Text Feature Extraction

- scikit-learn provides utilities for the most common ways to extract numerical features from text content:
  - tokenizing strings and giving an integer id for each possible token
  - counting the occurrences of tokens in each document
  - normalizing and weighting with diminishing importance tokens that occur in the majority of samples / documents.

- Features and samples are defined as follows:
  - each individual token occurrence frequency (normalized or not) is treated as a feature.
  - the vector of all the token frequencies for a given document is considered a multivariate sample.

- A corpus of documents can thus be represented by a matrix with one row per document and one column per token (e.g. word) occurring in the corpus.
- Vectorization is the general process of turning a collection of text documents into numerical feature vectors.
- This strategy (tokenization, counting and normalization) is called the Bag of Words or “Bag of n-grams” representation.
- Documents are described by word occurrences while completely ignoring the relative position information of the words in the document.

In [1]:
```
from sklearn.feature_extraction.text import CountVectorizer

from sklearn.feature_extraction.text import TfidfVectorizer
```

- CountVectorizer implements both tokenization and occurrence counting

In [2]:
```
corpus = [
      'This is the first document.',
      'This document is the second document.',
      'And this is the third one.',
      'Is this the first document?',
]
```

In [3]:
```
vectorizer1 = CountVectorizer()
```

In [4]:
```
X = vectorizer1.fit_transform(corpus)
```

```
In [5]: X
```

```
Out[5]: <4x9 sparse matrix of type '<class 'numpy.int64'>'
                with 21 stored elements in Compressed Sparse Row format>
```

```
In [6]: vectorizer1.get_feature_names()
```

```
Out[6]: ['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

```
In [7]: print(X.toarray())
```

```
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

```
In [8]: vectorizer1.vocabulary_.get('document')
```

```
Out[8]: 1
```

```
In [9]: vectorizer1.transform(['Hello how is everyone?']).toarray()
```

```
Out[9]: array([[0, 0, 0, 1, 0, 0, 0, 0, 0]])
```

```
In [10]: vectorizer1.vocabulary_.get('is')
```

```
Out[10]: 3
```

- the first and the last documents have exactly the same words hence are encoded in equal vectors
- extract 2-grams of words in addition to the 1-grams (individual words)

```
In [11]: bigram_vectorizer = CountVectorizer(
             ngram_range=(1, 2))
```

```
In [12]: X2 = bigram_vectorizer.fit_transform(corpus)
         X2
```

```
Out[12]: <4x22 sparse matrix of type '<class 'numpy.int64'>'
                with 39 stored elements in Compressed Sparse Row format>
```

In [13]:
```
bigram_vectorizer.get_feature_names()
```

Out[13]:
```
['and',
 'and this',
 'document',
 'document is',
 'first',
 'first document',
 'is',
 'is the',
 'is this',
 'one',
 'second',
 'second document',
 'the',
 'the first',
 'the second',
 'the third',
 'third',
 'third one',
 'this',
 'this document',
 'this is',
 'this the']
```

In [14]:
```
print(X2.toarray())
```

```
[[0 0 1 0 1 1 1 1 0 0 0 0 1 1 0 0 0 0 1 0 1 0]
 [0 0 2 1 0 0 1 1 0 0 1 1 1 0 1 0 0 0 1 1 0 0]
 [1 1 0 0 0 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0]
 [0 0 1 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 1]]
```

In [ ]:

- Tf-idf term weighting
  - In a large text corpus, some words will be very present (e.g. “the”, “a”, “is” in English) hence carrying very little meaningful information about the actual contents of the document. If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms.
  - In order to re-weight the count features into floating point values suitable for usage by a classifier it is very common to use the tf–idf transform.
  - Tf means term-frequency while tf–idf means term-frequency times inverse document-frequency:

$$\text{tf-idf(t,d)} = \text{tf(t,d)} \times \text{idf(t)}$$

$$\text{idf}(t) = \log \frac{n}{1+\text{df}(t)}$$

In [ ]:

In [15]:
```
vectorizer2 = TfidfVectorizer()
```

In [16]:
```
X3 = vectorizer2.fit_transform(corpus)
```

In [17]:
```
X3
```

Out[17]:
```
<4x9 sparse matrix of type '<class 'numpy.float64'>'
	with 21 stored elements in Compressed Sparse Row format>
```

In [18]:
```
vectorizer2.get_feature_names()
```

Out[18]:
```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

In [19]:
```
print(X3.toarray().round(2))
```

```
[[0.   0.47 0.58 0.38 0.   0.   0.38 0.   0.38]
 [0.   0.69 0.   0.28 0.   0.54 0.28 0.   0.28]
 [0.51 0.   0.   0.27 0.51 0.   0.27 0.51 0.27]
 [0.   0.47 0.58 0.38 0.   0.   0.38 0.   0.38]]
```

In [ ]: