Natural Language processing

TP 4 – Word2vec

Dans ce TP, vous allez mettre en œuvre l'algorithme de word2vec. Word embeddings est une technique de modélisation du langage utilisée pour mapper des mots sur des vecteurs de nombres réels. Il représente des mots ou des phrases dans un espace vectoriel à plusieurs dimensions. Les incorporations de mots peuvent être générées à l'aide de diverses méthodes. Word2Vec se compose de modèles pour générer l'embeddings. Ces modèles sont des réseaux de neurones peu profonds à deux couches ayant une couche d'entrée, une couche cachée et une couche de sortie (CBOW et Skip-gram)

Prérequis

```
pip install nltk
pip install gensim
```

Exercice 1 – Entrainer son propre word2vec

1. Importer les dépendances

```
# importing all necessary modules
from nltk.tokenize import sent_tokenize, word_tokenize
import warnings
import gensim
from gensim.models import Word2Vec


warnings.filterwarnings(action = 'ignore')
```

2. Importer les données et déclarer quelques variables

```
# Reads 'alice.txt' file
sample = open("./data/alice_wonderland.txt")
s = sample.read()
```

3. Prétraitement des données

```
# Replaces escape character with space
f = s.replace("\n", " ")
data = []

# iterate through each sentence in the file
for i in sent_tokenize(f):
    temp = []

    # tokenize the sentence into words
    for j in word_tokenize(i):
        temp.append(j.lower())

    data.append(temp)
```

4. Entrainement du CBOW

```
# Create CBOW model
model1 = gensim.models.Word2Vec(data, min_count = 1, vector_size = 100, window = 5 )

# Print results
print("Cosine similarity between 'alice' " + "and 'wonderland' - CBOW : ",
    model1.wv.similarity('alice', 'wonderland'))

print("Cosine similarity between 'alice' " + "and 'machines' - CBOW : ",
    model1.wv.similarity('alice', 'machines'))
```

5. Entrainement du Skip-gram

```
# Create Skip Gram model
model2 = gensim.models.Word2Vec(data, min_count = 1, vector_size = 100, window = 5, sg = 1)

# Print results
print("Cosine similarity between 'alice' " + "and 'wonderland' - Skip Gram : ",
     model2.wv.similarity('alice', 'wonderland'))

print("Cosine similarity between 'alice' " + "and 'machines' - Skip Gram : ",
    model2.wv.similarity('alice', 'machines'))
```

   - Jouez avec le parametre **vector_size =[2, 10,500]** sur le Skipgram et CBOW**,** quel est l'effet sur les distances ?

6. Mots les plus similaires

```
#top 10 words contributing positively or negatively
model1.wv.most_similar(positive="wonderland")
model1.wv.most_similar(negative="wonderland")
```

Exercice 2 – utiliser un modèle pré entrainé

1. Télécharger le modèle pré entrainé (5 à 10 minutes)

```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
```

2. Trouver le vocabulaire du modele

```
#retrieve the vocabulary of a model.
for index, word in enumerate(wv.index_to_key):
    if index == 10:
        break
    print(f"word #{index}/{len(wv.index_to_key)} is {word}")
```

3. Retrouver un vecteur pour un mot

```
#obtain vectors for terms the model is familiar with:
vec_king = wv['king']
vec_king
```

4. Trouver la similitude entre 2 mots

```
#Word2Vec supports several word similarity tasks out of the box.
#You can see how the similarity intuitively decreases as the words get less and less similar.

pairs = [
    ('car', 'minivan'),   # a minivan is a kind of car
    ('car', 'bicycle'),   # still a wheeled vehicle
    ('car', 'airplane'),  # ok, no wheels, but still a vehicle
    ('car', 'cereal'),    # ... and so on
    ('car', 'communism'),
]
for w1, w2 in pairs:
    print('%r\t%r\t%.2f' % (w1, w2, wv.similarity(w1, w2)))
```

5. Jouez un peu avec la similitude des mots

```
#Print the 5 most similar words to "car" or "minivan"
print(wv.most_similar(positive=['car', 'minivan'], topn=5))
```

```
#Which of the below does not belong in the sequence?
print(wv.doesnt_match(['fire', 'water', 'land', 'sea', 'air', 'car']))
```