Natural Language Processing (NLP)

Some of the most important data in our society is represented as unstructured text:

- Medical records
- Court cases
- Insurance documents

Other data perhaps not as fundamental but that provides interesting insights into trends and mindsets:

- Twitter and other online blogs
- News feeds

NLP

In all of these cases we want to extract meaning from the unstructured text:

- Perhaps we want to do classification (medical records - high risk/low risk)
- Perhaps we want to do a topic analysis of the twitter feeds
- Perhaps we would like to construct a recommendation engine for news feeds

Regardless, what the task, we need to convert the unstructured text into something that we can work with and perhaps most importantly, our models can work with.

☞ The Vector Model of text (sometimes called the Bag-of-Words model)

The Vector Model

The vector model converts a document with unstructured text into a point in an n-dimensional coordinate system where the coordinate system is defined by the words contained in the text.

Consider: *the quick brown fox jumps over the lazy dog*

We can consider this text to be represented as point in the coordinate system (rearranged in alphabetical order):

(brown,dog,fox,jumps,lazy,over,quick,the)

The Vector Model

Let's consider the fact that we have multiple documents:

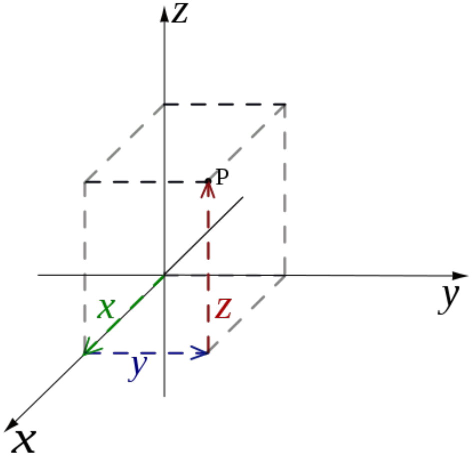
Doc 1: *the quick brown fox jumps over the lazy dog*
Doc 2: *Rudi is a lazy brown dog*

In order to have both documents appear in our vector model we have to extend the coordinate system as the <u>union</u> of all the words appearing in doc 1 and in doc 2:

(a,brown,dog,fox,is,jumps,lazy,over,quick,rudi,the)

Now we can represent the documents as vectors in this coordinate system:

Doc 1: (0,1,1,1,0,1,1,1,1,0,1)
Doc 2: (1,1,1,0,1,0,1,0,0,1,0)



The Vector Model

The nice thing about this vector representation is that we can start doing mathematics on text!

Consider adding another document to our collection:

Doc 3: *Princess jumps over the dog*

The Vector Model

Now we have the following:

Doc 1: *the quick brown fox jumps over the lazy dog*
Doc 2: *Rudi is a lazy brown dog*
Doc 3: *Princess jumps over the lazy dog*

Our coordinate system has the following coordinates:

(a,brown,dog,fox,is,jumps,lazy,over,princess,quick,rudi,the)

And our vectors:

Doc 1: (0,1,1,1,0,1,1,1,0,1,0,1)
Doc 2: (1,1,1,0,1,0,1,0,0,0,1,0)
Doc 3: (0,0,1,0,0,1,1,1,1,0,0,1)

The Vector Model

Given our vector model of the three docs we ask the question:

Is doc2 or doc3 more similar to doc1?

We can answer this question by considering the Euclidean distances doc1⇔doc2 and doc1⇔doc3 in our coordinate system.

The Euclidean distance **d** in n-dimensional space between two points **p** and **q** is defined as:

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

The Vector Model

Doc 1: (0,1,1,1,0,1,1,1,0,1,0,1)
Doc 2: (1,1,1,0,1,0,1,0,0,0,1,0)
Doc 3: (0,0,1,0,0,1,1,1,1,0,0,1)

In our case we are dealing with a 12-dimensional space. To answer our question which doc is more similar to doc1 we compute:

$$d(doc1,doc2) = sqrt((0-1)^2+(1-1)^2+(1-1)^2+(1-0)^2+(0-1)^2+(1-0)^2+(1-1)^2+(1-0)^2+(0-0)^2+(1-0)^2+(0-1)^2+(1-0)^2)$$
$$= sqrt(1+0+0+1+1+1+0+1+0+1+1+1) = sqrt(8) = 2.8$$

$$d(doc1,doc3) = sqrt((0-0)^2+(1-0)^2+(1-1)^2+(1-0)^2+(0-0)^2+(1-1)^2+(1-1)^2+(1-1)^2+(0-1)^2+(1-0)^2+(0-0)^2+(1-1)^2)$$
$$= sqrt(0+1+0+1+0+0+0+0+1+1+0+0) = sqrt(4) = 2.0$$

So, doc3 is more similar to doc1 than doc2:

Doc 1: *the quick brown fox jumps over the lazy dog*
Doc 2: *Rudi is a lazy brown dog*
Doc 3: *Princess jumps over the lazy dog*

The Vector Model

```
import pandas
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances

doc_names = ["doc1", "doc2", "doc3"]
docs = ["the quick brown fox jumps over the lazy dog",
        "Rudi is a lazy brown dog",
        "Princess jumps over the lazy dog"]

# process documents
vectorizer = CountVectorizer(analyzer = "word", binary = True)
docarray = vectorizer.fit_transform(docs).toarray()
coords = vectorizer.get_feature_names()
docterm = pandas.DataFrame(data=docarray,index=doc_names,columns=coords)
print(coords)
print(docterm)

# pairwise distances
distances = euclidean_distances(docterm)
distances_df = pandas.DataFrame(data=distances, index=doc_names, columns=doc_names)
print(distances_df)
```

The Vector Model

```
['brown', 'dog', 'fox', 'is', 'jumps', 'lazy', 'over', 'princess', 'quick', 'rudi', 'the']


      brown  dog  fox  is  jumps  lazy  over  princess  quick  rudi  the
doc1      1    1    1   0      1     1     1         0      1     0    1
doc2      1    1    0   1      0     1     0         0      0     1    0
doc3      0    1    0   0      1     1     1         1      0     0    1


          doc1      doc2      doc3
doc1  0.000000  2.645751  2.000000
doc2  2.645751  0.000000  2.645751
doc3  2.000000  2.645751  0.000000
```

Traditionally the array that holds the vectors for each document is called the ‘docterm’ matrix.

Real World Data

News feed data: http://qwone.com/~jason/20Newsgroups/

from sklearn.datasets import fetch_20newsgroups

“The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups.”

comp.graphics
comp.os.ms-windows.misc
comp.sys.ibm.pc.hardware
comp.sys.mac.hardware
comp.windows.x
rec.autos
rec.motorcycles
rec.sport.baseball
rec.sport.hockey
sci.crypt
sci.electronics

sci.med
sci.space
misc.forsale
talk.politics.misc
talk.politics.guns
talk.politics.mideast
talk.religion.misc
alt.atheism
soc.religion.christian

Real World Data

Each news item has two fields:

- Data - the actual text
- Target - index of the category the news item belongs to

See the 18a-NLP notebook for running code

Real World Data

```
import pandas
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.datasets import fetch_20newsgroups

cats = ['talk.politics.misc', 'sci.space']
newsgroups_train = fetch_20newsgroups(subset='train', categories=cats)
print(len(newsgroups_train.data))
print(list(newsgroups_train.target_names))
print(newsgroups_train.target.shape)
print(newsgroups_train.data[5])
print(newsgroups_train.target[5])
```

The newsgroups_train set contain 1058 news articles from both categories.

Real World Data

Let us compute the docterm matrix for the news articles

```
import pandas
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.datasets import fetch_20newsgroups

cats = ['talk.politics.misc', 'sci.space']
newsgroups_train = fetch_20newsgroups(subset='train', categories=cats)

# process documents
vectorizer = CountVectorizer(analyzer = "word", binary = True)
docarray = vectorizer.fit_transform(newsgroups_train.data).toarray()
print("docarray shape: {}".format(docarray.shape))
```

```
docarray shape: (1058, 23537)
```

Real World Data

First 10 coordinates:
['00', '000', '0000', '00000', '000000', '000007', '000021', '000062david42', '00041032', '0004136']

From this it is clear that we want to do some additional filtering:

1. Minimum doc frequency = 2 -- that is, any word has to appear at least twice in the document collection
2. Delete anything that is not a word - get rid of things like ‘000’ etc.

Real World Data

Let us compute the docterm matrix for the news articles

```
import pandas
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.datasets import fetch_20newsgroups
from re import sub

cats = ['talk.politics.misc', 'sci.space']
newsgroups_train = fetch_20newsgroups(subset='train', categories=cats)

# process documents
vectorizer = CountVectorizer(analyzer = "word", binary = True, min_df=2)
new_data = []
for i in range(len(newsgroups_train.data)):
    new_data.append(sub("[^a-zA-Z]", " ", newsgroups_train.data[i]))
docarray = vectorizer.fit_transform(new_data).toarray()
coords = vectorizer.get_feature_names()

print("docarray shape: {}".format(docarray.shape))
```

```
docarray shape: (1058, 11836)
```

Real World Data

The first few coordinates are now:

['aa', 'aammmaaaazzzzzzziinnnngggg', 'aaron', 'aas', 'ab', 'abandon', 'abandoned', 'abandonment', 'abbey', 'abc']

Much better! One more issue, three different shapes of the same root word.

Solution: Stemming!

Stemming

In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form.

A stemmer for English, for example, should identify the string "cats" (and possibly "catlike", "catty" etc.) as based on the root "cat", and "stems", "stemmer", "stemming", "stemmed" as based on "stem".

A stemming algorithm reduces the words "fishing", "fished", and "fisher" to the root word, "fish".

Real World Data

```
import pandas
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.datasets import fetch_20newsgroups
from re import sub
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
cats = ['talk.politics.misc', 'sci.space']
newsgroups_train = fetch_20newsgroups(subset='train', categories=cats)

new_data = []
for i in range(len(newsgroups_train.data)):
    new_data.append(sub("[^a-zA-Z]", " ", newsgroups_train.data[i]))

lowercase_data = []
for i in range(len(new_data)):
    lowercase_data.append(new_data[i].lower())

stemmed_data = []
for i in range(len(lowercase_data)):
    words = lowercase_data[i].split()
    stemmed_words = []
    for w in words:
        stemmed_words.append(stemmer.stem(w))
    stemmed_data.append(" ".join(stemmed_words))
```

```
vectorizer = CountVectorizer(analyzer = "word", binary = True, min_df=2)
docarray = vectorizer.fit_transform(stemmed_data).toarray()
coords = vectorizer.get_feature_names()
print(coords[:10])
print("docarray shape: {}".format(docarray.shape))
```

```
docarray shape: (1058, 8631)
```

Real World Data

The first few coordinates are now:

['aa', 'aammmaaaazzzzzziinnnnggggg', 'aaron', 'ab', 'abandon', 'abbey', 'abc', 'abdkw', 'abett', 'abid']

Real World Data

We can now compute the distance between our documents in the 8000+ dimensional space:

	0	1	2	3	4
0	0.000000	11.916375	12.806248	13.638182	11.445523
1	11.916375	0.000000	12.328828	13.490738	11.000000
2	12.806248	12.328828	0.000000	14.071247	11.789826
3	13.638182	13.490738	14.071247	0.000000	13.000000
4	11.445523	11.000000	11.789826	13.000000	0.000000