Stat 766 Homework 3

Due on 11/8/2022 11:59pm

1. For entire question 1 parts (1) - (5), we will use Google's pre-trained word embeddings based on word2vec word2vec-google-news-300.

   (1). For each word in 'No banks to deposit money near the river bank' and 'Stick to the rule by stirring with a wood stick', find the top 10 words among all 3 million that are most similar to each word in terms of cosine similarity. If a word does not exist in word2vec vocabulary, skip that word. Combine the returned top 10 results from different words in a pandas dataframe for each sentence. Store the resulting dataframe for the first sentence in sentence1_most_similar and that for the 2nd sentence in sentence2_most_similar.

   To make the format consistent with my answer on autograder, first initialize a dictionary to store the returned result. For each word, you call the .most_similar( ) and store the result as the value for the key in the dictionary, where the key is the word. In the end, convert the entire dictionary to a pandas dataframe and save to csv file with the command below.

   ```
   import csv
   sentence1_most_similar.to_csv('sentence1_most_similar.csv', sep='~')
   sentence2_most_similar.to_csv('sentence2_most_similar.csv', sep='~')
   ```

   You can use word_vectors.most_similar() to find the top 10 most similar words, where word_vectors is from either api.load('word2vec-google-news-300') or KeyedVectors. Which of the word's meaning do not match the returned results? Why do you think they did not work?

1. (2). The .most_similar function finds words that are most similar to the words in the positive list and most dissimilar from the words in the negative list. For example, we can get the top 10 answers to analogy question "man : king :: woman : x", what is x?

   ```
   word_vectors.most_similar(positive=['woman', 'king'], negative=['man'])
   ```

   returns the top 10 words ranked most similar (largest numerical value).

   Find the top 10 answer to the following analogy questions using word2vec's pre-trained vectors:

   'better:good :: worse: x' store your answer in q1_2_1
   'U.S.:dollar :: China: x' store your answer in q1_2_2

   Then put the two in a pandas dataframe with

   ```
   ques1_2 = pd.DataFrame( {'q1_2_1':q1_2_1,  'q1_2_2':q1_2_2})
   import csv
   ques1_2.to_csv('ques1_2.csv', sep='~')
   ```

   How do the results compare to GLOVE results in our lecture slides?

1. (3). Define a function to get the pre-trained word2vec vector for each word in sorted words_list (in increasing order) and then put the vectors into a matrix M such that one row of M is the word2vec vector of one word. Return the matrix M as a numpy array and word2Ind, which is a dictionary mapping each word to its row number in M. If a word does not exist (i.e. KeyError), do not worry about finding its embedding. Instead, collect all such words in a list called non_exist for those words that gives KeyError from word_vectors.get_vector( ) command.

```
def get_word2vec_matrix(  words_list ):
    # first sort words_list and then complete the function

    return M, word2Ind, non_exist
```

(4). Preprocessing the following text by removing punctuation, English stop words, and numbers, and then converting each word token to lower case. Sort the unique tokens in increasing order. Then get the embedding matrix for each of the sorted unique tokens using your function in part (3).

A large oil spill off the southern California coast was described as an "environmental catastrophe" by the mayor of Huntington Beach on Sunday, as the breach of an oil rig pipeline left dead fish and birds strewn on the sand and offshore wetlands clogged with oil. An estimated 126,000 gallons, or 3,000 barrels, had spread into an oil slick covering about 13 square miles of the Pacific Ocean since it was first reported on Saturday morning, said the mayor of Huntington Beach, at a press conference. The beachside city, about 40 miles south of Los Angeles, was bearing the brunt of the spill.

Store the resulting matrix in Mat, the word to index in word2Ind and the list of words with KeyError in non_exist. Then save the matrix Mat to csv file and collect the other two in ques1_4 as follows:

```
pd.DataFrame(Mat).to_csv('M_word2vec.csv',
    header=False, index=False )
ques1_4 = { 'word2Ind':word2Ind, 'non_exist':non_exist}
```

(5) For embedding matrix in part (4), perform a principal component decomposition and project each token's word2vec vector to first two principal components. Store the projected 2-d vectors in M2d_word2vec as a numpy array with shape (n, 2), where n is the number of unique words after proprocessing in part (4).

Save the projected 2-d matrix M2d_word2vec in csv file with the command below

```
pd.DataFrame(M2d_word2vec).to_csv('M2d_word2vec.csv',
    header=False, index=False )
```

Produce a scatterplot of the projected embedding. Label each point with the word as text in the graph.

(6) Repeat the questions in parts (3), (4) and (5) using pre-trained GLOVE vectors glove.6B.300d.txt.

- ```
  def get_glove_matrix(  words_list ):
      # first sort words_list and then complete the function

      return M, word2Ind, non_exist
  ```
- Save the matrix for part (4) with GLOVE to Mat_glove in csv file as follows

  ```
  pd.DataFrame(Mat_glove).to_csv('M_glove.csv',
      header=False, index=False )
  ```
- Store the word2Ind and non_exist in dictionary ques1_4_glove, where ques1_4_glove has the same keys 'word2Ind', 'non_exist' as ques1_4. Save the projected 2-d matrix M2d_glove in csv file with the command below

  ```
  pd.DataFrame(M2d_glove).to_csv('M2d_glove.csv',
      header=False, index=False )
  ```

(7) Repeat the questions in parts (3), (4) and (5) using pre-trained fastText vectors wiki-news-300d-1M.vec. Since fastText is able to create word embedding for unseen words, do not eliminate any word from the words_list. Instead, for words not in wiki-news-300d-1M.vec, learn new word vectors using the given text as training data. Collect the word vectors for newly trained words in M_new as a numpy array and store word to index mapping in word2Ind_new.
  ```

- ```
  def get_fasttext_matrix(  words_list, training_text ):
      # first sort words_list and then complete the function

      return M, word2Ind, M_new, word2Ind_new
  ```
- Save the matrix for part (4) with fastText to Mat_fasttext in csv file as follows

  ```
  pd.DataFrame(Mat_fasttext).to_csv('M_fasttext.csv',
      header=False, index=False )
  ```
- Store word2Ind and word2Ind_new in dictionary ques1_4_fasttext, where ques1_4_fasttext has the keys 'word2Ind' and 'word2Ind_new'. Save the projected 2-d matrix M2d_fasttext in csv file with the command below

  ```
  pd.DataFrame(M2d_fasttext).to_csv('M2d_fasttext.csv',
      header=False, index=False )
  ```

(8) Compare the performance of word2vec, GLOVE, fastText using the plots from parts (5), (6), and (7) by considering the following questions: what words cluster together? What words were not clustered together but you think should have?

Question 2: Word embedding based on co-occurrence matrix and SVD

A co-occurrence matrix counts how often words co-occur in some environment. Given some word w_i occurring in the document, we consider the context window surrounding w_i of fixed window size k in the same sentence. That is, w_i's window contains n preceding and n subsequent words around w_i in that sentence: $w_{i-n}, \ldots, w_{i-1}$ and $w_{i+1}, \ldots, w_{i+n}$. (If w_i is at the beginning or the end of a sentence, the window contains less context words.)

The co-occurence matrix M is a symmetric word-by-word matrix in which M_{ij} is the number of times w_j appears inside w_i's window.

For example, if the corpus docs contains two documents:

doc1= 'This is reproducible result. Setting seed is helpful.'

doc2= 'This result is not helpful.'

docs =[doc1, doc2]

After converting to lower case letters and removing punctuations (keeping only letters in A-Z and a-z), with window size k=2, the co-occurence matrix is

	helpful	is	not	reproducible	result	seed	setting	this
helpful	0	2	1	0	0	1	0	0
is	2	0	1	1	2	1	1	2
not	1	1	0	0	1	0	0	0
reproducible	0	1	0	0	1	0	0	1
result	0	2	1	1	0	0	0	1
seed	1	1	0	0	0	0	1	0
setting	0	1	0	0	0	1	0	0
this	0	2	0	1	1	0	0	0

Our computations will use numpy array. Hence no row or column names are attached to the numpy matrix. Instead, we will use a word to index dictionary word2Ind to tell which word corresponds to each row of the matrix. The word2Ind for above example is

```
{'helpful':0, 'is':1, 'not':2, 'reproducible':3, 'result':4, 'seed':5, 'setting':6, 'this':7}
```

a) Define a function to compute co-occurrence matrix for a given list of documents and window size k. Each window contains k words before and k words after the center word in the same sentence. If the center word is at the beginning or the end of a sentence, the window only extends to what is available. Return the co-occurrence matrix as a NumPy array and the word to index dictionary. The word to index dictionary's keys are the unique words in all documents. Its keys are in increasing alphabetical order. Its value gives the row index of the word in the co-occurrence matrix (see previous page for example input and output).
```

```
# docs is a list of documents.
def co_occurrence(docs,  k=2):
    # write your implementation to compute co-occurrence matrix M
    # and word to index dictionary word2Ind here
    return M, word2Ind
```

Note:

Directly computing the matrx could take overnight. If you successfully finished computing the matrx M and word2Ind for Q2 (d), save them to hard disk.

