SAE S1.01/02

Project "Automatic classification"

By Jarod TIVOLLIER et Mattéo Banroques Mathian

Introduction

This week, we have been working in intelliJ IDEA on a java project by group of two. The goal was to make a reliable and efficient programm that automatically classify news between 5 categorys : “ENVIRONNEMENT-SCIENCES, CULTURE, ECONOMIE, POLITIQUE, SPORT”, based on the word of the news and their relevance into a certain category. To mesure their importance we had to use score and “weight”. Score is calculated with the sum of the weight attributed for each word of a news (part 1 of the SAE) or the number of time a word occurs(part 2 of the SAE), and weight is the choice of the programmer to attribute a weight from 1 to 3 on the word he choose in his lexicon(part1) or we established a range value to attribute a certain weight depending on the final score of a word(part2). Then once the word are sort to the right category we send the result into a text file with their score, it isnt the same score as the one i presented this one is calculated. So the project is composed of two parts. The first one is the “training” part of the SAE creating a programm that is able to do what i talked about previously with manually created lexicon. Secondly it was the learning part, quite the opposite of the part 1. What we had to do was creating an algorithm that this time makes almost everything itself such as taking the news and making his own lexicon based on score and weight that he attributes to the word alone, to cut a long story short, we had to automate everything.

Focus on what we did / didn’t do

part1

This SAE is divided into 2 parts the first part was to code several small methods/procedures which will be used in part 2 to create a machine learning program on this first part here are the coded methods:

-**new class <PaireChaineEntier>**, we had to create the class as we do usually, with the private attributes, constructor, getters, and setters.

-**initLexique**: this procedure allows you to initialize the lexicon variable with words and an associated score from a .txt file

-**integerForString**: this method (dichotomous version) returns an integer here in the context of the SAE the score associated with the word

-**score** : this method returns the sum of the scores of each word contained in a dispatch

-**maxchain**: this method returns the word with the highest score in a list

-**indexForString**: this method (dichotomous version) returns the index corresponding to the position of a word entered in a list and if this word does not exist in the list returns -1

-**average**: returns an average

-**classificationDepeches**: this method was the most complicated/long/hard to carry out, it returns a file which contains 500 lines with for each line the ID of a dispatch with that most representative category for this category then 6 other lines with their validity percentages (the one whose initial categories have not been modified) and finally a last line for the average of the percentages of the categories

In the first part of the project we had to manually create the lexicon file, this was the opposite in the part 2. The main goal was to make the construction of the lexicon automatic as well as putting the most relevant word into the right category.

To begin,

we needed to have all the word of a news that are associated to the same category thats exactly what the “initDico” method does, It would make things easier for us later because we will have only to sort the word that should be in another category.

Secondly, following the tought process of the second part of the SAE, the “calculScores” method was made to evaluate the pertinence of the word in a certain categoy. For each time the word is in the news, if he is in the lexicon and in the right category his score is up 1, if he isnt score goes down.

Then with the “poidsPourScore” function, based on the score of a word we create a range of value to assign them a “weight” from 1 to 3 a reflection of their relevance in a category, by exemple it could be like if the value is between 1 et 3 his weight is equal to 1 because he doesnt appear that much.

Finally, it was time to finalise the project. The last method is about everything we used this part, the function create a lexicon for a category. Then calculate the score of the word, based on that it determines the weight, all the data is given into a text file.

At the end of the coding part we tried to reduce the algorithm cost and the execution time of our method as well as changing the range value of the “poidScores” method to obtain more accurate results.

We achieved to do everything, except to add RSS feed, and the KNN comparaison to our code.

Presentation of the results

We obtained two differents results, as we were asked to do two differents things in both part of the SAE. We first used a classification of the dispatches with a self-made lexicon (with about 50 words per category chosen from the depeche.txt file) which gave us this result:

```
501 ENVIRONNEMENT-SCIENCES: 72%
502 CULTURE: 58%
503 ECONOMIE: 58%
504 POLITIQUE: 65%
505 SPORTS: 73%
506 MOYENNE: 65.0%
```

The other classification, this time in the 2nd part of the SAE where the lexicon is no more created by us but by a method that based off the depeche.txt file assign alone a score and weight to rate the relevancy of a word into a certain category :

```
501 ENVIRONNEMENT-SCIENCES: 60%
502 CULTURE: 70%
503 ECONOMIE: 67%
504 POLITIQUE: 67%
505 SPORTS: 87%
506 MOYENNE: 70.0%
```

We can see a difference between these two results. The 2nd one has a better average rate which is logical because the self-made lexicon is lacking precision and content (only 50 word) while the one generated automatically is therefore much more precise because it cover all the words in the file.

```
a-t-elle:1
a-t-il:1
abord:1
aboutira:1
accords:1
accueilli:1
accusée:1
accéder:1
acharné:1
acheter:1
action:3
active:1
actives:1
actrice:1
administratif:1
administration:1
adoré:1
adressé:1
adversaire:1
agglomération:3
agir:1
agissant:1
airy:3
aix-en-provence:1
alarment:1
```

This is an example of our lexicon files generated with the method “initDico” and “generationLexique” we can see that the file is sorted alphabetically.

```
time generation file ResponsePart1.txt classification with handmade lexicon: 68 ms
lexicon file generation time:769 ms (call 5 times)
generation time file ResponsePart2.txt classification with lexicon generated by method (generationLexique):58 ms
```

Here is the execution time of our methods (generationLexique and ClassificationDepeche) we can see that the methods take less than 1s to execute. The time here depends on the number of comparisons, the size of the code, the size of the Arraylist, and the way we search in it, as well as the speed of the processor in the PC (result obtained on a laptop).

Complexity analysis

In this part we will study the number of comparisons of two methods “score” and “calculScore”. The “calculScore” method is of the form (n), n = number of comparaison. It uses a lot of comparaison because of the number of other method it calls and the complexity of the method, for the “calculScore” method we did 3 versions, the first version is composed of 3 fulls courses with 3 for loop to travel the whole Arraylist which wasnt really satisfying algorithm-cost wise i would say it was terrible, with as you can see below 26 million comparaison. we did the easiest way but not the most efficient one and here is without dichotomous search.

```
for (PaireChaineEntier motDico : dictionnaire) {
        for (Depeche depeche : depeches) {
            for (String motDepeche : depeche.getMots()) {
```

```
Nombre de comparaisons avec la méthode calculScore: 27458491
Nombre de comparaisons avec la méthode calculScore: 22143477
Nombre de comparaisons avec la méthode calculScore: 23744542
Nombre de comparaisons avec la méthode calculScore: 25947708
Nombre de comparaisons avec la méthode calculScore: 25292668
```

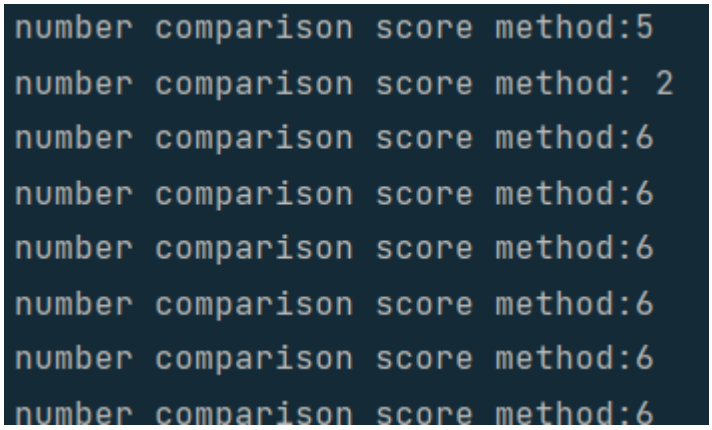
For the second version we were like “at least we need to remove one for loop” because when you put for loop inside other for loop the number of comparaison exposes, it basically travel and compares every element of the first Arraylist for every element of the next Arraylist ect. We removed one loop and we used the “indicePourChaine” method which is coded dichotomously (log(n) n = number of word in lexicon)*number of news * number of word in news, changed the other method that “calculScore” uses to make it as efficient as possible and less costly algorithmically speaking. The code was really more efficient 18k comparaison against 26M before. But was a bit longer to execute.

```
Nombre de comparaisons avec la méthode calculScore: 18203
Nombre de comparaisons avec la méthode calculScore: 18203
Nombre de comparaisons avec la méthode calculScore: 18203
Nombre de comparaisons avec la méthode calculScore: 18203
Nombre de comparaisons avec la méthode calculScore: 18203
```

So we tried to make it faster, we did it faster but the numbers of comparaison went up. So we preferred to keep V2 which performs less comparison than the others even tought its not the fastest one.

```
Nombre de comparaisons avec la méthode calculScore: 30415
Nombre de comparaisons avec la méthode calculScore: 29458
Nombre de comparaisons avec la méthode calculScore: 30057
Nombre de comparaisons avec la méthode calculScore: 30644
Nombre de comparaisons avec la méthode calculScore: 30604
```

In comparaison the method “score” does way less comparaison than calculScores because it's simply, the number of words in the lexicon divided by log(n) “n” is number of word in lexicon multiplied by the number of words in a news. where for each element, you add the weight associated to the word. So the number of comparaison isnt a big amount it only depends on the number of word(element in the Arraylist) from a news.



Conclusion

In conclusion, we did our best to optimize the code of our classification it goes from using different way to travel an arraylist (dichotomous searching) try to remove useless comparaison. its to make sure you only do what is asked, and thats maybe one our biggest issues also im not sure we made the most efficient method everywhere so our code isnt the fastest one to execute, the improvement could come from this and from implementing the knn comparaison or learning new stuff on the internet that could be useful for us. To summarise, our code can be surely better in many different ways as i said. But the code works and the optimization is not that bad.