2.2_logReg_remove_transfusion_terms

May 26, 2021

1 2.2_logReg_remove_transfusion_terms

1.1 Clean up top terms Logistic Regression output for top 5000 terms

- needs to be run on matrix output from 2.0_classification-models.ipynb
- used where the **model.coef_** was used to rank terms
  1. load, extract
  2. remove transfusion only terms
  3. collapse to longest n-gram
  4. save for review and clustering analysis

1.2 0 import packages and initialize database

[ ]:
```
from __future__ import print_function, division
```

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

from datetime import datetime

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')
cur = conn.cursor();
```

```
cur.execute("""SET search_path = mimiciii;""")

from importlib_metadata import version

path = "./"

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('logits_top_5000_matrix.pickle', 'rb') as f:

    mat,terms0=pickle.load(f, encoding='latin1')

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)

#mat.head()
```

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()
```

1.6 2.1 Find and Remove

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

[ ]:
```
original_ngrams = len(mat.feature.unique())
print('original ngrams=',original_ngrams)

data = ~mat['feature'].isin(list(xfus['ngrams'].values))
print (data.value_counts())
mat=mat[data]
print (mat.shape)
print('removed '+str(original_ngrams-len(mat.feature.unique())) + ' transfusion␣
 ↪related terms')
```

1.7 2.2 get IDs

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

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

terms2.head()
```

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

    ids=pickle.load(, 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

[ ]:
```
df = mat.pivot(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)
```

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):
    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[0:pos_a]+''+b
            elif pos_b>pos_a and (pos_a==0 or pos_b==0) and size>0:
                a=b[0:pos_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 mat to the mat 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)
```

3 merge back to coefs

[ ]:
```
top = pd.DataFrame()
top['feature']= out.index
# merge with terms0 to go get the ratios for ordering
topt = terms0.merge(top, how='right', right_on='feature', left_on='feature')
topt.sort_values('coef',inplace=True,ascending=True)
```

4 there were no 'new' ngrams created by collapsing to the longest ngram for Logistic Regression.

[ ]:
```
new_ngrams=topt[topt.coef.isna()]
new_ngrams
```

4.1 plot top terms by coef

[ ]:
```
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.coef.tail(top_words), alpha=0.5)
    plt.title(fig_title, fontsize=20)
    plt.yticks(y_pos,df.feature.tail(top_words), fontsize=14)
    plt.xlabel('coef', fontsize=20)

plot_top_terms(topt,'Top Terms by Coef',45)
```

4.2 4.0 save

- mat to sparse csr
- top = terms and coefs

[ ]:
```
#out1=scipy.sparse.csr_matrix(out.values)

with open('LR_5000_final.pkl','wb') as f:
        pickle.dump(topt, f)
```

4.3 4.1 look at distribution of terms

- see if there are a few terms showing up in all the docs and remove them to reduce number of admissions

[ ]:
```
plot_mat = out.merge(topt,how='left',on='feature')
plot_mat.sort_values(by='coef',ascending=False, inplace=True)
plot_mat.drop('coef',axis=1,inplace=True)
plot_mat.set_index('feature',drop=True,inplace=True)
```

4.3.1 calc the total freq 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_mat.total_count_freq.plot.
 ↪hist(logy=True,figsize=(20,15),use_index=True,bins=(100),title='Total Count␣
 ↪per Term Histogram')
```

[ ]:
```
plot_mat.total_hadmids.plot.
 ↪hist(logy=True,figsize=(20,15),use_index=True,bins=(100),title='Total␣
 ↪HADMIDs per Term Histogram')
```

[ ]:
```
plot_mat.loc['total_words'] = 0
plot_mat.loc['total_words'] = plot_mat.astype(bool).sum(axis=0)
```

[ ]:
```
plot_mat.sort_values(by='total_hadmids',ascending=False,inplace=True)
```

[ ]:
```
"""
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(title='Heatmap of terms (rows) x transfused admissions (cols)')

x = plot_mat.drop('total_count_freq',axis=1)
x.drop('total_hadmids',axis=1,inplace=True)
x.drop('total_words',axis=0,inplace=True)

ax4.spy(x, precision=0,markersize=.05, aspect='auto');
```

[ ]:
```
fig = plt.figure(dpi=100, figsize=(30,30));

ax4 = plt.gca(title='Heatmap of transfused admissions(rows) x  terms (cols)')

xx = plot_mat.drop('total_count_freq',axis=1)
xx.drop('total_hadmids',axis=1,inplace=True)
xxt = xx.T
xxt.sort_values('total_words',ascending=False,inplace=True)
xxt.drop('total_words',axis=1,inplace=True)

ax4.spy(xxt, precision=0,markersize=.05, aspect='auto');
```

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

4.4 4.2 save for SME review

- merge with ratios to get hadmids per term

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

cols = list(plot_mat_out.columns.values)
# reorder cols
cols = cols[-3:] + 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_out1 = plot_mat_out.loc[:,['coef', 'total_count_freq','total_hadmids']]
#plot_mat_out1.drop('total_words',axis=0,inplace=True)
plot_mat_out1.to_csv('LR_top_5000_terms_only.csv')
```