

CONCORDIA UNIVERSITY

COMP6721

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

FALL 2018

Mini-Project 3 Report

Author:

Adrien POUPA
40059458

Professor:

Dr. Leila KOSSEIM

November 30, 2018



Contents

1	Java Implementation of the Mini-Project	2
1.1	Running the Program	2
1.2	The Ngrams Package	3
1.2.1	The Unigram Model	3
1.2.2	The Bigram Model	4
1.3	The Languages Package	5
2	Identifying Languages with the Basic Setup	6
3	Experimenting with Automatic Language Identification	7
3.1	Experimenting with Additive Smoothing	7
3.2	Experimenting with Languages: Spanish, Italian	8
3.3	Experimenting with Training Files	9
	References	10
A	UML Class Diagram	12
B	Expectations of Originality	13

1 Java Implementation of the Mini-Project

In this section, I will be describing my implementation of the automatic language identification software. I have added code comments and a JavaDoc for a better understanding. A UML class diagram is available in figure 5. Code references: [12] [2] [11] [14] [13].

1.1 Running the Program

The "root" package contains the `Main` class, which is the entry point used to execute the program. It deletes the old output files, runs the unigram model and the bigram. Each model is trained, outputs its probabilities and predicts the language of the given sentences. When the detection is complete, it displays a table containing the results. The `Util` class is responsible for handling file creation, writing and deletion as well as counting the characters, explode a string into bigrams and clean a string so that it can be used by the program (ie removing the spaces and the diacritics of the languages). The `CommandLineTable` class is used to display the command line table displayed before the program terminates as shown in figure 1.

```
COMP6721 Mini-project 3
Running the Unigram model...
Running the Bigram model...
Output files generated
```

#	Sentence	Language	Unigram	Bigram
1	What will the Japanese economy be like next year?	English	English ✓	English ✓
2	She asked him if he was a student at this school.	English	English ✓	English ✓
3	I'm OK.	English	English ✓	English ✓
4	Birds build nests.	English	French x	English ✓
5	I hate AI.	English	English ✓	English ✓
6	L'oiseau vole.	French	Spanish x	French ✓
7	Woody Allen parle.	French	English x	English x
8	Est-ce que l'arbitre est la?	French	French ✓	French ✓
9	Cette phrase est en anglais.	French	English x	French ✓
10	J'aime l'IA.	French	French ✓	French ✓

```
Unigram accuracy: 6/10 = 60.000004%
Bigram accuracy: 9/10 = 90.0%
```

Figure 1: Output of the command line for the original 10 sentences

1.2 The Ngrams Package

The **Ngrams** package offers prediction with the Unigram and Bigram models. The **Ngram** interface has the following functions:

1. `train` to train the model;
2. `output` to generate the probabilities files;
3. `predict` to identify the language of the sentences given as input.

Both **Unigram** and **Bigram** classes extend an abstract class, **AbstractNgram**. Its purpose is to map each language to its training file and to define the value of the add-delta smoothing.

1.2.1 The Unigram Model

To identify languages using the Unigram model, the **Unigram** class begins by training for each language defined in the **AbstractNgram** class. First, we clean the training file, meaning that we normalize the diacritics (turning "é" into "e" for example), remove any character that is not a letter, and transform all the letters to lowercase. Then, we count the number of occurrences of each characters, and we store the results into a **HashMap**; that is, each character is mapped to the number of its occurrences. Finally, the probabilities can be calculated from the number of occurrences. For each character, we calculate the following, where c represents any character:

$$P(c) = \frac{\text{numberOfOccurrencesOfCharacter} + 0.5}{\text{totalNumberOfCharacters} + \text{numberOfLetters} * 0.5}$$

- *numberOfOccurrencesOfCharacter* is the number of times the character c is present in the training file;
- 0.5 represents the add-delta smoothing value for the basic setup;
- *totalNumberOfCharacters* is the total number of characters in the training file;
- *numberOfLetters* is the number of unique letters in the training file; for English and French, granted that we use a training file complete enough, that would be 26.

To predict the language of a sentence contained in the `sentences.txt` file, we begin by cleaning the sentence the same way we cleaned the training file, and for each language, for each character, the score is updated by adding the probability of every character using \log_{10} to avoid underflow. The highest score is the most probable language.

If a character in a sentence is not present in the probability `HashMap`, we set *numberOfOccurrencesOfCharacter* to 0.

1.2.2 The Bigram Model

The bigram model is more complex than the unigram model, because we need an additional `HashMap` sublevel. Instead of having just two imbricated `HashMap`s (for each language, associate a `HashMap` that contains the probability for every character), we now have three. For each language (level 1), there is a `HashMap` (level 2) that maps the second character of the bigram to another `HashMap` (level 3) that associates the first character of the bigram and the probability of the bigram. Figure 2 shows how the probability `HashMap` could look like for bigrams "ba" and "ca":

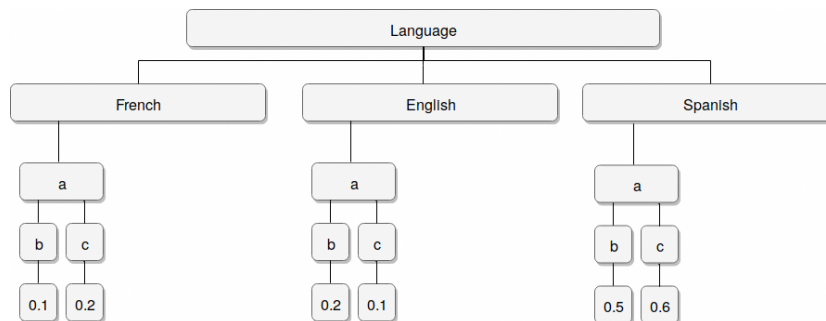


Figure 2: `HashMap`s structure for the bigram model

Otherwise, the mechanisms are similar. First, the bigram models will be trained by collecting the number of occurrences for each bigram that is found in the training file, then we will calculate the probability for each of those bigram. For the bigram "wh", the following formula would apply:

$$P(h|w) = \frac{C(wh) + 0.5}{totalNumberOfBigrams + 0.5 * numberOfDifferentBigrams}$$

- $C(wh)$ is the number of occurrences of "wh" in the training file;
- 0.5 represents the add-delta smoothing value for the basic setup;
- *totalNumberOfBigrams* is the total number of bigrams in the training file;
- *numberOfDifferentBigrams* is the number of unique bigrams in the training file.

We apply the same logic as in the unigram model to predict the language of a sentence by cleaning the sentence and calculating the score by adding the probabilities with \log_{10} to avoid underflow.

Like in the unigram model, if a bigram in a sentence is not present in the probability `HashMap`, we set $C(wh)$ to 0.

1.3 The Languages Package

The `Languages` package is composed of the `Language` interface, that defined the `getCode` function, returning a two letter language code ("EN" for English, "FR" for French for example). This is almost a marker interface [4]. Then, each language class is composed of the definition of this function as well as an overriding of the `toString` function that returns "French", "English", etc.

Overall, this is a very simple package whose sole purpose is to be able to define a `Language` type that can be used genetically in the data-structures of the project.

2 Identifying Languages with the Basic Setup

For the basic setup, I chose to work with Spanish as the third language, since it has the same Latin roots as French. I used the full text [5] of the novel "Don Quixote" by Miguel de Cervantes Saavedra. The add-delta smoothing value was set to 0.5.

The results for the ten original sentences can be seen in figure 1: for those, the unigram model has an accuracy of 60% whereas the bigram model is more efficient with an accuracy of 90%. The only sentence that the bigram labelled incorrectly was "Woody Allen parle", that it identifies as English whereas it is French, but this is understandable since nine bigrams out of fourteen (the filmmaker's name) are coming from the English language. Table 1 shows the ngram results for a sentence that was incorrectly classified by the unigram model. The score results for the unigram model are too close to conclude, with a difference of only 0.28 between the maximum score and the minimum score whereas the bigram shows a larger difference and is able to conclude.

Language	Unigram Score	Bigram Score
French	-19.12	-37.28
English	-19.40	-35.80
Spanish	-19.31	-41.68

Table 1: Ngram model results for the sentence "Birds build nests".

Then, I found 10 sentences that are correctly classified by both the unigram and the bigram, 10 sentences that are incorrectly classified by the unigram, and 10 sentences that are incorrectly classified by the bigram. To find such sentences, I tried to include false friends and words shared by two languages that can be detected. During my research, I realized that it was easier to fool the unigram than the bigram; this is not surprising since the bigram is more "informed" as it is related to the arrangement of characters unlike the unigram. I noticed that both models are more accurate when the sentence is longer; this is normal because a longer sentence means more characters or more bigrams, thus a more relevant score.

Using the basic setup, on the 40 sentences tested, the unigram accuracy was $\frac{19}{40} = 47.5\%$ and the bigram accuracy was $\frac{28}{40} = 70.0\%$.

3 Experimenting with Automatic Language Identification

I experimented by modifying the add-delta smoothing value, adding two additional languages and changing the training files.

3.1 Experimenting with Additive Smoothing

Using the 40 sentences that I found for the basic setup, I changed the value of delta in the additive smoothing. Figure 3 shows the influence of the delta value for the accuracy of both the unigram and the bigram models.

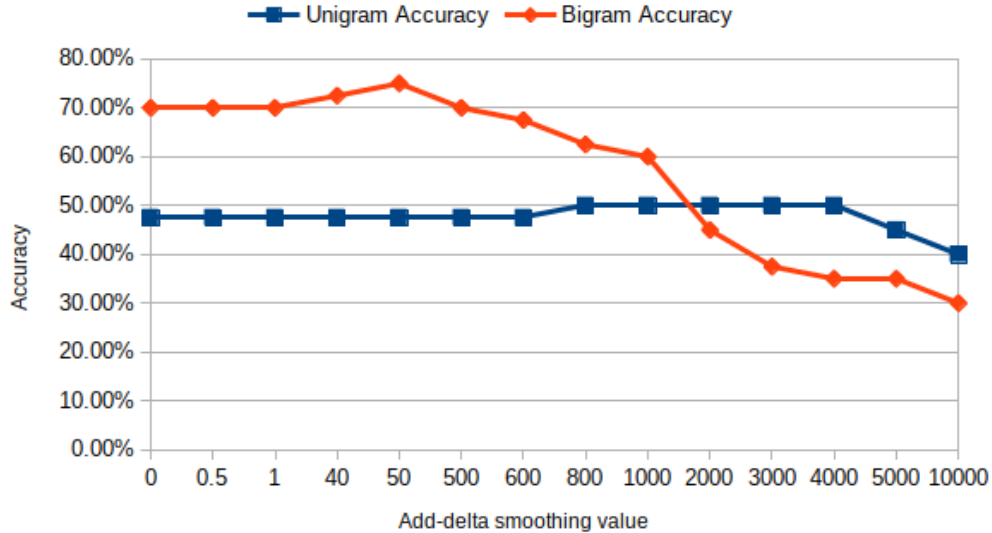


Figure 3: Ngrams accuracy depending on the additive smoothing delta value

From these measures, it appears that the delta value has a bigger influence on the bigram model than on the unigram model. The accuracy peaks at 75% with a delta of 50 for the bigram model before falling down. Overall, the additive smoothing is not really important to improve the accuracy. It would be interesting to compare those results with other methods of smoothing, such as Collection smoothing or Jelinek-Mercer smoothing [15].

3.2 Experimenting with Languages: Spanish, Italian

In addition to the Spanish that was delivered as part of the basic setup that required a third language, I wanted to experiment with another romance language, Italian. To train it, I used the novel "La Seconda e Terza Guerra Punica" by Antonio Ceruti [6].

Then, I added 10 sentences in Italian to test the accuracies of both ngram models. Figure 4 shows the sentences used to test the models and the accuracy of the models. For this experiment, all languages were available.

#	Sentence	Language	Unigram	Bigram
1	Buon giorno.	Italian	Italian ✓	Italian ✓
2	Mi dispiace, ma non parlo bene l'italiano.	Italian	Italian ✓	Italian ✓
3	È stato un piacere conoscerla.	Italian	Italian ✓	Spanish ✗
4	Può parlare lentamente?	Italian	French ✗	Italian ✓
5	Capisco benissimo.	Italian	Italian ✓	Italian ✓
6	Mi innervosisco sempre quando parlo in italiano.	Italian	Italian ✓	Italian ✓
7	La capisco benissimo.	Italian	Italian ✓	Italian ✓
8	Posso pagare con la carta?	Italian	Italian ✓	Spanish ✗
9	Dove sono i camerini?	Italian	Italian ✓	Italian ✓
10	C'è uno sconto?	Italian	Spanish ✗	Spanish ✗

Unigram accuracy: 8/10 = 80.0%
Bigram accuracy: 7/10 = 70.0%

Figure 4: Testing the ngram models on 10 Italian sentences

We can see that for Italian, the unigram model performs better than the bigram model. For those 10 sentences, it is interesting to see that English, the only non romance language that our system can detect, is never wrongly identified. Spanish and French, two romance languages, are.

For the 40 sentences from the basic setup, adding Italian to the list of available languages drops the accuracies to $\frac{17}{40} = 42.5\%$ for the unigram model and $\frac{27}{40} = 67.5\%$ for the bigram model. That may happen because the training file is not diverse enough [3]. Out of the 7 sentences that the unigram model mislabels, only two are in English ("Arcade games are incredible" and "I see a mirage.") and they contain words very close to Italian: "incredible" is "incredibile" in Italian, and "mirage" is "miraggio". Others are in French and Spanish. Thus, we can conclude that the common roots of the romance language clearly appears in ngram language detection.

3.3 Experimenting with Training Files

My last experiment consisted of changing the training files. Instead of taking excerpts of books for English and French, I decided to use the more text and complete books, like I did for Spanish.

For English, the original training file contained 1,250,744 characters. The new training file consists of the books "Moby Dick; Or, The Whale" by Herman Melville [7], "The Life and Adventures of Robinson Crusoe" by Daniel Defoe [8] and "War and Peace" by Leo Tolstoy [10]. The new training file contains 5,291,359 characters.

For French, the training file originally had 885,336 characters. I enlarged it by using "Vingt mille Lieues Sous Les Mers" by Jules Verne [9], "Le tour du monde en quatre-vingts jours" by Jules Verne, "L'île mystérieuse" by Jules Verne, the three volumes of "War and Peace" by by Leo Tolstoy [10] and "A la recherche du temps perdu" by Marcel Proust [1]. The new training file contains 5,676,768 characters, this is somewhat similar to the new English training file.

Whereas I expected the accuracy of the models to improve with more training data, it remained the same with the unigram accuracy being $\frac{19}{40} = 47.5\%$ and the bigram accuracy $\frac{28}{40} = 70.0\%$.

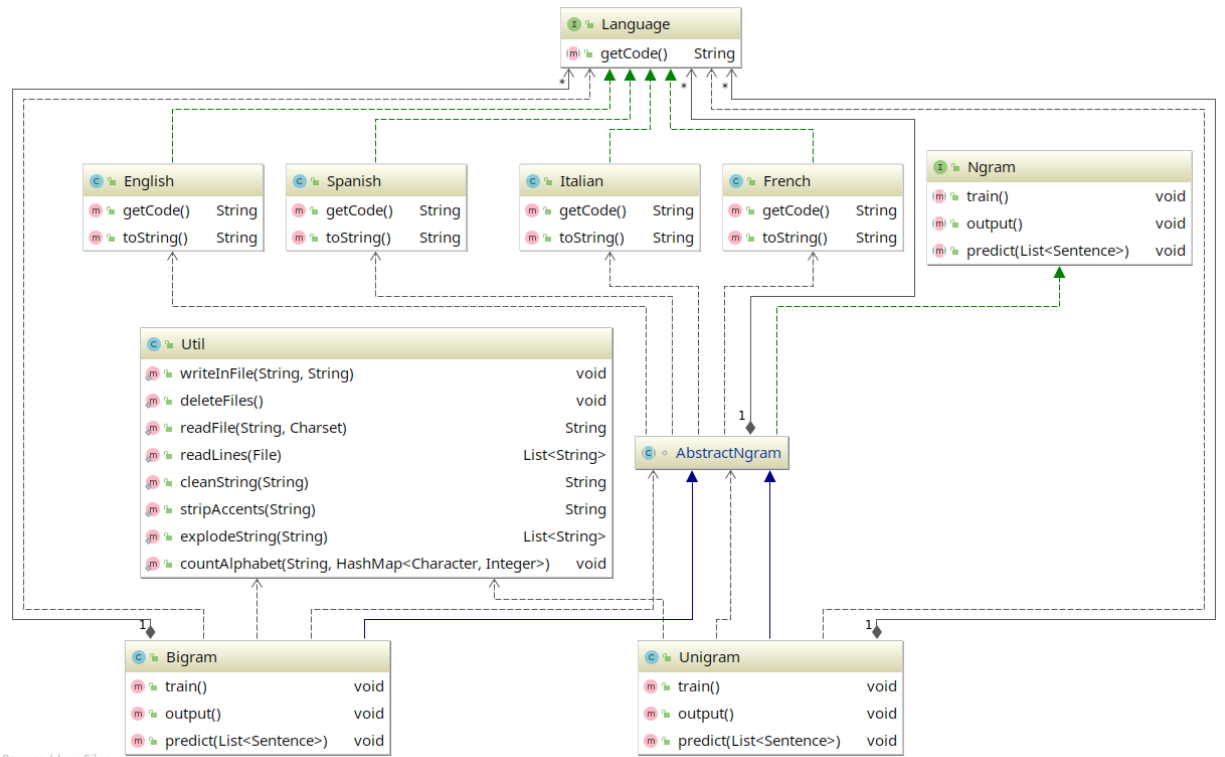
Thus, I conclude that ngram models have a training threshold after which it reaches its peak and can no longer be improved by feeding it additional training material, and the training files given with the mini-project handout were already sufficient.

References

- [1] Archive.org. *Vingt mille Lieues Sous Les Mers — Complete by Jules Verne*. 2018. URL: https://archive.org/stream/larecherchedut12prou/larecherchedut12prou_djvu.txt (visited on 11/28/2018).
- [2] erickson. *Java Command Line - How to create table in a Command line application in Java?* 2018. URL: <https://stackoverflow.com/a/326440> (visited on 11/28/2018).
- [3] errantlinguist. *When are uni-grams more suitable than bi-grams (or higher N-grams)?* 2018. URL: <https://stackoverflow.com/a/36584612> (visited on 11/28/2018).
- [4] GeeksForGeeks. *Marker interface in Java*. 2018. URL: <https://www.geeksforgeeks.org/marker-interface-java/> (visited on 11/28/2018).
- [5] Gutenberg. *Don Quijote by Miguel de Cervantes Saavedra*. 2018. URL: <http://www.gutenberg.org/ebooks/2000> (visited on 11/28/2018).
- [6] Gutenberg. *La Seconda e Terza Guerra Punica by Leonardo Bruni*. 2018. URL: <http://www.gutenberg.org/ebooks/45126> (visited on 11/28/2018).
- [7] Gutenberg. *Moby Dick; Or, The Whale by Herman Melville*. 2018. URL: <http://www.gutenberg.org/ebooks/2489> (visited on 11/28/2018).
- [8] Gutenberg. *The Life and Adventures of Robinson Crusoe by Daniel Defoe*. 2018. URL: <https://www.gutenberg.org/ebooks/521> (visited on 11/28/2018).
- [9] Gutenberg. *Vingt mille Lieues Sous Les Mers — Complete by Jules Verne*. 2018. URL: <http://www.gutenberg.org/ebooks/5097> (visited on 11/28/2018).
- [10] Gutenberg. *War and Peace by graf Leo Tolstoy*. 2018. URL: <http://www.gutenberg.org/ebooks/2600> (visited on 11/28/2018).
- [11] Java2S. *ReadLines: read file to list of strings*. 2018. URL: http://www.java2s.com/Tutorial/Java/0180__File/ReadLinesreadfiletolistofstrings.htm (visited on 11/28/2018).
- [12] LogicBig.com. *Java Command Line - How to create table in a Command line application in Java?* 2018. URL: <https://www.logicbig.com/how-to/code-snippets/jcode-java-cmd-command-line-table.html> (visited on 11/28/2018).

- [13] Arun Prakash. *How can I write a Java program to find each occurrence of a character in a string which is given as an input from a console without using any built-in functions?* 2018. URL: <https://www.quora.com/How-can-I-write-a-Java-program-to-find-each-occurrence-of-a-character-in-a-string-which-is-given-as-an-input-from-a-console-without-using-any-built-in-functions> (visited on 11/28/2018).
- [14] Rob. *Easy way to remove accents from a Unicode string?* 2018. URL: <https://stackoverflow.com/a/15190787> (visited on 11/28/2018).
- [15] ML Wiki. *Smoothing for Language Models*. 2018. URL: http://mlwiki.org/index.php/Smoothing_for_Language_Models (visited on 11/28/2018).

A UML Class Diagram



Powered by yFiles

Figure 5: UML Class diagram of the Automatic Language Identifier

B Expectations of Originality

Faculty of Engineering and Computer Science Expectations of Originality

This form sets out the requirements for originality for work submitted by students in the Faculty of Engineering and Computer Science. Submissions such as assignments, lab reports, project reports, computer programs and take-home exams must conform to the requirements stated on this form and to the Academic Code of Conduct. The course outline may stipulate additional requirements for the course.

1. Your submissions must be your own original work. Group submissions must be the original work of the students in the group.
2. Direct quotations must not exceed 5% of the content of a report, must be enclosed in quotation marks, and must be attributed to the source by a numerical reference citation¹. Note that engineering reports rarely contain direct quotations.
3. Material paraphrased or taken from a source must be attributed to the source by a numerical reference citation.
4. Text that is inserted from a web site must be enclosed in quotation marks and attributed to the web site by numerical reference citation.
5. Drawings, diagrams, photos, maps or other visual material taken from a source must be attributed to that source by a numerical reference citation.
6. No part of any assignment, lab report or project report submitted for this course can be submitted for any other course.
7. In preparing your submissions, the work of other past or present students cannot be consulted, used, copied, paraphrased or relied upon in any manner whatsoever.
8. Your submissions must consist entirely of your own or your group's ideas, observations, calculations, information and conclusions, except for statements attributed to sources by numerical citation.
9. Your submissions cannot be edited or revised by any other student.
10. For lab reports, the data must be obtained from your own or your lab group's experimental work.
11. For software, the code must be composed by you or by the group submitting the work, except for code that is attributed to its sources by numerical reference.


You must write one of the following statements on each piece of work that you submit:

For individual work: **"I certify that this submission is my original work and meets the Faculty's Expectations of Originality"**, with your signature, I.D. #, and the date.

For group work: **"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality"**, with the signatures and I.D. #s of all the team members and the date.

A signed copy of this form must be submitted to the instructor at the beginning of the semester in each course.

I certify that I have read the requirements set out on this form, and that I am aware of these requirements. I certify that all the work I will submit for this course will comply with these requirements and with additional requirements stated in the course outline.

Course Number: COMP6721
Name: Adrien Poupa
Signature: 

Instructor: Dr Leila Kosseim
I.D. # 40059458
Date: October, 5th 2018

¹ Rules for reference citation can be found in "Form and Style" by Patrich MacDonagh and Jack Bordan, fourth edition, May, 2000, available at <http://www.encs.concordia.ca/scs/Forms/Form&Style.pdf>.
Approved by the ENCS Faculty Council February 10, 2012