Enron_unfiltered_sent

June 24, 2016

We will look at the "sent" directory of each of the 150 employees of Enron. We need to import the data and in turn, clean up the data. Info from here and here here proved to be very useful. Also see http://www.colorado.edu/ics/sites/default/files/attached-files/01-11_0.pdf

```
In [12]: %install_ext https://raw.github.com/cpcloud/ipython-autotime/master/autotime.py
         %load_ext autotime
         from os import listdir, chdir
         import re
```

```
Installed autotime.py. To use it, type:
  %load_ext autotime
The autotime extension is already loaded. To reload it, use:
  %reload_ext autotime
time: 1.64 s
```

```
/home/peter/anaconda3/lib/python3.5/site-packages/IPython/core/magics/extension.py:47: UserWarning: %ins
  "as a python packages.", UserWarning)
```

We are going to place all the emails of each user into one large list. In order to utalise the LDA algorithm we require there to me multiple documents. The obvious question that arises is whether to consider each email as a seperate document, or to consider the collection of each user's emails as a seperate document. For example:

Consider person A has emails A_1, A_2, A_3 and person B has emails B_1 and B_2. Then we can create a list that is L = [A_1, A_2, A_3, B_1, B_2] or L = [$A_1A_2A_3$, B_1B_2]. For now, all the emails are going to be treated as seperate documents.

Once the LDA algorithm has been implemented, we want to be able to list all the documents that fall under a given catagory.

We now set up the regular expressions to remove the 'clutter' from the emails. (Note, they are purposefully long to avoid successive searches through large data)

```
In [65]: # Defining regular expressions

         re1 = re.compile('(Message-ID(.*?\n)*X-FileName.*?\n)|'
                          '(To:(.*?\n)*?Subject.*?\n)|'
                          '(< (Message-ID(.*?\n)*.*?X-FileName.*?\n))')
         re2 = re.compile('<|'
                          '>|'
                          '(---(.*?\n)?.*?---)|'
                          '(\*\*[.*?\s]\*\*)|'
                          '(.*?:(\s|(.*?\s)|))|'
                          '(\(\d+\))|'
                          '(\s.*?\..*?\s)|'
                          '(\s.*?\_.*?\s)|'
                          '(\s.*?\-.*?\s)|'
```

```
                        '(\s.*\/.*?\s)|'
                        '(\s.*@.*?\s)|'
                        '([\d\-\(\)\\\/\#\=]+(\s|\.))|'
                        '(\n.*?\s)|\d')
        re3 = re.compile('\\\'')
        re4 = re.compile('( . )|\s+')
        re5 = re.compile('( \S{1,3} )|( com )|( can )') # Some problem characters
```

time: 16 ms

In [66]:
```
docs = []

chdir('/home/peter/Downloads/enron')
# For each user we extract all the emails in their inbox

names = [i for i in listdir()]
for name in names:
    sent = '/home/peter/Downloads/enron/' + str(name) + '/sent'
    try:
        chdir(sent)
        for email in listdir():
            text = open(email,'r').read()
            # Regular expressions are used below to remove 'clutter'
            text = re.sub(re1,' ',text)
            text = re.sub(re2,' ',text)
            text = re.sub(re3,'',text)
            text = re.sub(re4,' ',text)
            text = re.sub(re5,' ',text)
            docs.append(text)
    except:
        pass
```

time: 46.7 s

We can make use of either a) Stemming or b) Lemmatizing to find word roots. See here for a more detailed explination of the two. Right below, the stemmer is implemented, while two cells below, the lemmatizer is implemented. Make sure to choose which one to use before proceeding to Constructing the document-term matrix.

