C1_W3_Assignment

October 29, 2020

1 Assignment 3: Hello Vectors

Welcome to this week's programming assignment on exploring word vectors. In natural language processing, we represent each word as a vector consisting of numbers. The vector encodes the meaning of the word. These numbers (or weights) for each word are learned using various machine learning models, which we will explore in more detail later in this specialization. Rather than make you code the machine learning models from scratch, we will show you how to use them. In the real world, you can always load the trained word vectors, and you will almost never have to train them from scratch. In this assignment, you will:

- Predict analogies between words.
- Use PCA to reduce the dimensionality of the word embeddings and plot them in two dimensions.
- Compare word embeddings by using a similarity measure (the cosine similarity).
- Understand how these vector space models work.

1.1 1.0 Predict the Countries from Capitals

In the lectures, we have illustrated the word analogies by finding the capital of a country from the country. We have changed the problem a bit in this part of the assignment. You are asked to predict the **countries** that corresponds to some **capitals**. You are playing trivia against some second grader who just took their geography test and knows all the capitals by heart. Thanks to NLP, you will be able to answer the questions properly. In other words, you will write a program that can give you the country by its capital. That way you are pretty sure you will win the trivia game. We will start by exploring the data set.

1.1.1 1.1 Importing the data

As usual, you start by importing some essential Python libraries and then load the dataset. The dataset will be loaded as a Pandas DataFrame, which is very a common method in data science. This may take a few minutes because of the large size of the data.

```
In [ ]: # Run this cell to import packages.
        import pickle
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
from utils import get_vectors
```

```
In [ ]: data = pd.read_csv('capitals.txt', delimiter=' ')
        data.columns = ['city1', 'country1', 'city2', 'country2']

        # print first five elements in the DataFrame
        data.head(5)
```

1.1.2 To Run This Code On Your Own Machine:

Note that because the original google news word embedding dataset is about 3.64 gigabytes, the workspace is not able to handle the full file set. So we've downloaded the full dataset, extracted a sample of the words that we're going to analyze in this assignment, and saved it in a pickle file called `word_embeddings_capitals.p`

If you want to download the full dataset on your own and choose your own set of word embeddings, please see the instructions and some helper code.

- Download the dataset from this page.
- Search in the page for 'GoogleNews-vectors-negative300.bin.gz' and click the link to download.

Copy-paste the code below and run it on your local machine after downloading the dataset to the same directory as the notebook.

```
import nltk
from gensim.models import KeyedVectors


embeddings = KeyedVectors.load_word2vec_format('./GoogleNews-vectors-negative300.bin', binary =
f = open('capitals.txt', 'r').read()
set_words = set(nltk.word_tokenize(f))
select_words = words = ['king', 'queen', 'oil', 'gas', 'happy', 'sad', 'city', 'town', 'villag
for w in select_words:
    set_words.add(w)

def get_word_embeddings(embeddings):

    word_embeddings = {}
    for word in embeddings.vocab:
        if word in set_words:
            word_embeddings[word] = embeddings[word]
    return word_embeddings


# Testing your function
word_embeddings = get_word_embeddings(embeddings)
```

```
print(len(word_embeddings))
pickle.dump( word_embeddings, open( "word_embeddings_subset.p", "wb" ) )
```

Now we will load the word embeddings as a Python dictionary. As stated, these have already been obtained through a machine learning algorithm.

```
In [ ]: word_embeddings = pickle.load(open("word_embeddings_subset.p", "rb"))
        len(word_embeddings)  # there should be 243 words that will be used in this assignment
```

Each of the word embedding is a 300-dimensional vector.

```
In [ ]: print("dimension: {}".format(word_embeddings['Spain'].shape[0]))
```

1.1.3 Predict relationships among words

Now you will write a function that will use the word embeddings to predict relationships among words. * The function will take as input three words. * The first two are related to each other. * It will predict a 4th word which is related to the third word in a similar manner as the two first words are related to each other. * As an example, "Athens is to Greece as Bangkok is to ______"? * You will write a program that is capable of finding the fourth word. * We will give you a hint to show you how to compute this.

A similar analogy would be the following:

You will implement a function that can tell you the capital of a country. You should use the same methodology shown in the figure above. To do this, compute you'll first compute cosine similarity metric or the Euclidean distance.

1.1.4 1.2 Cosine Similarity

The cosine similarity function is:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1}$$

A and B represent the word vectors and A_i or B_i represent index i of that vector. & Note that if A and B are identical, you will get $cos(\theta) = 1$. * Otherwise, if they are the total opposite, meaning, $A = -B$, then you would get $cos(\theta) = -1$. * If you get $cos(\theta) = 0$, that means that they are orthogonal (or perpendicular). * Numbers between 0 and 1 indicate a similarity score. * Numbers between -1-0 indicate a dissimilarity score.

Instructions: Implement a function that takes in two word vectors and computes the cosine distance.

Hints

Python's NumPy library adds support for linear algebra operations (e.g., dot product, vector norm …).

Use numpy.dot .

Use numpy.linalg.norm .

```
In [ ]: # UNQ_C1 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
        def cosine_similarity(A, B):
            '''
            Input:
                A: a numpy array which corresponds to a word vector
                B: A numpy array which corresponds to a word vector
            Output:
                cos: numerical number representing the cosine similarity between A and B.
            '''

            ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

            dot = np.dot(A,B)
            norma = np.sqrt(np.dot(A,A))
            normb = np.sqrt(np.dot(B,B))
            cos = dot / (norma*normb)

            ### END CODE HERE ###
            return cos
```

```
In [ ]: # feel free to try different words
        king = word_embeddings['king']
        queen = word_embeddings['queen']

        cosine_similarity(king, queen)
```

Expected Output:
≈ 0.6510956

1.1.5 1.3 Euclidean distance

You will now implement a function that computes the similarity between two vectors using the Euclidean distance. Euclidean distance is defined as:

$$\begin{aligned} d(\mathbf{A}, \mathbf{B}) = d(\mathbf{B}, \mathbf{A}) &= \sqrt{(A_1 - B_1)^2 + (A_2 - B_2)^2 + \cdots + (A_n - B_n)^2} \\ &= \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2} \end{aligned}$$

- n is the number of elements in the vector
- A and B are the corresponding word vectors.
- The more similar the words, the more likely the Euclidean distance will be close to 0.

Instructions: Write a function that computes the Euclidean distance between two vectors.
Hints
Use numpy.linalg.norm .

```
In [ ]: # UNQ_C2 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
        def euclidean(A, B):
```

```
    """
    Input:
        A: a numpy array which corresponds to a word vector
        B: A numpy array which corresponds to a word vector
    Output:
        d: numerical number representing the Euclidean distance between A and B.
    """

    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

    # euclidean distance

    d = np.linalg.norm(A-B)

    ### END CODE HERE ###

    return d
```

In [ ]:
```
# Test your function
euclidean(king, queen)
```

Expected Output:
2.4796925

1.1.6 1.4 Finding the country of each capital

Now, you will use the previous functions to compute similarities between vectors, and use these to find the capital cities of countries. You will write a function that takes in three words, and the embeddings dictionary. Your task is to find the capital cities. For example, given the following words:

- 1: Athens 2: Greece 3: Baghdad,

your task is to predict the country 4: Iraq.
Instructions:

1. To predict the capital you might want to look at the *King - Man + Woman = Queen* example above, and implement that scheme into a mathematical function, using the word embeddings and a similarity function.
2. Iterate over the embeddings dictionary and compute the cosine similarity score between your vector and the current word embedding.
3. You should add a check to make sure that the word you return is not any of the words that you fed into your function. Return the one with the highest score.

In [ ]:
```
# UNQ_C3 (UNIQUE CELL IDENTIFIER, DO NOT EDIT)
def get_country(city1, country1, city2, embeddings):
    """
    Input:
```

```python
        city1: a string (the capital city of country1)
        country1: a string (the country of capital1)
        city2: a string (the capital city of country2)
        embeddings: a dictionary where the keys are words and values are their embeddi
    Output:
        countries: a dictionary with the most likely country and its similarity score
    """
    ### START CODE HERE (REPLACE INSTANCES OF 'None' with your code) ###

    # store the city1, country 1, and city 2 in a set called group
    group = set((city1, country1, city2))

    # get embeddings of city 1
    city1_emb = word_embeddings[city1]

    # get embedding of country 1
    country1_emb = word_embeddings[country1]

    # get embedding of city 2
    city2_emb = word_embeddings[city2]

    # get embedding of country 2 (it's a combination of the embeddings of country 1, c
    # Remember: King - Man + Woman = Queen
    vec = country1_emb - city1_emb + city2_emb

    # Initialize the similarity to -1 (it will be replaced by a similarities that are
    similarity = -1

    # initialize country to an empty string
    country = ''

    # loop through all words in the embeddings dictionary
    for word in embeddings.keys():

        # first check that the word is not already in the 'group'
        if word not in group:

            # get the word embedding
            word_emb = word_embeddings[word]

            # calculate cosine similarity between embedding of country 2 and the word
            cur_similarity = cosine_similarity(vec,word_emb)

            # if the cosine similarity is more similar than the previously best simila
            if cur_similarity > similarity:

                # update the similarity to the new, better similarity
                similarity = cur_similarity
```