Université de TOULOUSE III

Département science et ingénierie

Projet Stage

Rapport de stage

soutenu le 29/06/2022

par

Thierno Mbengue

Encadrant universitaire : Mathieu Serrurier

Remerciements

Merci à Monsieur Serrurier ainsi qu'a Blase Vincent d'avoir répondu à nos questions .

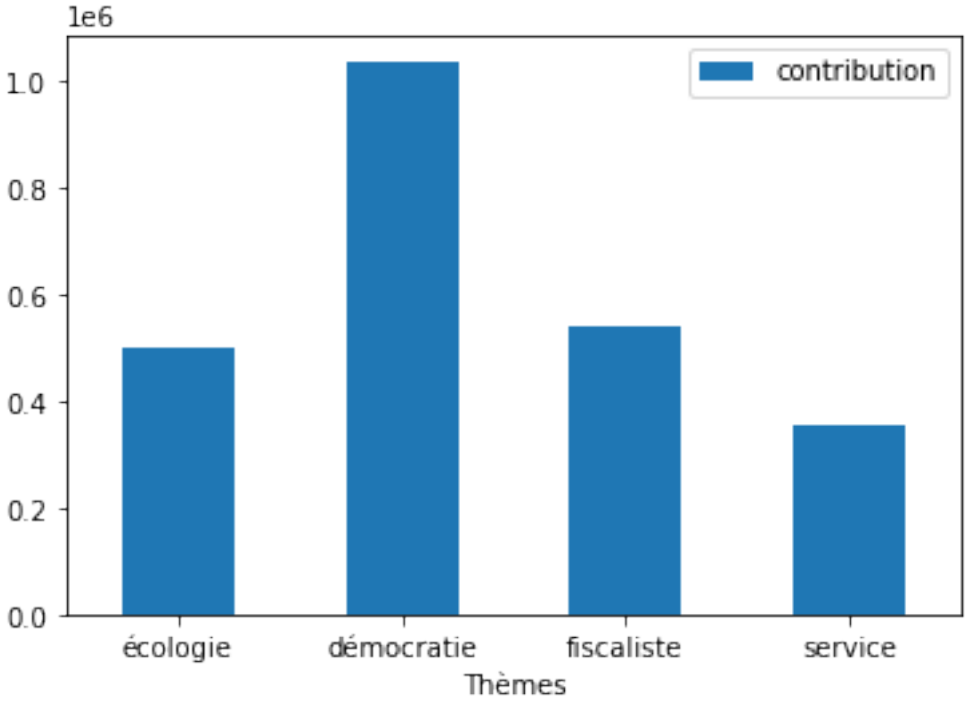
Table des matières

Introduction

Dans le but de valider notre année par un projet de stage obligatoire,nous avons été amenés à faire du Clustering automatique de texte.Le but de ce projet est d'analyser automatiquement les contributions libres du grand débat 2019. Afin de construire des catégories de contributions,le grand débat national est un débat public français lancé le 15 janvier 2019 par le président de la République, Emmanuel Macron, dans le contexte du mouvement des Gilets jaunes.Le but était d'utiliser l'intelligence artificielle pour classer les contributions dans une quinzaine de grandes catégories, charge à l'IA de reconnaître ce qui relève de la critique, de l'avis favorable ou de la simple proposition.C'est ce que l'on va essayer de refaire avec l'ensemble des données recueillies dans le site du grand débat,en nettoyant les données jusqu'à la classification et du clustering.

1 Description des données

Les données sont constituées d'un ensemble de textes libres. Les textes sont divisés en quatre grands thèmes. L'approche est fondée sur la recherche des mots importants du texte. Ces mots importants sont déterminés en fonction de leur fréquence d'apparition dans le corpus (l'ensemble des textes considérés). Une fois que l'ensemble des "n" mots les plus pertinents est défini, chaque texte sera alors décrit comme le vecteur des fréquences d'apparition des mots pertinents. Le but sera de faire une analyse supervisée et/ou non supervisée des données. Voici un diagramme en barre des nombres de contributions pour chaque thème avant nettoyage.



2 Traitement des fichiers

Pour le triatement de fichier texte en python,on peut utiliser plusieurs bibliothèques :

```
import nltk
import math
import string
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from collections import Counter
import pandas as pd
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem.snowball import FrenchStemmer
from wordcloud import WordCloud
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
```

Le bibliotheque NLTK rassemble les algorithmes les plus communs du traitement naturel du langage comme le tokenizing„ le stemming, l'analyse de sentiment, la segmentation de

topic ou la reconnaissance d'entité nommée.Et c'est ce qui va nous permettre de faire les étapes suivantes.

2.1 La tokenization

La tokenization est un processus permettant de réduire ou de diviser un texte en plusieurs sous-parties appelées tokens. Cette méthode permet aussi d'extraire des statistiques depuis le corpus du texte, par exemple le nombre de phrases.

Ces statistiques peuvent être ensuite utilisées pour régler les paramètres lors de l'entraînement d'un modèle. Cette technique sert aussi à trouver des " patterns " dans le texte, indispensables pour effectuer des tâches de traitement naturel du langage. On a utilisé la tokenisation pour supprimer les ponctuations et récupérer l'ensemble des mots de chaque texte sous forme de token.

2.2 Le stemming et la lémmatisation

La méthode du Stemming permet de convertir un ensemble de mots dans une phrase en une séquence. Les mots ayant le même sens qui varient en fonction du contexte sont ainsi normalisés. Le but est de trouver la racine à partir des différentes variations du mot. Le NLTK regroupe plusieurs " stemmers " comme le Porter Stemmer, le Snowball Stemmer et le Lancaster Stemmer.

La technique de Lemmatisation est un processus algorithmique permettant de trouver le lemme d'un mot en fonction de son sens. Il s'agit de l'analyse morphologique des mots, visant à supprimer ses affixes. Sur NTLK, la fonction morph native de WordNet est utilisée pour la Lemmatisation. Mais dans notre cas, la lemmatisation n'est pas intéressante, le bibliothèque NLTK lemmatise que les verbes dans la langue française, c'est pourquoi on a surtout adopté la méthode du stemmer, bien qu'il nous donne des mots qui n'ont pas de sens, mais au moins, il nous permet de ne pas considérer des mots identiques comme des mots différents.

