Projet L3
Ingénierie des langues
Fouille de données

GOEHRY Martial
16711476

4 octobre 2022

Table des matières

1 Introduction

Ce projet à pour but de permettre de faire la détection des spam et ham dans des mails. 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 en point de départ les éléments suivants :

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

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

2 Récolte des données

2.1 Recherche de dataset

J'avais dans l'idée de faire la recherche de mails en français. Cependant, je n'ai pas trouvé de dataset dans cette langue. Je me suis donc retourné vers les datasets de mails en anglais. J'ai pu alors récupéré 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 SpamAssasin afin de pouvoir vérifier les résultats de l'analyse.

2.2 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.

3 Pré-traitement

Les étapes de pré-traitement regroupe 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 certains traitement de nettoyage et de récupération d'informations sommaires.

3.1 Extraction des corps des mails

L'extraction du corps du mail se fait avec le module natif de python *email*.

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

Fonction d'importation des fichiers

```
def import_from_file(chemin):
    """ Importe un email contenu dans un fichier
    :param chemin: <str> - Chemin vers le fichier
    :return: <email.message.EmailMessage>
    """
    try:
        with open(chemin, 'rb') as data:
            msg = message_from_binary_file(data, policy=policy.default)

    except FileNotFoundError:
        print("Fichier : '{}' non trouve".format(chemin), file=sys.stderr)
        return None

    return msg
```

Une fois le fichier importé au format *EmailMessage*, il est possible d'en extraire le corps

Extraction du corps du mail

```
def extract_body(msg):
    """ Extraire le corps du mail
    :param msg: <email.message.EmailMessage> Mail
    :return: <str> corps du mail
    """
    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
```

Exemple

3.2 Nettoyage

Par regex

Par module

3.3 Mise en base

Stockage des données : ElasticSearch

Stockage des données statistiques du traitement : SQLite

3.4 Recherche de caractéristiques

Références

3.5 Analyse préliminaire

4 Traitement

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 mon analyse, 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: occurences
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 :

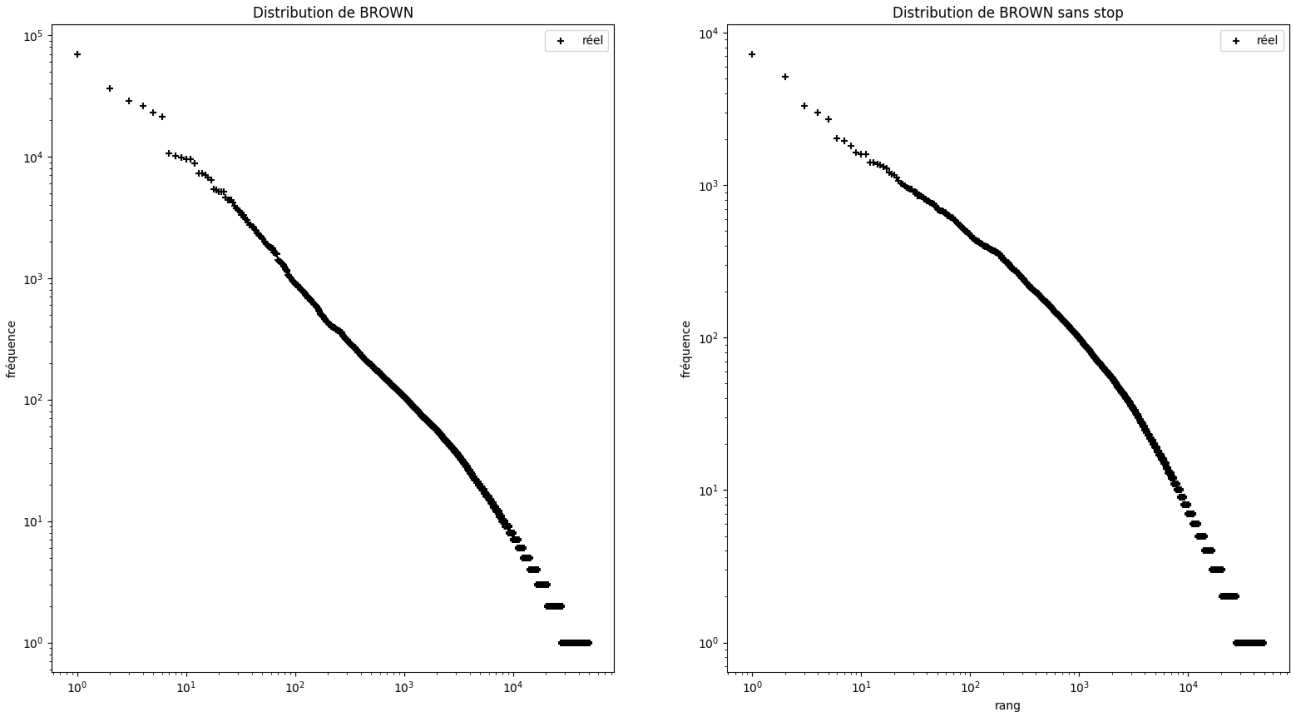

FIGURE 1 – Distribution de Zipf pour les deux corpus

— 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 que je détermine 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.

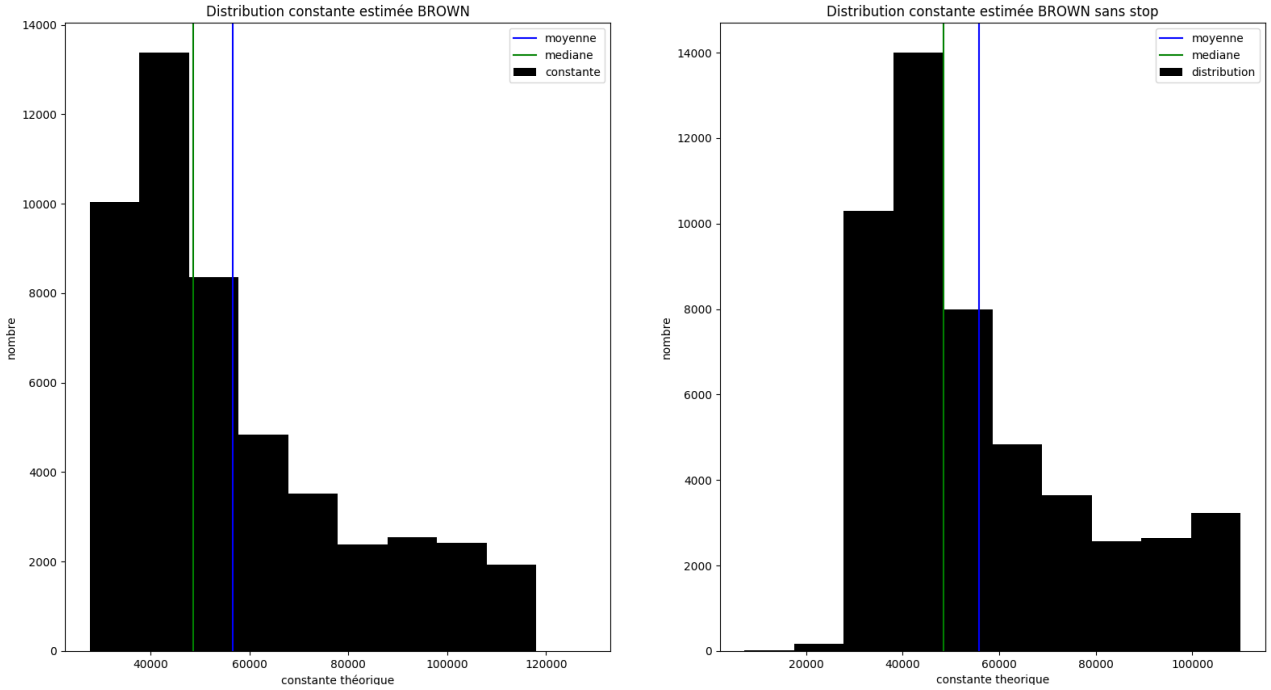

FIGURE 2 – Distribution des constantes théoriques pour les deux corpus

On voit qu'il y a une majorité de mots donnent 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 avec en constante la moyenne et la médiane de chaque corpus.
3. Calculer la moyenne des coûts absolus entre les fréquences théoriques par coefficient avec la fréquence réelle observée pour chaque corpus.

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

Fonctions utilisées dans la recherche du coefficient

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

1. Borne et pas, 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 :

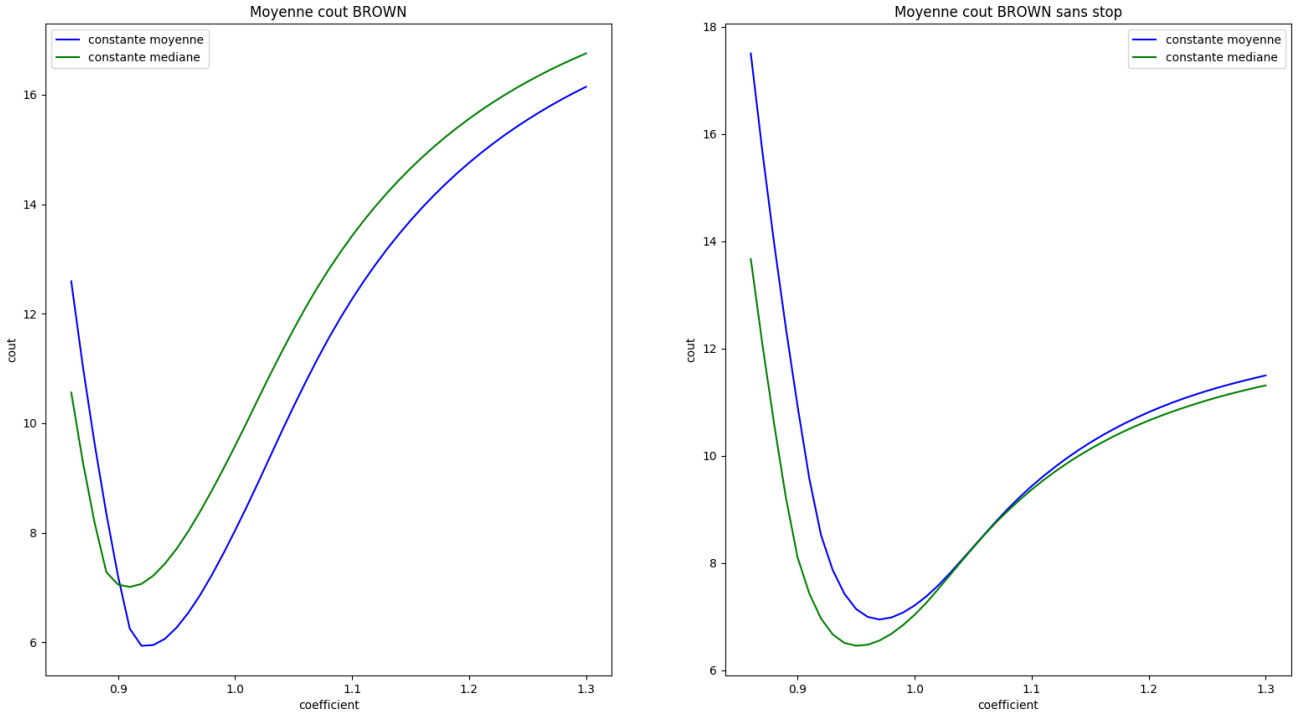

FIGURE 3 – Coût absolu moyen par coefficient

— 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.

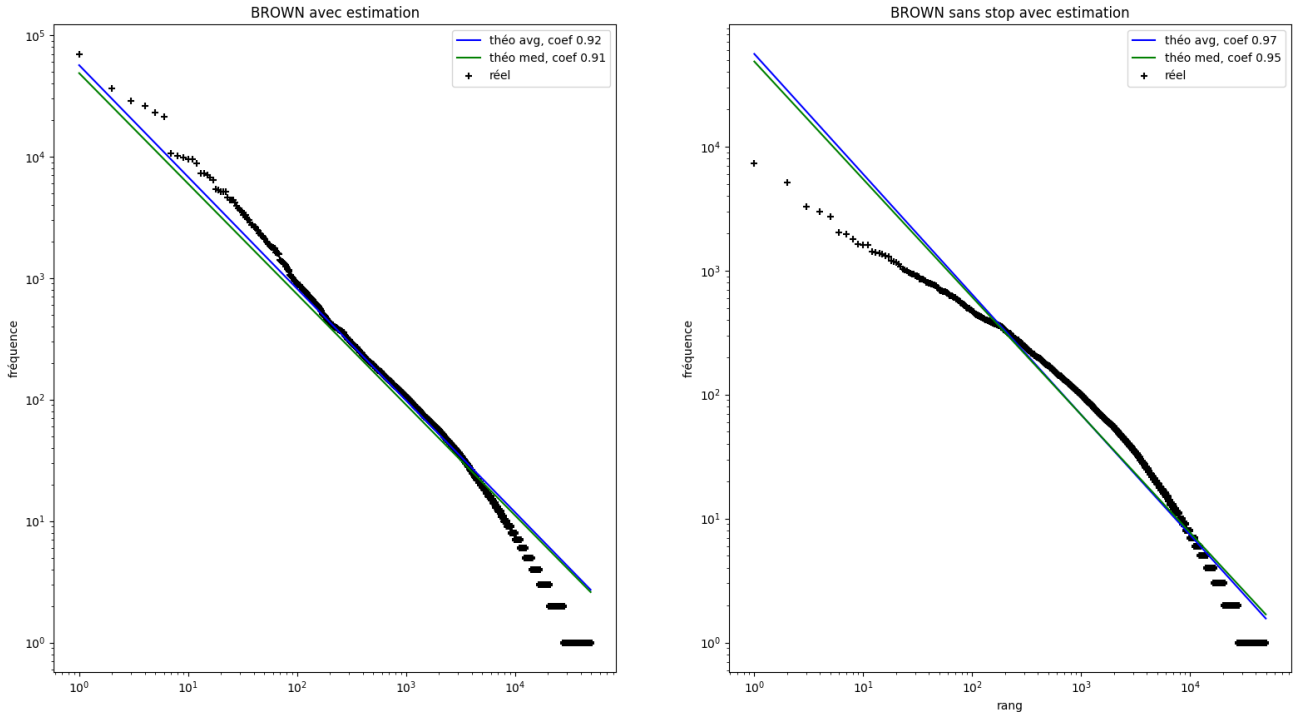

FIGURE 4 – Distribution de Zipf avec les estimations

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 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
Bases de données			
ElasticSearch	Oui	Technologie utilisée dans mon entreprise. Présence d'une interface de visualisation des données Kibana.	Application dockerisée.
SQLite	Oui	Base de données légère pour stocker uniquement les données statistiques des étapes	Rapide à mettre en place et déjà intégrée

C Bibliographie

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)