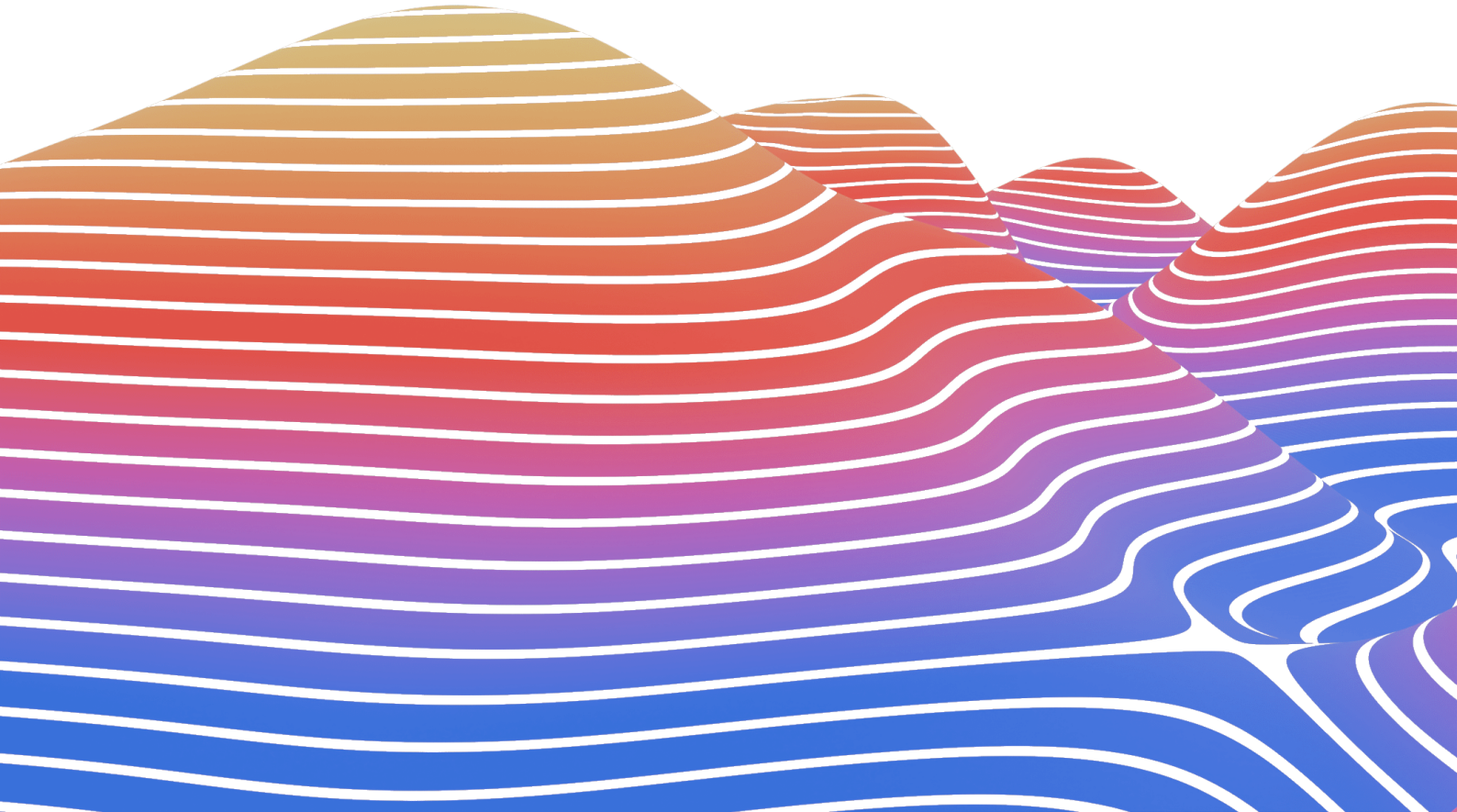
BUT1 INFO - Groupe A1

SAÉ 1.01
Report

Manu Thuillier
Lucas Maillet



Contents

1 Introduction

As a remainder, the goal of this SAÉ was to create a program able to classify given news into those categories:

- **Environment/Science**
- **Economy**
- **Sports**
- **Culture**
- **Politics**

In order to do so, we implemented a supervised automatic learning algorithm that gives a weight (*ranging from 1 to 3*) to each word found in the news of a category (those wordlists will be called **glossaries** during this report).

We, at first, manually wrote those glossaries by arbitrary choosing meaningful words. Secondly, they were automatically created using a method that calculates a word's score based on its frequency in its category and other categories (similar to *TF-IDF*) from a training dataset (depeches.txt). It then normalizes those scores into weights and save those glossaries into several files.

The program then uses those glossaries to score an unclassified news and find its best-fitting category (the one where it gets the highest score).

Throughout the development of this project, we worked not only on the asked implementation previously described, but also on some variations to improve both the accuracy and the processing time of our program. All these steps are detailed further in this document.

2 What did we do?

2.1 Manually written glossaries

For each category, we read certain news and selected words that appeared to be the most recurrent. We assigned a weight to each of these chosen words.

This initial step allowed us to test the first part of the SAÉ without automatically generating glossaries, as we later do.

For example, the first 5 words of the `SPORTS` category are :

```
sport:3
sportif:3
tour:3
partie:3
leader:3
```

Fig. 1: The 5 first lines of the sports.txt file

2.2 Glossary parsing

The first thing we did was implementing the `initLexique` procedure in the `Category` class (managing the glossary files reading). Glossary files have a certain format as mentioned in the instructions, so we used the way of reading files from the `lectureDepeches` procedure in the `Classification` class and adjusted it to read the glossary files format correctly.

After writing the code, we tested it by printing the words from the glossary of the `SPORTS` category.

```
SPORTS [(sport, 3), (sportif, 3), (tour, 3), (partie, 3), (leader, 3), ...]
```

Fig. 2: Output of the program, printing the glossary of the `SPORTS` category

These five words are the same as in the Figure 1, showing that our procedure is working properly.

2.3 News' score calculation for a category

The score of a news in a category is the sum of the weights assigned to each word based on the category's glossary.

In order to implement this, we, at first, needed a function to find the weight of a word in a category's glossary, that is why we implemented the `entierPourChaine` function of the `UtilitairePaireEntierChaine` class. To test this function, we made our program ask for a word, and then print the weight of it in the `SPORTS` category.

```
Choisir un mot : joueur
Le poids du mot joueur est 3 dans SPORTS.
```

Fig. 3: Output of the program when the user writes the word "joueur"

Then, we needed a function to get the sum of the weights of the words in a specific news, so we implemented the `score` function in the `Category` class.
To test this function, we made our program print a specific news, print every word of its content with a weight greater than 0 in the `SPORTS` category, and then the score calculated with the `score` function:

```
---------------------
depêche 463
date:200308
catégorie:SPORTS
Franck Ribéry, nouvelle idole du foot allemand
Depuis son arrivée à Munich, l'attaquant français
séduit. Rencontre.

---------------------
Mots ayant un poids supérieur à 0 :
- foot 3
Cette dépêche a un score de 3.
```

Fig. 4: Output of the program, showing the 463rd news, its word with a weight > 0 and its score for the `SPORTS` category

2.4 Classification of the news into categories

To classify a news between our categories, we need to:

1. calculate its score in each category,
2. choose the category where it gets the highest score.

We classify a news into the category where it scores the most. We implemented the `chaineMax` function in the `UtilitairePaireChaineEntier` class to do this.

```
La dépêche numéro 463 a
- un score de 3 dans la catégorie SPORTS
- un score de 0 dans la catégorie CULTURE
- un score de 0 dans la catégorie ECONOMIE
- un score de 0 dans la catégorie POLITIQUE
- un score de 0 dans la catégorie ENVIRONNEMENT-SCIENCES
La catégorie ayant le score maximal est SPORTS.
```

Fig. 5: Output of the program, showing the score of the 463rd news in each category, and the category where it gets the highest score (`SPORTS`) using `chaineMax`

Then, to calculate the percentage of correct answers for each category, we need to calculate the number of correct answers per category and divide it by the number of news in this category. To find the global

percentage of right answers, we just need to calculate the sum of the correct answers in all categories and divide the result by the total number of news.

We implemented the `classementDepeches` procedure in the `Classification` class to :

1. classify each news to the category where it gets the highest score,
2. calculate the percentage of correct answers for each category,
3. calculate the global percentage of correct answers,
4. save the results in the given format in a text file.

After testing, our procedure successfully generated a file in the required format. Here's an example of result produced by the procedure:

```
001:ENVIRONNEMENT-SCIENCES
...
500:POLITIQUE
SPORTS:88,0%
CULTURE:59,0%
ECONOMIE:68,0%
POLITIQUE:78,0%
ENVIRONNEMENT-SCIENCES:60,0%
MOYENNE:70,6%
```

Fig. 6: Content of the file generated by `classementDepeches`

2.5 Automatically created glossaries

To automatically create a glossary for a category from its various news articles, we simply need

1. for each word in the category's article, to calculate its score based on its frequency (*e.g. increment the score by 1 if we find it in the category, and decrement it by 1 if we find it in another category*),
2. to determine a weight with its score.

We implemented `initDico` in the `Classification` class because to create a list of the different words in the news articles of a category, then we implemented `calculScores` to calculate the score of a word based on its frequency in the news of the category.

At this point, we had to think of a way to assign a weight based on a score, and we decided to give a weight of :

- 3 for the words with a score greater than ($>$) 10,
- 2 for the words with a score $\leqslant$ 10 but greater than ($>$) 0,
- 1 for all the remaining words.

Finally, we had to implement the `generationLexique` procedure to generate the list of words, give them scores and then weights for each category, and finally write all of this in a file. Here's an example of a glossary generated by this procedure:

```
l:1
art:2
lyrique:2
s:1
invite:1
sur:1
arte:2
...
théâtre:3
```

Fig. 7: Content of the `culture.txt` file generated by `generationLexique`

```
001:ENVIRONNEMENT-SCIENCES
...
500:SPORTS
SPORTS:100.0%
CULTURE:99.0%
ECONOMIE:95.0%
POLITIQUE:98.0%
ENVIRONNEMENT-SCIENCES:98.0%
MOYENNE:98.0%
```

Fig. 8: Content of the answer file using auto-generated glossaries

2.6 *FranceInfo* and *Le Monde*'s RSS feeds news addition

To obtain more accurate results, we decided to add recent news articles from *FranceInfo* and *Le Monde* to the ones we are already using. We downloaded several files by theme from their RSS feeds and created a Python script to convert them into the format of a news in our system.

Here is the script for *FranceInfo* and *Le Monde*'s RSS feeds and the generated text we added to a new `nouvelles-depeches.txt` file:

```python
class NewsParser:
    id: int
    content: StringIO
    ...
    def read(self, category: str, *sources: str):
        for src in sources:
            tree = parse_xml(src)
            root = tree.getroot()
            for child in root[0]:
                if (child.tag == "item"):
                    date = parse(
                        child.find("pubDate")
                            .text
                    ).strftime("%d%m%y")
                    self.content.write(f".N {self.id}\n")
                    self.content.write(f".D {date}\n")
                    self.content.write(f".C {category.upper()}\n")
                    self.content.write(f".T {child[0].text}\n\n")
                    self.id += 1
...
```

Fig. 9: Part of the script converting the RSS feeds news into the desired format

```
.N 501
.D 160124
.C CULTURE
.T Cinéma : "Godzilla Minus One", un retour
    aux sources réussi pour la saga

.N 502
.D 160124
.C CULTURE
.T REPORTAGE. "Les lieux-dits, ça fait
    partie de notre identité" : les
    habitants des petites communes
    contrariés par l'obligation d'adressage


...


.N 504
.D 160124
.C CULTURE
.T Hausse des tarifs du Louvre : le musée
    vise "la grande masse de visiteurs é
    trangers", estime l'économiste
    Françoise Benhamou
```

Fig. 10: Part of the text generated by the script

Results of this modification are available in section 3.1.

2.7 Plural 's' trim & special character filter

To make the results even better, 2 extra conditions have been added while creating our automatic glossary:

- if a word contains non-alphabetical characters or contains less than 2 characters, the word is not inserted into the glossary,
- if a word ends with 's' (as a mark of plural form), the final 's' is trimmed and then the word is inserted into the glossary.

For example,

- '14e', '12', 'a', or ':' would not be inserted into the glossary,
- 'plats' would be inserted as 'plat',
- 'plat' would still be inserted as 'plat',
- 'dessous' would be inserted as 'dessou' (*not a correct word, but would not impact the classification*).

This helps us to filter words with special characters and not to have the same word repeated in both its singular and plural forms with different weights.

Results of this modification are available in section 3.1.

2.8 Processing time improvement

After attempting to increase result accuracy, we tried to reduce the processing time of the glossary generation.

To do this, we sort the glossary words during initialization (in `initDico`): for each new word, we determine at which index to add it to the glossary that is being created using a binary search and we replace every sequential search we implemented before by a binary search implementation.

Results of this modification are available in section 3.1.

2.9 k-NN implementation

In order to increase the success rate, two implementation were made around the k-NN method, which unlike our previous implementations don't choose the result based on the best score of categories but on the "proximity" with previously classified news.

2.9.1 By content proximity

A first simple implementation was made based on what was asked in the document, which said that in order to compare two news we will base our proximity on their content by counting the commonalities, such as the more there are commonalities the nearest are the news.

In the end, the results were even worse than our very first basic implementation (see section 3.2).

2.9.2 By spacial proximity

For a more complex implementation, we base our proximity on a distance between two news. But to do so, we first needed to represent our news into a space, which is why we added a vector attribute to each news, whose dimensions corresponds to categories and values to the score of said categories. Then to avoid rounding error, we preferred using the Manhattan distance formula to the Euclidean one to compare the news.

Even if the results are significantly better than the content-proximity version (see section 3.2), we did not gain much compared to our first basic implementation.

3 Results

3.1 Accuracy with score

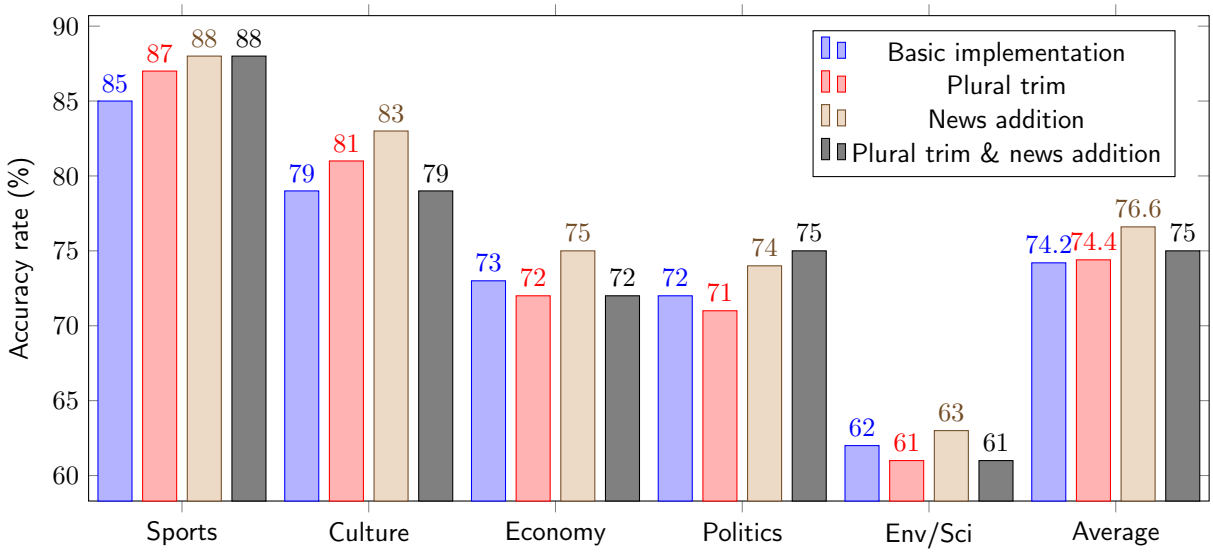


Fig. 11: Accuracy rates during classification with `test.txt`

In the Sports category, the results are more accurate compared to other categories. But there are also more false-positive results classified in this category, thus we deduce that it is just the most likely to came out no matter what the right category is. This is also because Sports have specific words mostly used in that

category. On the other hand, in the Env/Sci category, the vocabulary is more general, with words used in other categories, making it harder to classify in this category accurately.

Overall, adding news from *FranceInfo* and *Le Monde*, and using the plural trim, makes the system more accurate.

3.2 Accuracy with k-NN

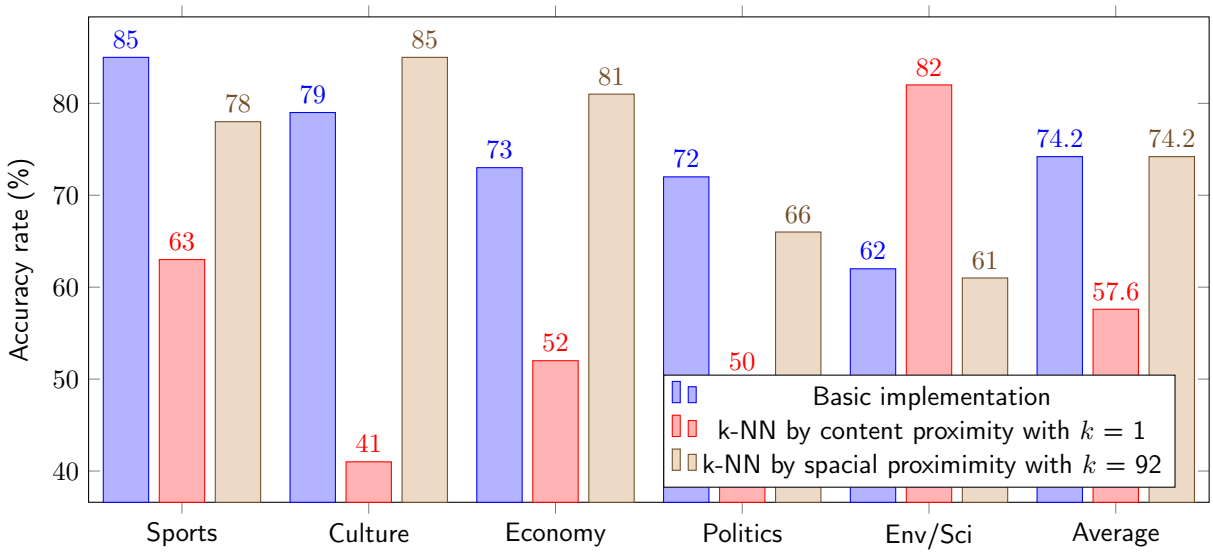


Fig. 12: Accuracy rates during classification using k-NN with `test.txt`

Other than the fact that the k-NN algorithm is significantly slower than our previous implementation (due to more comparison with pre-classified news), this implementation was not even really that more successful, at best equal. And thus without counting the fact that we add to determine the best number of k neighbors.

3.3 Processing time

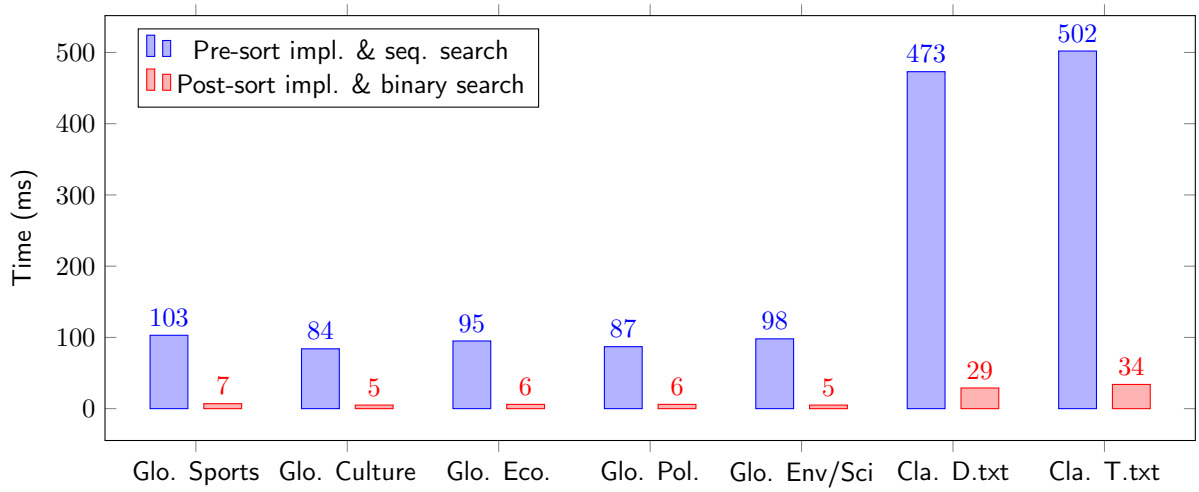


Fig. 13: Processing times during glossaries generation and classification with `depeches.txt` and `test.txt`

When creating glossaries and classificating, sorting the glossaries during their creation and using binary searches instead of sequential searches results in an processing time that is approximately 20 times faster.

4 Complexity analysis

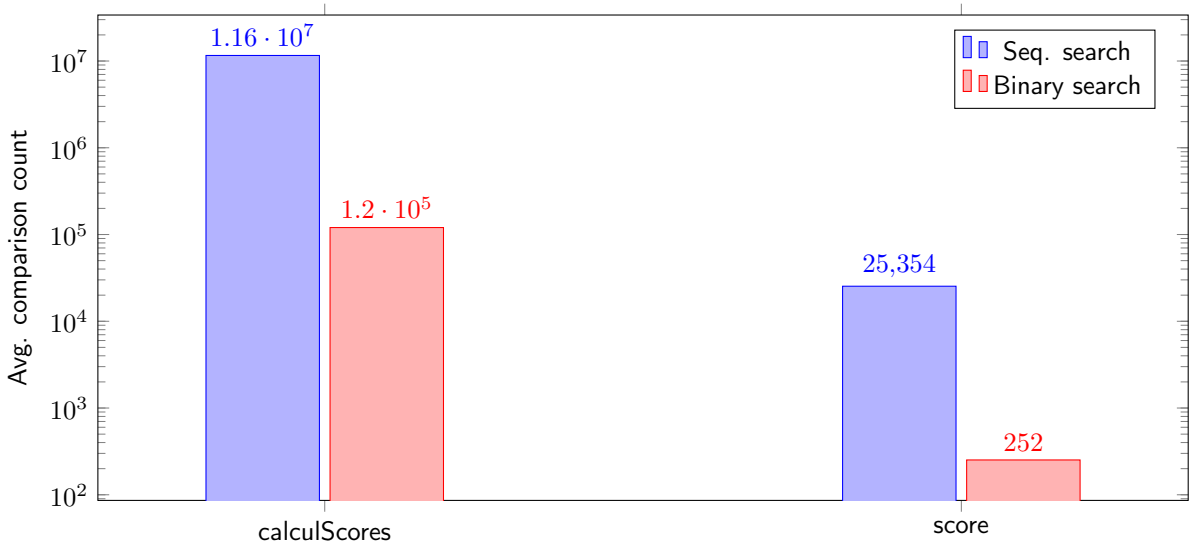


Fig. 14: Avg. count of comparison between 2 elements of a vector using `score` and `calculScores` methods

Using a binary search results in around 100 times less comparisons than a sequential search.

4.1 `score`

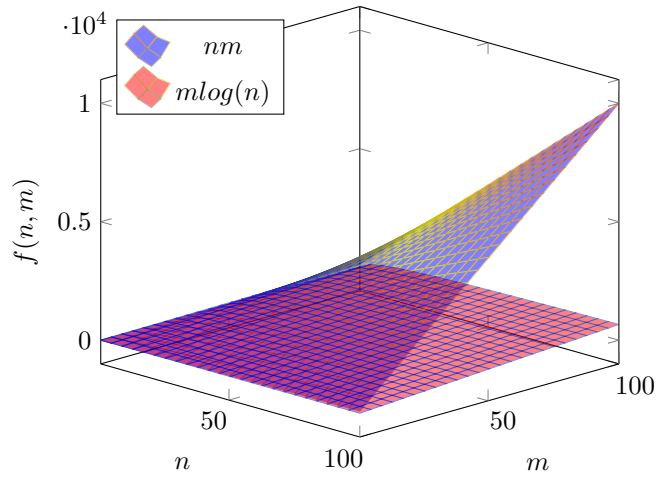


Fig. 15: Comparison of worth case complexity of the `score` function using seq. & binary search

In the `score` function, we're searching in the glossary of a category the weight of every word of a news.

As we're searching for every word of a news, the complexity of the `score` procedure is the complexity of the used search algorithm multiplied by the number of words in the news.

With n the number of words in the glossary, and m the number of words in the news:

- Using binary search, the worst-case complexity of the `score` procedure is around $mlog(n)$,
- Using sequential search, the worst-case complexity of the `score` procedure is around mn.

4.2 `calculScores`

In the `calculScores` procedure, we're searching for the frequency of every word in every news in its category and other categories to calculate the score of a word.

With w the total number of words in every news, and n the number of words in the glossary of the category:

- Using binary search, the worst-case complexity of the `score` procedure is around $wlog(n)$,
- Using sequential search, the worst-case complexity of the `score` procedure is around wn.

The complexity of this procedure is more or less the same as the `score` procedure, we're just multiplying the complexity of the search algorithm we're using with the total number of words in every news instead of the number of words in a specific news.

5 How could we improve our classification system?

5.1 Word processing

A way to improve our results would be to change the way we represent the words to make them more meaningful. First by getting rid of useless words like pronouns, coordinating conjunctions, etc. Then by normalizing the remaining words by turning them into their canolized form to summarize their meaning.

For instance,

- the word "`sportif`" would be canolized into "`sport`",
- the words "`étudiant`", "`étudier`" would be canolized into "`étude`",
- and so on...

5.2 Algorithm

Another way to enhance the results could be by changing the algorithm we use to classify our news.

First, we could change our scoring method to create our glossaries by using TF-IDF formula which is known to perform well in such case.

We could even change completely the algorithm by using a more efficient one, such as the Gaussian Naive Bayes classifier.