ALOUGES Damien
AMSALLEM Cécile IBO1

Rapport du projet 1
Advanced Machine Learning

Lien vers le notebook :
https://colab.research.google.com/drive/1cQsNuvDWEmOqdgvSXFnow9zZifVcaS7q?usp=sharing

Importation des données :

Après avoir téléchargé le fichier xml nous avons commencé par enlever la partie renseignements du corpus, ou figure la structure du xml et la licence. Pour parser nous avons utilisé la librairie xml.etree, puis nous allons chercher “à la main” les éléments que nous cherchons dans les articles du corpus.

```
tree1 = etree.parse("corpus_taln_v2.tei.xml")
```

Cette ligne permet de récupérer le fichier xml sous forme d’un arbre, ce qui permet d’accéder au contenu des articles assez simplement, en commençant par boucler sur les articles de l’arbre :

```
root = tree1.getroot()
for article in root:
```

Ensuite il suffit de chercher les données voulues en étudiant la structure des articles du xml, par exemple pour prendre les abstracts en français nous avons procédés comme ceci :

```
abstract_fr = article[1][0][0][0].text
```

Ici chaque [ ] correspond à un niveau de l’arbre, ce qui correspond à une bannière dans le xml, par exemple article[1][0][0][0] correspond à <text> <front> <p> en xml, ce qui est l’emplacement de l’abstract en français dans les articles.

Nous avons choisi de d’importer les données dans un dataframe qui a 5 colonnes :

```
df_cols = ["abstract_fr", "keywords_fr", "intro", "titre", "date"]
```

ALOUGES Damien
AMSALLEM Cécile IBO1

La fonction text_cleaner sert à retirer le “\t\n” présent dans les données récupérées, ainsi que transformer les données type None en string “None” afin de récupérer que des données type string.

```
def text_cleaner(string):
  if string != None :
    string = string.translate(str.maketrans("\n\t", "  "))
  elif string == None:
    string = "None"
  return (string)
```

Quand la boucle sur les articles se termine on met toutes les données dans le dataframe ci-dessous :

```
1 df = pd.DataFrame(rows, columns = df_cols)
2 df
```

	abstract_fr	keywords_fr	intro	titre	date
0	Nous considérons dans notre travail la tâche ...	None	Le modèle de la Grammaire Applicative et Cogn...	Éléments de conception d'un système d'interpr...	1997
1	Nous donnons ici un aperçu du logiciel DECID ...	None	Dans le domaine de l'ingénierie linguistique ...	Informatisation du dictionnaire explicatif et...	1997
2	Diverses méthodes ont été proposées pour cons...	None		Construction d'une représentation sémantique ...	1997
3	Le terme de lambda-DRT désigne un ensemble de...	None	La « Théorie des Représentations Discursives ...	Systèmes de types pour la (lambda-)DRT ascend...	1998
4	Dans cet article, nous comparons deux modèles...	None	TAG est un formalisme initialement développé ...	Une grammaire TAG vue comme une grammaire Sen...	1998
...	...	...	...	...	...
1597	Dans cet article, nous présentons une approch...	Curriculum d'apprentissage, transfert d'appre...	L'apprentissage humain est réalisé par étapes...	Curriculum d'apprentissage : reconnaissance d...	2019
1598	Cet article présente une méthodologie de déte...	ellipse, anglais, corpus, sous-titres, détect...	L'ellipse renvoie à une incomplétude syntaxiq...	Détection des ellipses dans des corpus de sou...	2019
1599	La génération automatique de poésie est une t...	génération de poésie, réseaux de neurones, fa...	La génération automatique de poésie est une t...	La génération automatique de poésie en français	2019
1600	Nous proposons une architecture neuronale ave...	Réseaux neuronaux, modélisation de séquences,...	L'étiquetage de séquences est un problème imp...	Modèles neuronaux hybrides pour la modélisati...	2019
1601	Nous présentons la base PolylexFLE, contenant...	expressions polylexicales vebales, niveau CEC...	Les expressions polylexicales (EP) constituen...	PolylexFLE : une base de données d'expression...	2019

1602 rows x 5 columns

Comme on peut le voir sur cet affichage certain articles n’ont pas de keywords ou d’introduction ou d’abstract en français

Pré-processing :

Par la suite, nous avons utilisé un replacer pour retirer les abréviations et la ponctuation : les d’ deviennent de, les l’ deviennent des, ce qui facilitent le traitement des données. Les paternes de remplacement ont été défini par nous selon nos besoins. Nous avons également défini un tokenizer.

```
tokenizer=RegexpTokenizer("[\w]+")

replacement_patterns = [
(r'd\'', 'de '),
(r'l\'', 'le '),
(r'qu\'', 'que '),
(r',', ''),
(r'-', ' '),
(r'\.', ''),
(r'\;', '')
]

class RegexpReplacer(object):
   def __init__(self, patterns=replacement_patterns):
      self.patterns = [(re.compile(regex), repl) for (regex, repl) in patterns]

   def replace(self, text):
      s = text
      for (pattern, repl) in self.patterns:
           s = re.sub(pattern, repl, s)
      return s

replacer = RegexpReplacer()
```

