Text Data - Natural Language Processing (NLP)

Text data usually consists of a collection of documents (called the corpus) which can represent words, sentences, or even paragraphs of free flowing text.

The inherent unstructured (no neatly formatted data columns!) and noisy nature of textual data makes it harder for machine learning methods to directly work on raw text data.

Feature Engineering

Feature engineering dramatically improve performance of machine learning models and wins Kaggle competitions. This is especially true for text data, which is unstructured, noisy, and complex.

This section will cover the following types of features for text data

1. Bag of Words
2. Bag of N-Grams (uni-gram, bi-gram, tri-gram, etc.)
3. TF-IDF (term frequency over inverse document frequency)

In [1]:
```
import pandas as pd
import numpy as np
import re
import nltk
import matplotlib.pyplot as plt
nltk.download('stopwords')
```

Out[1]:
```
[nltk_data] Downloading package stopwords to /home/magni/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

In [2]:
```
corpus = ['The sky is blue and beautiful.',
          'Love this blue and beautiful sky!',
          'The quick brown fox jumps over the lazy dog.',
          'The brown fox is quick and the blue dog is lazy!',
          'The sky is very blue and the sky is very beautiful today',
          'The dog is lazy but the brown fox is quick!'
]

labels = ['weather', 'weather', 'animals', 'animals', 'weather', 'animals']
corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus,
                          'Category': labels})
corpus_df = corpus_df[['Document', 'Category']]
corpus_df
```

Out[2]:

	Document	Category

	Document	Category
0	The sky is blue and beautiful.	weather
1	Love this blue and beautiful sky!	weather
2	The quick brown fox jumps over the lazy dog.	animals
3	The brown fox is quick and the blue dog is lazy!	animals
4	The sky is very blue and the sky is very beaut...	weather

In [3]:
```
display(type(corpus))
display(corpus)
```

```
numpy.ndarray
array(['The sky is blue and beautiful.',
       'Love this blue and beautiful sky!',
       'The quick brown fox jumps over the lazy dog.',
       'The brown fox is quick and the blue dog is lazy!',
       'The sky is very blue and the sky is very beautiful today',
       'The dog is lazy but the brown fox is quick!'], dtype='<U56')
```

In [4]:
```
corpus_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Document  6 non-null      object
 1   Category  6 non-null      object
dtypes: object(2)
memory usage: 224.0+ bytes
```

Text pre-processing

Depending on your downstream task, cleaning and pre-processing text can involve several different components. Here are a few important components of Natural Language Processing (NLP) pipelines.

1. Removing tags: unnecessary content like HTML tags
2. Removing accented characters: other languages such as French, convert ASCII
3. Removing special characters: adds noise to text, use simple regular expressions (regexes)
4. Stemming and lemmatization: Stemming remove prefixes and suffixes of word stems (i.e. root words), ex. WATCH is the root stem of WATCHES, WATCHING, and WATCHE. Lemmatization similar but lexicographically correct word (present in the dictionary).
5. Expanding contractions: helps text standardization, ex. do not to don't and I would to I'd
6. Removing stopwords: Words without meaningful significance (ex. a, an, the, and) but high frequency.

Additional pre-processing: tokenization, removing extra whitespaces, lower casing and more

advanced operations like spelling corrections, grammatical error corrections, removing repeated characters.

In [5]:
```
wpt = nltk.WordPunctTokenizer()
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
    # lower case and remove special characters\whitespaces
    doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I)
    doc = doc.lower()
    doc = doc.strip()
    # tokenize document
    tokens = wpt.tokenize(doc)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)
```

In [6]:
```
norm_corpus = normalize_corpus(corpus)
norm_corpus
```

Out[6]:
```
array(['sky blue beautiful', 'love blue beautiful sky',
       'quick brown fox jumps lazy dog', 'brown fox quick blue dog lazy',
       'sky blue sky beautiful today', 'dog lazy brown fox quick'],
      dtype='<U30')
```

1. Bag of Words Model

This is perhaps the most simple vector space representational model for unstructured text. A vector space model is simply a mathematical model to represent unstructured text (or any other data) as numeric vectors, such that each dimension of the vector is a specific feature\attribute. The bag of words model represents each text document as a numeric vector where each dimension is a specific word from the corpus and the value could be its frequency in the document, occurrence (denoted by 1 or 0) or even weighted values. The model's name is such because each document is represented literally as a 'bag' of its own words, disregarding word orders, sequences and grammar.

In [7]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(norm_corpus)
cv_matrix = cv_matrix.toarray()
cv_matrix
```

Out[7]:
```
array([[1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0],
       [0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0],
```

```
       [1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 1],
       [0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0]])
```

Thus you can see that our documents have been converted into numeric vectors such that each document is represented by one vector (row) in the above feature matrix. The following code will help represent this in a more easy to understand format.

In [8]:
```
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
pd.DataFrame(cv_matrix, columns=vocab)
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[8]:

	beautiful	blue	brown	dog	fox	jumps	lazy	love	quick	sky	today
0	1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	1	0	1	0
2	0	0	1	1	1	1	1	0	1	0	0
3	0	1	1	1	1	0	1	0	1	0	0
4	1	1	0	0	0	0	0	0	0	2	1
5	0	0	1	1	1	0	1	0	1	0	0

This should make things more clearer! You can clearly see that each column or dimension in the feature vectors represents a word from the corpus and each row represents one of our documents. The value in any cell, represents the number of times that word (represented by column) occurs in the specific document (represented by row). Hence if a corpus of documents consists of N unique words across all the documents, we would have an N-dimensional vector for each of the documents.

This should make things more clearer! You can clearly see that each column or dimension in the feature vectors represents a word from the corpus and each row represents one of our documents. The value in any cell, represents the number of times that word (represented by column) occurs in the specific document (represented by row). Hence if a corpus of documents consists of N unique words across all the documents, we would have an N-dimensional vector for each of the documents.

2. Bag of N-Grams Model¶

A word is just a single token, often known as a unigram or 1-gram. We already know that the Bag of Words model doesn't consider order of words. But what if we also wanted to take into account phrases or collection of words which occur in a sequence? N-grams help us achieve that. An N-gram is basically a collection of word tokens from a text document such that these tokens are

contiguous and occur in a sequence. Bi-grams indicate n-grams of order 2 (two words), Tri-grams indicate n-grams of order 3 (three words), and so on. The Bag of N-Grams model is hence just an extension of the Bag of Words model so we can also leverage N-gram based features. The following example depicts bi-gram based features in each document feature vector.

In [9]:
```
# you can set the n-gram range to 1,2 to get unigrams as well as bigrams
bv = CountVectorizer(ngram_range=(2,2))
bv_matrix = bv.fit_transform(norm_corpus)

bv_matrix = bv_matrix.toarray()
vocab = bv.get_feature_names()
pd.DataFrame(bv_matrix, columns=vocab)
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[9]:

	beautiful sky	beautiful today	blue beautiful	blue dog	blue sky	brown fox	dog lazy	fox jumps	fox quick	jumps lazy	lazy brown	lazy dog	love blue
0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	1	0	1	0	1	0	1	0
3	0	0	0	1	0	1	1	0	1	0	0	0	0
4	0	1	0	0	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	1	0	1	0	1	0	0

3. TF-IDF Model

There are some potential problems which might arise with the Bag of Words model when it is used on large corpora. Since the feature vectors are based on absolute term frequencies, there might be some terms which occur frequently across all documents and these may tend to overshadow other terms in the feature set. The TF-IDF model tries to combat this issue by using a scaling or normalizing factor in its computation. TF-IDF stands for Term Frequency-Inverse Document Frequency, which uses a combination of two metrics in its computation, namely: term frequency (tf) and inverse document frequency (idf). This technique was developed for ranking results for queries in search engines and now it is an indispensable model in the world of information retrieval and NLP.

Mathematically, we can define TF-IDF as tfidf = tf x idf, which can be expanded further to be represented as follows.

Here, tfidf(w, D) is the TF-IDF score for word w in document D. The term tf(w, D) represents the term frequency of the word w in document D, which can be obtained from the Bag of Words model. The term idf(w, D) is the inverse document frequency for the term w, which can be

computed as the log transform of the total number of documents in the corpus C divided by the document frequency of the word w, which is basically the frequency of documents in the corpus where the word w occurs. There are multiple variants of this model but they all end up giving quite similar results. Let's apply this on our corpus now!

In [10]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(norm_corpus)
tv_matrix = tv_matrix.toarray()

vocab = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)
```

```
/home/magni/python_env/ML1010_env2/lib64/python3.7/site-packages/sklearn/util
s/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated;
get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use
get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[10]:

	beautiful	blue	brown	dog	fox	jumps	lazy	love	quick	sky	today
0	0.60	0.52	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00
1	0.46	0.39	0.00	0.00	0.00	0.00	0.00	0.66	0.00	0.46	0.00
2	0.00	0.00	0.38	0.38	0.38	0.54	0.38	0.00	0.38	0.00	0.00
3	0.00	0.36	0.42	0.42	0.42	0.00	0.42	0.00	0.42	0.00	0.00
4	0.36	0.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.72	0.52
5	0.00	0.00	0.45	0.45	0.45	0.00	0.45	0.00	0.45	0.00	0.00

The TF-IDF based feature vectors for each of our text documents show scaled and normalized values as compared to the raw Bag of Words model values. Interested readers who might want to dive into further details of how the internals of this model work can refer to page 181 of Text Analytics with Python (Springer\Apress; Dipanjan Sarkar, 2016).