in NLP. Around 2012, Jurafsky [5] (the author of the book that I recommend for this course and from which this material is strongly inspired) decided to give on-line NLP courses. These courses – that I also highly recommend – engaged more than 40.000 people (just search for them in the internet – not the people, the courses[3]), and that was before the recent

[2] Still ambiguous!

[3] See what an anaphoric ambiguity can do? :-)

NLP hysterical hype.

You can find people with different backgrounds working in NLP. If you take a look at L2F, you will find people from Computer Science, Electronic Engineering and Linguists. A psychologist would also be more than welcome to help us understand feelings/emotions/personalities, as this is strongly related with the way people express themselves (remember that NL is only one of the things we use to communicate; gestures and facial expressions are also used in the communication process).

1.4 What about the involved knowledge?

In fact, having people with different backgrounds in a research group can be a big advantage. For instance, linguists study, analyze and dissect language until exhaustion. Some computer scientists profit from their knowledge to create their symbolic (rule-based) models/tools. However, it is also true that other computer scientists completely ignore linguists and use other weapons against NL: Statistics/Machine Learning. For instance, in applications such as Machine Translation, very little linguistically motivated sources of information lead to amazing results. Sometimes, even the most polluted ones offer interesting results (have you ever looked at a phrase table in Statistical Machine Translation?). However, the control of different types of knowledge is certainly very important in this field and there are many hybrid systems combining linguistic information with statistics and Machine Learning techniques. The different types of knowledge involved in NLP are:

- **Phonology**: studies the relation between words and sounds. For instance, try to say the word *almoço*. You don't know how to say it, do you? This is because it sounds differently if you say *Hora do almoço* or *Hoje almoço tarde*.
- **Morphology**: related with words and its units of meaning. Here is where we dissect words (ex: gat+o/os/a/as) and understand how important are some of their constituents for some applications.
- **Syntax**: studies how words can be put together in order to originate sentences. We will review some of the concepts that you have learned in the Compilers course, namely the concept of grammars. We will also study several algorithms for syntactic analysis. Hope that at the end of this chapter you will be totally convinced that NLs are much more complicated than the formal languages you have been studying in the past.
- **Semantics**: respects the study of (the literal) meaning. This is the level of information where everything turns to be almost impossible to control in NLP. Do you remember in LP or IA when you were using First Order Logic to represent the meaning of a sentence? Now try to do that automatically... for all sentences in your own language (if you succeed, don't forget to tell me how you did it). We will take a look on the main efforts and we will learn about some interesting sources of information that can help improving many NLP applications.
- **Pragmatics**: studies how the context contributes to the meaning.

- **Discourse**: study of language beyond the sentence boundaries. When we read a text, an object is not always referred to in the same manner. Thus, in order to understand the text, we need to be able to follow the different denominations of an object, that is, to establish its *co-references* (ex: *John loves his brother. He is so nice.*).
- **Knowledge about the word**: to allow inferences. (ex: *murder* and *to kill* mean the same in some contexts).

It is a fact that in NLP more knowledge means more possibilities, but sometimes it is difficult to integrate different forms of knowledge. You can find in Linguateca's page[4] many linguistic resources for Portuguese.

1.5 Which are the main weapons used in NLP?

There are **techniques**, **formalisms**, **tools** and **linguistic resources** that are extremely important in NLP. We will study many of these "weapons" in this course. It should be clear that there are tools for almost everything in NLP (although most of the times not in the language/domain in which we are working). My advice: don't implement your own morphologic analyzer or syntactic parser. Somebody already did it and you are just losing your time. The only reasons I can accept for implementing an existing tool are the following: a) the existing tool is really bad; b) the existing tool is a black box, and you need to have control of some of its internal behaviour.

1.6 Can you give me examples of NLP applications?

People working in NLP are trying to create computational models that can be used in the many involved problems they have at hands. There are so many targets that it is difficult to enumerate all of them. Here follow some applications that you probably heard about:

- Google Translator;
- Siri;
- Microsoft Word spell and grammar checker;
- Wolfram Alpha.

Involved in these systems are research lines such as:

- Speech Recognition/Synthesis: allow you to transcribe what is said and to transform text into speech, respectively.
- Speech/Natural Language Understanding: which allows you to understand what was said.

[4] `http://www.linguateca.pt`

- Machine Translation: translation systems. These can also take speech as input and return speech in the output (Speech-to-Speech Machine Translation).
- Conversational agents: systems with which you can talk to in order to ask for information or make them do some task for you (reserve tickets, switch on/off the TV, etc.);
- Question/Answering (QA): systems that contrary to search engines, accept queries in NL (also called *questions* :-)) and try to return the exact answer to that question (and not a set of documents where the answer may be found);
- Sentiment Analysis: systems that analyze the polarity of some text. They can help you, for instance, to decide to buy a book (or not), as some of these systems analyze people comments about a certain item (a car, a book, a computer, a politician, etc.) and tell you if the general opinion is positive or not;
- Summarization: systems that make synopsis of texts (ya, it would be great to have a system summarizing your books);
- ...

Chapter 2

Experimental Setup

Hi! Depending on the problem at hand, your experimental setup will vary. However, there are some ingredients that are usually common in all the experiments, across research areas, including in NLP. For instance, the existence of a baseline or the way you should break your datasets, so that your experiments are trustworthy. We will discuss these concepts in this chapter. I hope they will also be useful to you when preparing your master thesis.

2.1 The Scientific Method

Do you remember studying the **scientific method** in science classes, when you were a kid? According to the Merriam-Webster dictionary, a scientific method consists of

"the collection of data through observation and experimentation, and the formulation and testing of hypotheses".

In order to explain different phenomena, scientists suggest hypotheses, and design experiments to test them. In NLP you should also follow the scientific method. First, you should know something about the phenomena that is the target of your research and you should be able to perfectly identify the goals of your work and the hypothesis (or hypotheses) that you want to test. Then, you should think on how you will test this – which architecture, which **corpora** (corpora is the plural of corpus and no, it doesn't mean "dead body"), which metrics, which tools – and design the experiments that will corroborate (or not) your point. Remember that if you change more than one variable at the same time, you will not be able to understand what was responsible for the obtained results. Next, you implement the system that will allow you to run your experiments, you will run it, you will evaluate the results using (known) evaluation metrics and you will see if your hypothesis still rules. If it does, perfect, you made your point! Nice! Clap-clap-clap! If it doesn't, don't cry and/or cross your arms, you should try to understand what went wrong (make a detailed error analysis – results don't bite) and reformulate your hypothesis. Notice that not having a good result may also be interesting: if your hypothesis is well justified, I'm sure many people will want to know about it. There are even venues that accept papers with bad results. If you

follow a promising path that led to a dead-end why not to warn tour research fellows about it?

Notice, also, that if similar systems also report results, you should run your experiments with the same datasets and in the same conditions, so that you can make a straightforward comparison with them. This is exactly the process that I want you to follow in this course (this is really important!).

2.2 Corpora

2.2.1 Building Corpora

One of NLP researchers' best friends are corpora. Regardless of whether your research falls into a symbolic or a statistical approach, corpora are always welcome. However, collecting corpora and processing it, so that it can be used for training or testing, is a time consuming and expensive task, that sometimes can only be done by experts. Would you be able to label a text with the morpho-syntactic category of each word? Well... me neither. Many researchers use the Mechanical Turk[1] (or equivalent) for building corpora, and many people from the NLP community are using "Requesters" to create several types of corpora.

Many efforts were done in recent years and there are many corpora at your disposal, as well as many annotation guidelines, which can be used depending on your application needs. Check, for instance, the Brown Corpus, Floresta Sintáctica, Público, the Stanford Question Answering Dataset (SQuAD). There are also organizations that centralize many corpora and sell them. See, for instance, the European Language Resources Association (ELRA) or the Linguistic Data Consortium (LDA) webpage. Interesting, NLP datasets can be found, for instance, in Kaggle [2]. Nevertheless, it might happen that you will have to build your own corpora (at least for testing your application). If so, take some time to think how you will do it and don't neglect this task, as the success of your work is highly dependent on it. Also, notice, that many questions arise when you are building corpora. For instance, how to segment it: should I use "no" or "em o" (in Portuguese, "no" = "em o" (in the))? Another problem lies with corpora annotation: which labels should I use? Are these labels expressive enough? Can I find guidelines somewhere? Do I need to define my own? (uhhh... scary)

2.2.2 Inter-annotator Agreement

The concordance among **raters** (also called inter-rater reliability or inter-rater agreement) tells you how much consensus there is in the ratings given by judges on some task. For instance, consider that I give you a text and tell you to classify a set of answers of a virtual agent as being appropriate or not, in a scale from 1 to 10. Consider also that I propose the same exercise to a friend of yours. Then, I will see how much agreement there is between you. If the degree is low, either the scale is defective or you need to be trained in doing such job.

[1] http://en.wikipedia.org/wiki/Amazon_Mechanical_Turk
[2] https://www.kaggle.com

There are many measures that can be used to determine inter-annotator agreement. An example of one such measure is the **Cohen's kappa coefficient** [2], as defined in Equation 2.1.

$$K = \frac{Pr(a) - Pr(e)}{1 - Pr(e)}. \tag{2.1}$$

Where:

- Pr(a) is the relative observed agreement among raters;
- Pr(e) is the probability of random agreement.

As an example, consider that two annotators have to classify some questions with tags A, B or C, and that Table 2.1 summarizes the inter-annotator agreement results. For instance, considering the value 3, it means that annotator 1 (A1) labeled 3 things with tag B and annotator 2 (A2) with tag C. As another example, the number 21 means that both labeled 21 instances with tag B.

A1/A2	A	B	C	Total
A	**15**	0	0	15
B	1	**21**	3	25
C	0	0	**1**	1
Total	16	21	4	**41**

Table 2.1: Inter-annotator agreement.

Annotators agreed on the evaluation of 37 of the 41 questions:

- 15 A
- 21 B
- 1 C

Therefore, the relative observed agreement among raters, Pr(a), is:
(15 + 21 + 1)/41 = 0.90.
To calculate the probability of chance agreement, Pr(e), we need to find:

- the probability that both would say "A" randomly. Considering that 15/41 is probability of A1 saying A and 16/41 is the probability of A2 saying A, we have 15/41*16/41 = 0.14
- the probability that both would say "B" randomly (21/41*25/41 = 0.31), and
- the probability that both would say "C" randomly (4/41*1/41 = 0.002).

Then, we sum all the obtained values: 0.14 + 0.31+ 0.002. Pt(e) is, then, 0.46.

Finally, we can calculate the kappa coefficient (K), which is 0.82.

Now, you are probably thinking if this is good or bad. Well, there is not much agreement on this (ahahah!), but some authors (and the Wikipedia) consider:

- values < 0: no agreement;
- 0 – 0.20: slight agreement;
- 0.21 – 0.40: fair agreement;
- 0.41 – 0.60: moderate agreement;
- 0.61 – 0.80: substantial agreement;
- 0.81 – 1: almost perfect agreement.

Thus, the previous result can be considered an almost perfect agreement (ufa!).

To conclude, let me just note that this measure can only be used with two raters; a similar measure of agreement, called the Fleiss' Kappa [3] can be used when more than two raters are involved (and many other measures exist with this purpose).

Notice also, that this was just to give you an idea of what the concordance among raters is, as **there are many different measures to calculate the agreement between raters, depending on the task in hand**. We will talk about this in class.

2.2.3 Training, Testing and Development Sets

Regarding the usage of corpora, several restrictions need to be respected if you want to have a methodologically correct work. So, please, during the whole course, don't forget the following: divide your corpora in **training and test sets**. Typically, 80%–90% of the corpus is used for training and the rest for testing. However, when we are dealing with big data, a smaller dataset for testing can be used. very important: the test set (also called reference) should only be used by the end of your experiences!!!. You should not look at it before that. If you are (machine) learning from a corpus or if you are writing rules for it, you should use the training corpus, never the test dataset. Otherwise, when you evaluate your system you will do it with something that the learning algorithm (or yourself) already know, which leads to an untrustworthy evaluation.

You are probably thinking: *but, how do I know when I can "close" my experiments? That is, how do I know if results are good enough to put an end in the development of my system if I cannot use the test set?*. Good question! You need to test your system before testing it with the test set (test, test, test!). You should take part of your training set and extract a **development set** from it, which simulates a test corpus. Therefore, before testing your system with the test corpus you can test it with the development corpus, look at results, tune your work and test again and again with the development set. There is no problem with that. The final evaluation should be made with the test corpus that was left untouched until then.

Notice also that sometimes it is so time-consuming and expensive to build some types of corpora (for instance, it takes years (literally) to annotate a video with people communicating via sign languages and experts are absolutely needed for this task) that only a small collection is built (sometimes called the **gold-collection**) and used for testing, as there is no way to train a system with it.

2.2.4 Cross Validation

If you are using your corpus for a learning purpose, you can also not break it in training/development/testing and, alternatively, make a **k-fold cross validation**: you split your corpus into k sets, train your system using k-1 sets and test it with the remaining one. Usually, multiple rounds are performed using different partitions and, at the end, the average over the rounds is returned.

