TP05 Bases de l'IA : Traitement automatique du langage naturel

Partie 1. Pre-traitement du texte

Dans cette premieme partie du TP, nous allons utiliser NLTK pour effectuer des pre-traitements du texte (tokenisation, lemmatisation,...). D'abord, suivre les instructions pour installer NLTK (`http://www.nltk.org/`)

Un exemple de tokenisation et POS tagging (fonctions grammaticales des mots).

```
import nltk
text = nltk.word_tokenize("Now we are having a Computer Science class")
nltk.pos_tag(text)
```

Un exemple de lemmatisation with PoS tag.

```
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize,pos_tag

text = "She jumped into the river and breathed heavily"
wordnet = WordNetLemmatizer()

for token,tag in pos_tag(word_tokenize(text)):
    pos=tag[0].lower()

    if pos not in ['a', 'r', 'n', 'v']:
        pos='n'

    print(token,"--->",wordnet.lemmatize(token,pos))
```

Testez plusieurs phrases (en Anglais) !

==========

Partie 2. Entrainer des word embeddings et tester des similarités entre vecteurs

Dans deuxième premieme partie du TP, nous allons :

- Entraîner les word embeddings avec gensim en utilisant le corpus Brown. Nous allons utiliser la librairie Python *gensim* (Installation : `https://pypi.org/project/gensim/`).
- Chargez le pretrained model et tester des similarités entre vecteurs (comme vu en cours)

1. Entrainer un word embedding avec le Corpus Brown :

```
import gensim
from nltk.corpus import brown
model =gensim.models.Word2Vec(brown.sents())
```

Cela peut prendre un certain temps. Ainsi, une fois entreiné, il peut être sauvegardé comme suit :

```
model.save('brown.embedding')
new_model = gensim.models.Word2Vec.load('brown.embedding')
```

Le modèle sera la liste des mots avec leur embedding. Nous pouvons facilement obtenir la représentation vectorielle d'un mot.

```
from len(new_model['university'])
```

Certaines fonctions déjà implémentées dans Gensim permettent de manipuler des word embeddings. Par exemple, pour calculer la similarité de cosinus entre 2 mots :

```
new_model.similarity('university','school') > 0.3
```

2. Utiliser le modèle pre-trained

NLTK inclut un modèle pre-trained qui fait partie du modèle entrainé sur 100 milliards de mots du jeu de données Google News. Le modèle complet est disponible sur `https://code.google.com/p/word2vec/` (environ 3 Go). Ce modèle n'inclue que les mots les plus courants (~ 44 000 mots). Each word is represented in the space of 300 dimensions.

```
from nltk.data import find
word2vec_sample =str(find('models/word2vec_sample/pruned.word2vec.txt'))
model = gensim.models.KeyedVectors.load_word2vec_format(word2vec_sample,binary=False)
```

Trouver les n premiers mots qui sont similaires à un mot cible est simple. Le résultat est la liste de n mots avec le score.

```
model.most_similar(positive=['university'], topn = 3)
```

Testez-le avec quelques mots !

Comme montré en cours, les word embeddings capturent une grande partie des régularités syntaxiques et sémantiques. Par exemple, le vecteur "King - Man + Woman" est proche de "Queen" et "Germany - Berlin + Paris" est proche de "France".

```
model.most_similar(positive=['woman','king'], negative=['man'], topn = 1)
model.most_similar(positive=['Paris','Germany'], negative=['Berlin'],topn = 1)
```

Testez-le avec les exemples vu en classe (et/ou des autres exemple) !

3. Visualisation des embeddings.

On peut visualizer les word embeddings en utilisant t-SNE (`http://lvdmaaten.github.io/tsne/`). On souhaite visualiser les premiers 50 mots les plus proches d'un mot cible. Nous avons une liste des labels associées aux vecteurs et une matrice nm (n=longueur du vecteur= K_nearest, m=dimensions=300) Vous devez peupler le tableau avec les premiers 50 mots les plus proches d'un mot cible et le tracer sur un graphe ("plot it"), en utilisant les fonctions suivantes pour la reduction des dimensions des vecteurs.

```
import numpy as np
labels = []
count = 0
max_count = 1000
X = np.zeros(shape=(max_count,len(model[‘university’])))

for term in model.index_to_key:
X[count] = model[term]
labels.append(term)
count+= 1
if count >= max_count: break

# It is recommended to use PCA first to reduce to ~50 dimensions
from sklearn.decomposition import PCA
pca = PCA(n_components=50)
X_50 = pca.fit_transform(X)

# Using TSNE to further reduce to 2 dimensions
from sklearn.manifold import TSNE
model_tsne = TSNE(n_components=2, random_state=0)
Y = model_tsne.fit_transform(X_50)

# Show the scatter plot
import matplotlib.pyplot as plt
plt.scatter(Y[:,0], Y[:,1], 20)

# Add labels
for label, x, y in zip(labels, Y[:, 0], Y[:, 1]):
```

```
plt.annotate(label, xy = (x,y), xytext = (0, 0), textcoords = 'offset points', size = 10)

plt.show()
```

==========

Partie 3. Classification des textes : analyse de sentiments

Dans cette troisième partie du TP, nous allons développer quelques modèles pour l'analyse de sentiments (classement de textes). Le dataset utilisé est un dataset d'analyse de sentiments dans le contexte de la finance (`https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis`). Nous allons utiliser **pandas**, **scikit-learn** et **nltk** [1]

```
import nltk
import numpy  as np
import pandas as pd
from sklearn.metrics                 import classification_report
from sklearn.preprocessing           import LabelBinarizer
from sklearn.neural_network          import MLPClassifier
from sklearn.model_selection         import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
import re
token_pattern = re.compile(r'(?u)\b\w\w+\b')
tokenizer = token_pattern.findall
```

1. Préparation de données

- Importer le dataset CSV fourni en utilisant **pandas** et afficher les 10 premiers échantillons.
- Encoder les sentiments en utilisant **LabelBinarizer** pour avoir un encodage OneHot.
- Diviser le dataset en entrainement et test, en utilisant **train_test_split**, la taille du test est 30% et random_state=0.
- Entrainer un modèle de vectorisation **TF** sur le texte d'entrainement et vectoriser le.
- En utilisant le même modèle, vectoriser le dataset de test.
- Définir une fonction **tokenstem** qui prend un texte et qui utilise **tokenizer** pour avoir une liste des tokens ensuite elle génère une liste des tokens radicalisés en utilisant **nltk.stem.porter.PorterStemmer** et elle ne considère pas les tokens appartenant à **nltk.corpus.stopwords.words('english')**
- Refaire les deux étapes précédentes, mais en limitant la taille du vecteur **max_features = 3000** et en utilisant **tokenstem** comme **tokenizer**.

2. Entrainement

- Créer un modèle de type linear support vector machine (SVM)`https://scikit-learn.org/stable/modules/svm.html#svm`.
- Entrainer le modèle sur les représentations vectorielles obtenues à l'étape precendente.
- Prédire les classes en appliquant le modèle sur les représentations d'entrainement.
- Afficher le rapport de classification.

3. Test

- Prédire les classes en appliquant le modèle sur les représentations vectorielles de test.
- Afficher le rapport de classification.

[1] Remerciements : ARIES Abdelkrime