SAE 1.02

Projet "Classification automatique"

20/01/2024

IUT2

Dép. Informatique

N.Rayot M.Erdogan

I. Introduction

To begin, we will summarize the objectives of this project. This is a project where the objective is to sort a "test.txt" dispatch file.

Every dispatch will be ranked in one of these 5 categories: sciences et environement , culture , economy , pilitique , sports. And obtain the best possible result.

For this, we will create 5 lexicons, containing the words corresponding to their category. First of all, we created the lexicon ourselves. Then we will create an algorithm to automatically create the lexicons.

The lexicon file is a file with the extension type ".txt".

Into this file, there are two types of information, separated by a : . The first piece of information is one word. The second is the importance of this word, which is translated with a number between 1 and 3.

II. Summary of what we did

In this project, In this part, we explain how we have realised the project.

We had to first make one piece of code for the basis of the project. Create the classes "Categorie" and "PaireChaineEntier". That will serve us all the time in the next step.

The second step is to create several algorithms :

There are many simple functions to make different things. In The class "UtilitairePaireChaineEntier".

- An algorithm to sort dispatches by category, thanks to the words of a lexicon. Then compare the results we found with the true category of that dispatch. And write they results in file.

The third step is to automatically create the lexicon of words, that we created ourselves before.

For lexicons automatically, we have need to create the functions : "initDico","calculScores","poidsPourScore" and "generationLexique".

I first put in calculScore an increment of 2 and a decrement of 1. My correct answer average was from 68%

But I find that when I put an increment of 1 and a decrement of 3. My correct answer average increases up to 73%

The last step consists of testing and upgrading our algorithm.

- Try to reduce the number of errors.

- Count the number of comparisons and the execution time of the program.

- Reduce the time of execution.

- Reduce the number of comparisons of our algorithm.

To count the number of function comparisons, I chose to equip several functions thanks to the class "Paireresultatcompteur".

We didn't have time to look for opportunities for improvement for a long time. In the second part of the report, we will analyze the results.

III. Result and analysis

We will see the presentation of the results of the algorithm on the different lexicons:

1. Result of the algorithm on yourself lexicons:

- The result of the good answers:

SPORTS:95%

CULTURE:2%

POLITIQUE:37%

ECONOMIE:21%

ENVIRONNEMENT-SCIENCES:0%

MOYENNE:31.0%

Is not a very good result because creating the lexicon takes a lot of time, And this is not of great importance for the following.

-The execution time is between 60ms and 80ms (I'm running my code on my old laptop, maybe this has a consequence for the time)

-The number of comparisons of my code is 91170 comparisons.

2. Result of the algorithm on the lexicons created automatically.

-The result of good answers:

CULTURE:70%

POLITIQUE:70%

ECONOMIE:68%

ENVIRONNEMENT-SCIENCES:61%

MOYENNE:70.0%

It’s a decent score, but it’s not perfect yet. I know I can still improve it, but we will talk about it later in this report.

-The execution time is between 600ms and 800ms. Execution time is between 600ms and 800ms. About 30% of the time is used to create the lexicon and 70% for classification.

-The number of comparisons of my code is 1 427 304. Among these comparisons, 1 327 945 are caused by the creation of the lexicon and 99 359 by the classification.

With this small number of successes, we could ask ourselves the question, "The algorithm has a bug?”.

But no, because when we test the algorithm on the lexicons with which it was trained. It has 99% correct answers.

It's the fault of the lexicons, in the first the lexicons are not long enough. And in the second they is not enough accurate.

IV. Analysis of the complexity of several function

The score function is executed in 1 ms and by doing 87 421 comparisons.

The callculscore function executed in 25 ms an does 225 364 comparisons.

V. Conclusion

There are many points that could be improved on this project.

- In the categorization results, we could increase the % of correct answers. By creating variations on the words, by adding the s in the end of the word, for example.

- In number of comparisons when creating automatic registers. For example, sort the vector and have fewer comparisons to make. Or delete the useless words in the lexicons.

- We can try to create a lexicon with the root of the word to see if the program is better

- The score interval is small (1,2,3) increasing the number of possibilities would be a good thing