Josh Bohde

Document Summarization using TextRank

Posted 2012-09-02 by Josh Bohde

For a gift recommendation side-project of mine, I wanted to do some automatic summarization for products. A fairly easy way to do this is TextRank, based upon PageRank. In this example, the vertices of the graph are sentences, and the edge weights between sentences are how similar the sentences are.

To go from an string of text to a list of scored sentences based upon how much they represent the overall text, we need to go through several steps:

1. Tokenize the text into sentences
2. Tokenize each sentence into a collection of words
3. Convert the sentences into graphs
4. Score the sentences via pagerank

Sentence Splitting

To get started, we'll need to split the document into sentences. For this, we'll use nltk's included Punkt module, and the opening paragraph of Sir Authur Conan Doyle's *A Scandal In Bohemia*.

```
>>> from nltk.tokenize.punkt import PunktSentenceTokenizer

>>> document = """To Sherlock Holmes she is always the woman. I have
seldom heard him mention her under any other name. In his eyes she
eclipses and predominates the whole of her sex. It was not that he
felt any emotion akin to love for Irene Adler. All emotions, and that
one particularly, were abhorrent to his cold, precise but admirably
balanced mind. He was, I take it, the most perfect reasoning and
observing machine that the world has seen, but as a lover he would
have placed himself in a false position. He never spoke of the softer
passions, save with a gibe and a sneer. They were admirable things for
the observer-excellent for drawing the veil from men’s motives and
actions. But for the trained reasoner to admit such intrusions into
his own delicate and finely adjusted temperament was to introduce a
distracting factor which might throw a doubt upon all his mental
results. Grit in a sensitive instrument, or a crack in one of his own
high-power lenses, would not be more disturbing than a strong emotion
in a nature such as his. And yet there was but one woman to him, and
that woman was the late Irene Adler, of dubious and questionable
memory.
"""
>>> document = ' '.join(document.strip().split('\n'))

>>> sentence_tokenizer = PunktSentenceTokenizer()
>>> sentences = sentence_tokenizer.tokenize(document)
```

Bag of Words

Before we can do any analysis, we need to convert the document into a bag-of-words, which is an unordered collection of word counts.

For the first sentence, we should have something like the following:

```
{'always': 1,
```

```
 'holmes': 1,
 'is': 1,
 'she': 1,
 'sherlock': 1,
 'the': 1,
 'to': 1,
 'woman': 1}
```

A simple implementation for converting a sentence to a bag-of-words can be done via the base Python datastructures:

```
from collections import Counter

def bag_of_words(sentence):
    return Counter(word.lower().strip('.,') for word in sentence.split(' '))
```

An alternative way to a bag-of-words is in a vector, where each column corresponds to a word. Instead of the previous dictionary, we'd have something like the following:

```
[1, 1, 1, 1, 1, 1, 1, 1]
```

The package **scikit-learn** provides text feature extraction, allowing us to build SciPy matrices out of a collections of texts.

```
>>> from sklearn.feature_extraction.text import CountVectorizer

>>> c = CountVectorizer()
>>> bow_array = c.fit_transform([sentences[0]])
>>> bow_array.toarray()
[[1 1 1 1 1 1 1 1]]
```

The resulting array is a bit different, since hyphens are not included in words, and the stopword "a" is excluded.

We can apply this to the entire collection, and get back a matrix.

```
>>> from sklearn.feature_extraction.text import CountVectorizer

>>> c = CountVectorizer()
>>> bow_matrix = c.fit_transform(sentences)
>>> bow_matrix
<11x127 sparse matrix of type '<type 'numpy.float64'>'
   with 183 stored elements in COOrdinate format>
```

Converting to a Graph

Now we have a matrix where the rows are sentences and the columns are words. We need to transform this into a graph relating the sentences to each other. To do this, we'll first normalize our matrix using Scikit-learn's `TfidfTransformer`.

```
>>> from sklearn.feature_extraction.text import TfidfTransformer
>>> normalized_matrix = TfidfTransformer().fit_transform(bow_matrix)
```