Pour nettoyer nos données, on a utilisé les 3 fonctions :

Pour supprimer les valeurs numériques :

```
def is_valid_string(element):
```

Pour supprimer les stopsword plus quelques mots très fréquents :

```
def stop_word_plus():
```

Pour nettoyer le texte ,obtenir une liste de mots pour chaque texte,un dictionnaire de mots avec leurs fréquences et le nombre de mots pour chaque texte :

```
def Nettoyage_author(df):
  j=0
  texte=''
  lemmatizer = WordNetLemmatizer()
  tokenizer = nltk.RegexpTokenizer(r'\w+')
  stemmer = FrenchStemmer()
  dfFeat = pd.DataFrame(columns=['id','reponse',
  'dictionnaire_mot','nombre_mot'])
  for i in df['author_id'].unique():
    df1=df[df['author_id']==i]
    for tx in df1['reponse']:
      texte+=' '+tx
    texte=tokenizer.tokenize(texte)
    texte=[word.lower() for word in texte]
    texte= list(filter(lambda token:
    nltk.tokenize.punkt.PunktToken(token).is_non_punct, texte))
    texte= list(filter(lambda token:
    token not in stop_word_plus(), texte))
    texte= [stemmer.stem(word) for word in texte]
    texte= list(filter(lambda token:
    token not in stop_word_plus(), texte))
    for mt in texte:
        if len(mt)<=2 or is_valid_string(mt)==True :
            texte.remove(mt)
    dfFeat.loc[j]=[i,texte,Counter(texte),len(texte)]
    texte=''
    j+=1
  return dfFeat
```

3 Analyse de texte

La fonction nettoyer nous retourne une nouvelle datafram ,dans cette datafram on a une colonne dictionnaire qui nous permet de récupérer l'ensemble des mots d'un texte

et leurs fréquences sous forme de collection d'objets.Dans cette partie on a aussi quatre fonctions pour calculer tf*idf. Cette mesure statistique permet d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus.

Cette fonction nous permet de calculer le nombre de texte dans le corpus où le mot apparaît

```
def n_containing(word, liste_document):
  return sum(1. for document in liste_document if word in document)
```

Calcul du tf

```
def tf(word, document):
    return document.count(word) / (len(document) * 1.)
```

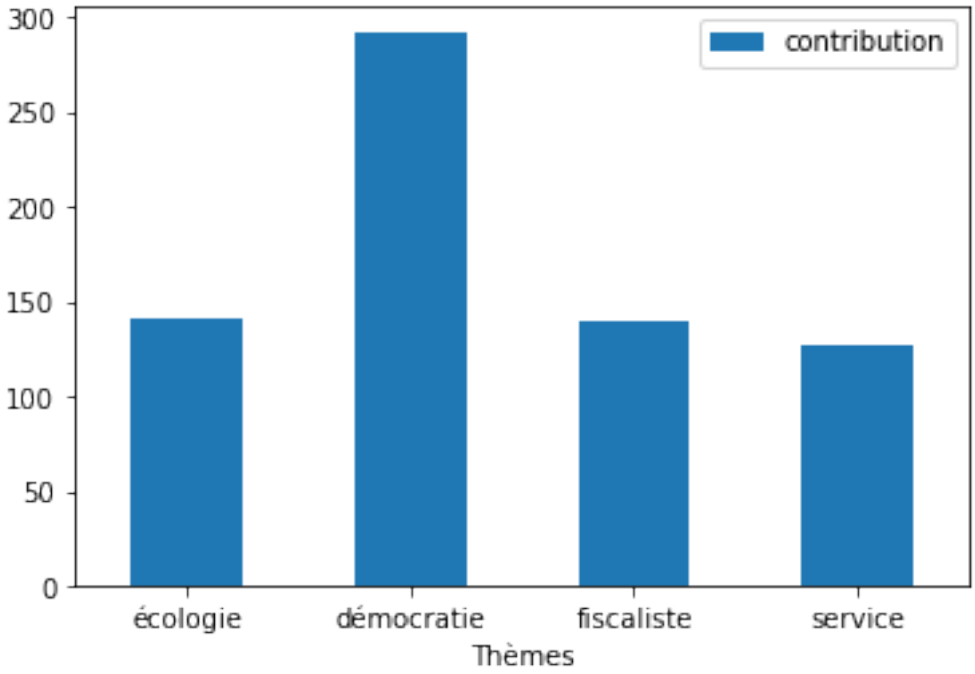
Calcul du idf

```
def idf(word, liste_document):
    return math.log(len(liste_document) / (1. + n_containing(word,
    liste_document)))
```

Cette fonction nous permet de calculer le poids d'un mots dans un corpus

```
def tfidf(word, document, liste_document):
    return tf(word, document) * idf(word, liste_document)
```

Enfin dans cette partie nous avons essayé de représenter un diagrammes en barre pour observer le nombre de mots moyens dans un texte de chaque thème.

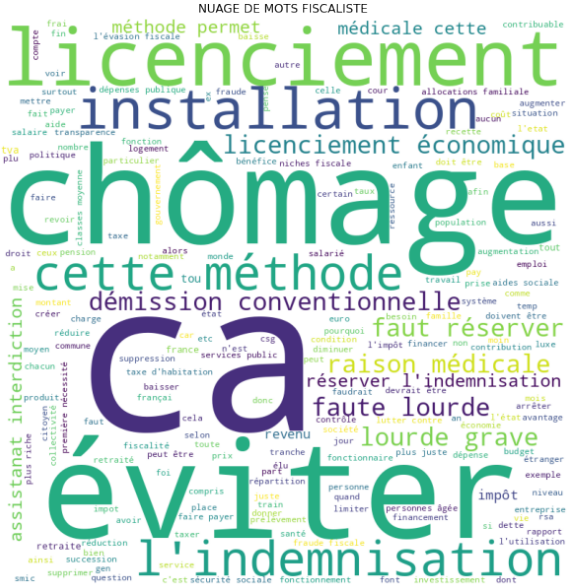


4 Nuage de mots

Le nuage de mots-clés, est une représentation visuelle des mots-clés les plus utilisés sur texte ou corpus de texte.Plus un mot est cité dans le texte ou dans le corpus, plus il apparaîtra en gros dans le nuage que représente le nuage de mots.Pour cela on a créé deux fonctions qui nous permet d'avoir le nuage de mots de nos corpus de départ et de nos corpus après nettoyage.Cela nous permet de retrouver facilement les mots qui n'ont pas de sens après stémmisation de nos corpus.

```
def Nuage_mot(df):
  comment_words = ''
  Stopwords = stopwords.words('french')
  for val in df.reponse:
    comment_words += " ".join(val)+" "

  wordcloud = WordCloud(width = 800, height = 800,
                  background_color ='white',
                  stopwords = Stopwords,
                  min_font_size = 10).generate(comment_words)

  plt.figure(figsize = (8, 8), facecolor = None)
```

```
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



5 Distance entre texte

Pour pouvoir utiliser notre texte, on a besoin de faire la transformation physique (sous forme de vecteur), et pour cela, on peut utiliser l'ensemble des mots dans le corpus ou bien choisir les "n" mots les plus pertinents dans le corpus. Dans notre cas, on travaille avec les "n" mots les plus pertinents. Cela va nous permettre de transformer un texte à un vecteur et de pouvoir faire des algorithmes de classifications, en se basant sur la distance entre les textes.

-Cette fonction nous permet de récupérer chaque texte sous forme string

```
def corpus_document(df):
    corpus=[]
    for liste in df['reponse']:
        corpus.append(" ".join(liste))
    return corpus
```

-cette fonction nous permet de récupérer les "n" mots les plus fréquents avec leurs fréquences sous forme de tuple

```
def get_top_n_words(corpus, n=None):
    vec = CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx])
    for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
```

-Cette fonction nous permet de transformer un texte en un vecteur basé sur tf*idf ,la fonction prend en argument le corpus de texte et la liste des mots pertinents

```
def vectorisation(df,liste):
    corpus=corpus_document(df)
    # instantiate the vectorizer object
    tfidfvectorizer =
    TfidfVectorizer(analyzer='word',stop_words= stopwords.words('french'),
    vocabulary=liste)
    # convert th documents into a matrix
    tfidf_wm = tfidfvectorizer.fit_transform(corpus)
```

```
    get_feature_names() methods)
    count_tokens = tfidfvectorizer.get_feature_names()
    tfidf_tokens = tfidfvectorizer.get_feature_names()
    df_tfidfvect = pd.DataFrame(data = tfidf_wm.toarray(),
    index = [i for i in range(1,len(df)+1)],columns = tfidf_tokens)

    return df_tfidfvect,tfidf_wm
```

6 Classification

La tâche de classification est, avec la régression, une des deux principales tâches de Machine Learning en apprentissage supervisé. Elle signifie associer à chaque donnée une étiquette (ou label) parmi un ensemble de labels possibles (ou catégories). Dans les cas les plus simples, il n'y a que deux catégories, c'est une classification binaire ou binomiale. Sinon, il s'agit d'une classification en classes multiples, aussi appelée classification multiclasse. Les catégories doivent être déterminées avant toute forme d'apprentissage. De plus, les données servant à l'apprentissage doivent toutes recevoir un label, permettant de connaître la réponse attendue : il s'agit d'un apprentissage supervisé.
Avec Scikit-learn, le processus sera toujours le même :

-Créer un modèle en lui indiquant les paramètres souhaités. Dans le cas de la classification, la classe sera un classifié (dans l'exemple, il s'agit d'un TreeClassifier, c'est-à-dire un arbre de décision) ;

-Faire l'apprentissage grâce à fit, en lui fournissant les données X et y d'apprentissage (il est possible d'utiliser de la validation croisée ou une opti. misation des hyperparamètres) ;

-Prédire des résultats sur le dataset souhaité grâce à predict (à partir de données X. obtention des données prédites) ;

-Appeler les différentes métriques souhaitées (présentes dans sklearn.metrics avec en paramètres les résultats attendus et les données prédites).

-Cette fonction nous permet de créer une nouvelle data avec une colonne classe qui prend 0,1,2,3 qui représente respectivement service,démocratie,écologie et fiscalité.

```
s=len(serv)
d=len(demo)
e=len(eco)
f=len(fis)
def creation_classe(vec):
    liste=vec.columns
    col=[]
    for m  in liste:
        col.append(m)
    col.append('classe')
    dfFeat = pd.DataFrame(columns=col)
    val=[]
    j=0
    for i in range(1,len(df)+1):
        if i<=s:
            index=0
        if i>s and i<=d:
            index=1
        if i>d and i<=e:
            index=2
        if i>e and i <=f:
            index=3

        for mot in liste:
            val.append(vec[mot][i])
        val.append(index)
        dfFeat.loc[j]=val
        j+=1
        val=[]
    return dfFeat
```

Les modules utilisés pour faire la classification avec différentes méthodes

```
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
```

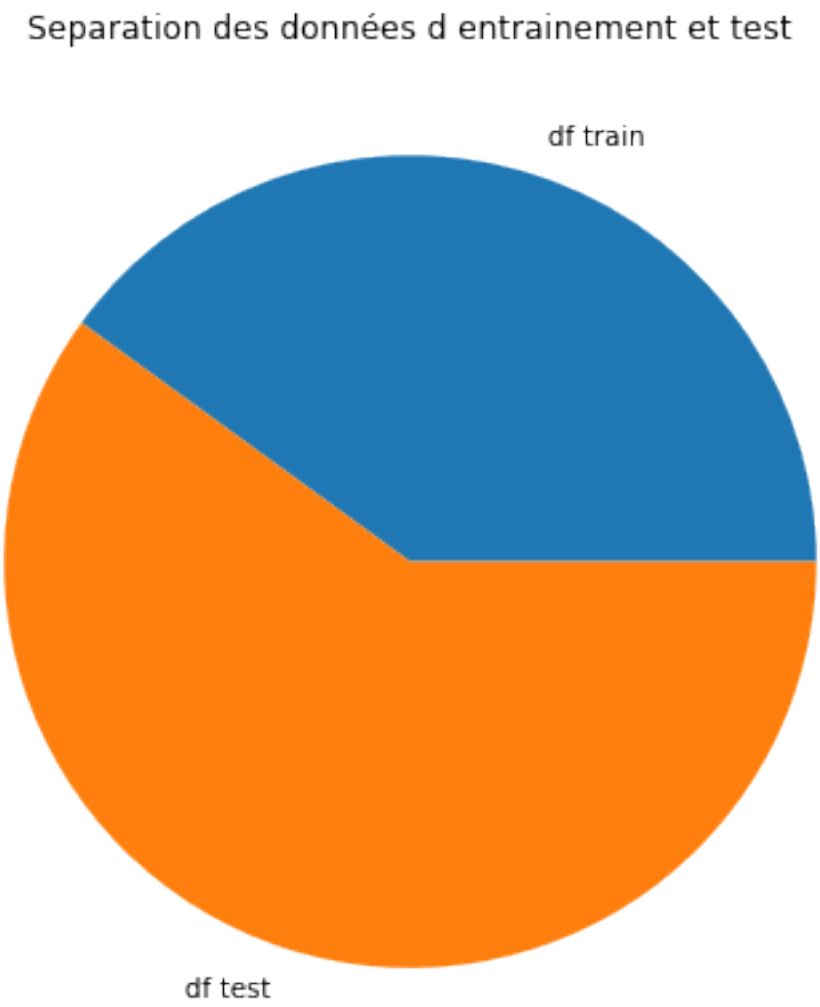
```
from sklearn.metrics import accuracy_score
```

