Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Text, Web and Social Media Analytics Lab

Prof. Dr. Diana Hristova

Exercise 4. Text Representation (2)

In this exercise, we will apply the following models to the data from Exercise 2:

1. Word2Vec
2. Doc2vec
3. BERT (lemmatized)

At the end, we will derive a corpus with each of them which can be used in downstream tasks such as classification and clustering (see next exercises).

1. Open GoogleColab and create a New Notebook. Change to GPU session by Edit ➔ Notebook settings. Install the transformers library:
    a) !pip install transformers
2. Import the following packages:
    b) import pickle
    c) import pandas as pd
    d) from gensim.models import Word2Vec
    e) from scipy.spatial.distance import cosine
    f) from gensim.models.doc2vec import Doc2Vec, TaggedDocument
    g) import tensorflow as tf
    h) import torch
    i) from transformers import BertTokenizer, BertModel
    j) from keras.preprocessing.sequence import pad_sequences
    k) from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
2. Import the dataset Lemma in the notebook with pickle.load(). Print the first entry of the data frame to assure that it was imported correctly.
3. Generate a corpus of splitted texts similar to Exercise 3. Call it corpus_gen.

Part 1: Word2vec

4. Run the following code. What is it doing?
    a) *model = Word2Vec(corpus_gen, size=100, min_count=566)*
    b) *model.save('word2vec.model')*
5. Print the sorted features of the model using *model.wv.vocab.keys().* Compare them with those from Exercise 3.
6. Show *model.wv['car']*. What is it doing? Why does it have this size?
7. Run *model.wv.most_similar('car').* Does the result make sense?
8. Run *model.wv.most_similar(positive=['bike', ' machine'], topn=1).* What is it doing? Test other combinations.

9. Generate a corpus as a list, where for each document the list entry consists of the list of word2vec embeddings of the words in this document. Remove from it the empty documents. Why were those generated? Aggregate the values at a document level (i.e., one embedding per document) by averaging the word embeddings. Store the result for all documents in a data frame and then in GoogleDrive as WordtoVecModel.pkl.
10. Find the most similar words to the embedding representing the first document from 10. Use cosine() to determine the document on the corpus most similar to it.

Part 2: Doc2vec

11. Run gensim doc2vec model on *corpus_gen (vector_size=100, min_count=566)*. (**Note:** *Doc2vec* requires tagged documents). Determine the embedding representation for the first document as well as the document in the dataset most similar to it. Compare it to the result in 10. Generate the final corpus as a data frame and store it in GoogleDrive as DoctoVecModel.pkl..

Part 3: BERT

12. Confirm that GPU is detected and assign the GPU device to torch:
    a) *device_name = tf.test.gpu_device_name()*
    if device_name == '/device:GPU:0':
    print('Found GPU at: {}'.format(device_name))
    else:
    raise SystemError('GPU device not found')
    b) *if torch.cuda.is_available():*
    device = torch.device("cuda")
    print('There are %d GPU(s) available.' % torch.cuda.device_count())
    print('We will use the GPU:', torch.cuda.get_device_name(0))
    else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")

13. Estimate a BERT model:
    a) add [CLS] at the beginning and [SEP] at the end of each text
    sentences = ["[CLS] " + query + " [SEP]" for query in data_lemma]
    print(sentences[0])
    b) tokenize the texts using BERT tokenizer
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    tokenized_texts = [tokenizer.tokenize(sent) for sent in sentences]
    print(tokenized_texts[0])
    c) Analyse the length statistics of tokenized_texts to determine the maximum sequence length *m*
    d) Pad the text to the maximum sequence length
    sentences_padded = pad_sequences(tokenized_texts,
    dtype=object,maxlen=m, value='[PAD]', truncating="post",padding="post",
    return_tensors = 'pt')
    e) Map the tokens to BERT dictionary
    sentences_converted = [tokenizer.convert_tokens_to_ids(s) for s in sentences_padded]

f) Create attention masks

```
masks = []
for seq in sentences_converted:
  seq_mask = [float(i>0) for i in seq]
  masks.append(seq_mask)
```

g) Convert all of our data into torch tensors *inputs = torch.LongTensor(sentences_converted)*
masks = torch.LongTensor(masks)

h) Apply the model

- *model = BertModel.from_pretrained('bert-base-uncased')*
- *model.to(device)*
- *batch_size = 16*
  prediction_data = TensorDataset(inputs, masks)
  prediction_sampler = SequentialSampler(prediction_data)
  prediction_dataloader = DataLoader(prediction_data, sampler=prediction_sampler, batch_size=batch_size)
- *result=[]*
  i=0
  for batch in prediction_dataloader:
  batch = tuple(t.to(device) for t in batch)
  b_input_ids, b_input_mask = batch
  with torch.no_grad():
  outputs = model(b_input_ids)
  embeddings = outputs.pooler_output #CLS embeddings for the batch
  embeddings = embeddings.detach().cpu().numpy()
  result.append(embeddings)
  i=i+1
  print(' DONE.')

i) Store result in a data frame and print its head. What does this object contain? How can you use it for further tasks? Store the data frame as BertModel.pkl.