SAE1.01

Samuel Dubiez
Ethan Hermil

A introduction of the SAE:

The SAE1.01 is a project in pairs consisting of creating a program that can sort news according to 5 categories witch are: ENVIRONNEMENT-SCIENCES, CULTURE, ECONOMIE, POLITIQUE, SPORTS. These categories will be determinatedin the program by a specific list of words for that categorie. Then we will have to code an AI that will learn from already sorted files to create the lists of words for each category in order to have the best result. Then finally improve and optimize the program so that it is the fastest, most precise and reliable possible by modifying the AI responsible for the classification of the news.

What we manage to do and not to do:

During this SAE, we manipulated a lot of code and especially created many functions and improvements necessary for the proper functioning of the program. We managed to code each of the required functions which are for part 1:

initLexique() which is used to create the lexicon list from the desired category lexicon file using a scanner and the fact that each line of the file is constructed in 2 parts.

```
public void initLexique(String nomFichier) throws FileNotFoundException {
    PaireChaineEntier mots;
    String ligne;
    FileInputStream file = new FileInputStream(nomFichier);//début du fichier
    Scanner scanner = new Scanner(file);//mettre le fichier dans le scanner
    while (scanner.hasNextLine()) {//tant qu'il y a une ligne
        ligne = scanner.nextLine();//scanner la ligne
        String[] tabchaine = ligne.split( regex: ":");//créer une liste de string compo
        mots = new PaireChaineEntier(tabchaine[0], Integer.parseInt(tabchaine[1]));/
        lexique.add(mots);//rachoujet mots dans lexique
    }
}
```

entierPourChaine() which uses a dichotomic search to find whether a specific string is in the lexicon and if so, returns the associated value.

```
public static int entierPourChaine(ArrayList<PaireChaineEntier> listePaires, String chaine) {
    int ref = indicePourChaine(listePaires, chaine);//récuperer l'index de "chaine" dans "listePaire"
    return (ref < 0) ? 0 : listePaires.get(ref).getEntier();//si "chaine" n'est pas dans "listePaire"
}                                                           // return 0 sinon return sont "getEntier"
```

score() which for each word in news n°x, we retrieve its index in the lexicon and then take its corresponding weight which we add to the score. At the end, we return the score (the total weight of all the words in the news item).

```
public int score(Depeche d) {
    int score = 0;
    for (String str : d.getMots()) {//pour chaque String dans "d.getMots()"
        score += UtilitairePaireChaineEntier.entierPourChaine(lexique, str);//ajouter le poids de "str" dans "score"
    }
    return score;
}
```

chaineMax() which returns the string associated with the largest value from a list of paireChaineEntier by comparing every number in the list.

```java
public static String chaineMax(ArrayList<PaireChaineEntier> listePaires) {
    PaireChaineEntier max = new PaireChaineEntier( chaine: "",  entier: 0);
    for (PaireChaineEntier pce : listePaires) {//pour chaque classe de PaireChaineEntier dans "listePaires"
        if (pce.getEntier() >= max.getEntier()) {//si l'entier de "pce" est plus grand que celui de "max"
            max = pce;
        }
    }
    return max.getChaine();
}
```

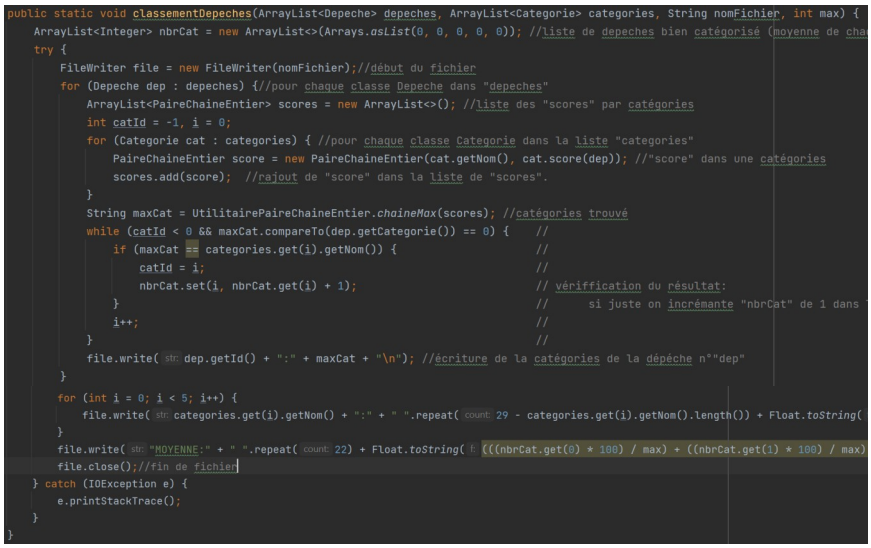
indicePourChaine() which returns the index of a string in a list, and if the string is not present, it returns the negative index of where to insert the element in the list to keep it sorted. This function uses a dichotomic search.

```java
public static int indicePourChaine(ArrayList<PaireChaineEntier> listePaires, String chaine) {
    if (listePaires.isEmpty()) {//si "listePaires" est vide
        return -1;
    } else if (listePaires.get(listePaires.size() - 1).getChaine().compareTo(chaine) < 0) {//s
        return -listePaires.size() - 1;
    } else {
        int inf = 0;
        int sup = listePaires.size() - 1;
        int m;
        while (inf < sup) {
            m = (inf + sup) / 2;
            if (listePaires.get(m).getChaine().compareTo(chaine) >= 0) {
                sup = m;
            } else {
                inf = m + 1;
            }
        }
        if (listePaires.get(sup).getChaine().compareTo(chaine) == 0) {
            return sup;
        } else {
            return -(sup + 1);
        }
    }
}
```

moyenne() which calculates the average of all the numbers in a given list and returns it (we did not use it so it is commented out).

```java
/**  inutilisé
public static float moyenne(ArrayList<PaireChaineEntier> listePaires) {
    float moy = listePaires.get(0).getEntier();
    for (int i = 1; i < listePaires.size(); i++) {
        moy = (moy + listePaires.get(i).getEntier()) / 2;
    }
    return moy;
}
**/
```

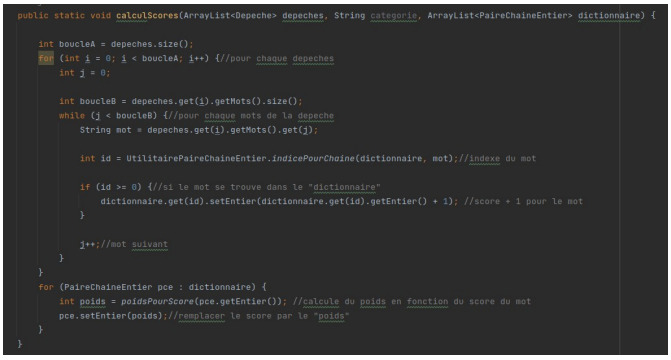
classementDepeches() which opens the edition of the file "nomFichier" with the FileWriter object. It initializes nbrCat which serves to count the news that have been placed in the correct category. For each news in "depeches", it initializes an ArrayList called "scores" which is used for the score. Then it calculates the score of the news item for each category stored in "categories" with the help of .score() which will then be added to "scores". Once done, it looks at the category that has the highest score using chaineMax() and stores it in "maxCat". It checks if the category found corresponds to the one specified in the news item, if so it increments "nbrCat" in the corresponding index and writes the result using file.write(). When all the news items in "depeches" are sorted, it writes the success rate of each category found with the calculation ("nbrCat"*100)/max) max representing the actual number of news items in the category. It then displays the overall success rate and stops editing the file with file.close().



Then we have coded for the part 2:

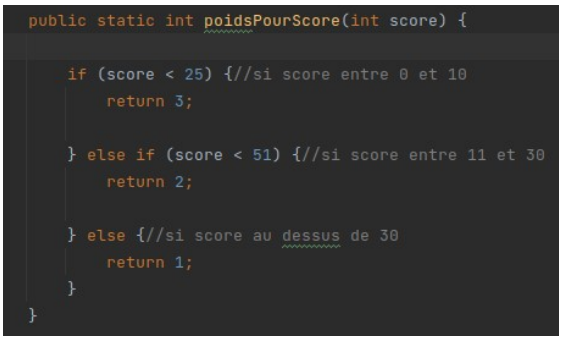
initDico() witch for each word of each news, looks if it is in the lexicon, if not the word is added to the lexicon in the right place so that the lexicon stay sorted for the dichotomic search.

calculScores() which for each word of a given news item, looks if the word is in the lexicon, if yes, it adds 1 to its score, otherwise it removes it from the news

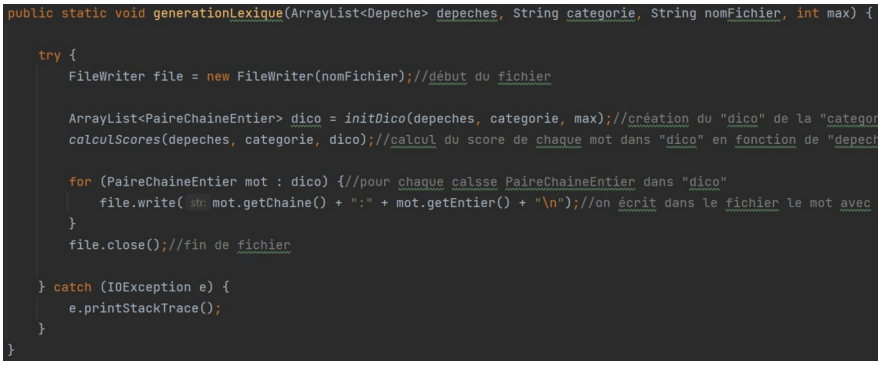


item so as not to calculate it. Then it calculates the weight of each word of the news item in the lexicon.

poidPourScore() which takes the numbers and depending on their value changes them to 1(between 0 and 5), 2(between 6 and 9), 3(greater than 9), or -1(less than 0) because they are removed from the category dictionary due to the fact that the word in question belongs to another category.



generationLexique() which overall, for each category, takes the previous functions to have the word list of the category, then calculates the score of each word and finally writes them in the category lexicon file.

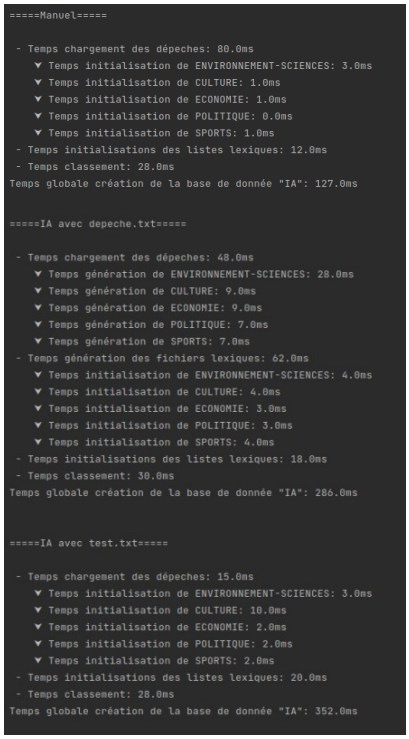


Then we added things that were not asked for to optimize the program, we put a system that as soon as initDico() adds a string in a dictionary it inserts it in a sorted way thanks to a dichotomic search that gives the location of the value if it is already present or the negative value of where the value should be placed.

Finally, in the main we added several parts corresponding to the different tests of this SAE: a first part testing the program with the lexicons generated by hand that takes the lexicons and sorts the news items from the depeches.txt file, then we have the part with the AI that consists of generating the lexicons with the depeches.txt file and then testing the efficiency of the lexicons on news items by putting the result in a first result file, then testing without generating new lexicon on the test.txt file and putting the results in a second answer file. But we did not succeed in using the average function because we did not understand what it could serve us, nor optimizing results using the rss method because we did not have the time.

Results:

For the result, we have a global sucess rate of:
-67% with manual lexicals and the depeches.txt file
-98% with the AI and the depeches.txt file
-69% withe the AI and the test.txt file
We can see thet the results are very similar between manual and AI withe the test.txt file.
This can be that becose of the fact that the database is very small or that the words in the exemples are not to specific to one single category.



Complexity analise:

After a long analise, we have found that every part have the following complexity :
-score(): O(n log (n))
-calculScores(): O(n^2 log(n))
-Depeche(): O(n)
-Classification(): O(n^3)
We can see that the complexity of Depeche() is small so there is not to much upgrade compare to calculScore(). Some upgrades will be possible on Classification() to change from O(n^3) to O(n2 log(n)).

Possible upgrades:

From what we can see, some parts can be improved. On the initDico() function, for example, we can create a single ArrayList with all the words present in depeches.txt with a new function, and then distribute them in the different category lexicons. This would allow us to avoid calculating the weight of a word several times since it does not depend on the category and therefore remove some comparisons. It would also be possible to change the binary search system to a more efficient one to try to reduce execution time or complexity. Other improvements are possible in terms of the result, such as adding more starting data when generating to have more reliable lexicons or changing the weight model that is currently functional only for small samples. The last thing that we can do is to replace all the ArrayList that are used to stock the values, by hashMap because the search is more simple with the keys of the hashMaps.