



Projet étudiant : Classification automatique d'emails

Auteurs : Oscar BENOIT et Baptiste GUICHARD

Tuteur : Babiga BIRREGAH

A22

# Sommaire

Introduction .....	3
1. Mise en contexte .....	4
2. Compréhension du sujet .....	5
3. Collecte et préparation des données .....	7
a. Collecte des données	
b. La préparation des données	
4. Classification non-supervisée .....	16
a. Premier modèle : Kmeans et LDA	
b. Nouveau modèle : BERTopic	
5. Classification supervisée .....	30
a. Etiquetage des données	
b. Implémentation des 3 modèles supervisés	
6. Evaluation des modèles .....	44
7. Evolutions potentielles du projet .....	48
a. La confidentialité des données utilisateurs	
b. Mock-up de l'outil	
Conclusion .....	51
Retours d'expérience .....	52
Annexe .....	53

## Introduction

Au cours de notre cursus à l'UTT, nous avons pu tous les deux nous familiariser au Data Processing et plus particulièrement aux techniques de Machine Learning, grâce aux cours que nous avons suivi et aux stages que nous avons faits. Nous nous sommes alors trouvé une passion en l'Intelligence Artificielle et les possibilités quasiment infinies de projets utilisant ces technologies. Ayant tous les deux envi de poursuivre notre carrière dans ce domaine vaste, nous voulions découvrir une partie du Machine Learning encore inconnue à nos yeux, le Natural Language Processing (NLP), ou traitement du langage naturel en français, et plus particulièrement le traitement de texte. N'ayant pas eu l'occasion d'étudier ce domaine à l'UTT ou en stage, nous avons alors décidé de mener un Projet Etudiant conjoint, afin d'apprendre en faisant.

De plus, nous sommes également tous les deux intéressés par l'entrepreneuriat, c'est pourquoi nous avons imaginé notre projet au travers d'un outil qui pourrait être commercialisé, pas forcément dans le but de réellement créer cette entreprise, mais plus pour ajouter un objectif à notre projet qui doit donc répondre à des besoins. Ainsi, nous avons pensé à la création d'un outil qui permettrait de classer et gérer automatiquement ses emails. Nous avons alors défini le périmètre de notre projet grâce à l'aide de M.Birregah afin de nous assurer que nous ne nous aventurons pas dans quelque chose d'infaisable, puis nous avons pu commencer à imaginer notre outil et surtout tester la classification d'emails. Nous remercions par ailleurs M.Birregah qui a été le tuteur de ce projet et qui a su nous guider tout au long de son déroulement lorsque nous en avons besoin.

Dans ce rapport, nous allons tout d'abord remettre en contexte la problématique du projet, puis nous verrons comment nous avons défini et compris le sujet suivi de la collecte et la préparation de nos données d'entraînement. Nous verrons expliquerons ensuite l'implémentation d'une classification non-supervisée et d'une classification supervisée suivi de leur évaluation. Enfin, nous aborderons les évolutions potentielles du projet suivies d'une conclusion sur le projet et de nos retours d'expérience.

# 1. Mise en contexte

La classification automatique de mails est une tâche courante dans de nombreux environnements de travail, où il est souvent nécessaire de trier rapidement et efficacement de nombreux courriels entrants. Cette tâche peut être complexe, car il existe de nombreuses catégories possibles de mails, allant des messages personnels aux courriels professionnels en passant par les spams et les newsletters. Pour réaliser cette tâche de manière automatisée, nous pouvons utiliser des algorithmes de Machine Learning, qui sont capables d'apprendre à prédire la catégorie d'un mail en analysant de nombreux exemples de mails. Il existe deux principaux types d'algorithmes de Machine Learning pour la classification automatique de mails : les algorithmes supervisés et les algorithmes non-supervisés.

Les algorithmes supervisés sont basés sur l'apprentissage par exemples, c'est-à-dire qu'ils sont entraînés sur un ensemble de données étiquetées, qui comprennent des exemples de mails et leurs catégories respectives. L'algorithme utilise ces exemples pour apprendre à prédire la catégorie de nouveaux mails. Les algorithmes supervisés sont souvent utilisés lorsque nous avons un grand nombre d'exemples étiquetés et que nous voulons prédire une variable cible précise, comme la catégorie d'un mail.

Les algorithmes non-supervisés, en revanche, ne sont pas entraînés sur des données étiquetées. Au lieu de cela, ils utilisent des techniques de clustering pour regrouper des mails en groupes en fonction de leurs similitudes. Les algorithmes non-supervisés sont utiles lorsque nous avons peu ou pas d'exemples étiquetés et que nous voulons découvrir des structures ou des groupes cachés dans nos données.

Lors de ce projet, nous avons décidé de nous intéresser et implémenter ces deux types d'algorithmes et de les comparer.

## 2. Compréhension du sujet

Avant de commencer à travailler sur le projet, il nous fallait être sûr de bien comprendre le sujet et la direction dans laquelle nous voulions aller.

Ainsi, nous avons défini notre objectif, qui est de pouvoir créer un algorithme capable de classer tous les messages d'une boîte mail. Malheureusement, lors d'un échange avec M.Birregah, nous nous sommes rendus compte que cet objectif était irréalisable au vu de nos ressources, nos connaissances et du temps dédié au projet.

En effet, les possibilités de catégories sont infinies en fonction du profil de l'utilisateur et de son environnement (profession/études, centres d'intérêts, utilisations de la boîte mail etc...), il est donc quasiment impossible de créer un modèle capable de classer tous les mails de tous les utilisateurs possibles à coup sûr.

Afin que notre projet soit réalisable, nous avons décidé de nous cantonner à un type d'utilisateur précis que nous devons définir. Le fait de définir le profil de cet individu nous permettrait d'obtenir plus ou moins les mêmes types d'e-mails à classer, ce qui faciliterait la tâche au modèle.

Nous avons alors considéré l'individu suivant :

- Etudiant en études supérieures
- Adeptes du e-commerce
- Autonome sur ses prises de rdv
- Ayant déjà recherché un stage/emploi
- Utilisateur de réseaux sociaux
- S'occupant de tâches administratives
- Discutant avec ses proches par e-mail

Ce profil d'individu correspond à notre profil à tous les deux ainsi que celui de nos amis proches. Nous avons choisi ce profil tout simplement par rapport à la facilité d'accès aux données e-mails.

Une fois que nous avons défini l'individu que nous considérons, nous devons imaginer les catégories d'e-mails que nous voulions classer. Cette réflexion nous permettrait de mieux comprendre où nous voulons aller. Nous avons alors identifié 12 catégories distinctes :

- Université
  - Messages concernant les cours
  - Messages concernant les examens
  - Messages concernant les événements au sein de l'université/école
  - Informations administratives de l'université/école

- Publicités
  - Publicités des marques/sites auxquelles l'utilisateur s'est abonné
  - Newsletters
  - Publicités extérieures
  - Alertes auxquels l'utilisateur a souscrit
- Commandes
  - Commandes sur les sites d'e-commerce
  - Suivi de commandes
- Economie
  - Factures d'achats en ligne
  - Quittances possibles de loyers
  - Factures liées au logement (assurances, eau, électricité, gaz)
  - Informations bancaires (virements, soldes, épargne etc...)
- RDVs
  - Confirmations de rdvs pris en ligne
  - Programmation de réunions (Zoom, Teams, appels etc...)
- Réseaux Sociaux
  - Mails provenant de réseaux sociaux auxquels l'utilisateur s'est inscrit
- Administratifs
  - Messages provenant des assurances
  - Messages de démarches administratives (visas, bourses...)
- Tickets
  - Tickets achetés en ligne (transports, concerts, cinema etc...)
- Jobs
  - Message de recherche d'emploi/stage (candidatures, suivis etc...)
- Perso
  - Messages informels avec des proches de l'utilisateur
- Comptes et mdp
  - Messages concernant la création de comptes
  - Messages concernant un changement de mdp
- Autres
  - Messages qui n'appartiendraient à aucune de ces catégories

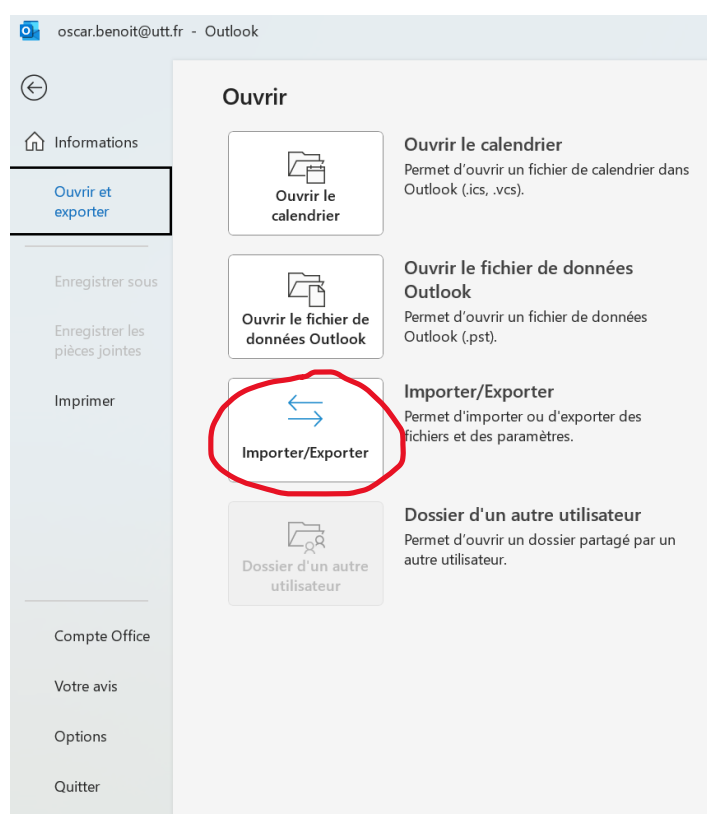
### 3. Collecte et préparation des données

Une fois que nous avons défini l'individu que nous considérons ainsi que les catégories dans lesquelles nous voulons classer les emails, nous pouvons commencer à collecter les données qui nous ont servi à entraîner notre modèle.


#### a. La collecte des données

Dans un premier temps, nous savions que nous pouvons accéder à nos propres emails respectifs, ce qui a été notre base de travail. En effet, il était important de tester au préalable sur nos données afin d'être certain que nos catégories soient réalistes. De plus, cela facilitait la tâche à notre algorithme, car nous sommes tous deux étudiants à l'Université de Technologie de Troyes, nous ne considérons donc qu'un seul environnement universitaire.

Maintenant, il faut savoir comment récupérer nos emails. Pour ce faire, nous avons trouvé une solution simple et efficace. Si l'on connecte notre boîte mail sur Microsoft Outlook, l'outil nous permet de télécharger directement un fichier CSV de tous les messages.



# Assistant Importation et exportation



Sélectionnez une action à exécuter :


- Exporter des données vers un fichier
- Exporter des flux RSS vers un fichier OPML
- Importer à partir d'un autre programme ou fichier
- Importer des flux RSS à partir d'un fichier OPML
- Importer des flux RSS à partir de la liste globale des flux
- Importer un fichier iCalendar (.ics) ou vCalendar (.vcs)
- Importer un fichier VCARD (.vcf)

Description

Exporter des informations Outlook vers un fichier pour les utiliser dans d'autres programmes.

< Précédent
 Suivant >
 Annuler

## Exporter vers un fichier




Créer un fichier de type :

- Fichier de données Outlook (.pst)
- Valeurs séparées par une virgule

< Précédent
 Suivant >
 Annuler

## Exporter vers un fichier




Sélectionner le dossier à exporter de :

- Boîte de réception
- Boîte d'envoi
- Notes
- Problèmes de synchronisation (uniquement cet ordinateur)
  - Défaillances locales (uniquement cet ordinateur)
- salome.sevin@edu.em-lyon.com
  - Archive
  - Boîte de réception
  - Dossier sans nom
  - Boîte d'envoi
  - Brouillons
  - Calendrier
    - Anniversaires
    - Jours fériés - France
  - Contacts

< Précédent
 Suivant >
 Annuler

## Exporter vers un fichier



Enregistrer le fichier exporté sous :

C:\Users\oscar\projects\SI05\PE\_classif\_mail\data\mails.CSV

Parcourir ...

< Précédent
 Suivant >
 Annuler



Après élimination des features inutiles, il ne nous en reste plus que 3, que nous utiliserons pour la suite du projet :

- Objet : Objet (Titre) du courriel
- Corps : Contenu principal du message
- Adresse : Adresse électronique de l'émetteur

Nous avons ensuite commencé la préparation de données en Python à l'aide d'un Jupyter notebook.

## b. La préparation des données

La préparation des données est une étape cruciale dans le domaine du traitement du langage naturel (NLP). Elle consiste à préparer et à transformer les données brutes en un format adapté à l'analyse par les algorithmes de Machine Learning. En effet, le texte non traité est souvent difficile à utiliser directement dans ces algorithmes, qui ont besoin de données structurées et normalisées pour fonctionner de manière efficace. La préparation des données en NLP comprend donc plusieurs étapes de nettoyage, de tokenization, de normalisation et de vectorisation, qui permettent de transformer le texte en une forme utilisable par les algorithmes.

Cette étape est particulièrement importante en NLP, car le langage naturel est riche et complexe, et il est difficile de l'analyser de manière automatisée sans une préparation préalable des données. Par exemple, il peut être nécessaire de supprimer les caractères indésirables, comme les caractères de contrôle ou les caractères spéciaux, ou de normaliser les tokens, c'est-à-dire de les mettre tous sur le même pied d'égalité, par exemple en les mettant tous en minuscules ou en lemmatisant (c'est-à-dire en ramenant chaque mot à sa forme de base). La préparation des données en NLP permet donc de rendre le texte plus facile à analyser et à traiter par les algorithmes de Machine Learning.

Nous allons donc vous indiquer ci-après quel a été notre cheminement lors de notre préparation de données :

### i. Fusion des CSVs

Une fois les csv récupérés via Outlook, que ce soit de nos boîtes UTT ou de nos boîtes personnelles, nous devons les récupérer dans notre code et les fusionner.

Pour ce faire, nous décidons d'utiliser la librairie pandas, qui est une librairie Python populaire pour l'analyse de données. Elle fournit des structures de données et des outils pour manipuler et traiter des données de manière efficace.

La principale structure de données de Pandas est le DataFrame, qui permet de stocker des données structurées sous forme de tableau à deux dimensions. Chaque

colonne du DataFrame peut avoir un type de données différent, comme des entiers, des flottants ou des chaînes de caractères. Le DataFrame est particulièrement utile pour manipuler et traiter des données enregistrées dans des fichiers tabulaires, comme des fichiers CSV ou Excel.

Pandas propose également de nombreuses fonctions pour manipuler et traiter les données, c'est pourquoi nous utiliserons des DataFrames pour faire toute notre chaîne de valorisation. Pour fusionner nos CSV nous définissons une fonction permettant de réaliser cela :

```
def fusion_csv(liste_csv):
    dataset = pd.read_csv(liste_csv[0])
    for ds in liste_csv[1:]:
        dataset = pd.concat([dataset, pd.read_csv(ds)], ignore_index=True)

    return dataset
```

## ii. Traitement des colonnes

Ensuite, on a réalisé un traitement sur les colonnes afin de ne garder que les colonnes qui nous intéressaient. Nous avons également renommé ces colonnes, voici la fonction que nous avons défini pour réaliser cela :

```
def columns_treatment(df):
    df = df.iloc[:, [0, 1, 3]]
    df = df.rename(columns={'Objet': 'objet', 'Corps': 'corps', 'De: (adresse)': 'adresse'})
    return df
```

## iii. Traitement des liens dans les emails

Par la suite, en explorant nos emails, nous nous sommes rendu compte qu'il y avait de nombreux liens https qui n'étaient que des parasites dans nos traitements et qui généreraient du bruit pour nos algorithmes par la suite, nous avons donc décidé de les supprimer. Cette fonction nous a aidé à réaliser cela :

```
def remove_https(corps):

    text_file = open(r'corps.txt', 'w', encoding="utf-8")
    text_file.write(corps)
    text_file.close()

    final_text_file = open(r'final_corps.txt', 'w', encoding="utf-8")
    reading_text_file = open(r'corps.txt', 'r', encoding="utf-8")
    for line in reading_text_file:
```

```
if "http" not in line:
    final_text_file.write(line)

final_text_file.close()
reading_text_file.close()

with open('final_corps.txt', 'r', encoding="utf-8") as file:
    return file.read()
```

#### iv. Suppression des caractères spéciaux

Par la suite, le but était de supprimer tous les caractères spéciaux, inutiles pour nos algorithmes de NLP. Pour ce faire nous avons utilisé des expressions régulières afin de réaliser ce travail et supprimer tous les caractères spéciaux, mais également les chiffres, et les retours chariots ect... Tous ces éléments étant parasites dans notre projet de classification de mails automatiques.

#### v. Traduction des emails

Ensuite, après plusieurs itérations de cleaning, nous nous sommes rendu compte que les différents processus de lemmatization, de feature extraction, de stop words ect... étaient bien plus précis et performant sur des textes en anglais. De ce fait, nous avons décidé de traduire tous nos emails en anglais.

Afin de réaliser cette tâche, nous avons utilisé Le modèle Helsinki-NLP/opus-mt-fr-en de Hugging Face, qui est un modèle de traduction automatique du français vers l'anglais, basé sur l'approche de Machine Translation (MT) utilisée par l'outil de traduction en ligne OPUS. Il a été entraîné et publié par l'équipe de recherche en traitement du langage naturel de l'université de Helsinki.

Ce modèle a été entraîné sur un grand corpus de textes en anglais et en français, ce qui lui permet de produire des traductions de haute qualité pour de nombreux types de textes. Il utilise une approche de traduction automatique basée sur le transfert de style, ce qui signifie qu'il essaie de reproduire le style et la structure du texte source dans la traduction finale.

Le modèle Helsinki-NLP/opus-mt-fr-en est disponible sur la plateforme de traitement du langage naturel Hugging Face et peut être utilisé avec leur bibliothèque de traitement du langage naturel en Python pour la traduction de textes en anglais vers le français.

Il est important de noter que, comme tous les modèles de traduction automatique, le modèle Helsinki-NLP/opus-mt-fr-en peut parfois produire des traductions qui sont imprécises ou qui ne capturent pas entièrement le sens du texte source.

La difficulté à surmonter était le nombre de caractères maximum que le modèle prenait en compte. En effet, Le modèle Helsinki-NLP/opus-mt-fr-en a été conçu pour traduire des textes de longueur modérée, généralement compris entre quelques phrases

et plusieurs paragraphes. Il est capable de traiter des textes d'une longueur allant jusqu'à environ 2048 caractères, mais il peut être plus efficace sur des textes plus courts.

Il est important de noter que la qualité de la traduction peut diminuer lorsque le texte à traduire est très long, car le modèle doit traiter de grandes quantités de données et peut avoir du mal à conserver le sens et la structure du texte original. De plus, traduire de longs textes peut prendre plus de temps et nécessiter plus de ressources informatiques.

Nous avons donc décidé de découper nos corps de mails en segments de 50 mots afin d'avoir des textes de longueur assez réduites mais comprenant tout de même un certain contexte, important pour la traduction.

## vi. Supprimer les stop words

Les stop words sont des mots courants qui sont fréquemment utilisés dans une langue mais qui ont peu de signification sémantique en eux-mêmes. Ils sont généralement ignorés lors de l'analyse de textes en informatique, car ils ne contribuent pas de manière significative au sens des phrases dans lesquelles ils apparaissent.

Voici quelques exemples de stop words en anglais :

- the
- a
- an
- and
- or
- but
- for
- nor
- so
- yet

La liste des stop words en anglais peut varier selon le contexte et l'application. Certains outils de traitement du langage naturel peuvent inclure des mots tels que "I" et "you" dans leur liste de stop words, tandis que d'autres peuvent les considérer comme étant trop importants pour être ignorés.

Il est important de noter que les stop words sont généralement utilisées lors de l'analyse de textes pour éliminer les mots qui n'ont pas de signification sémantique importante, ce qui peut aider à améliorer l'efficacité et la précision des outils de traitement du langage naturel.

Afin de réaliser cette tâche, nous avons utilisé la bibliothèque Python Natural Language Toolkit (nltk) qui est une ressource très utile pour le traitement du langage naturel et inclut un corpus de stop words pour plusieurs langues. Le module stopwords de cette

bibliothèque permet de charger et d'utiliser ces listes de stop words pour éliminer ces mots d'un texte dans le cadre d'un projet de traitement du langage naturel.

Voici la fonction que nous avons créé :

```
def stop_words_english(text):
    stop_words = stopwords.words('english')
    text = [word for word in text.split() if ((word not in stop_words) and (len(word)>1))]

    return text
```

## vii. La lemmatization

La lemmatization est un processus de traitement du langage naturel qui consiste à regrouper les formes flexionnées d'un mot (par exemple, les verbes au passé ou les noms au pluriel) sous leur forme de base, appelée "lemme". Cette opération peut être utile pour normaliser les textes et simplifier l'analyse de données en regroupant les mots qui ont le même sens sous une forme unique.

Par exemple, le lemme de "marche" (verbe) est "marcher", et le lemme de "chats" (nom pluriel) est "chat". En utilisant la lemmatization, on peut convertir ces formes flexionnées en leur forme de base et ainsi regrouper tous les mots qui ont le même sens sous une forme unique.

La lemmatization peut être utile pour diverses tâches de traitement du langage naturel, et notamment pour la classification de texte qui est notre objectif dans ce projet de classification automatique de mails. Il existe plusieurs outils et bibliothèques qui peuvent être utilisés pour effectuer la lemmatization, nous avons choisi d'utiliser la bibliothèque Python spaCy, qui est une bibliothèque de traitement du langage naturel en Python qui offre de nombreuses fonctionnalités pour le traitement de données textuelles, y compris la lemmatization.

Elle est considérée comme étant l'une des bibliothèques de traitement du langage naturel les plus populaires et les plus performantes disponibles en Python.

Pour utiliser la lemmatization dans spaCy, il suffit de charger un modèle pré-entraîné et de traiter un texte en utilisant les fonctionnalités de tokenization. Ensuite, nous pouvons utiliser la propriété lemma\_ des tokens (mots) du texte pour accéder à leur lemme.

Voici le code qui réalise cela :

```
def lemmatization(nlp, texte):
    i = 0
    # On regarde chaque mot dans le texte
    # Chaque mot a le numéro i
    for mot in texte:
```

```
# on va lemmatizer
doc = nlp(mot)
for token in doc:
    texte[i] = token.lemma_.lower()
    i += 1
return texte
```

La lemmatization termin  , nous avons fini la pr  paration de nos donn  es de corps et d'objet du mail.

Nous avons   galement r  alis   un traitement sur les adresses emails, afin de les s  parer en trois parties. Voici le pattern : [nom avant le '@' , nom entre le '@' et le '.' , domaine apr  s le '.']

Voici la fonction qui nous permet de r  aliser cela :

```
def clean_address(text):
    index = len(text)
    text = text.replace('@', ' ')
    text = text[:index-4] + text[index-4:].replace(".", ' ')
    text = text.split(' ')
    return text
```

Enfin, nous avons cr    une fonction 'data\_cleaning' qui nous permet de traiter tous les mails, en appelant les diff  rentes fonctions que nous avons r  alis  es :

```
def data_cleaning(df, nb):

    df = columns_treatment(df)

    nlp_en = spacy.load('en_core_web_md')

    for i in df.index:

        corps = remove_https(str(df['corps'][i]))

        corps = text_cleaning(corps)

        corps = text_translation(corps)

        objet = text_cleaning(df['objet'][i])

        objet = text_translation(objet)

        #stop words cleaning for object
        objet = stop_words_english(objet)
        objet_en = lemmatization(nlp_en, objet)
```

```
#stop words cleaning for corps
corps = stop_words_english(corps)
corps = lemmatization(nlp_en, corps)

df['objet'][i] = objet_en
df['corps'][i] = corps

df['adresse'][i] = clean_address(df['adresse'][i])
print("dernier mail traité est le numéro : ", nb)
nb = nb+1
if nb%500 == 0:
    df.to_csv('df_'+str(nb)+'.CSV')
return df
```

Cette fonction permet donc de réaliser toutes les étapes de data cleaning que ce soit sur l'objet, le corps ou bien l'adresse. Afin de suivre le traitement, nous avons également un index 'nb' qui permet de suivre en direct avec un print console à quel mail nous en sommes dans le traitement. En effet, ce traitement est très chronophage dû aux différents réseaux de neurones appelés qui mettent du temps à performer : notamment la lemmatisation et la traduction. De ce fait, afin d'assurer un back-up, nous avons enregistré dans un CSV les emails traités tous les 500 mails (à l'aide de la condition if à la fin de la fonction, avec le modulo).

Nous avons donc lancé cet algorithme sur 10 000 mails, et cela nous a pris 10h afin d'obtenir les 10 000 mails nettoyés.

Une fois le nettoyage et la préparation des données terminés, nous pouvons passer à la deuxième étape de notre projet de classification de mails automatiques : l'application d'algorithmes non-supervisés. Il faut noter que pour nos différents algorithmes, nous n'utiliserons finalement que le corps du mail, afin de ne classifier qu'à partir du contenu du message.

## 4. Classification non-supervisée

L'apprentissage non supervisé est une forme d'apprentissage automatique dans laquelle un modèle de machine learning est entraîné sur des données qui ne sont pas étiquetées ou annotées. Le modèle doit donc trouver des structures et des patterns dans les données de manière autonome. L'objectif de l'apprentissage non supervisé est généralement de comprendre les structures cachées dans les données et de découvrir des groupes ou des clusters similaires dans les données. En général, l'apprentissage non supervisé est utilisé lorsqu'il y a peu de données étiquetées disponibles ou lorsqu'il est difficile de définir clairement les étiquettes ou les objectifs de l'apprentissage.

### a. Premiers modèles : Kmeans et LDA

Dans notre projet, nous avons dans un premier temps eu l'idée de réaliser un KMeans car c'est un algorithme avec lequel nous sommes à l'aise, puisque l'on avait déjà utilisé des algorithmes de ce type dans certains de nos projets. De ce fait, la première chose à réaliser était la feature extraction.

#### i. Feature extraction

En traitement du langage naturel, la feature extraction consiste à extraire des caractéristiques ou des attributs pertinents d'un texte qui seront utilisés comme entrées pour un modèle de Machine Learning. Ces caractéristiques peuvent être utilisées pour effectuer diverses tâches en NLP, dont la classification de documents, ce qui correspond exactement à notre projet.

Il existe plusieurs techniques d'extraction de caractéristiques en NLP, notamment l'utilisation de représentations de mots pré-entraînées, l'utilisation de n-grammes et l'utilisation de vecteurs de fréquence de termes (TF-IDF).

Les représentations de mots pré-entraînées sont des modèles pré-entraînés qui ont été appris sur de grandes quantités de données de texte et qui peuvent être utilisés pour extraire des caractéristiques de nouveaux textes. Ils sont souvent utilisés pour initialiser des modèles de NLP et peuvent être très utiles lorsque les données de texte sont limitées. Nous avons choisi d'utiliser le TF IDF.

