Introduction to Text Analysis

Credits: Following lines have been taken from NLP guide available online at kaggle.com/learn-guide/natural-language-processing

- NLP a subfield of machine learning concerned with building models to demonstrate human-level understanding of written and spoken text.
- It is one of the areas where deep learning has dramatically improved in last couple of years.
- With advancements in this area computers can now:
  1. generate text
  2. translate automatically from one language to another
  3. analyze comments
  4. label words in sentences
- The most widely practical application of NLP is classification (Automatically classifying a document into some category) which can be used for:
  1. Sentiment analysis (e.g. are people saying +ve things or -ve things about a given product / service)
  2. Author identification (what author most likely wrote some document)
  3. legal discovery (which documents are in scope for a trial)
  4. organizing documents by topic
  5. Assigning degrees of urgency to emails received

Stemming and Lemmatization

Credits: Following explanation is reproduced from online page of Stanford NLP Group

- For grammatical reasons, documents use different forms of a word e.g. organize, organizes, and organizing
- Additionally families of derivationally related words with similar meanings exist e.g. democracy, democratic, democratization
- In many situations it would be useful for a search for one of these words to return documents that contain another word in the set
- The goal of both **stemming** and **lemmatization** is to reduce inflectional/ derivationally related forms of a word to a common base form. E.g. {am, are is -> be }, {car, cars, car's, cars' -> car}
- The result of this mapping of text will be something like: the boy's cars are different colors -> the boy car be differ color
- Stemming usually refers to a crude heuristic process that chops off the ends of words in of achieving this goal correctly most of the time.

- lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.
- The most common and empirically effective algorithm for stemming English is Porter's Algorithm Ba

Bag of words

Credits: Following contents have been taken from Victor Zhou post 'A Simple Explanation of the Bag of Words Model' available online on towardsdatascience.com

- The bag of words(BOW) model is a representation that turns arbitrary text into fixed-length vectors by counting how many times each word appears. This process is often referred to as vectorization.

Example: Suppose we wanted to vectorize the following: - the cat sat - the cat sat in the hat - the cat with the hat

1. Consider each of these lines to be a text document
2. Define vocabulary: set of all words found in our document set.
3. Vocabulary = {the, cat, sat, in, the, hat, with}
4. Count how many times each word appears

Document	the	cat	sat	in	hat	with
Doc 1	1	1	1	0	0	0
Doc 2	2	1	1	1	1	0
Doc 3	2	1	0	0	1	1

- We have length 6 vectors for each document

Doc 1: [1, 1, 1, 0, 0, 0] Doc 2: [2, 1, 1, 1, 1, 0] Doc 3: [2, 1, 0, 0, 1, 1]

- **Notice that the contextual information is lost e.g. where in the document the word appeared**
- **BOW only tells what words occur in the document not where they occurred.**

BOW in Python

```
from keras.preprocessing.text import Tokenizer

docs = [
  'the cat sat',
    'the cat sat in the hat',
    ' the cat with the hat'
    ]

    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(docs)
    print(f'Vocabulary: {list(tokenizer.word_index.keys())}')

    vectors = tokenizer.texts_to_matrix(docs, mode='count')
    print(vectors)
```

The output of the above code will give vectors of length 7 instead of 6 because Keras reserves index 0 and never assigns it to any word

TF-IDF (Term frequency - inverse document frequency)

- When performing text vectorization, one common approach is to use a vector of word counts.
- Instead of weighting each word by its count within the document, though, another approach that's often more helpful is to weight each word by its tf-idf score, defined as follows:

1. tf(t, d) = # of times term t appears in document d
2. idf(t, D) = log (total number of documents / total # of documents in which term t appears)
3. tf-idf(t, d, D) = tf(t, d) * idf(t, D)

For example, given the following corpus:

Document 1: The quick brown fox
Document 2: The brown brown dog
Document 3: The fox ate the dog

The tf-idf score of "brown" within Document 2 is:

tf(brown, Document 2) = 2
idf(brown, corpus) = log (3 / 2)
tf-idf(brown, Document 2, corpus) = 2 * log (3 / 2)

- Weighting terms by the number of times they appear in a document is often suboptimal.
- For example, words like “the” can appear frequently within a document, but are relatively meaningless.
- Tf-idf scores help mitigate this problem by also taking into account a term’s frequency across the entire corpus, not just within the document itself.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
corpus = [
 'the brown fox jumped over the brown dog',
 'the quick brown fox',
 'the brown brown dog',
 'the fox ate the dog'
]
X = vectorizer.fit_transform(corpus)
print(vectorizer.get_feature_names())
print(X.toarray())
```