Sameer_Sadruddin_310748_Ex8

June 28, 2022

1 Exercise Contextualized Embeddings

1.0.1 Load the dataset of movie lines which we used in the exercise last week

Extracting the movie lines dataset

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

[2]:
```
from zipfile import ZipFile
with ZipFile('movie_lines.tsv.zip', 'r') as zipObj:
    zipObj.extractall()
```

Extracting the movie lines and saving it into a list

[3]:
```
movie_lines = []
for line in open('movie_lines.tsv', encoding='utf8'):
    line = line.strip()
    while line[0] == '"' and line[-1] == '"':
        line = line[1:-1]
    movie_lines.append(line.split('\t')[-1])
movie_lines = [l.replace('""', '"').replace(' ', ' ') for l in movie_lines]
```

Tokenzing the sentences and words

[4]:
```
import nltk
#nltk.download('punkt')
from nltk.tokenize import sent_tokenize
from nltk.tokenize import TweetTokenizer

tokenizer = TweetTokenizer(preserve_case=False)
```

[5]:
```
sentences = []
for line in movie_lines:
    line_sents = sent_tokenize(line)
    for sent in line_sents:
        sentences.append(tokenizer.tokenize(sent))
```

[6]:
```
from collections import Counter

words = Counter(c for clist in sentences for c in clist)
```

1.0.2 Select (at least) two words which occur at least 100 times in this dataset. One word should have only one meaning. The other word should have (very many) multiple meanings.

Selecting two words

[7]:
```
multiple_meaning_word = 'date'
one_meaning_word = 'king'
```

Counting how often the word occurs in this dataset

[8]:
```
print('The word with multiple meaning: \'{}\', and has frequency in the dataset:
 ↪ {}'.format(multiple_meaning_word, words.get(multiple_meaning_word)))
print('The word with only one meaning: \'{}\', and has frequency in the dataset:
 ↪ {}'.format(one_meaning_word, words.get(one_meaning_word)))
```

```
The word with multiple meaning: 'date', and has frequency in the dataset: 371
The word with only one meaning: 'king', and has frequency in the dataset: 457
```

1.0.3 Compute the flair representations for all sentences where the word occurs in. From each of those sentence representations select the contextualized embedding vector of the word and save all these vectors in a list

Separating sentences for each word and saving it in a separate list

[9]:
```
sentences_multiple_word = []
sentences_one_word = []
for sent in sentences:
    if multiple_meaning_word in sent:
        sentences_multiple_word.append(sent)
    elif one_meaning_word in sent:
        sentences_one_word.append(sent)
```

Calculating the Contextualized embedding of a word in each sentence

[10]:
```
from flair.embeddings import WordEmbeddings, FlairEmbeddings, StackedEmbeddings
from flair.data import Sentence

def contextualized_embedding(word, sentences):
    embedding_vectors = []
    stacked_embeddings = StackedEmbeddings([
                                            WordEmbeddings('glove'),
                                            FlairEmbeddings('news-forward'),
                                            FlairEmbeddings('news-backward'),
                                           ])
    for sent in sentences:
        sentence = Sentence(' '.join(sent))
        stacked_embeddings.embed(sentence)
        for token in sentence:
            if token.text == word:
                embedding_vectors.append(token.embedding.cpu().detach().numpy())
```

```
    return np.array(embedding_vectors)
```

[11]:
```
multiple_embedding_vector = contextualized_embedding(multiple_meaning_word,␣
↪sentences_multiple_word)
one_embedding_vector = contextualized_embedding(one_meaning_word,␣
↪sentences_one_word)
```

1.0.4 Compute a centroid embedding vector for all contextualized embeddings of the word (compute the element-wise sum of all vectors and devide by the number of vectors - the resulting centroid vector is a vector in the same embedding space). Calculate the standard deviation of the cosine similarities of the contextualized embedding vectors to the centroid vector?

Computing the centroid embedding vector for each word

[12]:
```
def compute_centroid_vector(vectors):
    centroid = np.zeros(shape=(vectors[0].shape))
    for vec in vectors:
        centroid += vec
    return centroid / len(vectors)
```

[13]:
```
multiple_word_centroid = compute_centroid_vector(multiple_embedding_vector)
one_word_centroid = compute_centroid_vector(one_embedding_vector)
```

Calculating the Standard deviation of the cosine similarity between each vector and the centroid

[14]:
```
from scipy import spatial
def compute_cosine_similarity(vectors, centroid):
    cosine_similarities = []
    for vect in vectors:
        cosine_similarities.append(1 - spatial.distance.cosine(vect, centroid))
    return np.array(cosine_similarities)
```

[15]:
```
multiple_word_std = np.std(compute_cosine_similarity(multiple_embedding_vector,␣
↪multiple_word_centroid))
one_word_std = np.std(compute_cosine_similarity(one_embedding_vector,␣
↪one_word_centroid))
```

[16]:
```
print('Word: {}, with MULTIPLE meanings based on context:'.
↪format(multiple_meaning_word))
print('Standard deviation of the cosine similarities of the contextualized␣
↪embedding vectors to the centroid vector: {:.02}'.format(multiple_word_std))

print('\n\nWord: {}, with ONE meaning in whole context'.
↪format(one_meaning_word))
print('Standard deviation of the cosine similarities of the contextualized␣
↪embedding vectors to the centroid vector: {:.02}'.format(one_word_std))
```

```
Word: date, with MULTIPLE meanings based on context:
Standard deviation of the cosine similarities of the contextualized embedding
vectors to the centroid vector: 0.033
```

```
Word: king, with ONE meaning in whole context
Standard deviation of the cosine similarities of the contextualized embedding
vectors to the centroid vector: 0.018
```

It can be observe from the results above that the standard deviation of the word with one meaning only is very close to its centroid whereas on the otherhand, the word with multiple meanings in different context was a bit far from the centroid as compared to the other word.