Assignment3_HD_embeddings_TOEFL

September 21, 2021

[1]:
```
##Load modules and libraries
import numpy as np
from scipy import spatial
RNG = np.random.default_rng(42) #for reproducibility ADDED BY ME
```

1 Function to generate "index" vectors

[2]:
```
def get_random_word_vector(dimension, k):
    #k is a number of +1s and -1s if you decide to implement Random Indexing␣
 ↪algorithm. So the total number of nonzero components in the vector is 2k
    #For BEAGLE k is not relevant
    v = np.array(np.zeros(dimension),np.int8) #Vector in initialized.

    #ToDo Generate components of an "index" vector randomly using the␣
 ↪randomness suitable for the chosen algorithm
    # Per An Introduction to Random Indexing (2005):  These index vectors are␣
 ↪sparse [few +1 & -1s, rest 0], high-dimensional
    indices_of_ones = RNG.choice(dimension, k, replace=False)
    ones_vector = RNG.binomial(1, 0.5, k)
    v[indices_of_ones] = 2*ones_vector - 1

    return v
```

2 Parameters

[3]:
```
TEST_NAME = "TOEFL_synonyms.txt" # file with TOEFL dataset
DATA_FILE_NAME = "Glemmatized.txt" # file with the text corpus


dimension = 500 # Dimensionality for high-dimensional vectors

THRESHOLD = 100000 # Frequency threshold in the corpus

ONES_NUMBER = 5 # number of nonzero elements in randomly generated␣
 ↪high-dimensional vectors
```

```
window_size = 3 #number of neighboring words to consider both back and forth.␣
↪In other words number of words before/after current word
```

3 Initialize "index" vectors and embeddings

[4]:
```
# Making into function so I can run experiments automatically
def init_index_vectors(dimension: int = dimension):
    # Returns 3 dictionaries
    dictionary = {} # vocabulary and corresponing random high-dimensional␣
↪vectors
    amount_dictionary = {} # counts frequency of words
    word_space = {} # stores embedings

    # Count how many times each word appears in the corpus
    text_file = open(DATA_FILE_NAME, "r")
    for line in text_file:
        if line != "\n":
            words = line.split()
            for word in words:
                if amount_dictionary.get(word) is None:
                    amount_dictionary[word] = 1
                else:
                    amount_dictionary[word] += 1
    text_file.close()

    #Create a dictionary with the assigned random high-dimensional vectors
    text_file = open(DATA_FILE_NAME, "r")
    for line in text_file: #read line in the file
        words = line.split() # extract words from the line
        for word in words:  # for each word
            if dictionary.get(word) is None: # If the word was not yed added to␣
↪the vocabulary
                if amount_dictionary[word] < THRESHOLD:
                    dictionary[word] = get_random_word_vector(dimension,␣
↪ONES_NUMBER) # assign an "index" vector
                else:
                    dictionary[word] = np.zeros(dimension) # frequent words are␣
↪assigned with empty vectors. In a way they will not contribute to the word␣
↪embedding
    text_file.close()
    return dictionary, amount_dictionary, word_space
```

4 Choose embeddigns to construct

[5]:
```
def choose_embeddings(word_space: dict, dimension: int = dimension):
    #Returns word_space. number_of_tests

    #Note that in order to save time we only create embeddings for the words␣
↪needed in the TOEFL task
    number_of_tests = 0
    TOEFL_file = open(TEST_NAME, "r") # open TOEFL file

    #Find all unique words amongst TOEFL tasks and initialize their embeddings␣
↪to zeros
    for line in TOEFL_file:
            words = line.split()
            word_space[words[0]] = np.zeros(dimension)
            word_space[words[1]] = np.zeros(dimension)
            word_space[words[2]] = np.zeros(dimension)
            word_space[words[3]] = np.zeros(dimension)
            word_space[words[4]] = np.zeros(dimension)
            number_of_tests += 1 # counts the number of test cases in TOEFL file
    TOEFL_file.close()
    return word_space, number_of_tests
```

