Projet L3
Fouille de données
Ingénierie des langues

GOEHRY Martial
16711476

7 août 2024

Table des matières

1 Fouille de données **5**
1.1 Récolte des données . . . 5
1.2 Pré-traitement . . . 5
1.2.1 Importation . . . 6
1.2.2 Extraction du corps des mails . . . 6
1.2.3 Nettoyage . . . 7
1.3 Mise en base . . . 13

2 Ingénierie des langues **19**
2.1 Recherche de caractéristiques . . . 19
2.2 Traitement du langage . . . 25
2.2.1 Lemmatisation . . . 26
2.2.2 Vectorisation . . . 29

3 Modélisation **35**
3.1 Création des modèles . . . 36
3.1.1 Résultat . . . 37

A Développement visualisation distribution de Zipf **40**

B Modèles **46**
B.1 Naïves Bayes . . . 46

C Bibliographie **52**

D Sitotec **53**
D.1 Corpus . . . 53
D.2 Modules . . . 53
D.3 Modèles . . . 53

E Code Source **54**
E.1 GitHub . . . 54

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.

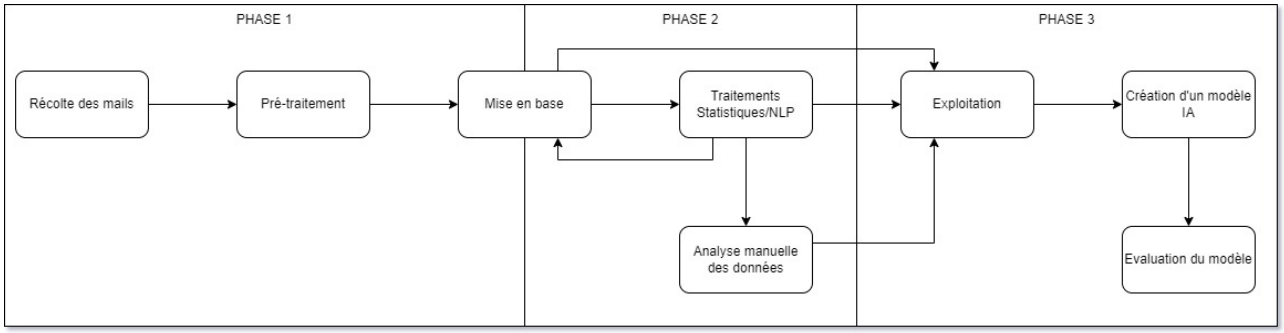


FIGURE 1 – Schéma des grandes étapes

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.

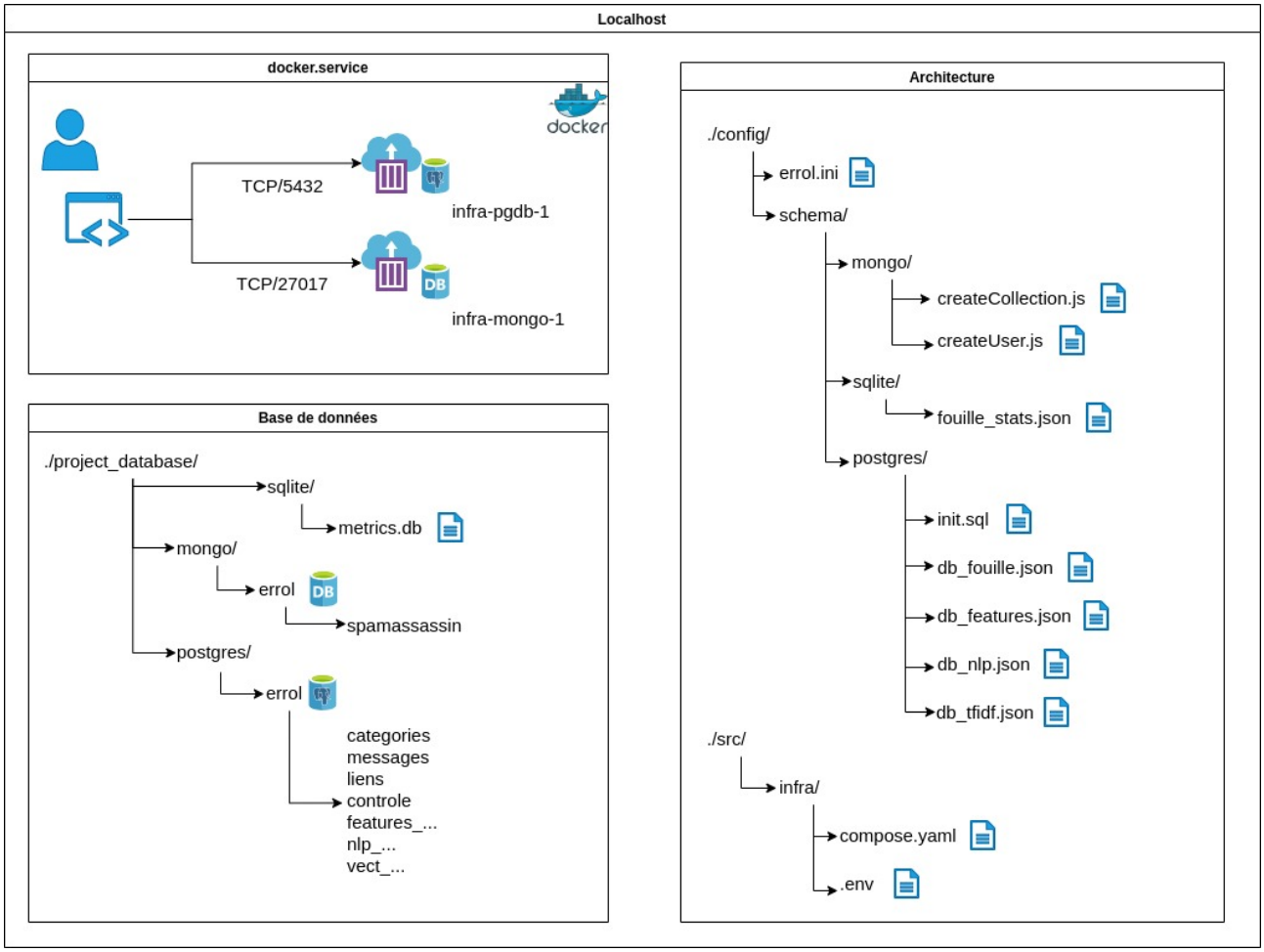


FIGURE 2 – Schéma de l'architecture Docker

Les différentes tables et collections sont créées dynamiquement au cours des étapes du projet.

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. Enfin 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)

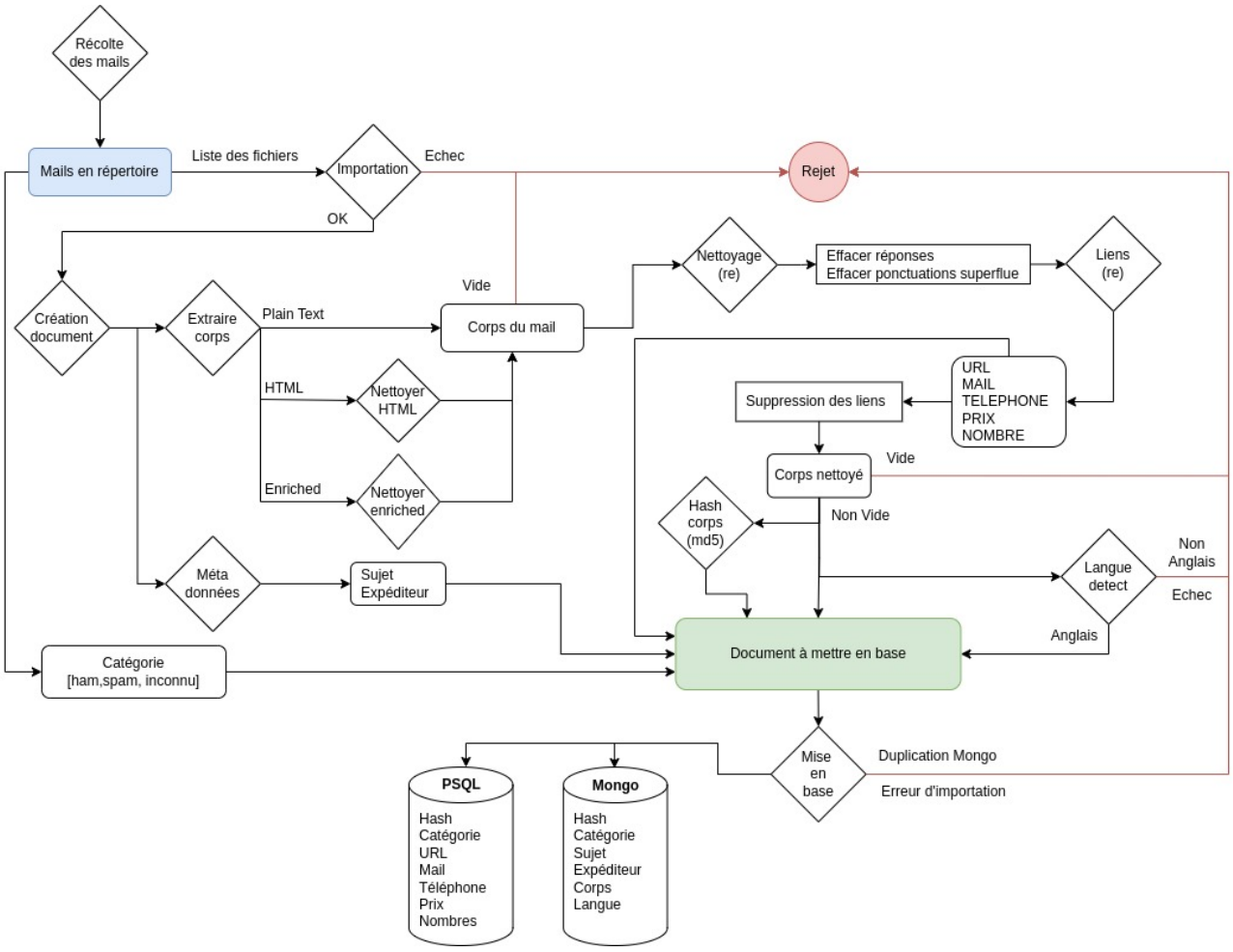


FIGURE 3 – Schéma des étapes de la phase 1

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.

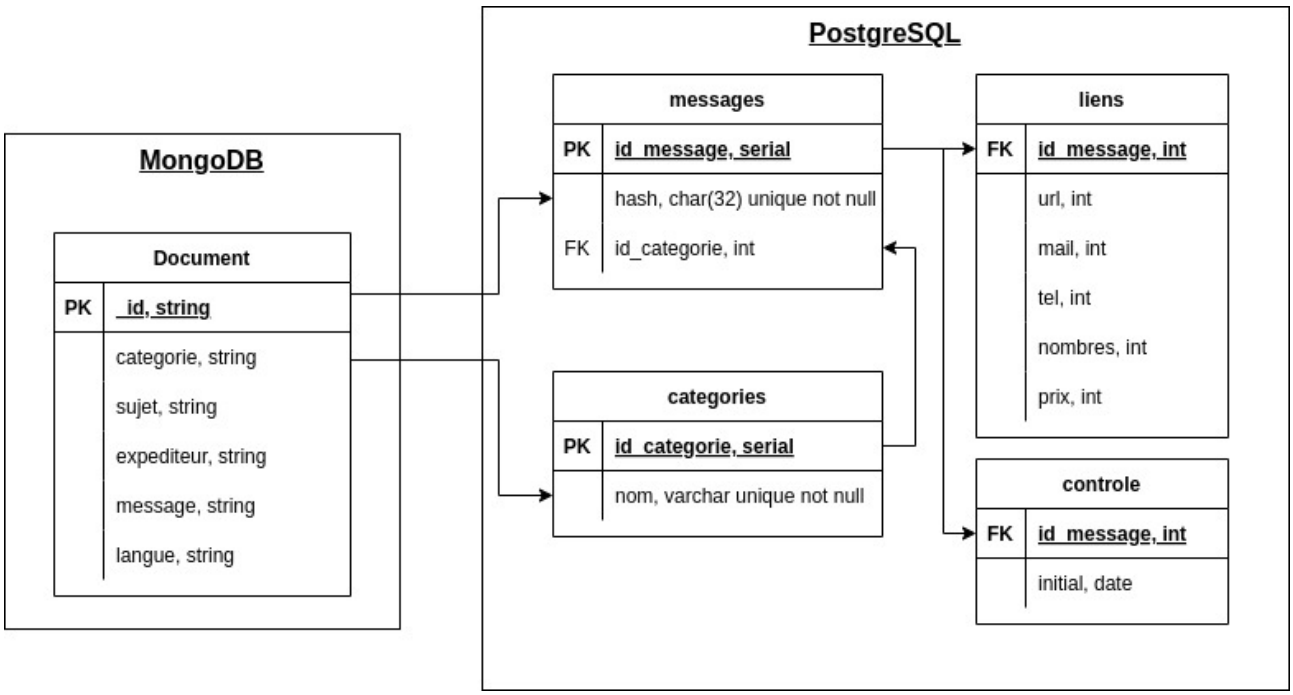


FIGURE 4 – Relation entre les bases de données

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 et les créations dataframes.
- 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.

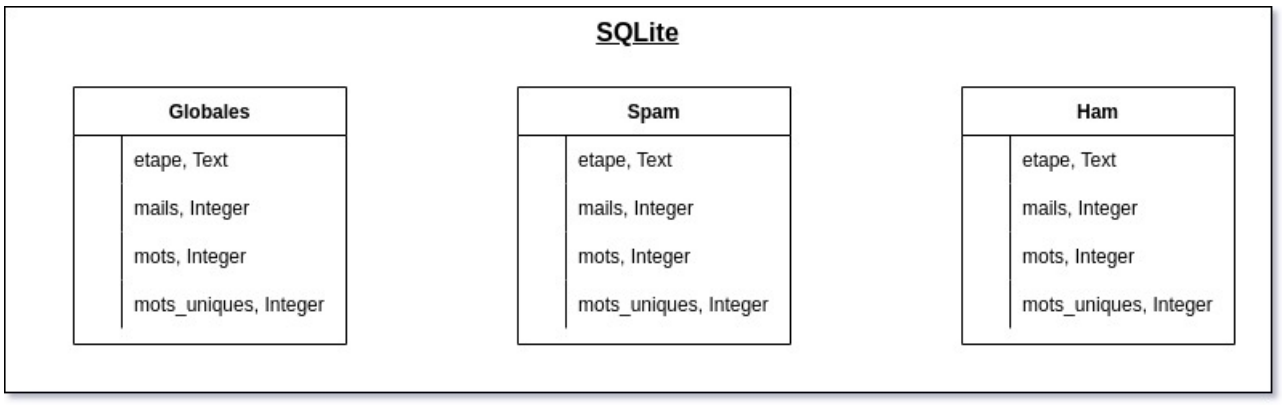


FIGURE 5 – Schéma de la base de données pour lors du traitement

Analyse

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

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

TABLE 3 – Données statistiques de la fouille

	mails			mots			mots_uniques		
categorie	globales	spam	ham	globales	spam	ham	globales	spam	ham
etape									
récolte	5798	1897	3901	2385120	916756	1468364	262614	129090	133524
création	5658	1784	3874	1222793	589119	633674	99772	40615	59157
mise_en_base	5333	1528	3805	1163550	533441	630109	99772	40615	59157

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

Après la phase de récolte, on remarque qu'il y a un nombre plus important de document en double dans la catégorie *spam*. Il y a également une réduction importante du nombre de mots ainsi que du nombre de mots uniques dans les *ham*. Cette réduction peut s'expliquer par le nettoyage du corps des mails :

— Retrait des réponses
— Retrait de certaines ponctuations
— Suppression des balises HTML et enriched text
— Retrait des liens, et nombres

Enfin, on peut voir que les *ham* utilisent plus de mots uniques que les *spam*. Il est donc possible que le vocabulaire des *spam* soit plus restreint.

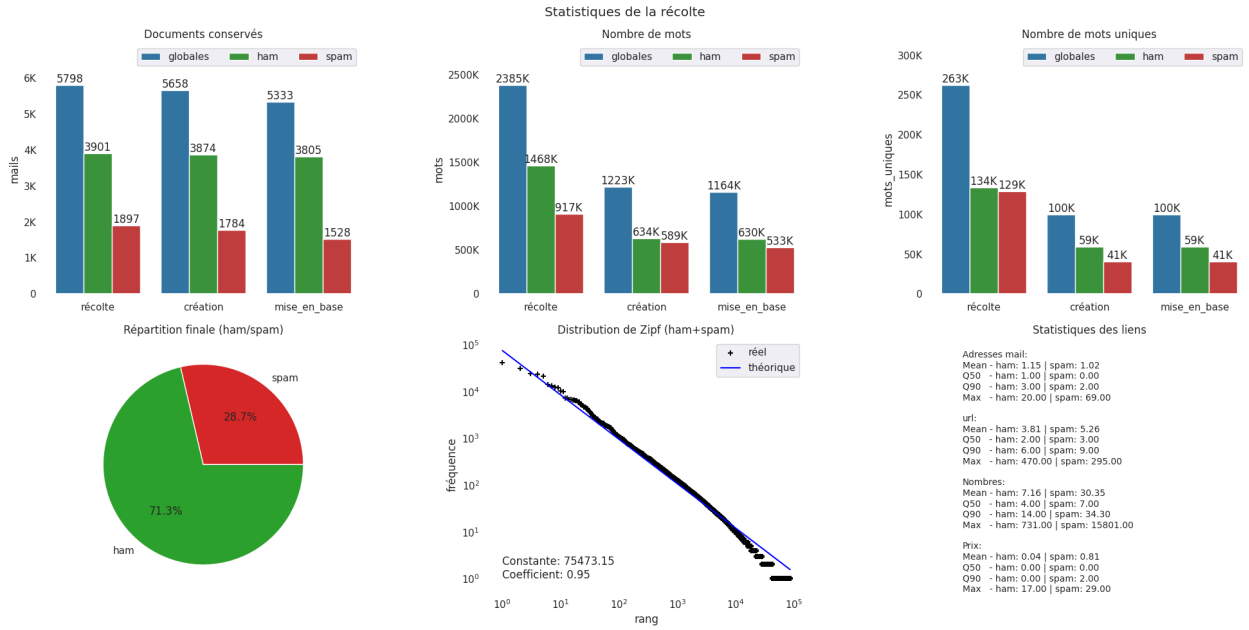


FIGURE 6 – Tableau de bord de la fouille

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

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

Les données statistiques sur les liens ne permettent pas mettre en avant une différence franche dans l'utilisation de ces éléments dans les catégories observées. Seul la présence de nombres ou d'URL seraient éventuellement susceptible d'apporter une aide à la catégorisation.

TABLE 4 – Statistiques sur les liens et informations numériques

	categorie	ham	spam
url	mean	3.81	5.26
	q50	2.0	3.0
	q90	6.0	9.0
	max	470.0	295.0
mail	mean	1.15	1.02
	q50	1.0	0.0
	q90	3.0	2.0
	max	20.0	69.0
tel	mean	0.21	0.06
	q50	0.0	0.0
	q90	1.0	0.0
	max	25.0	67.0
nombre	mean	7.16	30.35
	q50	4.0	7.0
	q90	14.0	34.3
	max	731.0	15801.0
prix	mean	0.04	0.81
	q50	0.0	0.0
	q90	0.0	2.0
	max	17.0	29.0

Choix techniques

Langage de programmation Python a été choisi pour développer ce programme pour les raisons suivantes :

- Large choix de modules, dont plusieurs spécialisés dans le traitement de données
- Langage simple et flexible
- Facilement portable et executable avec la fourniture des environnements virtuel

Multiprocessing Le module natif multiprocessing de python est utilisé pour effectuer un traitement sur plusieurs documents en parallèle.

— Comptage des mots et mots uniques par catégorie - word_count.fouille_wc()
— Création d'un document à partir d'un fichier - fouille_doc()

Le programme détermine au démarrage le nombre de processus qu'il va pouvoir lancer en parallèle en fonction du nombre de processeurs disponible sur la machine. Le programme limite son usage à la moitié des processeurs disponibles.
L'utilisation de ce module a réduit significativement le temps de traitement de la fouille par rapport à mes versions précédentes.

Mise en base La mise en base des documents utilise les capacités d'insertion par lots de mongo et de postgres. Le résultat de la création de document est coupé en lot de 50 documents. Chaque lot est inséré itérativement dans les deux bases.
Seule l'insertion dans la base mongo est susceptible d'échouer pour les raisons suivantes :

— Un champ ne possède pas le bon format - expediteur None alors que str est attendu
— Clé dupliquée - document déjà présent dans la base
— Erreur d'écriture - perte d'accès à la base

Dans le cas où l'insertion du lot échouerait, chaque document du lot est inséré individuellement. Seuls les documents insérés dans la base mongo auront une entrée correspondante dans la base PSQL.

Contrôle Une table de contrôle dans la base Postgres va permettre de suivre l'avancée du traitement pour chaque document et les raisons éventuelles d'un échec. Elle va aussi permettre d'exclure un document d'un traitement qui aurait déjà effectué. Cette table est amenée à évoluer au cours des différentes phases. Il a été décidé d'ajouter cette table PSQL car cette base est appelée dans toutes les phases du projet.

Métriques L'utilisation du nombre de documents, de mots et de mots uniques pour mesurer la performance de cette phase a été commandée par le cours. Il n'y a pas eu de difficultés majeures sur cette partie. Ces données ne sont pas conservées au fur et à mesure des diverses importations. Je n'ai pas jugé pertinent de les employer lors des traitements futurs.

La recherche des apparitions d'adresse mail, de liens url, de numéros de téléphone, de prix et de nombres est une idée personnelle. Je voulais voir si ces éléments étaient susceptible d'être discriminant pour la classification. Les résultats ne sont pas aussi marqués que je ne l'aurais cru. Je pense qu'il aurait été possible d'améliorer la justesse de ces données en travaillant sur plus de cas possible. Ou en analysant plus en profondeur les substitutions effectuées notamment sur les nombres.
Cela étant la substitution et effacement de ces éléments dans le corps de texte peut répondre à une nécessité d'anonymisation. De plus la suppression des liens URL évite de conserver et de rendre accessible des liens potentiellement frauduleux.

Le développement du code pour la distribution de ZIPF a été réalisée suite à une discussion avec le professeur. Initialement, je souhaitais également tester cette distribution sur chaque message individuellement. Mais la quantité de mots par message n'est peut-être pas suffisant pour obtenir des résultats probants. J'ai dû me renseigner beaucoup pour arriver à comprendre l'équation sous-jacente et comment je pouvais la coder de manière efficace et visuelle. Vous pouvez voir le résultat de ce travail dans l'annexe A.

Exemple de commandes pour exécuter la fouille

```
$ make venv
$ make docker_start
cd ./src/infra/; docker compose up -d; cd -
[+] Running 3/3
  Network infra_default    Created
  Container infra-mongo-1  Started
  Container infra-pgdb-1   Started

$ source .venv/bin/activate
(.venv)$ ./errol.py fouille -g \
        -a ./project_data/spamassassin/easy_ham \
           ./project_data/spamassassin/easy_ham_2 \
        -p ./project_data/spamassassin/spam \
```

```
./project_data/spamassassin/spam_2
```

Il est possible d'accéder aux bases de données avec les commandes suivantes :

```
$ sqlite3 project_database/sqlite/metrics.db
$ psql -h localhost -p 5432 errol psql_errol
$ mongosh -u mongo_errol -p <MOT DE PASSE> errol
```

2 Ingénierie des langues

Les opérations de cette phase ont pour but d'extraire les caractéristiques des textes collectés à la phase précédente. Nous allons rechercher des informations numériques sur la forme des messages comme le nombre de mots, de ponctuations, ...
Enfin, nous appliquerons des techniques de traitement du langage naturel pour travailler sur le fond des corps de mail.

2.1 Recherche de caractéristiques

Durant cette phase, nous allons rechercher les caractéristiques de la forme du texte. L'objectif est de voir si l'utilisation de ponctuation, de certaines formes de mots (majuscule/minuscule) ou de construction visuelle d'un texte (espace ligne) est différente pour chaque catégorie de mail.

Les messages seront directement extraits de la base mongo utilisée lors de la phase de fouille. Chaque message sera analysé individuellement dans une suite de fonction. Les étapes de cette phase sont décrites dans le schéma 7.

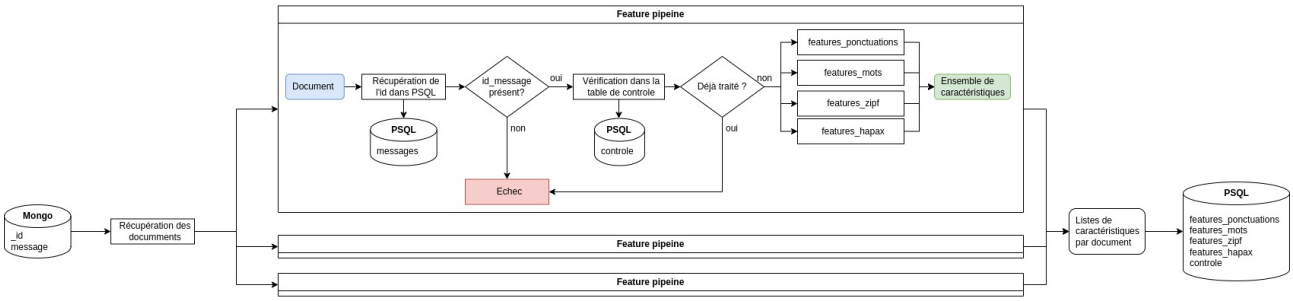


FIGURE 7 – Schéma de la recherche de caractéristiques

Fonctions de la recherche de caractéristiques

```
1
2  def features_ponctuations(texte):
3      return {
4          "point": texte.count('.'),
5          "virgule": texte.count(','),
6          "exclamation": texte.count('!'),
7          "interrogation": texte.count('?'),
8          "tabulation": texte.count('\t'),
9          "espace": texte.count(' '),
10         "ligne": texte.count('\n') + 1,
11         "ligne_vide": len(re.findall(r'^\s*$', texte, re.MULTILINE))
12     }
13
14 ef features_mots(texte):
15     tokens = re.findall(r'\w+', texte, re.MULTILINE)
16     return {
17         'char_minuscules': len(re.findall(r'[a-z]', texte, re.MULTILINE)),
18         'char_majuscules': len(re.findall(r'[A-Z]', texte, re.MULTILINE)),
19         'mots': len(tokens),
20         'mots_uniques': len(set(tokens)),
```

```
21          'mots_majuscules': sum(mot.isupper() for mot in tokens),
22          'mots_capitalizes': sum(bool(re.match(r'[A-Z][a-z]+', mot)) for
    mot in tokens)
23      }
24
25
26  def features_zipf(texte):
27      tokens = re.findall(r'\w+', texte, re.MULTILINE)
28      z_data = zipf.zipf_process(tokens)
29      return {
30          'constante': float(z_data.get('const_moy')),
31          'coefficient': float(z_data.get('coef_min')),
32          'taux_erreur': float(z_data.get('cout_min'))
33      }
34
35
36  def features_hapax(texte):
37      tokens = re.findall(r'\w+', texte, re.MULTILINE)
38      data = zipf.hapax(tokens)
39      data['nombre_hapax'] = data.pop('nombres')
40      return data
41
```

Choix technologiques

Traitements La recherche des caractéristiques de ponctuations, de type de char, ou de lignes est faite en utilisant la fonction standard *str.count()* et le module *re*.

Cette suite de traitement intègre également des fonctions pour le calcul de la distribution de Zipf (annexe A). Cela étant, il est possible que les messages soient trop court pour que ces fonctions donnent des résultats pertinents.

L'ensemble des messages sont récupérés dans la base mongo en une seule fois avec l'_id correspondant au hash. Le traitement est effectué sur chaque document en utilisant le module multiprocessing.

Base de données Les données des documents sont stockées dans des nouvelles tables PSQL initialisées au début du programme. Le nouveau schéma PSQL est montré dans la figure 8 Cette base a été choisie pour bénéficier des capacités d'insertion par lots et de jointure offertes par SQL. Plusieurs tables sont ajoutées en liaison avec nom de la fonction de traitement. Les tables ajoutées :

- features_hapax
- features_mots
- features_ponctuations
- features_zipf

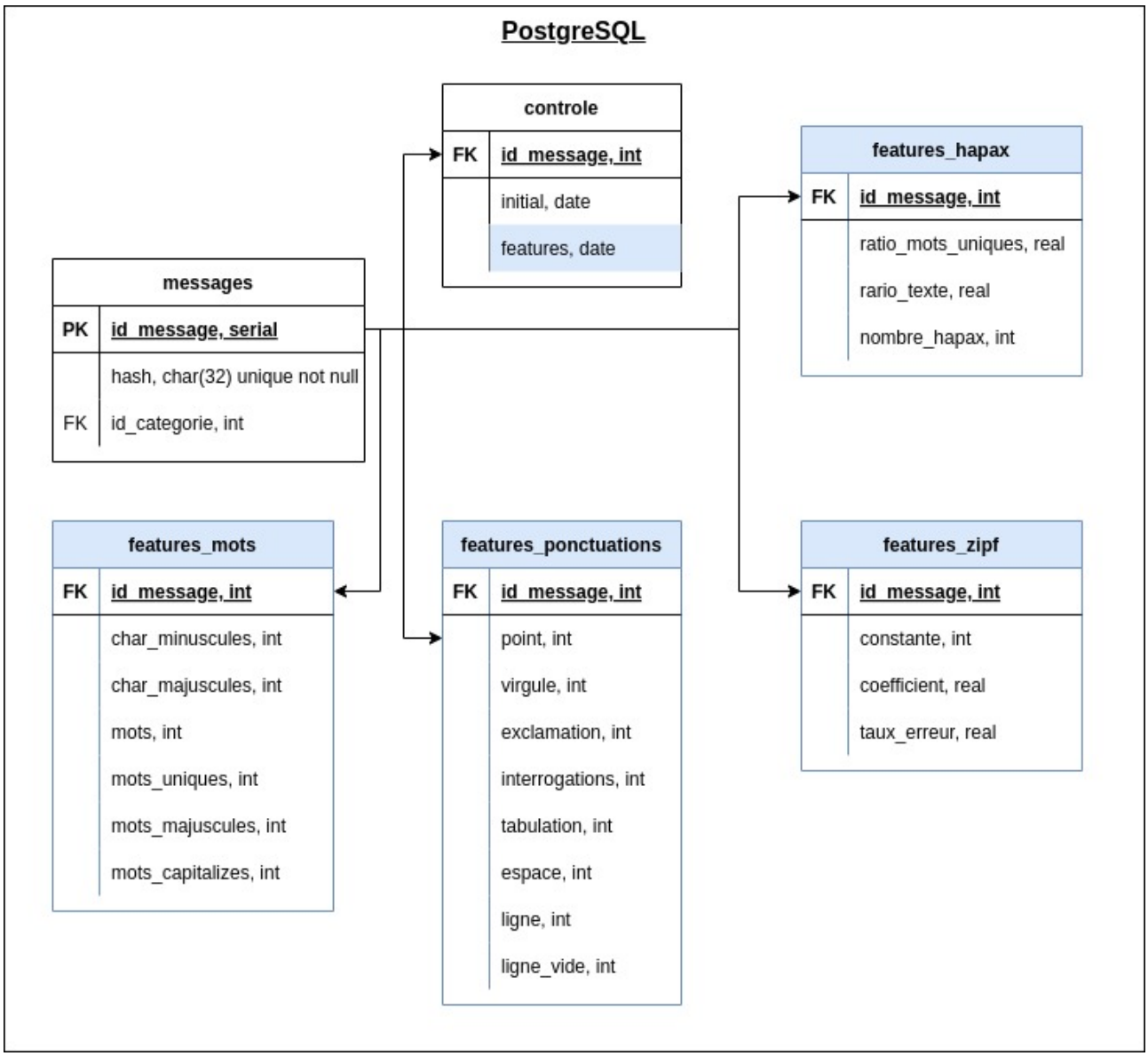


FIGURE 8 – Schéma de la base pour la recherche de caractéristiques

Résultats de la recherche de caractéristiques

TABLE 5 – Statistiques sur les données mots

	nom	ham	spam
mean	mots	164.09	348.35
	mots-uniques	97.36	168.36
q50	mots	81.0	163.0
	mots-uniques	65.0	111.0
q90	mots	298.0	611.0
	mots-uniques	181.0	299.6
max	mots	14297.0	11360.0
	mots-uniques	2778.0	2476.0

Le tableau 5 montre que les spam utilisent globalement plus de mots uniques. Il apparait que le nombre de mots par messages soit plus important dans les spam.

TABLE 6 – Statistiques sur les données char

	nom	ham	spam
mean	char-majuscules	42.93	231.46
	char-minuscules	729.21	1510.64
q50	char-majuscules	22.0	84.0
	char-minuscules	343.0	675.0
q90	char-majuscules	72.0	374.6
	char-minuscules	1296.2	2700.0
max	char-majuscules	5477.0	50543.0
	char-minuscules	68082.0	54406.0

Le tableau 6 montre une nette utilisation des caractères en majuscule dans les spam. Analogiquement cette représentation des majuscules est répercutée dans l'utilisation des mots contenant des majuscules comme montré dans le tableau 7.

TABLE 7 – Statistiques sur les données format-mots

	nom	ham	spam
mean	mots-capitalizes	24.12	60.59
	mots-majuscules	6.16	29.76
q50	mots-capitalizes	14.0	32.0
	mots-majuscules	3.0	10.0
q90	mots-capitalizes	39.0	124.3
	mots-majuscules	12.0	58.0
max	mots-capitalizes	2503.0	2238.0
	mots-majuscules	805.0	767.0

Les données des calculs de la distribution de Zipf (tableau 8) et de la recherche d'Hapax (tableau 9) semblent aller dans le sens de spam de plus grande taille. Cependant, ces données sont à prendre avec des pincettes, car au moins 10% des documents ont un coefficient de 1.3 qui est la limite haute pour la recherche de cette valeur.

TABLE 8 – Statistiques sur les données zipf

	nom	ham	spam
mean	coefficient	1.21	1.19
	constante	54.4	103.35
	taux-erreur	1.35	1.55
q50	coefficient	1.22	1.19
	constante	34.0	64.0
	taux-erreur	1.37	1.47
q90	coefficient	1.3	1.3
	constante	100.0	187.0
	taux-erreur	1.54	1.83
max	coefficient	1.3	1.3
	constante	2335.0	1871.0
	taux-erreur	3.35	3.51

TABLE 9 – Statistiques sur les données hapax

	nom	ham	spam
mean	nombre-hapax	74.99	112.11
	ratio-mots-uniques	0.84	0.73
	ratio-texte	0.69	0.52
q50	nombre-hapax	55.0	81.0
	ratio-mots-uniques	0.85	0.77
	ratio-texte	0.69	0.54
q90	nombre-hapax	134.0	207.0
	ratio-mots-uniques	1.0	0.9
	ratio-texte	1.0	0.78
max	nombre-hapax	1609.0	1695.0
	ratio-mots-uniques	1.0	1.0
	ratio-texte	1.0	1.0

Enfin les tableaux 10 et 11 montrent que le formatage du texte via le nombre de lignes, des espaces et de la ponctuation, est plus prononcé dans les spam.

TABLE 10 – Statistiques sur les données espace

	nom	ham	spam
mean	espace	184.06	411.8
	ligne	41.82	86.51
	ligne-vide	10.16	19.6
	tabulation	0.52	3.27
q50	espace	88.0	196.0
	ligne	29.0	52.0
	ligne-vide	8.0	13.0
	tabulation	0.0	0.0
q90	espace	335.2	806.4
	ligne	72.0	191.3
	ligne-vide	16.0	41.0
	tabulation	0.6	1.0
max	espace	30981.0	11436.0
	ligne	2524.0	1695.0
	ligne-vide	749.0	291.0
	tabulation	411.0	379.0

TABLE 11 – Statistiques sur les données ponctuation

	nom	ham	spam
mean	exclamation	0.48	5.88
	interrogation	0.75	1.38
	point	9.82	20.39
	virgule	8.92	16.58
q50	exclamation	0.0	3.0
	interrogation	0.0	0.0
	point	5.0	9.0
	virgule	4.0	6.0
q90	exclamation	2.0	12.0
	interrogation	2.0	4.0
	point	18.0	34.0
	virgule	16.0	30.0
max	exclamation	47.0	92.0
	interrogation	24.0	58.0
	point	848.0	643.0
	virgule	661.0	1268.0

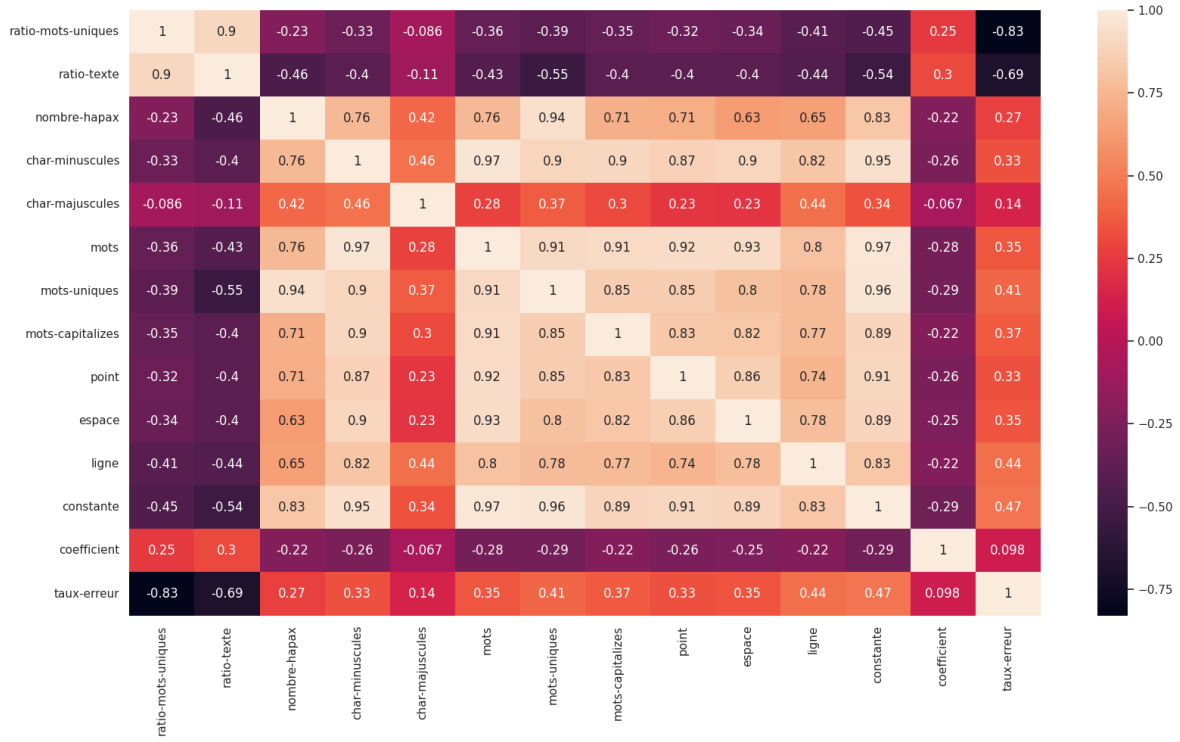
Les caractéristiques suivantes seront écartées, car il y a trop peu d'occurrence :
— ligne vide
— tabulation
— interrogation
— exclamation
— mots en majuscules

Matrice de corrélation La corrélation entre 2 variables permet de voir l'interdépendance entre elles. Il est alors possible de voir si leurs augmentations ou diminutions sont de nature à être liées. Cette relation peut dans certain cas montrer une relation de cause à effet.

Le calcul des coefficients présentés dans la matrice de la Figure 9 utilise la méthode de corrélation linéaire de Pearson. Il s'agit ici de calculer le ratio (r) de la covariance entre 2 variables par le produit de leurs écart-types.

$$\begin{aligned} r_{X,Y} &= \frac{cov(X,Y)}{\sigma_X \sigma_Y} \\ r_{X,Y} &= \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}} \end{aligned}$$

Une corrélation forte sera proche des bornes 1 et -1. Une corrélation faible sera proche de 0. L'idée finale étant de ne garder que des caractéristiques indépendantes les unes des autres



yy

FIGURE 9 – Matrice de corrélation des caractéristiques

Dans cette matrice, il est possible d'affirmer les éléments suivants. Le ratio d'hapax entre les mots uniques et l'ensemble de texte est fortement lié. Mais le nombre d'hapax semble indépendant des deux ratios dont il est une partie du calcul.

Le nombre de caractères en majuscule est indépendant avec la majeure partie des autres caractéristiques.

Les caractéristiques mots, char-minuscule, mots-uniques, mots-capitalizes, point, espace, ligne, constante sont très liées entre elles. Nous allons conserver ici le nombre de lignes. Car c'est la caractéristique la plus indépendante de celles déjà sélectionnées et qui possède le plus grand nombre moyen d'occurrences.

Les deux dernières caractéristiques (coefficient et taux d'erreur) sont également fortement liées et indépendante des autres valeurs. Nous allons conserver ici uniquement le coefficient. Il est plus indépendant que le taux d'erreur vis à vis du ratio de mots uniques.

Nous allons donc conserver pour la création du modèle les caractéristiques suivantes :
— ratio-mots-unique
— nombre-hapax
— char-majuscule
— espaces

2.2 Traitement du langage

Dans cette section, nous aborderons les traitements du langage naturel qui vont être employés afin de tenter de détecter des similitudes dans les thèmes abordés dans les mails de chaque catégorie. Nous allons transformer les messages en un dictionnaire de mot et de fréquence. Les données récoltées seront utilisées pour vectoriser les documents avec la méthode TF-IDF.

2.2.1 Lemmatisation

La lemmatisation d'un texte vise à réduire la taille d'un texte en ramenant chaque mot à la forme du mot présente dans le dictionnaire. Ce traitement permet donc de compter le nombre d'occurrences d'un mot sans se soucier de sa forme ou des modifications grammaticales appliquées lors de la rédaction du texte. A l'issue de ce traitement, nous serons en mesure de connaître les mots les plus utilisés dans les corps des mails et tenter de déterminer les thèmes les plus récurrents. Dans ce projet, nous allons utiliser le moteur de lemmatisation de *StandfordNLP*[2][4] Les traitements nécessaire pour arriver à une lemmatisation sont :

1. Tokenisation - Séparer les phrases en token. Ici chaque token correspond à un mot ou à une ponctuation.
2. Multi-Word Token Expansion - Ce traitement n'est pas nécessaire en anglais selon la documentation *Stanza*. Il permet d'étendre un token s'il correspond à une contraction de plusieurs terme. Par exemple en français le terme *du* sera transformé en *de le*.
3. Part of Speech Tagging - Cette opération vise à marquer chaque mot du texte avec sa position grammaticale dans la phrase qui le contient et va permettre d'effectuer le transfert vers le lexème correspondant.

L'initialisation de la pipeline de lemmatisation de Stanza se fait en appelant la classe *Pipeline* avec les arguments de langage et de processus dans l'ordre d'utilisation. A cette étape, j'ajoute 2 filtres. Le premier va me permettre de ne conserver que les mots avec des lettres. La deuxième étape permet de retirer les stopwords en anglais qui ne permettent pas d'analyser le sens du message.
La dernière étape avant la mise en base permet de compter la fréquence de chaque mot dans le texte. Le schéma 10 illustre l'enchainement des étapes de cette phase.

Pipeline de lemmatisation

```
1  import nltk
2  import stanza
3  from src.annexes import zipf
4
5  nltk.download("stopwords")
6  en_stopwd = set(stopwords.words('english'))
7  pipe = stanza.Pipeline(lang='en', processors='tokenize,mwt,pos,lemma')
8  pat = re.compile(r'\w+')
9
10 def lemmatise(message, stopwds, pipeline, pattern):
11     doc = pipeline(message)
12     lemma = [mot.lemma for phrase in doc.sentences for mot in phrase.words
       ]
13     return [lem.lower() for lem in lemma if re.match(pattern, lem) and lem
       .lower() not in stopwds]
14
15 zipf.freq_mot(lemmatise(value, en_stopwd, pipe, pat))
16
```

Ci-dessous un exemple des étapes du traitement sur un message de la base

```
>>> print(value)
Heres the hottest thing in DVDs. Now you can make a personal backup
copy of a DVD right onto CDR.  Our Hot new software easily takes you through
the steps to make a copy of your own DVDs.
NOW INCLUDED FOR FREE! Copy PLAYSTATION, MUSICMPs and all Software.
```

```
 Step by Step Interactive Instructions
 All Software Tools Included On CD
 No DVD Burner Required
 FREE Live Technical Support
  Day Risk Free Trial Available
 FREE Dvd Movie of your choice LIMITED TIME OFFER!
We have All the software you need to COPY your own DVD Movies.
This email has been screened and filtered by our in house OPTOUT system in
compliance with state laws. If you wish to OPTOUT from this mailing as well
as the lists of thousands  of other email providers please visit
ZKZmblanzxaDBmnTTIcorikgl

>>> lemmatise(value, en_stopwd, pipe, pat)
['hot', 'thing', 'dvd', 'make', 'personal', 'backup', 'copy', 'dvd', 'right', 'onto',
 'cdr', 'hot', 'new', 'software', 'easily', 'take', 'step', 'make', 'copy', 'dvd',
 'include', 'free', 'copy', 'playstation', 'musicmps', 'software', 'step', 'step',
 'interactive', 'instruction', 'software', 'tool', 'include', 'cd', 'dvd', 'burner',
 'require', 'free', 'live', 'technical', 'support', 'day', 'risk', 'free', 'trial',
 'available', 'free', 'dvd', 'movie', 'choice', 'limited', 'time', 'offer',
 'software', 'need', 'copy', 'dvd', 'movie', 'email', 'screen', 'filter', 'house',
 'optout', 'system', 'compliance', 'state', 'law', 'wish', 'optout', 'mailing',
 'well', 'list', 'thousand', 'email', 'provider', 'please', 'visit',
 'zkzmblanzxadbmntticorikgl']

>>> zipf.freq_mot(lemmatise(value, en_stopwd, pipe, pat))
{'hot': 2, 'thing': 1, 'dvd': 6, 'make': 2, 'personal': 1, 'backup': 1, 'copy': 4,
 'right': 1, 'onto': 1, 'cdr': 1, 'new': 1, 'software': 4, 'easily': 1, 'take': 1,
 'step': 3, 'include': 2, 'free': 4, 'playstation': 1, 'musicmps': 1,
 'interactive': 1, 'instruction': 1, 'tool': 1, 'cd': 1, 'burner': 1,
 'require': 1, 'live': 1, 'technical': 1, 'support': 1, 'day': 1, 'risk': 1,
 'trial': 1, 'available': 1, 'movie': 2, 'choice': 1, 'limited': 1, 'time': 1,
 'offer': 1, 'need': 1, 'email': 2, 'screen': 1, 'filter': 1, 'house': 1,
 'optout': 2, 'system': 1, 'compliance': 1, 'state': 1, 'law': 1, 'wish': 1,
 'mailing': 1, 'well': 1, 'list': 1, 'thousand': 1, 'provider': 1, 'please': 1,
 'visit': 1, 'zkzmblanzxadbmntticorikgl': 1}
```

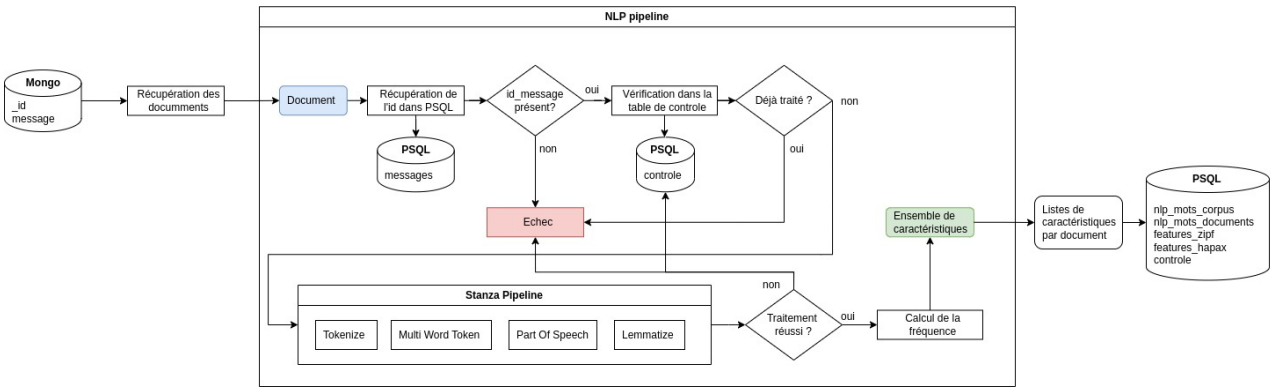


FIGURE 10 – Schéma du traitement NLP

Erreurs possibles lors du traitement NLP

— TypeError - Provient de la présence de données non textuelle dans le corps du mail (ex : MIME)
— OutOfMemoryError - Présence de mot beaucoup trop long pour être parsé.

Choix techniques Seules les librairies de Stanza ont été utilisées pour ce projet pour le traitement de texte. J'ai apprécié la simplicité d'intégration de la pipeline dans le code. De plus Stanza intègre une capacité de MWT qui aurait pu être utile dans le cas de traitement des mails en français. D'autres concurrents sérieux auraient pu être NLTK et SpaCy. Je n'ai pas eu le temps de les approfondir.

Dans cette phase l'utilisation du mutliprocessing n'a pas pu être mis en place à cause d'une RAM limitée qui aurait pu entrainer des échec de traitement. De plus, l'utilisation de torch.cuda par la pipeline Stanza nécessite trop d'ajustement au niveau du code pour être utilisées dans ce context.

Base de données Le schéma de la base de données PSQL pour cette phase est montrée dans la figure 11. La table de contrôle récupère 2 nouvelles colonnes

— nlp - date du traitement
— nlp_status - résultat OK, Type error, Memory error

Deux nouvelles tables sont ajoutées pour stocker les informations nécessaires pour la vectorisation

— nlp_mots_corpus - liste des mots présents après les phases de traitement
  — freq_corpus - occurrences du mot dans tout le corpus
  — freq_documents - nombre de documents ayant au moins une occurrence de ce mot
  — freq_ham - nombre de ham ayant au moins une occurrence de ce mot
  — freq_spam - nombre de spam ayant au moins une occurrence de ce mot
— nlp_mots_documents - liste d'association entre les mots et les documents.
  — occurrence - fréquence d'un certain mot dans un document particulier

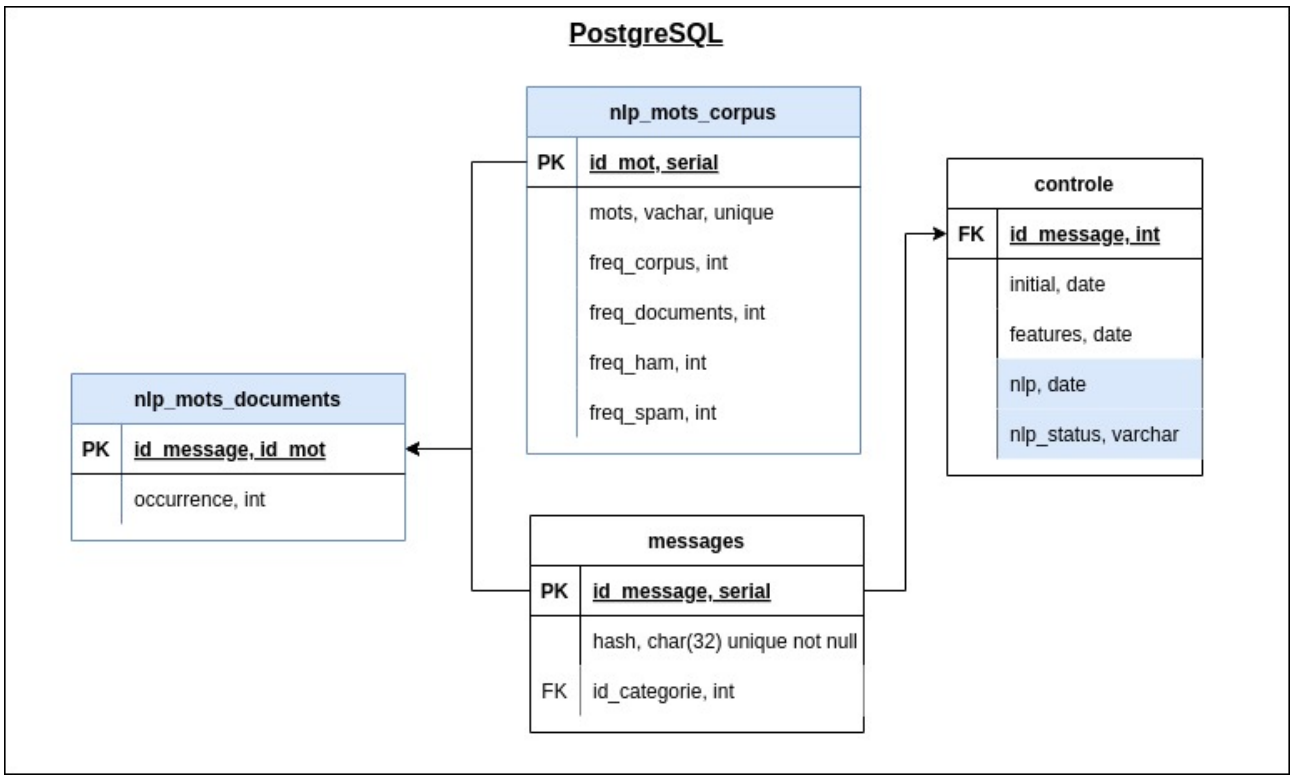


FIGURE 11 – Schéma de la base pour le traitement du langage

Résultats du traitement du langage A l'issue de la phase de lemmatisation, on obtient les données présentées dans le tableau 12 pour l'ensemble du corpus.

catégorie	mots	mots uniques
global	658524	39947
ham	355595	27787
spam	302929	19852

TABLE 12 – Nombre de mots après la lemmatisation

2.2.2 Vectorisation

La vectorisation d'un texte doit permettre de transposer les données textuelles en données numériques afin de pouvoir les utiliser dans des calculs statistiques ou dans les modèles d'apprentissage automatiques.
J'ai choisi d'utiliser une vectorisation TF-IDF[1] (Term Frequency-Inverse Document Frequency). Cette méthode se rapproche de la distribution de Zipf détaillée précédemment. En effet, le score d'un terme est dépendant de la fréquence dans le document et de la fréquence de ce terme dans l'ensemble du corpus. La formule utilisée est :

$$W_{i,j} = tf_{i,j} * log(\frac{N}{df_i})$$

Avec $tf_{i,j}$ la fréquence d'apparition du mot i dans le document j, N le nombre de documents dans le corpus et df_i le nombre de documents dans le corpus contenant le terme i.
Cette méthode permet d'attribuer un score plus élevé aux mots apparaissant plus dans un document que dans les autres documents du corpus. Nous pouvons ainsi conserver une forme de contexte d'utilisation. La figure 12 montre les étapes pour vectoriser l'ensemble des documents.

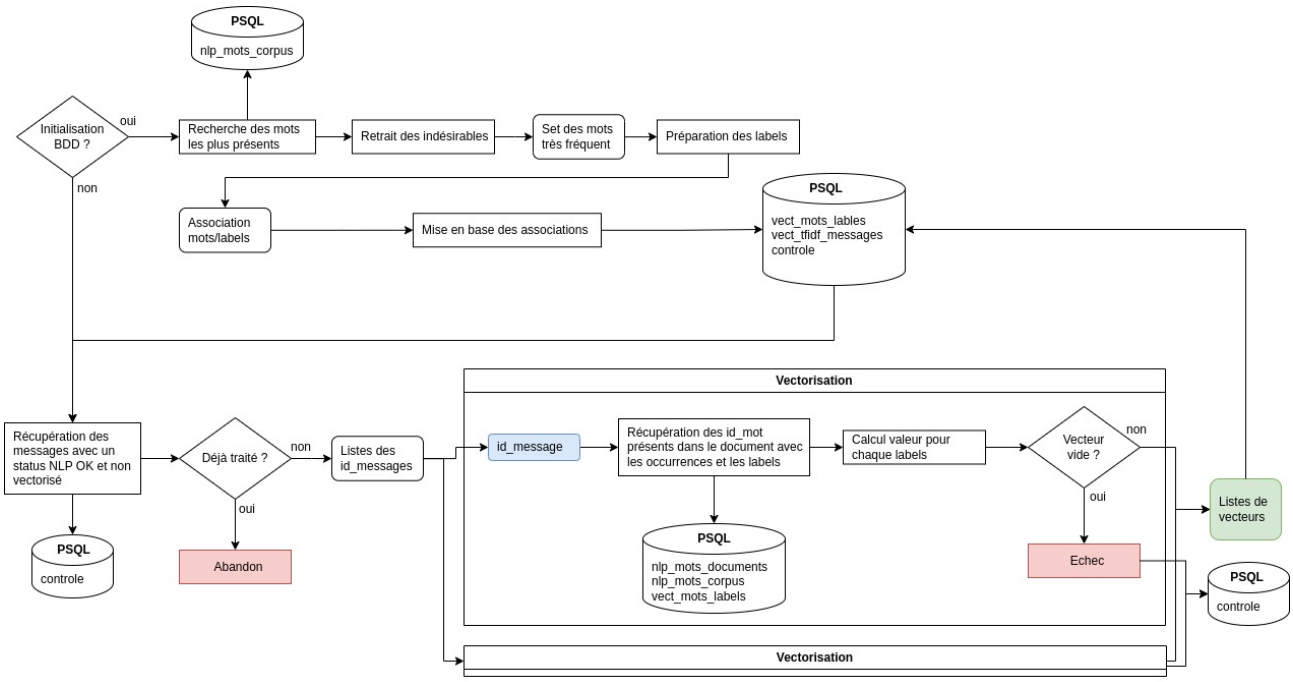


FIGURE 12 – Schéma de la vectorisation TFIDF

Une fois la phase de récupération des mots faites, il a fallu sélectionner quels mots seront utilisés pour vectoriser les documents. J'ai décidé de me limiter sur un ensemble de mots les plus courants dans le corpus sous plusieurs critères :

— Les plus grandes fréquences dans le corpus
— Les plus grandes occurrences par catégorie
— les occurrences les plus importantes dans les documents d'une catégorie que dans l'autre

Ces mots sont récupérés via les commandes SQL suivantes :

Commandes SQL pour récupérer les mots les plus présent

```
1  SELECT mot FROM nlp_mots_corpus ORDER by freq_corpus DESC;
2  SELECT mot FROM nlp_mots_corpus ORDER by freq_spam DESC;
3  SELECT mot FROM nlp_mots_corpus ORDER by freq_ham DESC;
4  SELECT mot FROM nlp_mots_corpus ORDER by freq_documents DESC;
5  SELECT mot FROM nlp_mots_corpus ORDER by freq_ham, freq_spam DESC;
6  SELECT mot FROM nlp_mots_corpus ORDER by freq_spam, freq_ham DESC;
7  SELECT mot FROM nlp_mots_corpus WHERE freq_ham >= 2*freq_spam ORDER BY
       freq_ham DESC;
8  SELECT mot FROM nlp_mots_corpus WHERE freq_spam >= 2*freq_ham ORDER BY
       freq_spam DESC;
9  SELECT mot, freq_ham/freq_spam as "ratio ham/spam" FROM nlp_mots_corpus
       WHERE freq_ham > 0 AND freq_spam > 0 ORDER BY "ratio ham/spam" DESC;
10 SELECT mot, freq_spam/freq_ham as "ratio spam/ham" FROM nlp_mots_corpus
       WHERE freq_ham > 0 AND freq_spam > 0 ORDER BY "ratio spam/ham" DESC;
11
```

Des *LIMIT X* sont ajoutés à la fin de chaque instruction pour que conserver que les X mots les plus importants. Certains mots très fréquents avec une connotation forte sur le dataset ont été retirés (spamassassinsightings, spamassassindevel) Le label de chaque mot est construit de la manière suivante, $m\_<id\_mot>$.

Chaque document sera vectorisé en fonction des mots retenus et uniquement avec les informations présentes dans les tables :

— nlp_mots_documents(occurrence) - fréquence du mot dans le document
— nlp_mots_corpus(freq_documents) - nombre de documents du corpus contenant ce mot
— controle(nlp_status) - count sur le nombre de documents ayant réussi le processus NLP

Choix techniques Le choix de la vectorisation TFIDF a été décidé par sa proximité avec la distribution de Zipf qui est transposée en python dans ce projet. L'idée principale de cette vectorisation était d'avoir un minimum de mots dans la base vectorielle. Initialement, j'avais fixé une limite à 200 pour les requêtes SQL. Cela m'a généré une base d'environ 1200 mots. Le problème est que j'ai 5 messages qui ont terminé en vecteur null à la fin de la vectorisation. En basculant cette limite à 500 j'obtiens une liste de 2940 mots et tous les documents sont vectorisés.

Le processus de vectorisation utilise également le module multiprocessing pour accélérer au maximum le traitement. L'insertion des vecteurs dans la base de données se fait itérativement par document.

Base de données Le schéma de la base de données pour la phase de vectorisation est montré dans la figure 13.

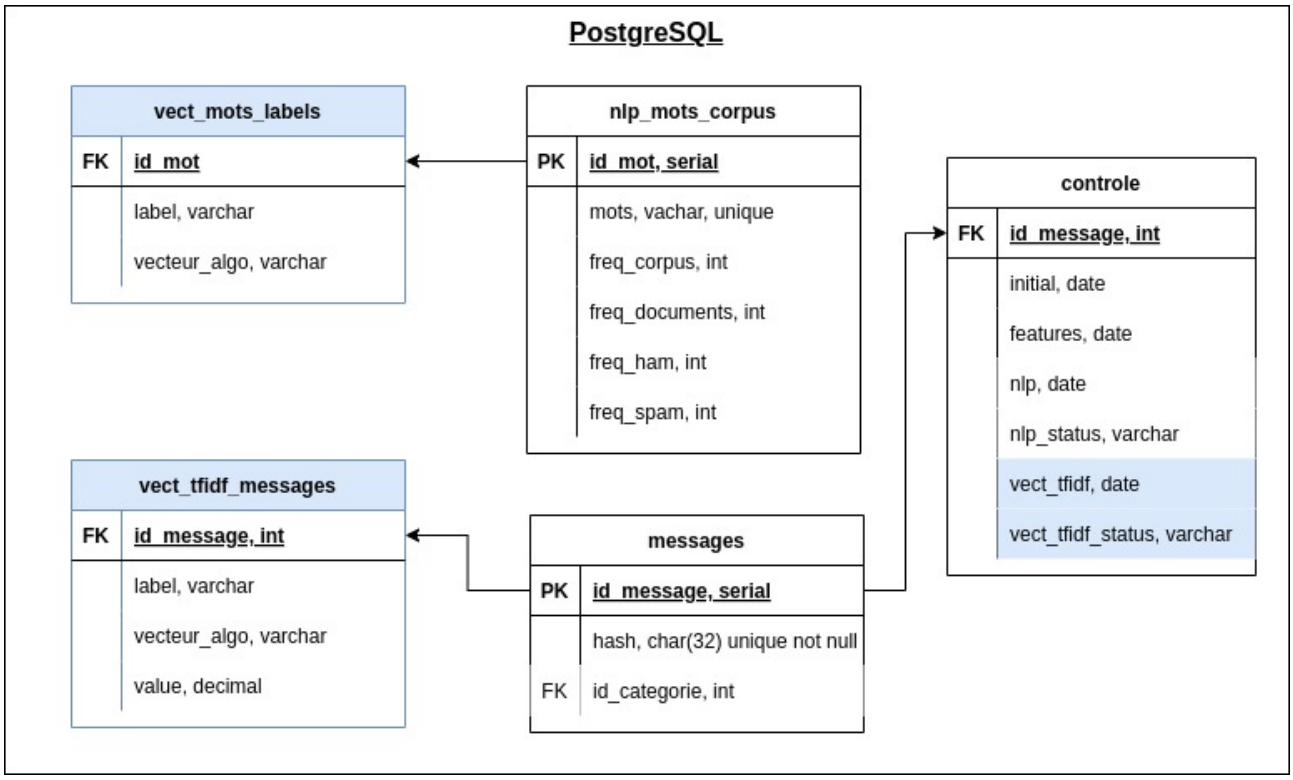


FIGURE 13 – Schéma de la vectorisation TFIDF

Initialement, la table contenant les vecteurs devait avoir comme colonne les labels de chaque mot. Cependant, PSQL limite ses tables à 1600 colonnes. Ce format aurait dû permettre une intégration plus simple et plus rapide un dataframe pandas pour le traitement via IA.
Les tables présentes dans la base sont :
— vect_mots_labels - contient la liste d'association entre les mots et les labels définis pour un type de vectorisation
— vect_tfidf_message - liste toutes les valeurs de la vectorisation par message et par label
  — id_message

— vecteur_algo
— label
— value - valeur tfidf pour le message et pour le label

Cette architecture est plus flexible pour l'ajout de nouvelle méthode de vectorisation. Seules les labels non null sont stockés dans la base. Cela implique que les labels manquants pour un message doivent être initialisés à 0 avant le début du traitement IA. La table *vect_mots_labels* doit permettre de retrouver la liste complète des labels valident pour un vecteur.

Résultat de la vectorisation La commande SQL ci-dessous va permettre de récupérer la liste des documents vectorisés avec leur catégorie ainsi que le nombre de dimensions utilisées avec la somme des dimensions de chaque vecteur.

Commande SQL pour lister les documents vectoriser

```sql
SELECT sq.document, cat.nom categorie, sq.dimensions, sq.somme
FROM (SELECT id_message document, COUNT(label) dimensions, SUM(value)
    somme
        FROM vect_messages
        WHERE vecteur_algo LIKE 'tfidf'
        GROUP BY document) as sq
JOIN messages me ON me.id_message = sq.document
JOIN categories ca ON me.id_categorie = ca.id_categorie;

```

Il est alors possible d'obtenir les distributions montrées dans la figure 14.

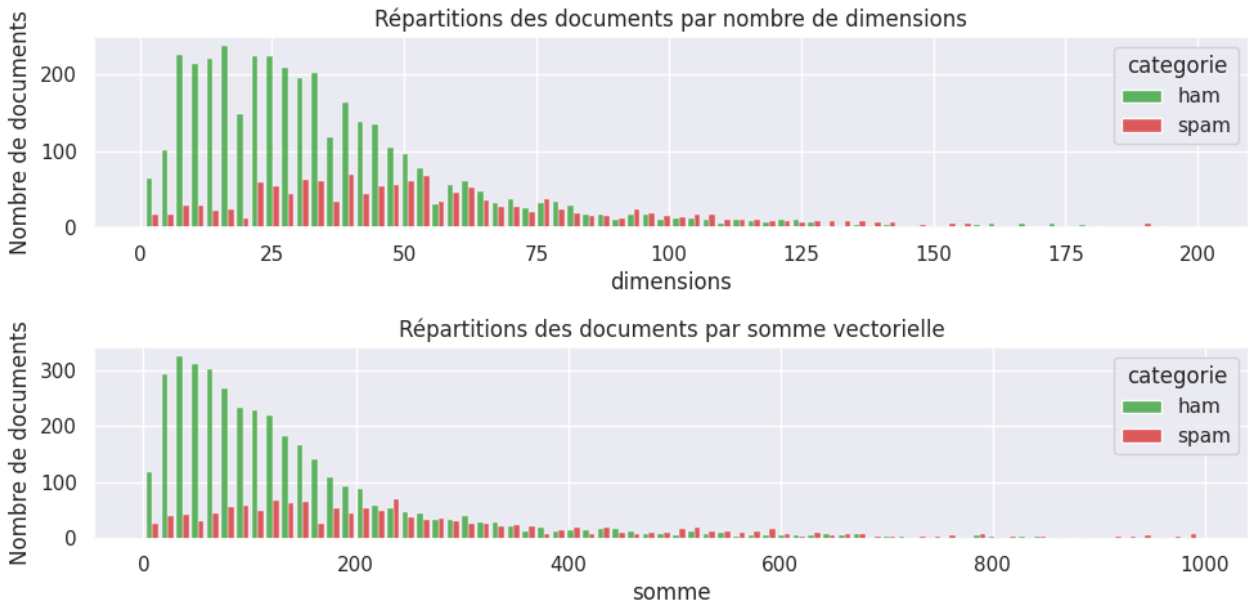


FIGURE 14 – Répartition des documents suite à la vectorisation

Le premier graphique montre la distribution de documents selon le nombre de mots utilisés pour les vectorisés. On voit que la grande majorité des documents est définis avec moins de 100 dimensions (mots). Le deuxième graphique présente une distribution de la somme de toutes les dimensions de chaque message. Dans les deux cas, on voit que les ham concentre la majeure partie des documents vers les bords gauches des tableaux et ont une courbe dégressive. La répartition des spam quant à elle semble plus homogène.

La commande SQL suivante va nous permettre de voir si certains mots du vecteur plus utilisés dans un type de document que dans l'autre. Les données sont représentées dans la figure 15.

Commande SQL pour lister les mots du vecteur

```sql
SELECT vml.label, nmc.freq_corpus, nmc.freq_ham, nmc.freq_spam
FROM vect_mots_labels vml
JOIN nlp_mots_corpus nmc ON vml.id_mot = nmc.id_mot
WHERE vml.vecteur_algo LIKE 'tfidf';

```

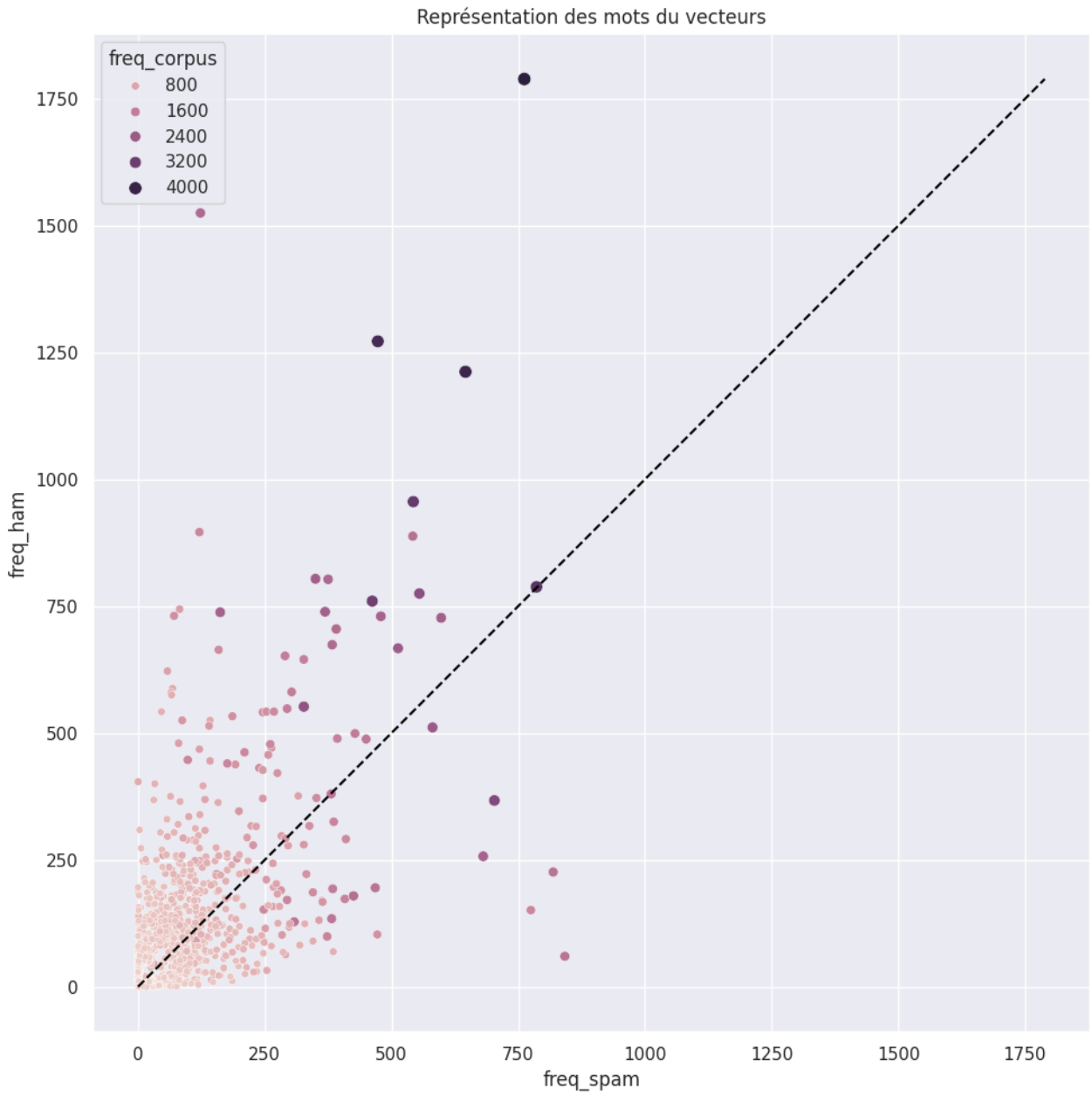


FIGURE 15 – Représentation de l'apparition des mots du vecteur

Chaque mot du vecteur est représenté par un point avec en abscisse le nombre de spam dans lesquels il est présent, et en ordonnée le nombre de ham. La fréquence d'apparition du mot dans le corpus est représenté par la couleur et la taille du point. Pour faciliter la lecture une

ligne droite $y = x$ a été ajoutée. Tous les points au-dessus de cette ligne seront plutôt associé aux ham et les mots en-dessous au spam. Les points sur cette ligne sont présents dans autant de document des deux catégories.
De manière générale, on remarque un léger déséquilibre du nombre de mots en faveur des HAM. Les mots à faible occurrence ont tendance à graviter autour de la ligne médiane, même si une bonne quantité n'est présente que dans les ham. Les mots a forte occurrence se retrouvent plus éloignés du milieu. Cela est potentiellement un avantage pour les étapes de catégorisation.

3 Modélisation

La dernière phase de ce projet est de réaliser un modèle d'intelligence artificielle capable de classifier les mails, en se basant sur les données extraites des phases précédentes. Nous allons tester ici 2 types de modèle et voir si l'un est plus performant qu'un autre.

Généralités

Random Tree Forest Une Random Tree Forest va se baser sur un ensemble d'arbre de décision pour réaliser la catégorisation. L'idée principale d'un arbre de décision est de sélectionner une variable dit de segmentation, qui va être utilisée pour couper l'ensemble de données en deux. Par défault scikit learn utilise l'indice de GINI pour déterminer cette variable. L'équation suivante est utilisée dans le *RandomTreeClassifier*

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

Q_m représente le nœud dans l'arbre de décision. p_{mk} est la proportion des éléments de la classe k dans le nœud. Le résultat de ce calcul va donner une idée de la dispersion des données au sein du groupe. Un indice Gini à 0 pour un nœud signifie que tous les éléments de ce nœud sont du même type. Un indice de 0.5, correspond au niveau maximum de distribution. Il y a autant d'éléments de chaque catégorie dans le groupe observé.

Pour chaque nœud l'algorithme va alors chercher la meilleure caractéristique pour séparer les données. Cette opération va se répéter jusqu'à arriver à un nœud pur ou à la profondeur maximale autorisée.

Les arbres sont créés en utilisant des échantillons aléatoires des données. La classification finale est décidé au résultat majoritaire entre tous les arbres.

Les paramètres principaux sont :
— n_estimator - nombre d'arbres - ici, on utilise la valeur par défaut 100
— max_depth - profondeur maximum de l'arbre - en se basant sur la distribution des mails dans les dimensions du vecteur (voir Figure 14), on va aller à une profondeur de 100

Support Vector Machine L'algorithme Support Vector Machine va chercher à trouver un hyperplan capable de séparer les données. Cet hyperplan doit avoir la plus grande marge possible pour être capable de généraliser le plus possible.
Plusieurs variables sont paramétrables pour un modèle SVM. L'utilisation de *GridSearchCV* va permettre de fournir une liste de valeur possible pour chaque paramètre et de sélectionner association la plus performante. Les paramètres donnés à *GridSearchCV* pour évaluation sont :

Kernel Le Kernel est fonction noyau utilisée pour transformer les données et trouver un hyperplan linéaire. Dans notre cas, on n'utilisera que le noyau *rbf* (Radial Basis Function). Ce mode semble être plus adapté à nos données qu'un noyau linéaire ou polynomial.

C Ce paramètre va permettre de donner une tolérance à l'erreur plus grande lors de phase d'apprentissage. Cela va permettre d'avoir des marges plus grandes autour de l'hyperplan, et d'avoir une meilleure capacité de généralisation. Cette opération se fait en attribuant un certain poids aux erreurs. Plus le paramètre C va être petit plus les marges seront grandes.
Valeurs utilisées - $[0.1, 1, 5, 10, 50, 100]$

Gamma Ce paramètre n'est utilisé qu'avec les noyaux non linéaires. Il permet de gérer la distance pour que 2 points soient considérés comme faisant partie d'un même groupement. Avec une valeur élevée de Gamma, il faut que ces 2 points soient très proche. A contrario, une valeur faible les observations seront réparties dans des groupes quasi linéaires.
Valeurs utilisées - $[0.0001, 0.001, 0.005, 0.01, 1, 10]$

Normalisation La normalisation doit permettre de mettre à la même échelle toutes les caractéristiques d'un jeu de données. Nous avons dans notre dataset une grande disparité d'échelle et de distribution. La normalisation Standard, disponible dans scikit learn, va transformer chaque caractéristique pour que la moyenne soit égale à 0 avec un écart type de 1. Cette opération doit permettre de faciliter le traitement par les algorithmes d'apprentissage automatique.

3.1 Création des modèles

Cette section va s'intéresser aux étapes de la création des modèles.

Phase préparatoire L'ensemble des données (vecteur tfidf et caractéristiques retenues) est récupérée depuis la base Postgres et stockée dans un dataframe Pandas. L'ensemble des caractéristiques du dataset est normalisé pour harmoniser les échelles.

Trois sous-ensembles seront utilisés :
- Vecteur TFIDF + caractéristiques (nombre, url, nombre_hapax, ratio_hapax_uniques, majuscules, espaces)
- Vecteur TFIDF
- Id_categorie - correspond à la table de résultat

Ces jeux de données sont séparés en ensemble d'entrainement (75%) et de test (25%) aléatoirement.

L'identifiant de la catégorie *ham* est également récupéré pour être utilisé comme résultat positif.

Entrainement Les deux types de modèles Random Tree Forest (rtf) et Support Vector Machine (svm) sont entrainé avec les ensembles vecteur seul et avec addons.

Entrainement des modèles

```
1  alg_decision_tree = RandomForestClassifier(n_estimators=100, max_depth
       =100, n_jobs=-1)
2   model = alg_decision_tree.fit(x_train, y_train)
3   ...
4  alg_svm = svm.SVC()
5  svm_params = {'kernel': ['rbf'],
6                'gamma': [0.0001, 0.001, 0.005, 0.01, 1, 10],
7                'C': [0.1, 1, 5, 10, 50, 100]}
8  hyper_params_grid = GridSearchCV(alg_svm, svm_params, cv=2, scoring='
       accuracy', n_jobs=-1)
9  hyper_params_models = hyper_params_grid.fit(x_train, y_train)
10 best_svm = hyper_params_models.best_estimator_
11
```

La recherche du modèle SVM nécessite l'utilisation de GridSearchCV pour trouver la meilleure combinaison entre les différents paramètres possibles.

Evaluation des modèles L'évaluation des modèles générés est faites avec les données de test mises de cotés lors de la phase de préparation.

Evaluation des modèles

```
predictions = model.predict(x_test)
precision, recall, fscore, _ = score(y_test, predictions, pos_label=
    pos_label, average='binary')
...
predictions = best_svm.predict(x_test)
precision, recall, fscore, _ = score(y_test, predictions, pos_label=
    pos_label, average='binary')

```

Les métriques utilisées sont :

- Precision (P) - ici Ham correctement classifiés
- Recall (R) - ici Spam correctement classifiés
- Accuracy - Documents correctement classifiés sur l'ensemble du jeu de test
- Fscore - Moyenne harmonique utilisée dans les classes sont déséquilibrées - $2(P \cdot R)/(P + R)$

A l'issue de cette phase d'évaluation, les modèles sont sauvegardés en utilisant le module pickle.

3.1.1 Résultat

```
$ ./errol.py train rtf svm
22:32:24,888 : Initialisation du programme pour la phase train - OK
22:32:24,888 : Entrainement des modèles
22:32:24,888 : Récupération des caractéristiques
22:32:26,068 : Nombre de messages: 5333 - Nombre de caractéristiques: 2951
22:32:28,439 : ID pour les ham - 1
22:32:28,439 : RandomTree Forest avec les vecteurs tfidf et caractéristiques
22:32:28,868 : RandomTree Forest avec les vecteurs tfidf seulement
22:32:29,314 : Support Vector Machine avec les vecteurs tfidf et caractéristiques
22:38:44,097 : Support Vector Machine avec les vecteurs tfidf seulement
22:45:06,374 : rtf_vect_feat - Prec: 0.981 - Rec: 0.996 - Acc: 0.984 - Fsc: 0.989
22:45:06,388 : rtf_vect_only - Prec: 0.981 - Rec: 0.981 - Acc: 0.973 - Fsc: 0.981
22:45:06,402 : svm_vect_feat - Prec: 0.984 - Rec: 0.991 - Acc: 0.982 - Fsc: 0.987
22:45:06,430 : svm_vect_only - Prec: 0.982 - Rec: 0.991 - Acc: 0.981 - Fsc: 0.986

$ du -h models/*
5,9M    models/rtf_vect_feat.pickle
6,6M    models/rtf_vect_only.pickle
18M     models/svm_vect_feat.pickle
19M     models/svm_vect_only.pickle
```

Constatations

La première constatation concerne le temps d'entrainement. La génération des forets est quasiment instantanée même avec 100 arbres et une profondeur de 100. L'entrainement des modèles SVM via GridSearchCV a nécessité environ 7 minutes dans les deux cas et consommant 100% des 12 CPU installé sur ma machine.

La taille des modèles sauvegardés est également très différente. Les forêts sont généralement plus petite que les SVM. De plus, les modèles ayant été entrainés avec les données complètes (tfidf + addons) sont plus léger que les modèles analogues entrainés uniquement avec le vecteur tfidf.

Concernant les métriques à proprement parler, les résultats peuvent varier en fonction du découpage aléatoire lors de la création des sets d'entrainement et de tests. Pour les 4 modèles le FScore est de 98% et la différence se joue au dixième. Cet entrainement semble montrer que l'ajout des caractéristiques (addons) a une influence positive sur la classification. Cependant, cela n'a pas été systématiquement le cas lors de mes tests précédents. Les valeurs de tests d'une seconde execution m'ont donné les résultats suivants :

```
rtf_vect_feat - Precision: 0.975 - Recall: 0.997 - Accuracy: 0.980 - Fscore: 0.986
rtf_vect_only - Precision: 0.985 - Recall: 0.994 - Accuracy: 0.985 - Fscore: 0.989
svm_vect_feat - Precision: 0.966 - Recall: 0.995 - Accuracy: 0.971 - Fscore: 0.980
svm_vect_only - Precision: 0.976 - Recall: 0.996 - Accuracy: 0.980 - Fscore: 0.986
```

Dans tous les cas, le fscore avoisine les 98%. Je pense qu'une validation pourrait être utile avec des mails qui n'ont pas été traité lors des étapes précédentes (NLP et autres).

Conclusion

Avec ce projet nous avons vu le montage d'une infrastructure basée sur docker pour le stockage des informations récoltées et générées lors des différentes phases. Les fonctions développées pour les bases SQL et NoSQL sont utilisées dans toutes les phases de traitement.

Faute de sources, il n'a pas été possible de mettre en place une automatisation pour la récupération des mails. Il a été nécessaire de les récupérer manuellement dans d'autres projet du même type.

Le traitement initiale des mails est totalement automatisé grâce à Python et aux modules email et re. Ce nettoyage a permis de ne conserver que l'essentiel des corps de message tout en écartant les mails corrompu, sans texte, ou avec un mauvais encodage.

Pour agrémenter notre jeux de données la recherche de caractéristique s'est concentré sur la forme des textes ainsi que sur la présence d'éléments non textuel. Les résultats de cette recherche n'a pas été aussi concluant que ce qui était espéré.

La phase de traitement du langage visait à réduire les textes au minimum en tentant de conserver un maximum de sens. Pour cela, la lemmatisation semblait la meilleure option. Le modèle neuronale développer par StandfordNLP à permis de réaliser cette opération.

Enfin la vectorisation a été réalisée en utilisant la méthode TF-IDF (Term Frequency - Inverse Document Frequency). Cette méthode est en accord avec l'idée d'analyse les mails en se basant sur la présence de certains mots. L'ensemble des documents pu être généré mais des ajustements ont du être fait pour ne pas avoir de vecteur null.

Le tableau 13 montre l'évolution du nombre de document et des mots au fur et à mesure des phases du projet

Phase/Etape	nombre de documents	nombre de mots	nombre de mots uniques
Récolte	5798	2385120	262614
Transformation	5658	1222793	99772
Sauvegarde	5333	1163550	99772
Traitement du langage	5333	658524	39947
Vectorisation	5333	453068	2942

TABLE 13 – Problèmes possibles avec la mise en base

L'utilisation d'une Random Tree Forest a permis d'entrainer un modèle avec un résultat de 98% sur la classification des document. Cependant, il convient de valider ce résultat sur des messages n'ayant encore jamais été vu par Errol

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 :

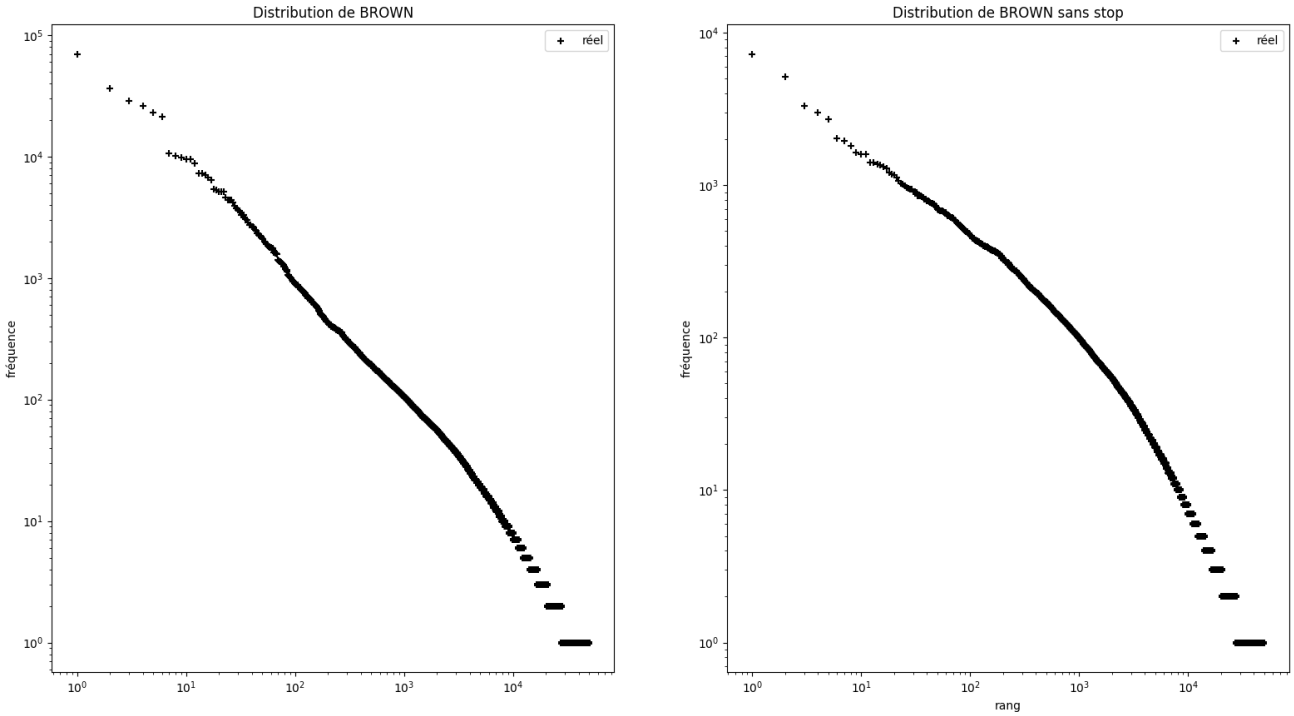


FIGURE 16 – Distribution de Zipf pour les deux corpus

— 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.

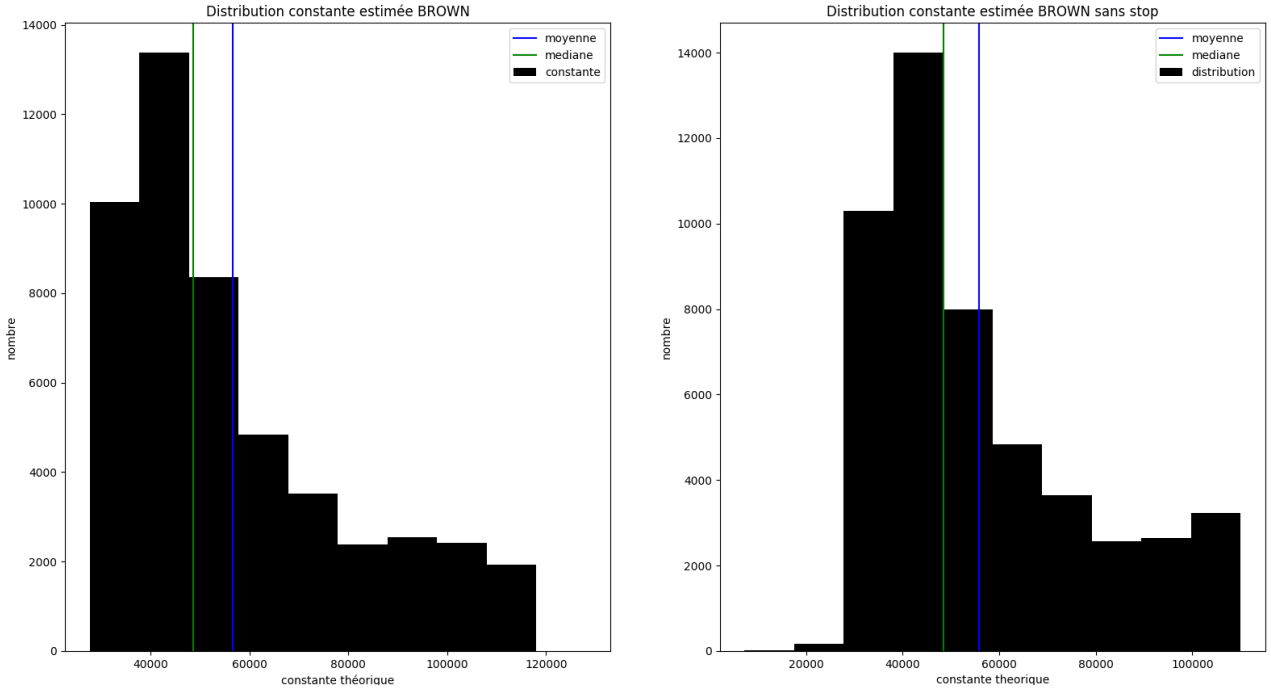


FIGURE 17 – Distribution des constantes théoriques pour les deux corpus

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     return constante / (rang ** coef)
3
4 def cout(l1, l2, methode):
```

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

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

La recherche du coefficient nous retourne les éléments suivants :

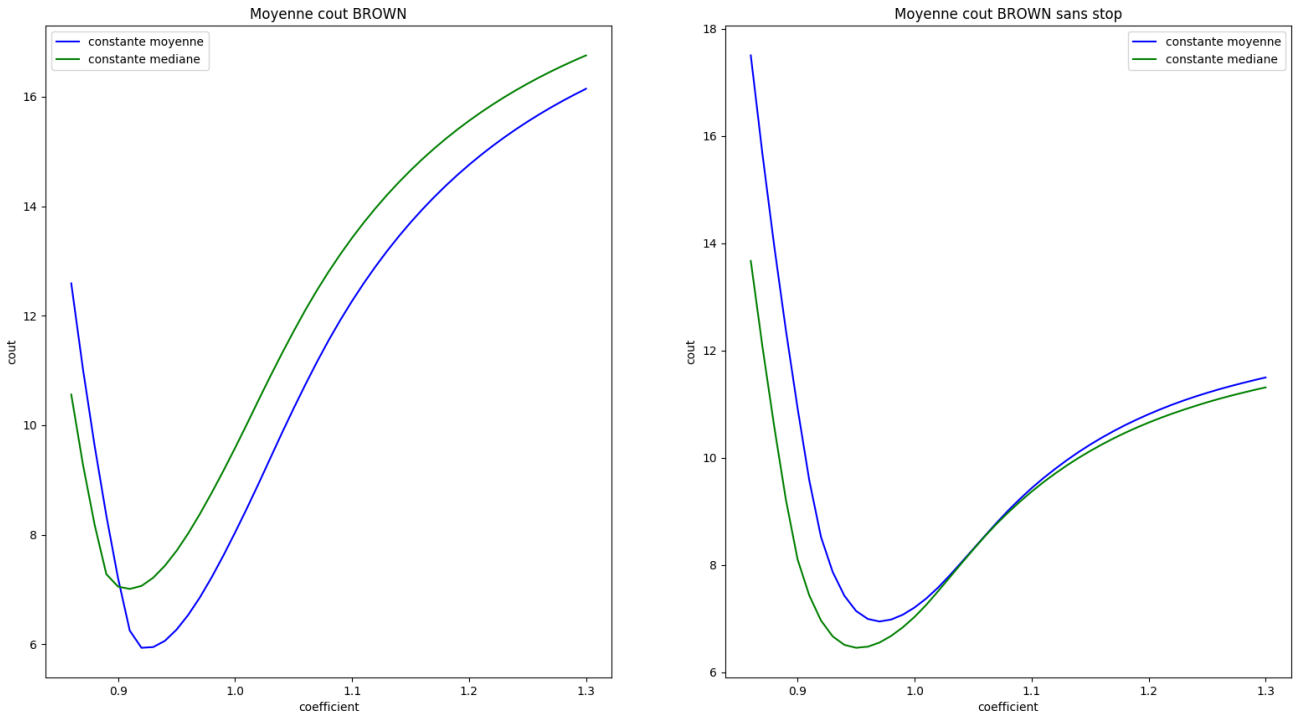


FIGURE 18 – Coût absolu moyen par coefficient

— 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.

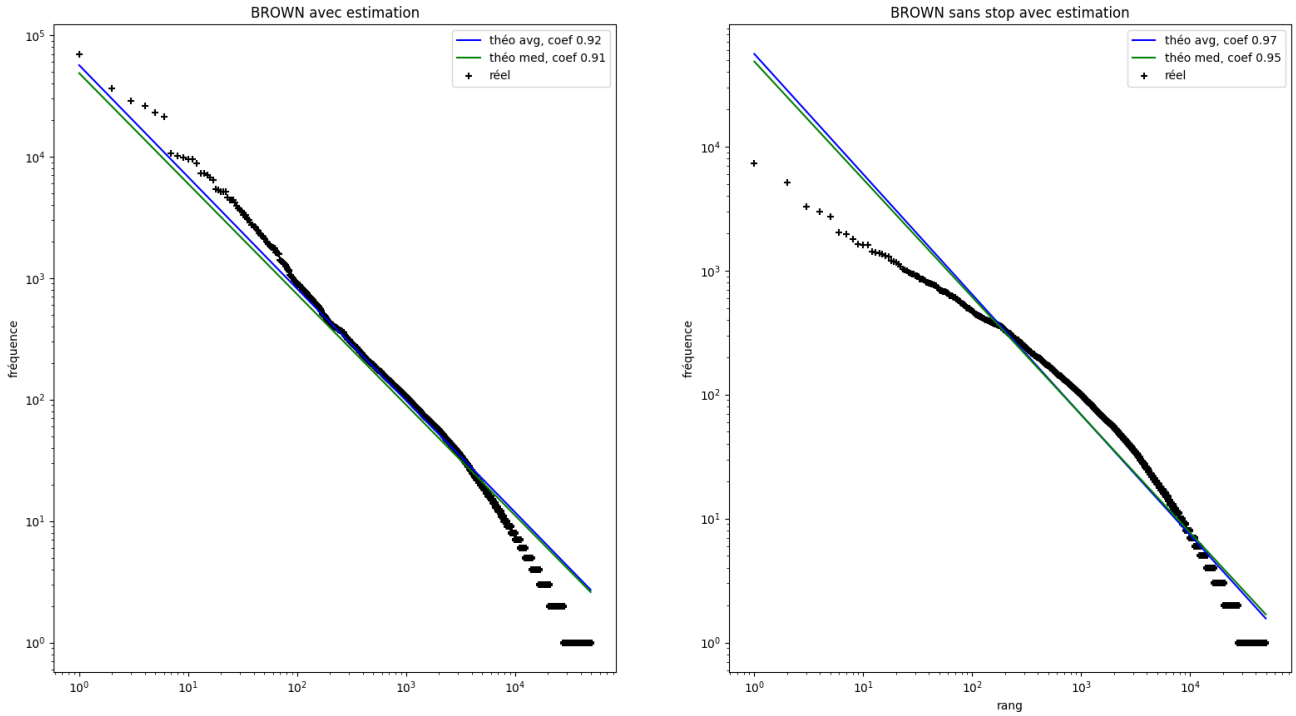


FIGURE 19 – Distribution de Zipf avec les estimations

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 :

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

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 14 – 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 15 – 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 16 – 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 17 – 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[3]. 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.

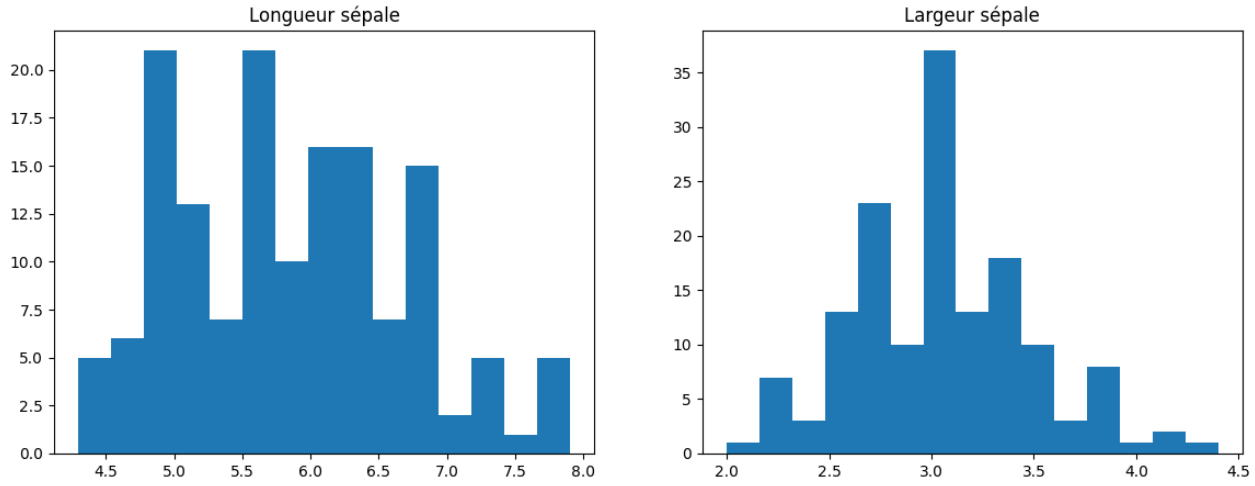


FIGURE 20 – Distribution des longueurs et largeurs des sépales

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

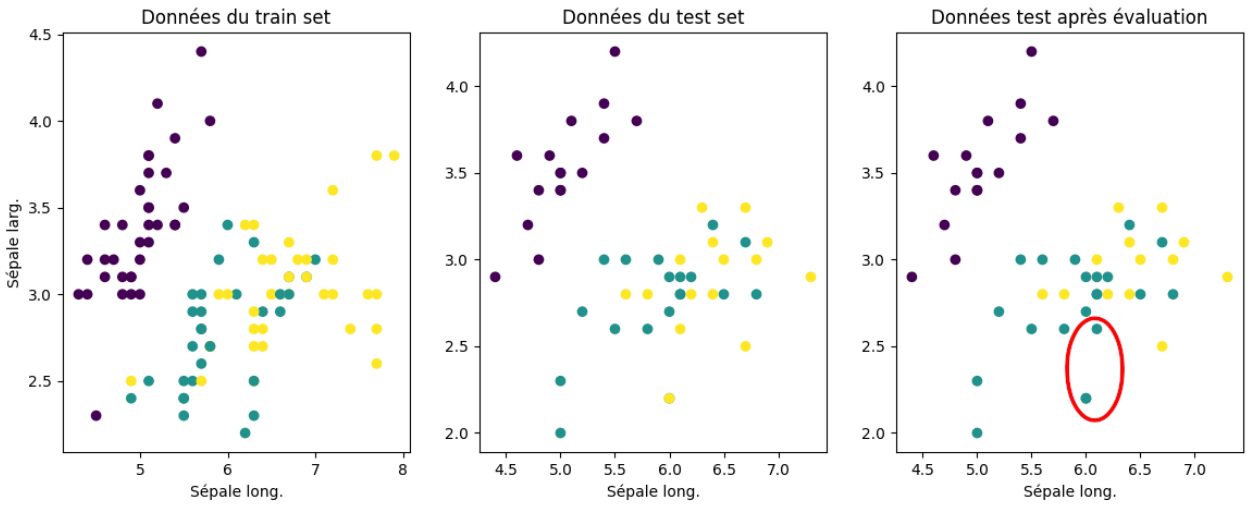


FIGURE 21 – Représentation des données

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]

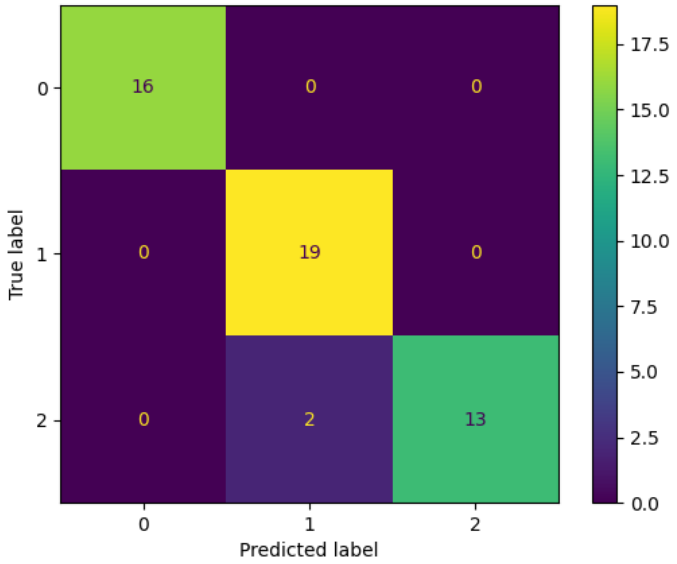


FIGURE 22 – Matrice de confusion

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] Madjid Khichane. *Le Machine Learning avec Python*. 2021.

[2] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[3] 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.

[4] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza : A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics : System Demonstrations*, 2020.

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)
— Spam generated by LLM, fichier csv avec des mail généré par IA https://www.kaggle.com/datasets/trainingdatapro/generated-e-mail-spam (consulté le 06/08/2024)

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)

Random Tree Forest Le modèle Random Tree Forest est utilisé dans la section 3
— Formules mathématiques utilisées dans SciKit learn [en ligne] https://scikit-learn.org/stable/modules/tree.html#tree-mathematical-formulation (consulté le 06/08/2024)

E Code Source

E.1 GitHub

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