```
def Melange(t):
    df_shuffled=t.iloc[numpy.random.permutation(t.index)].reset_index(drop=Tru
    return df_shuffled
```

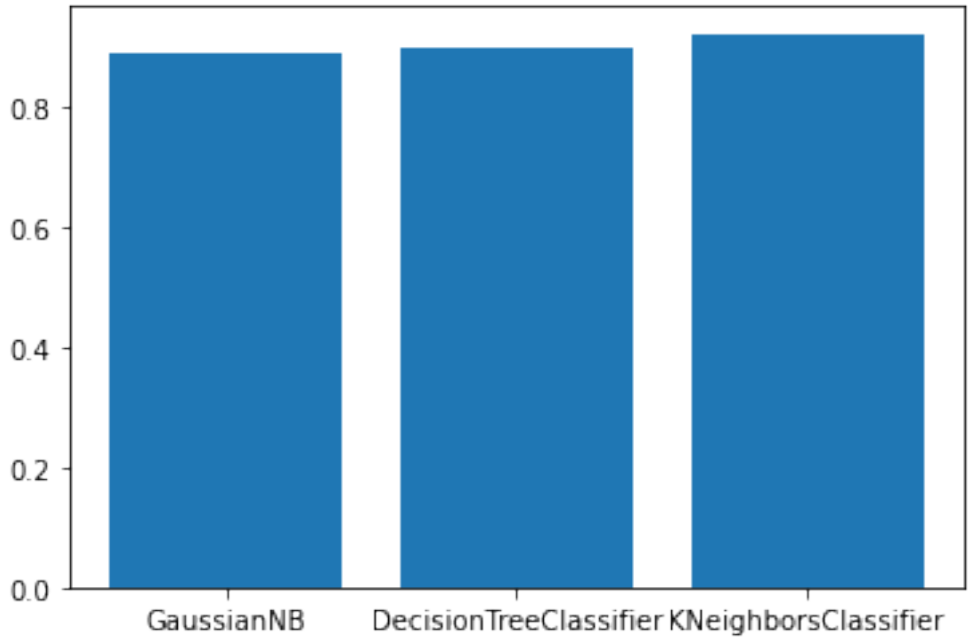
```
def ScindSplit(df,train_size):
    train_end = int(len(df)*train_size)

    df_train =df[:train_end]
    df_test = df[train_end:]
    return df_train,df_test
```

```
def prediction(labels, pred) :
    return accuracy_score(labels, pred)
```

*Pour faire la classification, on prend des données d'entraînement avec 55 348 individus et des données test avec 83 024 individus.



*Quelque soit l'algorithme de classification utilisé, nous avons des résultats de précision qui sont satisfaisants(plus de 89 pourcent). Bien que dans notre problème, l'algorithme KNeighborsClassifier a un meilleur score suivi de l'algorithme DecisionTreeClassifier, et GaussianNB. Voici les résultats obtenus



6.1 classification supervisée

6.2 classification non supervisée

L'apprentissage non supervisé permet d'analyser un grand volume de données,sans savoir à priori ce qui est voulu.Il couvre quatre tâches différentes,mais dans notre cas on a besoin que le clustering. Le but du clustering est de regrouper les données en pratiques dits clusters,pour mieux les étudier ou les comprendre.Dans notre cas nous allons utiliser la bibliothèque sciki-learn avec la méthode k-Means. Dans celle-ci,l'utilisateur choisit le nombre de clusters puis l'agorithme est le suivant :

1. Choisir les centroides (centres des clusters) aléatoirement ;
2.Associer chaque point au centroide duquel il est le plus proche ;
3.Calculer le barycentre de chaque cluster ainsi créé ;
4. Déplacer le centroide au niveau du barycentre ;
5. Reboucler sur 2 jusqu'à stabilisation.

Cet algorithme a cependant des défauts .

-Le nombre de clusters doit être précisé par l'utilisateur.Dans certains cas, il 'est difficile de l'estimer et il est alors nécessaire de faire des essais avec différentes valeurs.

-Les résultats sont souvent dépendants de la position de départ des 'centroides. Plusieurs démarrages avec des graines aléatoires sont donc généralement nécessaires. Une

comparaison des résultats en sortie permettra de trouver les clusters les plus courants ou ceux donnant les meilleurs résultats.

Dans les deux cas, il est nécessaire de pouvoir comparer plusieurs résultats pour décider du modèle à sélectionner. Une mesure existe pour cela : l'inertie. Pour chaque point du dataset, la distance au carré au centroïde le plus proche est calculée, l'inertie est alors la somme de ces distances. Plus l'inertie est faible et plus les points du dataset sont proches de leur centroide, ce qui signifie un meilleur résultat. Comparer plusieurs modèles revient à chercher celui qui présente l'inertie la plus faible, l'optimum théorique étant 0.Cependant on peut utiliser d'autres méthodes comme la méthode du coude et la méthode de Score de silhouette dans sklearn. **Pour la suite nous allons considérer que le nombre maximum de cluster à choisir est de 10, donc on cherche le nombre de groupe optimal parmi les 10.**

1. Méthode du coude :

Cette méthode est basée sur l'observation. L'augmentation du nombre de clusters peut aider à réduire la somme de la variance intra-cluster de chaque cluster. Avoir plus de clusters permet d'extraire des groupes plus fins d'objets de données qui sont plus similaires les uns aux autres. Pour choisir le « bon » nombre de clusters, le point de retournement de la courbe de la somme des variances intra-clusters par rapport au nombre de clusters est utilisé. Le premier tournant de la courbe suggère la bonne valeur de 'k'(pour tout 'k' > 0). Implémentons la méthode du coude en Python.

```
def coude(df,limit):

    wcss = {}
    for k in range(2,limit+1):
        model = KMeans(n_clusters=k)
        model.fit(df)
        wcss[k] = model.inertia_

    plt.plot(wcss.keys(), wcss.values(), 'gs-')
    plt.xlabel('Values of "k"')
    plt.ylabel('WCSS')
    plt.show()
```

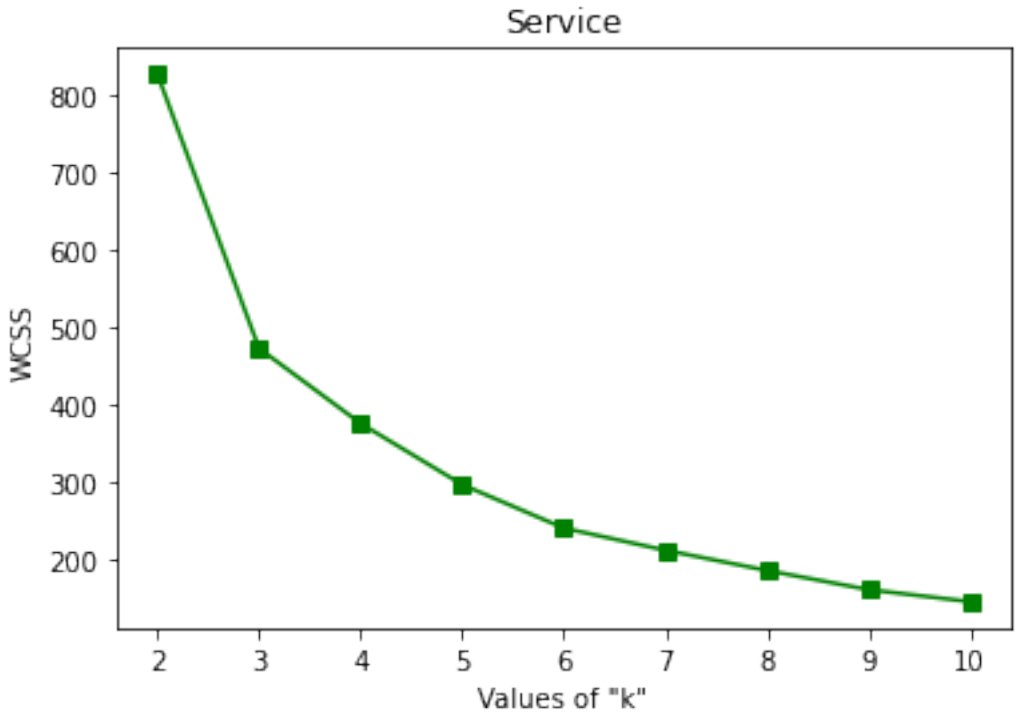
2. Score de silhouette :

Le score Silhouette est utilisé pour évaluer la qualité des clusters créés à l'aide d'al-

gorithmes de clustering tels que K-Means en termes de qualité du clustering des points de données avec d'autres points de données similaires les uns aux autres. Cette méthode peut être utilisée pour trouver la valeur optimale de « k ». Ce score est compris entre [-1,1]. La valeur de « k » ayant le score de silhouette plus proche de 1 peut être considérée comme le « bon » nombre de clusters. sklearn.metrics.silhouette score() est utilisé pour trouver le score en Python. Implémentons cela.

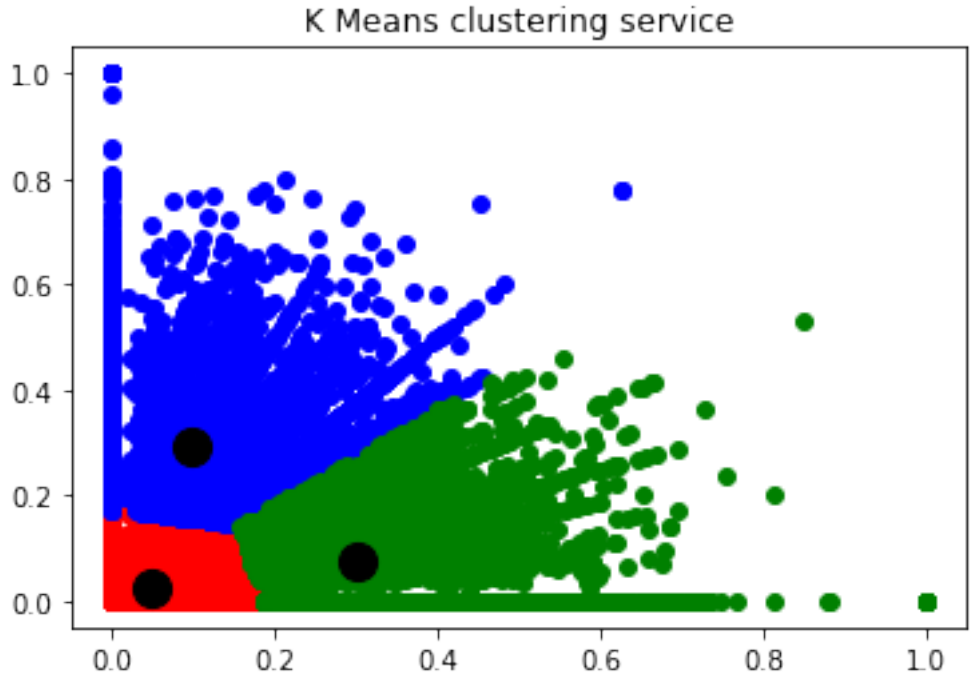
```
def silhouette(df,limit):
    dic={}
    for k in range(2, limit+1):
        model = KMeans(n_clusters=k)
        model.fit(df)
        pred = model.predict(df)
        score = silhouette_score(df, pred)
        dic[str(k)]=score
    return dic
```

Explication des résultats obtenus pour chaque thème dans les deux méthodes : *service



À travers le graphique ci-dessus, nous pouvons observer que le point de retournement de cette courbe est à la valeur de k = 3. Par conséquent, nous pouvons dire que le « bon

» nombre de clusters pour ces données est 3,le score de silhouette aussi nous donne la valeur 3.

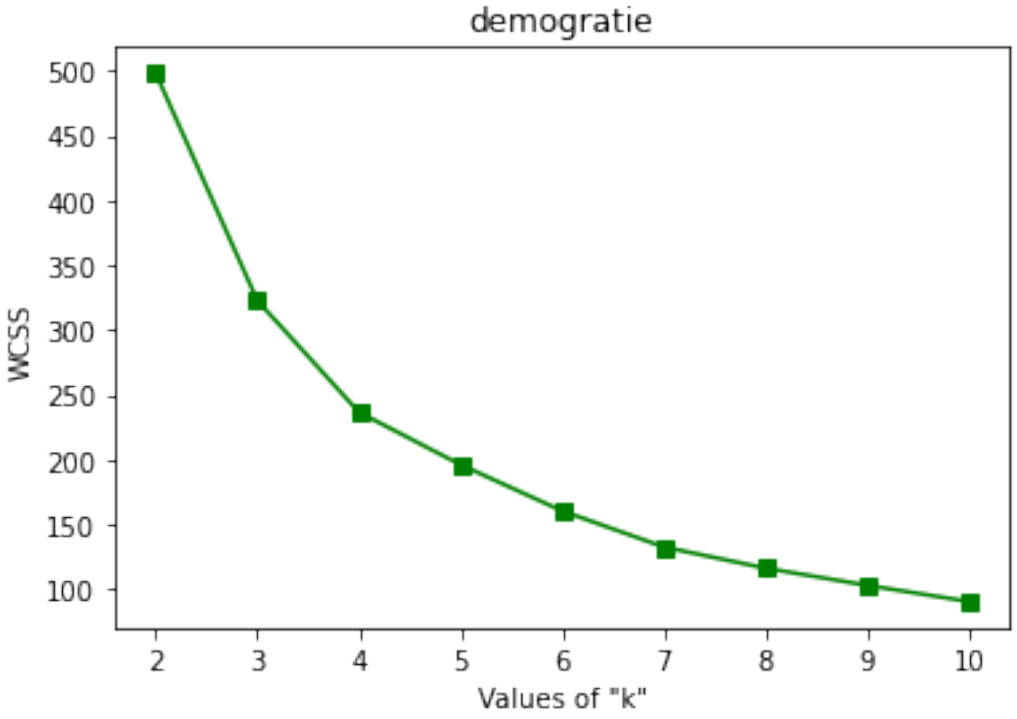


À partir du graphique ci-dessus, nous pouvons voir que 3 clusters efficaces ont été formés qui sont clairement séparables les uns des autres. Les centroïdes sont également visibles en noir.

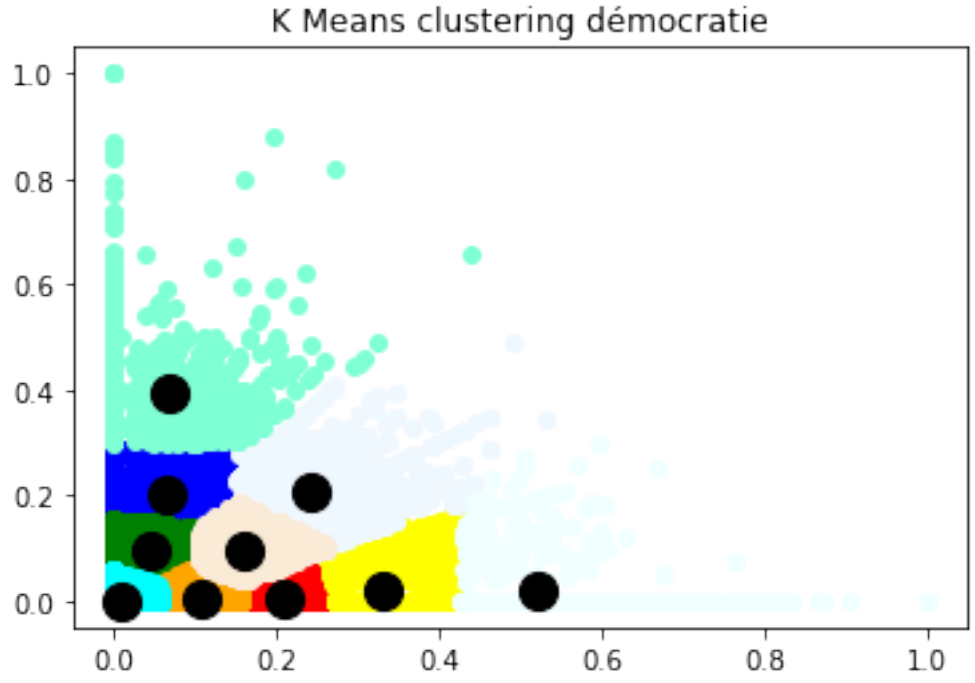
*En se basant sur nos résultats, on peut caractériser chaque groupe.

groupe 0	**groupe 2**	**groupe 1**
24992 textes	8779 textes	8453 textes
service	administration	public

***Démocratie :**



À travers le graphique ci-dessus, nous pouvons observer qu'il n'existe pas de point de retournement. Par conséquent, nous pouvons dire que le « bon » nombre de clusters pour ces données est 10(la valeur maximale),le score de silhouette aussi nous donne la valeur 10.



À partir du graphique ci-dessus, nous pouvons voir que les dix clusters efficaces qui ont été formés sont clairement séparables les uns des autres. Les centroïdes sont également

visibles en noir.

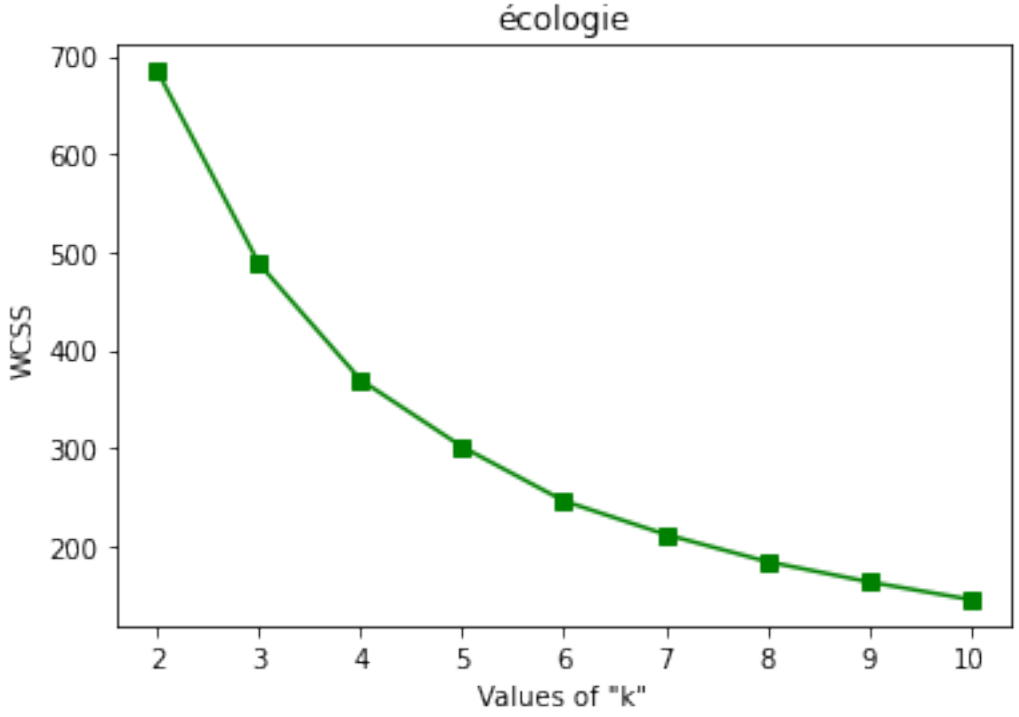
*En se basant sur nos résultats, on peut caractériser chaque groupe.

groupe 0	**groupe 5**	**groupe 6**
8530 textes	8052 textes	5788 textes
votes	social	public

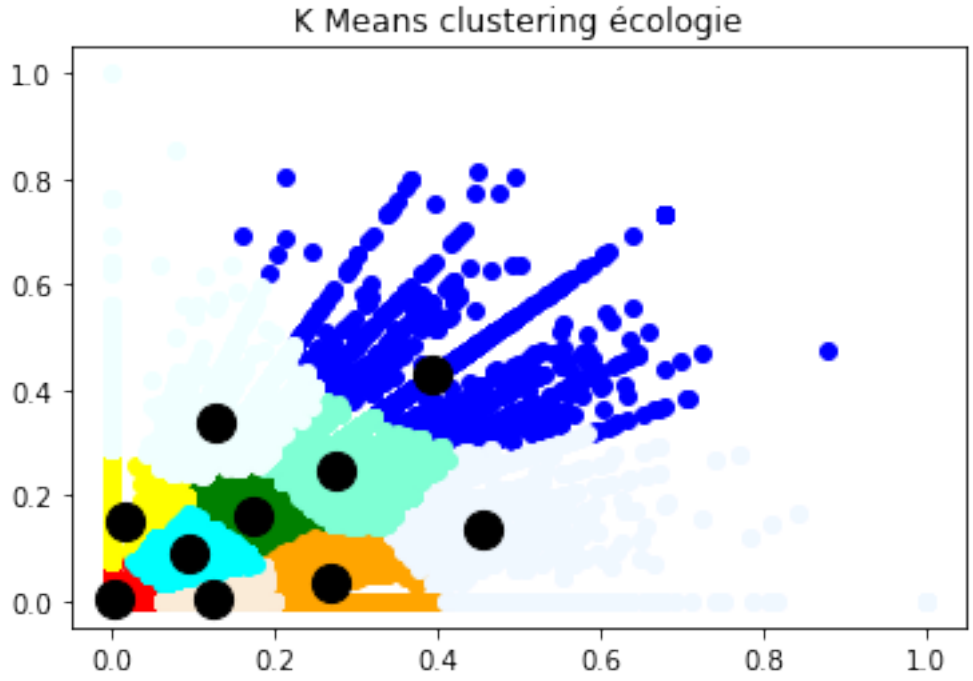
groupe 3	**groupe 8**	**groupe 1**
5257 textes	3900 textes	3344 textes
état	économie	conseil

groupe 9	**groupe 4**	**groupe 7**
3137 textes	1340 textes	877 textes
décisions	problèmes	actuel

***écologie :**



À travers le graphique ci-dessus, nous pouvons observer qu'il n'existe pas de point de retournement. Par conséquent, nous pouvons dire que le « bon » nombre de clusters pour ces données est 10(la valeur maximal),le score de silhouette aussi nous donne la valeur 10.

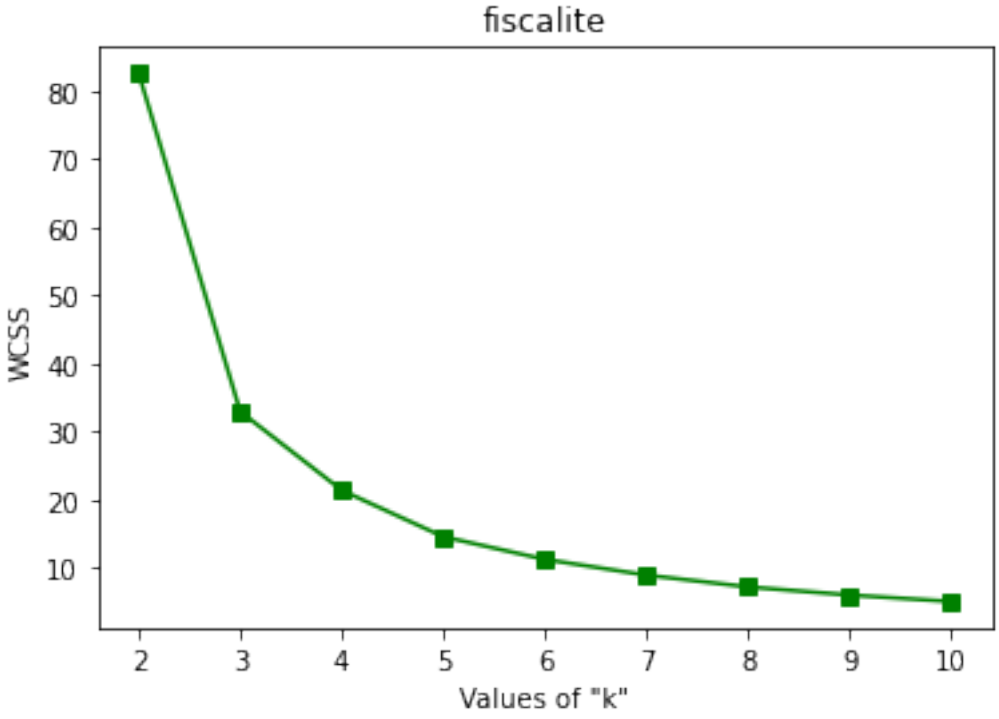


À partir du graphique ci-dessus, nous pouvons voir que 10 clusters efficaces ont été formés qui sont clairement séparables les uns des autres. Les centroïdes sont également visibles en noir.

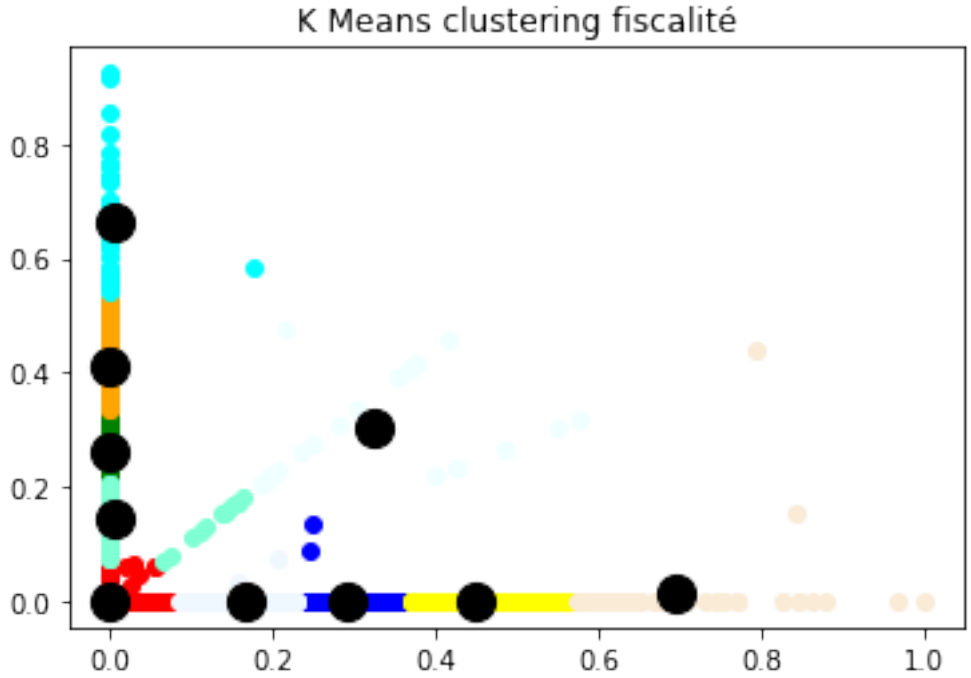
*En se basant sur nos résultats, on peut caractériser par exemple 3 groupes.

groupe 4	**groupe 6**	**groupe 3**
20542 textes	8213 textes	5842 textes
transport	commune	prendre

***fiscalité :**



À travers le graphique ci-dessus, nous pouvons observer qu'il n'existe pas de point de retournement. Par conséquent, nous pouvons dire que le « bon » nombre de clusters pour ces données est 10(la valeur maximal),le score de silhouette aussi nous donne la valeur 10.



À partir du graphique ci-dessus, nous pouvons voir que 10 clusters e ont été formés qui sont difficiles à séparer les uns des autres.Donc on peut essayer d'améliorer cela.

*En se basant sur nos résultats, on peut caractériser par exemple 3 groupes.

groupe 0	**groupe 4**	**groupe 1**
65464 textes	412 textes	397 textes
indemnisation,licenciement	impôt	

On constate qu'on pouvait améliorer nos résultats en prenant un 'k' limite plus élevé, car d'après les tests ,plus qu'on augmente 'k', plus l'inertie diminue. Mais on a choisi de rester sur le 'k' maximum de 10 pour nous simplifier le travail.

7 Problèmes rencontrés

*Le début du projet était compliqué du fait que je ne comprenais pas bien le sujet , d'abord le thème et aussi au final quelle conclusion on s'attendait à ce projet .

*Pour nettoyer les données, ce n'était pas évident surtout la lémmatisation , pour la langue française, python lemmatise que les verbes, et cela, était un vrai problème pour nous,c'est pourquoi on a opté la racinisation.

* Après avoir nettoyé les données et fait l'enregistrement ,c'était compliqué de réutiliser certaines colonnes qui contenaient des listes et après téléchargement tout redevient des chaînes de caractères, mais nous avons réussi à régler le problème grâce a la fonction eval sur chaque colonne.

*Pour avoir aussi les 'n' mots les plus pertinents dans un corpus, au début nous avons fait une fonction avec un boucle while et for . De ce fait, la fonction prenait des jours si on voulait avoir les 'n' mots les plus pertinents sur un grand corpus.

* À cause de ce dernier, j'étais obligé de prendre quelques textes, un corpus de 300 textes, mais une fois arrivé a la classification, j'avais des résultats de précisions de 12 pourcents. Au final, j'ai changé de fonction et pris l'ensemble des textes pour améliorer la précision.

Conclusion

Le but de ce projet était d'analyser automatiquement les contributions libres du grand débat 2019 afin de construire des catégories de contributions. Dans la dernière partie, on a trouvé des catégories de contributions pour chaque thème. Pour le thème écologie, on voit que la plupart des contributions tourne autour des sujets qui inclut des mots comme l'air, le transport, la commune, le train, etc... Dans le thème démocratie les contributions tournent autour des sujets comme Vote, social, public, état, économie, etc. Dans le thème, fiscalité, les contributions tournent autour des sujets comme indemnisation, licenciement, impôt, économie, chômage, etc. Dans le thème, service, les contributions tournent autour des sujets comme administration, emploi, vitesse ,fonctionnaire. Pour avoir une conclusion simple avec ce que nous avons pu faire, par exemple dans le thème service, nous pouvons dire que la plupart de la population, attend ou propose des solutions qui tournent autour de l'indemnisation des travailleurs, licenciement des travailleurs. Et donc l'état doit forcément avoir une préoccupation importante dans ces sujets. Je trouve que c'était plus intéressant de travailler avec les questions comme l'a fait les scientifiques qui ont travaillé sur ce sujet, afin d'avoir des solutions exactes dans chaque question posée. Mais vu le timing et nos connaissances un peu limitées dans ce domaine, on ne pouvait pas faire plus.

Pour conclure,ce projet a été un tremplin vers de nouvelles connaissances . Ce projet fut très chronophage, le temps d'adaptation était un peut long. En effet,nous manquions d'expériences dans les bibliothèques de traitement de texte , ce qui a été un frein mais aussi un moteur nous poussant à nous dépasser. Nous sommes désormais beaucoup plus à l'aise en python et en traitement de texte. Nous sommes plus que satisfaits de cette montée en compétences et espérons nous perfectionner encore d'avantage dans l'avenir.