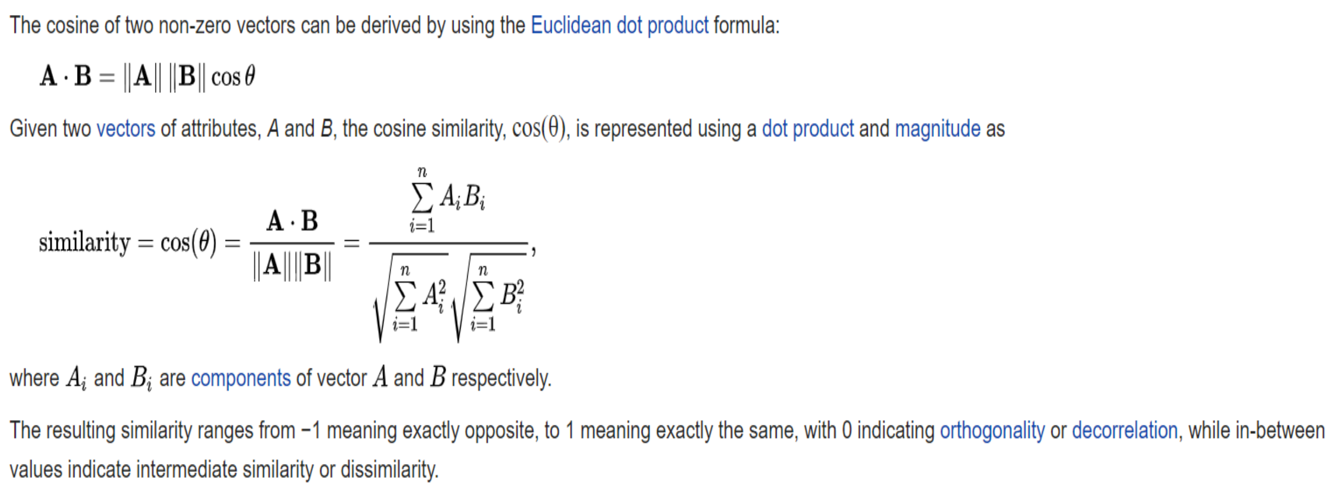
Cosine Similarity

Cosine similarity is a metric used to determine how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. In this context, the two vectors I am talking about are arrays containing the word counts of two documents.

As a similarity metric, how does cosine similarity differ from the number of common words?
When plotted on a multi-dimensional space, where each dimension corresponds to a word in the document, the cosine similarity captures the orientation (the angle) of the documents and not the magnitude. If you want the magnitude, compute the Euclidean distance instead.
The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size (like, the word 'cricket' appeared 50 times in one document and 10 times in another) they could still have a smaller angle between them. Smaller the angle, the higher the similarity.



The cosine of two non-zero vectors can be derived by using the Euclidean dot product formula:

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \, \|\mathbf{B}\| \cos\theta$$

Given two vectors of attributes, *A* and *B*, the cosine similarity, $\cos(\theta)$, is represented using a dot product and magnitude as

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

where A_i and B_i are components of vector A and B respectively.

The resulting similarity ranges from −1 meaning exactly opposite, to 1 meaning exactly the same, with 0 indicating orthogonality or decorrelation, while in-between values indicate intermediate similarity or dissimilarity.

Note: To use you need a bag of words

What is Bag-of-Words?

We need a way to represent text data for the machine learning algorithm and the bag-of-words model helps us to achieve that task. It is a way of extracting features from the text for use in machine learning algorithms.

In this approach, we use the tokenized words for each observation and find out the frequency of each token.

Let’s take an example to understand this concept in depth.

“It was the best of times”, “It was the worst of times”, “It was the age of wisdom”, “It was the age of foolishness”

We treat each sentence as a separate document and we make a list of all words from all the four documents excluding the punctuation. We get,

‘It’, ‘was’, ‘the’, ‘best’, ‘of’, ‘times’, ‘worst’, ‘age’, ‘wisdom’, ‘foolishness’

The next step is the create vectors. Vectors convert text that can be used by the machine learning algorithm.

We take the first document — *“It was the best of times”* and we check the frequency of words from the 10 unique words.

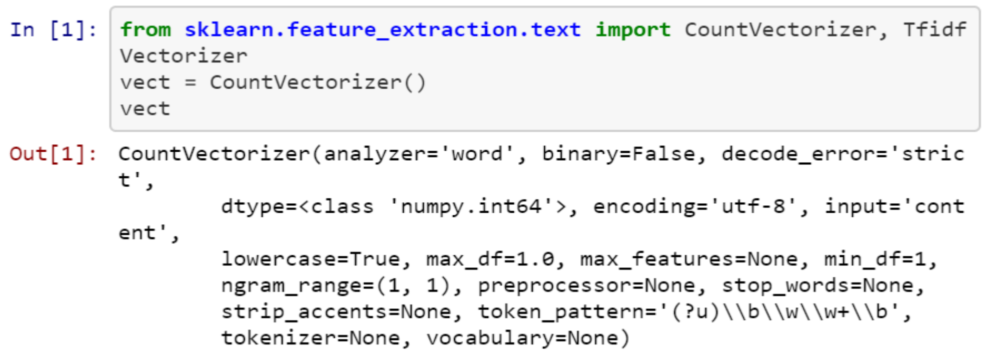
“it” = 1, “was” = 1, “the” = 1, “best” = 1, “of” = 1, “times” = 1, “worst” = 0, “age” = 0, “wisdom” = 0, “foolishness” = 0

Rest of the documents will be:
“It was the best of times” = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
“It was the worst of times” = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
“It was the age of wisdom” = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
“It was the age of foolishness” = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]
In this approach, each word or token is called a “gram”. Creating a vocabulary of two-word pairs is called a bigram model.
For example, the bigrams in the first document: “It was the best of times” are as follows:
“it was”, “was the”, “the best”, “best of”, “of times”
Similarly can also use trigrams

CountVectorizer

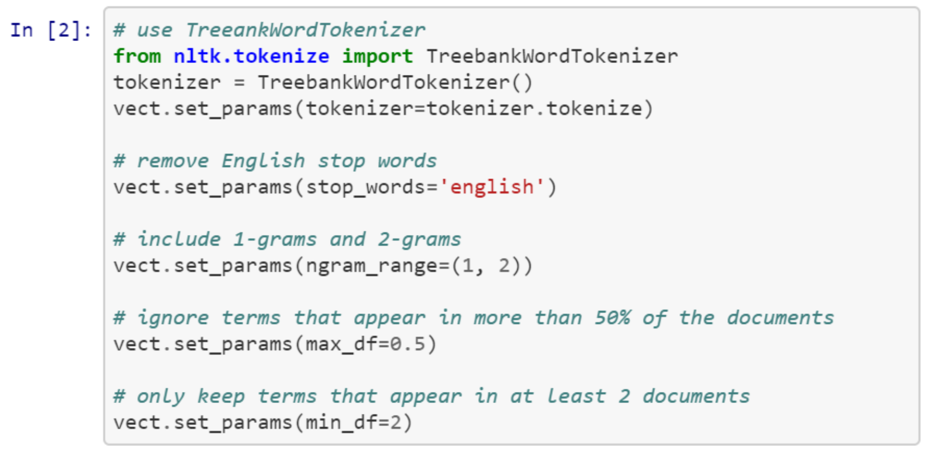
CountVectorizer works on Terms Frequency, i.e. counting the occurrences of tokens and building a sparse matrix of documents x tokens.

Sample code:



```
In [1]: from sklearn.feature_extraction.text import CountVectorizer, Tfidf
        Vectorizer
        vect = CountVectorizer()
        vect
```

```
Out[1]: CountVectorizer(analyzer='word', binary=False, decode_error='stric
        t',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='cont
        ent',
                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```



In [2]:

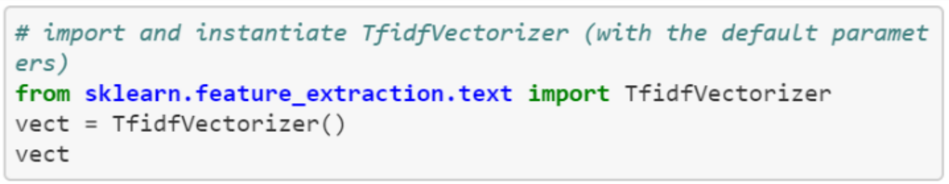
```
# use TreeankWordTokenizer
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
vect.set_params(tokenizer=tokenizer.tokenize)

# remove English stop words
vect.set_params(stop_words='english')

# include 1-grams and 2-grams
vect.set_params(ngram_range=(1, 2))

# ignore terms that appear in more than 50% of the documents
vect.set_params(max_df=0.5)

# only keep terms that appear in at least 2 documents
vect.set_params(min_df=2)
```

TF-IDF Vectorizer

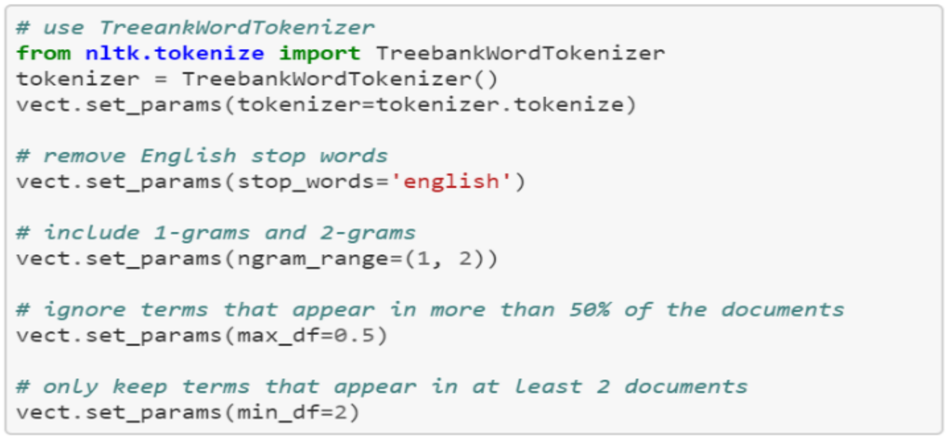
TF-IDF stands for term frequency-inverse document frequency. TF-IDF weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Whereas...

1. **Term Frequency (TF)**: is a scoring of the frequency of the word in the current document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. The term frequency is often divided by the document length to normalize.
2. **Inverse Document Frequency (IDF)**: is a scoring of how rare the word is across documents. IDF is a measure of how rare a term is. Rarer the term, more is the IDF score.

Sample Code:



```
# import and instantiate TfidfVectorizer (with the default paramet
ers)
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer()
vect
```



```
# use TreeankWordTokenizer
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
vect.set_params(tokenizer=tokenizer.tokenize)

# remove English stop words
vect.set_params(stop_words='english')

# include 1-grams and 2-grams
vect.set_params(ngram_range=(1, 2))

# ignore terms that appear in more than 50% of the documents
vect.set_params(max_df=0.5)

# only keep terms that appear in at least 2 documents
vect.set_params(min_df=2)
```

# code and example - *From StackOverflow*

```
import re, math
from collections import Counter

WORD = re.compile(r'\w+')

def get_cosine(vec1, vec2):
     intersection = set(vec1.keys()) & set(vec2.keys())
     numerator = sum([vec1[x] * vec2[x] for x in intersection])

     sum1 = sum([vec1[x]**2 for x in vec1.keys()])
     sum2 = sum([vec2[x]**2 for x in vec2.keys()])
     denominator = math.sqrt(sum1) * math.sqrt(sum2)

     if not denominator:
        return 0.0
     else:
        return float(numerator) / denominator

def text_to_vector(text):
     words = WORD.findall(text)
     return Counter(words)

text1 = 'This is a foo bar sentence .'
text2 = 'This sentence is similar to a foo bar sentence .'

vector1 = text_to_vector(text1)
vector2 = text_to_vector(text2)

cosine = get_cosine(vector1, vector2)

print 'Cosine:', cosine
```

Prints:

```
Cosine: 0.861640436855
```

#using library

```
# Compute Cosine Similarity
from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(df, df))
```