Text Analytics and Mining

- Unstructured text data is being generated all the time
- Text analytics / Text mining involves techniques and algorithms for analyzing text
- Traditional data mining techniques may be used if text is converted to numerical vectors

Key Techniques

- NLTK: stemming, stopwords, punctuation, top words
- WordCloud: visualization
- TF-IDF Vectorizer with sklearn
- Topic Modeling with gensim
- Sentiment analysis with TextBlob

TF-IDF Vectorizer with sklearn

- Vectorizers are used to transform words into numbers
- Some use a CountVectorizer – just raw counts of each word in each document
- But it is recommended to use TfidfVectorizer, which weights words by importance, not just by frequency

1) CountVectorizer (i.e. Term Frequency)

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

In [1]:

```
from sklearn.feature_extraction.text import CountVectorizer

# Mock Data For Demonstration Purposes
doc1 = "the moving finger writes and having writ moves on"
doc2 = "the gold finger or golden finger the question is moot"
doc3 = "he is a finger spinner and can write with it too"
doc4 = "the valiant never taste of death but once or so they say"
doc5 = "knights are valiant and never afraid of death"
corpus = [doc1, doc2, doc3, doc4, doc5]
```

> **fit_transform( )** method "tokenize the strings and give you a vector for each string".
>
> The vector is the total number of tokens for the whole corpus.

> Each dimension of which corresponds to the number of times a token is found in the corresponding string.
>
> So, it has both (1) determined which tokens it will count, and (2) how they correspond to entries in the count vector.

In [2]:

```
vectorizer = CountVectorizer(stop_words = 'english')            # Create an instance ob
matrix = vectorizer.fit_transform(corpus) # Tokenize all the strings in the corpus and 
# csr_matrix: Compressed Sparse Matrix
print(type(matrix))
# The Number of Documents and the Total Number of Tokens
print(matrix.shape)                       # Print a stucture of the outcome (i.e. matri
```

```
<class 'scipy.sparse.csr.csr_matrix'>
(5, 18)
```

> **vocabulary_** method "returns a dictionary that represents pairs of a token and its corresponding vector".

In [3]:

```
print(vectorizer.vocabulary_)
len(vectorizer.vocabulary_)
```

```
{'moving': 9, 'finger': 2, 'writes': 17, 'having': 5, 'writ': 15, 'moves': 8, 'gold': 3,
'golden': 4, 'question': 10, 'moot': 7, 'spinner': 12, 'write': 16, 'valiant': 14, 'tast
e': 13, 'death': 1, 'say': 11, 'knights': 6, 'afraid': 0}
```

Out[3]: 18

In [4]:

```
print(matrix)
```

```
  (0, 9)	1
  (0, 2)	1
  (0, 17)	1
  (0, 5)	1
  (0, 15)	1
  (0, 8)	1
  (1, 2)	2
  (1, 3)	1
  (1, 4)	1
  (1, 10)	1
  (1, 7)	1
  (2, 2)	1
  (2, 12)	1
  (2, 16)	1
  (3, 14)	1
  (3, 13)	1
  (3, 1)	1
  (3, 11)	1
  (4, 14)	1
  (4, 1)	1
  (4, 6)	1
  (4, 0)	1
```

> **get_feature_names( )** method "returns a list that represents all the tokens (i.e. word) appearing in the corpus". Each token is a feature of the instance object of CountVectorizer( ).

In [5]:

```
# Let's Investigate Features of the Instance Object
print(vectorizer.get_feature_names())
```

```
['afraid', 'death', 'finger', 'gold', 'golden', 'having', 'knights', 'moot', 'moves', 'm
oving', 'question', 'say', 'spinner', 'taste', 'valiant', 'writ', 'write', 'writes']
```

> **toarray ( )** method "return a dense ndarray representation of the given matrix".

In [6]:

```
# Each Document Is Represented as a Term-Frequency Vector, Where Each Dimension Correspo
matrix.toarray()
```

Out[6]:

```
array([[0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1],
       [0, 0, 2, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0],
       [1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]],
      dtype=int64)
```

In [7]:

```
print(matrix.toarray())
# But, It Is Not Clear Which Feature Name (i.e. Token or Word) Is Corresponding to the
```

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

In [8]:

```
# Let's Combine the Feature Names and the Frequency. Note That They Are List Objects and
for doc in matrix.toarray():
    for idx in range(len(doc)):
        print('{}:{}'.format(vectorizer.get_feature_names()[idx], doc[idx]), end = ' ')
    print('\n')
```

```
afraid:0 death:0 finger:1 gold:0 golden:0 having:1 knights:0 moot:0 moves:1 moving:1 que
stion:0 say:0 spinner:0 taste:0 valiant:0 writ:1 write:0 writes:1

afraid:0 death:0 finger:2 gold:1 golden:1 having:0 knights:0 moot:1 moves:0 moving:0 que
stion:1 say:0 spinner:0 taste:0 valiant:0 writ:0 write:0 writes:0

afraid:0 death:0 finger:1 gold:0 golden:0 having:0 knights:0 moot:0 moves:0 moving:0 que
stion:0 say:0 spinner:1 taste:0 valiant:0 writ:0 write:1 writes:0

afraid:0 death:1 finger:0 gold:0 golden:0 having:0 knights:0 moot:0 moves:0 moving:0 que
stion:0 say:1 spinner:0 taste:1 valiant:1 writ:0 write:0 writes:0

afraid:1 death:1 finger:0 gold:0 golden:0 having:0 knights:1 moot:0 moves:0 moving:0 que
stion:0 say:0 spinner:0 taste:0 valiant:1 writ:0 write:0 writes:0
```

> ***As we saw before, there are several words (i.e. fingers, etc.) that frequently appear in the text. But those words do not have the "distinguishing" power.***

2) Solution: TF-IDF Vectorizer

- TF: Term Frequency: how many times a word appear in a document?
- IDF: Inverse Document Frequency: how many documents include the word?
- TF-IDF(t, d, D) = TF(t, d) * IDF(t, D)
- IDF(t, D) = log(N / |{d in D: t in d}|)
- N: total number of documents in the corpus

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

In [9]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Mock Data For Demonstration Purposes
doc1 = "the moving finger writes and having writ moves on"
doc2 = "the gold finger or golden finger the question is moot"
doc3 = "he is a finger spinner and can write with it too"
doc4 = "the valiant never taste of death but once or so they say"
doc5 = "knights are valiant and never afraid of death"
docs = [doc1, doc2, doc3, doc4, doc5]

vectorizer2 = TfidfVectorizer(stop_words = 'english') # The only difference is the type
matrix2 = vectorizer2.fit_transform(docs)
print(vectorizer2.get_feature_names())
print(matrix2.shape)
print(matrix2.toarray())
```

```
['afraid', 'death', 'finger', 'gold', 'golden', 'having', 'knights', 'moot', 'moves', 'm
oving', 'question', 'say', 'spinner', 'taste', 'valiant', 'writ', 'write', 'writes']
(5, 18)
[[0.         0.         0.28691208 0.         0.         0.42841136
  0.         0.         0.42841136 0.42841136 0.         0.
  0.         0.         0.         0.42841136 0.         0.42841136]
 [0.         0.         0.55645052 0.41544037 0.41544037 0.
  0.         0.41544037 0.         0.         0.41544037 0.
  0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.42799292 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.63907044 0.         0.         0.         0.63907044 0.        ]
 [0.         0.44400208 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.55032913
  0.         0.55032913 0.44400208 0.         0.         0.        ]
 [0.55032913 0.44400208 0.         0.         0.         0.
  0.55032913 0.         0.         0.         0.         0.
  0.         0.         0.44400208 0.         0.         0.        ]]
```

Let's Compare (1) CounterVectorizer and (2) TF-IDF Vectorizer

In [10]:

```
print(vectorizer.get_feature_names()) # (1)CounterVectorizer
len(vectorizer.get_feature_names())
```

```
['afraid', 'death', 'finger', 'gold', 'golden', 'having', 'knights', 'moot', 'moves', 'm
oving', 'question', 'say', 'spinner', 'taste', 'valiant', 'writ', 'write', 'writes']
```

Out[10]: 18

In [11]:

```
print(vectorizer2.get_feature_names()) # (2)TFIDFVectorizer
len(vectorizer.get_feature_names())
```

```
['afraid', 'death', 'finger', 'gold', 'golden', 'having', 'knights', 'moot', 'moves', 'm
oving', 'question', 'say', 'spinner', 'taste', 'valiant', 'writ', 'write', 'writes']
```

Out[11]:

```
18
```

In [12]:

```
print(matrix.toarray()) # (1)CounterVectorizer
```

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

In [13]:

```
print(matrix2.toarray()) # (2)TFIDFVectorizer
```

```
[[0.         0.         0.28691208 0.         0.         0.42841136
  0.         0.         0.42841136 0.42841136 0.         0.
  0.         0.         0.         0.42841136 0.         0.42841136]
 [0.         0.         0.55645052 0.41544037 0.41544037 0.
  0.         0.41544037 0.         0.         0.41544037 0.
  0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.42799292 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.63907044 0.         0.         0.         0.63907044 0.        ]
 [0.         0.44400208 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.55032913
  0.         0.55032913 0.44400208 0.         0.         0.        ]
 [0.55032913 0.44400208 0.         0.         0.         0.
  0.55032913 0.         0.         0.         0.         0.
  0.         0.         0.44400208 0.         0.         0.        ]]
```

3) Practice: Let's Calculate the Pairwise Document Distance with TF-IDF

In [14]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Mock Data For Demonstration Purposes
doc1 = "the moving finger writes and having writ moves on"
doc2 = "gold finger or golden finger the question is moot"
doc3 = "he is a finger spinner and can write with it too"
doc4 = "the valiant never taste of death but once or so they say"
doc5 = "knights are valiant and never afraid of death"
docs = [doc1, doc2, doc3, doc4, doc5]
```

In [15]:

```
vectorizer = TfidfVectorizer(stop_words = 'english')
matrix = vectorizer.fit_transform(docs)
print(len(docs))
print(vectorizer.get_feature_names())
print(matrix.shape)
print(matrix.toarray())
```

```
5
['afraid', 'death', 'finger', 'gold', 'golden', 'having', 'knights', 'moot', 'moves', 'm
```

```
oving', 'question', 'say', 'spinner', 'taste', 'valiant', 'writ', 'write', 'writes']
(5, 18)
[[0.         0.         0.28691208 0.         0.         0.42841136
  0.         0.         0.42841136 0.42841136 0.         0.
  0.         0.         0.         0.42841136 0.         0.42841136]
 [0.         0.         0.55645052 0.41544037 0.41544037 0.
  0.         0.41544037 0.         0.         0.41544037 0.
  0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.42799292 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.63907044 0.         0.         0.         0.63907044 0.        ]
 [0.         0.44400208 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.55032913
  0.         0.55032913 0.44400208 0.         0.         0.        ]
 [0.55032913 0.44400208 0.         0.         0.         0.
  0.55032913 0.         0.         0.         0.         0.
  0.         0.         0.44400208 0.         0.         0.        ]]
```

> **cosine_distances( )** method "takes an object, computes cosine distance between samples in the object, and returns a distance matrix".
>
> Cosine distance is defined as 1.0 minus the cosine similarity.

In [16]:

```python
from sklearn.metrics.pairwise import cosine_distances

cos_dist = cosine_distances(matrix)
print(cos_dist.shape)
```

```
(5, 5)
```

In [17]:

```python
print(cos_dist)
```

```
[[0.         0.84034762 0.87720366 1.         1.        ]
 [0.84034762 0.         0.76184312 1.         1.        ]
 [0.87720366 0.76184312 0.         1.         1.        ]
 [1.         1.         1.         0.         0.60572431]
 [1.         1.         1.         0.60572431 0.        ]]
```

Extra: The Pairwise Document Cosine-Distance with TF (i.e. CountVectorizer)

In [18]:

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_distances

doc1 = "the moving finger writes and having writ moves on"
doc2 = "gold finger or golden finger the question is moot"
doc3 = "he is a finger spinner and can write with it too"
doc4 = "the valiant never taste of death but once or so they say"
doc5 = "knights are valiant and never afraid of death"
docs = [doc1, doc2, doc3, doc4, doc5]

vectorizer = CountVectorizer(stop_words = 'english')
matrix = vectorizer.fit_transform(docs)
cos_dist = cosine_distances(matrix)
print(cos_dist)
```

```
[[0.         0.71132487 0.76429774 1.         1.        ]
 [0.71132487 0.         0.59175171 1.         1.        ]
 [0.76429774 0.59175171 0.         1.         1.        ]
 [1.         1.         1.         0.         0.5       ]
 [1.         1.         1.         0.5        0.        ]]
```

Practice: The Pairwise Document Cosine-Distance with TF-IDF

In [19]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_distances

doc1 = open('frankenstein.txt').read()
doc2 = open('raven.txt').read()
doc3 = open('abbey.txt').read()
docs = [doc1, doc2, doc3]

vectorizer = TfidfVectorizer(stop_words = 'english')
matrix = vectorizer.fit_transform(docs)
cos_dist = cosine_distances(matrix)
print(cos_dist)
```

```
[[0.         0.88543955 0.65820921]
 [0.88543955 0.         0.91959811]
 [0.65820921 0.91959811 0.        ]]
```