1.3 Cleaning, collocation, count vectorization

- run on AWS instance or something with a large amt of ram (approx 109Gb)
- instance details (r4.8xlarge, us-east-1a, AMD64, 244GB memory, Windows 10)

1 remove PII strings

2 custom tokenizer to create vocabulary using gensim's phraser fxn with the following settings

```
        ngrams 1-5
        threshold = 2 min score for an n-gram to be taken into account  (dep
ends on scorer),
            higher number = less phrases
        min_count = 2 ignore all words/phrases with a count lower than this
        add additional vocab terms from the term collapsing in Naive bayes p
rocessing
```

3 vectorize the data using count vectorizer to create our document-term matrix most parameters for count vect are ignored b/c we pass a vocabulary that is created in the `TextFeatures` class below. 4 save as sparse pickle files that you can then import to a local machine for analysis

- postgres >= 9.4

In [1]:
```
# -*- coding: utf-8 -*-

import time
from datetime import datetime

import pickle
import re
import sys
import os.path

import numpy as np
import pandas as pd

import gensim
from gensim.models import Phrases
from gensim.models.phrases import Phraser
from gensim.utils import tokenize

import sklearn
from sklearn.feature_extraction.text import CountVectorizer, TfidfVector
izer

import sqlalchemy
from sqlalchemy import create_engine

engine = create_engine('postgresql://postgres:postgrespassword@localhos
t/mimic')

# print the versions of modules used here

libraries = (('Numpy', np), ('Pandas', pd),('Sklearn', sklearn),('Gensi
m',gensim),('sqlalchemy',sqlalchemy))
print('last ran: ',datetime.now() )
print("Python Version:", sys.version)
print( "operating system:", sys.platform)
for lib in libraries:
    print('{0} Version: {1}'.format(lib[0], lib[1].__version__))
```

```
C:\ProgramData\Anaconda2\lib\site-packages\gensim\utils.py:860: UserWar
ning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_seria
l")

('last ran: ', datetime.datetime(2019, 8, 26, 1, 14, 37, 433000))
('Python Version:', '2.7.14 |Anaconda custom (64-bit)| (default, Oct 15
2017, 03:34:40) [MSC v.1500 64 bit (AMD64)]')
('operating system:', 'win32')
Numpy Version: 1.14.2
Pandas Version: 0.22.0
Sklearn Version: 0.19.1
Gensim Version: 3.1.0
sqlalchemy Version: 1.1.13
```

Main functions

- **remove_masked_pii_string**:

```
the masked strings that identify individual patients or caregivers
```

- **tokenize_and_phrase**:

```
1. lowercase characters
2. remove newline characters (\n)
3. tokenize: split into words on whitespace
4. remove empty tokens
5. collocation to find multi-word tokens (create n-grams (1-5))
```

- **get_ngram_counts**:

```
vectorize (using count vectorizer)
```

In [3]:
```
class TextFeatures(object):
    def __init__(self, corpus, pg_conn=None):

        self.count_matrix = None
        self.vocab = None
        self.corpus = corpus

    def remove_masked_pii_string(self):
        ''' using regular expression, remove the masked strings that ide
ntify individual patients or caregivers.'''

        num_reports = len(self.corpus)
        for i in range(num_reports):
            self.corpus[i] = re.sub(re.compile("\[\*\*(.*?)\*\*\]"), "",
self.corpus[i])

    def tokenize_and_phrase(self):
        print 'Creating tokens'
        # tokenize: remove newline chars, lowercase, split into words on
whitespace
        sentence_stream = [doc.replace('\n', ' ').lower().split(" ") for
doc in self.corpus]

        #remove empty tokens
        for i in range(0,len(sentence_stream)):
            sentence_stream[i] =[w for w in sentence_stream[i] if w !=u
'']
        print "Done with tokens"

        # collocation detection
        # This doesn't work as expected from the docs -- trigram creatio
n actually creates up to 4 or 5 grams
        bigram = Phraser(Phrases(sentence_stream, min_count=2,threshold=
2, delimiter=' '))
        print datetime.now()

        trigram = Phraser(Phrases(bigram[sentence_stream], min_count=2,t
hreshold=2, delimiter=' '))
        print('trigram done')
        print datetime.now()

        tetragram = Phraser(Phrases(trigram[bigram[sentence_stream]], mi
n_count=2,threshold=2, delimiter=' '))
        print 'tetragram done'
        print datetime.now()

        pentagram = Phraser(Phrases(tetragram[trigram[bigram[sentence_st
ream]]], min_count=2,threshold=2, delimiter=' '))
        print 'pentagram done'
        print datetime.now()

        #=============================================================
====================================
        # Work around for unexpected behavior to reduce the gram count <
5
        bigrams=[gram for sent in list(bigram[sentence_stream]) for gram
```

```
in sent if len(gram.split(" ")) == 2]

        print 'bigrams done'
        print datetime.now()

        trigrams=[gram for sent in list(trigram[bigram[sentence_stream
]]) for gram in sent if len(gram.split(" ")) == 3]

        print 'trigrams done'
        print datetime.now()

        tetragrams=[gram for sent in list(tetragram[trigram[bigram[sente
nce_stream]]]) for gram in sent if len(gram.split(" ")) == 4]

        print 'tetragrams done'
        print datetime.now()

        pentagrams=[gram for sent in list(pentagram[tetragram[trigram[bi
gram[sentence_stream]]]]) for gram in sent if len(gram.split(" ")) == 5]

        print datetime.now()

        unigrams=[w for s in sentence_stream for w in s]

        print len(set(unigrams))
        print datetime.now()

        vocab = list(set(unigrams+bigrams+trigrams+tetragrams+pentagrams
))

        print(len(set(vocab)))
        print "Done with collocation detection"

        return vocab

    def get_ngram_counts(self, token_pattern, vocabulary):
        '''Vectorizer (count) OUTPUT: vectorized data, vocabulary(featur
e names)'''

        print 'Starting feature extraction'

        count_vectorizer = CountVectorizer(token_pattern=token_pattern,
vocabulary=vocabulary)
        ngram_matrix = count_vectorizer.fit_transform(self.corpus)
        ngram_vocab = count_vectorizer.get_feature_names()

        print(count_vectorizer)
        return ngram_matrix, ngram_vocab
```

