

```

1  #!/usr/bin/env python
2  # coding: utf-8
3  # Auteure: Eva Rolin
4  # Date: Novembre 2019
5  # Ce code a été créé par Paeztold (2015). Nous pouvons le retrouver au lien suivant:
6  # https://github.com/ghpaetzold/LEXenstein
7  # Nous avons supprimé certaines lignes qui nous semblaient inadéquates et nous
8  # avons également ajouté certaines fonctions pour que le code s'adapte à notre librairie
9  #*****
10 # Importation des bibliothèques (gensim, spacy, nltk) et ressources dont nous avons besoin.
11 # Nous avons une fois de plus utilisé le modèle préentraîné proposé sur le site officiel de
FastText:
12 # https://fasttext.cc/docs/en/crawl-vectors.html
13 #*****
14 import gensim
15 model =
gensim.models.KeyedVectors.load_word2vec_format('C:/Users/evaro/OneDrive/Documents/Github/LEXenstein/
EVAstein/cc.fr.300.vec/cc.fr.300.vec')
16 import numpy as np
17 import spacy
18 nlp = spacy.load("fr_core_news_sm")
19 import nltk
20 from nltk.tokenize import word_tokenize
21 from nltk.corpus import stopwords
22 stopWords = set(stopwords.words('french'))
23 from scipy.spatial.distance import cosine
24 #*****
25 # Suppression des éléments tels que des virgules et des parenthèses dans la phrase
26 #*****
27 def clean_phrase(phrase):
28     exclude = [",", "(", ")"]
29     phrase = ''.join(ch for ch in phrase if ch not in exclude)
30     return phrase
31 #*****
32 # Nous avons réutilisé la fonction à partir de la classe WordVectorSelector
33 # Nous avons remplacé le tokenizer initial par un autre plus approprié au français
34 # Cette fonction permet de calculer le vecteur d'une phrase ( et donc prend en compte le
contexte
35 # du mot complexe)
36 #*****
37 def get_vector_sentence(phrase, head, window):
38     valid_tokens = []
39     valid = 0
40     doc = nlp(phrase)
41     if head > 0:
42         for i in range(max(0, head-window), head):
43             for j in range(len(doc)):
44                 if doc[j] not in stopWords:
45                     valid = doc[j]
46                     valid_tokens.append(valid)
47     if head < len(doc)-1:
48         for i in range(head+1, min(len(doc), head+1+window)):
49             for j in range(len(doc)):
50                 if doc[j] not in stopWords:
51                     valid = doc[j]
52                     valid_tokens.append(valid)
53     result = np.array([])
54     valid_ok = set(valid_tokens)
55     for token in valid_ok:
56         tok = str(token)
57         if len(result) == 0:
58             try:
59                 result = model[tok]
60             except Exception:
61                 pass
62     else:

```

```

63         try:
64             result = np.add(result, model[tok])
65         except Exception:
66             pass
67     result = result/float(len(valid_ok))
68     return result
69 *****
70 # Cette fonction provient également de la classe WordVectorSelector de Paetzold.
71 # Nous l'avons reprise telle quelle. Cette fonction permet de renvoyer le vecteur
72 # de chaque candidat à la substitution.
73 *****
74 def get_vector_word(candidate):
75     result = np.array([])
76     try:
77         result = model[candidate]
78     except:
79         pass
80     return result
81 *****
82 # Cette fonction est le résultat d'une adaptation de la fonction select word de Paetzold.
83 # Au lieu de laisser la possibilité à l'utilisateur le nombre de mots qu'il veut garder dans
sa
84 # sélection, nous avons choisi de sélectionner uniquement les mots dont la similarité est
supérieure
85 # au seuil de 0.75.
86 # Cette fonction retourne un dictionnaire avec pour clé le mot cible et pour valeur, le
candidat
87 # dont la valeur de similarité est supérieure au seuil de 0.75.
88 *****
89 def select_word(candidate_dists,target):
90     liste = []
91     new = {}
92     for key,value in candidate_dists.items():
93         if value > 0.75:
94             if target not in new:
95                 new[target] = []
96                 new[target].append(key)
97             if value < 0.75:
98                 if target not in new:
99                     new[target] = []
100                     new[target].append(target)
101     return new
102 *****
103 # Cette fonction permet de retourner la liste de candidats de chaque mot complexe
104 *****
105 def dico(dic,target):
106     for key,val in dic.items():
107         if key == target:
108             return val
109 *****
110 # Cette fonction permet de convertir notre liste en chaines de caractères afin d'imprimer
111 # nos candidats dans le format adéquat. Nous sélectionnes tous les mots différents du candidat
112 # complexe et du mot cible
113 *****
114 def conversion_string(liste,target,lemma):
115     st = ""
116     try:
117         for elt in liste:
118             if elt != target:
119                 if elt != lemma:
120                     st += elt+" "
121     return st
122 except:
123     return st
124 *****
125 # Fonction de lemmatisation
126 *****

```

```

127 def lemmatize_fr(target,phrase,ident):
128     liste = []
129     doc = nlp(phrase)
130     for word in doc:
131         liste.append(word.lemma_)
132     return liste[ident]
133 *****
134 # Fonction principale prenant en argument un fichier au format victor et donne en sortie
135 # un fichier dont le format pour chaque ligne est le suivant:
136 # phrase mot indice sélection1 sélection2 sélection3
137 *****
138 def get_subst(victor_corpus,victor_corpus_bis):
139     with open (victor_corpus, "r",encoding="utf-8") as file:
140         with open(victor_corpus_bis,"w",encoding="utf-8") as file2:
141             texte = file.readline()
142             while texte:
143                 texte_cut = texte.strip().split("\t")
144                 phrase = texte_cut[0]
145                 target = texte_cut[1]
146                 index = int(texte_cut[2])
147                 try:
148                     candidat_cut = texte_cut[3]
149                 except:
150                     candidat_cut = ""
151                 phrase_clean = clean_phrase(phrase)
152                 lemma = lemmatize_fr(target,phrase_clean,index)
153                 cdt = candidat_cut.split()
154                 vec1 = get_vector_sentence(phrase_clean,index,2)
155                 candidate_dists = {}
156                 new_candidate = {}
157                 for c in cdt:
158                     vec2 = get_vector_word(c)
159                     try:
160                         candidate_dists[c] = cosine(vec2, vec1)
161                     except:
162                         candidate_dists = candidate_dists
163                 final_candidats = select_word(candidate_dists,target)
164                 result= dico(final_candidats,target)
165                 string = conversion_string(result,target,lemma)
166                 file2.write("{}\t{}\t{}\t{}".format(phrase,target,index,string))
167                 file2.write("\n")
168                 texte = file.readline()
169
170

```