2.2.5 The Nature of the Corpus

There is another thing that you should pay much attention to: **the nature of these corpora needs to be the same for both training and testing**, otherwise you won't be able to conclude much. For instance, it is a really bad idea to train your language model with a corpus of newspapers and then test it with a book for children; or to learn to identify writers in a poetry corpus and test the authors' identification in other literature genres. Recently, with this idea of pre-trained models (we will arrive there), there are works that try to adapt systems trained with a certain dataset to other domains (or languages). But this is another story. If you train a model on a corpus of movie reviews, labeled with the sentiment of the review, don't expect great results if you apply this sentiment predictor to a dialogue corpus. One of my students is exploring this idea and results are far from epic.

2.3 Evaluation

2.3.1 Baseline

One of the first things you should do is to implement a **baseline**. The baseline represents the starting point of your system: a specific configuration with which results are known. Then, you will design your experiments (add other corpus, change some parameters/algorithms, etc.) and compare the attained results with the baseline, in order to check if you are improving it. For instance, if your master thesis comes in sequence of a previous master, the results obtained by your colleague can be used as the baseline. Then, your work will be to (try to) improve them. Again, don't be depressed if you cannot improve results. If your idea is virtuous, you still have a story to tell in your master thesis.

2.3.2 Extrinsic vs. Intrinsic Evaluation

If you evaluate your NLP system independently of the context in which it will be used, you are making an **intrinsic evaluation** of it; however, if you evaluate your NLP system as a component of a more complex NLP system, you are performing an **extrinsic evaluation** of it.

As an example, consider that you have developed a **Question Classifier**, that is, a module that tells you what type of answer should be returned for any question (for instance, location for *Where is Marinhais?* or people for *Who won the Oscar for best actor in 2013*[3]). When you evaluate how good your Question Classifier is in returning

[3] Hum... Daniel Day-Lewis.

the correct answer type, you are performing an intrinsic evaluation of it. However, if you use your Question Classifier within a Question Answering (QA) system, the results of the QA system allow you to extrinsically evaluate your Question Classifier. What's funny is that sometimes a module with a (relatively) poor performance can really make an honest contribution when used in a more complex system.

2.3.3 Evaluation Measures

Another thing that you should know is that, in order to evaluate your work and/or allow a straightforward comparison with other works, several measures have been proposed in the literature, some being specific to certain frameworks. While **precision**, **recall** and **F-measure** are used in many applications (sometimes with small variations), other evaluation metrics have specific targets. Thus, **BLEU**, **METEOR** and many others are used to evaluate **machine translation** systems, **ROUGE** is used to evaluate **summarization** systems and **perplexity** is used to evaluate, for instance **N-grams** (N-grams: remember from Prof. Nuno's class?).

Considering the following, we will define precision, recall and F-measure, as used, for instance, in **Information Retrieval**:

- TP = true positives;
- TN = true negatives ;
- FP = false positives (type 1 errors);
- FN = false negatives (type 2 errors).

Precision is defined as the fraction of retrieved instances that are relevant, i.e.,

$$Precision = \frac{TP}{TP + FP}\,. \tag{2.2}$$

Recall is defined as the fraction of relevant instances that are retrieved, i.e.,

$$Recall = \frac{TP}{TP + FN}\,. \tag{2.3}$$

Notice that a maximum precision means no false positives and a maximum recall means no false negatives.

The **F-measure** can be used to combine the above two measures into a single metric, and it is defined as a weighted harmonic mean of precision and recall (usually we set β = 1, and we obrain the famous $\mathbf{F}_1$, which gives the same weight to both precision and recall):

$$F_\beta = \frac{(\beta^2 + 1) \times Precision \times Recall}{\beta^2 \times Precision + Recall}\,, \tag{2.4}$$

Another example of an evaluation metric is the **Mean Reciprocal Rank** (MRR), a specifically tailored evaluation measure for tasks such as QA, as it can be used to evaluate systems that return ranked lists of possible answers for a given question.

For a test set of N questions, the MRR is given by:

$$MRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{rank_i} \,. \tag{2.5}$$

Table 2.2 (taken from [7]) depicts a fictitious run of a system for a small test set of three questions, with $MRR = \frac{1+\frac{1}{3}+\frac{1}{2}}{3} \simeq 0.61$.

Question	**Ranked list of answers**	**Rank**	**RR**
What is the capital of Portugal?	**Lisbon**, Coimbra, Oporto, Viseu	1	1
When was Mozart born?	1884, 1870, **1756**, 2012	3	$1/3$
Who is the 44th president of the USA?	B. Clinton, **B. Obama**, R. Paul, Sampaio	2	$1/2$

Table 2.2: Example of the reciprocal ranks for a test set of questions, with correct answers shown in **bold**.

So, remember that there are many different evaluation metrics, tailored to the task in hand. In some NLP fields, there are evaluation metrics that allow for a good automatic evaluation. In other NLP fields, such as dialogue systems, measures are not very trustable and human evaluations, although subjective and expensive, are still mandatory. If you need to prepare a human evaluation for your master thesis, prepare everything carefully, in advance. It is never a trivial task.

2.3.4 Evaluation fora

In order to allow a straightforward comparison between systems, many evaluation *fora* take place nowadays (please, take a look at CLEF[4], TREC[5] or SEMEVAL[6] pages, for instance). Many different tasks are proposed in these *fora* (QA, Information Retrieval, Plagiarism detection, etc.) and you only need to register your system in the ones related with your work. You are given access to the training data and, at a certain previously scheduled date, the test sets are released (as we will do with in the second mini-project – MP2). Then, you have a limited amount of time (some hours, a week, ...; in the case of the project, some hours) to send the results of your system to the organizers (considering the project: me). After some time, all the results obtained by the competing systems are divulged and everybody can see how good (or bad) his/her system is, evaluated in the same conditions. Cool, isn't it?

***

Thank you for reading this chapter! If you found typos, sentences that you consider unclear, etc., please, let me know. Thanks!

[4] http://www.clef-initiative.eu
[5] http://trec.nist.gov
[6] https://semeval.github.io

Chapter 3

Regular Expressions

I'm sure that you have heard about **Regular Expressions (RE)**. Here, we will use a very informal definition: a RE is just a pattern that describes sequences. You will see how helpful a RE can be in many applications, as we can use them in **Vi** (hum... probably the best editor ever, in the opinion of people with grey hair), in **Emacs** and **BBEdit**; in a **Perl** program (hum... probably the best programming language ever, in the opinion of people with grey hair) and even with a **grep** (we will talk about grep in the first lab). Notice, however, that there are small differences in the implementation of some concepts regarding RE. Do not stress about this, please.

3.1 Learning to use Regular Expressions

Before we start, here are some things that you should not forget:

- Characters inside braces [] specify a disjunction of `characters` to match: [ola]pp means "o", "l" or "a" and will match "opp", "lpp" and "app" and not "olapp";
- RE are case sensitive: use [Bb]ola for "bola" and "Bola";
- Instead of [ABCDEFGHIJKLMNOPQRSTUVXWYZ], you can use [A–Z] and instead of [0123...9] use [0–9];
- RE always match the biggest string.

Now some basic stuff:

1. [] and $^{\wedge}$ can also be used to declare characters that should not appear in the RE. [$^{\wedge}a$] means any character, except a;
2. • means any character, ? means zero or one, * means zero or more (the wild card) and + means one or more (always considering what appears before). For instance:
   (a) a* represents: ϵ (empty string), a, aa, aaa, aaaa, ...
   (b) aa* represents: a, aa, aaa, aaa, ...
   (c) [ab]* represents: ϵ, aaa, abab, bbb, ...

   (d) [0–9][0–9]* represents all the integers
   (e) [0–9]+ also represents all the integers
   (f) ta.*ta matches all the lines in which “ta” occurs twice.

3. ^ indicates the beginning of a line and $ the end of a line
4. | stands for the disjunction
5. () groups tokens
6. {n} means exactly n occurrences of previous element
7. {n,m} means between n and m occurrences of previous element
8. {n,} means at least n occurrences of previous element
9. \d represents any digit
10. \D means any character that is not a digit
11. \w means any alpha-numeric character or a space
12. \W means any character that is not alpha-numeric
13. \s means any space (tab, blank, ...)
14. \S means any character that is not a space
15. *, \., \? represent, respectively, the special characters *, . and ?
16. \n, \t represent a newline and tab

And now, for something completely different, here goes a brilliant cartoon[1], which I hope will inspire your study on RE.

3.2 Some applications based on Regular Expressions

If you have not yet started applying RE you are probably still not convinced of how useful and expressive they can be. Do you remember talking about **ELIZA** [9] in your AI course? No? Well, ELIZA is a computer program that simulates a psychotherapist. The following is an example of an interaction with ELIZA:

```
Eliza: How do you do.  Please tell me your problem.
you: I’m going to start my Natural Language course tomorrow.
Eliza: Your Natural Languages course tomorrow?
you: yes.
Eliza: Please go on.
you: And I’m stressed about it.
Eliza: Do you know anyone else who is stressed about it?
you: Maybe some of my students.
```

[1] Completely stolen from `https://xkcd.com/208/`.

Figure 3.1: The Power of RE

```
Eliza: Your students?
you: yes
Eliza: I see.
```

ELIZA was written by Joseph Weizenbaum (MIT) in the mid 60s and when it appeared, it was taken seriously by many users. For instance, his secretary kept telling him that he should not enter her room when she was “talking” with ELIZA. The funny (or scary) part is that ELIZA was implemented using simple pattern matching techniques.

In the following excerpt, you can see some of the patterns that ELIZA uses[2]. For instance, if I say “i remember the war”, this will match with the first hypotheses: * i remember * (the first wildcard matches “i” and the second wildcard matches “the war”). Then, ELIZA would answer one of the following items, in which the (2) is replaced by “the war” ((1) is used to keep the tokens that match the first wildcard and (2) the ones that match the second wildcard). For instance, ELIZA could answer “Why do you remember the war just now ?".

```
 decomp: * i remember *
    reasmb: Do you often think of (2) ?
    reasmb: Does thinking of (2) bring anything else to mind ?
    reasmb: What else do you recollect ?
    reasmb: Why do you remember (2) just now ?
    reasmb: What in the present situation reminds you of (2) ?
    reasmb: What is the connection between me and (2) ?
    reasmb: What else does (2) remind you of ?
```

[2]Taken from `http://cpansearch.perl.org/src/JNOLAN/Chatbot-Eliza-1.04/doctor.txt`. Don’t need to check. It is just mentioned for credits.

```
decomp: * do you remember *
  reasmb: Did you think I would forget (2) ?
  reasmb: Why do you think I should recall (2) now ?
  reasmb: What about (2) ?
  reasmb: goto what
  reasmb: You mentioned (2) ?
decomp: * you remember *
  reasmb: How could I forget (2) ?
  reasmb: What about (2) should I remember ?
  reasmb: goto you
```

Notice however that some implementations are “intelligent” enough to make some extra processing, and if I say “I remember my cat", instead of answering “Why do you remember **my** cat just now?” it answers “Why do you remember **your** cat just now?”.

ELIZA was one of the first NLP applications and the first ever **chatbot** (although nobody was using the term “chatbot” by that time). You can find many implementations of ELIZA on the Internet. Also, if you feel the need to talk with ELIZA everyday, please find a real psychotherapist as soon as you can.

***

Thank you for also reading this document! Again: if you found typos, sentences that you consider unclear, etc., please, let me know. Thanks!

Chapter 4

Word's don't come easy: pre-processing text

In this chapter, we will study some preliminary processing that can be done to text, before moving to the main task. At the basis of this pre-processing is the concept of "word". As we will see, not even its definition is so obvious as it might seems. You will learn what NLP researchers will do to words and, believe me, we are not nice with the poor words: we cut them in little pieces, we analyze the little pieces, we compare the little pieces, we play with their sounds and, sometimes, we simply eliminate them, no mercy. Anyway, nowadays, with deep learning, some of these pre-processing techniques are no longer applied. In fact, there is a widespread assumption that lemmatized input is not needed considering deep learning models. However, there are some works showing that this can be true for some languages, but not for others. As usual: there is no 100% in NLP.

4.1 Tokenization

Tokenization (or **word segmentation**) is the (not so trivial) process of identifying **tokens** in a text. This process will depend on what you call "a word". Therefore, the first thing to do is to decide which sequences of words should be considered as a single one. Then, punctuation is on your way and you will have to deal with that... Do you think you can properly handle tokenization? Well... maybe you can, if you consider the Portuguese language. However, remember that many other languages exist and that some of these languages have specificities that can turn tokenization into a really challenge (not to say nightmare). In the following you will understand what I mean.

4.1.1 Compounds

It might be important to treat sequences of words, like *Viana do Castelo* (for Erasmus students: a beautiful city in the north of Portugal), as single term. You might be tempted to think that only proper nouns (such as *Joana Martins*, *Altered Carbon* or *Instituto Superior Técnico*) or some other sort of Named Entities should be grouped as a single term, and that these are easy to identify (mistake!). You are wrong: more than

proper names and/or Named Entities can be included in this process. For instance, what about *chapéu de chuva* (umbrella; something like "hat for rain") that is no longer written with the hyphens?

Depending on your application, it can be useful to group some sequences of words that linguists would not even call Compounds (Multiword Expressions). In some cases, you can build a lexicon containing these words, which will help your task. However, even in restricted domains some automatization is (probably) needed to generate the different/alternative forms of the words in the compounds. Nevertheless, as some of the words in these sequences can be expanded and others can't the task of building patterns to automatically generate all the associated forms is also complicated. As an example, we can say *chapéus de chuva* (umbrellas), but not **chapéus de chuvas* (something like "hats for the rains").

So, in conclusion, depending on the application, you might decide to group multiword expressions as a single unit of meaning or not. It might be really helpful to group them before moving to a posterior analysis, but, sometimes, it can be complex to do it.

4.1.2 Punctuation

And what about punctuation? Punctuation can bring several problems to the tokenization process. If question marks, for instance, do not cause many troubles in word identification, a period might be a problem. Consider, for instance, *Sr.*, *55.5* and *www.google.com*, *Rock ´n' roll*, *Toys'r us*, *U.S.A*. These terms contain periods and we have to inform the tokenizer that their dots do not represent the end of a sentence. Notice that if we can (more or less) easily build a list with a set of fixed words such as *Sr.*, but to be able to capture numbers or URLs you will probably need to use our beloved regular expressions. Some of these lists are language dependent. You can find list for several languages in the internet. Just check them before using them.

4.1.3 Other scenarios

You know English, maybe French and Spanish, but even if you know German, you can't say that you have the "whole picture". This is because there are languages with some particularities that never crossed our minds and that can transform tokenization (and other NLP tasks) in a very complicated process. A good example is Chinese, as their sentences are sequences of characters without spaces between. For instance, if this also happened in English, do you think *nowherefast* would mean *nowhere fast* or *now here fast*?[1]. Just to give an idea of how challenging this can be, there are competitions where systems try to find the best way to tokenize Chinese. Apparently, a simple algorithm that analyzes the input left-to-right and consults a dictionary trying to find the longest meaningful sequences of characters can bring good results.

A synonym for complicated tokenization is "*agglutinative languages*" (such as *Turkish*). In these languages, words are formed by joining morphemes together, (usually) without fusing them. Almost every language have some words formed in this way, but

[1] Best music ever (and probably worse movie ever, too)! `https://www.youtube.com/watch?v=Jh1ntyXM304`

for some this is the rule, not the exception. After some examples (in the class), you will understand that Portuguese is so... simple. :-).

To conclude this section, I invite Erasmus students to send me examples of complicated words in their own languages, if these are agglutinative languages. Thank you in advance!

4.2 Word's manipulation

After having identified our words in a text, there are many things we can do to them that will help us in further processing.

4.2.1 Elimination

Some words are not very informative for the task at hand, and their presence only represents more rules and/or more processing. Thus, the first thing we can do is to get rid of them. A good example is the sequence *é que* in some Portuguese questions, such as *Onde é que fica o bar?* (where is the bar?). In these sentences the *é que* can be eliminated without causing any interpretation problem: *Onde fica o bar?* (where is the bar). In what seems to be almost 200 years of NLP the only 100% rule I found was this one: we can get rid of the *é que*, and the meaning of the sentence will not change. Once a close friend from linguistics started to give me a counterexample, but I ignored him.

In many applications people get rid of what we call *stop words*, which usually include *functional words* (prepositions, articles, etc.). Although stop words are language dependent, it is not very difficult to find lists of stop words for different languages in the web. In many cases, to get rid of stop words is a good decision, but there are scenarios, such as authorship identification, where these words can be extremely important (I am sure that you know somebody that says "portanto" ("so") or "tipo" (some sort of "a kind of") all the time). Also, some lists contain pronouns like "Where", "When", 'Who", etc. Getting rid of those in a task related with QA is an **NLP-suicide**. So: never blindly apply a list of stop words found in the internet to your corpus.

4.2.2 Lowercasing

Another thing that can be very useful is to lowercase your data. This would allow *Comi* (I ate) and *comi* from the sentence *Comi sopa e comi muito bem* (something like: I ate soup and I ate very well; in Portuguese the subject can be omitted) to be treated as the same word. However, as (almost) nothing rules for 100% in NL, this can be a very bad idea in some applications. For instance, if you lowercase your text and you want to translate *us*, you don't know if you should translate the proper noun *US* or the pronoun *us*. The same for the words *figo* and *Figo* (well... you are each day younger, so I'm not so sure if you still know who is this national hero... just check Google).

Note, nevertheless, that in order to avoid **data sparseness**, it still can be important to lowercase your data.

Figure 4.1: Perfect answer (taken from Reddit).

4.2.3 Normalization

Lowercasing can be seen as a normalization process, but many more normalization processes exist. For instance, you may want to normalize dates (*April, 2009* vs. *April, 09*) vs. *04/09* vs. *04-09*, numbers (*0.34* vs. *0,34* or *2000 m* vs. *2 Km*) and names (*John Fitzgerald Kennedy* vs. *John F. Kennedy* vs. *John Kennedy*). If you think about a QA system that uses the most frequent answer as the correct one, it seems obvious the need to perform this normalization step.

4.2.4 Stemming and lemmatization

Stemming and lemmatization, although representing different concepts as Prof. Nuno deeply explained, are used as synonyms in many situations.

Both processes are very useful in many applications, as they not only help with the data sparseness problem, but they also allows the connection of words that, otherwise, would be left unrelated. For instance, if you type *assassino D. Carlos* (D. Carlos' murder) in a search engine, a possible result in the sentence *D. Carlos foi assassinado por Buiça* (D. Carlos was murdered by Buiça) can be found due to the stemming of the words *assassino* (murder) and *assassinado* (murdered). At the same time, if you are searching for a *table* in Google, either if you type *table* or *tables* you will probably get similar results, but (and considering the perfect example presented by Jurafsky in his courses) if you are searching for a new *window*, if you write *window* or *windows* you will not obtain similar results and, in the last case, you will (probably) never find a new window.

A very well known stemmer is the **Porter Stemmer**, developed in 1979 (uau!). It is rule-based and language dependent and might transform *organization* in *organ* (ahahah!), but has been used very successfully in many applications. If you have some time, give it a look (you can easily find many implementations in the web). Also, the **Snowball stemmer** is widely used by the NLP community.

***

Thank you for reading another Sebenta's chapter! As usual, if you found typos, sentences that you consider unclear, etc., please, let me know. Thanks! By the way, "Words don't come easy" is also a song from the 80s, but I'm not adding the link. Check it on your own responsibility.

Chapter 5

N-grams

Until now we have not make use of any linguistic information and in this chapter... we won't make it either. We will continue our NLP voyage just by counting stuff, and you will see the amazing things we can do with it.

5.1 N-grams estimation and Markov assumption

Although **N-grams** can be sequences of anything (characters, words, lemmas, etc.), in this section we will see N-grams as sequences of (N) words. Therefore, **bigrams** are sequences of two words, **trigrams** sequences of three words and so on (from now on, to impress your friends and family, start saying **unigram** instead of "word"). They can be very useful in several applications in NLP. In this section we will focus on **word prediction** and in calculating a **sentence probability**.

5.1.1 Word prediction

Can you predict the next word of the following sequences?

1. Once upon a...
2. O assassino é o...
3. Vai...
4. Bom...
5. And now for something...

What about this?

1. Once upon a time.
2. O assassino é o mordomo.
3. Vai chover.
4. Bombóca.

5. And now for something completely different.

Now imagine that you want to implement an application that predicts words. Word predictors can be done with N-grams. Let $W_1...W_{N-1}$ be a sequence of words (as a notational alternative we will use W_1^{N-1}). If you want to predict the next word, W_N, you should try to find the word with the highest probability of following the given sequence. Therefore, our goal is to compute the probability of a word W (= W_N), given some history H (= $W_1...W_{N-1}$). That is, we want to calculate P(W | H)[1]. Let us see how to do this.

As a first hypothesis, let us suppose we have at our disposal a gigantic corpus with all the sequences in Portuguese (for instance). Let us count all the observations of the sequence HW as well as all the occurrences of the sequence H alone. In this way we can calculate P(W | H) (notice that the sum of all the counts of sequences that start with H is the same as the counts of all the H):

$$P(W|H) = \frac{count(HW)}{\Sigma_K count(HK)} = \frac{count(HW)}{count(H)}. \tag{5.1}$$

As an example, consider that we have:

- H = *Once upon a*
- W = *time*.

In order to calculate P(*time* | *Once upon a*), we have to divide the number of occurrences of *Once upon a time*, by *Once upon a*.

The problem with this formula is the assumption that "we have at our disposal a gigantic corpus with all the sequences in Portuguese". Such thing is like Santa Claus: it simply does not exists[2]. Not even Google has a corpus with all the possible sequences in any language (you certainly have already asked Google for a certain sequence, between commas, and no result was found). Ya, sometimes not even the web is big enough to find a certain sequence of words. So, let us envisage another way of calculating P(W | H). As you will see, it is not perfect, and, due to that, it perfectly illustrates something that I want you to learn from this course (if you haven't learnt it yet). In many scientific fields, models are used in order to explain a certain phenomena. Here, in NLP, we use many models that we know that are not perfect to explain a certain phenomenon, but sometimes are the only **models that respect the commitment between explaining (most of) the phenomenon in hand and being computationally tractable**. That is, sometimes we have to adapt ourselves to what we have at our disposal and live with that (notice that this is valid in science and also in general life).

[1] Remember that:

- **Probability function**: P(A) denotes the probability of occurrence of an event A ($0 \leq$ P(A) ≤ 1);
- **Conditioned probability**: P(A|B) denotes the probability of A, being given B.

[2] Please, tell me you knew this.

Now is time to remember what a **Markov model** is and what states the **Markov assumption**. Markov models represent a class of probabilistic models that assume that we can predict the probability of some future event without having to look to far in the past; the Markov assumption states that the probability of a word depends only on the immediately previous ones. Thus, in order to compute the probability of a word given an history, P(W | H), we will **approximate** (as I told you) the history by using only the last words. And if we consider the last word, we will be talking about bigrams, if we consider the two previous words, trigrams, and so on. That is:

- bigram: P(W | H) = P(W_N | W_1...W_{N-1}) ≅ P(W_N | W_{N-1})
- trigram: P(W | H) = P(W_N | W_1...W_{N-1}) ≅ P(W_N | W_{N-2} W_{N-1})
- ...

Exercise 4: Using bigrams

Consider the following corpus, where <s> represents the beginning of the sentence and </s> its end (we assume no normalisation):
<s>Eu adoro a Maria</s>
<s>A Maria eu adoro</s>
<s>Eu adoro bolachas Maria</s>
If we consider bigrams, we have that:
P(Eu | <s>) = 2/3
P(A | <s>) = 1/3
P(adoro | Eu) = 1
...

5.1.2 Calculating a sentence probability

So, we have learnt to estimate the probability of occurrence of a word after a given sequence. And if now we want to know how probable is a sentence? Again, as a first hypothesis, we could calculate the probability of sentence W_1^N with Equation 5.2 and by using the **chain rule of probability**:

$$P(W_1^N) = P(W_1) * P(W_2|W_1) * ... * P(W_N|W_1^{N-1}) = \Pi_{k=1}^{N} P(W_K|W_1^{K-1}) \quad (5.2)$$

Nevertheless, we have here the same problem that we had before: we probably won't be able to calculate stuff like $P(W_N|W_1^{N-1})$, because $count(W_1^N)$ or $count(W_1^{N-1})$ (for instance) never occurred. Once again we will use the Markov assumption and, once again, we can use bigrams, trigrams, etc. to produce the following estimations:

- bigram: $P(W_1^N) \cong \Pi_{k=1}^{N} P(W_K|W_{K-1})$
- trigram: $P(W_1^N) \cong \Pi_{k=1}^{N} P(W_K|W_{K-2}W_{K-1})$
- ...

Exercise 5: Most probable sentence in Portuguese
Consider unigrams. Which is the most probable sentence in Portuguese, with 4 words.

Exercise 6: Chinese restaurant
Make (with lots of enthusiasm) the exercise your professor will provide you.

5.2 So, how far can we go with N-grams?

N-grams are certainly extremely helpful in many NLP areas, such as Statistical Machine Translation (SMT), Spell Correction or Speech Recognition. In SMT, for instance, two types of models are used:

- a **Translation Model** that is responsible for the **semantics** of the translation;
- a **Language Model**, which takes care of the **fluency** of translations.

In SMT (and in many other applications) N-grams can be used to build the Language Model. As an example, consider that you receive the following sentences as translation candidates of the sentence *Estou cansado*:

- *I'm exhausted.*
- *Tired me.*
- *I like superman.*

The Translation Model should give more points to the second sentence which is the one that better captures **the meaning** of the original sentence; the Language Model will score higher the first and third sentences as these are more fluent in English. An accurate combination of both models is essential to reach a good translation engine.

However, there are some drawbacks in this. In the following we will study a measure that allows you to make a preliminary evaluation of how N-grams will fit in your data, and we will briefly talk about the major problems when using N-grams.

5.2.1 Evaluating N-grams: perplexity

Perplexity is an evaluation metric for N-grams. The idea is the following: you calculate two probabilistic models (for instance, bigrams and trigrams) and then you use perplexity to see which one is the better model, that is, which one has a tighter fit to a corpus or, in other words, which one will make the corpus more probable. If a model is good it will give high probability to probable sentences and a low one to "aberrant" ones.

So imagine that you have calculated the N-grams of your model (bigrams, trigrams, etc.). Then, considering that your corpus is $W = W_1^N$, perplexity is given by Equation 5.3:

$$PP(W) = \sqrt[N]{1/P(W_1^N)} \tag{5.3}$$

Remember that the calculus of $P(W_1^N)$ is dependent of the language model that you are using (based in bigrams, trigrams, etc.). The lowest the perplexity, the best the model is (being less perplexed is better).

Some years ago, several of your colleagues did a project based on this concept. The idea was to compare several Portuguese writers in what respects the N-grams they use in their work. Although it is widely accepted that perplexity is not the best measure to compare literature, some interesting results were attained.

5.2.2 Main problems

One of the major drawbacks with N-grams is that they don't deal with **long distance dependencies**. For instance, the sequence *Gollum loves his precious* could be a frequent 4-gram, but if only sentences such as *Gollum loves in a very sick way his precious* are found in the text, that 4-gram will never be captured.

On the other hand, N-grams, as you can imagine are very dependent of the corpus where you did your training. Typically, best results are obtained by increasing the values of N, although higher values of N increase the number of possible N-grams and the resulting tables are each time more sparse (the **data sparseness** problem is common in many NLP applications).

Also, you have probably noticed that if an N-gram does not exists because it was never found in the training corpus, it will cause problems if it is needed during test, because the associated probability will be 0. A similar problem arises if an N-gram is extremely unfrequent. There are many techniques in the literature – the **smoothing techniques** – that deal with this problem. In this section we will talk very briefly about a few of them. However, you should understand the main problem: you steal some probability mass from observable things to give them to non observable events (yes, like Robin Hood did); thus, you have to recalculate the whole stuff again, because we are talking about probabilities (their values should be between 0 and 1).

Let us start with the **Laplace Smoothing** or **Add-one Smoothing**. The value one is added to all frequency counts, that is, we pretend that we saw each word one more time than we actually did. That is, being V the vocabulary size (number of the set of tokens), we have, for bigrams:

$$P_{Laplace}(W_N|W_{N-1}) = \frac{count(W_{N-1}W_N) + 1}{V + count(W_{N-1})} \tag{5.4}$$

Exercise 7: Add-one Smoothing

See what happens when you use Add-one Smoothing in the exercise your teacher will propose you.

Another technique is the **Good-Turing discounting** [**?**] where the counts of what was observed once is used to estimate the counts of what was never seen (other techniques implement this idea).

Exercise 8: Good-Turing discounting

Check with your professor the probability of having *lulas* for lunch tomorrow.

We can also count with **Interpolation**, which is a technique that combines N-grams from different orders to estimate the probability of an N-gram with 0 frequency, as stated in Equation 5.5.

$$\hat{P}(W_N|W_{N-2}W_{N-1}) = \lambda_1 P(W_N|W_{N-2}W_{N-1}) + \lambda_2 P(W_N|W_{N-1}) + \lambda_3 P(W_N) \quad (5.5)$$

In this cases, the training corpus is used to calculate the N-grams and a development set should be used to calculate each λ_i.

Finally, there are also the so called **backoff** techniques where you move to lower order N-grams to estimate the probability of an N-gram with 0 frequency.

Exercise 9: Smoothing

Check Jurafsky on-line courses about smoothing. They are very interesting. In `https://class.coursera.org/nlp/lecture` you can find a detailed explanation about some of the techniques previously presented. In fact, the *lulas* examples is based on a Jurafsky example presented in that lecture.

Chapter 6

More Interesting Measures

Similarity measures (and edit distances) allow us to calculate how close (or far) two objects are. They can be very useful in NLP, as they can be used to compare words, compare sentences or even to compare texts. Some metrics tell you how close two words (or sentences) are (the closer they are, the higher the value), others tell you how far two words (or sentences) are (the closer they are, the smallest the value). In this section we will study them.

6.1 Comparing words and sentences

There are many applications where we have to compare words, as for instance to correct an unknown word. You can compare words taking into consideration their orthographic form (*Monserate* vs. *Monserrate*), but you can also check the way they sound (*Monserrate* vs. *Munçerráte*). In this section we will learn about some measures that allow us to compare words. We will see that these measures can also be adapted to compare sentences and, without using any linguistic information, you will learn how to build another type of conversational agent: the retrieval-based ones.

6.1.1 Minimum Edit Distance

Algorithm 1 (next page) calculates the **Minimum Edit Distance (MED)** between two words, that is, the minimum number of transformations (**insert**, **replace** and **delete**) that need to be done in order to transform one of the words into the other. It builds a matrix and fills its cells with the cost associated with the needed transformations, taking into consideration the substrings of the words being processed (yes, **dynamic programming**). The cost associated with each one of the three possible transformations is given by:

- C_1 (delete in the source),
- C_2 (insert in the source), and
- C_3 (replace in the source).

I read somewhere that when $C_3 = 2$, we have the widely known **Levenshtein distance** [6], but many people use MED and "Levenshtein distance" interchangeably.

We will do the same, and, unless some explicit instruction informing otherwise is given, we will assume $C_1 = C_2 = C_3 = 1$.

Algorithm 1 MED between strings s (size n) and t (size m)

```
C_1, C_2, C_3 ← 1
if n = 0 then
  return m
end if
if m = 0 then
  return n
else
  Build matrix M, with m+1 lines and n+1 columns
  j ← 1
  while j ≠ n + 1 do
    i ← 1
    while i ≠ m + 1 do
      if s[i] = t[j] then
        M[i, j] = M[i − 1, j − 1] // Take the diagonal value if characters match
      else
        M[i, j] = min(M[i − 1, j] + C_1, M[i, j − 1] + C_2, M[i − 1, j − 1] + C_3)
      end if
      i ← i + 1
    end while
    j ← j + 1
  end while
  return M[m, n]
end if
```

In the class (lucky you!) we will learn to precisely apply this algorithm (although I suspect that you probably already know it from other course). For the moment, I am happy if you are able to understand that the MED between...

- *Lar* and *Mar* is 1, and corresponds to 1 action of the form “replace” (L -> M)
- *Their* and *There* is 2, and corresponds to 2 actions of the form “replace” (i -> r and r -> e)
- *do* and *undo* is 2, and corresponds to 2 actions of the form“insert” (-> u and -> n)
- *cats* and *cat* is 1, and corresponds to 1 action of the form “delete” (s ->)
- *Monserrate* and *Monçerat* is 3, and corresponds to 1 action of the form “replace” (s -> ç), 2 actions of the form “deletes” (r -> , e ->)

Notice that measures can be customised to different applications. For instance, they can be adapted to take into consideration the device in use: if we are using a QWERTY keyboard it is more probable to replace a “w” by a “q” than a “w” by a “l”. By the same token, if you are using an old mobile (oh, I miss them) it can be easy to replace an “a” by a “b” (confirm in Figure 6.1).

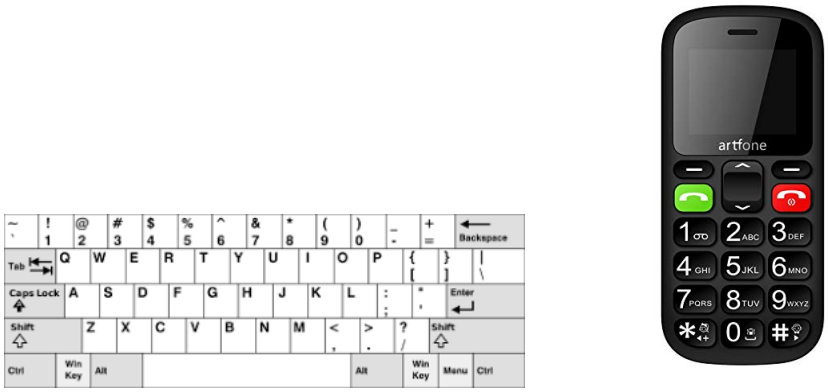

Figure 6.1: Keyboards

MED can also be used to calculate the distance between sentences (and not only words). For instance, consider an application that knows how to answer to *Onde foi assassinado D. Carlos?*. If the user states a new question as *Onde é que foi assassinado D. Carlos?*, it "only" has a MED of 2 from the original sentence. Thus, edit distances can be applied to this scenario as if a given sentence has a small difference from some sentence that the system knowns how to answer, maybe it can be considered similar to it and answered in the same way. Nevertheless, it might be necessary to use measures that do not take the order of words into consideration in such a strong way as MED does, so that *D. Carlos foi assassinado onde?* would also be considered similar to *Onde foi assassinado D. Carlos?*. Although the order of words in a sentence can be very important (*Quem é filho de XPTO?* vs. *XPTO é filho de quem?* or *anos 80* (80s) vs. *80 anos* (80 years old), a measure that do not penalise so strongly some reordering could be welcome in this scenario. So, if the MED takes the order of the characters into consideration, the measures from next section don't, as they treat sequences of characters/words as **sets** or **bags**.

6.1.2 Jaccard and Dice

The idea of dealing with sequences of characters/words as **sets** or **bags**, lead us to the concept of **bag of words**, a model widely used in NLP and Information Retrieval, in which sentences (and documents) are seen as a bag (or, sometimes a set) of words. Grammar is ignored, as well as word order. However, word multiplicity is kept.

Jaccard (Equation 6.1) obtains higher scores for strings that have similar length (a zero value means that there is nothing in common between two words; one is the highest possible value).

$$Jaccard(s,t) = \frac{|s \cap t|}{|s \cup t|} \tag{6.1}$$

A different philosophy is used in the Dice measure, in which strings with different lengths are not so strongly penalized:

$$Dice(s,t) = 2 \times \frac{|s \cap t|}{|s| + |t|} \tag{6.2}$$

From now on, let us consider the following: if we opt for a bag-approach, if a letter occurs twice in a string, the two occurrences will be considered; if we opt for the set-approach, **as a set (by definition) does not have repeated elements**, only one occurrence will be considered.

Let us do some exercises. Consider the words s=*Saturday*, t=*Sunday* and w=*day*. Considering the ***set-approach***, check that:

- Jaccard(s, t) = 5/8
- Jaccard(s, w) = 3/7
- Dice(s, t) = 10/13
- Dice(s, w) = 6/10

Many more distances exist (really, there are tons of them, just check Google), tailored for different applications. Also, remember that all these measures can be applied to sentences: instead of comparing character you compare words, and with this idea, you can create a conversational agent. For instance, Edgar Smith (Figure 6.2) and Filaipe, are examples of conversational agent based on these type of measures. They have a database with questions for which they know the answer and when the user poses a question they will check for the "closest question" in their knowledge base; if they find a question that is "close" enough, they will return the answer. Edgar's knowledge base is the specific domain of Monserrate, Sintra; Filaipe has subtitles in its knowledge base. These type of agents are called **retrieval-based conversational agents**.

Figure 6.2: Edgar Smith in the Monserrate palace, answering questions

6.1.3 The sound of Language

Besides comparing words by its characters, they can also be compared by the way they sound. **Soundex** (Russell, 1922) is a phonetic algorithm that allows to compare words by sound. It was written to be used for words pronounced in English. Its earlier applications targeted names' indexing. For instance, consider the word *Jurafsky*. Imagine that his name is coded somehow in order to capture the way it sounds. If you are trying to find his book but you don't know how to spell his name, if you manage to write something that is also mapped into the same key (representing the way it sounds), then you may find it.

The following[1] tries to capture the essence of Soundex:

1. Retain the first letter of the name;
2. Drop all occurrences of a, e, i, o, u, y, h, w (unless they appear in the first position).
3. Replace consonants by digits, as follows (after the first letter):
   (a) b, f, p, v $\rightarrow$ 1
   (b) c, g, j, k, q, s, x, z $\rightarrow$ 2
   (c) d, t $\rightarrow$ 3
   (d) l $\rightarrow$ 4
   (e) m, n $\rightarrow$ 5
   (f) r $\rightarrow$ 6
4. Two adjacent letters with the same number are coded as a single number (ex: 55 $\rightarrow$ 5)
5. Continue until you have one letter and three numbers. If you run out of numbers, add zeros until there are three numbers (ex: L2 $\rightarrow$ L200); if you have too much numbers drop them after the third one (ex: L2345 $\rightarrow$ L234).

Exercise 10: Soundex
Find your name in soundex-form. Mine is L200 C600. You can call me that from now on.

6.2 Weighting words

Another measure that should be mentioned, is the widely known **term frequency–inverse document frequency (tf–idf)**. This measure calculates how important a word is in a document from a document collection, and is used, for instance, in Edgar to give weights to the words in the sentences we are comparing (by using the "traditional" measures). The motivation for this is that there are words occurring in each

[1] Adapted from `http://en.wikipedia.org/wiki/Soundex`.

sentence that are also very frequent in many other sentences and that, thus, should not contribute to the comparison process with the same weight as words that are more specific to some questions.

There are many ways to calculate $tf(t, d)$. The simplest one is by counting the number of occurrences of term t in the document d. The formula used to calculate idf is given by Equation 6.3 and the one used to calculate $tf - idf$ is given by Equation 6.4 (being D a set of documents).

$$idf(t, D) = log\frac{|D|}{|\{d \in D : t \in d\}|} \tag{6.3}$$

$$tf - idf(t, d, D) = tf(t, d) \times idf(t, D) \tag{6.4}$$

During the class we will see why $tf - idf$ is so interesting (and, no stress: you do not need to memorize the formula for the exam).

***

Voilà: you finished another Sebenta's chapter! As usual, if you found typos, sentences that you consider unclear, etc., please, let me know. Thanks!