Etape 4 : Construction d'un modèle en passant par des résumés de section

La démarche ici est la suivante :

- pour chaque section de brevet on réalise un résumé à base des 5 phrases les plus importantes de la section
- on réalise l'embedding de la section par mean pooling des embeddings des phrases résumant la section
- on réalise l'embedding du brevet par mean pooling des embeddings des résumés de section

In [1]:
```
import json
import matplotlib.pyplot as plt
from tqdm import tqdm
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import pickle

from sentence_transformers import SentenceTransformer
from sentence_transformers.util import cos_sim
from sentence_transformers.quantization import quantize_embeddings
from sentence_transformers import losses
from sentence_transformers.readers import InputExample
from torch.utils.data import DataLoader
from transformers import AutoTokenizer

import nltk
import numpy as np
from LexRank import degree_centrality_scores
```

Création des embeddings après avoir appliqué des résumés de chacune des sections

Le but ici est d'utiliser un LLM pour créer des résumés de chaque section, puis de créer les embeddings des résumés

In [2]:
```
with open('../data/dataset_patent_sections.json', 'r') as outfile:
    dataset_patent_section = json.load(outfile)
    outfile.close()
```

In [21]:
```
model_name = 'intfloat/e5-small-v2'
model = SentenceTransformer(model_name)
```

In [84]:
```
# Construction du dataset de brevets resumes par section
```

```python
def summarize_vanilla(text, model, max_sentences=5):
    '''
    Fonction pour fournir le resume d'un texte sur la base des ses phrases l
    text -- str, le texte a resumer
    model -- SentenceTransformer, model de calcul des embeddings
    max_sentences -- int, le nombre de phrases max dans le resume
    '''
    sentences = nltk.sent_tokenize(text)
    embeddings = model.encode(sentences)
    similarity_scores = cos_sim(embeddings, embeddings).numpy()
    centrality_scores = degree_centrality_scores(similarity_scores, threshol
    most_central_sentence_indices = np.argsort(-centrality_scores)
    nb_sentences_summary = min(5, len(sentences))
    list_sentences_summary = []
    list_embeddings_summary = []
    for idx in most_central_sentence_indices[0:nb_sentences_summary]:
        list_sentences_summary.append(sentences[idx].strip())
        list_embeddings_summary.append(embeddings[idx])
    summary_embedding = np.mean(list_embeddings_summary, axis=0)
    summary = ' '.join(list_sentences_summary)
    return summary, summary_embedding

dataset_patent_section_summary = {}
for i in tqdm(range(len(dataset_patent_section)), desc ="Construction du dat
    dict_patent = {}
    dict_patent['query'] = dataset_patent_section[str(i)]['query']
    for key in ['pos', 'negative']:
        list_sections = dataset_patent_section[str(i)][key]
        list_sections_summary = []
        for j in range(len(list_sections)):
            text = list_sections[j]['content']
            summary, summary_embedding = summarize_vanilla(text, model, max_
            list_sections_summary.append({'section': list_sections[j]['secti
                                          'content': summary,
                                          'embedding': summary_embedding})
        dict_patent[key] = list_sections_summary
    dataset_patent_section_summary[str(i)] = dict_patent
```

```
Construction du dataset de resumes: 100%|███████████████████████████████████
████████████████████████████████████████████████████████████████████████████
██████████████████| 499/499 [32:49<00:00,  3.95s/it]
```

In [100…

```python
# Construction des embeddings de brevet a partir des embeddings des resumes
dataset_patent_section_embeddings = {}

for i in tqdm(range(len(dataset_patent_section_summary)), desc ="Constructio
    dataset_patent_section_embeddings[str(i)] = {}
    for key in ['pos', 'negative']:
        list_sections_dict = dataset_patent_section_summary[str(i)][key]
        list_embeddings = [list_sections_dict[j]['embedding'] for j in range
        patent_embedding = np.mean([emb for emb in list_embeddings if (not i
        dataset_patent_section_embeddings[str(i)][key] = patent_embedding
```

```
Construction des embeddings de brevets: 100%|███████████████████████████████
████████████████████████████████████████████████████████████████████████████
███████████████| 499/499 [00:00<00:00, 16355.83it/s]
```