In [50]:
```
# We now employ the techniques as outline in the second link at the top - see **
from stop_words import get_stop_words
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'\w+')

texts = []

for doc in docs:
    # Tokenization
    raw = doc.lower()
    tokens = tokenizer.tokenize(raw)

    # Removing stop words

    # create English stop words list
```

```
        en_stop = get_stop_words('en')

        # remove stop words from tokens
        stopped_tokens = [i for i in tokens if not i in en_stop]

        # Stemming

        # Create p_stemmer of class PorterStemmer
        p_stemmer = PorterStemmer()

        # stem token
        stemmed_tokens = [p_stemmer.stem(i) for i in stopped_tokens]

        texts.append(stemmed_tokens)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-50-dc42fdc20b24> in <module>()
     18
     19     # remove stop words from tokens
---> 20     stopped_tokens = [i for i in tokens if not i in en_stop]
     21
     22     # Stemming


<ipython-input-50-dc42fdc20b24> in <listcomp>(.0)
     18
     19     # remove stop words from tokens
---> 20     stopped_tokens = [i for i in tokens if not i in en_stop]
     21
     22     # Stemming


KeyboardInterrupt:
```

```
time: 14.7 s
```

```
In [68]: # We now employ the techniques as outline in the second link at the top - see **
         from stop_words import get_stop_words
         from nltk.stem.wordnet import WordNetLemmatizer
         from nltk.tokenize import RegexpTokenizer
         tokenizer = RegexpTokenizer(r'\w+')

         texts = []

         for doc in docs:
             # Tokenization
             raw = doc.lower()
             tokens = tokenizer.tokenize(raw)

             # Removing stop words
```

```
            # create English stop words list
            en_stop = get_stop_words('en')

            # remove stop words from tokens
            stopped_tokens = [i for i in tokens if not i in en_stop]

            # Stemming

            # Create p_stemmer of class PorterStemmer
            wordnet_lemmatizer = WordNetLemmatizer()

            # stem token
            lemmatized_tokens = [wordnet_lemmatizer.lemmatize(i) for i in stopped_tokens]

            texts.append(lemmatized_tokens)
```

```
time: 27 s
```

Below, we construct the document term matrix whereafter the fairly lengthy process of constructing the model takes place. Thus far the model seems be linear. With a single pass, the model takes just upward of a minute to execute, whereas for 5 passes, the model takes roughly 5.5 minutes.

See the following

```
In [69]: # Constructing a document-term matrix

         from gensim import corpora, models

         dictionary = corpora.Dictionary(texts)

         corpus = [dictionary.doc2bow(text) for text in texts]
```

```
time: 6.44 s
```

```
In [70]: ldamodel = models.ldamodel.LdaModel(corpus, num_topics=7, id2word = dictionary, passes=5)
```

```
time: 5min
```

```
In [71]: num_topics = 7
         num_words = 15

         List = ldamodel.print_topics(num_topics, num_words)
         for i in range(0,len(List)):
             word_list = re.sub(r'(.\....\*)|(\+ .\....\*)', '',List[i][1])
             print('Topic ' + str(i) + ': ' + '\n' + str(word_list))
             print('\n' + '-'*100 + '\n')
```

```
Topic 0:
will enron meeting please group business forwarded also mark team jeff information like management tradi

----------------------------------------------------------------------------------------------------

Topic 1:
will power california utility contract rate cost state customer market energy issue price commission e

----------------------------------------------------------------------------------------------------
```

```
Topic 2:
agreement please pl credit will change attached master copy draft isda form document party letter

----------------------------------------------------------------------------------------------------

Topic 3:
com enron smith street texas houston perlingiere dperlin legal department fax please sara phone debra

----------------------------------------------------------------------------------------------------

Topic 4:
deal forwarded price month daily gas please volume number transwestern shall book will product contract

----------------------------------------------------------------------------------------------------

Topic 5:
said state energy california company davis power d governor billion electricity e year market price

----------------------------------------------------------------------------------------------------

Topic 6:
know will jeff thanks like just think want get need time dont call going please

----------------------------------------------------------------------------------------------------

time: 16.3 ms
```

In [ ]: