

SAE 1.01

-

Project "Automatic  
classification"

DUCHOSAL Jolan  
MARROT Matys

19/01/2023

Group TP : A2

Table of contents

<b>1. Introduction</b>	<b>3</b>
<b>2. Part 1</b>	<b>3</b>
<b>3. Part 2</b>	<b>5</b>
<b>4. Result</b>	<b>6</b>
<b>5. Complexity</b>	<b>6</b>
<b>6. Conclusion</b>	<b>7</b>

## 1. Introduction

In this SAE 1.01, the requested work was done in the form of a project in order to create an automatic classification program. In this work, these are dispatches that have been classified as quickly and reliably as possible. Dispatches are short journalistic information, grouped in 2 text files made available to us, divided into 5 categories : ENVIRONNEMENT-SCIENCES, CULTURE, l'ECONOMIE, POLITIQUE and SPORTS.

To obtain this reliability in these classifications, we had to use a lexicon by category. Indeed, a significant word in a category is not necessarily in another category. So, the goal is to find a significant word for each of our 5 categories. Then, when we have those significant words, they are given an importance value between 1 and 3. The higher the value, the more important the word is in that category. In this work, the construction of this lexicon of words will be done manually in Part 1, and will be automated in Part 2.

Then, when our 5 lexicons are created (one per category), we calculate a score, by adding each value of each word for each category. The category of a dispatch is determined by the highest score obtained for a category.

Moreover, among our 2 text files which had all our dispatches, there is one, the *depeches.txt* file, which served as a support for us to create and check our lexicons. And the second file, named *text.txt*, serves us only as tests of our classification programs. These files are built with 100 dispatches per category, and are presented with the dispatch identification number (from 001 to 500) as the first line. Then, the second line corresponds to the date (written in the form: *ddmmyy*, with the last two digits for the year, for example: 190123). Then, for the third line, there is the category of the dispatch (one of the 5 categories written at the beginning of the introduction). And finally, from the fourth line which corresponds to the text of the dispatch (this is where we found our significant word).

Then, to stock our lexicons we used text files, one per category, created in the form of a line composed of the word of importance followed by a ":" as a separator, and finally its value (i.e. *word:value* , for example: *président:3*).

And finally, to obtain our classification results, we used an automatically generated file, which follows the form of identification number followed by a ":" separator, then its category name (i.e. *number:category*, for example : 001:SPORTS). In addition, in this answer file after the results of the 500 dispatches, there are 5 lines corresponding to the percentage of correct answers per category (example: SPORTS: 75%) and there is a last line which corresponds to the overall average of the percentages of correct answers among all our categories (example: MOYENNE: 80%).

## 2. Part 1

In the introduction, we talked about manually building the 5 lexicons by categories of important words in this first part. Indeed, to start this part 1, we added at the beginning between 20 and 30 words which seemed important to us that we found in the *depeches.txt* file. It was largely enough because, our goal at that time was to make the program work well, we did not try to have good results directly. When the program worked properly, little by little, we started to add more and more important words to complete our lexicons.

At the beginning of the word addition, we realized that a word could have its plural or a similar word in another dispatch, so we added this to our lexicons. Then, we realized that all the texts of our dispatches received the function : *toLowerCase()* which means that if, in the texts, there are uppercase letters, then they are reduced to lowercase letters, so no need to add capital letters in our lexicons (otherwise, the word we wrote can never be found).

Categorie.initLexique(String nomFichier)

→ Objective : convert a file with "string:int" type "paireChaineEntier" into a vector of "paireChaineEntier"

- We initialize "lexicon"
- We open the file named "nomFichier"
- Thanks to a scanner, we read the file line by line
- Each line contains a string/int pair separated by ":"
- So we take the index of the ":" in each line
- The string is from index 0 to the index of ":"

- The int (which corresponds to the score of the string and which is only one character) is then at the index of ":" + 1
- Next, we add to "lexique" a "paireChaineEntier" with the string and the int found previously

UtilitairePaireChaineEntier.entierPourChaine(ArrayList<PaireChaineEntier> listePaires, String chaine)

→ Objective : find the int associated with the "chaine" in the vector of "paireChaineEntier"

- We traverse the vector of paireChaineEntier as long as the vector is not finished and that "chaine" is different from the string of the "paireChaineEntier"
- Once the iteration is finished, we see if the index at which we stopped corresponds to the index of the wanted string, if so we recover the associated int

Categorie.score(Depeche d)

→ Objective : calculate the dispatch score "d" for the category with which the method was called

- We initialize "score" to 0
- We traverse the words of the dispatch "d"
- For each word, we add to "score" the result of the method

UtilitairePaireChaineEntier.chaineMax(ArrayList<PaireChaineEntier> listePaires)

→ Objective : find the largest int in all the paireChaineEntier and return her string

- We create 2 variables "indiceMax" and "scoreMax" initialized to 0 in order to save the highest score and save its position in "listePaires"
- We traverse each iteration of "listePaires"
- For each iteration, we check if the int of the "paireChaineEntier" is greater than "scoreMax"
- If the int is greater than "scoreMax", so "scoreMax" becomes the int and we save its position in "indiceMax"
- Else, we go to the next "paireChaineEntier"

UtilitairePaireChaineEntier.indicePourChaine(ArrayList<PaireChaineEntier> listePaires, String chaine)

→ Objective : find the index in "listePaires" of the "paireChaineEntier" object which have string is "chaine"

- We traverse the vector "listePaires" while the string of "paireChaineEntier" is different from the wanted string "string"
- Once out of the loop, we check if the string of "paireChaineEntier" is the wanted "string", so we return the index, else we return -1

UtilitairePaireChaineEntier.moyenne(ArrayList<PaireChaineEntier> listePaires)

→ Objective : calculate the average of all the int contained in the vector "listePairs"

- We define a variable "somme"
- We traverse all the elements of the vector "listePairs"
- For each "PaireChaineEntier" of the vector "listePairs", we take the int and we add it to "somme"
- Once all the vectors have been traverse, we divide the "somme" of all the int of the vector "listePaires" by the number of elements contained in "listePaires"

Classification.classementDepeches(ArrayList<Depeche> depeches, ArrayList<Categorie> categories, String nomFichier)

→ Objective : written in a file named "nomFichier" : the number and the category found by the algorithm for each dispatch, the number of correct answers per category and the average correct answer

- We create a result vector of "paireChaineEntier" which will contain as string the name of a category and as int the number of correct answers of this one
- We traverse all the dispatches of the vector "depeches"
- For each dispatch, we create a vector of "paireChaineEntier" named "VScore", which will contain as string, the name of a category and as int, the score of the dispatch in this category
- We calculate the score of the dispatch in each category, thanks to the Categorie.score method. Then, we add this score with the name of the category in the vector "VScore"

- Next, using the VScore vector and the "UtilitairePaireChaineEntier.chaineMax(VScore)" method, we determine the category with the highest score
- We check if the category found corresponds to the category of the dispatch, then we add a good answer to this category, then we increment the int associated with the string which corresponds to the category in the result vector, to do this, we use the "UtilitairePaireChaineEntier.indicePourChaine" method to find the index of "paireChaineEntier" which have string is category in the result vector
- Next, we write in the file named "nomFichier" the number of the dispatch and the category found
- Once we traverse all the dispatches, we write the number of correct answers for each category then calculate the average of correct answers

### 3. Part 2

Unlike part 1, in this part 2, the objective is to automatically build lexicons for each of our categories in order to obtain the best possible results, the program must be reliable and fast. To obtain the best possible results, we have developed machine learning, using artificial intelligence, which allows our program to learn on its own from the data that we will provide to it.

The data we provide are those of the *depeches.txt* file, the method will therefore analyze this file to obtain the important words for each category. Here, the important words are those present in at least one dispatch to calculate a score. This score is then calculated according to the number of times a word appears in dispatches of one category, as well as the number of times a word appears in dispatches of another category. Depending on the score of each of our words, we can then give a value of 1, 2, or 3 (as in part 1). Based on this, we tested several times with different scores for each value before we could find good results.

Classification.initDico(ArrayList<Depeche> depeches, String categorie)

→ Objective : add all the words of the dispatches of a category in a vector

- We create a vector dictionary of "PaireChaineEntier" which will have as string all the words of all the dispatches which are in category, and as int 0 for the moment
- We traverse each dispatch in the vector "depeches"
- For each dispatches, we check if her category corresponds to "categorie"
- If they are the same, we traverse the vector of words of each dispatch (the words that the dispatch contains)
- For each word of the dispatch, we check if it has more than 3 characters (to avoid useless little words)
- If it has more than 3 characters, we check if it is already in the dictionary vector
- If it is in the dictionary vector, we go to the next word, else we add it

Classification.calculScores(ArrayList<Depeche> depeches, String categorie,

ArrayList<PaireChaineEntier> dictionnaire)

→ Objective : increment the score of each word according to their recurrence in the dispatches

- We traverse each dispatch in the vector "depeches"
- For each dispatch, we traverse his vector of words (the words that the dispatch contains)
- For each word contained in the "depeches", we traverse the vector "dictionnaire" until we find the same word
- If the word of the dispatch is in a "dictionnaire", we compare the category of the "depeches" with "categorie". If they are the same : we add 1 to the score of the word, else we subtract 1
- If the word in the "depeches" is not in the "dictionnaire", we continue with the next word

Classification.poidsPourScore(int score)

→ Objective : determine the weight (or value) of a word according to its number of recurrences ("score") in the dispatches

Classification.generationLexique(ArrayList<Depeche> depeches, String categorie, String nomFichier)

→ Objective : create a file containing all the important words of a "categorie" with their weight (or value)

- We start by take all the words of a dispatch category in a vector "dictionnaire" using the Classification.initDico method
- Next, we increment the recurrence number of each of the words contained in the vector "dictionnaire" using the Classification.calculScores method
- Then, we traverse the vector "dictionnaire", and for each iteration ("paireChaineEntier" which have string is a word contain in a category type "depeches") we determine the score of a word according to its number of iterations using the "Classification.poidsPourScore" method
- If the score is greater than 0, so we write in the file named "nomFichier" the "paireChaineEntier"

TestXML.rss(String nomFichier, String nouveauFichier)

→ Objective: convert the XML file *nomFichier* of dispatches into a file suitable for our program of name *nouveauFichier*

- We create three variables: *categorie*, *date* (all dispatches contained in the *nomFichier* file are part of the same category and have the same date) and a dispatches counter *compteurDepeche*
- We browse the *nomFichier* file line by line
- For each line, we check if the line contains the `<title>` tag, which means that the category is in the line. If it contains it, we retrieve the category by storing it in *categorie*
- For each line, we check if the line contains the `<pubDate>` tag, which means the date is in the line. If it contains it, we convert the syntax of the date to conform it to our program
- For each line, we check if the line contains the `<item>` tag, which means the beginning of a dispatch. If it contains it, we browse the next lines independently to find the `</item>` tag (who notes the end of the dispatches)
- For each line in `<item>`, we check if the line contains the `<title>` tag, which means the next line is the title of the dispatches
- For each line in `<item>`, we check if the line contains the `<description>` tag, which means the next line is the description of the dispatches
- Once the `</item>` tag is reached, we write the dispatches in the *nouveauFichier* file

## 4. Result

In our first experiment, we started by creating our own lexic from the file *depeche.txt*. So we searched for the most recurrent words of each category with our hands. The result is obviously inconclusive because the number of keywords is low. We can already notice that the categories with the most words (sports and environnement-sciences) have a better score than the 3 others.

For our second experiment, we simply added more words to our lexicon (around sixty in each category) and tested with the same *depeche.txt* file. The result is better than before but remains questionable.

For the third experiment, we used the same lexicon as experiment 2 but with the *test.txt* file. The result is obviously less good than the previous experience because the lexicon was made from the *depeche.txt* file, so the results are necessarily less suitable.

In the fourth experiment, the lexicon was generated directly from the *depeche.txt* file. The average is 98%, which is a very good score.

In the fifth experiment, we generated the lexicon from the *test.txt* file. The average is therefore the same as in the previous experiment.

For the sixth experiment, we tested to implement a dispatches file from an XML file coming from an RSS feed.

In the seventh experiment, we tried to optimize the program to obtain a faster and less expensive result. The lexicon is always generated from *depeches.txt* file and the result is optimal, we will not be able to obtain a better result.

## 5. Complexity

The complexity of the *Categorie.score(Depeche d)* method can be note :

- in the worst case :  $cost_{score}(d) = O(d.getMots().size() * lexique.size())$
- in the best case :  $cost_{score}(d) = \Omega(d.getMots().size())$

Indeed, in the best case, all the words of the dispatch will be the first word of the lexicon. So, we can make only one comparison per word, the one with the word and the first element of the lexicon. This comparison gives a complexity equal to the number of words in the dispatch.

In the worst case, all the words in the dispatch will be the last word in the lexicon. Then, we can traverse all of the lexicon for each word of the dispatch in order to find its score. So, we can estimate that the average cost of  $Category.score(Depeche\ d)$  can be written :  $cost_{score}(d) = n^2$ .

The complexity of the method  $Classification.calculScores(ArrayList<Depeche> d, String c, ArrayList<PaireChaineEntier> dic)$  can be note :

- in the worst case :  $coût_{calculScores}(d, c, dic) = O(d.size() * d.getMots.size() * dic.size())$
- in the best case :  $coût_{calculScores}(d, c, dic) = \Omega(d.size() * d.getMots.size())$

Indeed, in the best case, it will be necessary to traverse all the dispatches contained in  $d$ , as well as all the words of each dispatch, and that all these words are the same so that they are all found at the first occurrence of  $dic$ . Which gives us a complexity equal to the number of dispatches times the number of words (on average) times per 1. On the contrary, in the worst case, we can scan all the dispatches contained in  $d$ , as well as all the words of each dispatch and that all these words are found at the last occurrence of  $dic$  (it will therefore be necessary to traverse the entire vector  $dic$ ). This gives us a complexity equal to the number of dispatches times the number of words (on average) times the number of occurrences in  $dic$ .

On average, we can estimate the cost of the method  $Classification.calculScores(...)$  can be written :  $cost_{calculScores}(d, c, dic) = n^3$ .

## 6. Conclusion

In conclusion, during the realization of this project "Automatic classification", we realized that the use of machine learning is much more effective to create lexicons automatically, when we did it manually, we did not have not so many good results. This project is a first step in learning about the development of artificial intelligence. We sometimes had some problems, because some notions were new to us. But, we were able to all realize the project.

On improving the program, we saw that it was possible to reduce the time by sorting our lexicons. Indeed, we decided to sort our 5 lexicons in alphabetical order, then by weight. This sort allows some of our programs to do a partial traversal, instead of a full traversal if they are not sorted. So, to reduce the execution time.

In Part 2, improving the programs saves about 300 ms over an unimproved program.