Par la suite, nous créons deux nouvelles colonnes dans notre dataframe : tokenized_abstract et cleaned_abstract. Le premier sert à contenir l'abstract tokenizé et le second sert à contenir l'abstract sans ponctuation et tout en miniscule.

```
tokenized_abstract = []
cleaned_abstract = []
for abstract in df['abstract_fr'] :
  tokenized_abstract.append((tokenizer.tokenize(replacer.replace(abstract.lower()))))
  cleaned_abstract.append(replacer.replace(abstract.lower()))

df['tokenized_abstract'] = tokenized_abstract
df['cleaned_abstract'] = cleaned_abstract
```

Par la suite, nous définissons la liste des mots les plus communs de notre jeu de données, ceux-là n'ayant pas de valeur s'ajouteront à nos stopwords.

```
common_word = pd.Series(' '.join(df['cleaned_abstract']).split()).value_counts()[:50]
common_word
```

Nous retirons ensuite les stopwords et les mots communs établis ci-dessus des mots contenus dans tokenized_abstract et nous lemmatizons tous les mots restants, comme l'indique la capture d'écran ci-dessous.

```
from string import digits
nltk.download('wordnet')
nltk.download('stopwords')

stopwords = nltk.corpus.stopwords.words('french')
lemmatizer_output=WordNetLemmatizer()
for index in range(len(df['tokenized_abstract'])) :
  df['tokenized_abstract'][index] =
    [lemmatizer_output.lemmatize(word.lower(), pos='v') for word in df['tokenized_abstract'][index] if word not in common_word]
  df['tokenized_abstract'][index] =
    [lemmatizer_output.lemmatize(word.lower(), pos='v') for word in df['tokenized_abstract'][index] if word not in stopwords]
```

Enfin, nous supprimons toutes les données n'ayant ni keywords ni abstract de notre dataframe, puis nous remettons les indexes à jour pour pouvoir manipuler plus simplement les données.

```
# Ici nous supprimons les lignes n'ayant ni abstract ni keyword

num_line = []
for index in range(0, len(df)):
  if (df['abstract_fr'][index] == 'None' or df['abstract_fr'][index] == ' ') and df['keywords_fr'][index] == 'None':
    num_line.append(index)

print(num_line)
num_line.append(1545)
df = df.drop(num_line)
df.reset_index(inplace = True)
len(df)
```

Par la suite, nous avons définis une liste de mot courant, moins que ceux définis dans les mots communs, mais non représentatifs du contenu de l'article : les nombres, les verbes conjugués, et les mots de logique. Une fois tous les mots inutiles supprimés, nous utilisons un counter pour repérer les mots importants de chaque abstract.

Dans le cas où nous n'avions pas de keywords, nous incrustons cette liste de mots dans la colonne keywords_fr.

Puis, nous avons créé une nouvelle colonne new_keywords, qui concatènent les keywords d'origine et ceux obtenus par notre algorithme afin de comparer les résultats.

```
for index in range(0, len(df)):
  keywords_list = []
  if df['keywords_fr'][index] != 'None' :
    keywords_list = str(df['keywords_fr'][index]).split(',')
  new_keywords[index] = new_keywords[index] + keywords_list


df['new_keywords'] = new_keywords
```

Voici à quoi ressemble notre dataframe une fois modifié :

	index	abstract_fr	keywords_fr	intro	titre	date	tokenized_abstract	cleaned_abstract	new_keywords
0	0	Nous considérons dans notre travail la tâche ...	None	Le modèle de la Grammaire Applicative et Cogn...	Éléments de conception d'un système d'interpr...	1997	[considérons, travail, tâche, traitement, visa...	nous considérons dans notre travail la tâche ...	[relative, grande, vitesse]
1	1	Nous donnons ici un aperçu du logiciel DECID ...	None	Dans le domaine de l'ingénierie linguistique ...	Informatisation du dictionnaire explicatif et...	1997	[donnons, ici, aperçu, logiciel, decid, dévelo...	nous donnons ici un aperçu du logiciel decid ...	[logiciel, développé, geta, informatiser, réda...
2	2	Diverses méthodes ont été proposées pour cons...	None		Construction d'une représentation sémantique ...	1997	[diverses, méthodes, proposées, construire, gr...	diverses méthodes ont été proposées pour cons...	[syntaxique, construction, représentation, sém...
3	3	Le terme de lambda-DRT désigne un ensemble de...	None	La « Théorie des Représentations Discursives ...	Systèmes de types pour la (lambda-)DRT ascend...	1998	[terme, lambda, drt, désigne, ensemble, méthod...	le terme de lambda drt désigne un ensemble de...	[terme, lambda, ensemble, construire, mise, oe...
4	4	Dans cet article, nous comparons deux modèles...	None	TAG est un formalisme initialement développé ...	Une grammaire TAG vue comme une grammaire Sen...	1998	[comparons, modèles, linguistiques, utilisés, ...	dans cet article nous comparons deux modèles ...	[tag]
...	...	...	...	...	...	...	...	...	...
1539	1597	Dans cet article, nous présentons une approch...	Curriculum d'apprentissage, transfert d'appre...	L'apprentissage humain est réalisé par étapes...	Curriculum d'apprentissage : reconnaissance d...	2019	[bout, bout, extraction, concepts, sémantiques...	dans cet article nous présentons une approche...	[extraction, particulier, pilotée, curriculum,...

Clustering :

ALOUGES Damien
AMSALLEM Cécile IBO1

Après avoir obtenu des keywords pour tous les articles nous avons procédés à un clustering afin de voir quels sont les différents sujets des articles du corpus.

Nous avons commencé par faire un clustering sur les keywords de base (donc avec des valeurs “None” pour certain articles) puis un autre clustering sur les keywords de la colonne new_keywords.

La première étape était de faire une liste de tous les keywords :

```
list_keywords = []
for i in range(len(df)):
  if df.keywords_fr[i] != "None":
    list_keywords.append(df.keywords_fr[i])
```

Puis nous avons appliqué un TF IDF sur cette liste pour valoriser les mots les plus importants avec la librairie sklearn.feature_extraction.text.TfidfVectorizer

```
X = vectorizer.fit_transform(list_keywords)
```

Avec le résultat du TF IDF on peut se lancer dans le clustering, nous avons fait un clustering en utilisant l’algorithme des K-means

```
num_clusters = 9
km = KMeans(n_clusters=num_clusters)
km.fit(X)
clusters = km.labels_.tolist()
```

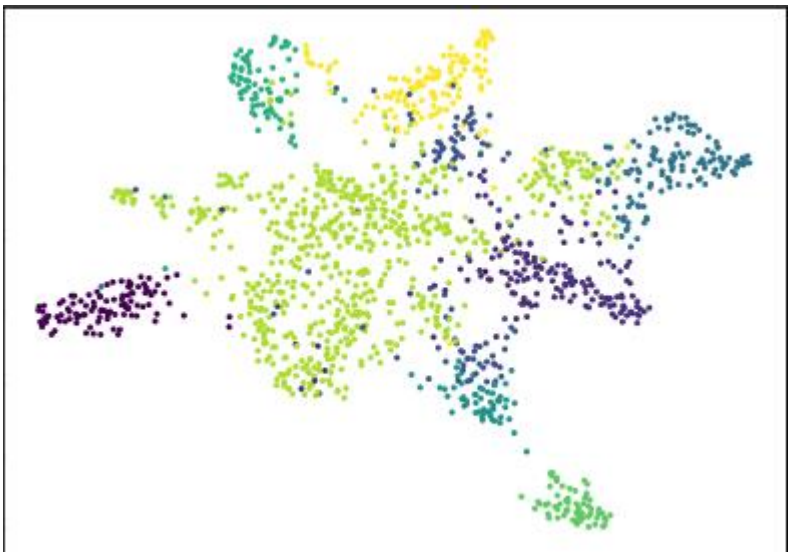
Ensuite on affiche les résultats en 2 parties, premièrement on affiche les thèmes des clusters.

```
themes 1:
analyse
syntaxique
analyse syntaxique
grammaire
grammaires
dépendance
morpho syntaxique

themes 2:
extraction
information
extraction information
entités
entités nommées
nommées
recherche
```

Puis on affiche en graphe pour voir si les clusters sont éloignés ou proches, ou chaque couleur correspond à un cluster.

ALOUGES Damien
AMSALLEM Cécile IBO1



On peut voir que bien que la plupart des clusters sont bien définit, certain se superposent un peu, ce qui est cohérent avec l'affichage précédent qui nous montrait que certains clusters se ressemblent.

Pour avoir des résultats plus proches du réel nous avons essayé de faire la même technique de clustering mais cette fois de l'appliquer sur la liste des new_keywords créée pour combler les articles n'ayant pas de keywords. Dans cette liste tous les keywords "None" sont donc remplacés par une liste de keyword et les listes de keywords déjà données par les articles sont complétés par des keywords obtenus de la même façon que pour remplacer les "None".

Finalement les résultats des deux clustering sont assez proches, certains clusters sont présents dans une méthode et pas dans l'autre et inversement, mais la majeure partie des clusters est commune aux deux méthodes.

Résultats :

Clusters présents dans les 2 techniques :

- analyse syntaxique
- dialogue homme machine
- entités nommées
- annotations
- système question réponse
- Similarité sémantique
- Arbres adjoints, grammaire
- Recherche et extraction d'information

Présents que dans la 1ere technique:

- Apprentissage automatique

Présent que dans la 2e technique :

- Résumé automatique, classification
- Traduction automatique

Avec ces résultats on obtient une liste des sujets et thèmes des articles du corpus.

Ajout de résumé des articles appartenant aux mêmes catégories :

ALOUGES Damien
AMSALLEM Cécile IBO1

Cette tentative est peu concluante, mais nous allons malgré tout la détailler ici. Le but était d'essayer de faire un résumé des articles appartenant à une même catégorie en ne prenant que leurs abstracts.

Nous avons d'abord essayé de compiler un méthode mot à mot qui n'a donné aucun résultat dans la mesure où les phrases n'étaient pas « françaises » mais une suite de mot français.

Nous avons alors choisi une approche par phrase : nous avons dû faire un nouveau type de pré-processing sur nos abstracts choisis en fonction des mots clefs comme il suit (ici nous travaillons sur un exemple pour faciliter la correction d'erreur de code) :

```
text = []
for index in range(0, len(df)):
  if keyword in df['new_keywords'][index]:
    text.append(df['abstract_fr'][index].split("."))

print(len(text))
```

Nous avons dû au passage changer nos paternes de remplacement pour éliminer les accents, difficiles à traiter par une expression régulière.

```
replacement_patterns_bis = [
(r'd\'', 'de '),
(r'l\'', 'le '),
(r'é', 'e'),
(r'à', 'a'),
(r'è', 'e'),
(r'ù', 'u'),
(r'ê', 'e')
]

replacer = RegexpReplacer(replacement_patterns_bis)

sentences = []
for sentence in text:
  sentence = replacer.replace(str(sentence))
  sentence = re.sub("[^a-zA-Z]", ' ', sentence)
  sentence = re.sub("[\s+]", ' ', sentence)
  sentences.append(sentence)
```

Nous avons en suite définie une liste de tous les mots présents dans la partie de corpus que nous étudions, afin de créer des dictionnaires.

```
bagofword = []
for sentence in sentences:
  bagofword = set(bagofword).union(set(sentence.split(" ")))
```

Ainsi qu'une liste de dictionnaire comptant les occurrences de chaque mot dans chaque abstract. Le but était de pouvoir recoder un TF-IDF adapté à notre besoin afin de l'appliquer sur des phrases, pour voir si nous pouvions utiliser uniquement des phrases déjà existantes pour faire un résumé de nos articles.

```
list_numOfWords =[]
for sentence in sentences :
  numOfWords = dict.fromkeys(bagofword, 0)
  by_word = sentence.split(" ")
  for mot in by_word:
    numOfWords[mot] = numOfWords[mot] + 1
  list_numOfWords.append(numOfWords)
```

ALOUGES Damien
AMSALLEM Cécile IBO1

Nous avons donc trois fonctions, l'une calcule un dictionnaire TF pour un abstract, l'autre calcule l'IDF du corpus. La dernière calcule le TF-IDF pour un document du corpus en prenant les résultats des deux précédentes fonctions d'après les formules suivantes :

$$\mathrm{tf}(t,d) = f_{t,d} \div (\text{number of words in d})$$

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

Ces fonctions servent à calculer le score de chaque phrase du corpus. Les phrases les plus pertinentes devraient être celles ayant le plus grand poids. Elles sont alors conservées dans le but de faire un résumé.

Ce résumé bien que compréhensible en français ne donne pas réellement d'information car cette méthode n'offre pas de lien logique entre chaque phrase. Le résultat n'est donc pas réellement pertinent. Ci-dessous, un exemple de résultat obtenu :

```
'   L apprentissage par transfert est une solution au probleme de le apprentissage de systemes de traduction automatique neuronaux
pour des paires de langues peu dotees     Dans cet article  nous proposons une analyse de cette methode     Nous souhaitons evalue
r le impact de la quantite de donnees et celui de la proximite des langues impliquees pour obtenir le meilleur transfert possible
```

Conclusion:

Avec nos résultats nous pouvons affirmer que dans le corpus d'articles fournit, les articles ont pour sujet principal le NLP, et plus précisément la liste des thèmes de cluster obtenus, c'est à dire, l'analyse syntaxique, le dialogue homme machine, la recherche, reconnaissance d'entités nommées ,les annotations, les systèmes question/réponse, les similarité sémantique, les arbres adjoints et la grammaire, la recherche et l'extraction d'information, l'apprentissage, le résumé et la traduction automatique.

Cependant nous n'avons pas réussi à faire des résumés des articles par thématiques abordées.