5 Construct embeddings

[6]:
```
def construct_embeddings(dictionary: dict,
                         word_space: dict,
                        dimension: int = dimension,
                        window_size: int = window_size,
                        technique: str = "both") -> dict:
    # Technique = "context", "order", "both"
    # Returns word_space

    #Each line in the corpus is a sentence so we only consider the window of␣
↪words within the sentence.
    text_file = open(DATA_FILE_NAME, "r")

    line = "dummy" # To avoid skipping while
    while line != "":
        line = text_file.readline()
        words = line.split()
        for i in range(0,len(words)):
            if not (word_space.get(words[i]) is None): # This line forces us to␣
↪create only embeddings for words present in TOEFL
                #Form "context" vector
                context=np.zeros(dimension) # initialize context vector
                for j in␣
↪range(max(i-window_size,0),min(i+window_size+1,len(words))): # align window␣
↪size with the location of the focus word in the sentence
                    #ToDo increment context vector with the corresponding␣
↪"index" vectors
                    #Note that the index" vector for the focus word in nor␣
↪included into the context vector
                    if j != i:
                        context += dictionary.get(words[j])

                #Form "order" vector
                order=np.zeros(dimension) # initialize order vector
                for j in range(max(i-window_size,0),␣
↪min(i+window_size+1,len(words))): # align window size with the location of␣
↪the focus word in the sentence
                    #ToDo increment context vector with the properly permuted␣
↪"index" vectors
                    order += np.roll(dictionary.get(words[j]), j - i)

                # Update the embedding with new context and order vectors
                if technique == "context":
                    word_space[words[i]] += context # update the embedding with␣
↪new context vector
                elif technique == "order":
                    word_space[words[i]] += order # update the embedding with␣
↪new order vector
                else:
                    word_space[words[i]] += context # update the embedding with␣
↪new context vector
                    word_space[words[i]] += order # update the embedding with␣
↪new order vector
    return word_space
```

6 Testing of the embeddings on TOEFL

[7]:
```
#Used to check if the answer for TOEFL synonyms task is correct
def get_answer_mod(words):
    min_value = min(spatial.distance.cosine(words[0], words[1]), spatial.
↪distance.cosine(words[0], words[2]), spatial.distance.cosine(words[0],␣
↪words[3]),
                    spatial.distance.cosine(words[0], words[4]))
    if min_value == spatial.distance.cosine(words[0],words[1]):
        return 1
    else:
        return 0
```

```
[8]: def evaluate(amount_dictionary: dict, word_space: dict,
                  number_of_tests: int, dimension: int = dimension) -> None:
         zero_vector = np.zeros(dimension) # used to check if an embedding is non␣
      ↪empty
         i = 0
         TOEFL_file = open(TEST_NAME, 'r')
         right_answers = 0.0 # variable for correct answers
         number_skipped_tests = 0.0 # some tests could be skipped if there are no␣
      ↪corresponding words in the vocabulary extracted from the training corpus
         while i < number_of_tests:
                 line = TOEFL_file.readline() #read line in the file
                 words = line.split()  # extract words from the line
                 try:
                     if not(amount_dictionary.get(words[0]) is None): # check if␣
      ↪there word in the corpus for the query word
                         k = 1
                         while k < 5:
                             if np.array_equal(word_space[words[k]], zero_vector): #␣
      ↪if no representation was learnt assign a random vector
                                 word_space[words[k]] = np.random.randn(dimension)
                             k += 1
                         right_answers +=␣
      ↪get_answer_mod([word_space[words[0]],word_space[words[1]],word_space[words[2]],
                                     word_space[words[3]],word_space[words[4]]])␣
      ↪#check if word is predicted right
                 except KeyError: # if there is no representation for the query␣
      ↪vector than skip
                     number_skipped_tests += 1
                     print("skipped test: " + str(i) + "; Line: " + str(words))
                 except IndexError:
                     break
                 i += 1
         TOEFL_file.close()
         accuracy = 100 * right_answers / number_of_tests # accuracy of the␣
      ↪embeddings
         print("Dimensionality of embeddings: " +str(dimension) + "; Percentage of␣
      ↪correct answers in TOEFL: " + str(accuracy) + "%")
```

```
[9]: import time
     def run_experiment(dimension: int = dimension, window_size: int = window_size,␣
      ↪technique: str = "both") -> None:
         print("=> Starting Experiment!")

         start = time.time()
         dictionary, amount_dictionary, word_space = init_index_vectors(dimension)
         print(f"=> Computed Dictionaries! Took {(time.time()-start) / 60} min")
```

```
    start = time.time()
    word_space, number_of_tests = choose_embeddings(word_space, dimension)
    print(f"=> Setup Embeddings! Took {time.time()-start}")


    start = time.time()
    word_space = construct_embeddings(dictionary, word_space, dimension,␣
↪window_size, technique)
    print(f"=> Constructed Embeddings! Took {(time.time()-start) / 60} min")

    print(f"=>Testing on D={dimension}, Window = {window_size}, and Technique =␣
↪{technique}")
    evaluate(amount_dictionary, word_space, number_of_tests,␣
↪dimension=dimension)
    print("--------------\n")
```

6.1 Q1 and Q3

6.1.1 Q1: Get the performance of the embeddings on the TOEFL synonymy assessment for several different dimensionalities.

6.1.2 Q3: Report the accuracy on the TOEFL synonymy assessment for all simulations. Elaborate how accuracy changes with the dimensionality.

6.1.3 From the wording, it seems like Q3 is asking to report results of Q1

6.1.4 ANSWER: As dimensionality increases, performance generally increases but in our case at 1000 the performance drop. In general, greater dimensionality should boost accuracy because greater dimensionality allows us to encode more information and avoid collisions or relative-collisions (in terms of being very similar) between words with different meanings. However, in our case our vectors are very sparse as k = 5 which means that as dimension a greater percent of our vectors are 0.

[10]:
```
# Running Q1
for dim in [250, 500, 1000]:
    run_experiment(dim)
```

```
=> Starting Experiment!
=> Computed Dictionaries! Took 0.952737029393514 min
=> Setup Embeddings! Took 0.007855892181396484
=> Constructed Embeddings! Took 3.1455177466074624 min
=>Testing on D=250, Window = 3, and Technique = both
Dimensionality of embeddings: 250; Percentage of correct answers in TOEFL: 40.0%
--------------

=> Starting Experiment!
=> Computed Dictionaries! Took 0.6510003169377645 min
=> Setup Embeddings! Took 0.001680135726928711
=> Constructed Embeddings! Took 3.136160699526469 min
=>Testing on D=500, Window = 3, and Technique = both
Dimensionality of embeddings: 500; Percentage of correct answers in TOEFL: 47.5%
--------------

=> Starting Experiment!
=> Computed Dictionaries! Took 0.6559061686197917 min
=> Setup Embeddings! Took 0.0017879009246826172
=> Constructed Embeddings! Took 3.506387631098429 min
=>Testing on D=1000, Window = 3, and Technique = both
Dimensionality of embeddings: 1000; Percentage of correct answers in TOEFL:
42.5%
--------------
```

6.2 Q2: (Random Indexing) How does the size of the window around the focus word affect the results on the TOEFL synonymy assessment?

6.2.1 Answer: Increasing window size helps to a point. We see that it is best at size 6 (out of 1,3,6) while there is a drop-off between size 1 and 3. This may be because in a window size like 3 you may be incorporating nearby prepositions or high frequency words of little use (e.g. the, a). Thus there is a "Goldilocks zone" where the window size encompasses enough nearby context without that contex encoding too much irrelevant information or too much context in general.

[11]:
```python
# Running Q2
for size in [1,3,6]: #Testing window_sizes of 1, 3, 6
    run_experiment(window_size=size)
```

```
=> Starting Experiment!
=> Computed Dictionaries! Took 0.7441611329714457 min
=> Setup Embeddings! Took 0.0018248558044433594
=> Constructed Embeddings! Took 1.723079784711202 min
=>Testing on D=500, Window = 1, and Technique = both
Dimensionality of embeddings: 500; Percentage of correct answers in TOEFL:
33.75%
--------------

=> Starting Experiment!
=> Computed Dictionaries! Took 0.6857862353324891 min
=> Setup Embeddings! Took 0.0015149116516113281
=> Constructed Embeddings! Took 3.4392805695533752 min
=>Testing on D=500, Window = 3, and Technique = both
Dimensionality of embeddings: 500; Percentage of correct answers in TOEFL:
31.25%
--------------
```

```
=> Starting Experiment!
=> Computed Dictionaries! Took 0.7124945521354675 min
=> Setup Embeddings! Took 0.0011870861053466797
=> Constructed Embeddings! Took 5.669906083742777 min
=>Testing on D=500, Window = 6, and Technique = both
Dimensionality of embeddings: 500; Percentage of correct answers in TOEFL: 47.5%
--------------
```

[12]:
```
# Q3 is Q1
```

6.3 Q4: How is accuracy on the TOEFL synonymy assessment affected when only context or only order parts of the embedding are used?

6.3.1 Answer: In this test, we are really concerned with meaning as we are assesssing similarity between synonyms. Thus, order complicates the matter as order is used to enforce grammar-like structure (in terms of relative orders or words) when in this case we just need words and their meanings in context as synonyms will be used interchangably in like contexts. Enforcing order as we do causes a loss of information as by shifting our vectors we are most likely creating new orthogonal vectors to the original index vector of those words.

[13]:
```
# Q4
run_experiment(technique="context")
run_experiment(technique="order")
```

```
=> Starting Experiment!
=> Computed Dictionaries! Took 0.8159211317698161 min
=> Setup Embeddings! Took 0.007192134857177734
=> Constructed Embeddings! Took 3.504171347618103 min
=>Testing on D=500, Window = 3, and Technique = context
Dimensionality of embeddings: 500; Percentage of correct answers in TOEFL: 57.5%
--------------

=> Starting Experiment!
=> Computed Dictionaries! Took 0.6906440019607544 min
=> Setup Embeddings! Took 0.0009257793426513672
=> Constructed Embeddings! Took 3.449483354886373 min
=>Testing on D=500, Window = 3, and Technique = order
Dimensionality of embeddings: 500; Percentage of correct answers in TOEFL:
38.75%
--------------
```

[ ]:

[ ]: