Rapport Projet 2

07 Avril 2023


Antonin MONTAGNE-21901206
Lou-Anne GAUTHERIE-22001251
Nathan SAKKRIOU-22003438
Yanis HABAREK-22010593

UNIVERSITÉ
CAEN
NORMANDIE

Table des matières

1 Introduction

1.1 Plan du rapport

Nous évoquerons d'abord quels étaient nos objectifs de départ (1.2), puis nous détaillerons les différentes étapes de la création de notre projet avec les rôles de chacuns (2). Ensuite nous présenterons l'architecture de ce projet (3), ainsi que les éléments techniques utilisées (4) dans notre code. Finalement nous présenterons certaines expérimentations (5) et terminerons par une courte conclusion (7).

1.2 Objectifs du projet

Nous avions pour but de créer une application d'analyse d'identité d'auteurs et détection de plagiat. Certaines contraintes nous étaient données :

- Créer un corpus d'entraînement et de test.
- Traiter ce corpus pour en obtenir une représentation appropriée.
- Mettre en oeuvre un ou plusieurs classificateurs.
- Évaluater les performances des classificateurs par rapport au rappel et à la précision.
- Intégrer des caractéristiques supplémentaires dans le processus.

2 Fonctionnalités implémentées

2.1 Description des fonctionnalités

Cette application possède une page web en tant qu'interface graphique. Cette page web est chargée via une API pour que les documents communiquent directement entre eux et que l'analyse de plagiat charge en temps réel.

Il suffit de lancer l'API dans un terminal, et le terminal donne l'adresse du serveur à rejoindre o

Sur cette page web se trouve une fonctionnalité pour comparer le document du client, avec ceux du corpus. Le client peut d'abord découvrir les similarités entre son document et les autres (mots revenants fréquemment), puis il peut ensuite voir les auteurs de ces documents.

La deuxième fonctionnalité de la page est la comparaison de deux documents. Le client choisit les deux documents à comparer, mais également l'algorithme utilisé.
Une fois la sélection effectuée, les données sont envoyées à l'API, qui charge les fonctions correspondantes.

2.2 Organisation du projet

Pour commencer, Nathan s'est occupé de créer tout le corpus d'article en scrappant les archives de Libération. Il s'est également occupé de la création d'un interface permettant de manipuler les articles stocké dans notre base de donnée et pour finir il s'est occupé de l'implémentation des algorithmes avec l'API.

Pendant ce temps, Yanis et Antonin se sont occupés des algorithmes. Antonin a codé l'algorithme de classification *Bayes* et Yanis les algorithmes de comparaison *Jaccard*, *Difflib* et *Vecteur Cosinus*.

Dans le fichier `Bayes.py`, nous retrouvons une classe `Bayes` qui contient plusieurs fonctions pour manipuler le contenu d'un fichier :

- `countClass` compte le nombre de fois où une classe apparait dans le corpus.
- `countOccurenceWordsClasse` compte le nombre de fois où un mot apparait dans une classe.
- `countWordClasse` compte le nombre de mots dans un classe.
- `countWordDifferent` compte le nombre de mots différents dans le corpus.
- `chooseClass` calcule l'auteur qui est le plus susceptible d'avoir écrit le texte.
- `recall` calcule l'efficacité du classificateur.
- `precision` calcule la précision du classificateur.

Le fichier `difflib_algo.py` contient la classe `Difflib` qui comprend :

- la fonction `compare` qui prend en argument deux fichiers .txt et qui les compare grâce à la classe `difflib` de python.

Le fichier `jaccard.py` contient la classe `Jaccard` qui comprend :

— la fonction `compare` qui compare cette fois ci deux fichiers .txt en utilisant leur taille, leur intersection et leur union.

Le fichier `techniques_vecteurs_cosinus.py` contient la classe `VecteurCos` qui comprend :

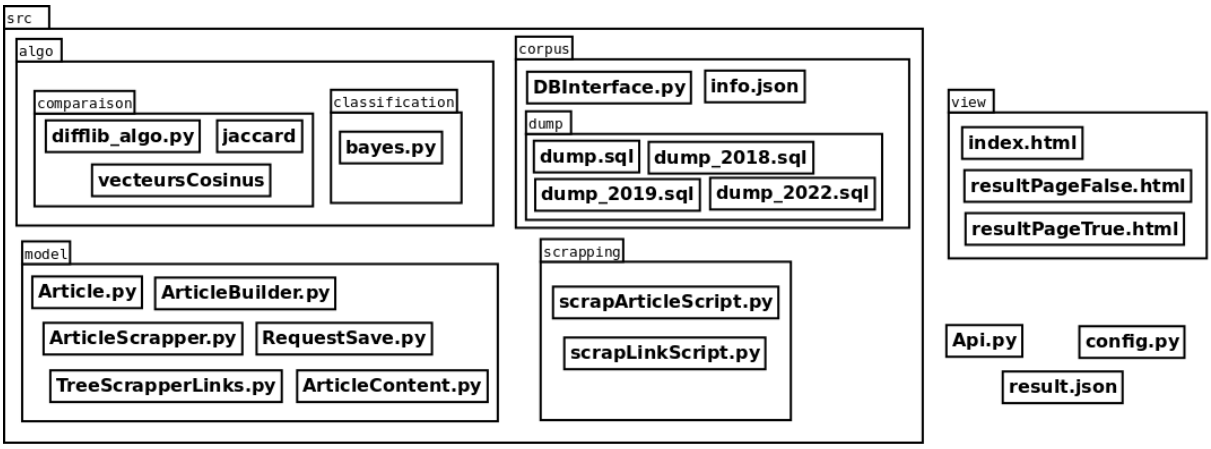
— La fonction `cosine_similarity` qui prend en argument deux string, elle créée ensuite un vecteur de mots pour chaque texte puis utilise la fonction `set` de python pour supprimer les doublons.

De son coté, Lou-Anne a créé l'API (codé en python) et la page web (html, css, js). Elle a implémentée des éléments `input-file` qui ouvre un explorateur de fichiers pour pouvoir choisir un fichier, et des éléments `button` et `select` pour sélectionner la comparaison et l'algorithme voulu.
Ces données sont stockées dans des dictionnaires `dict1` et `dict2` (codé en javascript) et ceux-ci sont envoyés à l'API via la méthode `post`. C'est ensuite cette même API qui gère les données.

3 Architecture du projet

3.1 Diagrammes des modules et des classes

Diagramme de tous les packages (et leurs dépendances) :



3.2 Paquetages utilisées

Nous avons utilisé plusieurs classes de Python nécessaires au lancement de l'application et à l'analyse de documents :

noms des classes	utilité	emplacement
fastapi	framework permettant de simplifier la création d'api	Api.py
psycopg2	intéraction avec base de données PostgreSQL	DBInterface.py
path	manipulation des chemins de fichiers et de répertoires	scrapArticleScript.py scrapLinkScript.py
sys	accès à des fonctionnalités spécifiques au système d'exploitation	scrapArticleScript.py scrapLinkScript.py Bayes.py
math	fournit des fonctions mathématiques	Bayes.py
os	accès à des fonctionnalités pour interagir avec le système d'exploitation sous-jacent	difflib _algo.py
difflib	fournit des fonctions pour comparer et différencier des séquences de texte	difflib_algo.py

TABLE 1 – Liste des classes Python utilisées

4 Éléments techniques

4.1 Corpus d'article

Le corpus est une partie très importante de notre projet car c'est grâce aux informations qu'il contient que l'algorithme de classification (bayes dans notre cas) va pouvoir attribuer le texte fourni en entrée a tel ou tel auteur.

4.1.1 Structure du corpus

Comment ce matérialise t'il : Nous avons fait le choix d'utiliser une base de donnée relationnelle, PostgreSQL. Cette base de donnée contient 2 tables :

— researched_table : qui va nous permettre de stocker les urls des articles que nous voulons scrapper (inutile dans l'utilisation finale mais

très importante pour la création du corpus). | *id SERIAL, year VARCHAR, month VARCHAR, day VARCHAR, url VARCHAR*
— articles_table : qui va nous permettre de stocker le contenu des articles et leurs auteurs. C'est les informations de cette table qui sera utilisé par notre algorithme de classification. | *id SERIAL, author TEXT, text_article TEXT*

4.1.2 Création du corpus

La création de notre corpus se passe en 2 étapes :

— 1 : Récupération d'articles
— 2 : Scrapping du contenu de ces articles et de leurs auteurs

Étape 1 :

Pour remplir notre corpus, nous avons fait un choix dès le début de notre projet.
Il nous fallait des articles et les auteurs associés en quantité de préférence. Pour trouver ce genre d'informations nous nous somme orienté vers des sites de journaux, dans notre cas Libération a été notre premier choix. Libération possède une section archive qui contient les articles paru de 1998 à 2023. Cette section archive est composé de manière très simple. Pour accéder aux articles paru le 23 Mai 2009 (un exemple au hasard), le site nous offre l'URL suivante https ://www.liberation.fr/archives/2009/05/23/. On se rend compte qu'il très facile d'itérer sur toutes les urls. Il suffit simplement de connaître le calendrier des jours que l'on souhaite. Cette information est facilement accessible grâce a la bibliothèque Python *calendar* et la fonction *monthrange*. Grâce a cette fonction nous pouvons savoir combien de jour il y avait en fonction d'une année et d'un mois donné. La génération des URL est théoriquement complétée, en pratique, nous avons crée 4 types d'objet : *LiberationScrapper*, *Year*, *Month* et *Day* :

— LiberationScrapper est la classe permettant de lancer le processus de création des URL sur une tranche d'année afin d'automatiser au mieux notre processus.

— Year, Month et Day permettent de générer les valeurs correspondante

et des les concaténer à la base de l'URL de Libération. La spécificité de Day, c'est qu'il lance le début du scrapping.

Nous n'avons pas encore précisé ce que nous renvoyais une URL du type : https ://www.liberation.fr/archives/2009/05/23/. Nous obtenons une liste de lien menant aux articles voulu. Avec les bibliothèques *BeautifulSoup (bs4)* et *Request* il est facile de faire une requête GET pour obtenir la page avec le lien qu'on a créer et de manipuler son DOM. La méthode de la classe Day qui s'occuper de cela est *scrapLiberation(url)*.
Une fois tous les liens des articles du jour voulu récupéré, nous les envoyons dans la base de donnée grâce un objet nommé *ScrapDatabase* qui nous permet de faire l'interface entre notre code et notre PostgreSQL, cette classe possède toutes les méthodes nécessaires pour l'insertion de donnée dans chacune de nos tables mais aussi pour récupérer et utiliser les informations nécessaires pour la classificateur (nous utilisons la bibliothèque *psycopg2* pour faciliter la connexion avec une instance postgresql en python).

Pour résumer cette étape 1 :

— 1 : Nous avons détecter un pattern simplement utilisable dans les archives de Libération pour accéder a une grosse quantité d'article *https ://www.liberation.fr/archives/annee/mois/jour*

— 2 : Nous avons créer des classes afin d'automatiser la création de ces URL

— 3 : Nous scrappons les liens des articles et les stockons dans une table de notre corpus.

Le script qui nous permet d'effectuer toutes ces actions est *scrapLinkScript.py*

Étape 2 :

Nous avons désormais notre table *researched_ table* remplie d'une liste d'URL correspondant a des articles. L'objectif maintenant est de les récupérer, les lire et récupérer le contenu de l'article et l'auteur pour ensuite le stocker dans

le corpus.

Pour cela nous introduisons quelques nouveaux objets :

— Article : Représente un Article avec sa date et son URL

— ArticleContent : Représente un article concret avec son auteur et son contenu

— ArticleBuilder : Un builder permettant de transformer les données brutes (sous forme de liste de liste de string) récupérés de la base de donnée en Article

— ArticleScrapper : Permet de scrapper l'URL d'un Article et de le transformer en ArticleContent

Toutes ces classes sont manipulés dans le script *scrapArticleScript.py* qui permet de sélectionner une année et de scrapper tous les articles correspondant pour ensuite insérer les informations récupérés dans la bonne table du corpus.

Le déroulement de cette étape est le suivant :

— 1 : Récupérer une année précise, (dans notre cas nous avons fait le choix de ne récupérer que les liens d'articles de 2018, 2019, 2021 et 2022) grâce au passage de paramètre d'un script python.

— 2 : Récupérer les URLs d'articles correspondant dans le corpus et les transformer en une liste d'Articles dans notre code.

— 3 : Itérer sur cette liste d'Articles et scrapper leur contenu. Il existe une petite subtilité dans cette étape. Certains articles ne sont pas complètement accessible gratuitement, il nous a fallu détecter lesquels l'était afin de ne pas les stocker dans le corpus. Pour cela nous avons detecter la balise CSS qui correspondait a la bannière représentant le besoin d'être premium pour lire cet article et si elle existait sur cette page, alors nous générons un ArticleContent d'erreur, avec des valeur facilement reconnaissable, qui seront filtré avant d'intégrer le corpus.

— 4 : Une fois avoir scrapper l'ensemble des URLs, nous avons a notre dispositions une liste d'ArticleContent, que nous venons insérer dans la base de donnée (avec l'interface ScrapDatabase) et filtrant les ArticleContent d'erreur.

A partir de ce moment, il ne suffit plus qu'a scrapper l'intégralité des liens d'articles de notre table articles_table afin d'obtenir notre corpus.

Nous avons rencontré certains problème, comme l'encodage de certains caractere qui ne correspondait pas forcément a ceux accepté par notre base de donnée. Nous avons réussi a surmonter ce problème en faisant en sorte de remplacer les caracteres problèmatique par leurs homologues ascii mais également en modifiant l'encodage de notre corpus.

4.2 API

Nous avons fait le choix de créer une API comme moyen d'interaction avec notre programme pour le rendre utilisable sans connaissance technique, et simplement. L'idée originale était de la déployer sur VPS, mais nous avons eu quelques problème pour l'accessibilité des ports sur certaines requêtes. Sur la forge, l'api est utilisable en local et parfaitement fonctionnelle.

Pour rentrer dans l'aspect nous avons une interface web permettant d'utiliser nos algorithmes de classification et comparaison. Un résultat nous est renvoyé. Pour rentrer cette fois dans la partie programmation nous retrouvons 3 endpoints :

— "/" [GET] : Pointe sur notre interface de choix d'algo.

— "verif" [POST] : Requête envoyé par l'utilsateur grâce a la page pointé par le endpoint "/", il nous envoyé les modalités voulu (algorithme, similarité, auteur, fichiers ...), avec ces informations, l'api effectue les instructions correspondantes et inscrit le résultat dans un fichier *result.json*.

— "/comparaison" [GET] : Affiche le résultat du la demande de l'utilisateur.

Nous avons utiliser le framework fastapi qui nous permet de facilement utiliser tous les élément pour créer une api également couplé a jinja2, un moteur de templating, qui nous permet de rendre nos page html dynamique en fonction des paramètres qu'on lui fourni à l'envoi du html.

4.3 Description des algorithmes

Algorithme 1 : BAYES classifie un ensemble d'observations selon des sacs de mots

Entrées : Un texte (par défaut vide), option (par défaut 0)
Sortie : L'auteur qui a le plus de probabilité d'avoir écrit le texte t

```
1  proba ← dictionnaire vide
2  pour auteur ← corpus faire
3  |  proba[auteur] = nbTexteAuteur/nbTexteCoropus
4  |  pour mots ← texte faire
5  |  |  si mots apparaît dans texteAuteur alors
6  |  |  |  proba[auteur]+ = log((nb de fois mots dans
   |  |  |    texteAuteur + 1)(nbMotsTextesAuteur +
   |  |  |    nbMotsCorpus))
7  |  |  fin
8  |  |  sinon
9  |  |  |  proba[auteur]+ =
   |  |  |    log(1/nbMotsTextesAuteur + nbMotsCorpus))
10 |  |  fin
11 |  fin
12 fin
13 si option == 1 alors
14 |  retourner max(proba, key = proba.get)
15 fin
16 retourner phrase + max(proba, key = proba.get)
```

Algorithme 2 : JACCARD compare deux fichiers grâce à la taille de leur unions, et de leur intersections

Entrées : 2 fichiers texte A et B

Sortie : Probabilité de similarité des fichiers A et B

1 $A = \text{set(A.split())}$

2 $B = \text{set(B.split())}$

3 **retourner** $\frac{|A \cap B|}{|A \cup B|}$

Algorithme 3 : VECTEUR COSINUS compare deux fichiers en les transformant en vecteur

Entrées : 2 fichiers texte A et B

Sortie : Probabilité de similarité des fichiers A et B

1 $vecteurA =$ fichier A transformé en vecteur

2 $vecteurB =$ fichier B transformé en vecteur

3 **retourner** $\frac{vecteurA.vecteurB}{||vecteurA||.||vecteurB||}$

Algorithme 4 : DIFFLIB est une classe Python qui a été importée et qui se base sur l'algorithme de Levenshtein.

1 $similarity = \text{difflib.}SequenceMatcher\text{(None,text1,text2).ratio()}$

```
doc_1 = "Data is the oil of the digital economy"
doc_2 = "Data is a new oil"

# Vector representation of the document
doc_1_vector = [1, 1, 1, 1, 0, 1, 1, 2]
doc_2_vector = [1, 0, 0, 1, 1, 0, 1, 0]
```

	data	digital	economy	is	new	of	oil	the
doc_1	1	1	1	1	0	1	1	2
doc_2	1	0	0	1	1	0	1	0

FIGURE 1 – cosinus

$$J(doc_1, doc_2) = \frac{\{'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy'\} \cap \{'data', 'is', 'a', 'new', 'oil'\}}{\{'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy'\} \cup \{'data', 'is', 'a', 'new', 'oil'\}}$$

$$= \frac{\{'data', 'is', 'new', 'oil'\}}{\{'data', 'a', 'of', 'is', 'economy', 'the', 'new', 'digital', 'oil'\}}$$

$$= \frac{4}{9} = 0.444$$

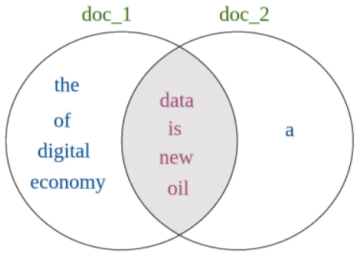


FIGURE 2 – jaccard

5 Expérimentations

5.1 Classification

Grâce aux méthodes recall et précision de la classe `Bayes`, nous avons lancé plusieurs fois le classificateur de Bayes en faisant varier la taille du corpus d'entraînement et la taille du corpus de test. Comme les résultats du classificateur peuvent varier d'une exécution à l'autre du fait que les corpus soit régénérer à chaque nouvelle instance de la classe Bayes. Nous avons décider de lancer l'algorithme 10 fois de suite avec les mêmes paramètres afin d'obtenir une moyenne. Voici les résultats obtenu :

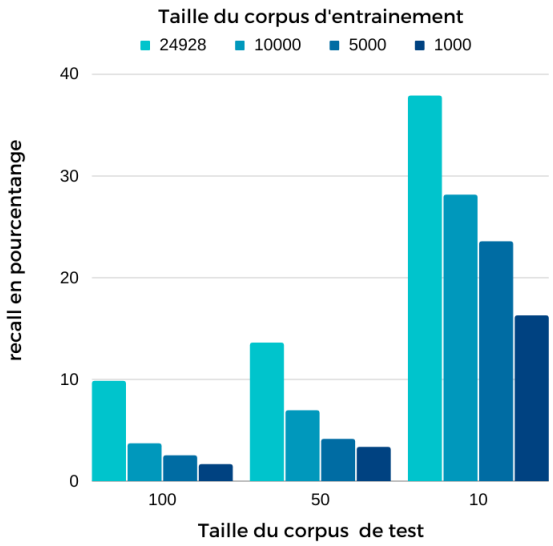


FIGURE 3 – Graphique de l'efficacité du classificateur

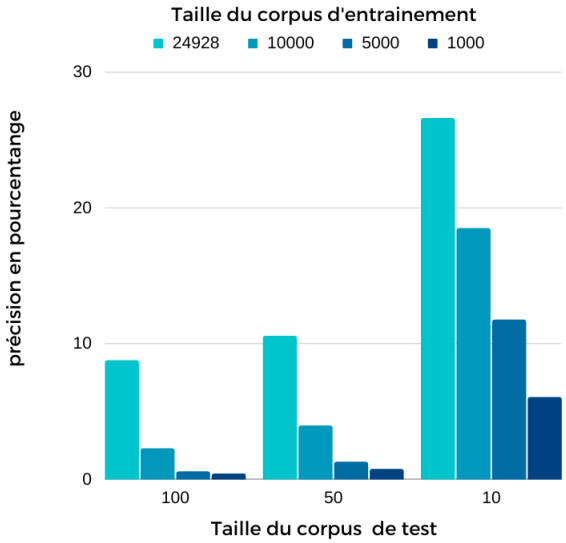
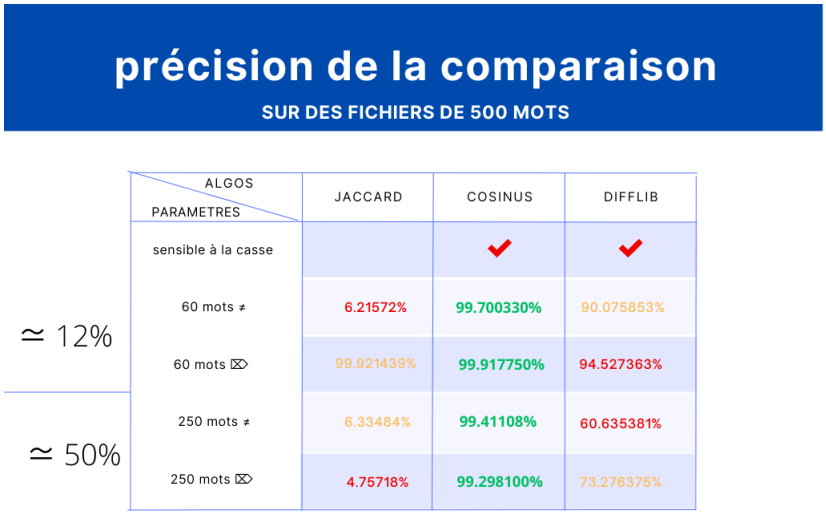


FIGURE 4 – Graphique de la précision du classificateur

5.2 Comparaison

Nous avons lancer la comparaison (jaccard, cosinus, et difflib) sur différents fichiers textes de différentes tailles (500, 1000 et 2000 mots), avec un nombre de mots retirés/remplacés (croissant).
Voici les résultats obtenu : (A noter que les résultats peuvent être très différents d'un fichier texte à un autre)



[t !]

FIGURE 5 – exécution sur des fichiers de 500 mots

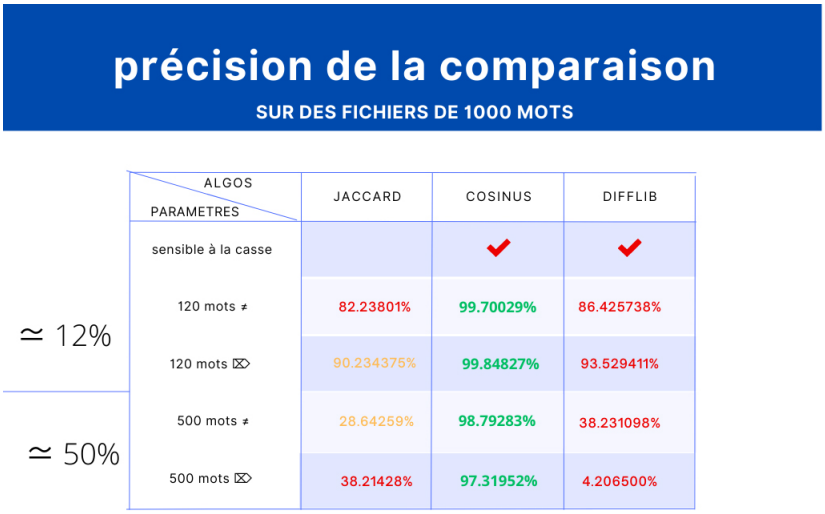


FIGURE 6 – exécution sur des fichiers de 1000 mots

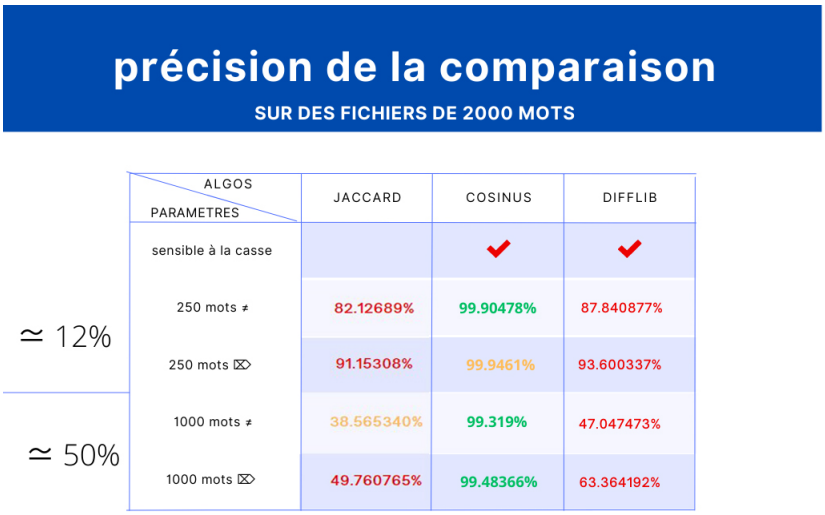


FIGURE 7 – exécution sur des fichiers de 2000 mots

6 Analyse des résultat

6.1 Classification

D'après les expérimentations faîtes sur le classificateur de bayes on peut en déduire deux choses :

— La première, on voit que la précision et l'efficacité augmente lorsque la taille du corpus de test diminue.
— La deuxième, on voit que la précision et l'efficacité diminue lorsque la taille du corpus d'entraînement diminue.

On peut donc en conclure que l'utilisation optimale du classificateur est avec un grand corpus d'entraînement et un petit corpus de test. De plus, plus les tailles des corpus d'entraînement et de test sont grandes, plus le classificateur mets du temps.

6.2 Comparaison

Il faut tout à bord savoir que les résultats ci dessus sont spécifiques au texte choisi, les résultats peuvent être différents à environ 6% selon les différents textes, cela est du notamment à la fréquence des éléments de chaque texte, mais aussi des mots utilisés. On peut tout de même en conclure certaines choses :

— La méthode de **Jaccard** est simple à calculer et est efficace pour des ensembles de données de petite à moyenne taille. Cependant, elle ne prend pas en compte la fréquence des éléments dans les ensembles, ce qui peut la rendre moins efficace pour des ensembles de données de grande taille ou pour des ensembles avec une forte variabilité de la fréquence des éléments.
— La méthode du **cosinus** retourne un résultat compris entre -1 et 1. Un score de similarité de 1 indique une correspondance parfaite entre les deux vecteurs de données, tandis qu'un score de similarité de -1 indique une complète différence entre les deux vecteurs.
Un score de similarité de 0 indique que les deux vecteurs sont orthogonaux et n'ont aucune similarité. En pratique, pour des ensembles de données textuelles, les scores de similarité obtenus avec la méthode

du cosinus sont souvent compris entre 0 et 1, car les vecteurs de mots sont rarement parfaitement opposés. Les scores de similarité proches de 0 indiquent que les documents sont peu similaires, tandis que les scores de similarité proches de 1 indiquent que les documents sont très similaires., Cette méthode prend en compte la fréquence des éléments dans les vecteurs et est plus efficace pour des ensembles de données de grande taille ou pour des ensembles avec une forte variabilité de la fréquence des éléments(ce qui n'est pas le cas pour nous qui testons sur max 1000 mots différents).

— La méthode de **Difflib** qui est une bibliothèque de Python3 qui se base sur l'algorithme de Levenshtein, peut être utile pour des tâches de comparaison de chaînes de caractères de petite ou moyenne taille, mais elle peut être moins efficace pour des séquences de grande taille ou pour des séquences avec une grande variabilité de la fréquence des éléments. Elle est également sensible à la casse et à la ponctuation, ce qui peut affecter la précision de la mesure de similarité.

7 Conclusion

Nous avons donc une application web d'analyse de plagiat avec expérimentations complètes, c'était bien l'objectif que nous nous étions fixés. Nous avons implémenté un classificateur fonctionnel ainsi que plusieurs algorithmes de comparaison et nous fait des expérimentations dessus afin de répondre à la problématique qui nous été posé. En améliorations, nous aurions pu implémenté un deuxième classificateur afin de pouvoir faire une comparaison avec celui de bayes.

8 Remerciements

Nous souhaitons adresser nos remerciements les plus sincères à Monsieur Marc Spaniol pour son aide apportée, grâce à qui nous avons pu mener à bien ce projet ainsi que le rapport associé.
Nous remercions également Monsieur Bonnet Gregory, pour les explications et directives données lors des cours magistraux.

A tous ces intervenants, nous présentons nos remerciements, notre respect et

notre gratitude.