In [101…

```
# Ajout des embeddings de query
for i in tqdm(range(len(dataset_patent_section_embeddings)), desc ="Calcul d
    list_sentences = [dataset_patent_section[str(i)]['query']]
    embeddings = model.encode(list_sentences)
    query_embedding = np.mean(embeddings, axis=0)
    dataset_patent_section_embeddings[str(i)]['query'] = query_embedding
```

```
Calcul des embeddings des query: 100%|██████████████████████████████████████
█████████████████████████████████████████████████████████████████████████████
██████████████████| 499/499 [00:10<00:00, 47.96it/s]
```

In [102…

```
with open('../data/patent_sections_embeddings_summary_e5-small-v2.pickle', '
    pickle.dump(dataset_patent_section_embeddings, fh)
    fh.close()
```

In [3]:

```
with open('../data/patent_sections_embeddings_summary_e5-small-v2.pickle', '
    dataset_patent_section_embeddings = pickle.load(fh)
    fh.close()
```

In [4]:

```
# Performances -- classification
nb_good_embeddings = 0
for i in range(len(dataset_patent_section_embeddings)):
    embeddings = [
        dataset_patent_section_embeddings[str(i)]['query'],
        dataset_patent_section_embeddings[str(i)]['pos'],
        dataset_patent_section_embeddings[str(i)]['negative']
    ]
    similarities = cos_sim(embeddings[0], embeddings[1:])
    sim_pos, sim_neg = similarities.flatten()
    if sim_pos > sim_neg :
        nb_good_embeddings+=1
perc_good_embeddings = round(100*nb_good_embeddings/len(dataset_patent_secti
print('Embeddings de documents compatibles avec la query: {}, {} %'.format(r
                                                                             p
```

```
Embeddings de documents compatibles avec la query: 408, 81.76 %
```

In [5]:

```
# Performances -- top_K_accuracy
list_all_embeddings = []
for i in range(len(dataset_patent_section_embeddings)):
    emb_query = dataset_patent_section_embeddings[str(i)]['query']
    emb_pos = dataset_patent_section_embeddings[str(i)]['pos']
    emb_neg = dataset_patent_section_embeddings[str(i)]['negative']
    list_all_embeddings.append([emb_query, emb_pos, emb_neg])

def compute_top_K_accuracy_score(list_embeddings, K=5):
    '''
    Fonction pour calculer le top_K_accuracy score a partir d'une liste d'em
    [[emb_query, emb_pos, emb_neg]...]
    list_embeddings -- list, list des embeddings des query, positive, negati
    K -- int, le top K accuracy
    '''
    list_embeddings_query = [list_embeddings[i][0] for i in range(len(list_a
    list_embeddings_pos = [list_embeddings[i][1] for i in range(len(list_all
    list_embeddings_neg = [list_embeddings[i][2] for i in range(len(list_all
```

```python
    nb_pos = 0
    nb_neg = 0
    for idx in tqdm(range(len(list_embeddings_query)), desc ="Calcul du top_
        query = list_embeddings_query[idx]

        similarities_pos = cos_sim(query, list_embeddings_pos).flatten().tol
        similarities_pos = [('pos_{}'.format(i), similarities_pos[i]) for i

        similarities_neg = cos_sim(query, list_embeddings_neg).flatten().tol
        similarities_neg = [('neg_{}'.format(i), similarities_neg[i]) for i

        similarities = similarities_pos+similarities_neg
        similarities = sorted(similarities, key = lambda x: -x[1])
        top_K_ids = [similarities[i][0] for i in range(K)]

        if 'pos_{}'.format(idx) in top_K_ids:
            nb_pos+=1
        if 'neg_{}'.format(idx) in top_K_ids:
            nb_neg+=1
    acc_K_pos = nb_pos/len(list_embeddings)
    acc_K_neg = nb_neg/len(list_embeddings)
    return acc_K_pos, acc_K_neg

acc_K_pos, acc_K_neg = compute_top_K_accuracy_score(list_all_embeddings, K=5
print('acc_K_pos : {} (a maximiser)'.format(acc_K_pos))
print('acc_K_neg : {} (a minimiser)'.format(acc_K_neg))
```

```
Calcul du top_K_accuracy score: 100%|███████████████████████████████████████
████████████████████████████████████████████████████████████████████████████████
████████████████| 499/499 [00:44<00:00, 11.24it/s]
acc_K_pos : 0.8837675350701403 (a maximiser)
acc_K_neg : 0.5811623246492986 (a minimiser)
```

In [ ]: