Projet L3
Fouille de données
Ingénierie des langues

GOEHRY Martial
16711476

26 mai 2024

Table des matières

Introduction

Ce projet a pour but de développer un modèle permettant de catégoriser des emails en spam ou ham. La définition d'un spam dans le dictionnaire *Larousse* est :

> Courrier électronique non sollicité envoyé en grand nombre à des boîtes aux lettres électroniques ou à des forums, dans un but publicitaire ou commercial.

Il est possible d'ajouter à cette catégorie tous les mails indésirables comme les tentatives d'hameçonnage permettant de soutirer des informations personnelles à une cible.

L'objectif est de travailler uniquement sur les données textuelles issues du corps du mail. Nous avons donc comme point de départ les éléments suivants :

— langue : anglais
— corpus : monolingue écrit
— type : e-mail

Déroulé Le développement de ce projet s'articule autour de 3 phases majeures

— Phase 1 : Récupération des données (Fouille de données)
— Phase 2 : Analyse des caractéristiques (Traitement de langage)
— Phase 3 : Construction d'un modèle d'analyse (IA)

Phase 1 La phase 1 concerne la récolte des informations et les traitements minimums nécessaires pour la mise en base. Les objectifs de traitement de cette phase sont :

— Extraire les corps des mails et éliminer les méta-données superflues
— Éliminer les mails non anglais
— Éliminer les mails en doublons
— Éliminer les parties de textes non pertinentes (liens, réponses, certaines ponctuations)

Cette phase se termine avec la mise en base des documents dans une collection Mongo.

Phase 2 La phase 2 vise à extraire des caractéristiques des textes. Les techniques de traitement du langage devront permettre d'effectuer une vectorisation des documents.

Phase 3 La phase 3 regroupe toutes les opérations d'exploitation des données et vise à développer et à créer un modèle de classement des mails et d'en évaluer les performances.

Afin de conserver une certaine cohérence dans le déroulé entre les phases chaque étape est automatisée avec Python. Seule la récolte initiale des mails a été réalisée à la main.

La Figure 1 donne une vue synthétique des étapes du projet.

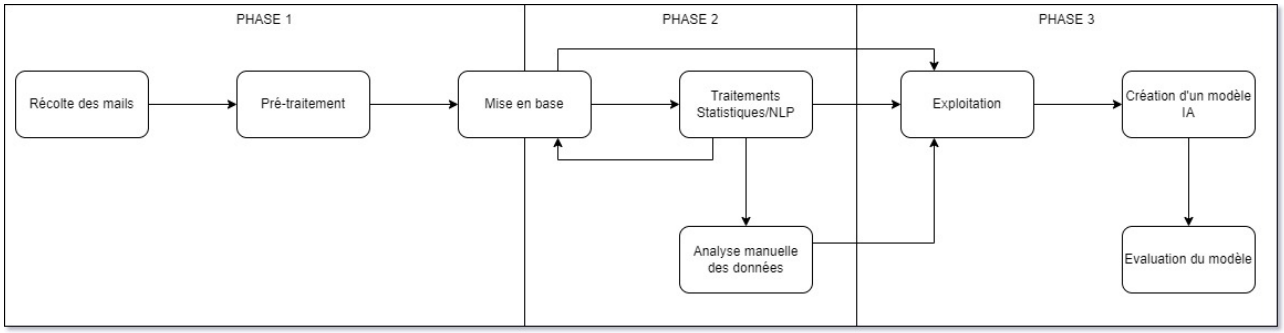

FIGURE 1 – Schéma des grandes étapes

Mise en place de l'infrastructure opérationnelle

L'architecture opérationnelle s'appuie sur des conteneurs docker. Plusieurs types de base de données sont mises en œuvre profiter des avantages de chacune. Les conteneurs peuvent être gérés à l'aide du fichier *Makefile* via les commandes suivantes :

— *make docker_start* : pour créer ou démarrer l'infrastructure
— *make docker_stop* : pour arrêter les conteneurs
— *make docker_prune* : pour nettoyer l'infrastructure

La figure 2 montre l'organisation de cette architecture ainsi que les documents nécessaires pour son montage.

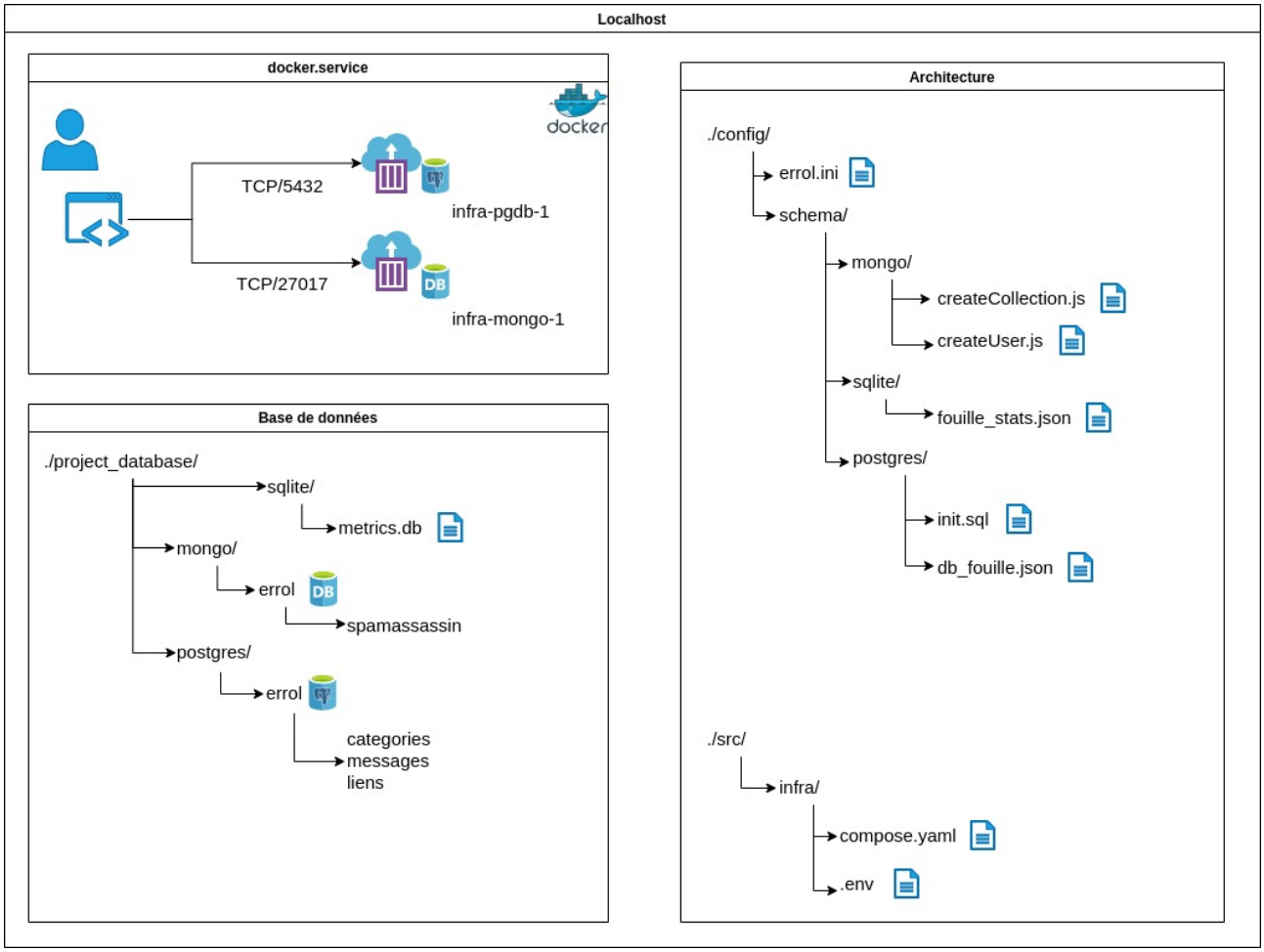

FIGURE 2 – Schéma de l'architecture Docker

Choix technologiques

Socle des services

Conteneurs La conteneurisation des services permet d'effectuer facilement une séparation entre les processus. Les ressources empruntées à la machine hôte sont réduites au strict besoin de l'application. Une fois le moteur installé les différentes applications ou services peuvent être déployé rapidement. Chaque application embarque dans son conteneur toutes ses dépendances. Par définition les conteneurs sont volatils, des configurations complémentaires doivent être mises en place pour permettre la persistence des informations des bases de données. Solution retenue. Le moteur *Docker* a été retenu, car il est simple d'utilisation, bien documenté et disponible sur

plusieurs systèmes d'exploitation. Il est possible d'utiliser la capacité *compose* de Docker pour déployer, démarrer ou arrêter plusieurs instances de manière coordonnée.

systemd Une solution susceptible de fonctionner aurait été d'installer directement sur ma machine les différents services requis. Cela étant, un long processus d'installation puis de désinstallation aurait été nécessaire, avec le risque d'omettre des composants. De plus l'ajout de service directement sur la machine hôte comporte toujours des risques d'isolation des processus. Solution non retenue.

Machine virtuelle L'installation des couches applicatives aurait pu être réalisé sur des machines virtuelles pour garantir une bonne isolation. Cela aurait également permit de simuler une infrastructure lourde avec plusieurs serveurs. Cependant, mon PC n'est pas en mesure de supporter l'exécution de plusieurs machines virtuelles en parallèle. Solution non retenue.

Base NoSQL Les bases de données NoSQL orientés documents sont plus performantes pour le stockage et l'accès à des ressources textuelles. Deux moteurs ont été testés :

MongoDB Moteur de base données flexible et raisonnable en utilisation de ressource. Le langage des requêtes est simple à prendre en main. L'utilisation avec Python est facilitée par le *Python developer path* disponible sur le site de l'éditeur. Solution retenue.

ElasticSearch Moteur puissant de base de données qui intègre un moteur Lucène pour la recherche de document par mot clé. L'interface graphique associée (Kibana) est agréable et facile à prendre en main. Néanmoins, ce moteur est très gourmand en ressource. Une fois l'index (schéma) créé, il est très compliqué de le modifier. Solution rejetée.

Base SQL Les bases de données SQL sont plus performante quand il s'agit de traiter des informations transactionnelles. Elles offrent également plus de garantie de sécurité des données que les bases de données NoSQL. Elles permettent aussi de faire plus facilement et rapidement des requêtes complexes avec jointure et agrégation. Pour ces raisons, ce type de moteur sera utilisé pour stocker les informations numériques générées à partir des textes.

SQLite Moteur de base de données SQL intégré avec Python. Les informations sont stockées dans un fichier défini. Cette base de données ne nécessite pas d'installer un service supplémentaire. Cette solution est généralement utilisée en phase de test. L'utilisation de cette solution pour stocker les données numériques des documents aurait été susceptible de générer des fichiers trop lourds et trop difficile à lier entre eux. Cependant, cette solution a été utilisée pour stocker les données générées lors de la phase de récolte (nombre de mails, nombre de mots uniques...). Solution retenue

Postgres SQL Moteur de base de données SQL solide et robuste. Elle permet également une gestion des utilisateurs ayant accès aux informations. L'intégration avec python est simple. Les données sont accessibles et peut être liées facilement. La taille des fichiers est gérée directement par le moteur. Solution retenue.

1 Fouille de données

Cette partie vise à expliquer les actions réalisées en vue de récupérer et stocker les données utilisées dans ce projet. Au cours de cette phase plusieurs traitements permettront de réduire la quantité d'information pour se concentrer sur les corps des mails. Les données MIME non textuelles sont écartés ainsi que les messages corrompus ou avec des encodages non convertibles en utf-8.

1.1 Récolte des données

Recherche de dataset Ma volonté initiale était de travailler sur des mails en français. Cependant, je n'ai pas trouvé de dataset dans cette langue. Je me suis donc retourné vers les dataset de mails en anglais.
J'ai pu alors récupérer deux dataset :

— Enron company mails (voir D.1)
— Dataset SpamAssassin (voir D.1)

Les mails de SpamAssassin ont l'avantage d'être pré-trié, contrairement aux mails de la compagnie Enron. Ainsi le développement du moteur se fera uniquement avec les mails du SpamAssassin afin de pouvoir vérifier les résultats de l'analyse.

Téléchargement des données Le téléchargement du dataset Enron est possible à partir du moment où l'on possède un compte sur la plateforme Kaggle. Le dataset SpamAssassin est ouvert, il suffit de télécharger les archives de chaque catégorie.

La récolte des données a été réalisée à la main sans automatisation. Les mails sont alors stockés dans plusieurs répertoires *HAM* et *SPAM* selon leur catégorie.

Format :
— *Enron* - 1 fichier CSV avec tous les mails
— *SpamAssassin* - 1 fichier texte par mail

Evolution possible La quantité de ressource disponible est assez limitée et principalement en anglais. Une idée aurait pu être de mettre en place un site internet où les utilisateurs peuvent fournir leurs mails avec la catégorie qu'ils estiment être la bonne.
Cette solution comporte des points d'attentions. La confiance en l'utilisateur ne peut pas être absolue. Le rapport entre les ham et spam sera très probablement disproportionné en faveur des spams. Au vu des traitements effectués, cette solution nécessitera une gestion des données personnelles en accord avec les RGPD.

1.2 Pré-traitement

La Figure 3 montre l'enchainement des étapes de traitement du mail puis du corps de texte jusqu'à la mise en base. Le chargement des mails en mémoire utilise le module python email (natif). Une grande partie des transformations sont effectuées en utilisant des expressions régulières. Trois points de contrôle permettent de décider si un mail sera effectivement utilisé ou non.

1. Échec de l'importation (fichier manquant lors de la tentative de lecture)
2. Échec de récupération du corps (charset non accepté, corps vide, type du mail non textuel)

3. Échec lors de la détermination de la langue (incapacité d'identifier les ngrams nécessaire à l'analyse)

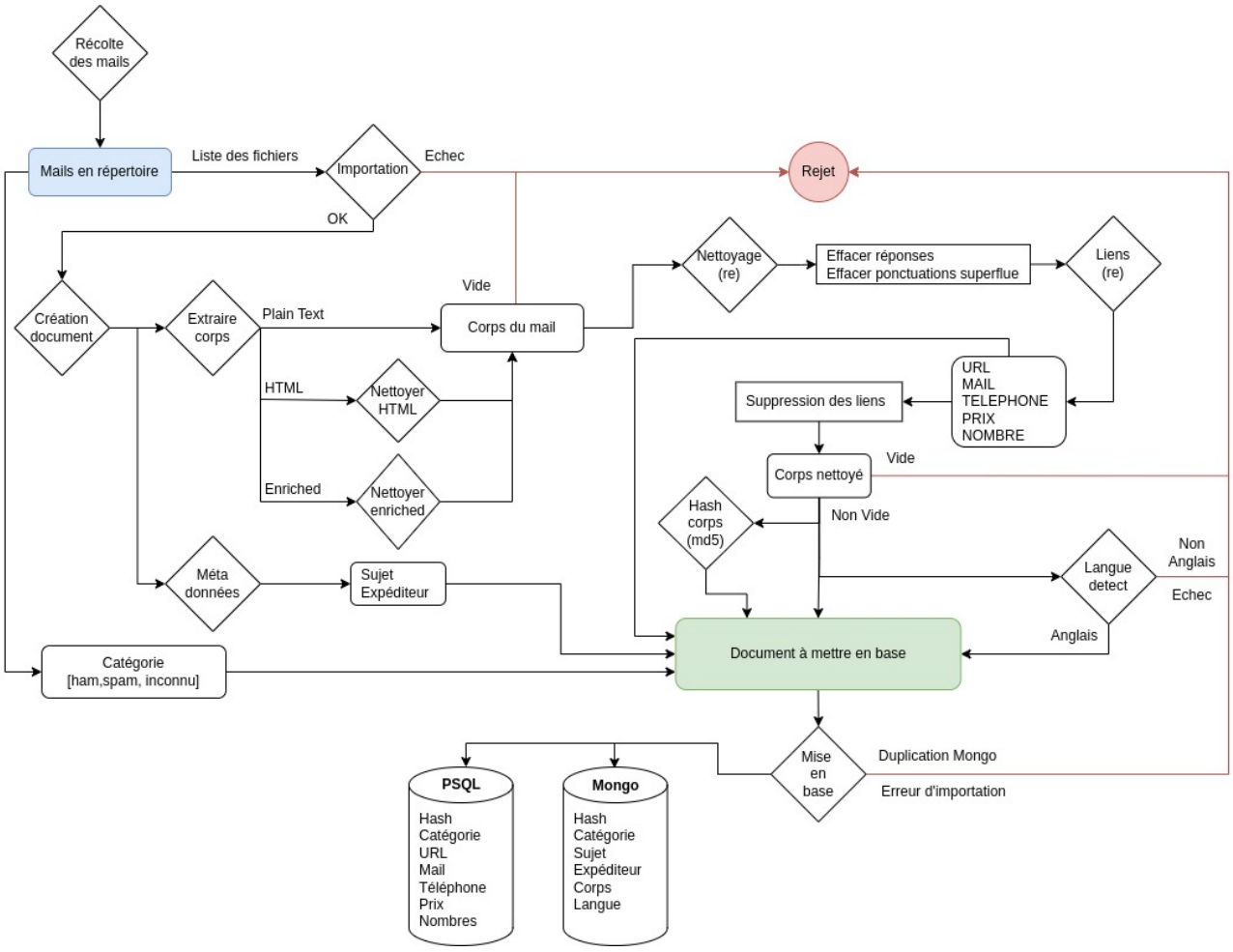

FIGURE 3 – Schéma des étapes de la phase 1

1.2.1 Importation

La fonction *email.message_from_binary_file* permet de transformer un fichier mail en objet python manipulable :

Fonction d'importation des fichiers

```
1 def load_mail(file):
2     with open(file, 'rb') as f_bin:
3         return email.message_from_binary_file(f_bin)
4
```

1.2.2 Extraction du corps des mails

Une fois le fichier importé au format *EmailMessage*, il est possible d'en extraire le corps. Le corps du mail peut être composé de plusieurs parties qui ne sont pas forcément du texte. Les parties non textuelles ne sont pas conservée.

Extraction du corps du mail

```
1 def extract_body(msg):
2     refused_charset = ['unknown-8bit', 'default', 'default_charset',
```

```
                        'gb2312_charset', 'chinesebig5', 'big5']
    body = ""

    if msg.is_multipart():
        for part in msg.walk():
            if not part.is_multipart():
                body += extract_body(part)
        return body

    if msg.get_content_maintype() != 'text':
        return ""

    if msg.get_content_charset() in refused_charset:
        return ""

    if msg.get_content_subtype() == 'plain':
        payload = msg.get_payload(decode=True)
        body += payload.decode(errors='ignore')

    if msg.get_content_subtype() == 'html':
        payload = msg.get_payload(decode=True)
        body += nettoyage.clear_html(payload.decode(errors='ignore'))

    if msg.get_content_subtype() == 'enriched':
        payload = msg.get_payload(decode=True)
        body += nettoyage.clear_enriched(payload.decode(errors='ignore'))

    return body

```

1.2.3 Nettoyage

Le nettoyage du texte utilise principalement les expressions régulières pour retirer un maximum d'éléments indésirables dans le texte. J'utilise également 2 modules externes afin de traiter le code HTML et faire la détection des mails qui ne sont pas écrits en anglais.

Par regex J'utilise le module python *re* pour réaliser les traitements suivants :

Suppression des réponses Lorsque l'on répond à un mail, le texte du message précédent est conservé dans le corps du mail. Pour permettre la distinction avec les mails précédant le caractère > est ajouté en début de ligne. Je retire toutes les lignes correspondant à des réponses afin de limiter les doublons dans les textes.

Nettoyage des réponses

```
def clear_reply(texte):
    pattern = re.compile('^>.*$', flags=re.MULTILINE)
    return re.sub(pattern, '', texte)

```

Suppression des ponctuations Pour ne pas surcharger la base de données et pour se concentrer sur le texte, une grande partie des caractères de ponctuation seront retirés. L'idée est de se concentrer sur les ponctuations classiques (., ? !).

Nettoyage des ponctuations

```
pattern_ponct = re.compile(r'[*#\\-_=:;<>\[\]"\'~)(|/$+}{@%&\\]', flags=re.MULTILINE)
def clear_ponctuation(texte):
    return re.sub(pattern_ponct, '', texte)

```

Suppression des balises pour les enriched text Certaines parties du corps de mail sont de type *enriched text*. Les balises ne sont pas pertinente dans notre analyse et sont donc retirées.

Nettoyage des balises enriched text

```
pattern_enriched = re.compile('<.*>')
def clear_enriched(texte):
    return re.sub(pattern_enriched, '', texte)

```

Suppression des liens La présence de certaines informations comment les liens URL, les adresses mails et les numéros de téléphone ne peuvent pas être utilisés dans l'analyse textuelle. Cependant, il peut être intéressant de conserver une trace de leur présence. Nous allons ainsi modifier ces liens qui seront comptabilisés avant d'être retirés du texte.

Nettoyage des liens

```
pattern_mail = re.compile('[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+')
pattern_url1 = re.compile(r'(http|ftp|https)?://([\w\-_]+(?:(?:\.[\w\-_]+)+))'
                          r'([\w\-.,@?^=%&:/~+#]*[\w\-@?^=%&/~+#])?', flags=re.MULTILINE)
pattern_url2 = re.compile(r'(\w+\.)+\w+', flags=re.MULTILINE)
pattern_tel1 = re.compile(r'\(\d{3}\)\d+-\d+')  # (359)1234-1000
pattern_tel2 = re.compile(r'\+\d+([ .-]?\d)+')    # +34 936 00 23 23

def change_lien(texte, liens):
    temp, liens['MAIL'] = re.subn(pattern_mail, '', texte)

    temp, liens['URL'] = re.subn(pattern_url1, '', temp)
    temp, nb = re.subn(pattern_url2, '', temp)
    liens['URL'] += nb

    temp, liens['TEL'] = re.subn(pattern_tel1, '', temp)
    temp, nb = re.subn(pattern_tel2, '', temp)
    liens['TEL'] += nb

    return temp

```

Suppression des nombres Comme pour les liens, les nombres sont comptabilisés et retirés. Je fais la distinction entre les nombres seuls et les nombres accompagnés de sigle monétaires.

```
MONNAIE = '$£€'
```

Nettoyage des nombres

```
pattern_prix1 = re.compile(f'[{MONNAIE}]( )?\\d+([.,]\\d+)? ', flags=re.
    MULTILINE)
pattern_prix2 = re.compile(f' \\d+([.,]\\d+)?( )?[{MONNAIE}]', flags=re.
    MULTILINE)
pattern_nb = re.compile('\\d+')

def change_nombres(texte, liens):
    temp, liens['PRIX'] = re.subn(pattern_prix1, '', texte)
    temp, nb = re.subn(pattern_prix2, '', temp)
    liens['PRIX'] += nb

    temp, liens['NOMBRE'] = re.subn(pattern_nb, '', temp)

    return temp

```

Par module Lors du processus de nettoyage, j'utilise deux modules externes plus performants que ce que j'aurais pu faire simplement avec des expressions régulières. L'un me permet de nettoyer le code HTML, l'autre de détecter la langue du message.

Suppression du code HTML Certaines parties du corps du mail sont de type HTML. J'utilise le module *BeautifulSoup* pour parser le code et récupérer le texte affiché.

Nettoyage des nombres

```
from bs4 import BeautifulSoup

def clear_html(texte):
    brut = BeautifulSoup(texte, "lxml").text
    return brut

```

Au cours du traitement avec ce module, j'ai eu une mise en garde.

```
MarkupResemblesLocatorWarning - The input looks more like a filename than markup.
You may want to open this file and pass the filehandle into Beautiful Soup
```

Après analyse, il semblerait que ce message soit levé dans le cas ou le module ne parvient pas à détecter et à récupérer le code html. Dans le mail concerné (*spamassassin/spam/00307.7ed50c6d80c6e37c8cc1b132f4a19e4d*) la partie marquée comme HTML est également encodée en base64.

```
------=_NextPart_7HZmySBWvSemNjin8Kg9YAA
Content-Type: text/html;
        charset="big5"
```

```
Content-Transfer-Encoding: base64

PGh0bWwgeG1sbnM6dj0idXJuOnNjaGVtYXMtbWljcm9zb2Z0LWNvbTp2bWwiDQp4bWxuczpvPSJ1
cm46c2NoZW1hcy1taWNyb3NvZnQtY29tOm9mZmljZTpvZmZpY2UiDQp4bWxuczp3PSJ1cm46c2No
ZW1hcy1taWNyb3NvZnQtY29tOm9mZmljZTp3b3JkIg0KeG1sbnM9Imh0dHA6Ly93d3cudzMub3Jn
L1RSL1JFQy1odG1sNDAiPg0KDQo8aGVhZD4NCjxtZXRhIGh0dHAtZXF1aXY9Q29udGVudC1UeXBl
IGNvbnRlbnQ9InRleHQvaHRtbDsgY2hhcnNldD1CaWc1Ij4NCjxtZXRhIG5hbWU9UHJvZ0lkIGNv
...
```

Sélection des mails en anglais Lors de mes tests, je me suis rendu compte que certains mails n'étaient pas en anglais. J'ai donc trouvé le module *langdetect* qui permet de détecter le langage utilisé dans un texte en utilisant un modèle *Naïve Bayes* avec une précision de 99% (voir D.2).
Je conserve dans les données à mettre en base le langage détecté avec l'idée de pouvoir traiter plusieurs langues dans le futur.

La détection de la langue est réalisée juste avant la mise en base Mongo.

Création d'un document

```python
import langdetect

def create_document(mail, categorie):
    corp = mail_load.extract_body(mail)
    corp, liens = nettoyage.clear_texte_init(corp)
    sujet, expediteur = mail_load.extract_meta(mail)

    if not corp:
        return None

    try:
        lang = langdetect.detect(corp)
    except langdetect.lang_detect_exception.LangDetectException:
        return None

    if lang != 'en':
        return None

    if categorie.lower() not in ['spam', 'ham']:
        categorie = 'inconnu'

    doc = {
        'hash': hashlib.md5(corp.encode()).hexdigest(),
        'categorie': categorie.lower(),
        'sujet': sujet,
        'expediteur': expediteur,
        'message': corp,
        'langue': lang,
        'liens': liens
    }
    return doc

```

Exemple de traitement La section suivante montre des exemples de traitement de la phase 1.

Traitement initial

```
message = '''
Message dedicated to be a sample to show how the process is clearing the
    text.

Begin reply :
> He once said
>>> that it would be great
End of reply.

Substitutions :
spamassassin-talk@example.sourceforge.net
https://www.inphonic.com/r.asp?r=sourceforge1&refcode1=vs3390
hello.foo.bar
between $ 25 and 25,21 $

A number is : 2588,8 588
Phone type a : (359)1234-1000
Phone type b : +34 936 00 23 23
Ponctuation : ---## ..
~ ~~~~~

'''
text, liens = clear_texte_init(message)
print(liens)
print(text)

```

Résultat traitement initial :

```
{'URL': 2, 'MAIL': 1, 'TEL': 2, 'NOMBRE': 3, 'PRIX': 2}

Message dedicated to be a sample to show how the process is clearing the text.

Begin reply


End of reply.

Substitutions



between and

A number is  ,
Phone type a
Phone type b
Ponctuation  ..
```

Traitement HTML

```
message_html = '''
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>Foobar</title>
</head>
<body>
I actually thought of this kind of active chat at AOL
bringing up ads based on what was being discussed and
other features
  <pre wrap="">On 10/2/02 12:00 PM, "Mr. FoRK"
  <a class="moz-txt-link-rfc2396E" href="mailto:fork_
  list@hotmail.com">&lt;fork_list@hotmail.com&gt;</a>
  wrote: Hello There, General Kenobi !?
<br>
</body>
</html>
'''
print(clear_html(message_html))

```

Résultat traitement HTML :

```
Foobar


I actually thought of this kind of active chat at AOL
bringing up ads based on what was being discussed and
other features
  On 10/2/02 12:00 PM, "Mr. FoRK"
  <fork_list@hotmail.com>
  wrote: Hello There, General Kenobi !?
```

Traitement enriched text

```
message_enriched = '''
<smaller>I'd like to swap with someone also using Simple DNS to take
advantage of the trusted zone file transfer option.</smaller>
'''
print(clear_enriched(message_enriched))

```

Résultat traitement enriched text :

```
I'd like to swap with someone also using Simple DNS to take
advantage of the trusted zone file transfer option.
```

1.3 Mise en base

La mise en base et le dernier traitement de cette phase. Les traitements précédents ont créé une liste de dictionnaires python contenant les valeurs à sauvegarder (document). Chaque document répond au schéma défini dans le tableau 1

Clé	Mongo	PSQL	Description
hash	_id	messages(hash)	signature md5 du corp
categorie	categorie	categories(nom)	ham, spam, inconnu
sujet	sujet		sujet du mail
expediteur	expediteur		source du mail
message	message		corps du mail
langue	langue		en
liens['URL']		liens(url)	nombre d'url
liens['MAIL']		liens(mail)	nombre d'adresses mail
liens['TEL']		liens(tel)	nombre de numéros de téléphone
liens['PRIX']		liens(prix)	nombre de références à des prix
liens['NOMBRE']		liens(nombre)	nombre d'apparitions de nombres

TABLE 1 – Mapping des clés python avec les bases de données

La valeur du hash va permettre d'exclure les doublons qui ont déjà été insérés dans les bases MongoDB et PSQL. Cette valeur va également permettre de faire la liaison entre les données des différentes bases.
Le schéma 4 illustre l'état des relations entre les bases de données PSQL et Mongo.

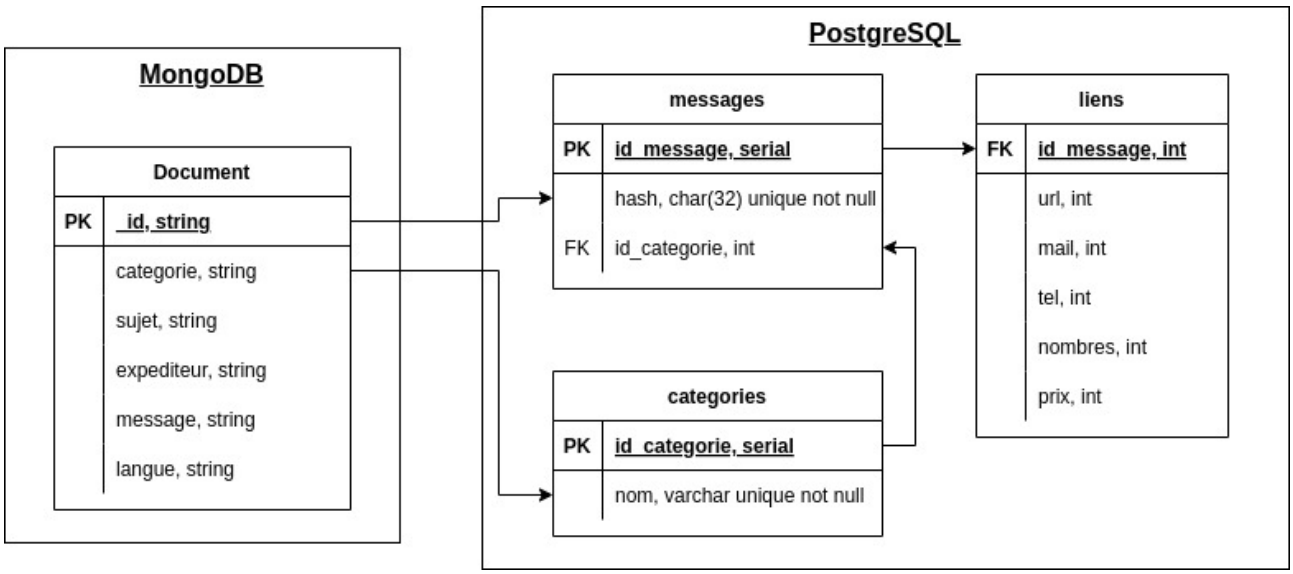

FIGURE 4 – Relation entre les bases de données

Wrapper en python La communication entre le programme et les bases de données se fait au moyen des modules suivants :

- MongoDB - pymongo (bibliothèque officielle maintenue par mongoDB)
- PSQL - psycopg2 (bibliothèque open-source), utilisée pour les requêtes simples
- PSQL - sqlalchemy (bibliothèque open-source), utilisée pour les requêtes complexes
- SQLite - sqlite3 (bibliothèque standard de python)

Des modules internes au programme ont été développer pour simplifier l'utilisation des modules d'interaction avec les bases de données. Les packages cmd_mongo.py, cmd_psql.py, cmd_sqlite.py regroupent les fonctions qui sont utilisées pour les intéractions entre le programme et les bases respectives.

Problèmes éventuels Lors de l'insertion les cas suivants sont susceptibles d'arriver si les deux bases n'ont pas été correctement nettoyées.

Situation	Raison	Solution
Mail présent dans Mongo et absent dans PSQL	Échec d'insertion dans la base PSQL	Supprimer l'entrée dans la base Mongo et relancer le traitement pour ce mail
Mail présent dans PSQL et absent dans Mongo	Suppression du mail dans la base Mongo	Supprimer le mail et toutes ses références dans la base PSQL et relancer le traitement de ce mail

TABLE 2 – Problèmes possibles avec la mise en base

Stockage des données statistiques du traitement - SQLite Les données présentent dans cette base permettent de suivre l'évolution de certaines métriques lors des différentes étapes du nettoyage. Lors de chaque étape de la phase 1 (Récolte, Création des documents, Mise en base), je calcule pour les HAM, SPAM et (HAM+SPAM) les éléments suivants :

- mails - nombre de mails
- mots - nombre de mots dans tout le corpus
- mots_uniques - nombre de mots uniques dans tout le corpus

Ces données me permettent d'estimer la quantité de données nettoyées durant cette phase.

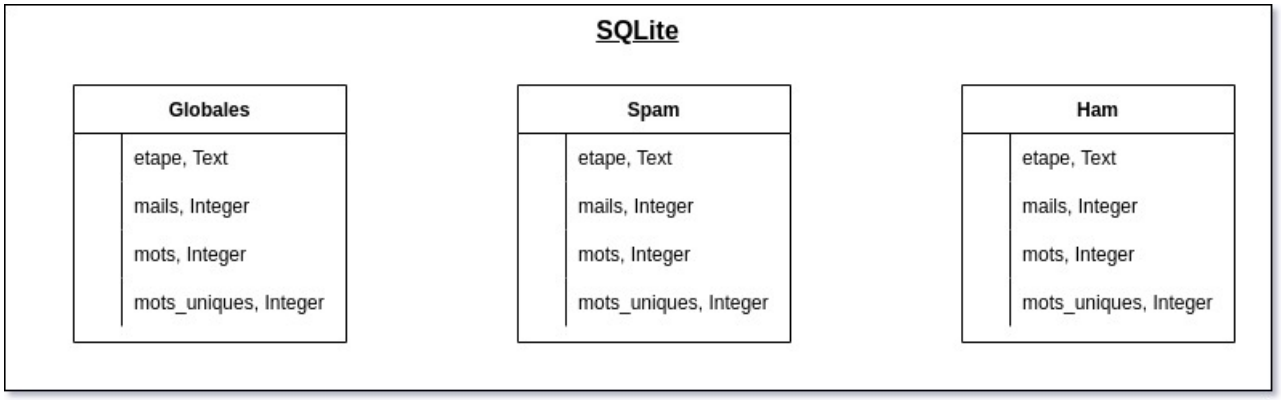

FIGURE 5 – Schéma de la base de données pour lors du traitement

Analyse

L'exécution du programme donne les informations présentées dans le tableau 3.

	mails			mot			mots uniques		
étapes	globale	spam	ham	globale	spam	ham	globale	spam	ham
récolte	5798	1897	3901	2385120	916756	1468364	262614	129090	133524
création	5662	1786	3876	1223062	589364	633698	99837	40676	59161
mise en base	5334	1530	3804	1163777	533686	630091	99834	40676	59158

TABLE 3 – Données statistiques de la fouille

La figure 6 montre un aperçu des données statistiques récoltées durant cette première phase. Les 3 premiers graphiques montrent l'évolution du nombre de documents, de mots et de mots uniques en fonction des étapes intermédiaires.
Cette visualisation permet de faire les observations suivantes :

- La diminution des documents spam est plus importante que celle des ham
- Le nombre de mots uniques ne diminue plus après la création de document
- La diminution du nombre de mots est plus importante dans les ham que dans les spam
- Le nombre de mots uniques est plus important dans les ham que dans les spam

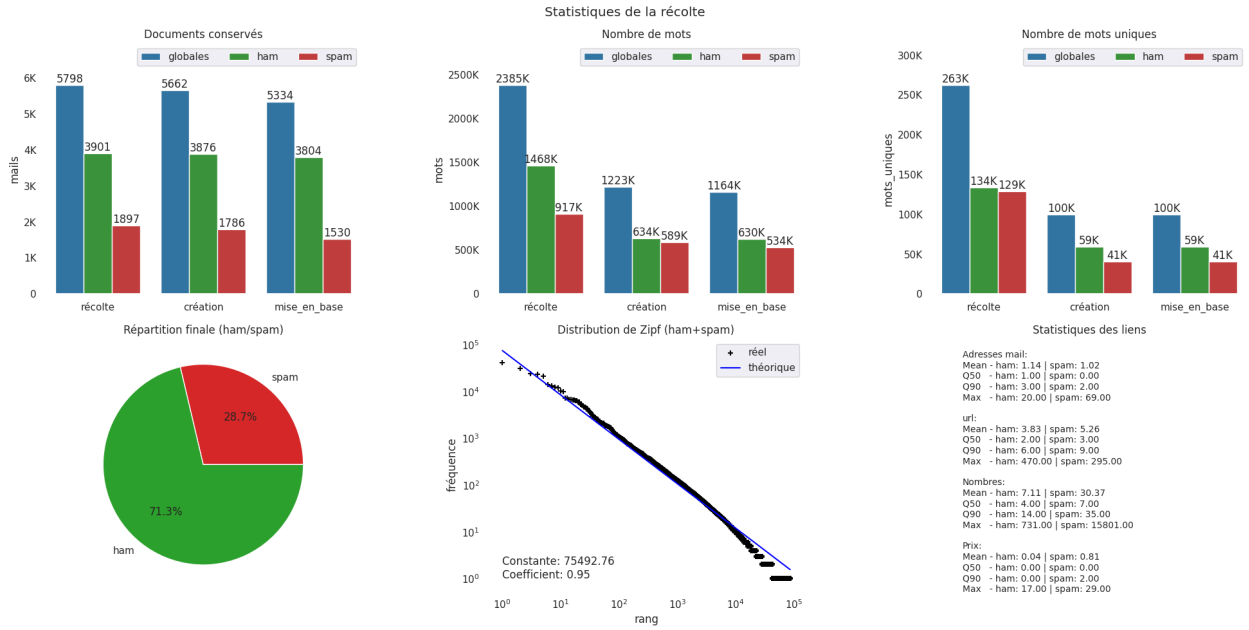

FIGURE 6 – Tableau de bord de la fouille

À l'issue de ces traitements, la proportion dans notre dataset ham/spam est d'environ 70/30. On voit également que la distribution du Zipf est globalement respectée. Il est donc fortement probable que notre dataset respecte les caractéristiques linguistiques naturelles.

L'extraction des liens informations numériques donne les statistiques du tableau 4.

	url		mail		tel		prix		nombre	
	spam	ham	spam	ham	spam	ham	spam	ham	spam	ham
moyenne	5.25	3.83	1.01	1.13	0.06	0.20	0.81	0.03	30.36	7.1
médiane	3	2	0	1	0	0	0	0	7	4
90%	9	6	2	3	0	1	2	0	35	14
maximum	295	470	69	20	67	25	29	17	15801	731

TABLE 4 – Statistiques sur les liens et informations numériques

En résumé,

Choix techniques

2 Ingénierie des langues

2.1 Recherche des caractéristiques

2.1.1 Analyse statistique

2.2 Traitement du langage

2.2.1 Lemmatisation

2.2.2 Vectorisation

3 Modélisation

3.1 Entraînements

3.2 Validation

Conclusion

A Développement visualisation distribution de Zipf

Présentation La loi de distribution de Zipf est une loi empirique (basée sur l'observation) qui veut que le mot le plus fréquent est, à peu de chose près, 2 fois plus fréquent que le 2^{eme}, 3 fois plus fréquent que le 3^{eme} etc.

La formulation finale de la 1^{ere} loi de Zipf est la suivante :

$$|mot| = constante \times rang(mot)^{k \approx 1}$$

avec $|mot|$ la fréquence d'apparition d'un mot, *constante* une valeur propre à chaque texte, *rang(mot)* la place du mot dans le tri décroissant par fréquence d'apparition et k un coefficient proche de 1.

Développement Afin de pouvoir utiliser les résultats de cette distribution dans ce projet, j'ai développé un ensemble de fonctions sur un corpus "*reconnu*". Mon choix s'est porté sur le corpus *Brown* (voir D.1) présent dans la librairie *nltk*. Ce corpus contient environ 500 documents contenant 1 millions de mot en anglais.

Le processus d'analyse se fait sur 2 versions de ce corpus.
— la première version contient tous les mots sans modifications
— le seconde version contient tous les mots sans les *stopwords*
Les *stopwords* sont des mots qui n'ont pas ou peu de signification dans un texte. Ces mots sont retirés dans la 2^e version pour voir l'effet d'une réduction sur la distribution de Zipf.

Les paragraphes ci-dessous détaillent les étapes du développement :

Étape 1 - Ordonner les mots La première étape est de compter les occurrences de tous les mots des 2 corpus et de les ranger en fonction de leur nombre d'occurrence.

Triage des mots

```
1  def frequence_mot(bag, freq=None):
2      """
3      Calcule la frequence de chaque mot dans un sac de mot
4      :param bag: <list> - liste de tous les mots d'un texte
5      :param freq: <dict> - dictionnaire avec {<str> mot: <int> frequence}
6      :return: <dict> - dictionnaire avec la frequence par mot {mot:
       frequence}
7      """
8      if freq is None:
9          freq = {}
10     for mot in bag:
11         freq[mot] = freq.get(mot, 0) + 1
12     return freq
13
14 def classement_zipf(dico):
15     """
16     Trie un dictionnaire de mots : occurence et leur assigne un rang en
       fonction du nombre d'occurence
```

```
17      :param dico: <dict> dictionnaire de mot: occurrences
18      :return: <list> {"rang": <int>, "mot": <str>, "frequence": <int>}
19      """
20      ranked = []
21      for rang, couple in enumerate(sorted(dico.items(), key=lambda item:
        item[1], reverse=True), start=1):
22          ranked.append({"rang": rang,
23                         "mot": couple[0],
24                         "frequence": couple[1]})
25
26      return ranked
```

On obtient les représentations suivantes :

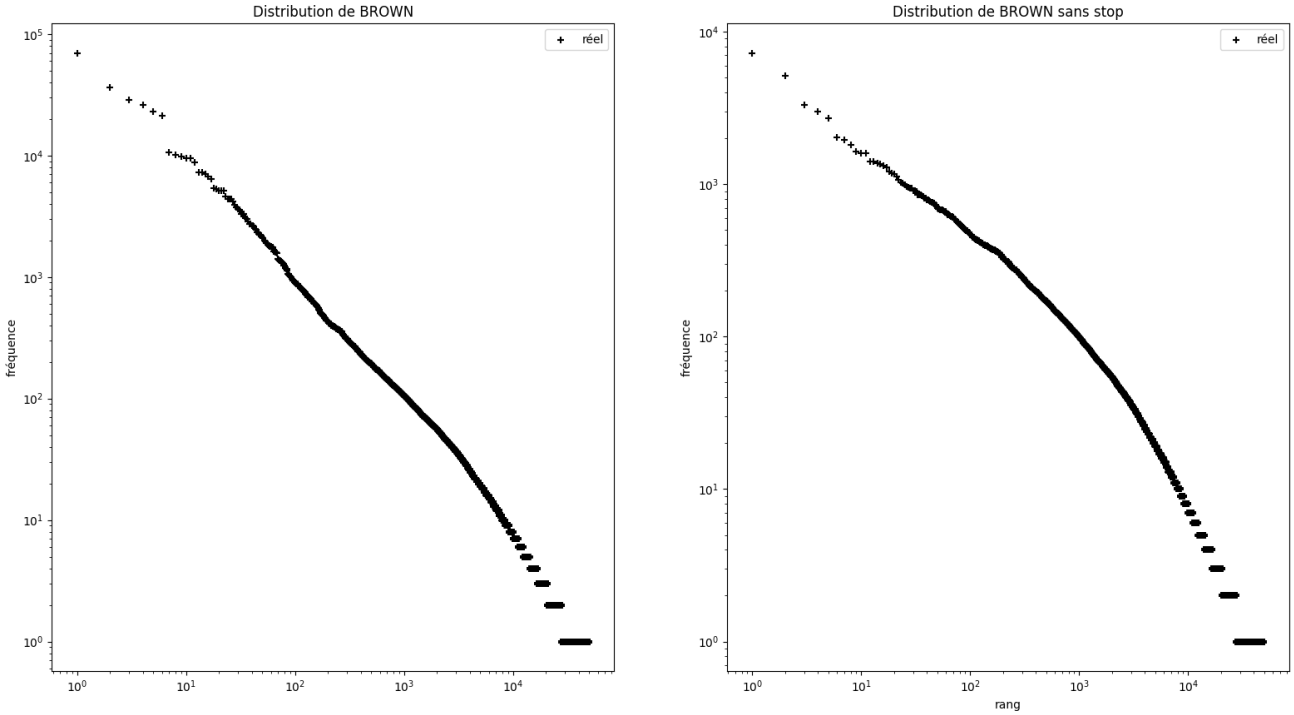

FIGURE 7 – Distribution de Zipf pour les deux corpus

— Nombre de mots dans brown : mots : 49398 occurences : 1012528
— Nombre de mots dans brown stop : mots : 49383 occurences : 578837

La distribution de la version complète du corpus semble à première vue plus fidèle à la représentation classique de la distribution de Zipf.

Etape 2 - calcul de la constante Le premier paramètre qu'il faut déterminer est la *constante*. Pour ce faire j'effectue le calcul suivant pour tous les mots :

$$constante = |mot| \times rang(mot)$$

On obtient une liste de toutes les constantes théoriques pour chaque mot selon son rang. De cette liste, nous allons extraire la moyenne et la médiane.

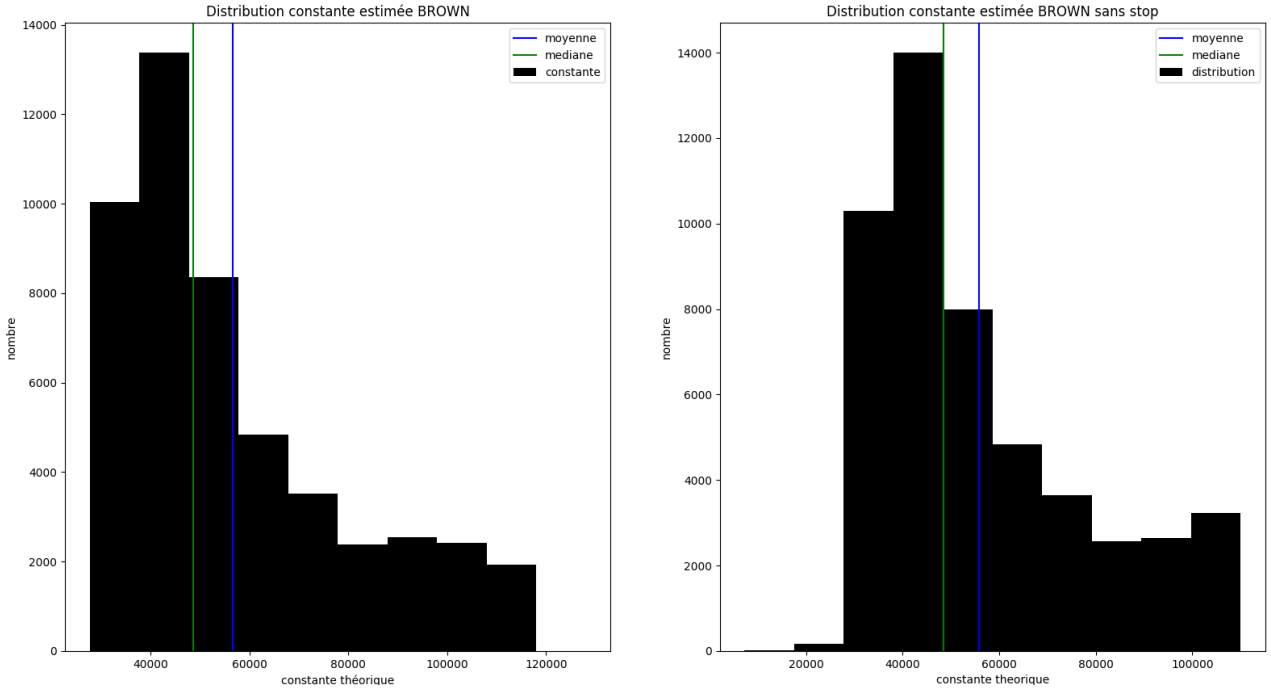

FIGURE 8 – Distribution des constantes théoriques pour les deux corpus

On voit qu'il y a une majorité de mots donnant une constante brute comprise entre 20.000 et 60.000. Dans les deux corpus La différence entre les moyennes et médianes des deux corpus n'est pas flagrante :

— Brown moyenne : 56525.81, médiane : 48601.50
— Brown (- stopwords) moyenne : 55809.97, médiane : 48494.00

Etape 3 - recherche du coefficient Le coefficient k permet d'ajuster le résultat, et pourra éventuellement donner une indication de complexité. La recherche de k se fera sur les deux corpus avec utilisant les moyennes et médianes.

Pour ce faire nous allons :

1. Faire la liste de tous les coefficients possibles dans l'intervalle $[0.86, 1.3]$ avec un pas de 0.01 [1].
2. Calculer toutes la fréquences théoriques de tous les rangs avec tous les coefficients possibles en utilisant les constantes moyenne et médiane de chaque corpus.
3. Calculer la moyenne des coûts absolus entre les fréquences théoriques par coefficient avec la fréquence réelle observée pour chaque corpus.

Le couple coefficient/constante avec le coup minimal sera retenu pour l'utilisation dans la phase de *feature engineering*.

Fonctions utilisées dans la recherche du coefficient

```
1 def zipf_freq_theorique(constante, rang, coef):
2     """
3     Calcul la frequence theorique d'un mot selon son rang, la constante du
      texte et un coeficiant d'ajustement
```

1. les bornes et le pas sont totalement arbitraire afin d'obtenir un graphique présentable

```
    :param constante: <int> constante determinee par la distribution de Zipf
    :param rang: <int> rang du mot selon sa frequence
    :param coef: <float> variable d'ajustement
    :return: <float> frequence theorique zipfienne
    """
    return constante / (rang ** coef)

def cout(l1, l2, methode):
    """
    Calcul le cout de l'ecart entre les elements de l1 et le l2, place par place
    :param l1: <list> liste d'entier
    :param l2: <liste> liste d'entier
    :param methode: <str> methode de calcul du cout
    :return: <float> cout selon methode
    """
    if len(l1) != len(l2):
        print("Erreur, fonction cout: l1 & l2 de taille differente", file=sys.stderr)
        return None

    if len(l1) == 0:
        print("Erreur, fonction cout: liste vide", file=sys.stderr)

    if methode.lower() not in ['absolue', 'carre', 'racine']:
        print("Erreur, fonction cout - methode '{}' inconnue".format(methode), file=sys.stderr)
        return None

    if methode.lower() == 'absolue':
        return np.mean([abs(x-y) for x, y in zip(l1, l2)])

    if methode.lower() == 'carre':
        return np.mean([(x-y)**2 for x, y in zip(l1, l2)])

    if methode.lower() == 'racine':
        return np.sqrt(np.mean([(x-y)**2 for x, y in zip(l1, l2)]))

    return None
```

Calcul des fréquences par coefficient

```
    ls_coef = list(np.arange(0.86, 1.3, 0.01))
    zbmo_th = {coef: [stats.zipf_freq_theorique(zb_const_moyen, r, coef) for r in zb_rang] for coef in ls_coef}
    zbme_th = {coef: [stats.zipf_freq_theorique(zb_const_median, r, coef) for r in zb_rang] for coef in ls_coef}
    zbmoth_cmoy = [stats.cout(zb_freq, zbmo_th[coef], 'absolue') for coef in ls_coef]
    zbmeth_cmoy = [stats.cout(zb_freq, zbme_th[coef], 'absolue') for coef in ls_coef]

    zbsmo_th = {coef: [stats.zipf_freq_theorique(zbs_const_moyen, r, coef)
```

```
    for r in zbs_rang] for coef in ls_coef}
8   zbsme_th = {coef: [stats.zipf_freq_theorique(zbs_const_median, r, coef
    ) for r in zbs_rang] for coef in ls_coef}
9   zbsmoth_cmoy = [stats.cout(zbs_freq, zbsmo_th[coef], 'absolue') for
    coef in ls_coef]
10  zbsmeth_cmoy = [stats.cout(zbs_freq, zbsme_th[coef], 'absolue') for
    coef in ls_coef]
```

La recherche du coefficient nous retourne les éléments suivants :

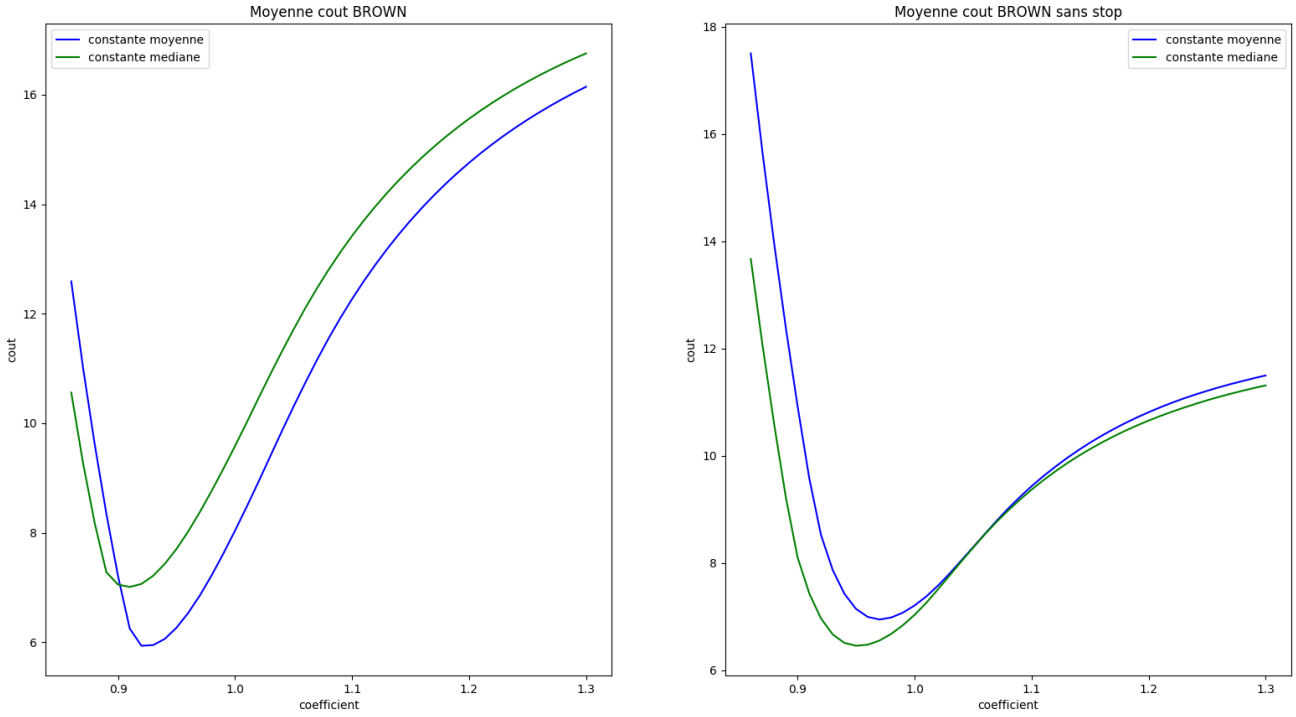

FIGURE 9 – Coût absolu moyen par coefficient

— Coût min brown moyenne : 5.93, median : 7.01
— Coût min brown (- stopwords) moyenne : 6.95, median : 6.46
— Coefficient min brown moyenne : 0.92, median : 0.91
— Coefficient min brown (- stopwords) moyenne : 0.97, median : 0.95

Résultats Le tableaux ci dessous rappelle les données récupérées au long de la recherche :

	BROWN avec stopwords	BROWN sans stopwords
nombre de mots uniques	49398	49383
nombre de mots total	1012528	578837
Constante moyenne	56525.81	55809.97
Constante médiane	48601.50	48494.00
Coefficient avec moyenne	0.92	0.97
Cout du coefficient moyenne	5.93	6.95
Coefficient avec médiane	0.91	0.95
Cout du coefficient médiane	7.01	6.46

D'après les données il est possible de dire que l'on obtient de meilleurs résultats si on conserve tous les mots du corpus. Dans ce cas l'utilisation de la moyenne des constantes génère un taux d'erreur plus faible que la médiane.

Ci-dessous la représentation des fréquences théoriques avec le coefficient optimal pour chaque corpus et chaque méthode. On voit que la courbe de la constante moyenne sur le corpus brute est celle qui suit le mieux les données réelles.

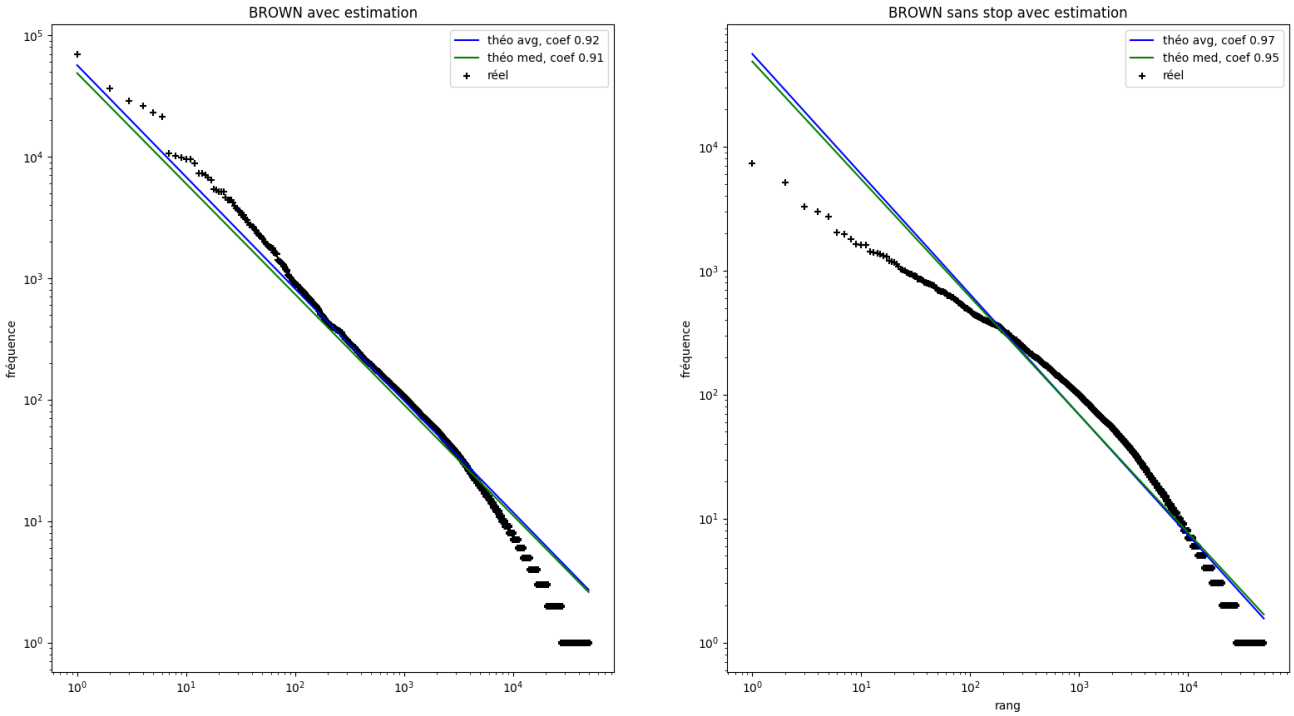

FIGURE 10 – Distribution de Zipf avec les estimations

En conclusion, j'utiliserais la moyenne des constantes sur un document complet afin de déterminer le coefficient dans ma recherche de spam.

B Modèles

B.1 Naïves Bayes

Ce type de modèle est utilisé par le module *langdetect* qui me sert pour la détection des langues.

Introduction Les modèles Naïves Bayes se basent sur le théorème de probabilité de Bayes. Il permet de déterminer la probabilité conditionnelle d'apparition d'un évènement A sachant qu'un évènement B s'est produit. Le terme naïf fait référence au fait que l'on présuppose que les évènements A et B ne sont pas corrélés.
Ces techniques sont utilisées pour des modèles de classification en apprentissage supervisé.

La formule mathématique de ce théorème est la suivante :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

On recherche ici $P(A|B)$, c'est a dire la probabilité d'apparition d'un évènement A sachant que l'évènement B s'est produit.

Pour ce faire nous avons besoin des données suivantes :
— $P(B|A)$ est la probabilité que l'évènement B s'est produit sachant que l'évènement A s'est produit
— $P(A)$ est la probabilité d'apparition de l'évènement A
— $P(B)$ est la probabilité d'apparition de l'évènement B

Exemples d'utilisation Les exemples ci dessous vont permettre d'illustrer l'utilisation de cette technique. D'abord manuellement sur un petit jeu de données puis à l'aide d'un code pré-existant sur un autre jeu de données plus important.

Manuel Dans cet exemple nous allons déterminer la probabilité qu'a un joueur d'aller sur le terrain selon les conditions météorologiques. Cette probabilité sera calculée en fonction des données récupérées lors des matchs précédents.[2]

On recherchera ainsi la probabilité de présence sur le terrain d'un joueur selon la météo $P(A|B)$. Pour ce faire nous auront besoin de :
— $P(A)$ Probabilité de jouer quelque soit le temps
— $P(B)$ Probabilité de l'évènement météorologique
— $P(B|A)$ Probabilité de l'évènement sachant que le joueur a été sur le terrain

Table 5 – Données de présence sur le terrain

météo	soleil	soleil	couvert	pluie	pluie	pluie	couvert
présent	non	non	oui	oui	oui	non	oui
météo	soleil	soleil	pluie	soleil	couvert	couvert	pluie
présent	non	oui	oui	oui	oui	oui	non

2. Les données présentées sont inventées

TABLE 6 – Synthèse et probabilité simple P(A) et P(B)

météo	oui	non	$P(B)$
couvert	4	0	4/14
soleil	2	3	5/14
pluie	3	2	5/14
$P(A)$	9/14	5/14	

On peut déterminer les probabilités de chaque météo en fonction de la présence du joueur sur le terrain P(B|A). Pour ce faire on divise le nombre d'évènements de présence du joueur lors d'un évènement météo par le nombre total d'évènements de présence du joueur

TABLE 7 – Probabilité météo selon présence du joueur

météo	P(B|oui)	P(B|non)
couvert	4/9	0/5
soleil	2/9	3/5
pluie	3/9	2/5

On va maintenant calculer la probabilité qu'à un joueur d'être sur le terrain si le temps est couvert.
On commence par la probabilité du oui :

$$\begin{aligned} P(A|B) &= \frac{P(B|A)P(A)}{P(B)} \\ P(A|B) &= \frac{\frac{4}{9} \cdot \frac{9}{14}}{\frac{4}{14}} \\ P(A|B) &= \frac{\frac{4}{14}}{\frac{4}{14}} \\ P(A|B) &= \frac{4}{14} \cdot \frac{14}{4} \\ P(A|B) &= 1 \end{aligned}$$

On enchaîne sur la probabilité de ne pas jouer si le temps est couvert

$$\begin{aligned} P(A|B) &= \frac{P(B|A)P(A)}{P(B)} \\ P(A|B) &= \frac{\frac{0}{5} \cdot \frac{5}{14}}{\frac{4}{14}} \\ P(A|B) &= 0 \cdot \frac{14}{4} \\ P(A|B) &= 0 \end{aligned}$$

On peut dire que si le temps est couvert le joueur très probablement sur le terrain On peut également déterminer la probabilité de jouer pour chaque évènement météo

TABLE 8 – Probabilité présence du joueur selon la météo

météo	oui	non	plus probable
couvert	1	0	oui
soleil	2/5	3/5	non
pluie	3/5	2/5	oui

Cas polynomial : Il est possible de déterminer la probabilité d'un évènement par rapport à plus autres. Dans ce cas, il faudra multiplier entre elles les probabilités de ces évènements selon l'apparition de l'évènement voulu.

Calcul pour un évènement (A) selon 2 autres évènements (B et C)

$$P(A|BC) = \frac{P(B|A)P(C|A)P(A)}{P(B)P(C)}$$

En code Dans cet exemple nous allons utiliser un code existant dans la librairie python scikit-learn[1]. Ce moteur Naïves Bayes va nous permettre cette fois-ci de catégoriser des variétés d'iris selon la longueur et la largeur des pétales et des sépales. Les données proviennent cette fois-ci d'un dataset également disponible dans scikit-learn.

Nous allons utilisé le modèle *GaussianNB* de scikit-learn qui est adapté lorsque les données utilisées suivent une distribution normale. Ce qui semble être le cas pour les longueurs et largeur des sépale.

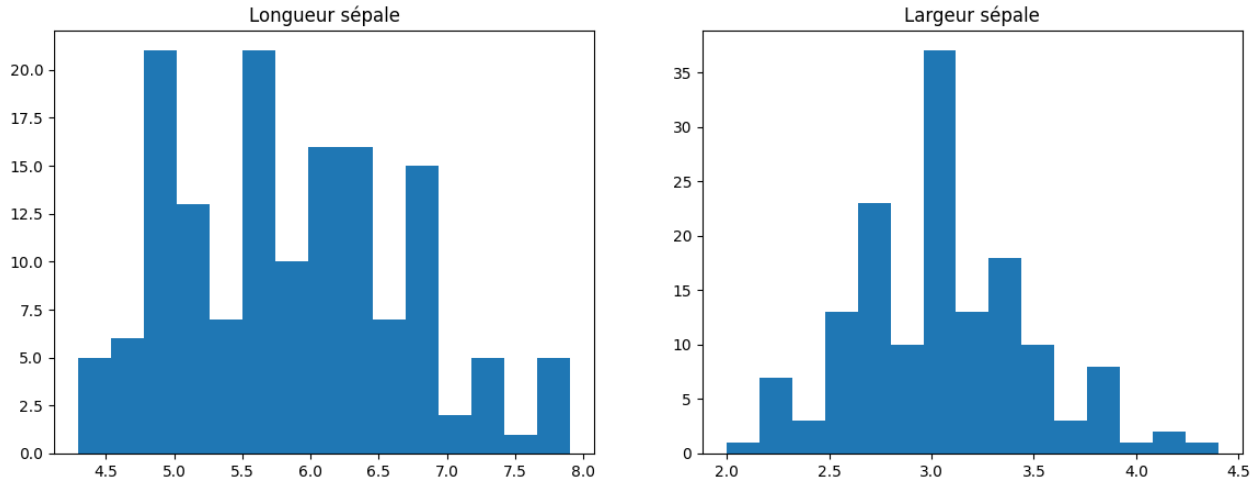

FIGURE 11 – Distribution des longueurs et largeurs des sépales

Progamme complet

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix,
    ConfusionMatrixDisplay, f1_score, \
     recall_score

import matplotlib.pyplot as plt

X, y = load_iris(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=0)
model = GaussianNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

```
precision = accuracy_score(y_pred, y_test)
recall = recall_score(y_test, y_pred, average="weighted")
f1 = f1_score(y_pred, y_test, average="weighted")

print("Precision:", precision)
print("Rappel:", recall)
print("Score F1:", f1)

plt.figure('Donnees du modele', figsize=(14, 5))
plt.subplot(1, 3, 1, title='Donnees du train set')
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
plt.xlabel('Sepale long.')
plt.ylabel('Sepale larg.')
plt.subplot(1, 3, 2, title='Donnees du test set')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
plt.xlabel('Sepale long.')
plt.subplot(1, 3, 3, title='Donnees test apres evaluation')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred)
plt.xlabel('Sepale long.')
plt.show()

cm = confusion_matrix(y_test, y_pred, labels=[0, 1, 2])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1,
    2])
disp.ax_.set_title('Matrice de confusion')
disp.plot()
plt.show()

plt.figure('Distribution des donnees Iris', figsize=(14, 5))
plt.subplot(1, 2, 1, title='Longueur sepale')
plt.hist(X[:, 0], bins=15)
plt.subplot(1, 2, 2, title='Largeur sepale')
plt.hist(X[:, 1], bins=15)
plt.show()
```

Les données du dataset ont été séparés en 2 jeux, un pour l'entraînement du modèle et un pour le test. On obtient alors la représentation suivantes après entrainement et test du modèle

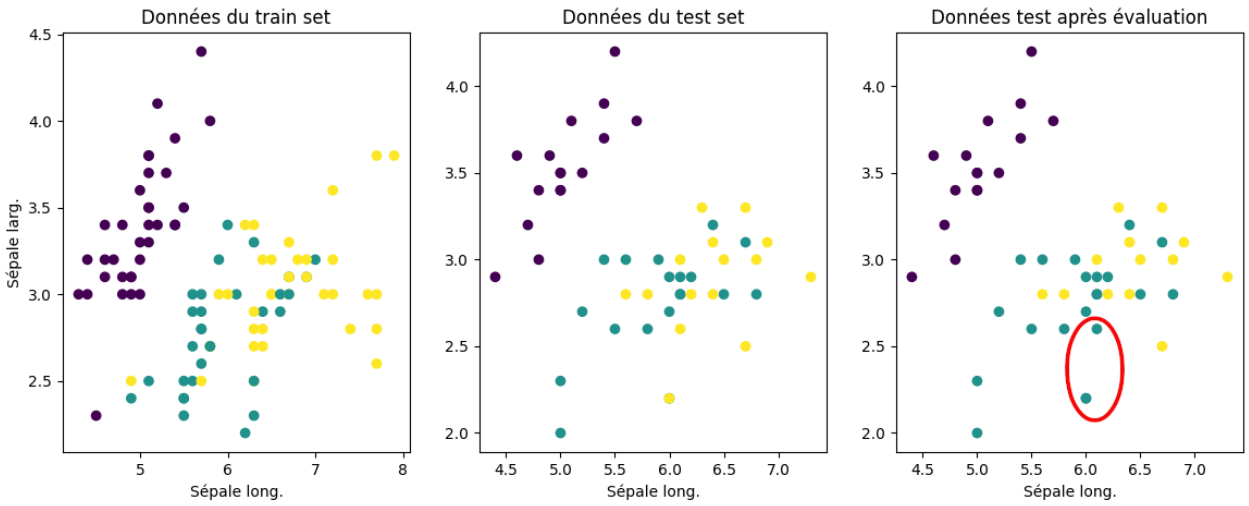

FIGURE 12 – Représentation des données

Dans les données de test nous avons 2 catégorisations qui n'ont pas été réalisées correctement. On obtient les scores suivants :

— Précision : 0.96 [3]
— Rappel : 0.96 [4]
— Score F1 : 0.9604285714285714 [5]

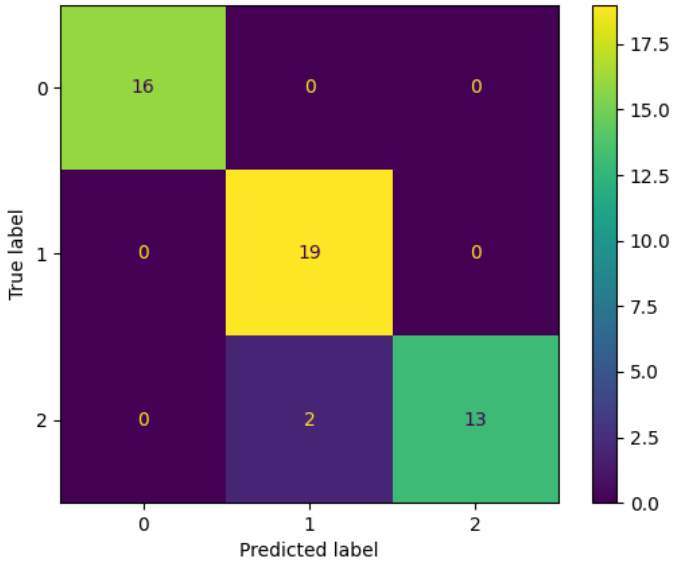

FIGURE 13 – Matrice de confusion

A l'aide de ce modèle nous devrions avoir une 96% de chance de déterminer la bonne variété d'iris en se basant sur la longueur et la largeur des sépales.

Avantages et inconvénients Le modèle Naïve Bayes est un modèle simple et rapide qui ne nécessite pas de grande capacités de calcul. De ce fait il permet de traiter une grande quantité

3. La précision est la proportion des éléments correctement identifiés sur l'ensemble des éléments prédit
4. Le rappel est la proportion des éléments correctement identifiés sur l'ensemble des éléments de la catégorie
5. Le Score F1 est la moyenne harmonique calculée de la manière suivante $2*(precision*rappel)/(precision+rappel)$

de données.

Cependant, les données qui lui sont fournies ne doivent pas être corrélées ce qui est rarement le cas dans les problèmes du monde réel. Ce type de modèle est limité à des problèmes de classification supervisée. Si on se fie à l'équation (1) la probabilité d'apparition de l'évènement B : P(B) ne peut pas être nulle.

C Bibliographie

Références

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830, 2011.

D Sitotec

D.1 Corpus

— Enron company mails, fichier CSV contenant l'ensemble des mails d'une entreprise ayant fermée ses portes (33.834.245 mails) [en ligne], https://www.kaggle.com/wcukierski/enron-email-dataset (consulté le 27/01/2022)
— Mails project SpamAssassin, projet opensource de détection de spam (6065 fichiers email déjà trier en ham et spam) [en ligne], https://spamassassin.apache.org/old/publiccorpus/ (consulté le 27/01/2022)
— Brown corpus, ensemble de texte en anglais publié en 1961 qui contient plus d'un million de mots https://www.nltk.org/book/ch02.html (consulté le 20/08/2022)

D.2 Modules

— Page Github du projet *langdetect* capable de différencier 49 langages avec une précision de 99%, [en ligne] https://github.com/Mimino666/langdetect (consulté le 04/12/2022)

— Language Detection Library, présentation du module (anglais) [en ligne] https://www.slideshare.net/shuyo/language-detection-library-for-java (consulté le 04/12/2022)
— Suite de cours et de ressources en ligne pour comprendre MongoDB et réussi a faire la connexion avec un programme Python ; [en ligne] https://learn.mongodb.com/learning-paths/mongodb-python-developer-path (consulté le 09/2023)
— Documentation de la librairie standard python sqlite [en ligne] https://docs.python.org/3/library/sqlite3.html (consulté le 09/2023)
— Documentation de la librairie psycopg2 [en ligne] https://pypi.org/project/psycopg2/ (consulté le 09/2023)
— Documentation de la librairie sqlalchemy [en ligne] https://docs.sqlalchemy.org/en/20/index.html (consulté le 09/2023)

D.3 Modèles

Naïves Bayes Le modèle Naïves Bayes est employé dans le module langdetect (D.2)
— Les algorithmes de Naïves Bayes, Explication sommaire du principe de ces type d'algorithme, [en ligne] https://brightcape.co/les-algorithmes-de-naives-bayes/ (consulté le 26/03/2023)
— Naive Bayes Classification Tutorial using Scikit-learn, exemple d'utilisation de ce type de modèle avec python (anglais) [en ligne] https://www.datacamp.com/tutorial/naive-bayes-scik (consulté le 26/03/2023)
— Scikit learn Naive Bayes, description des types d'algorithme disponibles dans le module Scikitlearn en python (anglais) [en ligne] https://scikit-learn.org/stable/modules/naive_bayes.html (consulté le 26/03/2023)

E Code Source

E.1 GitHub

L'ensemble du code est disponible dans mon repository GitHub. https://github.com/peredur0/errol