2.1_nBayes_remove_transfusion_terms

May 26, 2021

1 2.1_nBayes_remove_transfusion_terms

1.1 Clean up top 5k terms

- needs to be run on sparse matrix of 5000 terms from naive bayes model output from 2.0_classification_models.ipynb
- used where the **model.feature_log_prob_** was used to rank terms (not coefs). run with python 2.7

   1. load, extract
   2. remove transfusion only terms
   3. collapse to longest n-gram because it merges them and creates terms that are not in our original list … which means we can't 'drop them from the model b/c they don't exist
   4. save

1.2 0 import packages and initialize database

[ ]:
```
# Python 2 & 3 Compatibility
from __future__ import print_function, division
```

[ ]:
```
import pandas as pd
import pickle
from fuzzywuzzy import fuzz, process
from difflib import SequenceMatcher
import itertools
import scipy
import psycopg2
from sqlalchemy import create_engine
import sys
import numpy as np
from datetime import datetime

sys.setrecursionlimit(10000)
#pd.set_option('display.max_rows', 1000)
#pd.set_option('display.max_colwidth', -1)

import seaborn as sns
%matplotlib inline
```

```
import matplotlib.pyplot as plt


conn = psycopg2.connect("dbname=mimic user=xxxxxxxxxxx␣
 ↪options=--search_path=mimiciii");
engine = create_engine('postgresql://xxxxxxxxxxx@localhost/mimic')
path1 = './'


cur = conn.cursor();
cur.execute("""SET search_path = mimiciii;""")

from importlib_metadata import version


libraries = ['pandas','sqlalchemy','psycopg2','tqdm','numpy',␣
 ↪'scipy','fuzzywuzzy','seaborn','matplotlib']
print('last ran: ',datetime.now() )
print("Python Version:", sys.version[0:7])
print( "operating system:", sys.platform)

for lib in libraries:
    print(lib + ' version: ' + version(lib))
```

1.3 1.0 Load data

```
[ ]: with open('NB_top_5000_matrix.pickle', 'rb') as f:

         mat,terms0=pickle.load(f)

     terms=list(terms0.iloc[:,0])

     print(mat.shape)
```

1.4 1.1 get coo

- Return a Coordinate (coo) representation of the Compresses-Sparse-Column (csc) matrix.

```
[ ]: coo = mat.tocoo(copy=False)

     # Access `row`, `col` and `data` properties of coo matrix.
     mat = pd.DataFrame({'feature': coo.row, 'ID': coo.col, 'count': coo.data}
                      )[['feature', 'ID', 'count']].sort_values(['feature', 'ID']
                      ).reset_index(drop=True)

     mat.loc[:,'feature'] = mat.loc[:,'feature'].apply(lambda x: terms[x])
     print( mat.shape)
```

1.5 2.0 Load the transfusion related terms

- specified by SMEs

```
[ ]: xfus=pd.read_excel('terms_indicate_transfusion9.xlsx')

     xfus.columns=['ngrams']
     xfus['ngrams'] = xfus.ngrams.str.strip()
     xfus.shape
```

1.6 2.1 Find and Remove

- remove the exact matches to the transfusion related terms specified by SMEs

```
[ ]: ngrams_orig = len(mat.feature.unique())
     print('num of ngrams=',ngrams_orig)

     data = ~mat['feature'].isin(list(xfus['ngrams'].values))
     print (data.value_counts())
     mat=mat[data]
     print( mat.shape)

     print('num of ngrams after removing transfused terms =',len(mat.feature.
      ↪unique()))
     print('removed ' + str(ngrams_orig- len(mat.feature.unique())) + ' transfused␣
      ↪terms')
```

1.7 2.2 get IDs

- to output the terms with the ratios, merge back to the 'terms' mat that can be imported at the top (on=feature)

```
[ ]: terms1=mat[~mat.feature.duplicated()]
     terms0.columns=['feature','ratio']
     terms2 = terms0.merge(terms1,how='right',on='feature')
     terms2 = terms2.drop(['ID','count'],axis=1).sort_values('ratio',axis=0)

     terms2.head()
```

```
[ ]: with open('textfeatures_id.pickle','rb') as f:

         ids=pickle.load(f, encoding='latin1')

     h=pd.read_sql('''select hadm_id from mimiciii.transfused_notes_unique;
      ↪''',engine)

     mat.loc[:,'ID'] = mat.loc[:,'ID'].apply(lambda x: int(ids[x]))
     mat['hadm_id']=mat['ID']
```

```
mat = pd.merge(mat, h, how='left',on='hadm_id')

mat.ID = mat.hadm_id
mat=mat.reset_index()
mat.drop(['hadm_id','index'],axis=1, inplace=True)

mat.head()
```

2 3 Collapse to longest n-gram

- find rows that are duplicates
- see if those duplicates have a similar feature name (index)
  - if yes, then save into **dupes** and **no_dupes** matricies
  - input **dupes** to functions to collapse to longest feature name by removing shorter one (remove works b/c we are doing binary)
  - put **no_dupes** back together with the de-duplicated **dupes** for completed matrix

2.1 3.0 Transform the DataFrame

- each document (admission) is now a col, and a one is present for each feature that belongs to that doc

[ ]:
```
# this is pretty slow
df = mat.pivot_table(index='feature',columns='ID',values=None).fillna(0).
 ↪astype('int32').sort_index()
```

[ ]:
```
df.columns = df.columns.droplevel()
df.head()
```

2.2 3.1 Find and Separate out n-grams with same patterns of occurrence

- create a mat w/o any of these 'duplicates'
- after keeping longest ngram, we will append the de-duped ones to this mat **no_dupes**

[ ]:
```
data= df.duplicated(keep=False)

no_dupes=df.loc[~data,:]

print( no_dupes.shape)

dupes=df.loc[data,:]

ind=list(dupes.index)
dupes.shape
```

```
# grab the first of each duplicate for fuzzy matching below

data1=dupes.duplicated()

first=list(dupes.loc[~data1,:].index)
len(first)
```

[ ]:
```
# removing the words that end up without a matching n-gram after colllapsing␣
 ↪them.
first
```

2.3 3.2 keep longest n-gram

- use fuzzy matching to select longest n-gram of the duplicates
- results are in **dupes**

[ ]:
```
def find_dupes(x):
    if x.equals(dupes.loc[first[f],:]):
        dind.append(x.name)

def all_match(x):
    return all(fuzz.partial_ratio(i,max(x, key=len)) ==100 for i in x)



def some_match(x):
    x=list(x)
    return [i for i in x if fuzz.partial_ratio(i,max(x,key=len))==100], [i for␣
 ↪i in x if fuzz.partial_ratio(i,max(x,key=len))!=100]

def remove(m, dind):
    todrop=dind
    for n in m:
        del todrop[todrop.index(n)]

    dupes.drop(todrop, axis=0, inplace=True)

def rename(m, dind):
    new_ind=m+dind[len(m):]
    d = dict(zip(dind,new_ind))
    print(d)
    dupes.rename(index = d, inplace=True)

    dupes.drop(dind[len(m):], axis=0, inplace=True)

def duplicate_removal(match):
    m=max(map(len, match))
    m=[x for x in match if len(x) == m]
    if len(m)==1:

        remove(m,match)
    else:
        print( 'Error: Edge Case - Multiple longest terms')
        print( m)
        print (fuzz.partial_ratio(m[0],m[1]))
    return m

def recursive_match(match, unmatch):
    if len(unmatch)>0 and len(match)>1:
        l = duplicate_removal(match)
        catch.append(l[0])
        match, unmatch= some_match(unmatch)
        return recursive_match(match, unmatch)
    else:
        return match, unmatch

def word_match(astring,bstring):
    count=0
    for a in astring.split(' '):
        for b in bstring.split(' '):
            if a==b:
                count+=1
    if count==0:
        q.append(bstring)


def get_overlap(unmatch):
    '''this is all based on length. it is cutting off the longer term and␣
 ↪putting them together if one is shorter? '''

    a=unmatch[0]
    for b in unmatch:
        if a!=b:
            d=SequenceMatcher(None, a, b)
            pos_a, pos_b, size = d.find_longest_match(0,len(a),0,len(b))
            if pos_a>pos_b and (pos_a==0 or pos_b==0) and size>0 and fuzz.
 ↪partial_ratio(a,b)!=100:
                a=a + ' ' + b
            elif pos_b>pos_a and (pos_a==0 or pos_b==0) and size>0:
                a=b + ' ' + a
            elif pos_b==0 and pos_a == 0 and size>0:
                a=max([a,b],key=len)
            else:
                word_match(a,b)
    return a
```

```
dind=[]
catch=[]
q=[]
new=[]

for f in range(len(first)):
    dupes.apply(find_dupes, axis=1)
    if all_match(dind):
        m = duplicate_removal(dind)

        dind=[]
    else:
        match, unmatch=some_match(dind)
        match,unmatch=recursive_match(match,unmatch)

        l=duplicate_removal(match)
        unmatch=unmatch+l+catch
        rows=unmatch

        a = get_overlap(unmatch)
        new.append(a)
        if len(q)>0:
            unmatch=q
            q=[]
            a=get_overlap(unmatch)
            new.append(a)

        if len(q)==1:
            new.append(q[0])
        while len(q)>1:
            unmatch=q
            q=[]
            a=get_overlap(unmatch)
            new.append(a)
        if len(q)==1:
            new.append(q[0])

        rename(new, rows)



        dind=[]
        catch=[]
        q=[]
        new=[]
```

- note that there are similar ngrams here, it's b/c they don't have the same patterns of ocurrance. go look at original list to see what the matches were

[ ]:
```
dupes.iloc[:,1]
```

- append our de-duped matrix to the matrix w/o dupes

[ ]:
```
out=dupes.append(no_dupes)
```

2.4 3.3 remove rows that sum to zero (just in case)

- transpose, remove, transpose back

[ ]:
```
out3=out.transpose()
print(out.shape)

out3=out3[~(out3.sum(axis=1)==0)]
print(out3.shape)

out4=out3.transpose()
print(out4.shape)
```

2.5 3.4 see if there are any more duplicates left

[ ]:
```
print (out4.duplicated(keep=False).value_counts())
```

[ ]:
```
with open(path + path1 + 'NB_terms_ratio_all.pkl', 'rb') as f:

    terms_all=pickle.load(f)

terms_all.rename(columns={'vocab': 'feature'},inplace=True)
terms_all.head()
```

[ ]:
```
top = pd.DataFrame()
top['feature']= out.index

# merge with terms0 to go get the ratios for ordering
topt = terms_all.merge(top, how='inner', on='feature', copy=True,␣
↪indicator=True)
```

[ ]:
```
len(topt)
```

3 plot top terms by ratio

[ ]:
```
def plot_top_terms(df, fig_title, top_words):
    fig = plt.figure(figsize=(8, 12),dpi=100)
    y_pos = np.arange(top_words)

    plt.barh(y_pos,df.ratio.tail(top_words), alpha=0.5)
    plt.title(fig_title, fontsize=20)
    plt.yticks(y_pos,df.feature.tail(top_words), fontsize=14)
    plt.xlim(.70,.83)
    #plt.suptitle('Key words', fontsize=16)
    plt.xlabel('ratio', fontsize=20)
    #plt.subplots_adjust(wspace=0.8)
```

[ ]:
```
# drop the 2 terms with nans
top_p=  topt.dropna(axis=0, how='any')

# strongest terms at top
top_p.sort_values('ratio',ascending=False,inplace=True)
plot_top_terms(top_p,'Top Terms by Ratio',50)
```

[ ]:
```
top_p.size
```

Save results of these collapsed vocab terms

[ ]:
```
top_p.to_pickle('NB_5000_final.pkl')
top_p.to_csv('NB_5000_final.csv')
```

4 look at distribution of the terms

- to see if there are a few terms that are present in all documents.
- if so, remove those terms

[ ]:
```
plot_mat = out.merge(top_p,how='left',on='feature')
plot_mat.sort_values(by='ratio',inplace=True)
plot_mat.drop('ratio',axis=1,inplace=True)
plot_mat.set_index('feature',drop=True,inplace=True)
plot_mat.head()
```

4.1 plot histogram of total frequency for each term

[ ]:
```
plot_mat['total_count_freq'] = plot_mat.sum(axis=1)

plot_mat['total_hadmids'] = plot_mat.astype(bool).sum(axis=1)
```

```
[ ]: #plot_matt = plot_mat.drop('total')
     plot_mat.total_count_freq.plot.
      ↪hist(logy=True,figsize=(20,15),use_index=True,bins=(100))
```

```
[ ]: plot_mat.total_hadmids.plot.
      ↪hist(logy=True,figsize=(20,15),use_index=True,bins=(100))
```

```
[ ]: plot_mat.loc['total_words'] = 0
     plot_mat.loc['total_words'] = plot_mat.astype(bool).sum(axis=0)
     plot_mat.sort_values(by='total_words',axis=1,ascending=False,inplace=True)
```

```
[ ]: plot_mat
```

```
[ ]: """
     Plot the sparsity pattern of arrays
     """
     import matplotlib.pyplot as plt
     from matplotlib.pyplot import figure, show
     import numpy as np

     fig = plt.figure(dpi=100, figsize=(30,30));

     ax4 = plt.gca()

     x = plot_mat.drop(columns=['total_count_freq','_merge'],axis=1)
     x.sort_values('total_hadmids',inplace=True, ascending=False)
     x.drop('total_hadmids',axis=1,inplace=True)
     x.drop('total_words',axis=0,inplace=True)

     ax4.spy(x, precision=0, markersize=.2, aspect='auto')
```

```
[ ]: """
     Plot the sparsity pattern of arrays
     """
     import matplotlib.pyplot as plt
     from matplotlib.pyplot import figure, show
     import numpy as np

     fig = plt.figure(dpi=100, figsize=(30,30));

     ax4 = plt.gca()


     xx = plot_mat.drop(columns=['_merge','total_hadmids'],axis=1)
     xx.sort_values('total_count_freq',ascending=False, inplace=True)
     xx.drop('total_count_freq',axis=1,inplace=True)
     xxt = xx.T
```

```
xxt.drop('total_words',axis=1,inplace=True)

ax4.spy(xxt, precision=0,markersize=.2, aspect='auto')
```

```
[ ]: s = plot_mat.total_count_freq.sort_values()
     s = s.drop('total_words')
     s.tail(45).plot.barh(figsize=(30,50),fontsize=40,title='Terms by total count');
```

```
[ ]: s = plot_mat.total_hadmids.sort_values()
     s = s.drop('total_words')
     s.tail(45).plot.barh(figsize=(20,30),fontsize=28, title='Terms by number of␣
      ↪transfused admissions');
```

```
[ ]: # sort by the ratio
     plot_mat_out = plot_mat.merge(terms0,how='left',on='feature')
     plot_mat_out.sort_values(by='ratio',ascending=True,inplace=True)

     cols = list(plot_mat_out.columns.values)
     # reorder cols
     cols = cols[-1:] + cols[:-3]
     plot_mat_out = plot_mat_out[cols]

     plot_mat_out.set_index('feature',inplace=True,drop=True)
     #plot_mat_out.drop('total_words',axis=0,inplace=True)
     plot_mat_out.head(50)
```

save the terms, ratio, and freq count for SMEs

```
[ ]: plot_mat_out1 = plot_mat_out.loc[:,['total_hadmids','total_count_freq','ratio']]
     plot_mat_out1.drop('total_words',axis=0,inplace=True)
     plot_mat_out1.to_csv(path +  'NB_top_'+ str(top_p.shape[0]) +'_terms_only_dist.
      ↪csv')
```

now save terms, ratio, count, and all hadmids for SMEs

```
[ ]: plot_mat_out.to_csv(path + path1 + 'NB_top_'+ str(top_p.shape[0])␣
      ↪+'_hadmids_forSME.csv')
```

```
[ ]: conn.commit();
     cur.close();
     conn.close();
```