Projet L3
Ingénierie des langues
Fouille de données

GOEHRY Martial
16711476

11 juin 2023

Table des matières

Introduction

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

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

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

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

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

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

- Phase 1 : Récupération des données
- Phase 2 : Analyse des caractéristiques
- Phase 3 : Construction du modèle

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 un index ElasticSearch et le stockage du nombre de liens dans une base PSQL.

Phase 2 La phase 2 vise à générer les données statistiques et numériques à partir des corps de mails. A l'issue de cette phase, il est alors possible d'analyser manuellement les statistiques générés.

Phase 3 La phase 3 regroupe tous 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 et au vu du temps que j'ai pris pour réaliser ce projet, l'ensemble des étapes est automatisé avec Python. Seule la récolte initiale des mails a été réalisée à la main.

Le schéma ci-dessous donne une vue synthétique des étapes du projet

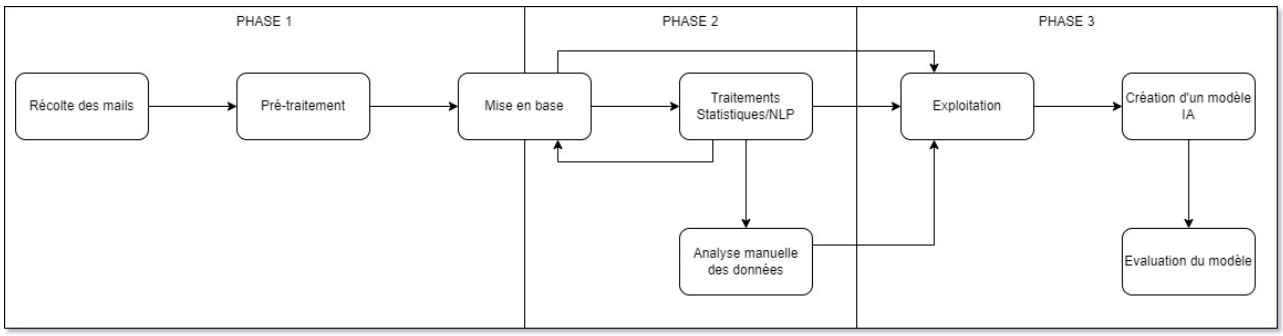


FIGURE 1 – Schéma des grandes étapes

1 Phase 1 : Récupération des données

La suite d'opération de cette phase vas permettre d'extraire un maximum d'information d'un email en essayant de ne dénaturer ni le fond ni la forme. Il pourra ensuite être stocké avec sa catégorie d'appartenance. Durant cette phase nous allons également initialiser les bases de données en créant les index (ES) et les tables (PSQL et SQLITE).

Ci-dessous le schéma général de cette phase.

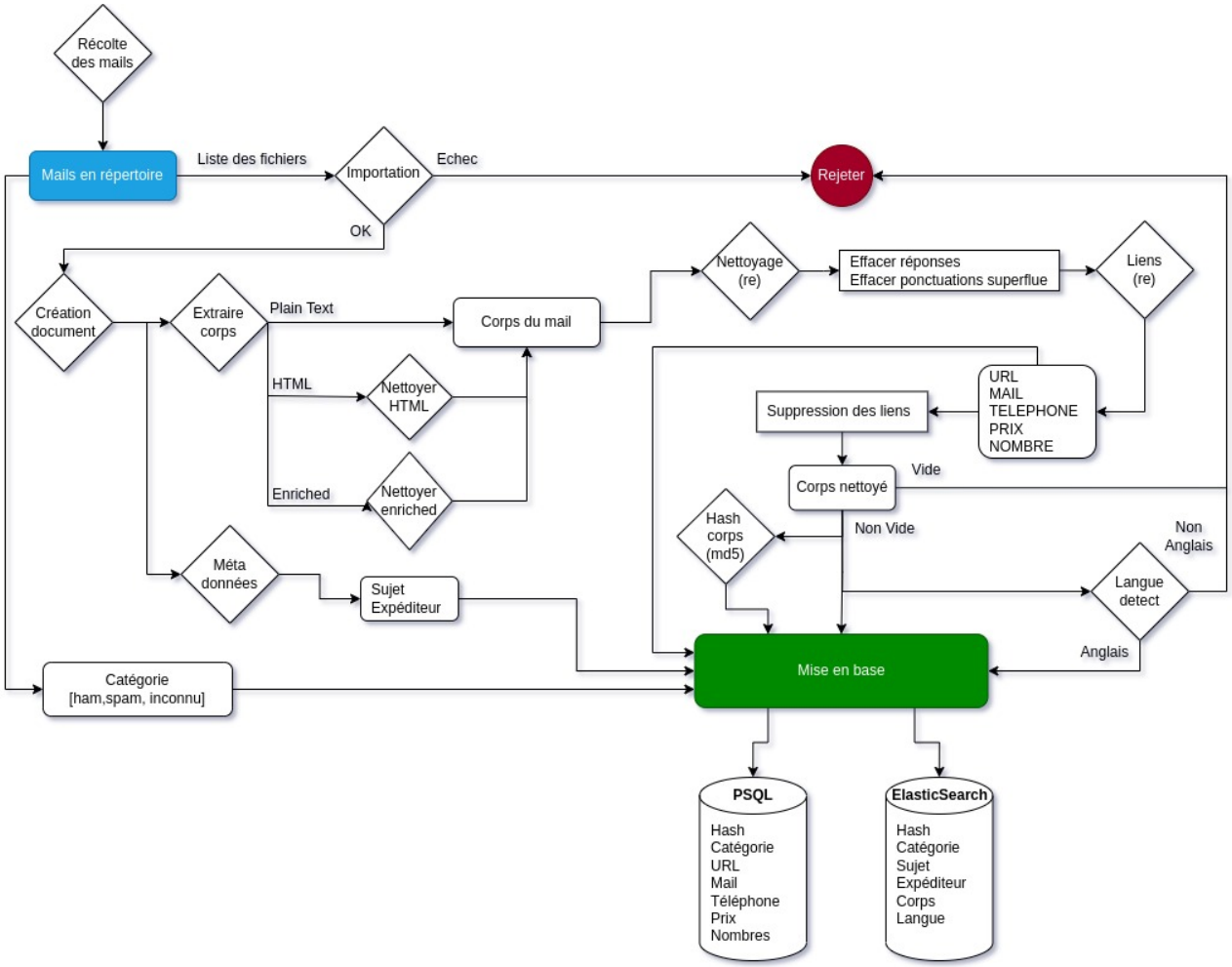


FIGURE 2 – Schéma des étapes de la phase 1

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 : F.1)
- Dataset SpamAssassin (voir : F.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 SpamAssasin afin de pouvoir vérifier les résultats de l'analyse.

Téléchargement des données Le téléchargement du dataset Enron est possible a 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

1.2 Pré-traitement

Les étapes de pré-traitement regroupent toutes les étapes et actions réalisées avant la mise en base. L'objectif de ces étapes est d'extraire le message en retirant les métadonnées du mail. Il va être possible d'effectuer certain traitement de nettoyage et de récupération d'informations primaires.

Les manipulations de messages dans Python se font principalement à l'aide du module python natif *email*.

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 import_from_file(chemin):
2      try:
3          with open(chemin, 'rb') as data:
4              msg = message_from_binary_file(data, policy=policy.default)
5              return msg
6
7      except FileNotFoundError:
8          print("Fichier : '{}' non trouve".format(chemin), file=sys.stderr)
9          return None
```

1.2.2 Extraction des 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',
3                         'gb2312_charset', 'chinesebig5', 'big5']
4      body = ""
5
6      if msg.is_multipart():
7          for part in msg.walk():
8              if not part.is_multipart():
9                  body += extract_body(part)
10         return body
11
```

```
12      if msg.get_content_maintype() != 'text':
13          return ""
14
15      if msg.get_content_charset() in refused_charset:
16          return ""
17
18      if msg.get_content_subtype() == 'plain':
19          payload = msg.get_payload(decode=True)
20          body += payload.decode(errors='ignore')
21
22      if msg.get_content_subtype() == 'html':
23          payload = msg.get_payload(decode=True)
24          body += nettoyage.clear_html(payload.decode(errors='ignore'))
25
26      if msg.get_content_subtype() == 'enriched':
27          payload = msg.get_payload(decode=True)
28          body += nettoyage.clear_enriched(payload.decode(errors='ignore'))
29
30      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. Afin de 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.

Nettoyage des réponses

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

Suppression des ponctuations Afin de 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

```
1 def clear_ponctuation(texte):
2     pattern_ponct = re.compile('[*#\\-_=:;<>\\[\\]"\'~)(|/$+}{@%&\\\]',
      flags=re.MULTILINE)
3     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

```
def clear_enriched(texte):
    pattern = re.compile('<.*>')
    return re.sub(pattern, '', 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 donc modifier ces liens qui seront comptabilisés avant d'être retirés du texte.

Nettoyage des liens

```
def change_lien(texte, liens):
    pattern_mail = re.compile('[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+')

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

    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

```
def change_nombres(texte, liens):
    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+')
```

```
    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
```

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 F.2).
Je conserve dans les données à mettre en base le langage détecté avec l'idée de pouvoir traité plusieurs langues dans le futur.

La détection de la langue est réalisée juste avant la mise en base dans l'index ElasticSearch.

Création d'un document

```
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 Les sections suivantes présentent 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

```python
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.
```

1.2.4 Mise en base

Cette section détaille les éléments relatifs à la mise en base des informations récoltées. Dans ce projet, j'utilise 2 moteurs de bases de données pour stocker les données des mails.

1. un index ElasticSearch pour faire le stockage des données textuelles
2. une base PostgreSQL pour le stockage des données numériques

J'utilise des conteneurs *docker* pour héberger les services de bases de données. L'utilisation des conteneurs me permet de partager plus facilement mes configurations et limite les erreurs d'installations.

Pour chaque mail récolté le programme de la phase 1 va générer un *document* avec les informations suivantes :

— hash - signature md5 du texte nettoyé
— catégorie - Ham, Spam ou Inconnu
— sujet - correspond à l'objet du mail
— expéditeur - adresse mail
— corps - corps du mail nettoyé
— langue - la langue détectée du mail (en)
— liens - données non textuelles extraites du corps :
  - URL - liens URL
  - Mail - adresses mail
  - Téléphone - numéros de téléphone
  - Prix - nombres avec un symbole de devise
  - Nombres

Chaque document va générer une entrée dans la base ElasticSearch et une entrée dans la base PostgreSQL.

Création d'un document

```
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
```

```
13
14      if lang != 'en':
15          return None
16
17      if categorie.lower() not in ['spam', 'ham']:
18          categorie = 'inconnu'
19
20      doc = {
21          'hash': hashlib.md5(corp.encode()).hexdigest(),
22          'categorie': categorie.lower(),
23          'sujet': sujet,
24          'expediteur': expediteur,
25          'message': corp,
26          'langue': lang,
27          'liens': liens
28      }
29      return doc
30
```

Ci-dessous le schéma des bases de données avec les relations entre elles.

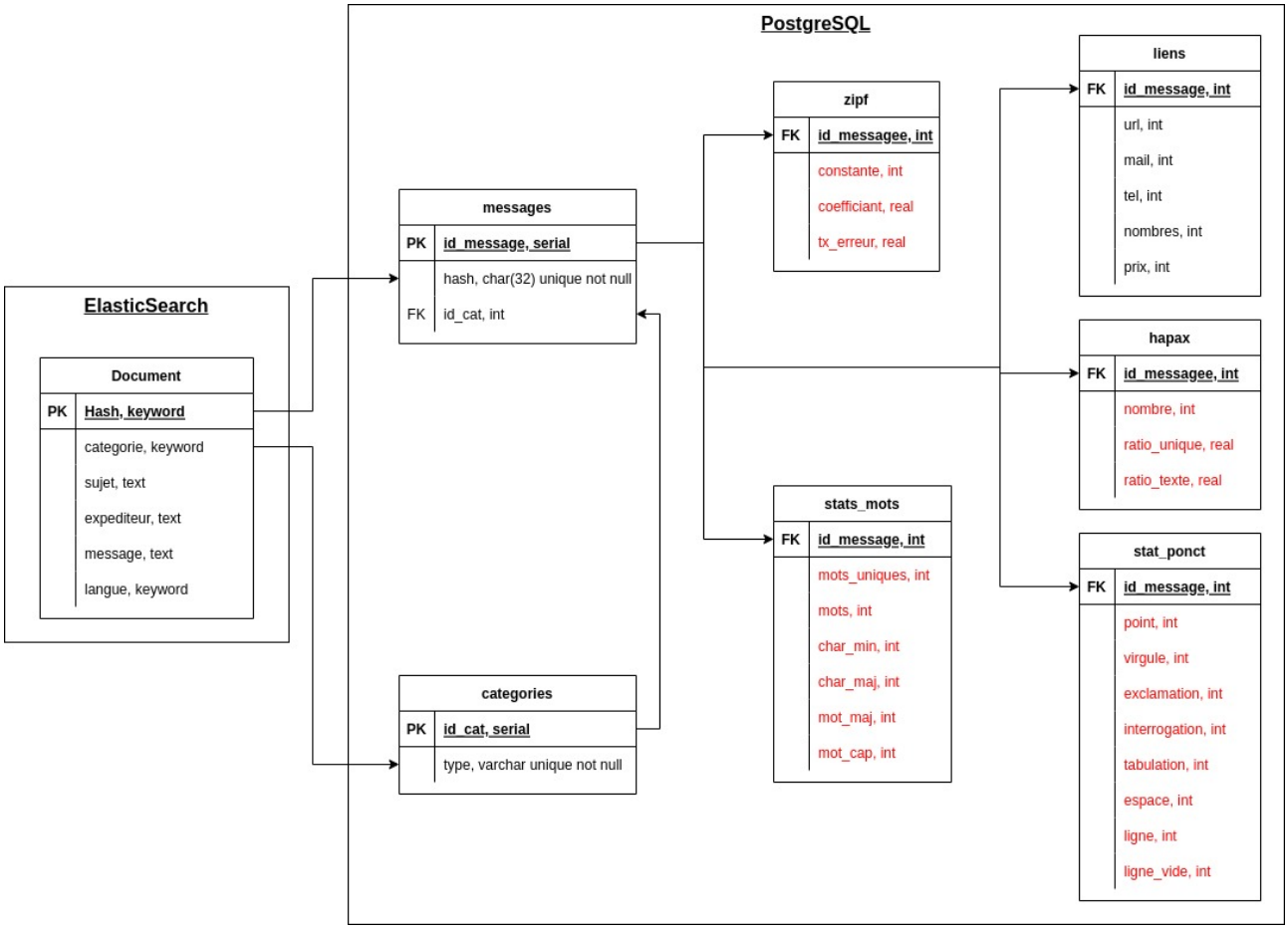


FIGURE 3 – Schéma des bases de données de l'application

Le *hash* calculé lors de la phase de traitement est l'identifiant unique du mail dans toutes les bases. La catégorie du mail est également présente dans les deux bases. Les champs en rouge sont des caractéristiques qui ne sont pas calculées lors de la phase 1.

Stockage des données : ElasticSearch ElasticSearch est un moteur de base de données NoSQL. Il intègre un moteur d'indexation des documents est assez performant pour stocker des données textuelles de taille aléatoire. Il est cependant assez compliqué de modifier le schéma d'un index une fois qu'il a été créé. Pour ces raisons, j'utilise cette technique pour ne stocker que les données textuelles après nettoyage car elles n'ont plus vocation à être modifiées.

Les corps de mail mis en base seront récupérés ultérieurement pour réaliser les opérations d'analyse. Les résultats seront stockés dans la base PostgreSQL, plus souple.

Ci-dessous les fonctions principales pour l'ajout d'un document dans la base ES.

Fonctions basique pour la liaison ElasticSearch

```python
def es_connect(server, creds, crt):
    """ Connexion au serveur ElasticSearch """
    client = Elasticsearch(server, api_key=creds, ca_certs=crt)

    try:
        client.search()
        client.indices.get(index="*")
        return client

    except (exceptions.ConnectionError, AuthenticationException,
    AuthorizationException) as err:
        print("ES:conn - Informations client ElasticSearch :\n\t", err)
        client.close()
        return None


def es_index_doc(es_cli, index, doc):
    """ Index un document dans la base ES """
    id_doc = doc['hash']

    if es_document_exists(es_cli, index, id_doc):
        return 1

    es_cli.index(index=index, document=doc)
    es_cli.indices.refresh(index=index)
    return


def es_document_exists(es_cli, index, hash):
    """ Regader dans l'index si le hash du document est deja present """
    try:
        resp = es_cli.search(index=index, query={"match": {"hash": hash}})
    except elasticsearch.NotFoundError as err:
        print("Error : hash", err, file=sys.stderr)
        return None

    return True if resp['hits']['total']['value'] == 1 else False

```

Le déploiement et les configurations de la base ElasticSearch sont disponibles dans l'annexe B.1.

Stockage des premières informations statistiques : PostgreSQL Pour le stockage des données statiques et d'analyse, j'ai décidé de m'orienter vers le système de gestion de base de données PostgreSQL. Une base de données relationnelle est plus flexible qu'un index Elastic-Search pour l'ajout de nouvelles caractéristiques.

Durant la phase 1, j'utilise que 3 tables :
— messages - liste des mails avec le hash permettant de faire le lien avec l'index ES
— categorie - liste des catégories de mails
— liens - comptabilise le nombre d'occurrences par type de lien ou nombre

Cette base a pour but de stocker les données formatées pour l'analyse, l'exploitation et l'entraînement du modèle.

Fonctions basiques pour la liaison PostgreSQL

```python
def connect_db(database, user, passwd, host, port):
    """ Connexion a la base de donnees Postgres """
    try:
        client_psql = psycopg2.connect(database=database, user=user, password=passwd, host=host, port=port)
    except psycopg2.Error as e:
        print("Erreur de connexion : \n{}".format(e), file=sys.stderr)
        return None

    client_psql.autocommit = True
    return client_psql


def insert_data(client_psql, table, data):
    """
    Insere les donnees d'un dictionnaire dans une table de la base de donnees PSQL
    Les cles du dictionnaire doivent correspondre aux colonnes de la table.
    """
    cols = ','.join([str(c) for c in data.keys()])
    vals = ','.join([str(v) if (type(v) != str) else f"'{v}'" for v in data.values()])
    query = f"INSERT INTO {table}({cols}) VALUES ({vals})"

    exec_query(client_psql, query)


def get_data(client_psql, table, champs, clause=None):
    """ Recupere les donnees de la base. """
    query = f"SELECT {','.join(champs)} FROM {table}"
    if clause:
        query += f" WHERE {clause}"

    result = exec_query(client_psql, query)
    return [dict(zip(champs, ligne)) for ligne in result]


def insert_document_init(client_psql, data, id_cat):
    """ Insere un nouveau document dans la base PSQL. """
```

```
37      insert_data(client_psql, 'messages', {'hash': data['hash'], 'id_cat':
    id_cat})
38      id_message = get_data(client_psql, 'messages', ['id_message'], f"hash
    LIKE '{data['hash']}'")[0]['id_message']
39
40      liens = data['liens']
41      liens.update({'id_message': id_message})
42      insert_data(client_psql, 'liens', liens)
43
44  def exec_query(client_psql, query):
45      """ Execute une query dans la base PSQL """
46      cursor = client_psql.cursor()
47
48      try:
49          cursor.execute(query)
50          if query.upper().find("SELECT", 0, 6) >= 0:
51              return cursor.fetchall()
52          return []
53      except psycopg2.Error as e:
54          print("Erreur d'execution de la requete : {}".format(e), file=sys.
    stderr)
55          print("requete : {}".format(query), file=sys.stderr)
56          return []
57
```

Le déploiement et les configurations de la base PostgreSQL sont disponibles dans l'annexe B.2.

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. A chaque grandes étapes de la phase 1 (Importation, Nettoyage, 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.

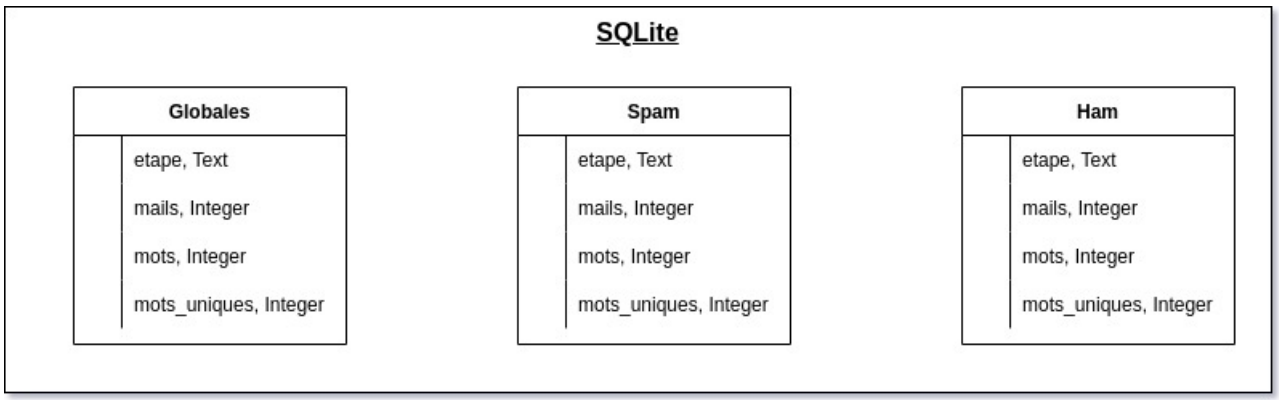


FIGURE 4 – Schéma de la base de données pour lors du traitement

1.3 Données de la phase 1

Sortie du programme : main_collecte.py

```
=== Vérification des conteneurs Docker ===
* Conteneur 'docker-es01-1'... OK
* Conteneur 'docker-kibana-1'... OK
* Conteneur 'docker_pgadmin_1'... OK
* Conteneur 'docker_pgdb_1'... OK
=== Phase 1 : collecte & mise en base ===
== Création de la base SQLITE
SQLITE table globales : CREATED
SQLITE table spam : CREATED
SQLITE table ham : CREATED
== Recolte ==
-- Création de la liste des fichiers... OK
--- Process de statistiques après la récole
-- Stats - étape : Récolte spam... OK
-- Stats - étape : Récolte ham... OK
--- Sauvegarde des stats de l'étape: recolte... OK
Données stats de l'étape: recolte:
HAM,  mails: 4153  mots: 1863674  mots uniques: 178812
SPAM,  mails: 1898  mots: 918256  mots uniques: 129601
GLOBALES,  mails: 6051  mots: 2781930  mots uniques: 287637
== Création document ==
-- Importation - Création spam... OK
-- Importation - Création ham... OK
--- Process de statistiques après la création de document
-- Stats - étape : création documents spam... OK
-- Stats - étape : création documents ham... OK
--- Sauvegarde des stats de l'étape: creation document... OK
Données stats de l'étape: creation document:
HAM,  mails: 4124  mots: 844151  mots uniques: 73725
SPAM,  mails: 1784  mots: 584012  mots uniques: 40099
GLOBALES,  mails: 5908  mots: 1428163  mots uniques: 97235
== Mise en base des documents ==
-- Création de l'index ElasticSearch... OK
-- Création de la base PostgreSQL
User 'data' déjà existant
-- Création des tables PostgreSQL... OK
-- Mise en base ES & PSQL des spam... OK (256 doublons)
-- Mise en base ES & PSQL des ham... OK (78 doublons)
-- Récupération des spam... OK
-- Récupération des ham... OK
--- Process de statistiques après la mise en base
-- Stats - étape : Mise en base spam... OK
-- Stats - étape : Mise en base ham... OK
--- Sauvegarde des stats de l'étape: mise en base... OK
Données stats de l'étape: mise en base:
HAM,  mails: 4046  mots: 839368  mots uniques: 73725
SPAM,  mails: 1528  mots: 529121  mots uniques: 40099
```

```
GLOBALES,  mails: 5574  mots: 1368489  mots uniques: 97235
== FIN ==
```

Évolution lors du traitement Les données recueillis pendant le traitement permettent de constater les comportements suivants :

- 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 mot uniques est plus important dans les ham que dans les spam

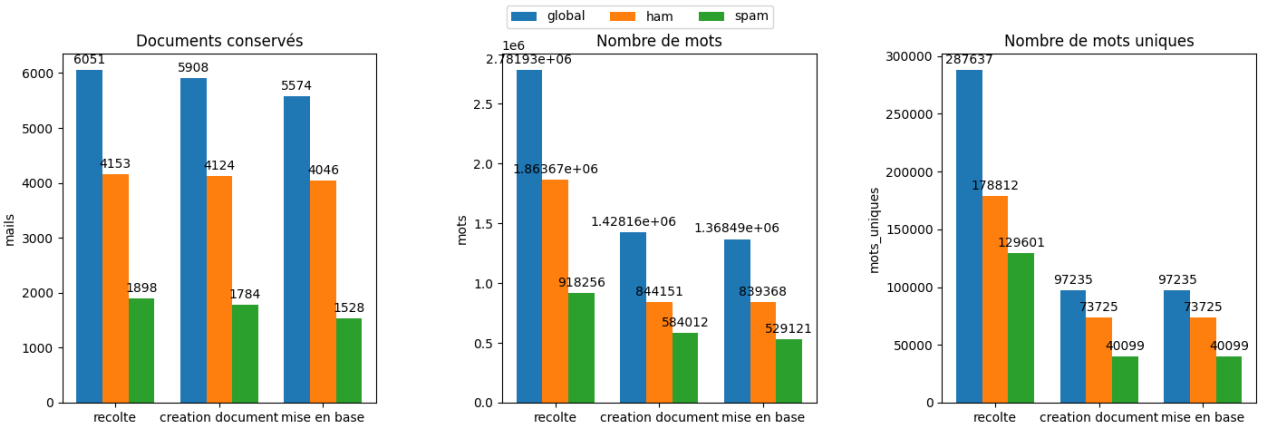


FIGURE 5 – Évolution des principaux marqueurs lors du traitement

Conclusion de la récolte A l'issue de 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.

Premières données statistiques Les documents conservés à l'issue de cette phase de récupération se constitue à $72,6\%$ de Ham et à 27.4% de Spam.

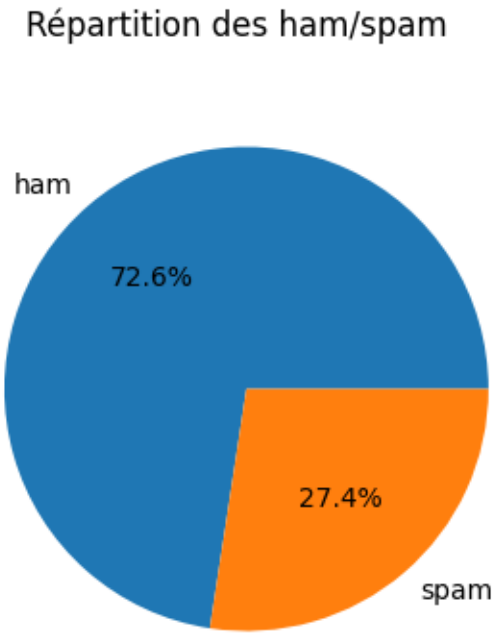


FIGURE 6 – Répartition des ham/spam dans le dataset

Grâce aux informations et des pré-traitements réalisés lors de cette première phase, il est possible de présenter quelques données statistiques. Notamment sur la présence et l'utilisation de liens (url, mail, téléphonique) et des données numériques (nombres et prix).

TABLE 1 – Statistiques sur les liens

	url			mail			tel		
	Globale	Ham	Spam	Globale	Ham	Spam	Globale	Ham	Spam
moyenne	5.47	5.54	5.26	1.16	1.21	1.01	0.16	0.20	0.64
écart-type	49.74	57.77	13.71	2.50	1.72	3.88	1.06	0.63	1.73
minimum	0			0			0		
25%	1			0			0		
médiane	2	2	3	1	1	0	0		
75%	4	4	5	2	2	1	0		
maximum	3557	3557	295	69	34	69	67	25	67

A partir des données des liens présentées dans le tableau ci-dessus il est possible d'émettre les hypothèses suivantes.
On trouve en moyenne environ 5 url dans les deux types de mails. Cependant la dispersion est notablement plus faible pour les spams. On voit également que 75% des spam contiennent 5 liens ou moins.
La présence d'adresses mail est assez marginale environ une adresse mail par corps de texte. L'écart-type est plus faible dans les ham. On voit également que 75% des ham contiennent deux adresse mails ou moins contre 1 pour les spam.
Enfin la présence de numéro de téléphone est quasiment nulle pour les deux catégories.

TABLE 2 – Statistiques sur les données numériques

	nombre			prix		
	Globale	Ham	Spam	Globale	Ham	Spam
moyenne	15.14	9.38	30.38	0.28	0.08	0.81
écart-type	219.16	49.35	410.53	1.54	0.66	2.66
minimum	0			0		
25%	1	1	2	0		
médiane	5	5	7	0		
75%	9	8	14	0		
maximum	15801	2794	15801	29	17	29

Les données numériques du dataset montrent les hypothèses suivantes.
Les nombres simples semblent être les données les plus présentes dans le corps du mail, environ 15 en moyenne par mail, cependant la dispersion est élevée. Cela étant 75% des spam ont 14 nombres ou moins contre 8 ou moins pour les ham.
Enfin la présence de prix est assez marginale par rapport aux nombres. Il y a cependant en moyenne 10 fois plus de prix dans les spam que les ham.

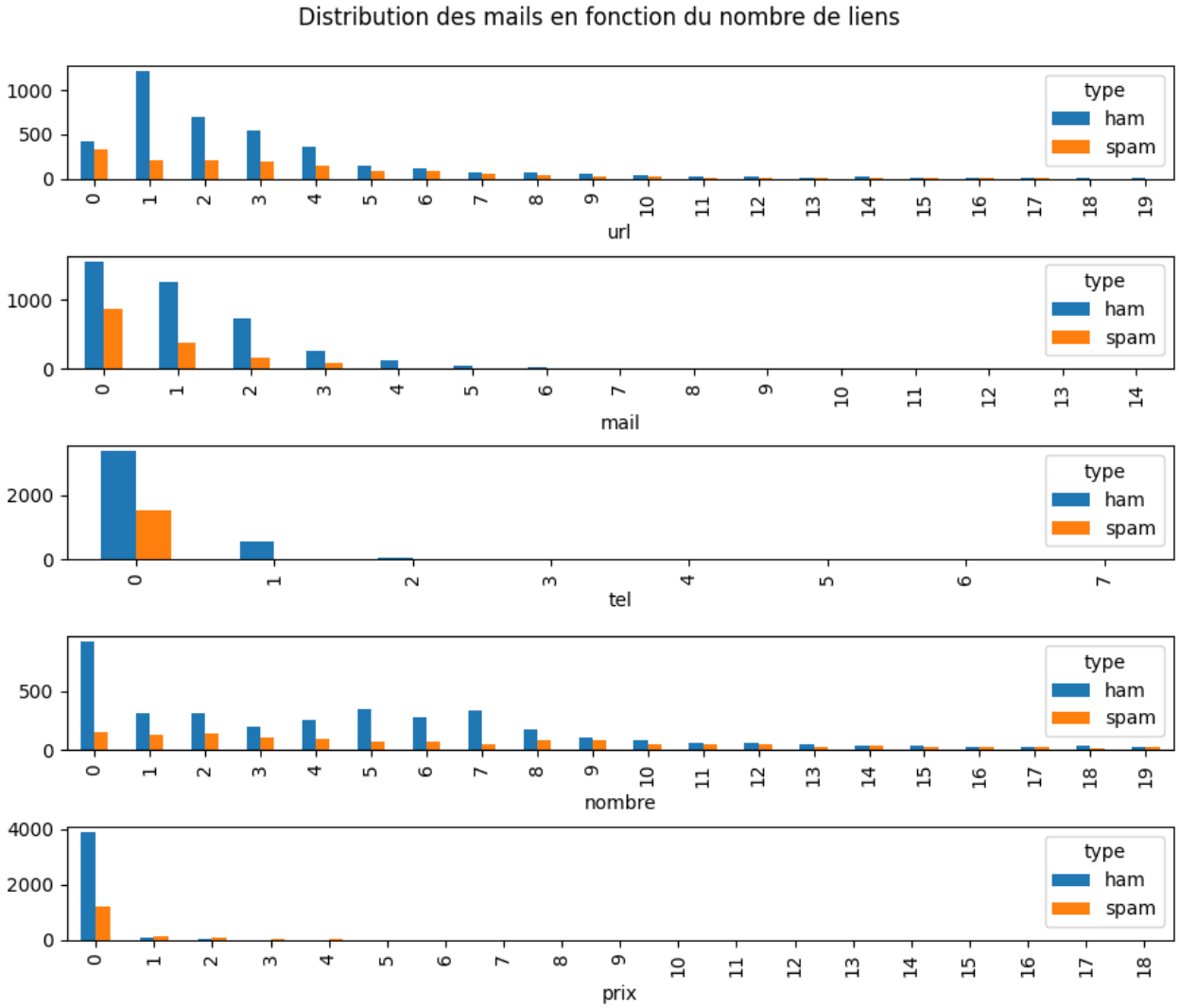


FIGURE 7 – Distribution du nombre de liens selon la catégorie du mail

Au vu des analyses précédents, on peut dire qu'il est statistiquement plus probable de trouver dans les spam plus de liens url, de nombre et de prix que dans les ham. La présence d'adresse mail est plus probable dans les ham. Il est enfin possible d'exclure la présence de numéro de téléphone qui est trop marginale.

Les codes sources de cette partie sont disponibles dans le fichier *./analyse/analyse_p1.py* ou dans la section G.2

2 Phase 2 : Traitement des données

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

2.1 Génération des données

Nous verrons ici le code utilisé pour récupérer ou générer les données des caractéristiques pour chaque message ainsi que la distribution de Zipf de manière globale.

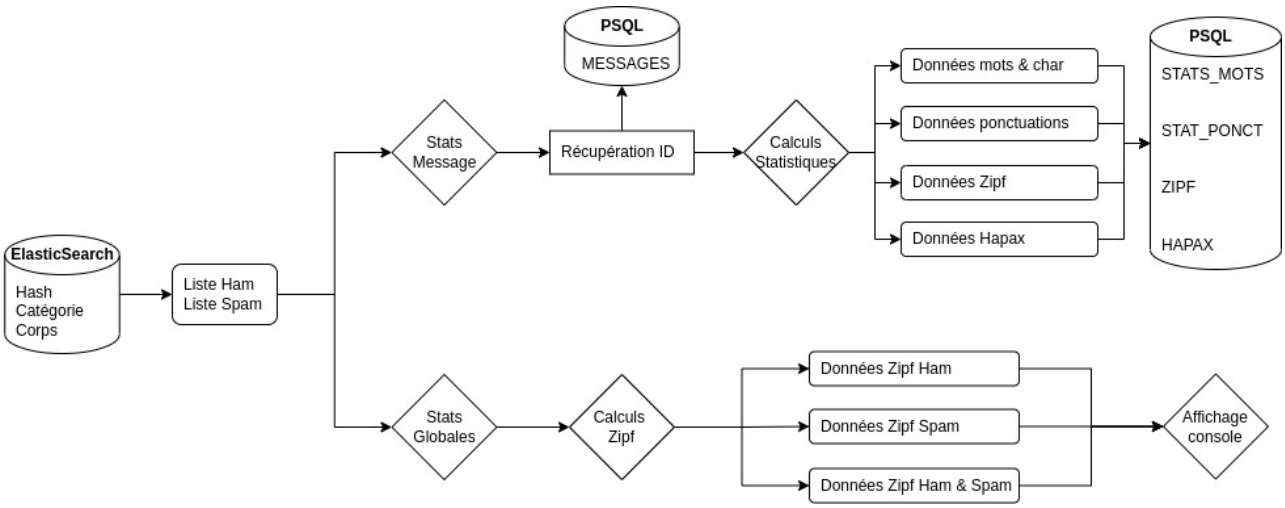


FIGURE 8 – Schéma des étapes de récupération des données

Les mails sont récupérés dans la base ElasticSearch. Les Ham et les Spam sont séparés dans deux listes distinctes.

Récupération des messages dans la base ES

```
1 def recup_mails(es_index):
2     es_cli = elastic_cmd.es_connect(es_secrets.serveur, (es_secrets.apiid,
      es_secrets.apikey), es_secrets.ca_cert)
3     data = {}
4     for cat in ['spam', 'ham']:
5         print("-- Recuperation des {}...".format(cat), end=' ')
6         data[cat] = {entry.get('_source').get('hash'): entry.get('_source'
    ).get('message') for entry in elastic_cmd.es_get_all(es_cli, es_index,
    sort={'hash': 'asc'},query={'match': {'categorie': cat}})}
7         print('OK')
8     return data
9
```

Chaque message va ensuite passer par une série de traitement pour générer les données des caractéristiques avant d'être mise en base dans Postgresql.

Pipeline pour chaque message

```
1 def stats_pipe_message(hash_message, message, psql_cli):
2     resp = psql_cmd.get_data(psql_cli, 'messages', ['id_message'], f"hash
    LIKE '{hash_message}'")
3     if not resp:
```

```
4           print(f"No id_message found for {hash_message}", file=sys.stderr)
5           return
6       id_mess = resp[0].get('id_message')
7
8       for table, stat_func in {'stat_ponct': stats_ponct,
9                                'stats_mots': stats_mot,
10                               'zipf': stats_zipf,
11                               'hapax': stats_hapax}.items():
12          resp = psql_cmd.get_data(psql_cli, table, ['id_message'], f"
    id_message={id_mess}")
13          if resp:
14              continue
15          psql_cmd.insert_data(psql_cli, table, stat_func(id_mess, message)
16
17  def stats_ponct(id_text, texte):
18      return {
19          "id_message": id_text,
20          "point": texte.count('.'),
21          "virgule": texte.count(','),
22          "exclamation": texte.count('!'),
23          "interrogation": texte.count('?'),
24          "tabulation": texte.count('\t'),
25          "espace": texte.count(' '),
26          "ligne": texte.count('\n') + 1,
27          "ligne_vide": len(re.findall(r'^\s*$', texte, re.MULTILINE))
28      }
29
30
31  def stats_mot(id_text, texte):
32      tokens = re.findall(r'\w+', texte, re.MULTILINE).
33      return {
34          'id_message': id_text,
35          'char_min': len(re.findall(r'[a-z]', texte, re.MULTILINE)),
36          'char_maj': len(re.findall(r'[A-Z]', texte, re.MULTILINE)),
37          'mots': len(tokens),
38          'mots_uniques': len(set(tokens)),
39          'mot_maj': sum(mot.isupper() for mot in tokens),
40          'mot_cap': sum(bool(re.match(r'[A-Z][a-z]+', mot)) for mot in
    tokens)
41      }
42
43
44  def stats_zipf(id_text, texte):
45      tokens = re.findall(r'\w+', texte, re.MULTILINE)
46      data = stats.zipf_process(tokens)
47      data['id_message'] = id_text
48      data['constante'] = data.pop('const_moy')
49      data['coefficient'] = data.pop('coef_min')
50      data['tx_erreur'] = data.pop('cout_min')
51      return data
52
53
54  def stats_hapax(id_text, texte):
```

```
    tokens = re.findall(r'\w+', texte, re.MULTILINE)
    data = stats.hapax(tokens)
    data['id_message'] = id_text
    data['nombre'] = data.pop('nombres')
    data['ratio_unique'] = data.pop('ratio_mots_uniques')
    return data

```

J'ai également réaliser les calculs de la distribution de Zipf sur l'ensemble du corpus ainsi que sur le regroupement des Ham et des Spam pour voir si des différences majeures pouvaient être mises en lumière. Les résultats de cette recherche sont affiché directement dans la console.

Pipeline pour les données globales

```
def stats_pipe_globale(data):
    ls_ham = [mess for mess in data['ham'].values()]
    ls_spam = [mess for mess in data['spam'].values()]
    tokens = []
    for mess in ls_ham + ls_spam:
        tokens += re.findall(r'\w+', mess, re.MULTILINE)
    data = stats.zipf_process(tokens, True)
    print("Donnees Zipf ham+spam:", data)
    data = stats.hapax(tokens)
    print("Donnees Hapax ham+spam:", data)

    tokens = []
    for mess in ls_ham:
        tokens += re.findall(r'\w+', mess, re.MULTILINE)
    data = stats.zipf_process(tokens, True)
    print("Donnees Zipf ham:", data)
    data = stats.hapax(tokens)
    print("Donnees Hapax ham:", data)

    tokens = []
    for mess in ls_spam:
        tokens += re.findall(r'\w+', mess, re.MULTILINE)
    data = stats.zipf_process(tokens, True)
    print("Donnees Zipf spam:", data)
    data = stats.hapax(tokens)
    print("Donnees Hapax spam:", data

```

Les fonctions pour les instructions pour la distribution de Zipf sont donnés dans l'annexe A.

2.2 Analyses statistiques

Dans cette partie nous nous intéressons aux données statistiques des caractéristiques récupérées lors de la phase précédente. Nous verrons dans un premier temps les données des mots :

- nombre de mots
- nombre de mots uniques
- nombre de caractère en minuscule
- nombre de caractère en majuscule

— nombre de mots écrits complètement en majuscule
— nombre de mots écrits complètement avec la première lettre en capitale

A cela nous ajouterons une recherche de caractéristique sur les données de ponctuation :

— nombre de points '.'
— nombre de virgules ','
— nombre de points d'exclamation
— nombre de points d'interrogation
— nombre de tabulations
— nombre d'espaces
— nombre de lignes totales
— nombre de lignes vides

Enfin nous appliquerons les méthodes de distribution de Zipf sur chaque message [1]. Nous récupérerons alors les données suivantes

— constante estimée
— coefficient estimées
— taux d'erreur absolu moyen entre les fréquences réelles et théoriques de chaque mot
— nombre de mots n'apparaissant qu'une seule fois dans le texte (hapax)
— nombre d'hapax par rapport au vocabulaire du texte
— nombre d'hapax par rapport au total des mots du texte

TABLE 3 – Statistiques sur les données des mots

	nombre de mots			mots uniques			mots capitalisés			mots majuscules		
	Global	Ham	Spam	Global	Ham	Spam	Global	Ham	Spam	Global	Ham	Spam
moyenne	243	205	343	130	116	167	41	34	59	14	8	29
écart-type	586	531	701	187	180	198	114	111	119	49	39	66
minimum	1	3	1	1	3	1	0	0	0	0	0	0
25%	53	44	94	46	39	73	9	8	15	2	2	3
médiane	105	85	161	80	68	110	17	14	32	4	3	10
75%	215	174	326	139	120	191	35	25	60	10	7	24
maximum	14127	14127	11274	4322	4322	2479	4656	4656	2192	1969	1969	763

Statistiques des mots et caractères Le tableau 3, sur les données des mots, montre que les Spam ont généralement plus de mots, 343 contre 205 en moyenne pour les Hams. Cela est également vrai pour les mots uniques par messages, les spams en contiennent globalement plus. Concernant la forme des mots, on voit que les mots avec majuscules sont plus souvent utilisés dans les Spam. Enfin, il est important de noter la forte dispersion des données de toutes ces métriques quelque soit la catégorie.

TABLE 4 – Statistiques sur les données des caractères

	caractères en minuscule			caractères en majuscule		
	Globale	Ham	Spam	Globale	Ham	Spam
moyenne	1084	923	1511	111	66	232
écart-type	2816	2518	3448	736	222	1351
minimum	0	3	0	0	0	0
25%	223	187	380	15	13	43
médiane	442	367	675	31	23	84
75%	917	755	1410	76	44	173
maximum	68082	68082	54406	50543	6450	50543

1. J'ai conscience que cette distribution est plus pertinente quand le corpus est grand

Le tableau 4 qui montre les données des caractères minuscules et majuscules montre une taille moyenne des Spam plus grande que celle des Ham. Il est également intéressant de noter que l'utilisation des majuscules est plus prononcée dans les Spam.

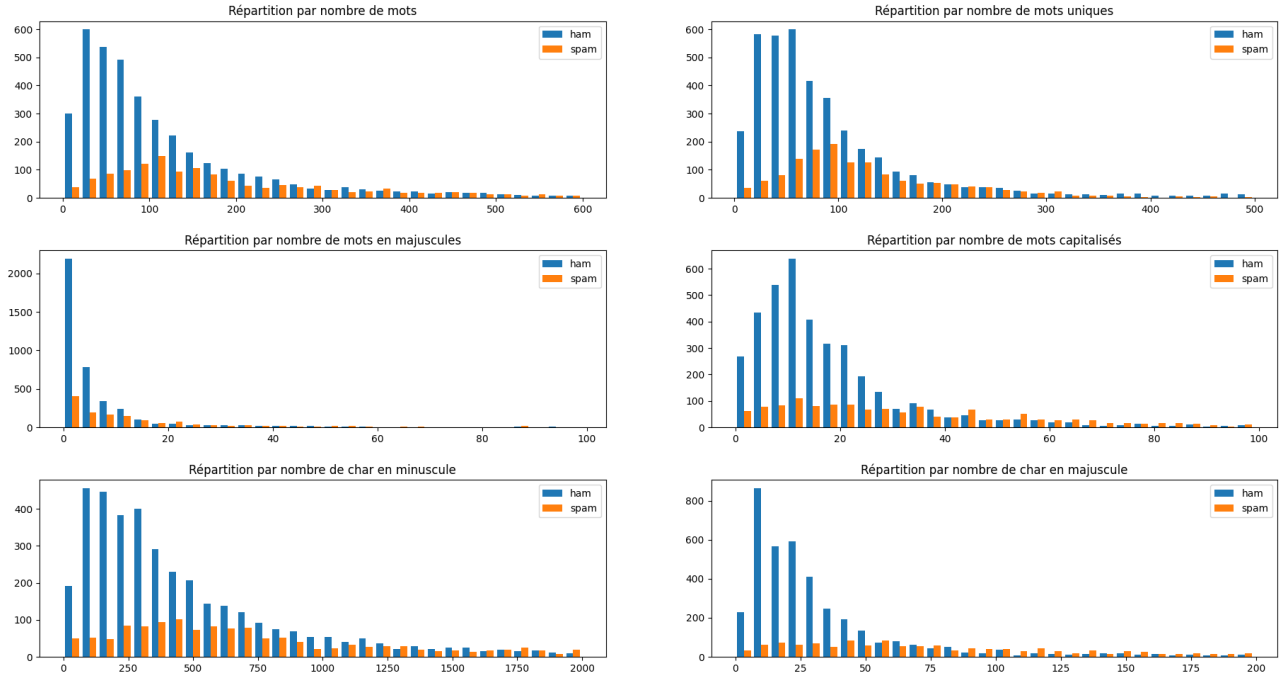


FIGURE 9 – Distribution des données de mots selon la catégorie du mail

Concernant les distributions des mails visibles dans la Figure 9 on remarque que les Ham se concentrent généralement vers des métriques basses puis l'effectif se réduit à mesure que la valeur de la métrique augmente. Les Spam quand à eux ont plus tendance à suivre une distribution de type normale ou uniforme.

TABLE 5 – Statistiques sur les ponctuations

	points			virgules			exclamations			interrogations		
	Global	Ham	Spam	Global	Ham	Spam	Global	Ham	Spam	Global	Ham	Spam
moyenne	14	12	20	12	11	16	2	0.7	5.9	1	0.9	1.3
écart-type	35	30	46	37	31	50	7	5	10	2	2	3
minimum	0	0	0	0	0	0	0	0	0	0	0	0
25%	3	3	3	2	2	2	0	0	1	0	0	0
médiane	6	5	9	5	4	6	0	0	3	0	0	0
75%	13	11	18	10	9	14	2	1	7	1	1	1
maximum	848	848	643	1268	890	1268	343	343	92	79	79	58

Statistiques des ponctuations Le tableau 5 des données de ponctuation donne l'information que les textes des Spam utilisent plus de signes de ponctuation que les Ham. Il faut relever que l'utilisation des points d'interrogation est anecdotique pour les deux catégories.

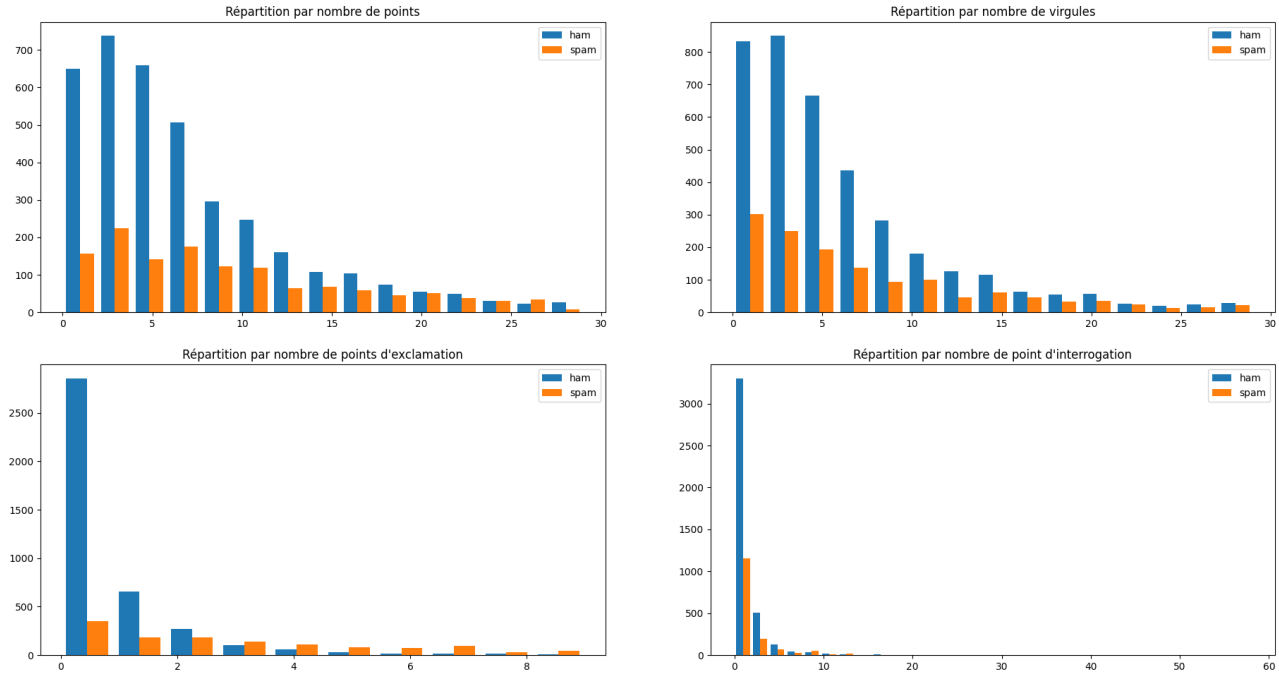


FIGURE 10 – Distribution des données de la ponctuation selon la catégorie du mail

Les schéma de la Figure 10 montrent un comportement similaire pour les deux catégories avec le nombre de points, de virgule et de point d'interrogation. Seule la répartition des effectifs de spam pour le nombre de point d'exclamation semble ne pas suivre le même cheminement et semble rester uniforme.

TABLE 6 – Statistiques sur les espaces et les lignes

	tabulations			espaces			lignes			lignes vides		
	Globale	Ham	Spam	Globale	Ham	Spam	Globale	Ham	Spam	Globale	Ham	Spam
moyenne	2	2	3	287	240	411	65	57	86	15	13	19
écart-type	29	32	19	874	917	733	137	144	113	44	50	22
minimum	0	0	0	1	2	1	1	1	1	0	0	0
25%	0	0	0	55	45	99	21	19	28	6	6	8
médiane	0	0	0	114	95	196	34	30	52	9	8	13
75%	0	0	0	255	196	435	61	48	97	14	12	23
maximum	1118	1118	379	36350	36350	11436	6320	6320	1695	2882	2882	291

Statistiques des espaces et des lignes Le tableau 6 regroupe les données des espace vide et le calcul du nombre de lignes. On remarque que les tabulations ne sont pas utilisées, sauf en grande quantité. Il pourrait être intéressant de voir si cet élément peut être utiliser pour détecter les documents générant des valeurs aberrantes. Pour ce qui est du nombre d'espaces, de lignes et de lignes vide, la moyenne des Spam est approximativement le double de celle des Ham.

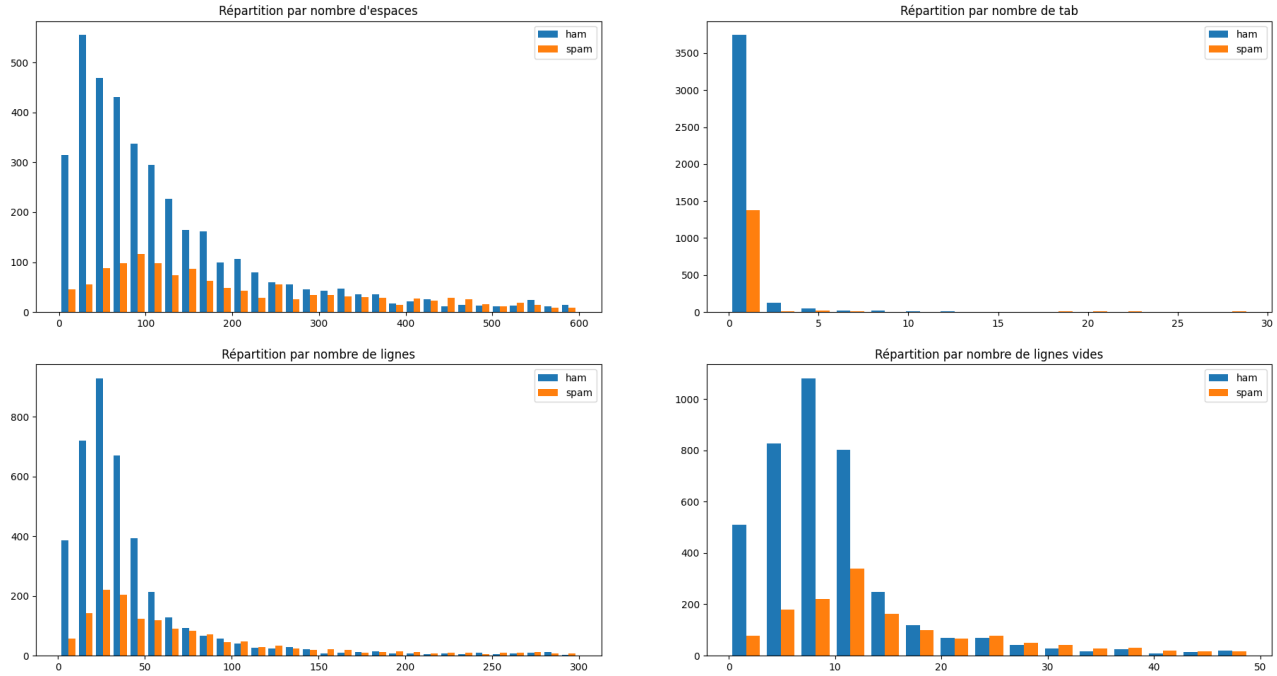
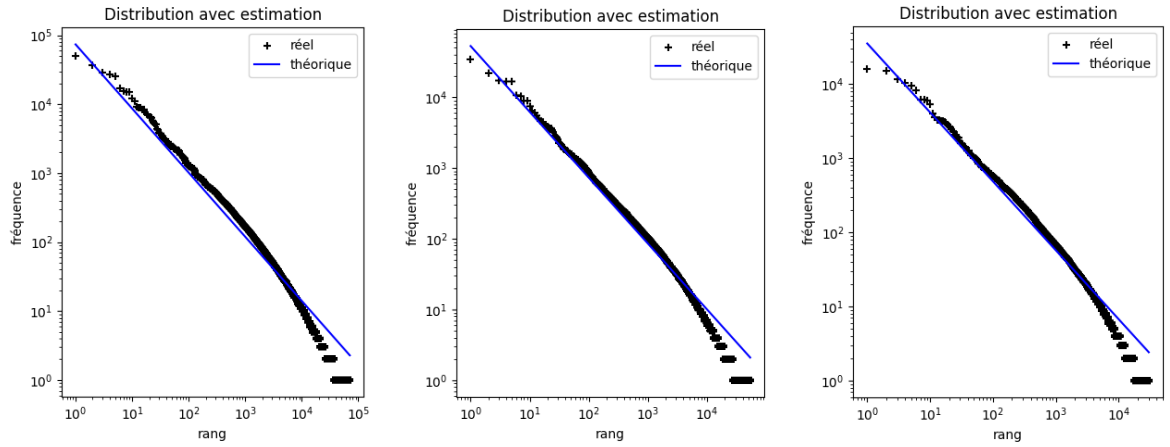


FIGURE 11 – Distribution des données de lignes et espaces selon la catégorie du mail

On peut voir dans les histogrammes de la Figure 11 que les mails se répartissent de manière relativement normale pour le comptage du nombre de lignes et de lignes vides pour des valeurs faibles. Puis à mesure que la valeur augmente, l'effectif semble se réduire de manière linéaire. La répartition du nombre d'espace ressemble fortement à la répartition du nombre de mots. Il est possible que le nombre d'espaces soit corrélé au nombre de mots.

Distribution de Zipf Cette distribution est une loi empirique qui permet de décrire un classement des mots selon leur fréquence d'apparition. Les fonctions utilisées pour la mise en œuvre de cette distribution ont été réalisées par mes soins. L'annexe A détaille le développement de cette fonctionnalité.

Les données suivantes montre la représentation de cette distribution pour l'ensemble du corpus.



(a) Ham + Spam (b) Ham (c) Spam

On voit dans les graphiques ci dessus que les corpus Ham, Spam et l'association des deux respectent une distribution zipfienne.

	constante	coefficient	erreur moyenne	hapax	hapax/vocab	hapax/total
Ham + Spam	73746.97	0.93	6.37	33666	0.47	0.02
Ham	52712.40	0.93	4.19	26231	0.48	0.03
Spam	35485.13	0.93	4.84	12396	0.40	0.02

On remarque à partir du tableau ci dessus que la constante estimée est plus importante dans les corpus Ham que dans le corpus Spam. La constante des Ham (52712) est assez proche de la constante moyenne découverte avec le corpus de Brown lors du développement de cette méthode (56525). Il est possible d'émettre l'hypothèse que le vocabulaire des Ham est plus fournis. On remarque également que le taux d'hapax dans le vocabulaire des Ham est de 48% alors que cette proportion est de 40% pour les Spam.

Distribution de Zipf par message Le processus appliqué pour le calcul de la distribution de Zipf sur les différent corpus a été utilisé sur chaque mail individuellement. Il a alors été possible d'ajouter ces données (constante, coefficient, taux d'erreur, nombre d'hapax et ratio d'hapax) pour chaque mail. Le résultat de ce traitement est détaillé dans le Tableau 7 et dans la Figure 13.

TABLE 7 – Statistiques sur les données des calculées de Zipf et Hapax

	constante		coefficient		taux d'erreur		hapax		ratio vocabulaire		ratio texte	
	Ham	Spam	Ham	Spam	Ham	Spam	Ham	Spam	Ham	Spam	Ham	Spam
moyenne	66	102	1.21	1.19	1.37	1.54	87.7	111.53	0.83	0.72	0.67	0.52
écart-type	115	141	0.08	0.07	0.24	0.41	123.8	123.4	0.12	0.21	0.20	0.22
minimum	2	1	0.86	0.86	0.44	0	0	0	0	0	0	0
25%	21	39	1.16	1.14	1.26	1.37	35	51	0.77	0.71	0.52	0.40
médiane	36	63	1.22	1.19	1.37	1.46	57	80	0.84	0.77	0.67	0.54
75%	65	112	1.29	1.26	1.46	1.57	93	136	0.91	0.83	0.83	0.67
maximum	2311	1866	1.30	1.30	3.49	3.49	3552	1690	1	1	1	1

Le tableau 7 montre que les constantes estimées sont globalement plus importantes dans les spams. Le coefficient est plus élevé dans les Ham. Cependant il est à noté que la moyenne des coefficient pour les Ham (1.21) et pour les Spam (1.19) est assez supérieure au coefficient estimé pour tout le corpus (0.93). On remarque également pour le coefficient que la valeur se rapproche rapidement de 1.30 qui est la limite haute fixée pour l'estimation du coefficient.

Le taux d'erreur, qui correspond à la moyenne de l'écart absolu entre la valeur réelle et estimée de la constante, est plus élevé dans les spam (1.54). Mais cette valeur reste inférieure à celle observée pour l'ensemble du corpus. Cela peut s'expliquer par un écart de fréquence moins important entre le mots les plus présents et les moins présents.

Le calcul des hapax semble plus parlant mail par mail que dans la globalité du corpus. On remarque effectivement un écart de 15 points entre la moyenne des hapax dans les textes des Ham (67%) que dans ceux des Spam (62%). En comparaison cet écart n'est que de 1 point sur l'ensemble de chaque corpus.

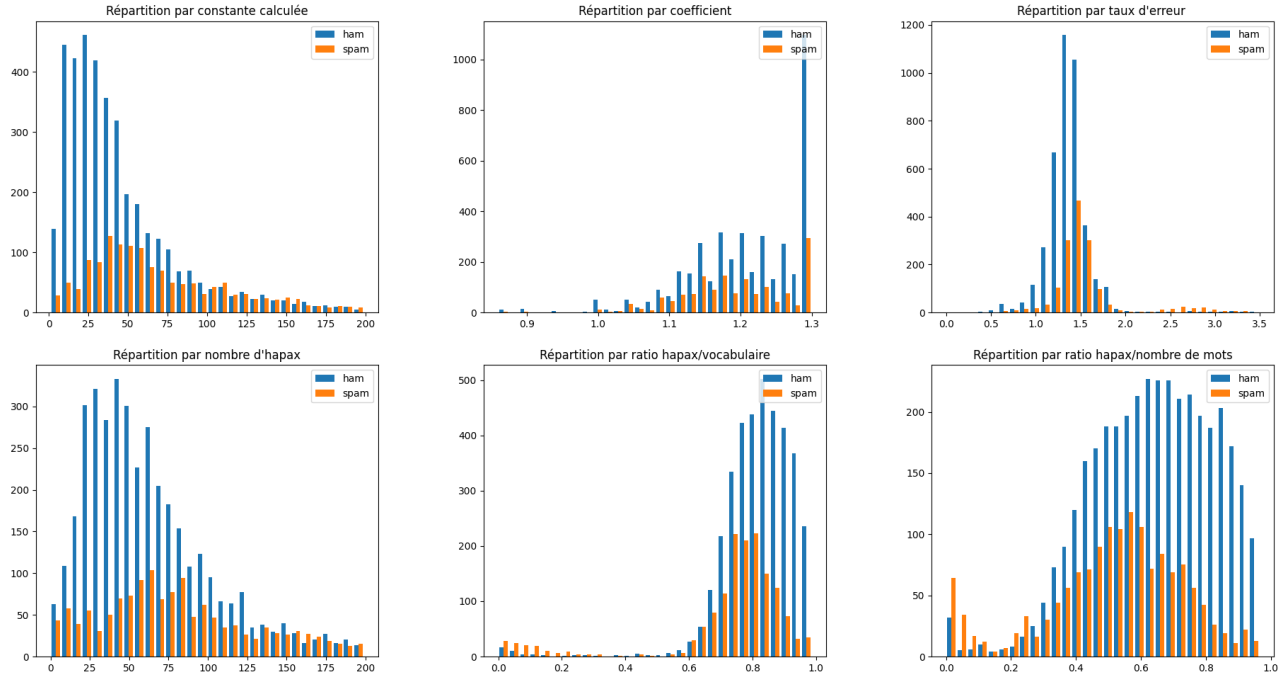


FIGURE 13 – Distribution des données de Zipf selon la catégorie du mail

On voit dans les schéma de la Figure 13 que les Ham se concentre également dans les valeurs basses de la constante. Les Spams semblent plus dispersés pour cette métrique.

Le nombre d'hapax lui semble suivre une distribution normale pour les Ham. Pour cette métrique la répartition est plus diffuse.

Les ratio du nombre d'hapax par rapport au vocabulaire et à l'ensemble du corps du mail ont une distribution normale sauf pour des valeurs marginales entre 0% et 20%. Il est intéressant de remarquer que les spam sont majoritaires dans cette tranche alors que le nombre de Spam est nettement inférieur au nombre de Ham dans le data set (70/30).

Le pic du taux d'erreur des Ham (1.4) est effectivement inférieur à celui des Spam (1.5). On remarque également un regroupement de Spam avec un taux d'erreur compris entre (2.3 et 3).

Enfin il semblerait que le coefficient de chaque type gravite autour de la médiane, avec base assez large. Le pic en bout de graphique peut être ignoré car il correspond à la limite haute de la recherche de coefficient bornée entre 0.8 et 1.30. augmenter cette limite impliquerait d'augmenter les temps que calcul qui ne semble pas justifié dans ce cas. Moins de 25% des documents ont été classés avec un coefficient supérieur à 1.29.

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 14 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.

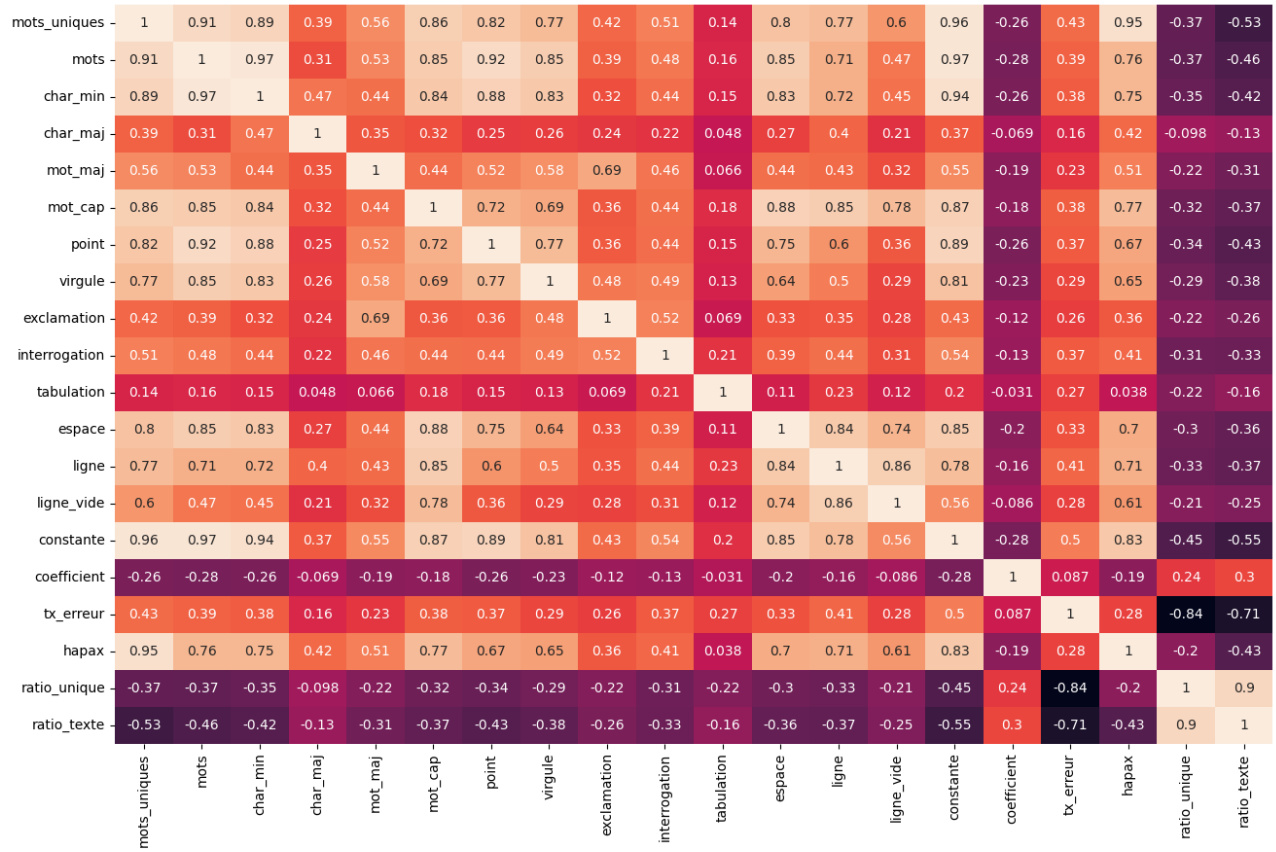


FIGURE 14 – Matrice de corrélation des métriques

On peut remarquer que les nombre de mots, de caractères en minuscule, de points et de virgules sont fortement corrélés. On voit également que certaines métriques sont quasiment indépendante de la plus part des autres variables :

— nombre de caractères en majuscules
— nombre de mots en majuscules
— nombre de point d'exclamation
— nombre de point d'interrogation
— nombre de tabulation
— coefficient Zipf déterminé
— ratio d'hapax dans le vocabulaire
— ratio d'hapax dans tout le texte

Les variables dénombrant les espaces et les lignes (vides ou non) sont faiblement liées aux signes de ponctuations.

Résumé L'analyse de l'utilisation des données des mots permet me mettre en lumière les points suivants :

— Les Spam sont généralement plus long que les Ham
— On retrouve plus de caractères en majuscules dans les Spam que dans les Ham
— Les nombres de mots et de mots uniques sont fortement corrélés

Les données sur l'utilisation des ponctuations ne montre pas d'écart majeur entre les Ham et les Spam sur l'utilisation des points et des virgule. Cependant l'utilisation des points d'exclamation est nettement plus forte dans les Spam.
L'utilisation de point d'exclamation et de majuscule est susceptible d'induire une notion d'urgence chez le lecteur.

Les données des lignes et des espaces ne permettent de faire une différence franche entre les Ham et les Spams au regards des autres données.

Globalement, les Ham et les Spam respectent les principes de la distribution de Zipf. Le coefficient est identique pour chacun des deux corpus. La constante et le nombre d'Hapax plus importantes pour les Ham s'expliquent par un nombre de document largement supérieur.

En regardant les données de la distribution de Zipf appliquée à chaque mail on remarque un coefficient moyen plus élevé et qui semble plus précis pour les Ham. Il est également notable que les mots se répètent moins dans les messages Ham.

En conclusion, on peut supposer les faits suivants :
— Le vocabulaire d'un Ham est plus fourni que celui d'un Spam.
— Il y a plus de mots dans un Spam.
— Un spam aura une construction induisant une notion d'urgence.

2.3 Traitement du langage

2.4 Sortie de la partie 2

```
=== Phase 2 : Stats Exploitation ===
-- Récupération des spam... OK
-- Récupération des ham... OK
-- Récupération des statistiques par message
Stats Spam... OK
Stats Ham... OK
-- Récupérations des statistiques globales
Données Zipf ham+spam: {'const_moy': 73746.97464716957, 'cout_min': 6.37798252533617,
'coef_min': 0.93}
Données Hapax ham+spam: {'nombres': 33666, 'ratio_mots_uniques': 0.47656526478207323,
'ratio_texte': 0.024792183385275213}
Données Zipf ham: {'const_moy': 52712.404742906976, 'cout_min': 4.193202011347324,
'coef_min': 0.93}
Données Hapax ham: {'nombres': 26231, 'ratio_mots_uniques': 0.48674175650850793,
'ratio_texte': 0.03150367568904908}
Données Zipf spam: {'const_moy': 35485.13912209956, 'cout_min': 4.848725944493132,
'coef_min': 0.93}
Données Hapax spam: {'nombres': 12396, 'ratio_mots_uniques': 0.4097309446684736,
'ratio_texte': 0.023598168648092978}
```

3 Phase 3

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 F.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>}
```

```
    """
    ranked = []
    for rang, couple in enumerate(sorted(dico.items(), key=lambda item:
item[1], reverse=True), start=1):
        ranked.append({"rang": rang,
                       "mot": couple[0],
                       "frequence": couple[1]})

    return ranked
```

On obtient les représentations suivantes :

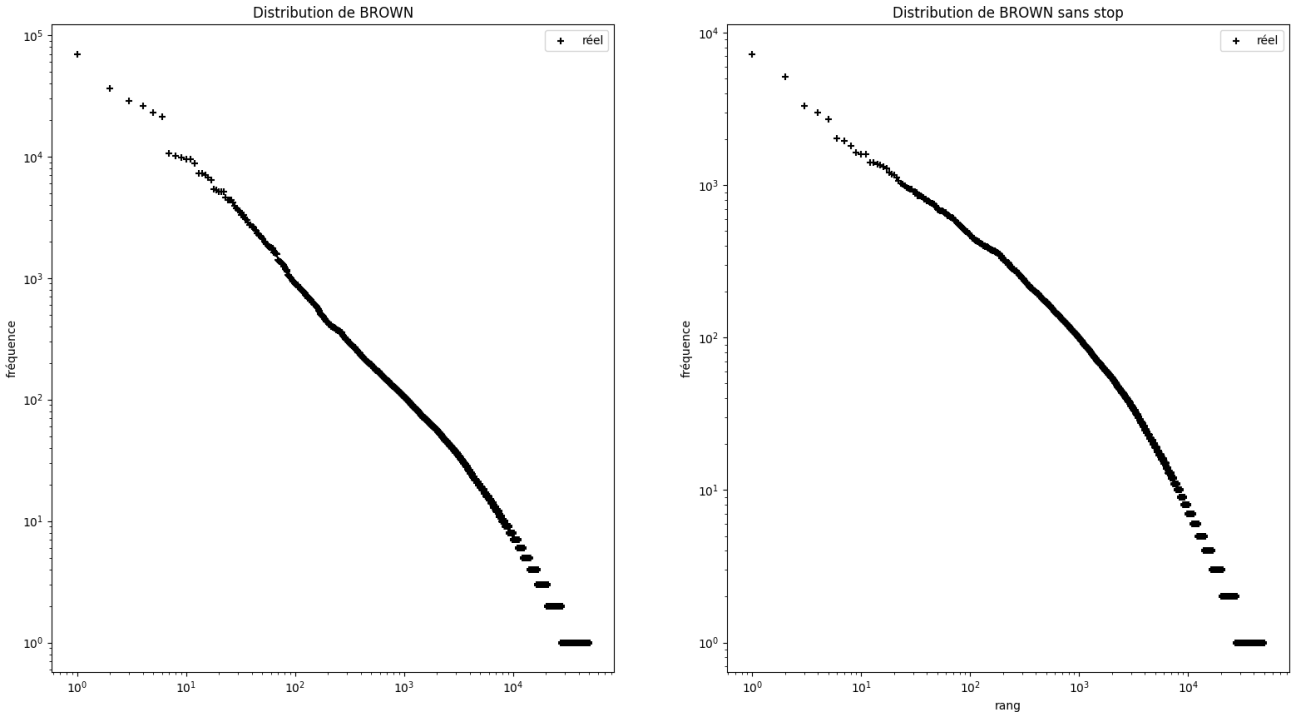


FIGURE 15 – Distribution de Zipf pour les deux corpus

— 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.

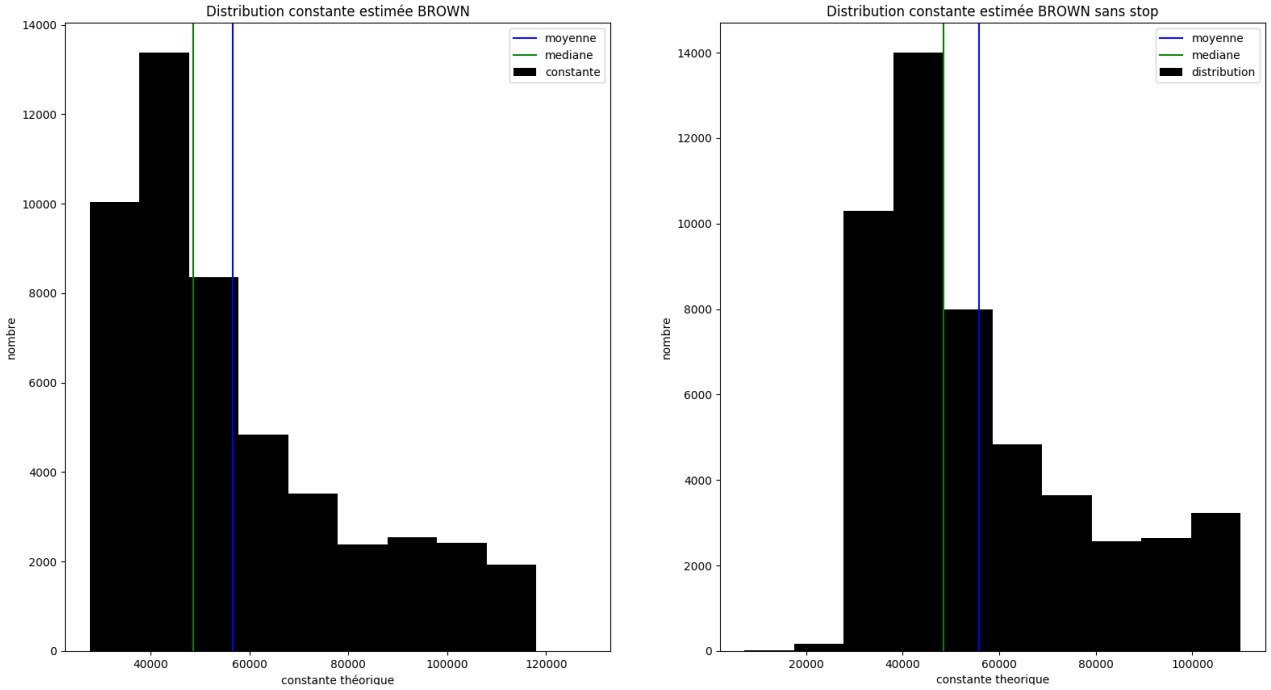


FIGURE 16 – Distribution des constantes théoriques pour les deux corpus

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 [2].
2. Calculer toutes la fréquences théoriques de tous les rangs avec tous les coefficients possibles en utilisant les constantes moyenne et médiane de chaque corpus.
3. Calculer la moyenne des coûts absolus entre les fréquences théoriques par coefficient avec la fréquence réelle observée pour chaque corpus.

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

Fonctions utilisées dans la recherche du coefficient

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

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

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

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

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

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

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

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

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

    return None
```

Calcul des fréquences par coefficient

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

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

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

La recherche du coefficient nous retourne les éléments suivants :

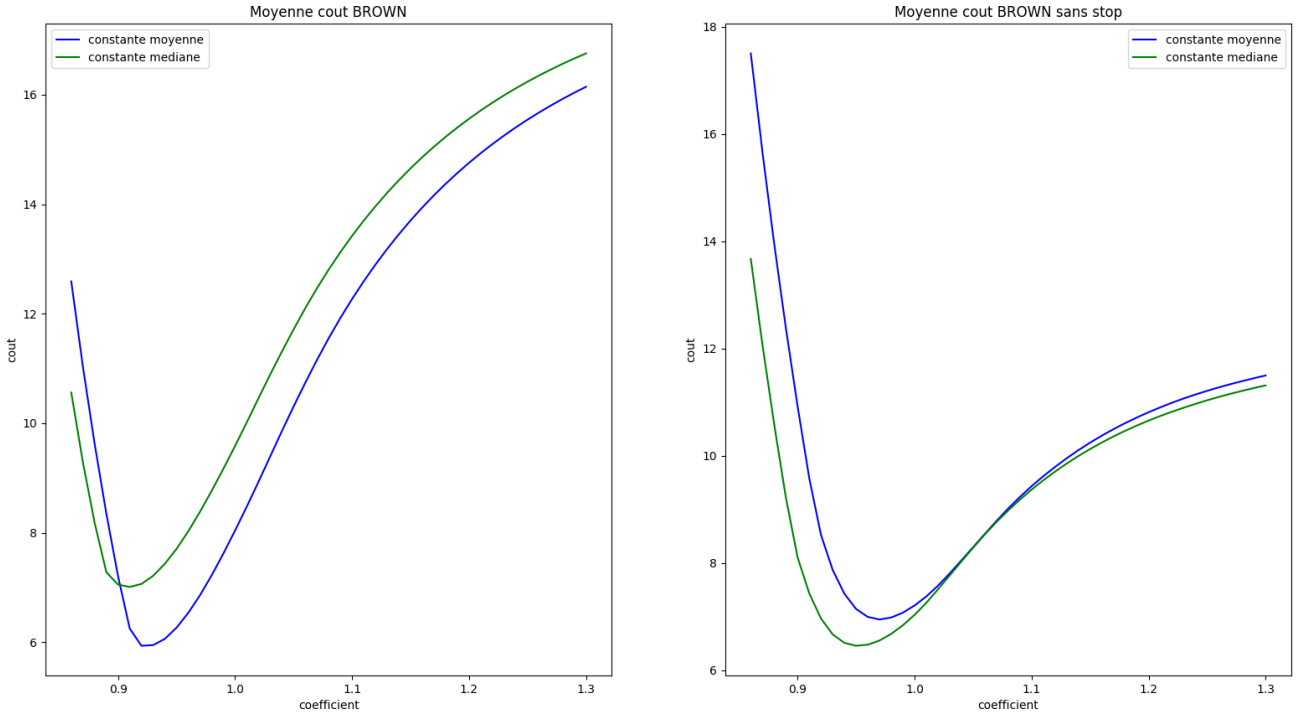


FIGURE 17 – Coût absolu moyen par coefficient

— 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.

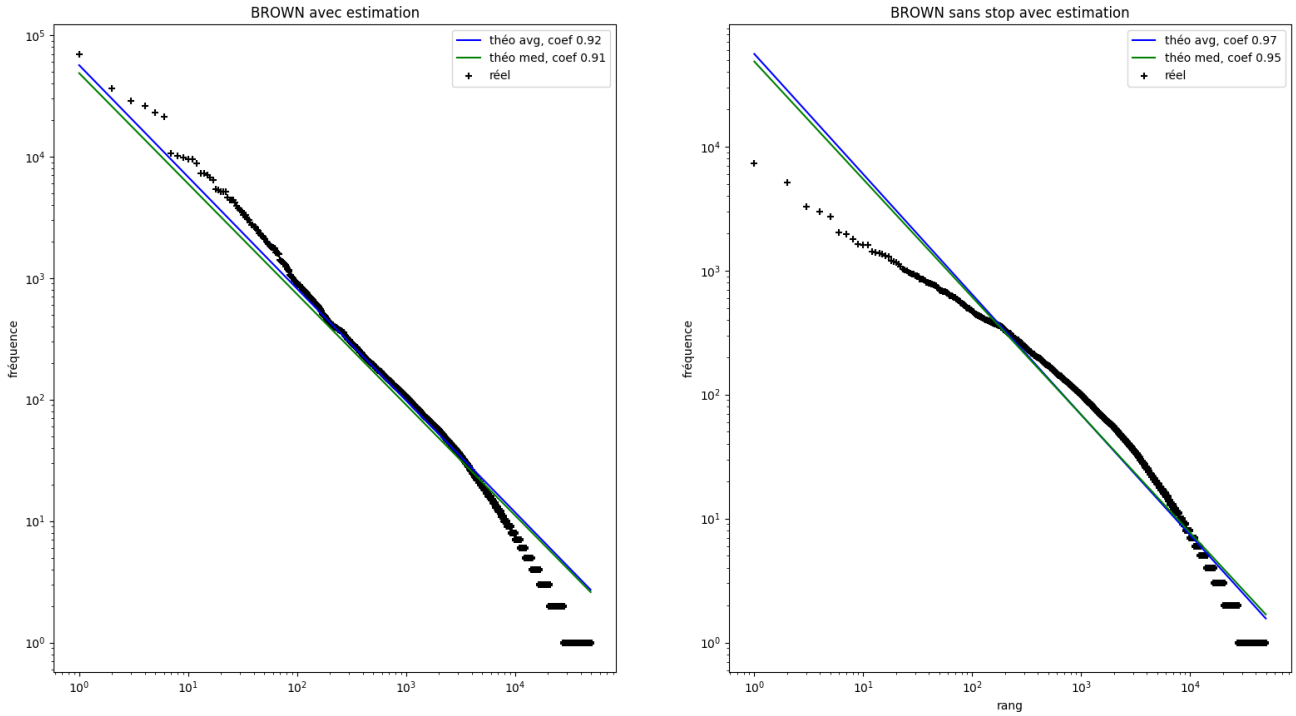


FIGURE 18 – Distribution de Zipf avec les estimations

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

Notes : L'ensemble des codes sources pour cette partie est disponible dans les fichiers :

- ./analyse/rech_zipf.py
- ./traitement/stats.py

B Déploiement des bases de données

Cette annexe détaille la mise en place de l'infrastructure de base de données pour le projet. J'utilise 2 environnements de base de données en version conteneur (docker version 23.0.4) :

- ElasticSearch
  - 1 Node Elastic, pour le service de base de données
  - 1 Service Kibana, pour la visualisation des index
  - 1 Service de certificat, pour sécuriser les échanges
- PostgreSQL
  - 1 service PostgreSQL, pour la base de données
  - 1 service PgAdmin, pour la visualisation de la base

Chaque environnement est indépendant, et possède une ouverture sur le PC hôte.

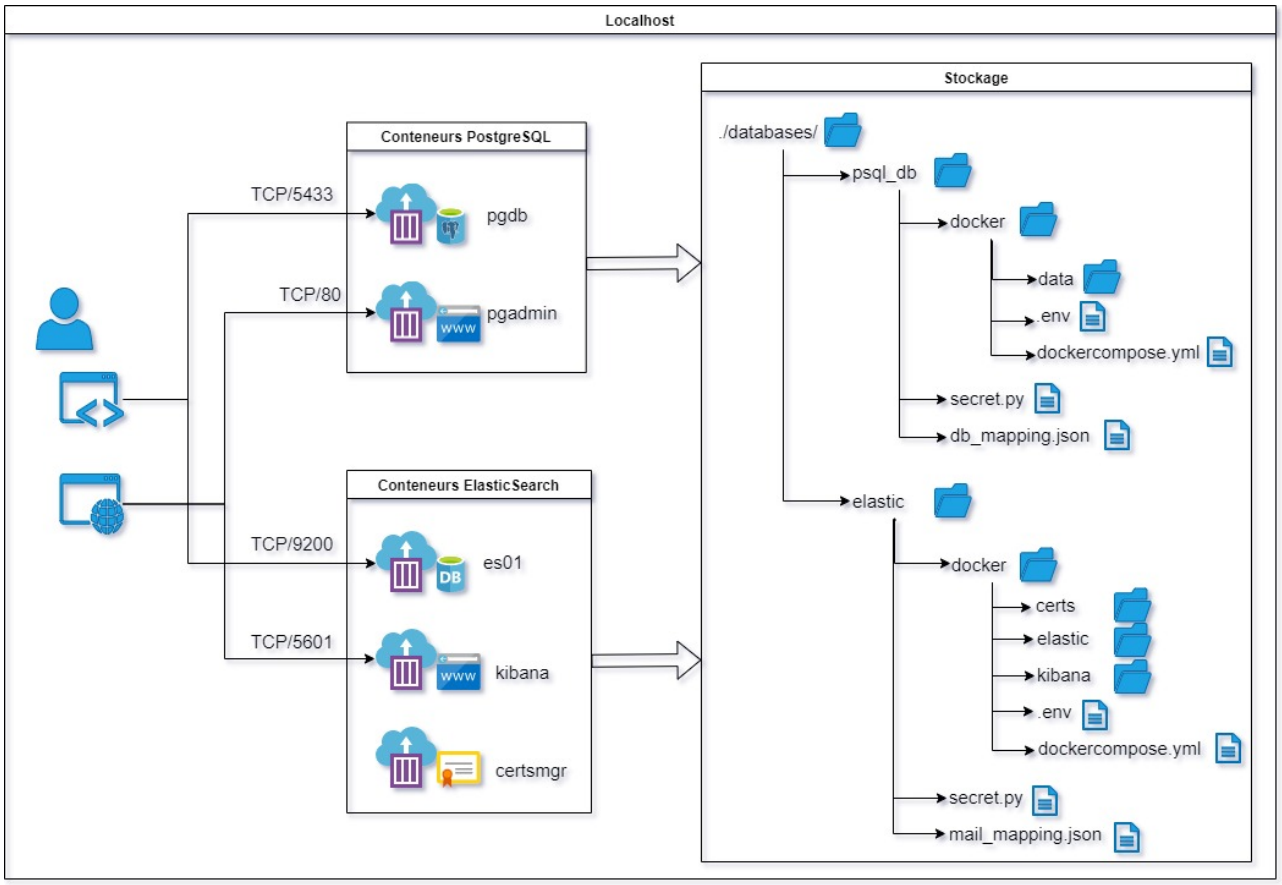


FIGURE 19 – Schéma de l'architecture Docker

B.1 ElasticSearch

Cet environnement se lance avec la commande suivante :

```
docker compose -f ./databases/elastic/docker/docker-compose.yml up -d
```

B.1.1 Conteneurisation

DockerCompose

```
1 version: "3.2"
2
3 services:
```

```
4       certsmgr:
5               image: elasticsearch:${VERSION}
6               volumes:
7                       - ./certs:/usr/share/elasticsearch/config/certs
8               user: "0"
9               command: >
10                      bash -c '
11                              if [ x${ELASTIC_PASSWORD} == x ]; then
12                                      echo "Set the ELASTIC_PASSWORD
    environment variable in the .env file";
13                                      exit 1;
14                              elif [ x${KIBANA_PASSWORD} == x ]; then
15                                      echo "Set the KIBANA_PASSWORD
    environment variable in the .env file";
16                                      exit 1;
17                              fi;
18                              if [ ! -f certs/ca.zip ]; then
19                                      echo "Creating CA";
20                                      bin/elasticsearch-certutil ca --
    silent --pem -out config/certs/ca.zip;
21                                      unzip config/certs/ca.zip -d
    config/certs;
22                              fi;
23                              if [ ! -f certs/certs.zip ]; then
24                                      echo "Creating certs";
25                                      echo -ne \
26                                              "instances:\n"\
27                                              "  - name: es01\n"\
28                                              "    dns:\n"\
29                                              "      - es01\n"\
30                                              "      - localhost\n"\
31                                              "    ip:\n"\
32                                              "      - 127.0.0.1\n"\
33                                             > config/certs/instances.
    yml;
34                                      bin/elasticsearch-certutil cert --
    silent --pem -out config/certs/certs.zip --in config/certs/instances.
    yml --ca-cert config/certs/ca/ca.crt --ca-key config/certs/ca/ca.key;
35                                      unzip config/certs/certs.zip -d
    config/certs;
36                              fi;
37                              # echo "Setting file permissions"
38                              echo "chown -R root:root config/certs";
39                              echo "find . -type d -exec chmod 750 \{\}
    \;";
40                              echo "find . -type f -exec chmod 640 \{\}
    \;";
41                              echo "Waiting for Elasticsearch
    availability";
42                              until curl -s --cacert config/certs/ca/ca.
    crt https://es01:9200 | grep -q "missing authentication credentials";
    do sleep 30; done;
43                              echo "Setting kibana_system password";
```

```
44                                until curl -s -X POST --cacert config/
    certs/ca/ca.crt -u elastic:${ELASTIC_PASSWORD} -H "Content-Type:
    application/json" https://es01:9200/_security/user/kibana_system/
    _password -d "{\"password\":\"${KIBANA_PASSWORD}\"}" | grep -q "^{}";
    do sleep 10; done;
45                                echo "All done!";
46                        '
47                healthcheck:
48                        test: ["CMD-SHELL", "[ -f config/certs/es01/es01.
    crt ]"]
49                        interval: 1s
50                        timeout: 5s
51                        retries: 120
52
53
54        es01:
55                depends_on:
56                        certsmgr:
57                                condition: service_healthy
58                image: elasticsearch:${VERSION}
59                volumes:
60                        - ./certs:/usr/share/elasticsearch/config/certs
61                        - ./elastic/data:/usr/share/elasticsearch/data
62                ports:
63                        - ${ES_PORT}:9200
64                environment:
65                        - discovery.type=single-node
66                        - ELASTIC_PASSWORD=${ELASTIC_PASSWORD}
67                        - bootstrap.memory_lock=true
68                        - xpack.security.enabled=true
69                        - xpack.security.http.ssl.enabled=true
70                        - xpack.security.http.ssl.key=certs/es01/es01.key
71                        - xpack.security.http.ssl.certificate=certs/es01/
    es01.crt
72                        - xpack.security.http.ssl.certificate_authorities=
    certs/ca/ca.crt
73                        - xpack.security.http.ssl.verification_mode=
    certificate
74                        - xpack.security.transport.ssl.enabled=true
75                        - xpack.security.transport.ssl.key=certs/es01/es01
    .key
76                        - xpack.security.transport.ssl.certificate=certs/
    es01/es01.crt
77                        - xpack.security.transport.ssl.
    certificate_authorities=certs/ca/ca.crt
78                        - xpack.security.transport.ssl.verification_mode=
    certificate
79                        - xpack.license.self_generated.type=${LICENSE}
80                mem_limit: ${MEM_LIMIT}
81                ulimits:
82                        memlock:
83                                soft: -1
84                                hard: -1
```

```
85                 healthcheck:
86                         test:
87                                 [
88                                         "CMD-SHELL",
89                                         "curl -s --cacert config/certs/ca/
    ca.crt https://localhost:9200 | grep -q 'missing authentication
    credentials'",
90                                 ]
91                         interval: 10s
92                         timeout: 10s
93                         retries: 120
94
95         kibana:
96                 depends_on:
97                         es01:
98                                 condition: service_healthy
99                 image: kibana:${VERSION}
100                volumes:
101                - ./certs:/usr/share/kibana/config/certs
102                - ./kibana/data:/usr/share/kibana/data
103                ports:
104                - ${KIBANA_PORT}:5601
105                environment:
106                        - SERVERNAME=kibana
107                        - ELASTICSEARCH_HOSTS=https://es01:9200
108                        - ELASTICSEARCH_USERNAME=kibana_system
109                        - ELASTICSEARCH_PASSWORD=${KIBANA_PASSWORD}
110                        - ELASTICSEARCH_SSL_CERTIFICATEAUTHORITIES=config/
    certs/ca/ca.crt
111                mem_limit: ${MEM_LIMIT}
112                healthcheck:
113                        test:
114                                [
115                                        "CMD-SHELL",
116                                        "curl -s -I http://localhost:5601
    | grep -q 'HTTP/1.1 302 Found'",
117                                ]
118                        interval: 10s
119                        timeout: 10s
120                        retries: 120
121
```

Fichier d'environnement

```
1  # Password for elastic user
2  ELASTIC_PASSWORD=XXXXXXXXX
3  # Password for kibana_system user
4  KIBANA_PASSWORD=XXXXXXXXX
5  # Version elastic product
6  VERSION=8.1.2
7  # Licence to use
8  LICENSE=basic
9  # Port for elastic HTTP API
10 ES_PORT=9200
```

```
#ES_PORT=127.0.0.1:9200
# Port for kibana access
KIBANA_PORT=5601
# Memory available (bytes)
MEM_LIMIT=322122547

```

B.1.2 Initialisation de l'index

Exemples de secrets

```
serveur = "https://localhost:9200"
apiid = "XXXXXXXXXXXXXXXXXXXXXXX"
apikey = "XXXXXXXXXXXXXXXXXXXXXX"
ca_cert = "databases/elastic/docker/certs/ca/ca.crt"

```

Il est possible de générer une clé API via l'interface Kibana > Stack Management > Sécurité > API keys > Create API Key > JSON.

Mapping de l'index

```
{
  "properties": {
    "hash": {"type": "keyword"},
    "categorie": {"type": "keyword"},
    "sujet": {"type": "text"},
    "expediteur": {"type": "text"},
    "message": {"type": "text"},
    "langue": {"type": "keyword"}
  }
}

```

Fonctions Python utiles

```
def es_connect(server, creds, crt):
    """ Connexion au serveur ElasticSearch """
    client = Elasticsearch(server, api_key=creds, ca_certs=crt)

    try:
        client.search()
        client.indices.get(index="*")
        return client

    except (exceptions.ConnectionError, AuthenticationException,
    AuthorizationException) as err:
        print("ES:conn - Informations client ElasticSearch :\n\t", err)
        client.close()
        return None


def es_create_indice(es_cli, index, mapping):
    """ Creer un indice s'il n'existe pas deja """
    indices = es_cli.indices.get(index='*')
```

```
    if indices and index in indices:
        print("Warning: Indice {} deja present".format(index), end=' ')
        return

    try:
        res = es_cli.indices.create(index=index, mappings=mapping)
    except elasticsearch.ApiError as err:
        print(err)
        return

    if not res['acknowledged']:
        print("Error : Echec de la creation de l'indice {}".format(index))

```

B.2 PostgreSQL

Cet environnement se lance avec la commande suivante :

```
docker compose -f ./databases/psql_db/docker/docker-compose.yml up -d
```

Le programme se charge automatiquement de créer le compte utilisé pour se connecter à la base.

B.2.1 Conteneurisation

DockerCompose

```
version: "3.3"
services:
        pgdb:
                image: postgres:${PS_VERSION}
                restart: always
                environment:
                        POSTGRES_PASSWORD: ${PS_PASSWORD}

                volumes:
                        - ./data:/var/lib/postgresql/data
                ports:
                        - ${PS_PORT}:5432

        pgadmin:
                image: dpage/pgadmin4:${PS_VERSION}
                environment:
                        PGADMIN_DEFAULT_EMAIL: ${PGA_MAIL}
                        PGADMIN_DEFAULT_PASSWORD: ${PGA_PASSWORD}
                ports:
                        - ${PGA_PORT}:80
                depends_on:
                        - pgdb

```

Fichier d'environnement

```
PS_VERSION=latest
```

```
2
3 PS_PASSWORD=XXXXXXXXX
4 PS_PORT=5433
5
6 PGA_MAIL=data@data.org
7 PGA_PASSWORD=XXXXXX
8 PGA_PORT=80
9
```

B.2.2 Initialisation de la base de données

Exemples de secrets.py

```
1 owner = "XXX"
2 owner_pw = "XXX"
3 admin = "XXXX"
4 admin_pw = "XXXX"
5 host = "localhost"
6 port = "5432"
7
```

Mapping

```
1 {
2   "mail_features": {
3     "categories": {
4       "id_cat": ["SERIAL", "PRIMARY KEY"],
5       "type": ["VARCHAR", "UNIQUE", "NOT NULL"]
6     },
7     "messages": {
8       "id_message": ["SERIAL", "PRIMARY KEY"],
9       "hash": ["CHAR(32)", "UNIQUE", "NOT NULL"],
10      "id_cat": ["INT", "NOT NULL"],
11      "fk": ["fk_message", "id_cat", "categories(id_cat)", "SET NULL"]
12    },
13    "liens": {
14      "id_message": ["INT"],
15      "url": ["INT"],
16      "mail": ["INT"],
17      "tel": ["INT"],
18      "nombre": ["INT"],
19      "prix": ["INT"],
20      "fk": ["fk_liens", "id_message", "messages(id_message)", "CASCADE"]
21    },
22    "stats_mots": {
23      "id_message": ["INT"],
24      "mots_uniques": ["INT"],
25      "mots": ["INT"],
26      "char": ["INT"],
27      "char_maj": ["INT"],
28      "mot_maj": ["INT"],
29      "mot_cap": ["INT"],
30      "fk": ["fk_stats_mot", "id_message", "messages(id_message)", "
    CASCADE"]
```

```
    },
    "stat_ponct": {
      "id_message": ["INT"],
      "point": ["INT"],
      "virgule": ["INT"],
      "exclamation": ["INT"],
      "interrogation": ["INT"],
      "espace": ["INT"],
      "tabulation": ["INT"],
      "ligne": ["INT"],
      "ligne_vide": ["INT"],
      "fk": ["fk_stats_ponct", "id_message", "messages(id_message)", "CASCADE"]
    },
    "zipf": {
      "id_message": ["INT"],
      "constante": ["INT"],
      "coefficient": ["REAL"],
      "tx_erreur": ["REAL"],
      "fk": ["fk_zipf", "id_message", "messages(id_message)", "CASCADE"]
    },
    "hapax": {
      "id_message": ["INT"],
      "ratio_unique": ["REAL"],
      "ratio_texte": ["REAL"],
      "fk": ["fk_hapax", "id_message", "messages(id_message)", "CASCADE"]
    }
  }
}
```

Fonctions utiles

```
def create_db(nom, owner, user, passwd, host, port):
    """ Creer une nouvelle base de donnees """
    client_psql = psycopg2.connect(user=user, password=passwd, host=host, port=port)
    client_psql.autocommit = True
    cursor = client_psql.cursor()

    cursor.execute("DROP DATABASE IF EXISTS {};".format(nom))
    cursor.execute("CREATE DATABASE {};".format(nom))
    cursor.execute("ALTER DATABASE {} OWNER TO {};".format(nom, owner))
    client_psql.close()


def connect_db(database, user, passwd, host, port):
    """ Connexion a la base de donnees Postgres. Penser a fermer la connexion """
    try:
        client_psql = psycopg2.connect(database=database, user=user, password=passwd, host=host, port=port)
    except psycopg2.Error as e:
        print("Erreur de connexion : \n{}".format(e), file=sys.stderr)
```

```
19          return None
20
21      client_psql.autocommit = True
22      return client_psql
23
24
25  def create_table(client_psql, nom, champs):
26      """ Creer une nouvelle table dans la base de donnees """
27      fk = ""
28      fields = []
29
30      curseur = client_psql.cursor()
31      curseur.execute(f"DROP TABLE IF EXISTS {nom}")
32
33      if 'fk' in champs.keys():
34          ls = champs.pop('fk')
35          fk = f"CONSTRAINT {ls.pop(0)} FOREIGN KEY({ls.pop(0)}) REFERENCES
    {ls.pop(0)}"
36          if ls:
37              fk += f" ON DELETE {ls.pop(0)}"
38
39      for key, value in champs.items():
40          fields.append(f"{key} {' '.join(value)}")
41
42      query = f"CREATE TABLE {nom} ({', '.join(fields)})"
43      if fk:
44          query = query[:-1] + f", {fk})"
45      curseur.execute(query)
46
47
48  def create_index(client_psql, nom, table, colonne):
49      """ Index sur une colonne """
50      query = "CREATE UNIQUE INDEX {} ON {}({})".format(nom, table, colonne)
51      exec_query(client_psql, query)
52
```

C Tableau des choix technologiques

Élément	Retenu	Raisons	Observations
Datasets			
Mail de la compagnie Enron	Non	Mails non classés	Non retenu pour la phase de développement car pas de moyen fiable de contrôler la sortie automatiquement
Mail du projet SpamAssassin	Oui	Mails déjà pré-triés	Mails principalement en Anglais déjà pré-trié en catégorie Spam et Ham
Brown dataset (nltk)	Oui	Corpus d'un million de mots en Anglais publié en 1961	Dataset utilisé pour le développement de la visualisation de la distribution de Zipf
Stopwords (nltk)	Oui	Corpus de mots commun non significatif dans un texte	Utilisation dans le développement de la visualisation de la distribution de Zipf
Langage et Modules			
Python	Oui	Langage polyvalent pour le traitement des données	
Module email	Oui	Module natif pour le traitement des mails	Grande flexibilité pour la lecture des mails
Virtualisation			
Docket	Oui	configuration et environnement dans les fichiers. Cela facilite le portage vers d'autre environnement	Lors du développement j'ai pu utilisé mon PC personnel sous Linux et un autre PC sous Windows. L'utilisation de Docker m'a évité de nombres configurations sur Windows
Bases de données			
ElasticSearch	Oui	Technologie utilisée dans mon entreprise. Présence d'une interface de visualisation des données Kibana.	Application dockerisée.
PostgreSQL	Oui	Moteur de base de données relationnelle plus facilement scalable que ElasticSearch pour l'ajout de nouvelle catégorie de données. Il n'est pas nécessaire de ré-indexer toute la base pour ajouter des champs	Application dockerisée
SQLite	Oui	Base de données légère pour stocker uniquement les données statistiques des étapes de la phase 1	Rapide à mettre en place et déjà intégrée

D Modèles

D.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. [3]

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 8 – 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

3. Les données présentées sont inventées

TABLE 9 – 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 10 – 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 11 – Probabilité présence du joueur selon la météo

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

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

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

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

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

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

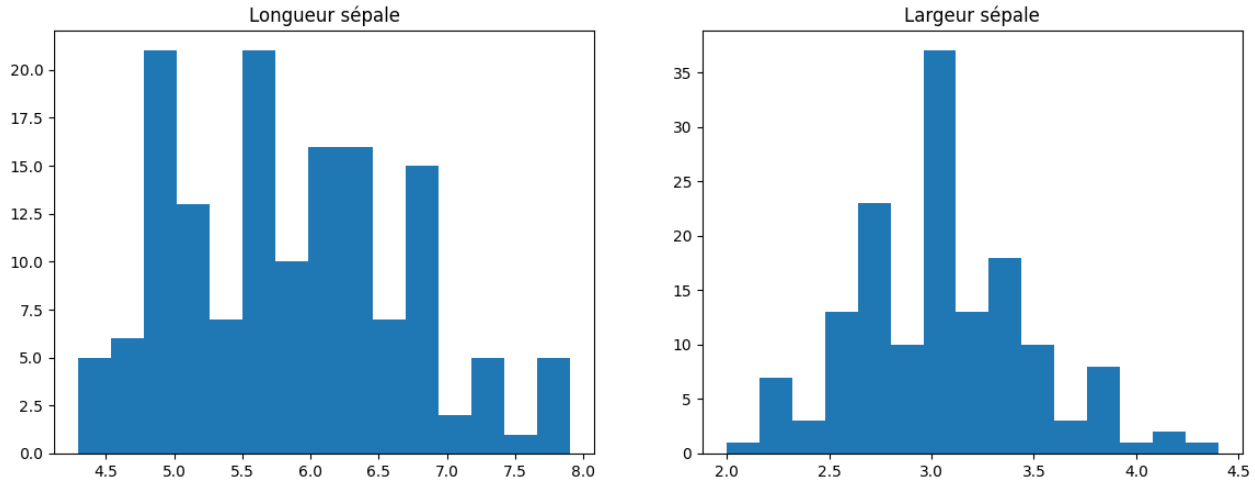


FIGURE 20 – Distribution des longueurs et largeurs des sépales

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

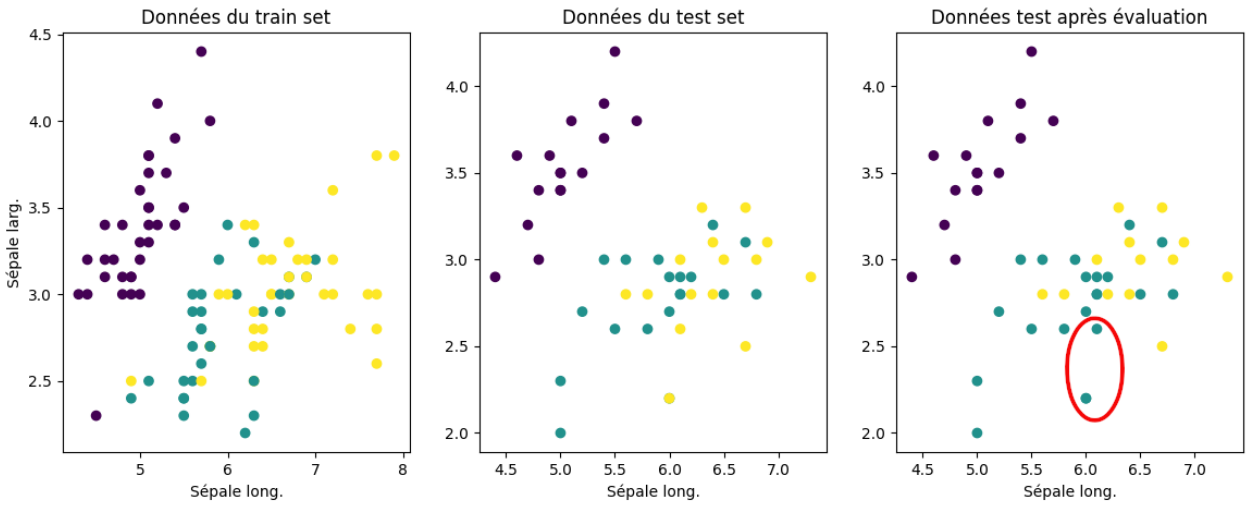


FIGURE 21 – Représentation des données

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

— Précision : 0.96 [4]
— Rappel : 0.96 [5]
— Score F1 : 0.9604285714285714 [6]

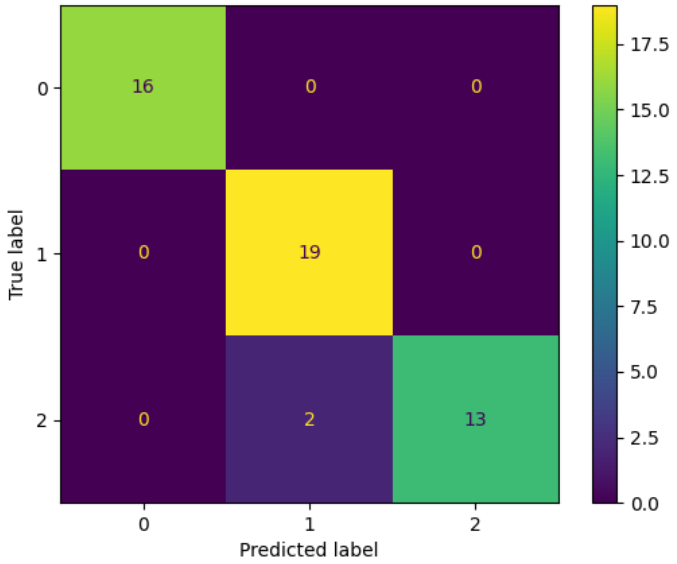


FIGURE 22 – Matrice de confusion

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

4. La précision est la proportion des éléments correctement identifiés sur l'ensemble des éléments prédit
5. Le rappel est la proportion des éléments correctement identifiés sur l'ensemble des éléments de la catégorie
6. 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.

D.2 Réseau de neurones

D.3 TensorFlow

E Bibliographie

Références

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

F Sitotec

F.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)

F.2 Modules

Module langdetect

- 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)

F.3 Modèles

Naïves Bayes Le modèle Naïves Bayes est employé dans le module langdetect (F.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)

G Codes sources

G.1 Github

Le lien vers l'ensemble des sources est disponible en publique via le lien ci dessous : https://github.com/peredur0/mercury

G.2 Analyse statistiques de la phase 1

Code pour l'affichage des statistiques de la phase 1

```
import matplotlib.pyplot as plt
import pandas as pd

from databases.psql_cmd import connect_db, exec_query
from databases.psql_db import secrets as psql_secrets


def get_p1_data():
    psql_cli = connect_db('mail_features_prod', psql_secrets.owner, psql_secrets.owner_pw,
                          psql_secrets.host, psql_secrets.port)

    column = ['id_message', 'type', 'url', 'mail', 'tel', 'nombre', 'prix']
    query = """select m.id_message, c.type, l.url, l.mail, l.tel, l.nombre, l.prix from messages
    as m
    join categories as c on m.id_cat = c.id_cat
    join liens as l on m.id_message = l.id_message"""

    df = pd.DataFrame(exec_query(psql_cli, query), columns=column)
    psql_cli.close()

    return df.set_index('id_message')


def set_bar_graph(data, feat, subplot, pos):
    df = data[data[feat] < 20].groupby(['type', data[feat]]).size()
    df.unstack(0).plot(kind='bar', ax=subplot[pos])


if __name__ == '__main__':
    df_all = get_p1_data()
    df_spam = df_all[df_all['type'] == 'spam']
```

```
32      df_ham = df_all[df_all['type'] == 'ham']
33
34      d_pie = df_all.groupby(['type']).size()
35      fig, ax = plt.subplots()
36      fig.suptitle('Repartition des ham/spam')
37      ax.pie(d_pie, labels=d_pie.index, autopct='%1.1f%%')
38      plt.show()
39
40      print("Statistiques Liens")
41      print("Gobales: \n", df_all.describe())
42      print("Ham: \n", df_ham.describe())
43      print("Spam: \n", df_spam.describe())
44
45      fig, ax = plt.subplots(nrows=5, ncols=1)
46      fig.suptitle("Distribution des mails en fonction du nombre de liens")
47      fig.tight_layout(pad=0.5)
48      position = 0
49      for feat in ['url', 'mail', 'tel', 'nombre', 'prix']:
50          set_bar_graph(df_all, feat, ax, position)
51          position += 1
52      plt.show()
53
```