Vector Representations

April 26, 2018

1 Task 1: K Nearest Neighbors (kNN) of Your Name

1.1 My name is 'Shreya', and so I ran the kNN on that name.

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import normalize

def loadGloveModel(gloveFile):
    print ("Loading GloVe Model.")
    f = open(gloveFile,'r')
    model = {}
    for line in f:
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done loading!")
    return model

model = loadGloveModel('glove.6B.50D.txt')
df = pd.DataFrame(model)

name = df['shreya']

simi_list = []
for i in df:
    comp = df[i]
    value = np.dot(comp, name)/(np.linalg.norm(comp) * np.linalg.norm(name))
    simi_list.append((value, i))

simi_list.sort(reverse=True)

k = 1
print("Shreya, your k-nearest neighbors are ...")
for k in range(1, 6):
    print(simi_list[k][1], ",", simi_list[k][0])
```

```
Loading GloVe Model.
Done loading!
Shreya, your k-nearest neighbors are ...
ghoshal , 0.9726679413448639
kaela , 0.7380312968473076
häggkvist , 0.6978907305313238
lovich , 0.6804466875983098
stromae , 0.6800098366433656
```

2 Task 2: Vector Representation of a Sentence

2.0.1 In this task, we want to create a vector representation for an entire sentence simply by taking the average of all word vectors for words in that sentence.

2.0.2 This requires: 1) Tokenizing the sentence, 2) Looking up each individual word in our GloVe text document, and 3) Averaging (component-wise sum of the word vectors, and then divide each component by the number of words in the sentence that were covered by the data)

2.0.3 Then, we are asked to choose a random sentence S0 and compute the vector representation of that sentence using the above method. List the nearest neighbour words to that sentence vector.

```
In [5]: import nltk
        nltk.download('punkt')
        import matplotlib
        from nltk.tokenize import sent_tokenize, word_tokenize

        # Choose a sentence and convert words into lowercase

        sent_0 = "Scott Fitzgerald is very handsome.".lower()

        # Tokenize the sentence - ignore tokens that are not covered by the vocabulary of the t

        words_0 = word_tokenize(sent_0)
        # df_words_0 = pd.DataFrame(words_0)
        n0 = len(words_0)

        # Look up word vectors

        sent_0_vec = []

        for j in words_0:
            curr = df[j]
            sent_0_vec.append(curr)

        # Compute the component-wise sum of the word vectors, then divide
```

```
        # each component by the number of words in the sentence that were
        # covered by the data

        comp_wise_sum_0 = np.add.reduce(sent_0_vec)
        result_0 = comp_wise_sum_0/n0

        # List the nearest neighbour words to that sentence vector
        # (i.e., determine which words in the data have a similar vector
        # representation to the vector for the sentence)

        similar_vec = []
        for i in df:
            comp = df[i]
            value = np.dot(comp, result_0)/(np.linalg.norm(comp) * np.linalg.norm(result_0))
            similar_vec.append((value, i))

        similar_vec.sort(reverse=True)

        k = 1
        print("The nearest neighbour words to the sentence vector: '" + sent_0 + "': ")
        for k in range(1, 6):
            print(similar_vec[k][1],",",similar_vec[k][0])
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/shreyabajpai/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
The nearest neighbour words to the sentence vector: 'scott fitzgerald is very handsome.':
though , 0.8733261596302394
but , 0.8714214559108278
this , 0.8682016931614753
very , 0.8669849373851483
same , 0.8666804874804009
```

3 Task 3: Vector Representation of a Sentence

3.0.1 Choose two other sentences S1 and S2 such that S1 is similar in meaning to S0, and S2 is dissimilar in meaning to S0. Create the sentence vectors using the method from Task 2, and report the cosine similarities between the vectors for S0 and S1, and between the vectors for S0 and S2.

3.0.2 Explain whether the obtained cosine similarity scores are reasonable and give a brief explanation of why or why not.

```
In [7]: import nltk
        import matplotlib
        import math
        from nltk.tokenize import sent_tokenize, word_tokenize
```

```
sent_0 = "Scott Fitzgerald is very handsome.".lower()
sent_1 = "Is Scott Fitzgerald very handsome?".lower()
sent_2 = "We know what we are, but know not what we may be.".lower()

words_0 = word_tokenize(sent_0)
n0 = len(words_0)
words_1 = word_tokenize(sent_1)
n1 = len(words_1)
words_2 = word_tokenize(sent_2)
n2 = len(words_2)

# Sentence 0 vector

sent_0_vec = []

for j in words_0:
    curr = df[j]
    sent_0_vec.append(curr)

comp_wise_sum_0 = np.add.reduce(sent_0_vec)
result_sent_0 = comp_wise_sum_0/n0

# Sentece 1 vector

sent_1_vec = []

for s in words_1:
    curr_1 = df[s]
    sent_1_vec.append(curr_1)

comp_wise_sum_1 = np.add.reduce(sent_1_vec)
result_sent_1 = comp_wise_sum_1/n1

# Sentence 2 vector

sent_2_vec = []

for t in words_2:
    curr_2 = df[t]
    sent_2_vec.append(curr_2)

comp_wise_sum_2 = np.add.reduce(sent_2_vec)
result_sent_2 = comp_wise_sum_2/n2

def cosine_similarity(v1,v2):
    # Cosine similarity of v1 to v2: (v1 dot v2)/{||v1||*||v2||)
    sumxx, sumxy, sumyy = 0, 0, 0
```

```
            for i in range(len(v1)):
                x = v1[i]; y = v2[i]
                sumxx += x*x
                sumyy += y*y
                sumxy += x*y
            return sumxy/math.sqrt(sumxx*sumyy)

        # Cosine similarity between S0 and S1

        vec_0_1 = cosine_similarity(result_sent_0,result_sent_1)
        print ("The cosine similarity between S0 and S1 is", vec_0_1)

        # Cosine similarity between S0 and S2

        vec_0_2 = cosine_similarity(result_sent_0,result_sent_2)
        print ("The cosine similarity between S0 and S2 is", vec_0_2)
```

```
The cosine similarity between S0 and S1 is 0.7353597901147194
The cosine similarity between S0 and S2 is 0.7050390463242394
```