Le TF-IDF (Term Frequency - Inverse Document Frequency) est une mesure statistique utilisée pour évaluer l'importance d'un terme dans un document par rapport à un ensemble de documents. Il est souvent utilisé en traitement du langage naturel pour extraire des caractéristiques de textes et peut être utile pour la classification de documents ou la reconnaissance de sentiments.

Le TF-IDF d'un terme  $t$  dans un document  $d$  est défini comme le produit de sa fréquence dans le document (TF) et de sa fréquence inverse dans l'ensemble de documents (IDF).



**TF(t)** = (nombre de fois où le terme t apparaît dans le document d) / (nombre total de termes dans le document d)

**IDF(t)** =  $\log((\text{nombre total de documents}) / (\text{nombre de documents contenant le terme t}))$

Le TF-IDF d'un terme t dans un document d est donc égal à :

$$\text{TF-IDF}(t, d) = \text{TF}(t) * \text{IDF}(t)$$

De ce fait, le TF détermine la fréquence relative d'un mot ou d'une combinaison de mots dans un document. Cette fréquence sera comparée à la survenance de tous les autres mots restants du texte, et le IDF calcule la fréquence inverse du document qui complète l'analyse de l'évaluation du mot et vient donc corriger le TF.

Le TF-IDF d'un terme mesure à la fois sa fréquence dans le document et sa rareté dans l'ensemble de documents. Un terme qui apparaît souvent dans un document mais rarement dans l'ensemble de documents aura un TF-IDF élevé, indiquant qu'il est important pour le document en question.

Le TF-IDF est souvent utilisé comme un vecteur de caractéristiques pour les modèles de Machine Learning en NLP. Il est calculé pour chaque terme dans chaque document et utilisé comme entrée pour un modèle comme notre KMeans.

Le TF-IDF présente certains avantages notoires pour notre projet :

- ✓ Il permet de mettre en valeur les termes les plus importants dans un document par rapport à un ensemble de documents => Idéal pour la classification
- ✓ Il peut être utilisé pour extraire des caractéristiques de textes qui seront utilisées comme entrées pour un modèle => Cela sera en entrée de notre K-Means
- ✓ Il est insensible aux variations de longueur de document, ce qui le rend adapté à l'analyse de textes de longueur variable => Nos emails ont des longueurs très variables, donc cet aspect est un grand avantage pour notre projet
- ✓ Il est facile à calculer et à utiliser comme vecteur de caractéristiques pour les modèles de machine learning

Pour réaliser cela en python, nous avons importé la librairie TfidfVectorizer qui est un outil de la bibliothèque scikit-learn, pour transformer nos documents textuels (nos emails) en vecteurs de caractéristiques.

## ii. PCA

La principal component analysis (PCA) est une technique de traitement des données utilisée pour réduire la dimensionnalité d'un ensemble de données. Elle permet de trouver les directions dans lesquelles les données varient le plus, et de les représenter sous forme de composantes principales. Ces composantes sont des combinaisons linéaires des variables d'origine et sont triées en fonction de leur importance en termes de variation des données.

La PCA est souvent utilisée en visualisation de données, en apprentissage automatique et en analyse statistique. La PCA a pour objectif de réduire la complexité des données tout en conservant la variabilité la plus importante, ce qui peut faciliter la compréhension et l'analyse des données. La PCA peut également être utilisée pour sélectionner les caractéristiques les plus importantes dans un jeu de données, ce qui peut améliorer les performances de certains modèles d'apprentissage automatique. Notre cas rejoint ces deux aspects : l'objectif de cette PCA est d'améliorer les performances de notre KMeans tout en lui donnant en entrée un jeu de données plus léger.

Nous avons réalisé cette PCA en python en utilisant le module `sklearn.decomposition.PCA` de scikit-learn : ce module propose une implémentation de la PCA qui peut être facilement intégrée.

Voici le code qui montre comment utiliser cette implémentation pour réduire notre ensemble de données :

```
from sklearn.decomposition import PCA

pca = PCA(n_components=0.95, random_state=42)
X_reduced= pca.fit_transform(X.toarray())
```

Le paramètre `n_components` spécifie le nombre de composantes à conserver lors de la réduction de dimension. Si `n_components` est un entier, cela signifie que le nombre de composantes à conserver est égal à cet entier. Si `n_components` est un nombre compris entre 0 et 1, cela signifie que le nombre de composantes à conserver est sélectionné de manière à conserver une fraction donnée de la variance totale des données.

Par exemple, si `n_components=0.95`, comme dans notre cas, cela signifie que l'on souhaite conserver 95% de la variance totale des données en utilisant un nombre de composantes aussi petit que possible. Utiliser `n_components=0.95` permet généralement de conserver une très grande partie de la variance totale des données en utilisant un nombre relativement faible de composantes.

Le paramètre `random_state` est utilisé ici pour garantir que les données sont mélangées de la même manière à chaque exécution de l'algorithme, il est souvent fixé à une valeur constante, comme 42, afin de pouvoir reproduire facilement les résultats d'une expérience.

Enfin on crée donc notre nouveau jeu de données, qui est `X_reduced`, en passant en paramètre de la fonction `fit_transform`, notre jeu de données de base, sur lequel nous avons réalisé notre feature extraction.

De ce fait, nous avons divisé notre nombre de données par presque 9, en passant de 34191 à 4059 features, ce qui est parfait pour notre objectif.

### iii. Elbow Method

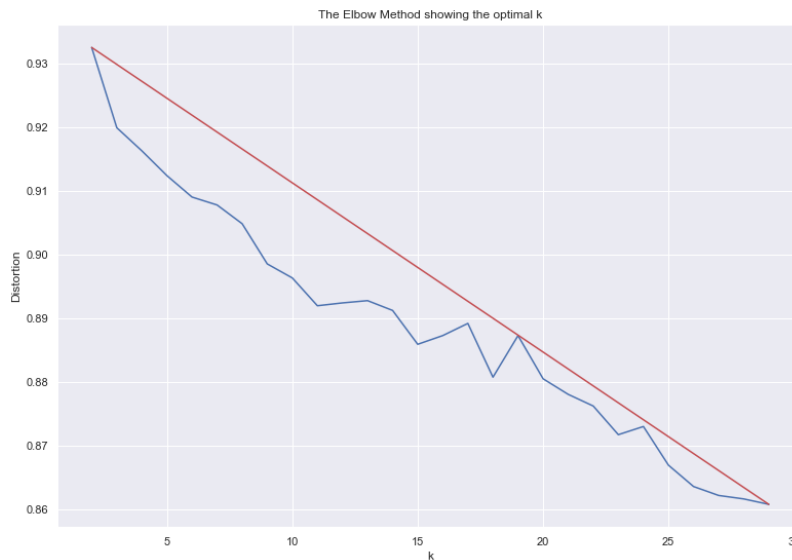
La méthode de coude (ou "elbow method" en anglais) est une technique utilisée pour déterminer le nombre optimal de clusters à utiliser dans l'algorithme de clustering k-means. Elle se base sur l'intuition que le nombre optimal de clusters est celui pour lequel l'augmentation de la distance moyenne entre les points de données et leur centre de cluster correspondant (appelée "inertie") commence à diminuer moins rapidement. Ce "coude" sur le graphique de l'inertie en fonction du nombre de clusters est censé être le nombre optimal de clusters.

Voici notre code permettant de réaliser cela :

```
from sklearn import metrics
from scipy.spatial.distance import cdist

# run kmeans with many different k
distortions = []
K = range(2, 30)
for k in K:
    k_means = KMeans(n_clusters=k, random_state=42).fit(X_reduced)
    k_means.fit(X_reduced)
    distortions.append(sum(np.min(cdist(X_reduced, k_means.cluster_centers_, 'euclidean'),
axis=1)) / X.shape[0])
```

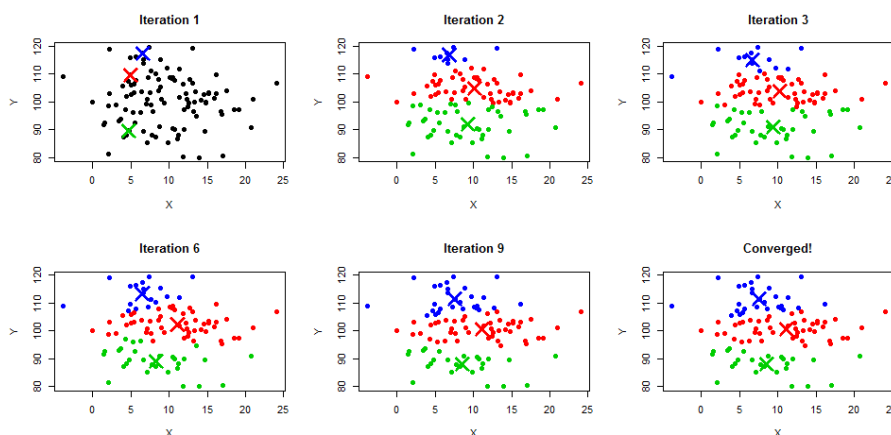
Nous réalisons donc un KMeans avec un nombre de clusters allant de 2 à 30, puis nous stockons les distorsions, et les affichons dans un graph à l'aide de la librairie python Matplotlib et voici les résultats :



Nous nous souvenons de nos 12 catégories et nous trouvons qu'un coude intéressant semble être celui à  $k=12$ , nous décidons alors de choisir ce nombre de clusters, qui correspond, qui plus est, parfaitement à notre projet de base.

#### iv. Kmeans

Le K-means est un algorithme de clustering qui permet de regrouper des points de données en "k" clusters distincts de manière à minimiser la distance euclidienne entre les points de données et le centre de leur cluster correspondant. Il utilise une approche itérative pour diviser les données en clusters, en commençant par initialiser de manière aléatoire les centres de clusters et en assignant chaque point de données au cluster dont le centre est le plus proche. Il ajuste ensuite les centres de clusters pour minimiser la distance entre les points de données et leurs centres de clusters respectifs, puis réassigne chaque point de données au cluster dont le centre est le plus proche. Cette procédure est répétée jusqu'à ce que les centres de clusters ne bougent plus ou jusqu'à ce qu'un nombre maximum d'itérations soit atteint. Voici un exemple visuel avec  $k=3$  :



De ce fait, nous lançons une itération d'un modèle KMeans sur nos données avec un nombre de clusters  $k=12$ . Enfin, nous stockons les labels trouvés.

Nous souhaitons maintenant visualiser nos données. Pour ce faire, nous allons utiliser un T-SNE.

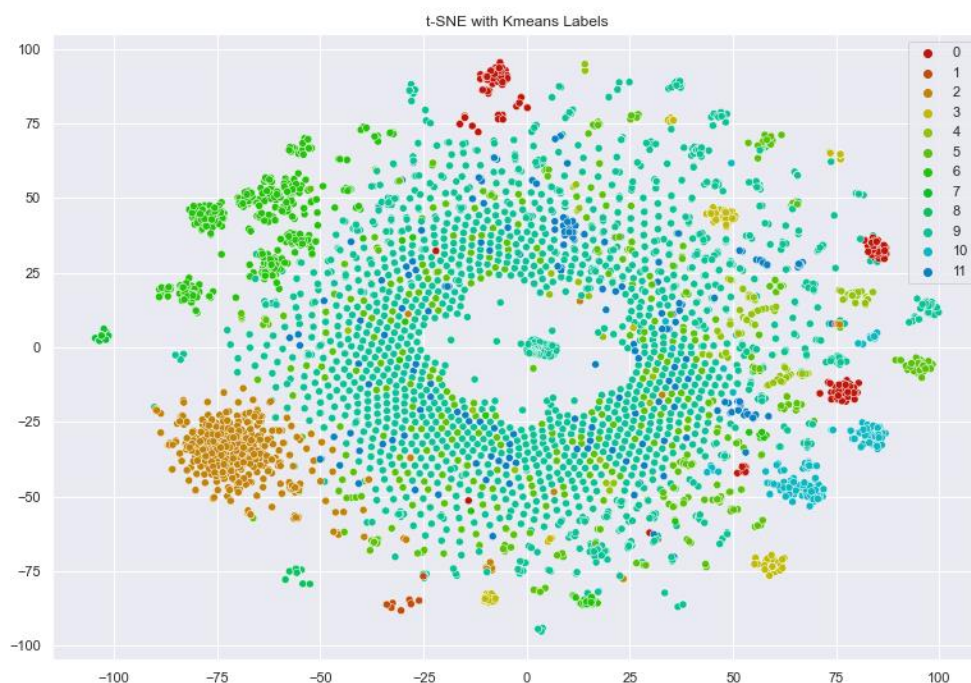
## v. TSNE

Le t-Distributed Stochastic Neighbor Embedding (t-SNE) est une technique de visualisation de données qui permet de représenter de manière efficace des données à haute dimension dans un espace à basse dimension, généralement en 2 ou 3 dimensions. Nous décidons de l'utiliser afin de visualiser les données du résultat de notre algorithme de clustering k-means.

Le t-SNE est basé sur l'idée de conserver la distance entre les points de données lors de la projection des données dans l'espace à basse dimension. Plus précisément, il tente de conserver la proximité des points de données qui sont proches dans l'espace à haute dimension dans l'espace à basse dimension, tout en séparant les points de données qui sont éloignés dans l'espace à haute dimension. Cela permet de visualiser de manière claire les structures et les groupes présents dans les données.

Le t-SNE est souvent considéré comme une technique plus efficace que la principal component analysis (PCA) pour la visualisation de données à haute dimension, car il permet de mieux conserver les structures non linéaires présentes dans les données.

Après itération, voici notre résultat :



On peut retrouver certains patterns comme les clusters 2 et 6 mais cela ne nous informe pas plus que ça pour notre objectif final.

Nous décidons à ce niveau, d'extraire les termes les plus récurrents dans nos clusters afin de comprendre à quoi ces derniers correspondent. Nous utilisons un algorithme LDA pour réaliser cette tâche.

## vi. LDA

La Latent Dirichlet Allocation (LDA) est un modèle de traitement de données qui permet de découvrir les thèmes cachés (ou "latents") présents dans un corpus de documents. Le modèle LDA utilise une approche probabiliste pour découvrir les thèmes dans un corpus de documents. Il suppose que chaque document est composé de mots qui sont générés par un certain nombre de thèmes sous-jacents. Chaque mot dans le document est censé être attribué à l'un des thèmes sous-jacents avec une certaine probabilité.

Le modèle LDA est capable de découvrir ces thèmes sous-jacents en utilisant un processus d'apprentissage par essais et erreurs. Il commence par attribuer de manière aléatoire chaque mot dans chaque document à l'un des thèmes sous-jacents, puis il ajuste ces attributions de manière à maximiser la probabilité du corpus de documents donné.

Nous avons donc, finalement, récupéré les thèmes présents dans chacun des douze clusters obtenus avec le Kmeans. En définissant `n_components=12` dans notre modèle LDA, nous avons récupéré les 12 thèmes de chaque cluster.

Voici les résultats :

- Cluster 0 : ['lydia', 'paris', 'offer', 'stage', 'work', 'new', 'transfer', 'change', 'payment', 'free', 'month', 'communication', 'placement', 'account', 'eur', 'bank', 'head', 'kid', 'use']
- Cluster 1 : ['rent', 'studio', 'room']
- Cluster 2 : ['lutt', 'address', 'troyes', 'sport', 'isi', 'prevent', 'decline', 'day', 'holiday', 'activity', 'student', 'par', 'registration', 'piece', 'international', 'content', 'use', 'send', 'semester', 'michel', 'mission', 'baptiste', 'guichard', 'list', 'document', 'benoit', 'fellow', 'enrich', 'reopen']
- Cluster 3 : ['digitale', 'converse', 'write', 'lusine', 'cnil', 'right', 'address', 'gaulle', 'infopro', 'mail', 'send', 'datum', 'contact', 'personal', 'essec', 'school', 'advert', 'business', 'ndeg', 'deposit']
- Cluster 4 : ['june', 'day', 'apr', 'oct', 'hollyhill', 'price', 'event', 'central', 'wednesday', 'time', 'thursday', 'summer', 'tuesday', 'jul', 'list', 'selection', 'return', 'master', 'talent', 'mar', 'game', 'monday', 'feb', 'dec', 'trainee']

- Cluster 5 : ['university', 'director', 'troyes', 'utt', 'asia', 'course', 'offer', 'hello', 'company', 'oscar', 'forum', 'student', 'benoit', 'application', 'study', 'office', 'international', 'cegep', 'password', 'canada', 'professional', 'training', 'day', 'registration']
- Cluster 6 : ['france', 'job', 'network', 'join', 'des', 'wilton', 'company', 'profile', 'mail', 'place', 'people', 'ago', 'day', 'opportunity', 'inmail', 'recruiter', 'benoit', 'comment', 'oscar', 'premium', 'view', 'share', 'enjoy', 'professional', 'help']
- Cluster 7 : ['boursorama', 'banque', 'canada']
- Cluster 8 : ['exactly', 'message', 'link', 'choose', 'place', 'close', 'pass', 'store', 'select', 'date', 'computer', 'dnum', 'connection', 'view', 'send', 'dictionary', 'self', 'irc']
- Cluster 9 : ['eur', 'message', 'paris', 'baptiste', 'new', 'application', 'thank', 'datum', 'hello', 'job', 'offer', 'guichard', 'utt', 'send', 'que', 'test', 'apple', 'week', 'account', 'vous', 'mail', 'use', 'interview', 'good', 'work', 'make', 'pour', 'travel', 'reservation', 'flight', 'pierre', 'united', 'nicolas']
- Cluster 10 : ['history', 'thom', 'eur', 'instead', 'offer', 'private', 'lunite', 'online', 'selection', 'limit', 'available', 'item', 'purchase', 'store', 'valid', 'gold', 'shop', 'product', 'datum', 'receive', 'delivery', 'rcs', 'nanterre', 'operation']
- Cluster 11 : ['hello', 'train', 'travel', 'serge', 'datum', 'mission', 'baptiste', 'project', 'rohmer', 'utt', 'troyes', 'lutt', 'fsdie', 'file', 'student', 'transaction', 'ticket', 'ter', 'technology', 'engineer', 'questionnaire', 'teaching', 'tour', 'green', 'balance', 'progress', 'colleague', 'inspire']

Cependant, il nous fallait trouver un moyen d'assigner chaque cluster à une potentielle catégorie à laquelle il appartient. Pour rappel, nous avons défini 12 catégories au départ. Pour réaliser cette tâche, nous avons donc créé un lexique, composé d'un certain nombre de mots pour chaque catégorie.

## vii. Création d'un lexique pour l'assignation des clusters

La création du lexique a été réalisé à l'aide de différents outils : Tout d'abord, les mots présents dans les thèmes des clusters, qui pouvaient nous aider à trouver à quel catégorie un cluster appartient : par exemple, un clusters avec comme thème récurrent les mots « password, account, change » a de grandes chances d'appartenir à la catégorie « Comptes et MDP ». Ce lexique a ensuite été étoffé à l'aide de ChatGPT, un chatbot d'OpenAI. Et enfin, à l'aide de notre vocabulaire personnel, nous avons ajouté les mots qui nous semblaient pertinents.

Voici le lexique final :

```
lexique = {
  "Université": ["campus", "professor", "lecture", "semester", "degree", "enrollment", "tuition",
    "dormitory", "library", "exam", "grade", "credit", "major", "minor", "class", "lecture hall", "research",
    "thesis", "dissertation", "plagiarism", "academic integrity", "curriculum", "internship", "fellowship",
    "grant", "bursary", "tuition fee", "scholarship", "fellowship", "grant", "bursary"],
  "Publicités et Newletters": ["advertising", "promotion", "marketing", "sales", "banner",
    "brochure", "catalog", "flyer", "poster", "billboard", "commercial", "infomercial", "endorsement",
```



```

"testimonial", "mailing list", "newsletter", "email marketing", "influencer marketing", "customer
acquisition", "conversion rate", "offer", "discount", "product"],
"Commandes et Tickets": ["order", "purchase", "checkout", "shopping cart", "invoice",
"receipt", "ticket", "reservation", "booking", "confirmation", "cancellation", "refund", "delivery",
"shipping", "tracking", "customer service", "call center", "hotline", "escalation", "complaint",
"feedback", "review", "rating", "warranty", "guarantee", "return policy", "exchange policy"],
"Banque et Factures": ["bank", "savings account", "checking account", "credit card", "debit
card", "ATM", "online banking", "mobile banking", "investment", "asset", "liability", "interest", "fees",
"charge", "balance", "transaction", "deposit", "withdrawal", "transfer", "loan", "mortgage",
"insurance", "bill", "invoice", "statement", "payment", "late fee", "penalty"],
"Rendez-vous": ["appointment", "meeting", "schedule", "agenda", "calendar", "deadline",
"reminder", "notification", "confirmation", "cancellation", "reschedule", "no-show", "waiting list",
"booking", "reservation", "availability", "flexibility", "urgency", "priority", "request", "proposal",
"offer", "counteroffer", "negotiation", "agreement", "disagreement", "compromise"],
"Réseaux Sociaux": ["social media", "network", "platform", "profile", "timeline", "feed", "friend",
"follower", "like", "comment", "share", "tag", "hashtag", "mention", "DM", "PM", "post", "tweet",
"status", "story", "live", "video", "photo", "image", "link", "hashtag", "trending", "viral", "influencer",
"advertising", "marketing"],
"Administratif": ["administration", "office", "manager", "staff", "employee", "human resources",
"personnel", "policy", "procedure", "regulation", "law", "contract", "agreement", "permit",
"license", "certificate", "document", "record", "file", "archive", "database", "system", "software",
"hardware", "network", "security", "access", "privilege", "permission"],
"Candidatures job et milieu Professionnel": ["job", "application", "employment", "career",
"professional", "work", "resume", "CV", "cover letter", "application", "interview", "selection", "hiring",
"promotion", "training", "development", "performance", "evaluation", "feedback", "review",
"salary", "benefit", "perk", "incentive", "bonus", "raise", "promotion", "advancement", "progression",
"succession"],
"Comptes et Mots de passe": ["account", "username", "password", "login", "logout", "security",
"privacy", "protection", "encryption", "authentication", "verification", "authorization", "access",
"permission", "privilege", "restriction", "violation", "hacking", "phishing", "spam", "virus", "malware",
"ransomware", "trojan", "worm", "spyware", "adware", "popup"],
"Logement": ["housing", "rent", "rental", "home", "residence", "apartment", "condo",
"townhouse", "mansion", "villa", "manor", "cottage", "farmhouse", "landlord", "tenant", "rent",
"lease", "agreement", "contract", "mortgage", "property", "real estate", "development",
"construction", "renovation", "furniture", "appliance", "utility", "amenity", "facility"]
}

```

Ensuite, pour assigner chaque cluster à une catégorie nous avons écrit ce petit algorithme :

```

# utiliser le lexique
i=0
categories = {}

for w in all_keywords:
    j = 0
    word_occurrence = []
    for cat in lexique:
        nb = 0
        for lex in lexique[cat]:
            if lex in w:

```



```

        nb = nb+1
        word_occurence.append(nb)
        cat_number = word_occurence.index(max(word_occurence))
        print(word_occurence)
        keys = list(lexique.keys())
        # Use the index to access the key
        key = keys[cat_number]
        categories[i] = key
        print("CATEGORIE pour le cluster "+str(i)+" : ", key)
        i=i+1

```

L'algorithme parcourt chaque cluster (for w in all\_keywords) et compte le nombre de mots-clés qui apparaissent dans chaque catégorie du lexique (for cat in lexique). Il stocke ces valeurs dans une liste word\_occurence. Ensuite, il détermine la catégorie qui contient le plus grand nombre de mots-clés dans le cluster en trouvant l'index du maximum de la liste word\_occurence (cat\_number = word\_occurence.index(max(word\_occurence))). Il utilise cet index pour accéder à la catégorie correspondante dans le lexique (key = keys[cat\_number]) et l'assigne au cluster (categories[i] = key).

Voici le résultat final :

- Cluster 0 : Banque et Factures
- Cluster 1 : Logement
- Cluster 2 : Université
- Cluster 3 : Banque et Factures
- Cluster 4 : Candidatures job et milieu Professionnel
- Cluster 5 : Candidatures job et milieu Professionnel
- Cluster 6 : Réseaux Sociaux
- Cluster 7 : Université
- Cluster 8 : Réseaux Sociaux
- Cluster 9 : Candidatures job et milieu Professionnel
- Cluster 10 : Publicités et Newsletters
- Cluster 11 : Banque et Factures

Ces résultats semblent logiques mais ne sont pas réellement satisfaisants. En effet, toutes nos catégories ne ressortent pas vraiment et plusieurs reviennent, sans réellement de certitude sur leur exactitude. Nous décidons alors de tester un autre algorithme, un modèle qui est un des plus performants pour les tâches que nous devons réaliser : le BERTopic.

## c. Nouveau modèle : BERTopic

BERTopic est une bibliothèque de modélisation de sujets en Python qui combine des embeddings de transformateur et des algorithmes de clustering pour identifier les sujets en NLP (traitement du langage naturel). Il a été développé par Google en 2018.

Nous allons dans un premier temps parler des algorithmes qui se trouvent derrière le modèle BERTopic.

Tout d'abord, nous devons obtenir les embeddings pour tous les documents. BERTopic utilise par défaut la version anglaise de sentence\_transformers pour obtenir les embeddings de documents. S'il y a plusieurs langues dans le document, nous pouvons utiliser BERTopic(language="multilingual") pour prendre en charge la modélisation de sujets dans plus de 50 langues.

Après que les documents texte aient été transformés en embeddings, la prochaine étape consiste à exécuter un modèle de clustering sur les documents. Comme les vecteurs d'embedding ont généralement des dimensions très élevées, des techniques de réduction de dimensions sont utilisées pour réduire les dimensions. L'algorithme par défaut pour la réduction de dimensions est UMAP (Uniform Manifold Approximation & Projection). Comparé à d'autres techniques de réduction de dimensions telles que l'analyse en composantes principales (PCA), UMAP conserve la structure locale et globale des données lors de la réduction de la dimensionnalité, ce qui est important pour représenter les sémantiques des données textuelles. BERTopic offre la possibilité d'utiliser d'autres techniques de réduction de dimensionnalité en modifiant la valeur umap\_model dans la méthode BERTopic. L'algorithme par défaut pour le clustering est HDBSCAN. HDBSCAN est un modèle de clustering basé sur la densité. Il identifie automatiquement le nombre de clusters et ne nécessite pas de spécifier le nombre de clusters à l'avance, contrairement à la plupart des modèles de clustering. Cependant, dans notre cas d'usage, on va lui passer en paramètres le nombre de topics souhaités car nous souhaitons retrouver 12 catégories dans notre cas.

Après avoir assigné chaque document du corpus à un cluster, la prochaine étape consiste à obtenir la topic representation en utilisant un c-TF-IDF (TF-IDF de classe). Les mots les plus importants avec les scores c-TF-IDF les plus élevés sont sélectionnés pour représenter chaque sujet. Le c-TF-IDF est similaire au TF-IDF en ce qu'il mesure l'importance des termes par les fréquences de termes tout en prenant en compte le corpus entier (toutes les données textuelles pour l'analyse). Le c-TF-IDF diffère du TF-IDF en ce que le niveau de fréquence de termes est différent. Dans le TF-IDF régulier, le TF mesure la fréquence de termes dans chaque document. Alors que dans le c-TF-IDF, le TF mesure la fréquence de termes dans chaque cluster, et chaque cluster inclut de nombreux documents.

Passons maintenant au code qui nous permet de réaliser ce clustering et topic representation avec BERTopic :

En effet, la première étape est bien sûr d'installer les packages nécessaires qui sont :

```
import pandas as pd
import numpy as np
import ast
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from bertopic import BERTopic
```

```
from umap import UMAP
```

Ensuite, on récupère évidemment notre dataset qui est déjà nettoyé, et on y récupère plus précisément le corps du mail, qui est ce que l'on va traiter.

Nous créons alors à ce moment des instances de modèle UMap et de BERTopic :

```
# Initiate UMAP
```

```
umap_model = UMAP(n_neighbors=15,  
                  n_components=5,  
                  min_dist=0.0,  
                  metric='cosine',  
                  random_state=100)
```

```
# Initiate BERTopic
```

```
topic_model = BERTopic(umap_model=umap_model, language = "multilingual", calculate  
_probabilities=True, nr_topics=12)
```

```
# Run BERTopic model
```

```
topics, probabilities = topic_model.fit_transform(emails)
```

Le modèle BERTopic produit par défaut des résultats différents chaque fois en raison de la stochasticité héritée de UMAP. Pour obtenir des sujets reproductibles, nous devons passer une valeur au paramètre `random_state` dans la méthode UMAP. Ici, `random_state=100`.

`n_neighbors=15` signifie que la taille de voisinage local pour UMAP est de 15. C'est le paramètre qui contrôle la structure locale versus globale dans les données. Un faible nombre de voisins oblige UMAP à se concentrer davantage sur la structure locale et peut perdre les insights sur le grand schéma. Un grand nombre de voisins pousse UMAP à regarder le voisinage plus large et peut perdre les détails sur la structure locale. La valeur par défaut de `n_neighbors` pour UMAP est 15. C'est la dimension des données qui sera passée au modèle de clustering.

`min_dist` contrôle à quel point UMAP est autorisé à empiler les points ensemble. C'est la distance minimale entre les points dans l'espace à faible dimension. De faibles valeurs de `min_dist` entraînent des plongées plus compactes, ce qui est bon pour le clustering. Comme notre objectif de réduction de dimension est de construire des modèles de clustering, nous fixons `min_dist` à 0. De grandes valeurs de `min_dist` empêchent UMAP d'empiler les points ensemble et préservent la structure large des données.

`metric='cosine'` indique que nous utiliserons le cosinus pour mesurer la distance.

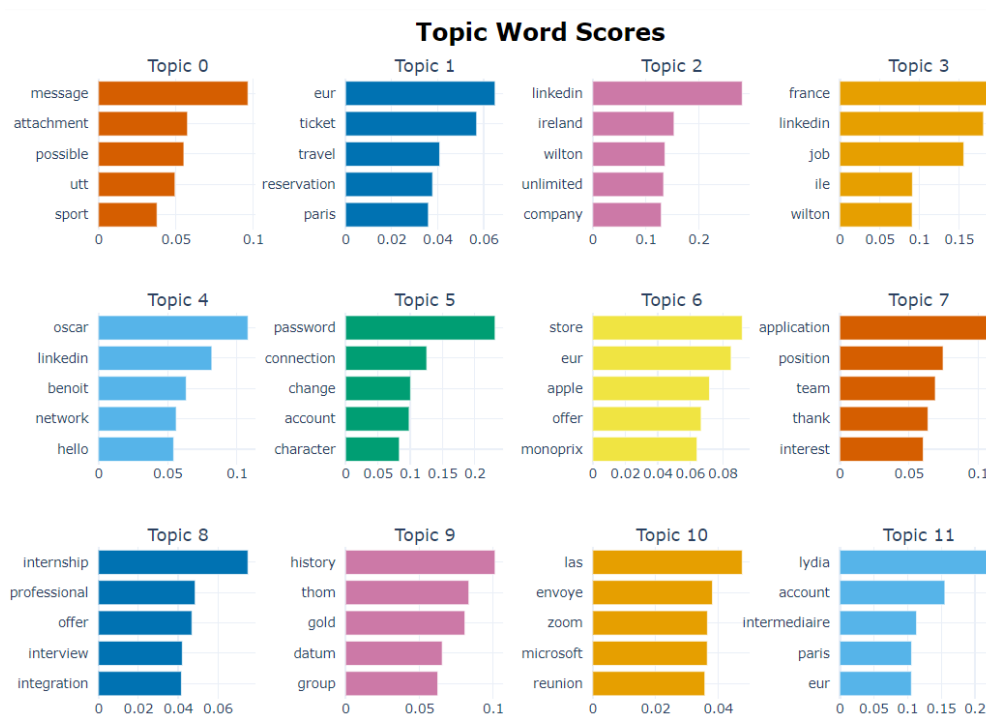
Après avoir initié le modèle UMAP, nous le passons au modèle BERTopic, définissons la langue sur `multilingual` (pour sécuriser le modèle) et définissons le paramètre `calculate_probabilities` sur `True`.

Enfin, nous passons les documents traités au modèle de sujet et enregistrons les résultats pour les topics et les probabilités de sujet.

Les valeurs de topics représentent le topic auquel chaque document est attribué. Les valeurs de probabilités représentent la probabilité qu'un document appartienne à chaque topic.

Par la suite, nous souhaitons visualiser les top words dans chacun de nos topics.

Voici le résultat :



Les résultats sont plutôt très satisfaisants et nous souhaitons donc classifier chaque cluster avec la technique du lexique, déjà utilisée, et expliquée précédemment lors du KMeans.

Voici alors les résultats obtenus :

- Topic 0 : RDV
- Topic 1 : Commandes/Tickets
- Topic 2 : Réseaux Sociaux
- Topic 3 : Pro/Candidatures
- Topic 4 : Réseaux Sociaux
- Topic 5 : Account & Password
- Topic 6 : Publicité
- Topic 7 : Pro/Candidatures

- Topic 8 : Pro/Candidatures
- Topic 9 : Publicité
- Topic 10 : Université
- Topic 11 : Economie

On retrouve donc ici toutes nos principales catégories, à l'exception de la catégorie Autres et la catégorie Perso, qui ne peuvent pas vraiment être détectées à l'aide d'un lexique. Nous trouvons alors la méthode BERTopic plus efficace et plus appropriée que notre première tentative et sommes satisfaits des résultats. Nous souhaitons désormais évaluer la performance de ce clustering. Cette partie sera décrite dans l'évaluation des modèles.

Nos modèles non-supervisés étant réalisés, il nous restait à regarder de quelle manière nous allons réaliser nos modèles supervisés. Et cela commence par la labellisation de nos données, un enjeu crucial pour notre projet.

## 5. Classification supervisée

Après avoir mis en place nos algorithmes de classification non-supervisée, nous avons ensuite essayé un autre type de classification, les algorithmes de classification supervisée. Contrairement à la méthode utilisée précédemment, celle-ci consiste à utiliser des données qui sont déjà étiquetées. Les classes sont donc définies en amont et l'ensemble de données étiqueté de ces catégories est utilisé par l'algorithme de classification pour 'apprendre' les caractéristiques associées à chaque classe. Une fois le modèle entraîné, il peut être utilisé par la suite pour prédire l'appartenance de nouvelles observations à l'une des classes en fonction de ses caractéristiques.

Ainsi, si l'on situe l'utilisation de ce type d'algorithmes dans notre projet, cela veut dire que nous devons attribuer une catégorie à chaque email, puis que le modèle apprendra à distinguer quels sont les mots les plus pertinents pour différencier telle ou telle catégorie, ce qui lui permettra ensuite d'attribuer une catégorie à un nouveau mail qu'il ne connaît pas, ce qui est exactement le but de notre projet. De plus, la classification supervisée nous permet d'avoir un contrôle total sur les catégories à différencier, et non pas seulement leur nombre comme précédemment, car c'est nous qui les choisissons lors de l'étiquetage des données.

### a. Etiquetage des données

Comme nous venons de l'expliquer, avant de mettre en place une classification supervisée, il faut dans un premier temps disposer de données étiquetées. Malheureusement, nous ne disposons pas d'emails classés selon les catégories que nous avons définies, il fallait donc attribuer une catégorie à chacun de nos emails manuellement.

Cette étape, appelée étiquetage de données ou Data Labeling en anglais, est une étape manuelle et chronophage, mais elle n'en reste pas moins une étape cruciale du projet. En effet, elle conditionne tout l'entraînement de l'algorithme car c'est grâce à elle qu'il peut apprendre à effectuer des prédictions correctes. De ce fait, si les données ne sont pas correctement étiquetées, le modèle ne pourra pas apprendre de manière adéquate et ses prédictions seront probablement incorrectes. Par conséquent, il est important de s'assurer que les données utilisées pour entraîner le modèle sont étiquetées de manière précise et complète afin d'obtenir des résultats précis lors de l'utilisation du modèle pour effectuer des prédictions.

En revanche, notre ensemble de données étant composé de 10 000 mails, il était impossible d'attribuer une catégorie à l'ensemble de ces données à la main, compte tenu du temps dont nous disposions. Nous avons alors réfléchi à des méthodes permettant de labelliser les messages de manière automatique, pour gagner du temps tout en gardant une bonne qualité de l'étiquetage, par la définition de règles. Cette automatisation était d'autant plus envisageable car notre ensemble de données n'était composé que de nos propres emails, nous connaissions donc bien leur contenu.

## i. Création de règles

Afin de créer ces différentes règles, nous pouvions nous baser sur 3 informations concernant les emails, à savoir leur objet, leur corps et l'adresse de leur expéditeur. Nous avons alors créé 3 fonctions en Python afin de définir des règles sur chacun de ces champs.

### Règles basées sur l'adresse de l'expéditeur :

Cette fonction est celle que nous avons le plus utilisé, étant donné que nos 10 000 emails ne provenaient que de 1 319 adresses différentes. Ainsi, pour les adresses dont nous étions sûrs qu'elles n'envoyaient des mails que d'une seule catégorie, nous pouvions directement étiqueter l'intégralité de ces emails expédiés par ces adresses.

```
# Labelling using email adress
def cat_with_list(df, list, str):
    i=0
    while i < len(df):
        if df['adressebis'][i] in list:
            df['cat'][i] = str
        i+=1
    return df
```

Son fonctionnement est simple, elle récupère un jeu de données, une liste d'adresses et un nom de catégorie, et attribue cette catégorie à chaque ligne dont l'adresse correspond à l'une des adresses de la liste entrée.

### Règles basées sur l'objet du message :

Cette fonction permet simplement d'attribuer une catégorie à chaque email dont l'objet contient un certain mot.

```
# Labelling with email object
def cat_with_objet(df, str_to_find, str_cat):
```

```
i = 0
while i < len(df):
    present = False
    for objet in df['objet'][i]:
        if str_to_find in objet:
            present=True
    if present == True:
        df['cat'][i] = str_cat
    i+=1
return df
```

Elle récupère un jeu de données, le mot à trouver et le nom de la catégorie associée. Elle attribue ensuite cette catégorie à chaque ligne dont l'objet contient ce mot.

### Règles basées sur le corps du message :

Cette fonction a l'air plus compliquée mais elle a un fonctionnement simple également. Elle permet d'attribuer une catégorie à chaque email dont le même corps contient plusieurs mots.

```
# Labelling with words in main content
def cat_with_words(df, words, str):
    words_bool = []
    x=0
    while x < len(words):
        words_bool.append(False)
        x+=1
    i=0
    while i < len(df):
        x=0
        while x < len(words):
            words_bool[x] = False
            x+=1
        for word in df['corps'][i]:
            x = 0
            while x < len(words):
                if word == words[x]:
                    words_bool[x] = True
                    x+=1
        x=0
        c=0
        while x < len(words):
            if words_bool[x] == True:
                c+=1
            x+=1
        if c == len(words):
            df['cat'][i] = str
```



```
i+=1
```

```
return df
```

Elle récupère un jeu de données, une liste de mots à trouver et le nom de la catégorie associée. Elle attribue ensuite cette catégorie à chaque ligne dont le corps contient l'ensemble des mots contenus dans la liste. Si la liste de mots à trouver contient 3 mots et que le corps n'en contient que 2 sur 3, le mail ne sera alors pas catégorisé.

## ii. Utilisation des règles pour chaque catégorie

Une fois que nous avons créé ces fonctions, nous avons pu les mettre en pratique avant d'étiqueter nos 10 000 mails, dans l'ordre des catégories :

### Catégorie 1 : Université

Pour cette catégorie, la méthode à privilégier était forcément via l'adresse des expéditeurs, qui devaient normalement être composées de la chaîne de caractère « utt ». Mais nous ne pouvions pas classer directement tous les emails utt dans cette catégorie, étant donné qu'il y a aussi des étudiants qui sont des amis avec qui nous avons discuté de manière informelle et/ou de sujets personnels ou en tout cas pas en rapport direct avec l'UTT. Nous avons alors affiché l'ensemble des mails utt afin de créer deux listes distinctes, une avec des expéditeurs concernant l'université et une composée d'étudiants/amis :

```
for i in data['adressebis'].unique():
    if 'utt' in i:
        print(i)
```

Nous avons ensuite attribué la catégorie 'Université' aux mails envoyés par les adresses contenues dans la liste univ :

```
data = cat_with_list(data, univ, 'Université')
```

Etant donné que nous sommes également tous les deux partis en échange universitaire à deux reprises, nous avons également des mails universitaires de 4 autres universités. En revanche, nous n'avons pas eu d'échange par mail avec des étudiants de ces universités, alors nous avons pu directement étiqueter tous les emails envoyés par des adresses contenant les chaînes de caractères correspondantes aux différentes universités :

- 'cnu' pour Chungnam National University
- 'skku' pour Sungkyuankwan University
- 'sofia' pour Technical University of Sofia
- 'cegep' pour Cégep de Trois-Rivières

```
univ_echange = []
for i in data['adressebis'].unique():
```

```
if 'cnu' in i or 'skku' in i or 'sofia' in i or 'cegep' in i:
    univ_echange.append(i)

data = cat_with_list(data, univ_echange, 'Université')
```

## Catégorie 2 : Publicité

Pour cette catégorie, nous avons utilisés 2 méthodes, celle passant par les adresses et celle passant par les objets.

Pour les adresses, nous avons dans un premier temps créé 2 listes, l'une en détectant toutes les adresses composées des mots 'alert' et 'newsletter', et l'autre en ajoutant à la main toutes les adresses dont nous étions sûrs qu'elles ne nous ont écrit que pour du démarchage.

```
alerts_newsletters = []
for i in data['adressebis'].unique():
    if 'newsletter' in i or 'alert' in i:
        alerts_newsletters.append(i)
```

Nous avons ensuite pu ajouter la catégorie à chacune des lignes contenant ces adresses :

```
pub = alerts_newsletters + demarches
data = cat_with_list(data, pub, 'Publicité')
```

En ce qui concerne l'objet des messages, nous avons décidé de catégoriser comme 'Publicité' tous les emails dont l'objet comportait '[pub]' ou 'newsletter' :

```
data = cat_with_objet(data, 'pub', 'Publicité')
data = cat_with_objet(data, 'newsletter', 'Publicité')
```

## Catégorie 3: Commandes et Tickets

Pour cette catégorie, nous n'avons utilisé que les adresses des expéditeurs. Nous avons d'abord détecté toutes les adresses composées des mots 'billet' ou 'ticket' puis nous avons également défini une liste d'adresses qui ne nous ont contacté que pour des suivis de commandes :

```
ticket = []
for i in data['adressebis'].unique():
    if 'billet' in i or 'ticket' in i:
        ticket.append(i)

data = cat_with_list(data, ticket, 'Commandes/Tickets')
```

```
data = cat_with_list(data, orders, 'Commandes/Tickets')
```

## Catégorie 4: Economie

Là encore, nous sommes passés par les adresses d'expéditeurs, notamment car nous connaissons forcément toutes les banques ou services bancaires dans lesquels nous sommes inscrits. De plus, nous avons repéré une liste d'adresses qui ne nous envoyait que des factures. Nous avons pu alors créer une liste composée de mails bancaires et de factures, que nous avons utilisé pour attribuer la catégorie 'Economie'.

```
economie = banks + factures  
cat_with_list(data, economie, 'Economie')
```

## Catégorie 5: RDVs

Cette catégorie était difficile à étiqueter de manière automatique. Néanmoins, nous avons trouvé 2 règles l'une par l'objet et l'une par les adresses.

En ce qui concerne l'objet, nous avons simplement étiqueter tous les emails dont l'objet contenait le mot 'rdv' :

```
data = cat_with_objet(data, 'rdv', 'RDV')
```

Puis nous avons également repérer des adresses emails utilisées que pour les rendez-vous, comme celles de type google calendar, calendly ou encore zoom. Nous avons alors créé une liste de ces adresses que nous avons utilisé :

```
cat_with_list(data, rdvs, 'RDV')
```

## Catégorie 6: Réseaux Sociaux

De la même manière que nous connaissons nos banques, nous connaissons les réseaux sociaux que nous utilisons, et surtout ceux qui nous envoient des emails. Ainsi, nous avons encore répertorié dans une liste toutes les adresses emails associées aux réseaux sociaux suivants :

- Instagram
- Facebook
- Snapchat
- Twitter
- LinkedIn
- Discord

Nous avons alors catégorisé les mails expédiés par ces différentes adresses :

```
data = cat_with_list(data, social_media, 'Réseaux sociaux')
```

## Catégorie 7 : Administratif

Concernant la catégorie 'Administratif', nous avons encore répertorié l'ensemble des adresses mails qui nous ont contacté au sujet de démarches administratives que nous avons réalisé, étant donné que nous connaissions ces différentes démarches (Finances, Crous, Visas, Urssaf, Bourses, Assurances etc...). Nous avons alors utilisé cette liste d'adresses :

```
data = cat_with_list(data, administrative, 'Administratif')
```

## Catégorie 8: Pro et Candidatures

Cette catégorie regroupe l'ensemble de nos messages concernant des candidatures, que ce soit pour des jobs, des stages ou encore des programmes de formation, mais aussi tous les échanges professionnels que nous avons eus.

Ainsi, nous avons utilisé 2 méthodes, à savoir utiliser les adresses emails et le corps des messages :

Concernant le corps des mails, nous avons repéré tous les mails contenant les mots 'internship' et 'position' car nous savons que nous avons énormément candidater lors de recherches de stages, étant donné que nous avons déjà tous les deux recherché et trouvé un ST05, ST09 et un ST10 :

```
data = cat_with_words(data, ['internship', 'position'], 'Pro/Candidatures')
```

Nous avons ensuite repéré toutes les adresses mails liées à des échanges professionnels, des retours de candidatures et des programmes de formations, que nous avons lié en une seule et même liste afin d'utiliser notre fonction :

```
candidatures = contacts + masters + welcome_candidates  
cat_with_list(data, candidatures, 'Pro/Candidatures')
```

## Catégorie 9 : Perso

Cette catégorie est très délicate car la plupart des emails persos pourraient aussi être catégorisés dans une autre classe, comme les échanges concernant des stages/démarches administratives/logements etc... avec nos parents ou des échanges concernant des cours avec des amis, mais aussi car certains emails ne contiennent rien car ils ont simplement servi à envoyer un fichier. Malgré cette difficulté, nous avons fait le choix de garder cette catégorie en partant du principe que si notre outil devait voir le jour, nous serions contents d'avoir une classe dédiée aux messages personnels, même s'ils concernent des sujets qui correspondent à d'autres classes.

Ainsi, pour étiqueter cette catégorie, nous avons là aussi créé une liste d'adresses emails répertoriant tous nos proches, en ajoutant celle des étudiants/amis que nous avons créé lors de l'étiquetage de la première catégorie, 'Université' :

```
perso = proches + etu
data = cat_with_list(data, perso, 'Perso')
```

## Catégorie 10: Comptes et mdp

Cette catégorie répertorie l'ensemble des messages concernant la création de comptes, le changement de mot de passe, les alertes de sécurité de ces comptes etc... Nous avons utilisé 2 méthodes pour étiqueter cette catégorie, le corps des mails et les adresses d'expéditeurs.

Concernant le corps des emails, nous avons cherché à détecter les emails de changement de mot de passe avec les mots 'password' et 'change' et ceux de création de compte avec les mots 'account' et 'creation' :

```
data = cat_with_words(data, ['password', 'change'], 'Account & Password')
data = cat_with_words(data, ['account', 'creation'], 'Account & Password')
```

Nous avons ensuite détecté toutes les adresses contenant le mot 'account' puis nous avons identifié une liste d'expéditeurs qui ne nous envoyait que des messages concernant des informations sur différents comptes :

```
account = []
for i in data['adressebis'].unique():
    if 'account' in i:
        account.append(i)

accounts = comptes + account
cat_with_list(data, accounts, 'Account & Password')
```

## Catégorie 11 : Logement

Nous n'avions pas prévu la création de cette catégorie au départ mais nous avons décidé de l'implémenter au vu du nombre de nos emails portés sur la recherche de logements.

Nous avons alors créé une liste contenant toutes les adresses emails de nos agences immobilières et des propriétaires de logement avec qui nous avons échangés :

```
cat_with_list(data, logement, 'Logement')
```

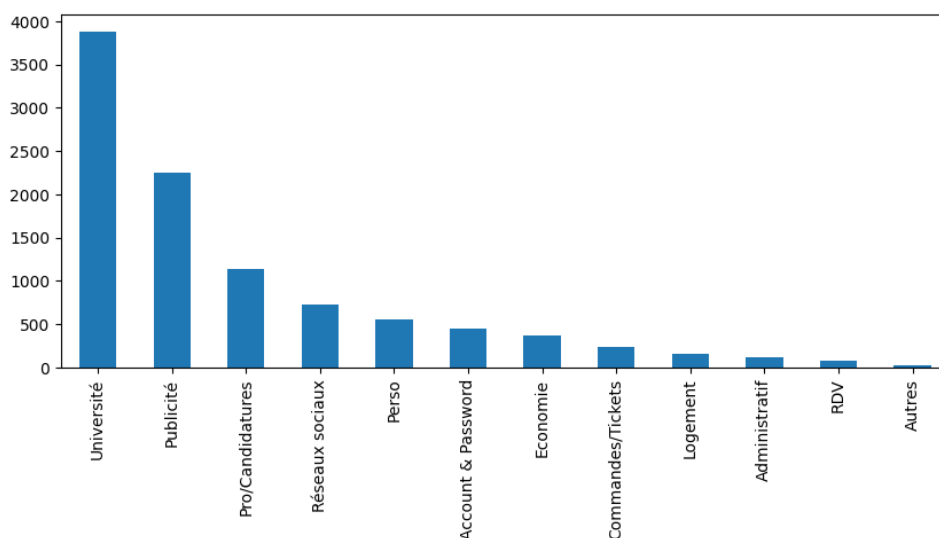
## Catégorie 12 : Autres

Cette catégorie concerne les messages qui sont écrits dans une autre langue que le français ou l'anglais, ils sont donc impossibles à catégoriser. L'automatisation de leur étiquetage n'étant pas simple, nous n'avons pas défini de règles.

Après avoir défini toutes ces règles, nous avons réussi à étiqueter 8 655 messages, il nous restait donc 1 345 emails sans catégorie. Nous avons alors décidé de terminer cet étiquetage manuellement à l'aide d'un fichier Excel.

Une fois l'intégralité des emails étiquetés, nous avons obtenus la répartition des catégories suivante :

- Université : 3886
- Publicité : 2253
- Pro/Candidatures : 1137
- Réseaux sociaux : 733
- Perso : 555
- Account & Password : 450
- Economie : 364
- Commandes/Tickets : 239
- Logement : 163
- Administratif : 113
- RDV : 81
- Autres : 26



Nous remarquons que la répartition des catégories est très inégale, mais cela paraît logique au vu des intitulés de chaque catégorie par rapport à notre profil, cela est tout simplement dû à la composition de nos données, qui sont tous nos emails.

## b. Implémentation de 3 modèles supervisés

Une fois l'ensemble de nos 10 000 données étiquetées, nous avons pu commencer à entraîner et tester des modèles de Machine Learning pour tester notre classification supervisée.

Nous avons dans un premier temps séparé notre jeu de données en 2, un jeu d'entraînement et un jeu de test, qui permettra d'évaluer la performance du modèle. Le jeu d'entraînement comporte 70% des données, soit 7 000 emails, et le jeu de test 30%, soit 3 000 emails. Ces jeux de données sont également séparés en 2 groupes, les features et les targets. Les features correspondent aux données que nous allons utiliser pour identifier des caractéristiques propres à chaque classe, alors que les targets sont les catégories correspondantes. Dans notre cas, nous n'utilisons que le corps des emails comme features, modifié suite à notre préparation de données.

```
X = df['corps']
y = df['cat']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Nous avons ensuite décidé d'implémenter et comparer 3 modèles de classification supervisée distincts afin de sélectionner le plus performant : une Classification Naïve Bayésienne, un Support Vector Machine et une Régression Logistique.

## i. Classification Naïve Bayésienne :

La classification Naïve Bayésienne (ou Naives Bayes Classifier en anglais) est un algorithme basé sur le théorème de Bayes, avec une forte supposition de l'indépendance des features. Le principe de base de cet algorithme est de calculer la probabilité d'une classe  $c$  comme étant la classe vraie pour un exemple donné :

$$P(c | x) = (P(x | c) * P(c)) / P(x),$$

où  $P(c | x)$  est la probabilité de la classe  $c$  étant vraie étant donné les caractéristiques  $x$ ,  $P(x | c)$  est la probabilité de l'observation  $x$  étant vraie étant donnée la classe  $c$ ,  $P(c)$  est la probabilité de la classe  $c$  et  $P(x)$  est la probabilité de l'observation  $x$ .

L'algorithme commence par calculer les probabilités de chaque classe ( $P(c)$ ) et la probabilité de chaque caractéristique  $x$  étant vraie pour chaque classe ( $P(x | c)$ ). Ces probabilités sont estimées à partir de l'ensemble de données d'entraînement étiqueté. Ensuite, pour chaque exemple de test, l'algorithme calcule la probabilité de chaque classe étant vraie en utilisant le théorème de Bayes. La classe avec la probabilité la plus élevée est sélectionnée comme étant la prédiction de la classe pour l'exemple de test.

La classification bayésienne est souvent utilisée en raison de sa simplicité et de sa vitesse de calcul, mais elle suppose que les caractéristiques sont indépendantes les unes des autres, ce qui peut entraîner des performances moins bonnes dans certains cas.

Dans notre cas, nous avons utilisé le modèle MultinomialNB de scikit-learn, qui est un modèle de classification naïve bayésienne spécifiquement conçu pour fonctionner avec des features catégorielles discrètes qui suivent une distribution de probabilité

multinomiale. La distribution de probabilité multinomiale est utilisée pour modéliser des features qui peuvent prendre un nombre fini de valeurs distinctes, chacune ayant une probabilité associée.

Une classification naïve bayésienne classique, en revanche, suppose que les features suivent une distribution de probabilité gaussienne (c'est-à-dire une distribution de probabilité de type "courbe en cloche"). Elle est donc mieux adaptée aux features numériques qui peuvent prendre une valeur sur une plage continue.

Le modèle MultinomialNB de scikit-learn est donc plus adapté aux features catégorielles discrètes qui suivent une distribution de probabilité multinomiale, tels que le nombre de fois qu'un mot apparaît dans un document, ce qui correspond parfaitement à notre cas d'usage.

Voici son implémentation :

```
nb = Pipeline([('vect', CountVectorizer()),
               ('tfidf', TfidfTransformer()),
               ('clf', MultinomialNB()),
               ])

nb.fit(X_train, y_train)

from sklearn.metrics import classification_report
y_pred = nb.predict(X_test)

print(classification_report(y_test, y_pred))
```

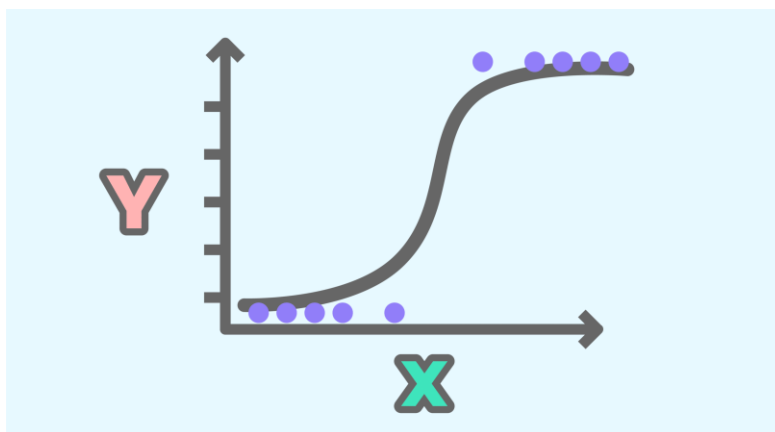
catégorie	precision	recall	f1-score	support
Account & Password	0.98	0.37	0.53	125
Administratif	0.00	0.00	0.00	38
Autres	0.00	0.00	0.00	9
Commandes/Tickets	0.00	0.00	0.00	61
Economie	1.00	0.36	0.53	106
Logement	0.00	0.00	0.00	44
Perso	0.95	0.11	0.20	158
Pro/Candidatures	0.96	0.42	0.59	347
Publicité	0.80	0.80	0.80	662
RDV	0.00	0.00	0.00	14
Réseaux sociaux	0.98	0.65	0.78	229
Université	0.63	1.00	0.77	1207
accuracy			0.71	3000
macro avg	0.52	0.31	0.35	3000
weighted avg	0.74	0.71	0.66	3000



Nous remarquons que la performance globale du modèle MultinomialNB n'est pas très bonne, ce dernier a même fait abstraction de 5 catégories. Cette sous-performance peut s'expliquer par son hypothèse de l'indépendance des features, qui implique que les mots d'une même phrase n'ont aucun lien entre eux.

## ii. Régression Logistique :

La régression logistique est utilisée pour prédire la probabilité d'un exemple appartenant à l'une des deux classes possibles (par exemple 0 ou 1). Ce modèle est basé sur la fonction sigmoïde, qui graphiquement a une forme de S et tend vers 0 en  $-\infty$  et vers 1 en  $+\infty$ . Le modèle cherche alors les meilleurs paramètres de cette fonction qui collent avec le jeu de données fourni, afin d'obtenir des résultats de ce type :



Pour notre cas d'usage, nous avons donc utilisé une régression logistique multiclasse. Elle utilise une approche « one vs rest » (un contre le reste), où plusieurs modèles de régression logistique binaire sont entraînés, chacun étant utilisé pour prédire la probabilité d'un exemple appartenant à une classe particulière par rapport aux autres classes. Par exemple, si vous avez 3 classes A, B et C, trois modèles de régression logistique binaire seraient entraînés : le premier prédirait la probabilité d'un exemple appartenant à la classe A par rapport aux classes B et C, le second prédirait la probabilité d'un exemple appartenant à la classe B par rapport aux classes A et C, et le troisième prédirait la probabilité d'un exemple appartenant à la classe C par rapport aux classes A et B.

Pour prédire la classe d'un nouvel exemple, chaque modèle de régression logistique binaire prédit la probabilité de l'exemple appartenant à chaque classe. La classe avec la probabilité la plus élevée est sélectionnée comme étant la prédiction de la classe de l'exemple.

La régression logistique multiclasse est souvent utilisée en raison de sa simplicité et de sa vitesse de calcul, mais elle peut ne pas donner de bons résultats si les classes sont très déséquilibrées (c'est-à-dire qu'il y a un très grand déséquilibre entre le nombre d'exemples appartenant à chaque classe).

Voici son implémentation :

```
logreg = Pipeline([('vect', CountVectorizer()),
                  ('tfidf', TfidfTransformer()),
                  ('clf', LogisticRegression(n_jobs=1, C=1e5)),
                  ])

logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

print(classification_report(y_test, y_pred))
```

catégorie	precision	recall	f1-score	support
Account & Password	0.81	0.84	0.82	125
Administratif	0.90	0.71	0.79	38
Autres	1.00	0.89	0.94	9
Commandes/Tickets	0.73	0.75	0.74	61
Economie	0.92	0.92	0.92	106
Logement	0.77	0.82	0.79	44
Perso	0.75	0.82	0.82	158
Pro/Candidatures	0.89	0.88	0.89	347
Publicité	0.93	0.93	0.93	662
RDV	0.83	0.71	0.77	14
Réseaux sociaux	0.97	0.94	0.95	229
Université	0.96	0.95	0.95	1207
accuracy			0.92	3000
macro avg	0.87	0.85	0.86	3000
weighted avg	0.92	0.92	0.92	3000

Nous observons que les résultats sont nettement mieux que ceux de la classification naïve bayésienne, et est relativement performant pour l'ensemble des catégories. Ces résultats sont déjà plus que satisfaisants mais nous avons quand même essayé le dernier modèle.

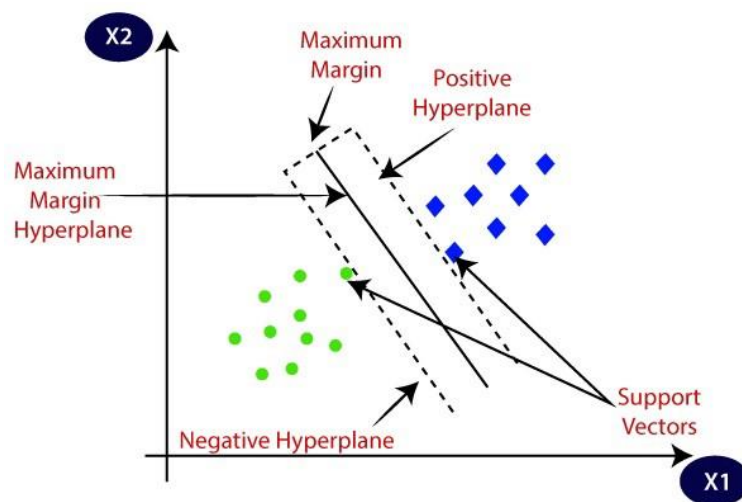
### iii. Support Vector Machine :

Un Support Vector Machine (SVM) est un algorithme qui permet de séparer les différentes classes en utilisant des hyperplans de séparation dans un espace à plusieurs dimensions. Lorsqu'il est utilisé pour la classification binaire, un SVM cherche à trouver un hyperplan qui sépare de manière la plus large possible les deux classes de points dans l'espace de caractéristiques.

Pour trouver l'hyperplan de séparation optimal, le SVM utilise des vecteurs de support qui sont des points appartenant aux classes qui sont les plus proches de l'hyperplan de séparation. La largeur de la marge, c'est-à-dire la distance entre l'hyperplan de séparation et les vecteurs de support, est maximisée afin de trouver un hyperplan de séparation qui soit le plus robuste possible.

Une fois l'hyperplan de séparation optimal trouvé, il peut être utilisé pour prédire la classe d'un nouvel exemple en mesurant la distance de cet exemple à l'hyperplan de séparation. Si l'exemple se trouve de l'autre côté de l'hyperplan par rapport à la classe majoritaire, il sera prédit comme appartenant à la classe minoritaire et vice versa.

Le SVM est souvent utilisé pour résoudre des problèmes de classification complexes en raison de sa capacité à trouver des hyperplans de séparation non linéaires en utilisant la transformation des caractéristiques. Cependant, il peut être coûteux en temps de calcul pour des ensembles de données très volumineux.



Voici son implémentation :

```
sgd = Pipeline([('vect', CountVectorizer()),
                ('tfidf', TfidfTransformer()),
                ('clf', SGDClassifier(loss='hinge', penalty='l2', random_state=42, max_iter=5, tol=None)),
                ])

sgd.fit(X_train, y_train)

y_pred = sgd.predict(X_test)

print(classification_report(y_test, y_pred))
```

catégorie	precision	recall	f1-score	support
Account & Password	0.98	0.80	0.82	125
Administratif	0.91	0.76	0.83	38
Autres	0.89	0.89	0.89	9
Commandes/Tickets	0.83	0.82	0.83	61
Economie	0.92	0.92	0.92	106
Logement	0.78	0.80	0.79	44
Perso	0.79	0.73	0.76	158
Pro/Candidatures	0.93	0.86	0.89	347

Publicité	0.93	0.94	0.93	662
RDV	0.90	0.64	0.75	14
Réseaux sociaux	0.97	0.94	0.96	229
Université	0.93	0.97	0.95	1207
accuracy			0.92	3000
macro avg	0.88	0.84	0.86	3000
weighted avg	0.92	0.92	0.92	3000

Nous remarquons que cet algorithme est à peine plus performant que la régression logistique, ce qui en fait tout de même le meilleur modèle avec une exactitude à 92%. Ce résultat est très satisfaisant, nous retenons donc ce modèle comme étant celui à utiliser dans le cadre de notre classification supervisée.

## 6. Evaluation des modèles

Une fois que nous étions satisfaits de nos modèles non-supervisés et supervisés, nous sommes passés à l'évaluation de ces modèles, afin de déterminer leur pertinence globale mais aussi pour savoir lequel est le plus adapté. Evidemment, il semble que le modèle de classification supervisée soit plus adapté dans l'état actuel des choses, étant donné que nous avons pu définir nous-même les différentes catégories d'emails et que l'exactitude du modèle était très satisfaisante.

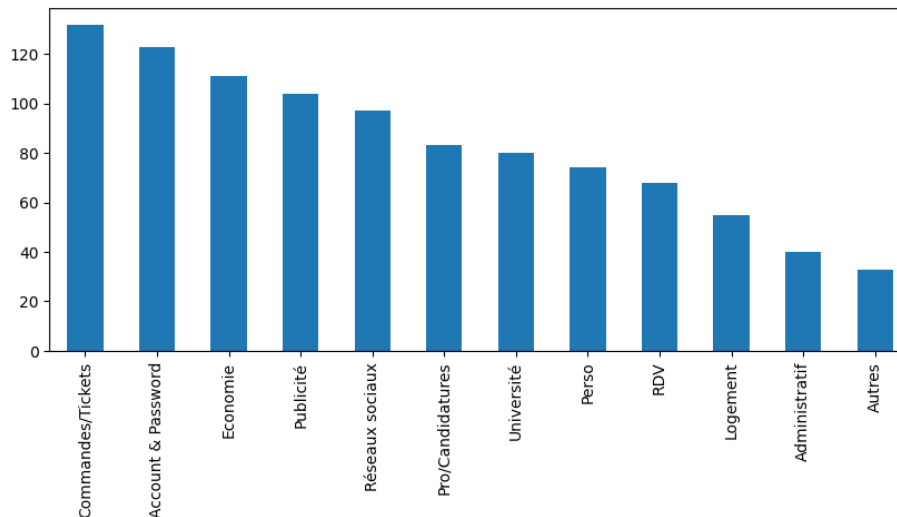
En revanche, il est probable que notre modèle soit dans une situation d'overfitting, c'est-à-dire qu'il s'est trop entraîné sur nos emails et est donc trop adapté à ces données d'entraînement et de test, au détriment de sa généralisation à de nouvelles données. Cette évaluation va donc surtout nous permettre de déterminer si le modèle reste pertinent sur des emails qui ne proviennent pas de nos boîtes mails, mais qui proviennent bien d'une personne correspondant au profil déterminé au début du projet.

Ainsi, nous avons rassemblé les emails d'une amie à nous en école de commerce, afin de faire cette vérification. Nous avons alors étiqueté ces emails manuellement, étant dans l'incapacité de définir des règles d'étiquetage automatique face à des données que nous ne connaissions pas. Au final, nous avons étiqueté 1 000 emails, en tentant d'équilibrer au maximum la répartition des classes pour avoir une meilleure approximation de la performance pour chaque classe.

Voici l'échantillon d'évaluation que nous obtenons :

- Commandes/Tickets : 132
- Account & Password : 123
- Economie : 111
- Publicité : 104
- Réseaux sociaux : 97

- Pro/Candidatures : 83
- Université : 80
- Perso : 74
- RDV : 68
- Logement : 55
- Administratif : 40
- Autres : 33



Tout d'abord, nous avons réalisé l'évaluation de notre modèle non-supervisé BERTopic. Pour ce faire, nous avons donc itéré sur nos 1000 nouveaux emails le modèle BERTopic qui avait été entraîné sur nos 10000 emails.

Nous avons ensuite récupéré la catégorie prédite par notre modèle et la catégorie que nous avons labellisé, et avec une table de contingence, nous avons pu regarder la pourcentage de prédiction correcte pour chaque clusters.

Voici les résultats :

	cat pred	Account & Password	Commandes/Tickets	Economie	Pro/Candidatures	Publicité	RDV	Réseaux Sociaux	Université
cat									
Account & Password		0.65	0.01	0.00	0.00	0.13	0.18	0.03	0.00
Administratif		0.00	0.25	0.12	0.00	0.00	0.62	0.00	0.00
Commandes/Tickets		0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
Economie		0.03	0.27	0.25	0.03	0.21	0.21	0.00	0.00
Logement		0.00	0.88	0.00	0.04	0.00	0.00	0.00	0.08
Pro/Candidatures		0.00	0.00	0.00	0.77	0.00	0.08	0.08	0.08
Publicité		0.00	0.14	0.00	0.02	0.75	0.05	0.02	0.02
RDV		0.00	0.00	0.00	0.17	0.00	0.50	0.00	0.33
Réseaux sociaux		0.03	0.00	0.00	0.00	0.00	0.00	0.90	0.07
Université		0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Voici donc des résultats mitigés pour notre modélisation non-supervisée dans ce type de projet. En effet, On peut voir que certaines catégories sont parfaitement bien prédites : Université ou encore Commandes/Tickets. Quand d'autres sont un peu moins performantes, comme l'Administratif par exemple, ou le Logement, peut être que cela est dû à une labellisation qui n'est pas forcément optimale ou alors à un choix des catégories qui font que certains sujets peuvent malheureusement se croiser. Le modèle n'est également pas capable de prédire les mails persos et Autres. Au global, sur les catégories prédites, on a une accuracy de 58,2%, ce qui est loin d'être satisfaisant pour notre projet. Quoi qu'il en soit, il nous faut désormais réaliser l'évaluation sur notre modèle supervisé afin de comparer les deux et voir, au final, lequel est le plus efficace et le plus approprié à notre problématique.

Pour ce faire, nous avons séparé le corps et la catégorie des nouveaux emails, afin d'attribuer une catégorie prédictive à partir du corps de chaque message en utilisant le modèle SVM entraîné au préalable.

Nous avons ensuite pu évaluer et afficher la performance de ce modèle sur ces nouvelles données :

catégorie	precision	recall	f1-score	support
Account & Password	0.89	0.71	0.79	123
Administratif	0.77	0.68	0.72	40
Autres	1.00	0.03	0.06	33
Commandes/Tickets	0.86	0.91	0.89	132
Economie	0.95	0.51	0.67	111
Logement	0.84	0.47	0.60	55
Perso	0.69	0.68	0.68	74
Pro/Candidatures	0.61	0.69	0.64	83
Publicité	0.64	0.90	0.75	104
RDV	0.85	0.66	0.74	68
Réseaux sociaux	0.86	0.88	0.87	97
Université	0.41	0.88	0.56	80
accuracy			0.72	1000
macro avg	0.78	0.67	0.66	1000
weighted avg	0.78	0.72	0.71	1000

Nous remarquons que le modèle atteint cette fois une exactitude de 71%. Comme nous pouvions l'imaginer, les performances sont nettement moins bonnes que lors du test sur nos propres emails, ce qui veut dire que le modèle a bien été sujet à de l'overfitting. En revanche, les résultats sont tout de même encourageants : on remarque que les performances sont bonnes pour les catégories 'Account & Password', 'Commandes/Tickets' et 'Réseaux sociaux' et très correctes pour les catégories 'Administratif', 'Economie', 'Publicité' et 'RDV'. Ces catégories contiennent probablement

un contenu très similaire peu importe l'utilisateur, au vu de leur caractère très général. En revanche, les performances sont moins bonnes pour les catégories 'Université', 'Logement', 'Pro/Candidatures', 'Perso' et 'Autres'. Ces catégories sont probablement plus personnelles, d'où les performances moins bonnes. En effet, bien que notre amie ait un profil qui correspond à notre profil cible, son profil diffère légèrement du nôtre, de par ses études (écoles de commerce) et donc ses recherches de jobs et de stages (Marketing/Communication), mais également de par les publicités qui la contacte (Marques plus féminines), ses démarches administratives (demande de permis que l'on avait pas fait) etc...

Finalement, Il est clair que notre modèle supervisé a été nettement plus performant que notre modèle non supervisé. En effet, le modèle supervisé a atteint une précision de 71%, tandis que le modèle non supervisé n'a atteint qu'une précision de 58,2%. Cette différence de performance est significative et montre que notre modèle supervisé est supérieur au modèle non supervisé pour la tâche en question. Il est possible que la nature supervisée de notre modèle lui ait permis de mieux apprendre les caractéristiques discriminantes des données et de fournir ainsi des prédictions plus précises.

Cependant, bien que 71% d'exactitude soit un résultat encourageant mais pas suffisant dans le cadre de notre outil, nous pouvons quand même imaginer qu'avec plus de données à disposition, nous pourrions largement améliorer le modèle. Ainsi, avec des données provenant de profils très hétérogènes (toujours dans notre profil cible), que ce soit par leurs études, leurs centres d'intérêts, leur sexe etc... Nous pourrions probablement avoir de meilleurs résultats. Finalement, comme beaucoup de problématique de Machine Learning, la limite est plus dans l'accessibilité des données d'entraînement que dans la technologie elle-même.

## 7. Evolutions potentielles du projet

Bien que nos résultats actuels ne soient pas assez satisfaisants pour déployer le modèle en tant qu'outil de classification d'emails et qu'il nous faudrait y consacrer plus de temps et de moyens pour y parvenir, nous avons quand même tenté d'imaginer à quoi ressemblerait le déploiement de cet outil, s'il venait à voir le jour.

Nous nous sommes donc concentrés sur 2 aspects du déploiement : la confidentialité des données de l'utilisateur et le design imaginatif de l'outil.

### a. La confidentialité des données de l'utilisateur

La confidentialité des données de l'utilisateur est un enjeu très important et non-négligeable dans le cadre de notre projet. En effet, si un utilisateur veut un jour essayer notre outil pour classer et trier ses emails, il ne faut pas que l'on puisse accéder au contenu ses emails, pour des raisons de sécurité et préservation de sa vie privée. D'un autre côté, le modèle sera de plus en plus performant s'il accède à de plus en plus de données, il faut donc pouvoir l'entraîner sur de nouvelles données de manière simple. Pour ce faire, nous avons alors identifié une possibilité, le Federal Learning.

Le Federal Learning est une technique qui permet de déployer un modèle de Machine Learning sur plusieurs appareils de différents utilisateurs, tout en préservant la confidentialité des données de chaque utilisateur.

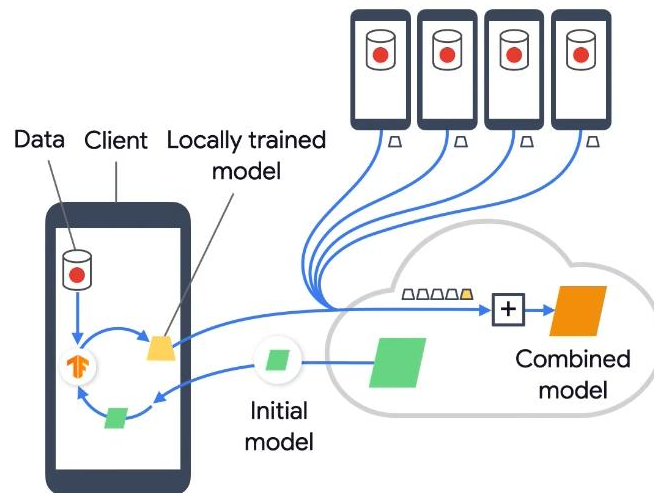
Ainsi, chaque utilisateur télécharge le modèle depuis un centre de données dans le cloud, généralement un modèle de base pré-entraîné. Il l'utilise alors pour classer ses emails sans que nous ne puissions avoir accès aux données elles-mêmes. Le modèle est quant à lui entraîné sur ses données privées, puis résumé et encrypté dans une nouvelle configuration. Les mises à jour du modèle sont renvoyées vers le cloud, décryptées et intégrées dans le modèle centralisé. Itération après itération, la formation collaborative se poursuit jusqu'à ce que le modèle soit entièrement formé.

Voici les étapes détaillées de son fonctionnement ainsi qu'un schéma explicatif :

1. Nous entraînons un modèle sur des données d'entraînement centralisées.
2. Nous déployons le modèle sur les appareils de différents utilisateurs.
3. Les appareils de chaque utilisateur se servent du modèle pour analyser leurs propres données locales et produire des mises à jour du modèle. Ces mises à jour sont envoyées à un serveur central.



4. Le serveur central agrège les mises à jour du modèle de tous les appareils et met à jour le modèle global.
5. L'utilisateur peut se servir du modèle mis à jour pour répondre à ses besoins, en lui fournissant des données d'entrée et en recevant le résultat du modèle.



Le Federated Learning permet donc à chaque utilisateur d'utiliser le modèle sans avoir à télécharger ses données sur un serveur central. Il peut alors à la fois participer au processus d'entraînement du modèle tout en l'utilisant pour classer ses emails. Cela préserve la confidentialité des données de chaque utilisateur et permet de construire un modèle global qui peut être utilisé par tous les utilisateurs.

Cependant, il est important de noter que si nous étions amenés à déployer un modèle de ce type, nous devrions mettre en place des mesures pour gérer les différences entre les appareils et garantir la qualité des données fournies par chaque appareil.

## b. Mock-up de l'outil

Bien que l'avancée de notre projet soit loin de nous permettre de déployer notre outil à l'heure actuelle, nous avons voulu imaginer à quoi ressemblerait l'interface utilisateur de cet outil si nous venions à le développer complètement un jour.

Ainsi, on remarque que les catégories de nos emails sont affichées à gauche, ce qui permet à l'utilisateur d'accéder directement à une version triée de ses messages. De plus, on remarque la présence de 3 boutons : un bouton de rafraîchissement, un bouton de classification et un bouton de règles.

Le bouton de rafraîchissement sert simplement à l'utilisateur d'actualiser son interface afin de recevoir ses nouveaux messages dans sa boîte de réception. Le bouton de classification sert quant à lui à trier les emails présents dans la boîte de réception pour les envoyer dans la bonne catégorie correspondante. Enfin, le bouton de création de règles sert à aider l'utilisateur dans sa gestion de sa boîte email, comme par exemple la création d'une règle pour supprimer tous les messages d'une certaine catégorie qui ont été envoyés il y a plus de x temps, l'archivage de mails récents d'une certaine catégorie etc...

Voici une vidéo démo de notre design tel que nous l'avons imaginé :



## 8. Retours d'expérience

Baptiste

Blablabla

Oscar

J'ai appris à comprendre les enjeux du NLP et à apprivoiser la donnée textuelle, que ce soit du data cleaning à la réalisation de modèles non supervisés et supervisés, en passant par de la feature extractio. Ce projet m'a permis de mener un projet de Machine Learning de A à Z, ce qui a été très enrichissant.

Bien que j'ai passé plus de temps sur le nettoyage de données et les algorithmes non supervisés, les rendez-vous réguliers que j'ai eus avec mon binôme m'ont permis de m'informer sur tous les sujets abordés dans ce projet. Cela m'a permis de m'immerger complètement dans l'univers du NLP et de développer mes connaissances dans ce domaine.

Ce projet a été très challengeant, mais j'ai beaucoup appris au cours de cette expérience. Je suis convaincu que ces compétences me seront utiles dans l'avenir et me permettront de poursuivre mon développement professionnel dans le domaine de la data science.

## Conclusion

Dans le cadre de ce projet, nous avons utilisé deux modèles de clustering non supervisé : Kmeans et BERTopic. Après évaluation, nous avons décidé de privilégier BERTopic car il s'est avéré plus adapté à notre problématique et est considéré comme l'un des modèles les plus performants pour le clustering. Lors de l'évaluation, nous avons obtenu un taux d'exactitude global de 58,2% sur nos catégories.

Nous avons également utilisé trois modèles supervisés : SVM, régression logistique et naïve Bayes. Le SVM s'est révélé plus robuste pour notre problème et a obtenu un taux d'exactitude exceptionnel sur nos données (plus de 90%), mais nous avons eu peur de surajuster le modèle (overfitting en anglais). Pour vérifier cette hypothèse, nous avons évalué le modèle sur de nouvelles données et avons finalement obtenu un taux d'exactitude de 71%.

Pour améliorer nos résultats, plusieurs pistes sont possibles. Tout d'abord, nous pourrions étoffer notre jeu de données en récupérant des emails de nombreux étudiants, mais le problème de confidentialité des données reste un obstacle important à surmonter. En effet, ces données sont considérées comme sensibles et il est important de veiller à leur protection. Une piste pour contourner ce problème serait de recourir à ce qu'on appelle le federative learning. Le federative learning consiste à entraîner un modèle de machine learning sur des données réparties sur différents ordinateurs ou serveurs, sans jamais les fusionner ou les stocker dans un seul endroit. Ainsi, les données restent protégées et ne sont jamais exposées à un tiers.

Une autre solution serait d'améliorer notre labellisation, qui peut parfois être erronée, mais cela nécessiterait des moyens supplémentaires.

Enfin, nous pourrions utiliser un modèle de deep learning pour la classification avec du NLP, mais cela nécessiterait l'utilisation de ressources informatiques plus puissantes (comme des GPU très performants) et un grand nombre de données d'entraînement.

## Annexes

### **Lien github du projet :**

[https://github.com/Oscarben1/PE\\_classif\\_mail.git](https://github.com/Oscarben1/PE_classif_mail.git)