Synthèse

Chargement des données

Comment obtenir la donnée à traiter ?

La donnée à traiter est récupérée d'un fichier .csv à l'aide de la librairie panda.

Préprocessing

Comment préparer la donnée à traiter ?

Avant le traitement et la classification des données, un prétraitement est nécessaire au NLP.

Ce prétraitement peut, pour l'instant, être fait à l'aide de la classe *preprocessing.py* qui regroupe les différentes méthode de préprocessing, autrement dit la normalisation, la tokénisation, la suppression des stopwords, ainsi que la lemmatisation. Cette classe est encore une ébauche de celle qui sera définitive.

Toutes ces opérations ne sont pas forcément nécessaires mais ont été implémentées dans le but de préparer le travail à venir. Dans certains papiers lus, le retrait des stopwords n'est pas considéré comme nécessairement utile. (référence, article (4))

Durant cette phase de prétraitement, deux façons de considérer les sentences / utterances ont émergé.

Dans un premier cas, il est possible de considérer les sentences / utterances comme une suite de tokens représentatifs résultants des différentes étapes de preprocessing. La normalisation, la tokénisation, le retrait des stopwords ainsi que la lemmatisation sont nécessaire.

Exemple : La sentence 'Salut, je suis en train d'apprendre' serait donc représentée par la suite de tokens ['salut', 'train', 'apprendre'].

Dans un second cas, il est possible de considérer les sentences / utterances dans leur intégralité. Aucune étape de preprocessing n'est absolument nécessaire, à l'exception de la normalisation.

Question : De quelle façon faut-il considérer la sentence ?

Représentation vectorielle d'une sentence / utterance

Comment obtenir la représentation vectorielle d'une sentence / utterance ?
Commente définir la distance / similarité entre deux sentences / utterances ?

Représentation avec SpaCy
class Svectorization.py (incomplète, ébauche)

La librairie SpaCy est dotée de fonctions permettant le word embedding. Ces différents outils ont été testés dans la classe *Svectorization.py* qui n'est encore qu'un brouillon de la solution définitive.

La librairie SpaCy permet notamment :

- d'obtenir le vecteur représentatif d'un mot / token.
  Exemple : myToken.vector

- d'obtenir la similarité entre deux tokens :
  Exemple : myToken1.similarity(myToken2)

- d'obtenir le vecteur d'une sentence :
  Exemple : myDoc1.vector

- d'obtenir la similarités entre deux sentences :
  Exemple : myDoc1.similarity(myDoc2)

Attention, l'utilisation de .similarity est toutefois accompagnée de Warnings, il faudra corriger le chargement des models.

- d'obtenir le vecteur d'une suite de tokens. Pour ce faire, plusieurs approches sont possibles, chacune justifiée par l'article (11) des références. L'addition étant une opération admise dans le word embedding, la somme des vecteurs est une approche possible permettant de définir de vecteur comme étant :
  - la moyenne des vecteurs : il est possible de seulement prendre la moyenne de tous les word vectors d'une sentence
  - la moyenne des vecteurs avec TF-IDF : il est possible de prendre la moyenne des word vectors pondérés par leurs TF-IDF

Pour ces approches 'suite de tokens', la similarité / distance reste néanmoins difficile à appréhender.

Représentation avec word2vec
code word2vec.py (brouillon pour expérimenter la méthode)

La méthode word2vec permet le word embedding. Cette approche nécessite de ne travailler qu'avec des mots / tokens ou suite de mots / tokens.

Avec cette méthode, il est nécessaire de créer un modèle ou d'en utiliser un pré-existant.

Exemple : model = Word2Vec(sentences, size=10, window=1, min_count=1, workers=1)

La méthode word2vec permet, après création et entraînement du modèle :

- d'obtenir la représentation vectorielle d'un mot / token
  Exemple : model.wv['sentence']

- d'obtenir la similarité entre deux mots
  Exemple : model.similarity('second', 'first')

- d'obtenir les mots du vocabulaire les plus similaires à celui donné
  Exemple : model.wv.most_similar('sentence')

- d’obtenir le vecteur d’une suite de tokens : à l’aide des méthodes ‘moyenne des vecteurs’ et ‘moyenne des vecteurs avec TF-IDF’ décrites ci-dessus.
La similarité reste néanmoins difficile à appréhender avec cette approche.

- de considérer des bigrames avec la fonction Phrase()

Cependant, pour une telle mise en œuvre, il est nécessaire de créer un modèle et de l’entraîner. Pour ce faire, il est possible de sauvegarder un modèle pour ensuite le recharger,

Exemple : model.save('model.bin')

```
…..
new_model = Word2Vec.load('model.bin')
more_sentences=….
new_model.build_vocab(more_sentences, update=True)
new_model.train(more_sentences, total_examples=len(more_sentences) , epochs=model.epochs)
```

Il est également possible d’utiliser des modèles déjà existants comme ceux de Google ou encore Glove mais ceux-ci sont très coûteux en espace mémoire.

La similarité / distance reste cependant difficile à appréhender si plusieurs tokens sont considérés pour représenter la sentence / utterance.

Représentation avec Doc2vec
code word2vec.py (brouillon pour expérimenter la méthode)

La méthode Doc2vec permet le document embedding (ou sentence embedding). Cette approche nécessite de travailler avec des sentences / utterances complètes .

Avec cette méthode, tout comme la précédente, il est nécessaire de créer un modèle ou d’en utiliser un pré-existant.

La méthode Doc2vec permet, après création et entraînement du modèle :

- d’obtenir le vecteur d’une sentence / utterance complète

- comparer des sentences / utterances complètes ???
- similarité ?

(travail en cours)

Question : Quel model choisir ? Quel model entrainer ?

Algorithme de classification

Quel algorithme de classification choisir ?