Vector Representation Assignment

April 22, 2018

Justin Chou 156006871
198:439 Intro to Data Science

```
In [1]: #read in data
        file = open("glove.6B/glove.6B.300d.txt","r")
        data = {}
        for line in file:
            word = line.split()
            data[word[0]] = word[1:]
        file.close()
```

1 Task 1

We calculate CosineSimilarity with the following equation:
CosineSimilarity = $\frac{A \cdot B}{\|A\|\|B\|}$

CosineSimilarity is interpreted as -1 meaning exactly opposite, 1 meaning exactly the same, 0 indicating orthogonality (decorrelation)

```
In [2]: #Compute magnitude for each word vector
        magnitude = {}
        for key in data:
            total = 0.0
            for x in range(len(data[key])):
                total += float(data[key][x])**2
            total = total**0.5
            magnitude[key]=total

        #print(magnitude['justin'])
```

```
In [3]: #Compute Cosine Similarity between 'justin' and all words in data

        justin = data['justin']
        justinMag = magnitude['justin']

        CosSim = []
        for key in data:
```

```
            dotProd=0.0
            for x in range(len(data[key])):
                current = data[key]
                dotProd += float(justin[x])*float(current[x])
            similarity = dotProd/(magnitude[key]*justinMag)
            newTuple = (key,similarity)
            CosSim.append(newTuple)

        #print first 3 Cosine Similarities
        #print(CosSim[:3])
```

In [4]:
```
        #Sort words by their CosineSimilarity, output the words with the highest scores

        CosSim.sort(key = lambda tuple:tuple[1], reverse = True)
        for x in range(6):
            print(CosSim[x][0])

        #First word is 'justin', as 'justin' will have a CosineSimilarity with itself of 1
        #The next 5 closest neighbors follow:
```

```
justin
timberlake
bieber
ricky
matthew
langer
```

The five nearest neighbors of "justin" according to CosineSimilarity are timberlake, bieber, ricky, matthew, and langer.

2 Task 2

In [5]:
```
        #using random sentence generator https://randomwordgenerator.com/sentence.php,
        #our random sentence is:
        #"I love eating toasted cheese and tuna sandwiches."

        #Compute Sentence Vector
        sentence = 'I love eating toasted cheese and tuna sandwiches'
        words = sentence.lower().split()
        words.append('.')

        sentenceVector = []
        count = 0
        for word in words:
            #print(word)
            if word not in data:
                continue
```

```
            current = data[word]
            if not sentenceVector:
                for x in range(len(current)):
                    sentenceVector.append(float(current[x]))
            else:
                for x in range(len(current)):
                    sentenceVector[x] += float(current[x])
            count += 1

        #print('Number of words: ' + str(count))
        for x in range(len(sentenceVector)):
            sentenceVector[x] = sentenceVector[x]/count

        #compute sentence Magnitude
        total = 0.0
        for x in range(len(sentenceVector)):
            total += sentenceVector[x]**2
        total = total**0.5
        sentenceMagnitude = total
```

In [6]:
```
        #Compute Cosine Similarity with sentence average vector

        SenCosSim = []
        for key in data:
            dotProd=0.0
            for x in range(len(data[key])):
                current = data[key]
                dotProd += float(sentenceVector[x])*float(current[x])
            similarity = dotProd/(magnitude[key]*sentenceMagnitude)
            newTuple = (key,similarity)
            SenCosSim.append(newTuple)

        #print(SenCosSim[:5])
```

In [7]:
```
        #10 most similar words to random sentence
        SenCosSim.sort(key = lambda tuple:tuple[1], reverse = True)
        for x in range(10):
            print(SenCosSim[x][0])
```

```
cheese
eat
sandwiches
eating
bread
you
sandwich
chicken
```

```
salad
cooked
```

The ten nearest neighbor words to the randoly generated sentence are cheese, eat, sandwiches, eating, bread, you, sandwich, chicken, salad, and cooked.

3 Task 3

```
In [8]: #S1, sentence similar in meaning:
        #"I enjoy consuming roasted cheddar and salmon burgers."

        sentence1 = 'I enjoy consuming roasted cheddar and salmon burgers'

        words1 = sentence1.lower().split()
        words1.append('.')

        sentence1Vector = []
        count = 0
        for word in words1:
            #print(word)
            if word not in data:
                continue

            current = data[word]
            if not sentence1Vector:
                for x in range(len(current)):
                    sentence1Vector.append(float(current[x]))
            else:
                for x in range(len(current)):
                    sentence1Vector[x] += float(current[x])
            count += 1

        #print('Number of words: ' + str(count))
        for x in range(len(sentence1Vector)):
            sentence1Vector[x] = sentence1Vector[x]/count

        #compute sentence1 Magnitude
        total = 0.0
        for x in range(len(sentence1Vector)):
            total += sentence1Vector[x]**2
        total = total**0.5
        sentence1Magnitude = total

        #Computer S0 and S1 CosineSimilarity

        dotProd=0.0
```

```
        for x in range(len(sentenceVector)):
            dotProd += sentenceVector[x]*sentence1Vector[x]
        s0s1similarity = dotProd/(sentenceMagnitude*sentence1Magnitude)
        print(s0s1similarity)
```

```
0.8671338301344433
```

In [9]:
```
        #S2, sentence dissimilar in meaning:
        #"Fish adore devouring me with cooked mozzarella sliders."

        sentence2 = 'Fish adore devouring me with cooked mozzarella sliders'

        words2 = sentence2.lower().split()
        words2.append('.')

        sentence2Vector = []
        count = 0
        for word in words2:
            #print(word)
            if word not in data:
                continue

            current = data[word]
            if not sentence2Vector:
                for x in range(len(current)):
                    sentence2Vector.append(float(current[x]))
            else:
                for x in range(len(current)):
                    sentence2Vector[x] += float(current[x])
            count += 1

        #print('Number of words: ' + str(count))
        for x in range(len(sentence2Vector)):
            sentence2Vector[x] = sentence2Vector[x]/count

        #compute sentence2 Magnitude
        total = 0.0
        for x in range(len(sentence2Vector)):
            total += sentence2Vector[x]**2
        total = total**0.5
        sentence2Magnitude = total

        #Computer S0 and S2 CosineSimilarity

        dotProd=0.0
        for x in range(len(sentenceVector)):
            dotProd += sentenceVector[x]*sentence2Vector[x]
```

```
s0s2similarity = dotProd/(sentenceMagnitude*sentence2Magnitude)
print(s0s2similarity)
```

```
0.8202846252840599
```

The Cosine Similarity between sentence0 and sentence1 is 0.86713, which is reasonably close to 1. The two sentences have nearly interchangable meanings, as most of the words in sentence1 are merely synonyms of those in sentence one. Thus, we should expect the two sentences to have a cosine similarity close to 1.

However, sentence2 has a completely dissimilar meaning from sentence0, despite deliberately choosing similar diction. Sentence0 makes logical sense in an ordinary context: "I love eating toasted cheese and tuna sandwiches." Sentence2, however, does not: "Fish adore devouring me with cooked mozzarella sliders." Even though the second sentence describes carnivorous fish, the calculated cosine similarity is still close to 1: 0.82028, only slightly worse than sentence1. This is because, again, similar diction is used, even though the syntax of the sentence is completely different. We can improve the accuracy of our cosine similarity computation by using an algorithm that accounts for the syntax as well as the diction of a sentence. This would allow us to calculate vector representations that more accurately capture the true meaning of the sentence than simply calculating the average vector of the individual words used.