SAE 1.01/02

I. Introduction :

The goal of this project is to create a program that can automatically classify news dispatches as quickly and accurately as possible. A news dispatch is a short text that provides journalistic information. To do this classification, the program will focus on words that show which category a dispatch belongs to. For instance, terms like "match" and "player" would suggest that the dispatch is related to sports.

Each category will have a set of key words that help determine its topic. Every word will be assigned a weight to indicate how strongly it relates to the category. The weight can be 1, 2, or 3, where 3 means the word is strongly associated with this category.

For example, the lexicon for the SPORT category could include words like "sport" (3), "cup" (3), "match" (3), "player" (2), and "home" (1). In the first part of the project, we will manually create these lists by looking at about 100 example dispatches for each category. The goal is to pick the words that best represent each category, meaning words that are specific to that category and not used much in others. In the second part, we will create a method to build these lists automatically and at the end, with one file, have the result of the ranking.

II. Program :

II.a) Setting manual and automatic mode + result

Program:

Setting manual and automatic mode + result

In the first part, we have implemented manual classification using the following (simplified) system:

-For each category, look at each news item in that category and put the most redundant words (between 40 and 50 words) into a text file

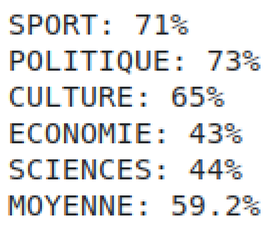
-Create the class «PaireChaineEntier» which will serve several things but here will allow to assign a category and its score for a dispatch

-Creation of the function «classementDepeches» which will use several other functions (created beforehand) to create a new ranking text file that will display the ID + category of each news item and has at the end the percentage of precision of each Category as well as an overall average

-In the hand, we create a Category by Category object and initialize the lexicon that we created in step 1 then call the function «classementDepeches»

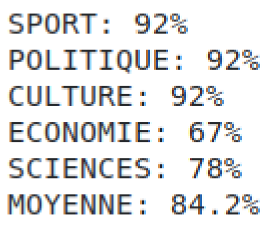
In the second part, an automatic generation of the lexicon is set up using the number of similar words in the same category.

We will therefore study the results on a "test" dataset



```
SPORT: 71%
POLITIQUE: 73%
CULTURE: 65%
ECONOMIE: 43%
SCIENCES: 44%
MOYENNE: 59.2%
```

manual



```
SPORT: 92%
POLITIQUE: 92%
CULTURE: 92%
ECONOMIE: 67%
SCIENCES: 78%
MOYENNE: 84.2%
```

Automatic

The automatic version is more precise and in the «poidsPourScore» function it is possible, by modifying the score scale, to increase the percentage of precision.

II.b) Adding the RSS flux

It is possible to add dispatches of a category in the database directly from internet using the «add files» function and these will be added to the following category already present inside. It is also possible to extract the lexicon from new dispatches with «generationLexique» which will generate a lexicon automatically.

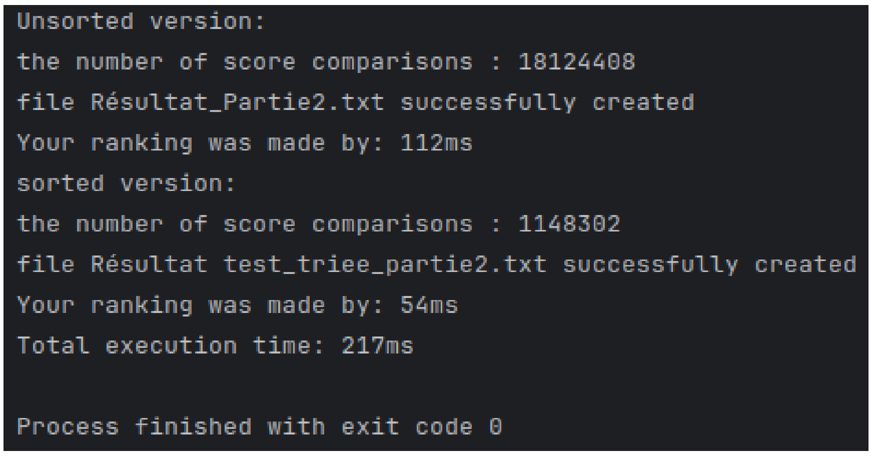
II.c) Comparison of the unsorted and sorted version

We have added the functions “classementDepeches_triee”, “score_triee” and “entierPourChaine_triee” which together will be able to calculate the score by doing a dichotomous search in a sorted lexicon. We therefore also had to add the method “lexiques_triee” of the Category class which allows you to take an unsorted lexicon text file and create a new similar and sorted file.

