Auteur: CAMARA Laby Damaro

Email: ldamaro98@gmail.com

Résumé Automatique

En général, les algorithmes de récapitulation sont soit extractifs, soit abstractifs en fonction du résumé généré. Les algorithmes d'extraction forment des résumés en identifiant et en collant ensemble les sections pertinentes du texte d'entrée. Ainsi, ils ne dépendent que de l'extraction des phrases du texte original. Pour une telle raison, les méthodes d'extraction produisent naturellement des résumés grammaticaux et nécessitent relativement peu d'analyse linguistique.

En revanche, les algorithmes abstractifs sont les plus humains et imitent le processus de paraphrase d'un texte, ce qui peut générer un nouveau texte qui n'est pas présent dans le document initial. Les textes résumés à l'aide de cette technique ressemblent davantage à des humains et produisent des résumés condensés. Cependant, les techniques abstractives sont beaucoup plus difficiles à mettre en œuvre que les techniques de synthèse extractive. Il convient de mentionner que les résumés abstractifs existants reposent souvent sur un composant de prétraitement extractif pour produire le résumé du texte.

Projet

Dans cet article, nous allons partir sur le site de la radio **Mosaïque FM**. Lorsqu'un utilisateur saisi une URL on fait le résumé de l'article en question en utilisant l'approche extractive.
Nous allons décomposer le travail en deux partie:

- une première partie consistera à faire du web scraping avec les packages `request` et `BeautifulSoup`
- la partie II, nous allons utiliser le `sklearn`, le Framework `spacy` pour faire notre résumé automatique

Partie I: Web Scraping en python

Avant de commencer, il faut installer **requests** et **bs4**:

- pip install requests --user
- pip install bs4 --user

Importation des packages pour webscrapy

```
    import requests as req
    from bs4 import BeautifulSoup
```

L'URL de l'article à résumé

Ici dans cette section, vous allons inviter l'utilisateur à saisir l'URL de l'article à résumer.

```
    url  = input("Veuillez donner l'url de
l'article:")
    url
```

Faire une requête et créer un objet BeautifulSoup

Ici nous allons faire une requête avec l'url qu'on vient de saisir ci-dessus et créer un objet BeautifulSoup pour récupérer le contenu de l'article à travers le webscrping.

```
    reponse = req.get(url)
    soup = BeautifulSoup(reponse.text)
```

Créer article

Après avoir inspecter le site web de **mosaïque FM** nous constatons que l'article est contenu dans le div qui a la class dont la valeur est **desc**. Nous allons créer une variable **article** pour affecter l'article récuperer,
la methode **find_all('div')** permet de récuperer tous les div de la page, une fois tous les div récuperer nous allons chercher seulement le div qui a la class **desc** et enfin on affecte son contenu textuel à notre variable **article**. voici le code python ci-dessous:

```
    article = ""
    for div in soup.find_all('div'):
        try:
            if 'desc' in div.attrs['class']:
                article = div.text
                break;
        except:
```

```
            pass
```

Partie II

Dans cette section, nous allons réaliser la synthèse automatique à travers le Frameword **spacy** et le package **sklearn**

Importatation des packages

Dans cette sous section, nous allons importer les la classe **CountVectorizer**, **STOP_WORDS**, **fr_core_news_sm** et le Framework **spacy**

```
    from sklearn.feature_extraction.text import
CountVectorizer
    import spacy
    from spacy.lang.fr.stop_words import STOP_WORDS
    import fr_core_news_sm
```

Charger le dictionnaire

Ici nous allons charger le dictionnaire français qui contient la plupart des mots français.

```
nlp = fr_core_news_sm.load()
```

Créer un document

Ici nous allons matcher notre article au dictionnaire qu'on vient de charger plus haut.

```
doc = nlp(article)
```

Créer des token

Nous allons créer un corpus de token de phrase:

- on utilise la syntaxe **list compréhension** pour créer un liste de phrase qui constituera nos tokens
- l'attribut **sents** transforme un texte en une liste de token de phrase
- la methode **lower()** transforme un texte tout en miniscule

Model CountVectorizer

Nous allons créer un model **CountVectorizer** en passant en paramètre les **STOP_WORDS**.

STOP_WORDS

les **STOP_WORDS** sont des mots les plus employés mais qui n'apportent pas assez d'information à la phrase.

Exemple: les adverbes, prépositions, auxiliaires... .

```
cv = CountVectorizer(stop_words=list(STOP_WORDS))
```

Lier notre corpus

Nous allons lier notre corpus au modèle que nous venons de créer:

```
cv_fit = cv.fit_transform(corpus)
```

Liste des mots et nombres de mots

Ici nous allons récupéerer tous **mots** de l'article et leurs nombre de répétition dans chaque phrase.

```
mots = cv.get_feature_names();
occurs_mots = cv_fit.toarray().sum(axis=0)
```

Dictionnaire de mot

Nous allons créer un dictionnaire de mots et leurs nombre d'occurence dans l'article.

- dict() permet de créer un dictionnaire en python
- zip() prend deux liste: utilise la première comme les clées et la deuxième comme les valeurs

```
freq_mots = dict(zip(mots, occurs_mots))
```

Trier et afficher les mots les plus fréquents

- **sorted()** permet de trier les éléments d'une liste
- **items()** retourne deux liste une première les **clées** et une deuxième les **valeurs** et je stocke les clées dans **mot** et leurs **valeurs** dans **freq**
- je stocke les mots les plus fréquents dans la variable **mot_plus_freq**
- et enfin j'affiche le résultat

```
    freq_mots = dict(zip(mots, occurs_mots))
    valeurs = sorted(freq_mots.values())
    mot_plus_freq = [ mot for mot, freq in
freq_mots.items() if freq in valeurs[-3:] ]
    print("Les mots les plus fréquents sont: ",
mot_plus_freq)
```

Determiner la fréquence des mots

Récupérer le nombre max d'occurence

```
        freq_mot_plus_freq = valeurs[-1]
        freq_mot_plus_freq
```

Dictionnaire de fréquence

```
        for mot in freq_mots:
            freq_mots[mot] =
(freq_mots[mot]/freq_mot_plus_freq)
```

Affectation des score aux phrases

Le score d'une phrase est égal à la somme des fréquence des mots qui la compose.

voir le code ci-dessous

```
        score_phrase = {}
        for phrase in doc.sents:
            for mot in phrase:
                if mot.text.lower() in
freq_mots.keys():
                    if phrase in
```

```
score_phrase.keys():
                            score_phrase[phrase] +=
freq_mots[mot.text.lower()]
                        else:
                            score_phrase[phrase] =
freq_mots[mot.text.lower()]
        top_importante_phrase =
(sorted(score_phrase.values())[::-1])
        top_synthese = top_importante_phrase[:3]
```

Résumé de l'article

Ici nous allons faire la synthèse automatique de l'article maintenant.

```
        brouillon = [];
        resumer = ""
        for phrase, score in score_phrase.items():
            if score in top_synthese:
                brouillon.append(phrase)
            else:
                continue
```

```
    for phrase in brouillon:
        resume = '{} '.format(phrase)
```