Save

- the large matricies from the analysis as compressed sparse row matricies
- chunk the vectorized data in pieces as pickle files so we can move them/store easily

In [4]:
```
def feature_pickle(r, path):
    t=time.strftime("%Y%m%d%H%m",time.localtime())
    if not os.path.exists(path+'\\'+t):
        os.mkdir(path+'\\'+t)
    path=path+'\\'+t+'\\'

    print( r[0].shape)
    chunk = (r[0].shape[0])/10
    for i in range(0,9):
        with open(path+'textfeatures_mat'+str(i+1)+'.pickle', 'wb') as f
:
            pickle.dump(r[0][chunk*i:chunk*(i+1)], f, protocol=pickle.HI
GHEST_PROTOCOL)
    with open(path+'textfeatures_mat10.pickle', 'wb') as f:
        pickle.dump(r[0][chunk*9:], f, protocol=pickle.HIGHEST_PROTOCOL)
    with open(path+'textfeatures_vocab.pickle', 'wb') as f:
        pickle.dump(r[1], f, protocol=pickle.HIGHEST_PROTOCOL)
    with open(path+'textfeatures_id.pickle', 'wb') as f:
        pickle.dump(r[2], f, protocol=pickle.HIGHEST_PROTOCOL)
    with open(path+'textfeatures_source.pickle', 'wb') as f:
        pickle.dump(r[3], f, protocol=pickle.HIGHEST_PROTOCOL)
```

Get Data from Postgres

- pull notes and hadm_id and label as transfused or non-transfused (control)

In [7]:
```
def get_adm_notes():
    '''Get data from mimic Postgres database for transfused and non-tran
sfused patients.
    OUTPUT: pandas dataframe with rows=admission, cols=hadmid, source (t
ransfused or control), text.'''

    # transfused
    xf_adm = pd.read_sql("""select * from mimiciii.transfused_notes_uniq
ue
    """, engine)

    xf_adm['source'] = 'transfusion'

    # control
    ctrl_adm = pd.read_sql("""select * from mimiciii.ctrl_notes_unique
""", engine)

    ctrl_adm['source'] = 'control'


    mimic_notes = pd.concat([xf_adm, ctrl_adm])
    mimic_notes = mimic_notes[['hadm_id', 'source', 'text']]

    return mimic_notes
```

In [6]:
```
xf, ct =get_adm_notes()
len(xf)
```

Out[6]: 21443

This will run everything. takes about 6:12 hours on the large AWS instance

- Time to extract features: 6:12
- pulls notes
- remove masked pii string
- tokenize
- collocations (detect multi-word tokens)
- add additional vocab terms
- count vectorize using vocab we created in the first part (so, setting the n-grams, and all those params won't matter b/c we pass our own vocabulary)
- save in sections as pickle files

In [8]:
```
if __name__ == "__main__":

    print datetime.now()
    print "Getting notes"

    mimic_notes=get_adm_notes()

    startTime = datetime.now()
    print "Text Features"

    text_features = TextFeatures(corpus=mimic_notes.text.values)
    text_features.remove_masked_pii_string()

    vocab = text_features.tokenize_and_phrase()

    r = list(text_features.get_ngram_counts(token_pattern='(?u)\\b\w\w+
\\b', vocabulary=vocab))

    r.append(mimic_notes.hadm_id.values)
    r.append(mimic_notes.source.values)

    print("...Done extracting text features...")
    print r[0].shape
    print ("Time to extract features: %s" % (str(datetime.now() - startT
ime)))

    feature_pickle(r, 'D:\\vectorization_results')
```

```
2019-08-26 01:15:52.441000
Getting notes
Text Features
Creating tokens
Done with tokens
2019-08-26 01:37:39.470000
trigram done
2019-08-26 02:16:29.620000
tetragram done
2019-08-26 03:05:22.402000
pentagram done
2019-08-26 04:15:04.895000
bigrams done
2019-08-26 04:34:29.751000
trigrams done
2019-08-26 05:09:05.855000
tetragrams done
2019-08-26 05:58:31.219000
2019-08-26 07:01:59.621000
2391685
2019-08-26 07:02:51.596000
7422044
Done with collocation detection
Starting feature extraction
CountVectorizer(analyzer=u'word', binary=False, decode_error=u'strict',
        dtype=<type 'numpy.int64'>, encoding=u'utf-8', input=u'conten
t',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None,
        vocabulary=[u'can have regular diet,', u'5 cmh2o rsbi: 61', u'd
ouble vision. functional status:', u'be constructed', u'5 cmh2o rsbi: 6
8', u's/p y stent placement .', u'glass opacities,', u'moist, trachea',
u'glass opacities.', u'vs: t 98.4 hr', u'spo2>94.', u'sodium 17. magnes
ium', u'spo2>94%', u'k-...n', u'lumenal narrowing.', u'schalatter,',
u'lumenal narrowing)', u'the overall appearance has not'])
...Done extracting text features...
(49331, 7422044)
Time to extract features: 6:10:52.417000
(49331, 7422044)
```