All this will allow us to take the lexicons generated automatically, sorted them and use them in the same way as before, that is by creating a ranking automatically but comparing the effectiveness of unsorted and sorted versions.

We have therefore equipped the 3 previous functions as well as their unsorted version to look at the number of comparisons. We have also added a total processing time calculation and a calculation specific to the functions “classementDepeches” and “classementDepeches_triee”.

Result :



```
Unsorted version:
the number of score comparisons : 18124408
file Résultat_Partie2.txt successfully created
Your ranking was made by: 112ms
sorted version:
the number of score comparisons : 1148302
file Résultat test_triee_partie2.txt successfully created
Your ranking was made by: 54ms
Total execution time: 217ms

Process finished with exit code 0
```

The unsorted version has 18 Millons and 1 Million for the sorted version.
It can be seen that the unsorted version has much more comparisons and that this has an impact on processing time and therefore becomes slow for large amounts of data while the sorted version is much faster and more efficient. In the example there is 58 milliseconds difference.

So we will continue using the sorted version.

II.d) Implementation of the KNN method

To implement the KNN method, we have added a function “methode_KNN” which will allow it to be carried out.
The KNN method consists of finding the nearest neighbors K to the dispatch search among those in the database according to a distance, here the number of words in common in order to find the category that repeats more among the neighbors K.
Here we will use the class PaireChaineEntier which will have for chain the category of the dispatch of the database and in whole the number of words in common with the dispatch that we try to determine.

After having gone through each word of each dispatch we get an Arraylist of PaireChaineEntier which will be sorted (by «tri_entier») by Integer then we will keep the K first and we end up looking at the category among the neighbouring K (with «choix_categorie»).
In the hand we added a loop that will use this method for each test file and with the database. the ID and category determined by the method is displayed in the test_knn file and then at the end an average that displays the correct percentage of answers is added.
We were therefore able to determine for which K the method is the most effective.

Result :

For k = 3	Average : 51.8%
For k = 4	Average : 55.2%
For k = 5	Average : 53.6%

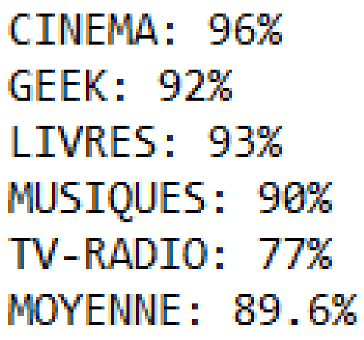
So the method is more efficient with k=4.

II.e) Creation of Classification 2

Classification 2 is a new class that will be based on the Classification class but which will automate absolutely everything to allow any data to be processed.

The system works as follows:
-We have created a new function “classement_automatique” which takes a list of dispatches and will return a file ranking and via the different lexicons of each category present among the dispatches.
-This function will start by browsing each dispatch in order to extract the different categories then by reusing «generationLexique» will create a lexicon for each category
-Then will sort the lexicons with the method «lexiques_triee».
-It will end by calling the function “classementDepeches_triee” which will create the file based on the lexicons and categories found previously.

In the "lexiques_classement_automatique" file, we tested with the data of depeches2 and obtained the following result:



```
CINEMA: 96%
GEEK: 92%
LIVRES: 93%
MUSIQUES: 90%
TV-RADIO: 77%
MOYENNE: 89.6%
```

In conclusion, the system is working perfectly, automatically and with a number of comparisons to decrease sharply.

III. Conclusion :

In order to improve our program we could look for ways to improve our results by optimizing our code and making it more complex but our level of programming does not allow us at the moment. We could also improve the usability of tests to make them more understandable and efficient for solving problems in code.

However by studying the complexity of "calculScores" which is between 40 Million and 45 Million comparisons according to the lexicon generate. I think it would be possible to reduce this number by sorting directly the words of the lexicons when creating the lexicon, notably by modifying the "initDico" function.