If your computer has multiple processors, you can speed up the computation by letting different threads or processors compute different rows of the co-occurrence matrix. To do so,

- you define a function to compute the co-occurrence count for one word with all words in the vocabulary; define another function (referred as callback function) to collect results from processing one word.
- Then call the function in a loop that iterate through all the words in the vocabulary. Within the loop, you can use pool.apply_async( ) to parallelize the computation. See https://www.machinelearningplus.com/python/parallel-processing-python/ for more illustration.

b) Define a function to perform SVD on the co-occurrence matrix and project the column vectors in the co-occurence matrix to the first m singular vectors. Since the matrix's dimension will be huge, we would like to use truncated SVD, in which only the t left-singular vectors and t right-singular vectors corresponding to the t largest singular values are calculated. Take t large enough while doing truncated SVD such that the remaining singular values are negligible. The scikit-learn library (sklearn) provides an efficient randomized algorithm for calculating large-scale Truncated SVD. See sklearn.decomposition.TruncatedSVD (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html) . After decomposition, the .fit_transform method gives the projection. Return the projected matrix M_reducted that has shape $V \times m$, where V is the number of unique words in the corpus. M_reducted is a reduced dimension embedding matrix.

```
# M is co-occurrence matrix.
def SVD_embedding(M,  m=2):
    # write your implementation here
    return M_reduced
```

The output for the example docs on the first page of question 2 is on the right side.

```
[[ 1.69444516, -1.45463979],
 [ 3.13097349,  2.45697096],
 [ 1.35113514,  0.03784516],
 [ 1.39134292, -0.06440965],
 [ 2.12117342, -1.12887505],
 [ 1.09857002, -0.08786895],
 [ 0.8226602 , -0.70852139],
 [ 1.90116545, -1.11272762]]
```

Note:

- SVD and PCA are similar. The only difference is that SVD calculation does not center the columns before decomposition. This saves some time for large scale matrices.
- Websites https://medium.com/analytics-vidhya/matrix-factorization-for-word-vectors-61824ee9a751 and https://medium.com/analytics-vidhya/co-occurrence-matrix-singular-value-decomposition-svd-31b3d3deb305 have more information about SVD on co-occurrence matrix.
- Computing SVD is quadratic in memory demand. If you don't have a big memory, your computer might unexpectedly shut down in the middle of the computation.

c) Consider 2-d projected embeddings. Define a function to produce a scatterplot of the projected $V \times 2$ embeddings for a given list of words. Label each point with the word as text in the graph. (NOTE: do not plot all the words listed in the word to index dictionary).

```
#input:
#  M_reduced (numpy array of shape (V , k)): k-dimensional word embeddings
```

```
#  word2Ind (dict): dictionary that maps word to indices for matrix M
#  words (list of strings): words whose embeddings we want to visualize
def plot_embeddings(M_reduced, word2Ind, words):
    # write your implementation here
```

We have plotting examples in lecture slides. For future reference, a good way to find relevant code to make a plot is to look thought the Matplotlib gallery (https://matplotlib.org/gallery/index.html). Find a plot that looks somewhat like what you want, and adapt the code they give.

d) The Reuters news data contains news between 2019 and summer 2021. First select only news articles that has 'coronavirus' in the 'article' column of reuters_news_articles.sqlite. You can do this with INSTR(string, substring) condition as follows:

```
cur.execute('SELECT article FROM reuters_news WHERE INSTR(article, "coronavirus")>0')
```

Then apply your functions to the selected news articles. Compute the co-occurrence matrix with fixed window size of k=4.Then use TruncatedSVD to compute 2-dimensional embeddings of each word. For the following words:

covid, coronavirus, fever, sore, cough, deaths, unemployment, food, financial, jobs, market, airline, economy, digital, zoom, classes, violence, crime

normalize the returned vectors with the code below, so that all the vectors will appear around the unit circle (therefore closeness is directional closeness). Note: The code below does the normalizing with the NumPy concept of broadcasting. (see https://jakevdp.github.io/PythonDataScienceHandbook/02.05-computation-on-arrays-broadcasting.html).

What words cluster together in the 2-d embedding space? What words don't cluster together but you think they should have? How does this embedding model perform?

```
reuters_docs = # read in the data from database
M, word2Ind = co_occurrence(reuters_docs, k=4)
M_reduced = SVD_embedding(M, m=2)
# Rescale (normalize) the rows to make them each of unit-length
M_lengths = np.linalg.norm(M_reduced, axis=1)
M_normalized = M_reduced/M_lengths[:, np.newaxis] # broadcasting
words =  # put in list of words
plot_embeddings(M_normalized, word2Ind, words)
```

Homework - Question 3

For this question, we will train a fastText model with the posted reuters news data.

(1) First select only news articles that have 'coronavirus' in the 'article' column of reuters_news_articles.sqlite. Then combine all the selected news articles into a single string, convert to lowercase letters, and then keep only letters and a space between every two consecutive words. (i.e., remove all other characters).

Train a fastText model using the processed, selected news articles as training data. You can use default parameters during training. If you are not finishing parts (2) and (3) immediately, you can save your trained model and load it again later with the commands below

```
model.save_model('ques3fasttext.bin')
import fasttext
model = fasttext.load_model('ques3fasttext.bin')
```

Using your trained model, get the top 10 nearest neighbours of word 'coronavirus'. Put the result in a dataframe and save to file nearest10_to_coronavirus.csv Comment on the result accuracy.

(2) Use your trained model to get the word vector representations of words'covid', 'fly', 'flu' and 'coronavirus'. Compute the cosine similarity between each one of three words 'covid', 'fly', 'flu' and 'coronavirus'. Save the results in a dictionary as ques3_2 with keys 'sim_covid_coronavirus', 'sim_fly_coronavirus', 'sim_flu_coronavirus'. Comment on the result accuracy.

(3) For the same list of words as in Question 2 d):

covid, coronavirus, fever, sore, cough, deaths, unemployment, food, financial, jobs, market, airline, economy, digital, zoom, classes, violence, crime

Get the fastText word vectors using your trained model in part (1) and define a dictionary to give the word to index mapping word2Ind_ques3. Project these vectors onto the first two Principal components. Store the projected 2-d vectors in M2d_fasttext_ques3 as a numpy array and save to a csv file with the command below

```
pd.DataFrame(M2d_fasttext_ques3).to_csv('M2d_fasttext_ques3.csv',
    header=False, index=False )
```

Produce a scatterplot of the projected vectors. Label each point with the word as text in the graph. Compare the result with those in Question 2 d).

Submission Instructions

- Collect the dictionaries for questions 1 (4), 1 (6), 1 (7), 3 (2) and 3 (3). Save it to file 'answer.txt' with the command below

```
ques3_2_value_str = [str(value[0][0]) for value in ques3_2.values()]
ques3_2_dict = dict(zip(list(ques3_2.keys()), ques3_2_value_str))
answer = { 'ques1_4' : ques1_4, 'ques1_4_glove' : ques1_4_glove,
    'ques1_4_fasttext' : ques1_4_fasttext, 'ques3_2': ques3_2_dict,
    'word2Ind_ques3': word2Ind}
import json
answer_text = json.dumps(answer)
textfile = open('answer.txt', 'w')
textfile.write(answer_text )
textfile.close()
```

While you are working on part (4), you don't have answer to parts (6) or (7) or later questions, then temporarily set those latter ones (such as ques1_4_glove and ques1_4_fasttext) to 0 if you want to submit to gradescope.

- Make sure
  - your word2Ind for M_word2vec.csv, M2d_word2vec.csv are the same,
  - your word2Ind for M_glove.csv and M2d_glove.csv are the same, and
  - your word2Ind for M_fasttext.csv, M2d_fasttext.csv are the same.
  - your word2Ind_ques3 gives the correct mapping to the rows in your M2d_fasttext_ques3.csv
- Prepare your submission for gradescope
  - Collect the following files together and add them to a .zip file:
    your completed hw3_answer_template.py, answer.txt, sentence1_most_similar.csv and sentence2_most_similar.csv, ques1_2.csv, M_word2vec.csv, M2d_word2vec.csv, M_glove.csv, M2d_glove.csv, M_fasttext.csv, M2d_fasttext.csv, nearest10_to_coronavirus.csv, M2d_fasttext_ques3.csv
  - Upload the .zip file to Gradescope as your submission. As you do, you should see that Gradescope is automatically unzipping the files as you submit; that's fine. Correct errors if you see any returned from the autograder and resubmit.
- Prepare your submission for Canvas for peer review
  - Copy your code for each part to jupyter notebook in Visual Studio Code or google colab into code cell. Add print statement to show snapshot of outputs. Run the cell.

- Add a Text cell to answer the question in that part.
- Move on to next part of each question. Continue further until you are done with all parts in all questions.
- In the end, you should have a code cell, a text cell, and output for each part of each question. Note: The last part of question 2 might take too long to run in one session of colab (which will cause you to restart). So you can complete and save the results for all earlier questions first and work on the last part of question 2 separately.
- Download the source code .ipynb or .py file. Print a pdf file that shows all the code cells, text cells and outputs.
- Submit the source code and pdf file on Canvas.