This will re-weight each word is based upon its **tf-idf**, which will dimish the effect of words common to each sentence.

```
>>> similarity_graph = normalized_matrix * normalize_matrix.T
>>> similarity_graph.toarray()
array([[ 1.        ,  0.        ,  0.13737879,  0.04767903,  0.04305016,
         0.04345599,  0.03330044,  0.05261648,  0.07798958,  0.        ,
         0.20047419],
       [ 0.        ,  1.        ,  0.0842143 ,  0.07819597,  0.        ,
         0.05171612,  0.        ,  0.        ,  0.        ,  0.        ,
         0.05807146],
       [ 0.13737879,  0.0842143 ,  1.        ,  0.        ,  0.07004069,
         0.09648614,  0.1069042 ,  0.06701793,  0.09437203,  0.20474295,
         0.1197599 ],
       [ 0.04767903,  0.07819597,  0.        ,  1.        ,  0.07558987,
         0.18678911,  0.05853972,  0.09249592,  0.10892262,  0.09110741,
         0.24159019],
       [ 0.04305016,  0.        ,  0.07004069,  0.07558987,  1.        ,
         0.07055583,  0.02370685,  0.07272032,  0.17253418,  0.08262451,
         0.17789849],
       [ 0.04345599,  0.05171612,  0.09648614,  0.18678911,  0.07055583,
         1.        ,  0.12952649,  0.06859301,  0.06837492,  0.13015945,
         0.15423071],
       [ 0.03330044,  0.        ,  0.1069042 ,  0.05853972,  0.02370685,
         0.12952649,  1.        ,  0.06307559,  0.03194234,  0.02852116,
         0.11271501],
       [ 0.05261648,  0.        ,  0.06701793,  0.09249592,  0.07272032,
         0.06859301,  0.06307559,  1.        ,  0.09411725,  0.        ,
         0.07702234],
       [ 0.07798958,  0.        ,  0.09437203,  0.10892262,  0.17253418,
         0.06837492,  0.03194234,  0.09411725,  1.        ,  0.12388421,
         0.14327969],
       [ 0.        ,  0.        ,  0.20474295,  0.09110741,  0.08262451,
         0.13015945,  0.02852116,  0.        ,  0.12388421,  1.        ,
         0.04706138],
       [ 0.20047419,  0.05807146,  0.1197599 ,  0.24159019,  0.17789849,
         0.15423071,  0.11271501,  0.07702234,  0.14327969,  0.04706138,
         1.        ]])
```

This is a mirrored matrix, where both the rows and columns correspond to sentences, and the elements describe how similar the two sentences are. Scores of `1` mean that the sentences are exactly the same, while scores of `0` mean the sentences have no overlap.

Pagerank

With a graph of sentences, we can use pagerank to score them. To do this, we'll use the `pagerank` function from **NetworkX**.

```
>>> import networkx as nx
>>> nx_graph = nx.from_scipy_sparse_matrix(similarity_graph)
>>> scores = nx.pagerank(nx_graph)
>>> scores
{0: 0.0867131914937010 8,
 1: 0.0831652358209799 7,
 2: 0.0951394389060688 2,
 3: 0.0944066063656165 7,
 4: 0.0896858200843448 1,
 5: 0.094929054852450 2,
 6: 0.0861643041110893 8,
 7: 0.086273427821944,
 8: 0.0926332082722951 3,
 9: 0.0880839262888583 2,
 10: 0.1028057859826517 3}
```

This gives us a mapping of sentence indices to scores. We can use associate these back to our original sentences and sort them.

```
>>> ranked = sorted(((scores[i],s) for i,s in enumerate(sentences)),
                    reverse=True)
>>> ranked[0][1]
'And yet there was but one woman to him, and that woman was the late '
'Irene Adler, of dubious and questionable memory.'
```

We can combine all of this into one function as follows:

```
import networkx as nx
import numpy as np

from nltk.tokenize.punkt import PunktSentenceTokenizer
from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer

def textrank(document):
    sentence_tokenizer = PunktSentenceTokenizer()
    sentences = sentence_tokenizer.tokenize(document)

    bow_matrix = CountVectorizer().fit_transform(sentences)
    normalized = TfidfTransformer().fit_transform(bow_matrix)

    similarity_graph = normalized * normalized.T

    nx_graph = nx.from_scipy_sparse_matrix(similarity_graph)
    scores = nx.pagerank(nx_graph)
    return sorted(((scores[i],s) for i,s in enumerate(sentences)),
                  reverse=True)
```

Using this technique, we have a simple way of choosing relevant sentences from a text. This model can also be adapted to find keywords that are important in a text, or used as part of more sophisticated scoring methods.

Feel free to contact me via **josh@joshbohde.com** or **@joshbohde**.