NLP
Lab-5 Process textual data using TF-IDF

Lab Objectives

Computers are good with numbers, but not that much with textual data. One of the most widely used techniques to process textual data is TF-IDF.

In this lab, we will learn how it works and what its features are.

From our intuition, we think that the words which appear more often should have a greater weight in textual data analysis, but that's not always the case. Words such as "the", "will", and "you" — called **stopwords** — appear the most in a corpus of text, but are of very little significance. Instead, the words which are rare are the ones that actually help in distinguishing between the data, and carry more weight.

An introduction to TF-IDF

TF-IDF stands for "Term Frequency — Inverse Data Frequency". First, we will learn what this term means mathematically.

Term Frequency (tf): gives us the frequency of the word in each document in the corpus. It is the ratio of number of times the word appears in a document compared to the total number of words in that document. It increases as the number of occurrences of that word within the document increases. Each document has its own tf.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Inverse Data Frequency (idf): used to calculate the weight of rare words across all documents in the corpus. The words that occur rarely in the corpus have a high IDF score. It is given by the equation below.

$$idf(w) = log(\frac{N}{df_t})$$

Combining these two we come up with the TF-IDF score (w) for a word in a document in the corpus. It is the product of tf and idf:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

Example:

Consider a document containing 100 words wherein the word *cat* appears 3 times. The term frequency (i.e., tf) for *cat* is then (3 / 100) = 0.03. Now, assume we have 10 million documents and the word *cat* appears in one thousands of these. Then, the inverse document frequency (i.e., idf) is calculated as log (10,000,000 / 1,000) = 4. Thus, the Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.

Example

Let's take an example to get a clearer understanding.

Sentence 1: The car is driven on the road.
Sentence 2: The truck is driven on the highway.

In this example, each sentence is a separate document.
We will now calculate the TF-IDF for the above two documents, which represent our corpus.

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

From the above table, we can see that TF-IDF of common words was zero, which shows they are not significant. On the other hand, the TF-IDF of "car" , "truck", "road", and "highway" are non-zero. These words have more significance.

Using Python to calculate TF-IDF

Let's now code TF-IDF in Python from scratch. After that, we will see how we can use sklearn to automate the process.

450 lines (449 sloc) | 9.88 KB — Raw | Blame | History

In [54]:
```
docA = "The cat sat on my face"
docB = "The dog sat on my bed"
```

In [55]:
```
bowA = docA.split(" ")
bowB = docB.split(" ")
```

In [56]:
```
bowB
```

Out[56]:
```
['The', 'dog', 'sat', 'on', 'my', 'bed']
```

In [57]:
```
wordSet = set(bowA).union(set(bowB))
```

In [58]:
```
wordSet
```

Out[58]:
```
{'The', 'bed', 'cat', 'dog', 'face', 'my', 'on', 'sat'}
```

In [59]:
```
wordDictA = dict.fromkeys(wordSet, 0)
wordDictB = dict.fromkeys(wordSet, 0)
```

In [60]:
```
wordDictA
```

Out[60]:
```
{'The': 0, 'cat': 0, 'bed': 0, 'dog': 0, 'my': 0, 'face': 0, 'sat': 0, 'on': 0}
```

In [61]:
```
for word in bowA:
    wordDictA[word]+=1

for word in bowB:
    wordDictB[word]+=1
```

In [62]:
```
wordDictA
```

Out[62]:
```
{'The': 1, 'cat': 1, 'bed': 0, 'dog': 0, 'my': 1, 'face': 1, 'sat': 1, 'on': 1}
```

In [63]:
```
import pandas as pd
pd.DataFrame([wordDictA, wordDictB])
```

Out[63]:

	The	bed	cat	dog	face	my	on	sat
0	1	0	1	0	1	1	1	1
1	1	1	0	1	0	1	1	1

In [64]:
```
def computeTF(wordDict, bow):
    tfDict = {}
    bowCount = len(bow)
    for word, count in wordDict.items():
        tfDict[word] = count/float(bowCount)
    return tfDict
```

In [65]:
```
tfBowA = computeTF(wordDictA, bowA)
tfBowB = computeTF(wordDictB, bowB)
```

In [66]:
```
tfBowA
```

Out[66]:
```
{'The': 0.16666666666666666,
 'cat': 0.16666666666666666,
 'bed': 0.0,
 'dog': 0.0,
 'my': 0.16666666666666666,
 'face': 0.16666666666666666,
 'sat': 0.16666666666666666,
 'on': 0.16666666666666666}
```

In [67]:
```
tfBowB
```

Out[67]:
```
{'The': 0.16666666666666666,
 'cat': 0.0,
 'bed': 0.16666666666666666,
 'dog': 0.16666666666666666,
 'my': 0.16666666666666666,
 'face': 0.0,
 'sat': 0.16666666666666666,
 'on': 0.16666666666666666}
```

In [75]:
```
def computeIDF(docList):
    import math
    idfDict = {}
    N = len(docList)

    idfDict = dict.fromkeys(docList[0].keys(), 0)
    for doc in docList:
        for word, val in doc.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log10(N / float(val))

    return idfDict
```

In [79]:
```
idfs = computeIDF([wordDictA, wordDictB])
```

In [78]:
```
def computeTFIDF(tfBow, idfs):
    tfidf = {}
    for word, val in tfBow.items():
        tfidf[word] = val*idfs[word]
    return tfidf
```

In [80]:
```
tfidfBowA = computeTFIDF(tfBowA, idfs)
tfidfBowB = computeTFIDF(tfBowB, idfs)
```

In [83]:
```
import pandas as pd
pd.DataFrame([tfidfBowA, tfidfBowB])
```

Out[83]:

	The	bed	cat	dog	face	my	on	sat
0	0.0	0.000000	0.050172	0.000000	0.050172	0.0	0.0	0.0
1	0.0	0.050172	0.000000	0.050172	0.000000	0.0	0.0	0.0

Github link of the above code:

https://github.com/mayank408/TFIDF/blob/master/TFIDF.ipynb

sklearn

Now we will see how we can implement this using sklearn in Python.

Example#1:

```
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = ["The car is driven on the road",

        "The truck is driven on the highway"]

vectorizer = TfidfVectorizer()

X = vectorizer.fit_transform(corpus)

idf = vectorizer.idf_

# X has a dimension of 2 (document count) by 8 (# of unique words)

print (X.shape)  #(2,8)

print (vectorizer.get_feature_names())

print (dict(zip(vectorizer.get_feature_names(), idf)))
```

The tf-idf score of each feature can be retrieved via the attribute `idf_` of the `TfidfVectorizer` object:

The output:

```
['car', 'driven', 'highway', 'is', 'on', 'road', 'the', 'truck']

{'car': 1.4054651081081644, 'driven': 1.0, 'highway': 1.405
4651081081644, 'is': 1.0, 'on': 1.0, 'road': 1.405465108108
1644, 'the': 1.0, 'truck': 1.4054651081081644}
```

Example#2:

```
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = ["This is very strange",

        "This is very nice"]

vectorizer = TfidfVectorizer()

X = vectorizer.fit_transform(corpus)

idf = vectorizer.idf_

print (dict(zip(vectorizer.get_feature_names(), idf)))
```

The output:

```
{'is': 1.0, 'nice': 1.4054651081081644, 'strange':
1.4054651081081644, 'this': 1.0, 'very': 1.0}
```

References:

https://medium.freecodecamp.org/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3

https://github.com/mayank408/TFIDF/blob/master/TFIDF.ipynb

https://scikitlearn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html