Master Informatique, parcours MALIA

Carnets de note Python pour le cours de Network Analysis for Information Retrieval

Julien Velcin, laboratoire ERIC, Université Lyon 2

Représentation des documents (partie 2)

Différentes solutions existent pour représente un document dans un espace vectoriel :

- espace des mots (avec différents types de pondération : TF, TFxIDF, OKAPI)
- espace sémantique de faible dimension :
  - **approche de plongement (*sentence/document embedding*)**
  - approche thématique

On va déployer deux méthodes simples basées sur des plongements qui ne prennent pas en compte l'ordre des mots :

- solution naïve : centre d'inertie des vecteurs codant le sens des mots pris indépendamment
- méthode Doc2Vec

D'autres méthodes ont été proposés récemment mais elles se basent souvent sur des approches plus complexes voire coûteuses (comme le Transformer). Des éléments sont données à la fin de ce carnet.

Avant de voir comment construire ces représentations, voyons comment utiliser des plongements de mots pré-appris avec la librairie *spacy*.

In [1]:
```
import numpy as np
```

In [2]:
```
import spacy

# Il faut avoir installé la ressource en local en version "large" (d'où le "lg" à l
#python -m spacy download fr_core_news_lg

nlp = spacy.load('fr_core_news_lg')
#nlp = spacy.load('en_core_web_sm')
```

La première action que nous pouvons faire est d'interroger la librairie sur les mots les plus proches d'une requête :

In [3]:
```
def find_close_words(word):
    sim_words = nlp.vocab.vectors.most_similar(
        np.asarray([nlp.vocab.vectors[nlp.vocab.strings[word]]]), n=10)
```

```
    return [nlp.vocab.strings[w] for w in sim_words[0][0]]

find_close_words("roi")

# attention, nlp.vocab.strings fonctionne dans les deux sens :
#  - retourne l'identifiant (unique) correspondant à un mot
#  - retourne le mot correspondant à un identifiant
```

Out[3]:
```
['roi',
 'Roi',
 'prince',
 'monarque',
 'empereur',
 'régent',
 'Empereur',
 'suzerain',
 'coempereur',
 'l\x92empereur']
```

On peut également calculer des similarités entre mots.

In [4]:
```
print("entre roi et reine : {}".format(nlp("roi").similarity(nlp("reine"))))
print("entre roi et trône : {}".format(nlp("roi").similarity(nlp("trône"))))
print("entre roi et oiseau : {}".format(nlp("roi").similarity(nlp("oiseau"))))
```

```
entre roi et reine : 0.628108097894871
entre roi et trône : 0.6489303722402299
entre roi et oiseau : 0.10113929594015392
```

Les vecteurs qui représentent les mots sont directement accessibles si besoin.

In [5]:
```
nlp("roi").vector
```

Out[5]:
```
array([ 5.6351e+00, -4.7504e+00,  2.9866e+00, -6.5048e-02,  5.6314e+00,
       -2.6488e+00,  1.6728e+00,  3.6799e+00,  2.2033e+00, -2.3401e-01,
        1.7170e+00, -3.0931e+00,  3.8267e+00,  1.8726e-01, -2.2761e+00,
       -2.1582e+00,  3.8331e+00, -1.1703e-01,  1.2575e+00,  3.2968e+00,
       -1.1991e+00,  5.0724e-01, -2.6204e+00,  2.1288e+00,  9.9307e-01,
       -3.1832e+00, -2.3930e+00,  1.3869e+00, -1.7312e+00,  4.5163e+00,
        9.9608e-01, -3.1276e+00, -1.5539e+00,  1.0525e+00, -5.3267e-01,
       -5.5565e+00,  8.4429e-01,  4.9819e+00, -4.6006e-01,  2.2781e+00,
       -1.0639e+00,  1.2235e+00, -4.5606e+00,  3.4419e-01,  1.9167e+00,
       -1.5078e+00,  3.6443e+00,  2.8392e+00,  4.9837e-01, -5.5096e-02,
       -7.9940e+00, -1.9028e+00, -3.9536e+00,  3.7232e-03,  2.3186e+00,
        5.6297e+00,  2.2466e+00,  5.2483e-01,  1.2710e+00, -5.8875e-01,
       -2.2971e+00,  1.4909e-01,  2.7780e-03,  3.9281e+00,  6.5166e-01,
        8.9464e-02,  3.2467e+00,  9.5456e-01, -2.4231e+00, -4.3828e+00,
        1.2746e+00,  1.8955e+00, -2.9509e+00, -5.8867e+00,  3.0693e+00,
       -8.0891e-01,  6.9400e-01,  3.8128e+00,  3.5806e+00,  4.2890e+00,
       -2.7194e+00,  2.5940e+00,  4.6594e+00,  6.6699e-02, -4.7390e+00,
       -1.4998e-01, -9.3746e-01,  4.6751e+00, -1.4972e+00, -1.0867e+01,
        4.9722e+00, -3.9660e-01,  9.2787e-01,  4.9520e-01,  2.4602e+00,
       -1.0434e+00,  3.0316e+00, -1.7612e-02,  4.9520e+00, -4.8073e+00,
        3.8922e-01,  1.0257e+00,  1.0831e-01,  4.0534e+00, -2.1833e+00,
        1.5790e+00, -3.2116e+00,  2.6927e+00, -3.8434e-01, -4.7468e+00,
        7.2697e-01, -2.1205e+00,  5.9403e-01,  4.7841e+00, -3.8717e+00,
       -3.0401e+00, -1.2210e+00, -2.8635e+00, -6.9869e-01,  3.2319e+00,
        5.7720e+00,  5.9429e+00, -4.3025e+00, -4.9565e-01, -2.3339e+00,
       -1.5692e+00, -6.8929e-01, -1.1271e+00, -1.3472e+00,  2.0474e+00,
        4.7208e+00, -4.9401e-01,  1.2751e+00, -3.2224e+00, -1.0313e+00,
       -1.7439e+00,  8.7939e-01,  1.2688e-01, -4.7297e+00, -1.5638e+00,
       -2.3903e+00,  9.2193e-01,  1.1045e+00, -2.1525e+00,  9.6714e-01,
        1.3308e+00, -5.6572e+00, -2.5758e+00,  3.4210e+00,  1.8027e+00,
        6.2324e+00,  5.7700e+00,  2.3663e+00,  1.0878e+00, -1.3365e+00,
        1.1781e+00,  2.7440e+00,  2.3066e+00,  8.4879e-01,  2.2804e+00,
        1.5781e+00, -2.4117e+00, -1.3357e+00,  5.1102e+00, -8.7586e-01,
       -2.9436e+00, -1.9356e+00,  7.7401e+00,  5.9926e+00,  2.3883e+00,
       -3.1656e+00,  6.4872e+00, -2.2768e+00,  1.4991e+00, -3.8998e+00,
        5.2592e-01, -3.8706e-01, -9.6823e-01,  1.4808e+00, -3.9378e+00,
       -2.0357e+00, -4.4986e+00,  8.9750e+00,  2.2045e+00,  1.8716e-03,
        3.2258e+00,  4.7617e+00, -1.8651e+00, -6.1514e+00,  2.5549e+00,
       -1.6786e+00,  1.9050e+00,  3.4051e+00, -4.6565e+00,  4.1841e+00,
       -7.8164e+00,  1.6226e+00, -2.7039e+00, -5.7035e-01, -1.3518e+00,
       -8.5385e-01, -1.2273e+00,  1.2349e+00, -7.8515e-01,  5.0479e-01,
        2.8719e+00,  8.9905e-01,  4.9693e+00, -2.0078e+00,  4.3517e+00,
       -1.6480e+00, -5.0611e+00,  6.5107e+00, -3.6784e+00,  3.1451e+00,
       -2.8814e+00, -3.1464e+00, -1.5150e-01,  3.1440e+00, -3.6309e+00,
       -2.3865e+00,  4.2842e+00,  2.9188e+00, -2.8959e+00,  8.5420e-01,
        6.1502e+00,  3.0970e+00, -6.2933e-02, -7.2427e-01, -4.6472e+00,
       -5.1277e-01,  3.6802e+00,  4.1581e+00,  1.6980e+00,  3.4822e+00,
        1.8467e+00, -2.8244e+00, -8.6421e+00,  1.9419e+00, -2.4484e-01,
        2.5395e+00, -2.6593e+00, -2.9772e+00, -1.7609e+00, -1.1823e+00,
        6.7310e+00,  2.8250e+00, -2.4596e-01, -4.1260e+00,  4.5138e+00,
       -5.3808e+00, -2.3224e+00,  1.9299e+00,  1.6637e+00,  5.3128e+00,
       -2.1047e+00, -1.5359e+00, -2.2614e-01, -8.1183e+00,  3.4033e-01,
        1.6500e+00, -2.3227e+00,  4.0308e+00,  3.5514e+00,  6.6944e-01,
        1.4120e+00, -8.0441e-01, -6.5361e-01, -5.6505e+00,  2.4682e+00,
       -9.1946e+00, -6.3852e+00,  8.2396e+00, -2.7895e+00, -2.3564e+00,
        8.4714e+00, -2.6244e-01,  4.2987e+00,  1.5561e+00,  2.4657e+00,
```

```
       -6.1948e+00, -6.6181e-01, -9.9525e-01,  4.2478e+00, -6.1932e+00,
       -1.9624e+00, -7.0235e+00,  3.3948e-01, -7.3027e-01, -4.2027e+00,
        1.5683e+00, -6.5984e+00, -2.5014e+00,  1.3260e+00, -3.1824e+00,
       -5.0466e+00, -5.4672e+00, -4.6966e+00, -5.3983e+00,  1.5300e+00],
      dtype=float32)
```

On peut même résoudre des problèmes d'analogie. Par ex., qu'est-ce qui est à "femme" ce que "homme" est à "roi" ? Ou la relation "capitale de".

In [6]:
```
def close_words_from_vector(vec):
    ms = nlp.vocab.vectors.most_similar(np.array([vec]), n=10)
    return [nlp.vocab.strings[w] for w in ms[0][0]]
```

In [7]:
```
analogie = nlp("roi").vector-nlp("homme").vector+nlp("femme").vector
close_words_from_vector(analogie)
```

Out[7]:
```
['roi',
 'reine',
 'Roi',
 'prince',
 'régent',
 'duc',
 'princesse',
 'monarque',
 'suzerain',
 'coempereur']
```

In [8]:
```
analogie = nlp("France").vector-nlp("Paris").vector+nlp("Berlin").vector
close_words_from_vector(analogie)
```

Out[8]:
```
['Allemagne',
 'l´Allemagne',
 'ex-Allemagne',
 'lAllemagne',
 'Allemagne-',
 'Europe',
 'Grande-Bretagne',
 'ouest-allemande',
 'est-allemande',
 'Allemagnes']
```

Approche naïve

Nous allons calculer des représentations vectorielles de documents comme le centre d'inertie des mots qui le composent.

In [9]:
```
import os

with open(os.path.join("datasets", "Frank Herbert - Dune.txt")) as f:
    lines = [line.strip() for line in f.readlines()]
```

In [11]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

```
tf_vectorizer = CountVectorizer(stop_words="english", max_df=0.5, min_df=3, max_fea
tf_vectorizer.fit(lines)
D = tf_vectorizer.transform(lines)

features = tf_vectorizer.get_feature_names_out()
```

On récupère la taille des plongements car elle constitue la dimension de l'espace dans lequel on va plonger les documents.

In [12]:
```
dim = len(nlp.vocab.vectors[nlp.vocab.strings["dune"]])
```

In [13]:
```
ndocs, nwords = D.shape
print(ndocs)
```

```
8608
```

On définit une fonction qui calcule le centre d'inertie d'un ensemble de vecteurs mots.

In [14]:
```
import numpy as np

def centre(d):
    m = np.zeros(shape=(1,dim))
    nbw = 0
    for w in d:
        try:
            v = nlp.vocab.vectors[nlp.vocab.strings[str(w)]]
            m = np.append(m, v.reshape((1,dim)), axis=0)
            nbw += 1
        except:
            pass
    seuil = True
    if nbw>0:
        return (nbw, np.sum(m, axis=0)/nbw) # la normalisation est inutile si on ut
    else:
        return (0, m)
```

Puis on calcule la représentation pour chaque document du corpus. On en profite pour sauvegarder une liste avec la taille des documents (ici, le nombre de mots ayant un vecteur associé dans le plongement).

In [15]:
```
nbw_docs = []
i = 0
doc_vec = np.zeros(shape=(ndocs,dim))
id_docs_nonvides = []
for d in tf_vectorizer.inverse_transform(D):
    nbw, r = centre(d)
    doc_vec[i] = r.reshape((1,dim))
    nbw_docs.append(nbw)
    i += 1
```

In [16]:
```
len(doc_vec[0])
#doc_vec[10]
```

Out[16]: 300

In [17]:
```
nbw_docs[0:10]
```

Out[17]: [1, 0, 0, 0, 0, 1, 1, 0, 0, 0]

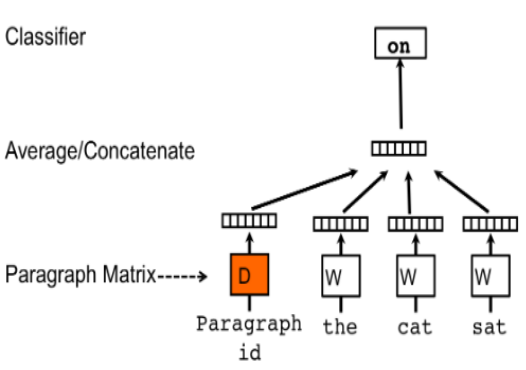
Il ne reste plus qu'à sauvegarder l'information pour une utilisation future.

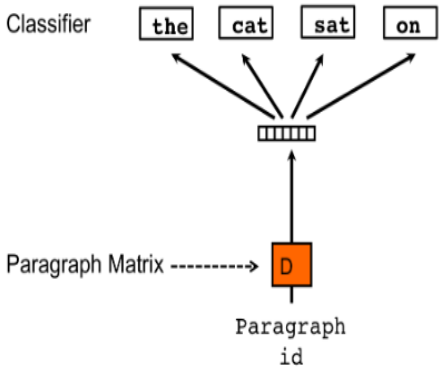
In [18]:
```
col_p = np.array(nbw_docs).reshape(ndocs,1)
col_ids = np.arange(1, ndocs+1).reshape(ndocs,1)
data_to_save = np.hstack([doc_vec, col_p, col_ids])
np.savetxt('vec_doc_naive.csv', data_to_save, delimiter='\t')
```

Dov2Vec

Doc2Vec est une extension des approches Word2Vec dans lesquelles on ajoute un "token" associé à chaque document (ici, un paragraphe). Il existe deux versions de cet algorithme (Le and Mikolov, 2014) :

- PV-DM : Distributed Memory Models of Paragraph Vectors
- PV-DBOW : Distributed Bag of Words version of Paragraph Vector





Il faut formatter les données pour pouvoir les données en entrée de l'algorithme Doc2Vec.

In [19]:
```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

# on rajoute une taille minimale dès à présent
min_docs = 4

tagged_docs = []
nbw_docs = []
for i, list_tokens in enumerate(tf_vectorizer.inverse_transform(D)):
    nbw = len(list_tokens)
    nbw_docs.append(nbw)
    if nbw > min_docs:
        tagged_docs.append(TaggedDocument(words=list_tokens, tags=[str(i+1)]))
```

In [20]:
```
len(tagged_docs)
tagged_docs[0:4]
```

Out[20]:
```
[TaggedDocument(words=array(['arrakis', 'begin', 'beginning', 'bene', 'born', 'c
adan', 'care',
        'dib', 'dune', 'emperor', 'fact', 'gesserit', 'known', 'knows',
        'life', 'lived', 'muad', 'padishah', 'place', 'planet', 'special',
        'study', 'taking', 'time', 'year', 'years'], dtype='<U13'), tags=['11']),
 TaggedDocument(words=array(['arrakis', 'boy', 'came', 'mother', 'old', 'paul', 'r
eached'],
       dtype='<U13'), tags=['14']),
 TaggedDocument(words=array(['ancient', 'atreides', 'caladan', 'change', 'family',
'feeling',
        'home', 'night', 'stone', 'weather'], dtype='<U13'), tags=['15']),
 TaggedDocument(words=array(['allowed', 'bed', 'door', 'lay', 'let', 'moment', 'ol
d', 'passage',
        'paul', 'room', 'woman'], dtype='<U13'), tags=['16'])]
```

In [21]:
```
dim_d2v = 10

model_doc2vec = Doc2Vec(tagged_docs, vector_size=dim_d2v, window = 3)
model_doc2vec.train(tagged_docs, total_examples = len(tagged_docs), epochs = 1000)
```

In [22]:
```
set1 = set(features)
set2 = set(model_doc2vec.wv.index_to_key)
set1.difference(set2)
```

Out[22]:
```
{'10', 'soo'}
```

In [23]:
```
model_doc2vec.dv
```

Out[23]:
```
<gensim.models.keyedvectors.KeyedVectors at 0x3022a2e90>
```

In [24]:
```
from nltk.tokenize import word_tokenize

test_doc = word_tokenize("Dune, the spice planet".lower())
test_doc_vector = model_doc2vec.infer_vector(test_doc)
res = model_doc2vec.dv.most_similar(positive = [test_doc_vector])
print(res)
```

```
[('6730', 0.9606164693832397), ('1374', 0.9565431475639343), ('1817', 0.948729515
6836), ('8355', 0.9132331013679504), ('4378', 0.9036451578140259), ('6204', 0.899497
9858398438), ('8556', 0.8993549346923828), ('5404', 0.8964237570762634), ('5758', 0.
8955005407333374), ('1469', 0.8944513201713562)]
```

In [25]:
```
for i, s in res:
    ind_doc = int(i)
    print("%s (%s): %s" % (i, s, lines[ind_doc-1]))
```

```
6730 (0.9606164693832397): "So it seemed," Paul said. "But this is deep into the d
ert, for smugglers."
1374 (0.9565431475639343): "These 'thopters are fairly conventional," Hawat said. "M
ajor modifications give them extended range. Extra care has been used in sealing ess
ential areas against sand and dust. Only about one in thirty is shielded--possibly d
iscarding the shield generator's weight for greater range."
1817 (0.9487295150756836): "Not from the deep desert," Kynes said. "Men have walked
out of the second zone several times. They've survived by crossing the rock areas wh
ere worms seldom go."
8355 (0.9132331013679504): DUNE MEN: idiomatic for open sand workers, spice hunters
and the like on Arrakis. Sandworkers. Spiceworkers.
4378 (0.9036451578140259): "I planted the thumper in the deepest part of the crevass
e," Paul said. "Whenever I light its candle it'll give us about thirty minutes."
6204 (0.8994979858398438): "We both know the figure for storm accretion," Hawat sai
d.
8556 (0.8993549346923828): SPICE DRIVER: any Dune man who controls and directs movab
le machinery on the desert surface of Arrakis.
5404 (0.8964237570762634): "You must admit it'd be a way to develop a substantial wo
rk force on Arrakis--use the place as a prison planet."
5758 (0.8955005407333374): He had seen a thing about the caverns and this room, a th
ing that suggested far greater differences than anything he had yet encountered.
1469 (0.8944513201713562): "Not for a briefing. It's said among the Fremen that ther
e were more than two hundred of these advance bases built here on Arrakis during the
Desert Botanical Testing Station period. All supposedly have been abandoned, but the
re are reports they were sealed off before being abandoned."
```

In [26]:
```
print(len(model_doc2vec.dv))
type(model_doc2vec.dv)
```

```
5007
```

Out[26]: gensim.models.keyedvectors.KeyedVectors

In [27]:
```
#set_tags = list(model_doc2vec.docvecs.doctags)
set_tags = list([t.tags[0] for t in tagged_docs])
nb_docs_small = len(set_tags)
print(nb_docs_small)
```

```
5007
```

On récupère le tableau des plongements pour le sauvegarder.

In [28]:
```
doc_vec_doc2vec = np.zeros(shape=(nb_docs_small, dim_d2v))

i = 0
for t in set_tags:
    doc_vec_doc2vec[i] = model_doc2vec.dv[t]
    i += 1

doc_vec_doc2vec.shape
```

Out[28]: (5007, 10)

In [29]:
```
doc_ids_small = [int(t) for t in set_tags]
nbw_docs_small = [nbw_docs[i-1] for i in doc_ids_small]

col_p = np.array(nbw_docs_small).reshape(nb_docs_small,1)
```

```
col_ids = np.array(doc_ids_small).reshape(nb_docs_small,1)
data_to_save = np.hstack([doc_vec_doc2vec, col_p, col_ids])
np.savetxt('vec_doc_doc2vec.csv', data_to_save, delimiter='\t')
```

De nombreuses autres méthodes existent pour construire des représentations de documents, par exemple :

- InferSent (EMNLP 2017)
- Universal Sentence Encoder (EMNLP 2018)
- SentenceBERT (EMNLP 2019)

N'hésitez pas à consulter la page suivante qui décrit ses approches et comment les implémenter :

https://www.analyticsvidhya.com/blog/2020/08/top-4-sentence-embedding-techniques-using-python/

Depuis l'arrivée des LLMs, de nombreux services d'*embeddings* sont devenus disponibles afin de mieux capturer la sémantique des textes (phrases, paragraphes, documents). Il est difficile de rester parfaitement à jour des toutes dernières méthodes proposées, n'hésitez pas à aller voir ce qui est proposé sur des plateformes comme HuggingFace. Vous pouvez aussi consulter la page suivante.

Clustering de documents

L'objectif est de vous montrer comment utiliser un algorithme simple de clustering (ici, k-means). Bien sûr, l'intérêt d'utiliser un espace vectoriel est de pouvoir utiliser de nombreux autres algorithmes, comme des modèles de mélange, etc.

In [30]:
```
doc_vec.shape
```

Out[30]:
```
(8608, 300)
```

In [31]:
```
from sklearn.cluster import KMeans

k = 10

km_10 = KMeans(n_clusters=k, random_state=0).fit(doc_vec)
```

```
/Users/jvelcin/miniforge3/envs/cours23/lib/python3.11/site-packages/sklearn/cluste
_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

In [32]:
```
import pandas
pandas.Series(km_10.labels_).value_counts()
```

Out[32]:
```
8    2463
4    2152
7     918
3     892
5     769
6     398
1     364
2     288
9     276
0      88
Name: count, dtype: int64
```

In [33]:
```
ndocs
```

Out[33]:
```
8608
```

In [34]:
```
col_p = np.array(nbw_docs).reshape(ndocs,1)
col_ids = np.arange(1, ndocs+1).reshape(ndocs,1)
clu_lab = np.array(km_10.labels_).reshape(ndocs,1)
data_to_save = np.hstack([doc_vec, col_p, col_ids, clu_lab])
np.savetxt('vec_doc_naive_cl10.csv', data_to_save, delimiter='\t')
```

Idem avec la représentation obtenue à l'aide de doc2vec.

In [35]:
```
km_10_doc2vec = KMeans(n_clusters=10, random_state=0).fit(doc_vec_doc2vec)
```

```
/Users/jvelcin/miniforge3/envs/cours23/lib/python3.11/site-packages/sklearn/clust
_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

In [36]:
```
doc_ids_small = [int(t) for t in set_tags]
nbw_docs_small = [nbw_docs[i-1] for i in doc_ids_small]
clu_lab_small = np.array(km_10_doc2vec.labels_).reshape(nb_docs_small,1)

col_p = np.array(nbw_docs_small).reshape(nb_docs_small,1)
col_ids = np.array(doc_ids_small).reshape(nb_docs_small,1)
data_to_save = np.hstack([doc_vec_doc2vec, col_p, col_ids, clu_lab_small])
np.savetxt('vec_doc_doc2vec_clu10.csv', data_to_save, delimiter='\t')
```

Références :

- Le, Quoc, and Tomas Mikolov. Distributed representations of sentences and documents. International Conference on Machine Learning (ICML), 2014.
- Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, Antoine Bordes. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data, EMNLP 2017.
- Daniel Cera, Yinfei Yanga, Sheng-yi Konga, Nan Huaa, Nicole Limtiacob, Rhomni St. Johna, Noah Constanta, Mario Guajardo-Ce śpedesa, Steve Yuanc, Chris Tara, Yun-Hsuan Sunga, Brian Stropea. Universal Sentence Encode for Englishr, EMNLP 2018.
- Nils Reimers, Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, EMNLP 2019.