2.4_filtered_vocab

May 26, 2021

1 2.4 Filter vocabulary

This notebook looks at feature importance using word frequencies in comparison to classification features. + Remove terms that are only digits. + Keep words that appear in < 10% of non-transfused documents, AND also change by more than 30% between the 2 groups (non transfused - transfused) + or are only in the transfused group + Inner Join with the vocabulary from 2.3 + save as all_filtered_features.csv

[1]:
```
import pandas as pd
import pickle

import os
import re
import sys

import numpy as np
from datetime import datetime
from sklearn.feature_extraction.text import CountVectorizer

import psycopg2
from sqlalchemy import create_engine

%matplotlib inline
```

[2]:
```
POSTGRES_CONNECT = os.environ.get("POSTGRES_CONNECT")
POSTGRES_ENGINE = os.environ.get("POSTGRES_ENGINE")

conn = psycopg2.connect(POSTGRES_CONNECT)
engine = create_engine(POSTGRES_ENGINE)

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

[3]:
```
path ="./"
```

1.0.1 Read in Notes

[4]:
```
mimic_transfused_notes = pd.read_sql("""select * from mimiciii.
 ↪transfused_notes_unique""", engine)
mimic_ctrl_notes = pd.read_sql("""select * from mimiciii.ctrl_notes_unique""",␣
 ↪engine)
```

1.0.2 Vectorize Dataframe A

FILTER: only keep words present in <10% of documents and words that are not just digits

[5]:
```
def get_words_a(df):

    # count vectorize
    vect = CountVectorizer(lowercase=True, strip_accents=None, analyzer =␣
 ↪'word', max_df = 0.1)
    vect_dat = vect.fit_transform(df['text'])
    vocab = vect.get_feature_names()

    return vect_dat, vocab
```

[6]:
```
def no_ints(vocab):
    '''remove terms that are only comprised of digits.
    INSERT: dtm = sparse document term matrix from vectorizer
            vocab = the .get_feature_names from vectorizer (sklearn)
    OUT: filtered dtm and vocab'''

    filtered_vocab = [x for x in vocab if not x.isdigit()]
    print('removed ' + str(len(vocab)-len(filtered_vocab)))

    return filtered_vocab
```

[9]:
```
def filter_sparse(dtm, vocab, filtered_vocab):
    # get index of terms we want to keep
    # this is slow, just give it some time (~20min?)
    vocab_idx = [vocab.index(k) for k in filtered_vocab]

    # handle sparse matrix
    sorted_dtm = dtm.sorted_indices()
    # filter the document term matrix
    filtered_dtm = sorted_dtm.T[vocab_idx]

    return filtered_dtm
```

[13]:
```
def wrapper_a(df):
    vect, vocabulary = get_words_a(df)
    filtered_vocabulary = no_ints(vocabulary)
```

```
    filtered_vect = filter_sparse(vect, vocabulary, filtered_vocabulary)
    return filtered_vect, filtered_vocabulary
```

[14]:
```
ctrl_vect, ctrl_vocab = wrapper_a(mimic_ctrl_notes)
```

```
removed 99806
```

[15]:
```
# to save
with open(path + 'ctrl_vect.pickle', 'wb') as picklefile:
    pickle.dump(ctrl_vect, picklefile)

with open(path + 'ctrl_vocab.pickle', 'wb') as picklefile:
    pickle.dump(ctrl_vocab, picklefile)
```

[16]:
```
#to load
with open(path1 + path2 + 'ctrl_vocab.pickle','rb') as f:
    ctrl_vocab=pickle.load(f)

with open(path1 + path2 + 'ctrl_vect.pickle','rb') as f:
    ctrl_vect=pickle.load(f)
```

1.1 then vectorize dataframe B (transfused) data using the vocabulary from A (control)

[17]:
```
def get_words_b(df, vocab_in):

    # count vectorize
    vect = CountVectorizer(lowercase=True, strip_accents=None, analyzer =␣
 ↪'word', vocabulary = vocab_in)
    vect_dat = vect.fit_transform(df['text'])
    vocab_out = vect.get_feature_names()

    # transpose to match filtered
    sorted_dtm = vect_dat.sorted_indices()
    transposed_dtm = sorted_dtm.T

    return transposed_dtm, vocab_out
```

[18]:
```
xf_vect, xf_vocab = get_words_b(mimic_transfused_notes, ctrl_vocab)
```

[19]:
```
# to save
with open(path1 + path2 + 'xf_vect.pickle', 'wb') as picklefile:
    pickle.dump(xf_vect, picklefile)

with open(path1 + path2 + 'xf_vocab.pickle', 'wb') as picklefile:
    pickle.dump(xf_vocab, picklefile)
```

```
[4]: #to load
with open(path1 + path2 + 'xf_vocab.pickle','rb') as f:
    xf_vocab=pickle.load(f)

with open(path1 + path2 + 'xf_vect.pickle','rb') as f:
    xf_vect=pickle.load(f)
```

2 compare

```
[28]: def compare_freq(matrix_a, vocab_a,  label_a, matrix_b, vocab_b, label_b,␣
 ↪threshold=0.3):
    '''calc the sum of each term (matrix should be transposed after␣
 ↪vectorization)
    calc percentage of change between a and b
    filter out <threshold% change'''

    sum_a = matrix_a.sum(axis=1)
    sum_b = matrix_b.sum(axis=1)

    both= pd.DataFrame(sum_a, index=vocab_a, columns=[label_a])
    both[label_b]= sum_b

    # combine
    both['percent_change'] = (both[label_b].map(int) - both[label_a].map(int)) /
 ↪ both[label_a].map(int)

    # FILTER keep words that changed more than or eq to %threshold
    both = both[both['percent_change'] >= threshold]

    return both
```

```
[21]: percent_df = compare_freq(ctrl_vect, ctrl_vocab, 'ctrl', xf_vect, xf_vocab,␣
 ↪'xf', 0.30)
```

3 get new words

```
[26]: # vectorize the full dataset to get words that are new for the  transfused when␣
 ↪compared to ctrl)

def get_new_word_freqs(df_a, label_a, df_b, label_b):

    # count vectorize a
    vect_a = CountVectorizer(lowercase=True, strip_accents=None, analyzer =␣
 ↪'word')
    vect_dat_a = vect_a.fit_transform(df_a['text'])
```

```
    vocab_a = vect_a.get_feature_names()

    # count vectorize b
    vect_b = CountVectorizer(lowercase=True, strip_accents=None, analyzer =␣
↪'word')
    vect_dat_b = vect_b.fit_transform(df_b['text'])
    vocab_b = vect_b.get_feature_names()

    # compare
    newwords = list(set(vocab_b) - set(vocab_a))

    # FILTER out digits and empty strings
    newwords_filtered = [x for x in newwords if not x.isdigit()]
    print('removed ' + str(len(newwords)-len(newwords_filtered)))

    filtered_vect = filter_sparse(vect_dat_b, vocab_b, newwords_filtered)
    sum_b = filtered_vect.sum(axis=1)
    sum_df = pd.DataFrame(sum_b, index=newwords_filtered, columns=[label_b])
    sum_df[label_a] = 0

    return sum_df
```

3.0.1 get words that appear in $<10\%$ of control-documents, that also change by more than 30% from non-transfused to transfused

3.0.2 OR new in transfused documents

[33]:
```
newwords_df = get_new_word_freqs( mimic_ctrl_notes, 'ctrl',␣
↪mimic_transfused_notes, 'xf')
newwords_df.head()
```

```
removed 15283
```

[33]:

	xf	ctrl
bfmw	1	0
qnoontime	1	0
dicline	1	0
apprpo	1	0
smoul	1	0

[34]:
```
newwords_df.sort_values('xf',ascending=False).head(50)
```

[34]:

	xf	ctrl
4uffp	144	0
percreta	91	0
1urbc	89	0
closures	83	0

accreta	80	0
normocarb	75	0
prbcx1	68	0
40ppm	63	0
pcells	58	0
defibritide	57	0
prbcx2	56	0
haps	55	0
trisodium	55	0
txn	54	0
refeed	47	0
esmark	43	0
400mcgs	43	0
cisaturcurium	41	0
3uffp	41	0
cytogam	40	0
cvhd	40	0
debranching	39	0
rectocolectomy	39	0
colure	37	0
cvvhf	37	0
acineterbacter	37	0
acidiotic	36	0
meshing	36	0
1ffp	36	0
octeretide	36	0
gastrrectomy	36	0
fusarium	35	0
sqcjojqj	35	0
dncjojqj	34	0
nasoduodenal	34	0
x1uprbcs	33	0
cvvdh	33	0
dialys	32	0
pcco	32	0
2un	31	0
recirc	31	0
angioembolization	30	0
coagulator	30	0
faschiotomy	30	0
methergine	30	0
iuprbc	30	0
nsiad	30	0
0portex	30	0
23st	29	0
rotarest	29	0

[32]:
```
newwords_df.sort_values('ctrl',ascending=False).head(50)
```

[32]:

	ctrl	xf
degranulation	183	0
ghb	167	0
gbl	98	0
achondroplasia	92	0
melas	65	0
hajdu	65	0
ifedsfq	61	0
hypereosinophilic	52	0
rickham	51	0
hq	40	0
lice	38	0
rufinamide	37	0
qwl	37	0
ycjojqj	35	0
ocjojqj	35	0
53yf	33	0
gastrocrom	32	0
cactus	32	0
intralipid	30	0
duchenne	28	0
flatbush	28	0
breakable	28	0
6ng	27	0
quadrantanopsia	27	0
lhm	26	0
coilinganesthesia	26	0
mcds	26	0
klcjojqj	25	0
qwcjojqj	24	0
viewscope	23	0
sfq	23	0
bornewood	23	0
mango	22	0
baycove	22	0
agraphia	22	0
ketostix	21	0
chyli	21	0
dofetelide	21	0
uconn	20	0
deac4	19	0
nasah	18	0
chanmber	17	0
ifedp	17	0
arrhythimia	17	0

```
springhouse          17   0
dyslexia             17   0
dyh                  17   0
ictally              17   0
stetting             17   0
decarboxylase        17   0
```

[35]:
```python
newwords_df.to_csv(path1 + 'newwords.csv')
```

[54]:
```python
newwords_df.reset_index(inplace=True)

percent_df.reset_index(inplace=True)

# removing any overlap
newwords_filtered =newwords_df[~newwords_df['index'].isin(percent_df['index'])]

# combining new words with words from pre/post meeting frequency threshold
allfreq = pd.concat([percent_df, newwords_filtered], axis = 0, sort = True)

# filling empty 'pre' counts with 0
allfreq['ctrl'].fillna(0, inplace = True)

allfreq.rename(columns = {'index': 'vocab'}, inplace = True)

allfreq['percent_change'] = allfreq.percent_change*100
```

[55]:
```python
# spot check pfi
allfreq[allfreq['vocab']=='4uffp']
```

[55]:

	ctrl	vocab	percent_change	xf
111550	0	4uffp	NaN	144

[56]:
```python
allfreq[allfreq['vocab']=='prbcs']
```

[56]:

	ctrl	vocab	percent_change	xf
40554	876	prbcs	1625.799087	15118

3.1 Comparing to Classification Features

[59]:
```python
features = pd.read_csv(path + 'final_classification_features.csv')
features.drop(columns= 'Unnamed: 0', inplace = True)
```

[60]:
```python
features.head()
```

[60]:

	vocab	NB_total_hadmids	NB_total_count_freq	NB_ratio	LR_l2_coef	\
0	rvad	22.0	380.0	0.748991	NaN	
1	apml	23.0	421.0	0.759843	NaN	

```
2  percreta             6.0                 91.0  0.769518          NaN
3   accreta            11.0                 80.0  0.782175          NaN
4      4923             4.0                 70.0  0.784472          NaN

   LR_l2_total_count_freq  LR_l2_total_hadmids  LR_l1_coef  chi2_pval_p_05
0                     NaN                  NaN         NaN   1.720376e-108
1                     NaN                  NaN         NaN   4.257674e-119
2                     NaN                  NaN         NaN    1.452410e-27
3                     NaN                  NaN         NaN    1.977585e-24
4                     NaN                  NaN         NaN    1.399844e-20
```

Taking intersection of classification features and frequency threshold words

[61]:
```
feats = features.merge(allfreq, on = 'vocab', how = 'inner')
```

[62]:
```
feats.shape
```

[62]:
```
(41664, 12)
```

[63]:
```
feats.isnull().sum()
```

[63]:
```
vocab                         0
NB_total_hadmids          36945
NB_total_count_freq       36945
NB_ratio                  36945
LR_l2_coef                39183
LR_l2_total_count_freq    39183
LR_l2_total_hadmids       39183
LR_l1_coef                39280
chi2_pval_p_05              115
ctrl                          0
percent_change            12625
xf                            0
dtype: int64
```

[ ]:
```
feats.
↪drop(columns=['NB_total_hadmids','NB_total_count_freq','LR_l2_total_count_freq','LR_l2_tota

feats.rename(columns={'ctrl':'freq_control','xf':
↪'freq_transfused'},inplace=True)

feats.head()
```

[104]:
```
feats.to_csv(path +'all_filtered_features.csv')
```