# 3.0_topic_model_on_filtered_vocab

May 26, 2021

## 1 3.0_topic_model_on_filtered_vocab

### 1.1 topic modeling on transfused admissions

This notebook demonstrates topic modeling on the transfused group, but can be used on any dataset. + python 3.7.x

[1]:
```
import pickle
import time
import os
import re

import numpy as np
import pandas as pd
from datetime import datetime
import sys

from nltk.tokenize import wordpunct_tokenize

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer,␣
↪TfidfTransformer
from sklearn.decomposition import LatentDirichletAllocation

from scipy.sparse import csr_matrix, vstack

import pyLDAvis, pyLDAvis.sklearn
from IPython.display import display
from tqdm import tnrange, tqdm_notebook
from time import sleep

from importlib_metadata import version

%matplotlib inline
import matplotlib.pyplot as plt

# things to connect to the posgres database
import psycopg2
from sqlalchemy import create_engine
```

```
POSTGRES_CONNECT = os.environ.get("POSTGRES_CONNECT")
POSTGRES_ENGINE = os.environ.get("POSTGRES_ENGINE")
conn = psycopg2.connect(POSTGRES_CONNECT)
cur = conn.cursor();
cur.execute("""SET search_path = mimiciii;""")
engine = create_engine(POSTGRES_ENGINE)


libraries =␣
 ↪['pandas','sqlalchemy','psycopg2','tqdm','scipy','numpy','nltk','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))
```

```
/Users/summerrankin/opt/anaconda3/envs/env_py3/lib/python3.7/site-
packages/past/types/oldstr.py:5: DeprecationWarning: Using or importing the ABCs
from 'collections' instead of from 'collections.abc' is deprecated, and in 3.8
it will stop working
  from collections import Iterable
```

```
last ran:  2020-08-04 07:18:14.409672
Python Version: 3.7.3 (
operating system: darwin
pandas version: 0.24.2
sqlalchemy version: 1.3.3
psycopg2 version: 2.7.6.1
tqdm version: 4.32.1
scipy version: 1.2.1
numpy version: 1.16.2
nltk version: 3.4
matplotlib version: 3.0.3
```

Get the date (which will become the filename) and set a path for the output

[ ]:
```
from datetime import date
y = date.today().year
d = date.today().day
m = date.today().month
day = str(y) + str(m) + str(d) + '_'

path1 ="./"
```

[ ]:
```
# vectorization and topic modeling functions
def get_xf_notes():
    """
```

```
    Function to retrieve the data for transfused patients from the mimic␣
↪postgres database.
    Returns a pandas dataframe of the hadmid, and text of the concatenated␣
↪notes.
    """
    mimic_notes = pd.read_sql("""SELECT * FROM mimiciii.transfused_notes_unique␣
↪""", engine)

    return mimic_notes
#============================================================================================
def clean_text(text, xf_term_ver=9):
    """
    Takes in a corpus of documents and cleans. Needs multiple docs.

    IN: corpus of documents

    1. remove id masks
    2. tokenize into words using wordpunct
    3. lowercase and remove stop words
    4. put into a list for the modeling

    OUT: cleaned text = a list (documents) of lists (cleaned words in each doc)
    """
    stop = ['I']

    cleaned_text = []

    for post in tqdm_notebook(text):
        cleaned_words = []

        #remove masked ids
        clean_ids = re.sub(re.compile(r'\[\*\*(.*?)\*\*\]'), ' ', post)

        # tokenize into words
        for word  in wordpunct_tokenize(clean_ids):

            # lowercase and throw out any words in stop words (doing it here␣
↪and later makes it faster)
            if word.strip().lower() not in stop:

                    # put into a list of words for each document
                cleaned_words.append(word.strip().lower())

        # keep corpus of cleaned words for each document
        cleaned_text.append(' '.join(cleaned_words))

    return cleaned_text
```

```
#========================================================================================
def topic_mod_mimic(vectorizer, vect_data, model='LDA',␣
↪topics=20,iters=5,no_top_words=50):
    """
    Run a topic modeling function and display the top terms from each topic.

    INPUT:
    vectorizer=the model from the vectorization (needs to be from sklearn)
    vect_data=output from the vectorization step (sklearn)
    model='LDA' or 'NMF'
    topics=number of topics
    iters=number of iterations
    no_top_words=number of top terms to print for each topic

    OUTPUT:
    mod=the trained topic model
    mod_dat=output of the topic model
    topics=number of topics
    iters=number of iterations
    """

    if model == 'NMF':
        mod = NMF(n_components=topics,
                  # init='nndsvd',
                 # max_iter=iters,
                  random_state=42,
                   verbose=False,
                   beta_loss='kullback-leibler',
                    solver='mu',
                    max_iter=1000,
                    alpha=.1,
                    l1_ratio=.5)

    else:
        mod = LatentDirichletAllocation(n_components=topics,
                                        max_iter=iters,
                                        random_state=42,
                                        learning_method='online',
                                        n_jobs=-1)

    mod_dat = mod.fit_transform(vect_data)


    # to display a list of topic words and their scores
    def display_topics(model, feature_names, no_top_words):
        for ix, topic in enumerate(model.components_):
            print("Topic ", ix)
```

```
            print(" ".join([feature_names[i]
                            for i in topic.argsort()[:-no_top_words - 1:-1]]) +␣
↪'\n')

    display_topics(mod, vectorizer.get_feature_names() , no_top_words)


    return mod, mod_dat, topics, iters
```

## 2 1.0 Import data

Get the transfused notes from postgres

[ ]:
```
mimic_notes = get_xf_notes()
corpus=mimic_notes.text.values
```

## 3 1.1 Clean

[ ]:
```
clean_corpus = clean_text(corpus)
```

## 4 1.2 Import features

Retrieve the list of features created in the previous steps

[ ]:
```
features1 = pd.read_csv(path1 + 'all_filtered_features.csv')
features = features1.vocab.to_list()
```

## 5 1.3 Vectorize

Count vectorize the data using the existing vocabulary that was created in steps 2.0, 2.1, 2.2, 2.3, 2.4

[ ]:
```
vect_mod = CountVectorizer(
    stop_words=None,
    lowercase=True,
    strip_accents='ascii',
    vocabulary=features
    )
vect_data = vect_mod.fit_transform(clean_corpus)
```

## 6 Set Hyperparameters

Set the type of topic model and the number of topics

[ ]:
```
model='LDA'
num = 45
```

## 7 1.4 Topic model

Run the topic model using the vectorized data and model from the previous step

[ ]:
```
mod, mod_dat, topic_n, iter_n = topic_mod_mimic(
    vect_mod,
    vect_data,
    model=model,
    topics=num,
    iters=10,
    no_top_words=15)
```

## 8 1.5 Display

Display the topic modeling results with an interactive pyPLAvis plot. This can be saved as an html and opened in any browser

[ ]:
```
import pyLDAvis, pyLDAvis.sklearn
from IPython.display import display

# Setup to run in Jupyter notebooks
pyLDAvis.enable_notebook()

 # Create the visualization
vis = pyLDAvis.sklearn.prepare(
    mod,
    vect_data,
    vect_mod,
    sort_topics=False,
    mds='mmds')

# view it in the notebook
display(vis)
```

[ ]:
```
def vis_sav( path1, number, vis, model = 'LDA'):
    """
    Save the .html output from pyLDAvis
    """

    pyLDAvis.save_html(
        vis,
        path1  + day + model + str(number) + '_filtered_v1.html'
        )
```

```
vis_sav(path1, num, vis )
```

## 9 1.6 Save

Save the models and results from previous steps

[ ]:
```
def save_model(path1, number, vec_model, vect_dat, lda_model=0, model_dat=0, ␣
↪model = 'LDA', stops='none'):
    """
    Save the models and results from the vectorization and topic modeling.
    """
    # save the count vectorizer model
    with open(path1 + day + model + str(number) + '_filtered_v1vect_model.pkl',␣
↪'wb') as picklefile:
        pickle.dump(vec_model, picklefile)

    # save the vectorized data (sparse matrix)
    with open(path1 + day + model + str(number) + '_filtered_v1vect_data.pkl',␣
↪'wb') as picklefile:
        pickle.dump(vect_dat, picklefile)

    # save parameters of the topic model as a text file
    with open(path1 +  day + model + str(number) + '_filtered_v1_params.txt',␣
↪"w") as parameters_f:
        print('model: '+ str(lda_model) + \
              '\n Vectorizer: ' + str(vect_mod) + \
              '\n stop words: '+ stops  \
              , file = parameters_f)

    # save the actual topic model
    with open(path1 + day + model + str(number) + '_filtered_v1_model.pkl',␣
↪'wb') as picklefile:
        pickle.dump(lda_model, picklefile)

     #save the output from the topic model
    with open(path1 + day + model + str(number) + '_filtered_v1_data.pkl',␣
↪'wb') as picklefile:
        pickle.dump(model_dat, picklefile)
```

[ ]:
```
save_model(
    path1,
    num,
    vect_mod,
    vect_data,
    mod,
```

```
    mod_dat,
    stops='vocab',
    model = model)
```

[ ]:
```
def save_topic_scores(path1, number, mod2, vect_mod, no_top_words, model):
    ''' to save the raw pesudocount for the top xx words/terms from _components␣
↪of each topic'''

    ff = vect_mod.get_feature_names()

    topicsdf = pd.DataFrame()

    for ix, topic in enumerate(mod2.components_):

        topicsdf.loc[:,'topic_' +str(ix) ] = [ff[i] for i in topic.argsort()[:
↪-no_top_words - 1:-1]]
        topicsdf.loc[:,'topic_' +str(ix) + '_scr' ] = [topic[i] for i in topic.
↪argsort()[:-no_top_words - 1:-1]]

    topicsdf.to_csv(
        path1 + day + model + str(number) + '_filtered_v1_' + str(no_top_words)␣
↪+ '_words_scores.csv',
        float_format='%.3f'
        )
```

[ ]:
```
save_topic_scores(
    path1,
    num,
    mod,
    vect_mod,
    9499,
    model=model
    )
```

[ ]:
```
def save_word_scores(path1, number, mod2, vect_mod, no_top_words, model):
    ''' to save the proba score for the top xx words from _components of each␣
↪topic'''

    ff = vect_mod.get_feature_names()

    topicsdf = pd.DataFrame()

    proba_scores = mod2.components_ / mod2.components_.sum(axis=1)[:,np.newaxis]

    for ix, topic in enumerate(proba_scores):
```

```
        topicsdf.loc[:,'topic_' +str(ix) ] = [ff[i] for i in topic.argsort()[:
↪-no_top_words - 1:-1]]
        topicsdf.loc[:,'topic_' +str(ix) + '_scr' ] = [topic[i] for i in topic.
↪argsort()[:-no_top_words - 1:-1]]

    topicsdf.to_csv(path1 + day + model + str(number) + '_filtered_v1_' +␣
↪str(no_top_words) + '_words_proba.csv',float_format='%.5f')
    return topicsdf
```

[ ]:
```
words = save_word_scores(
    path1,
    num,
    mod,
    vect_mod,
    9499,
    model=model
    )
words.head()
```

Save all topic scores for each document in a .csv

[ ]:
```
scores = pd.DataFrame(mod.components_)
scorest = scores.T
scorest.to_csv(
    path1 + day + model + str(num) + '_filtered_v1_all_topic_scores_hadmids.
↪csv')
```

Format all scores for easy plotting

[ ]:
```
scores_all = mod.components_ / mod.components_.sum(axis=1)[:, np.newaxis]
scores_all = scores_all.reshape([1,-1])
scores1=pd.DataFrame(mod.components_ / mod.components_.sum(axis=1)[:, np.
↪newaxis])
scores1.head()
```

## 10 1.7 Plots

Plot distributions of the topic scores

[ ]:
```
def plot_hist(df, fig_title, save_name, num, log_y=False, bins=100):
    """
    Plot and save a histogram of a pandas dataframe
    """
    fig = plt.figure(dpi=300)  #figsize=(8, 8),

    plt.hist(df, density=False, bins=bins, log=log_y)
    plt.title(fig_title, fontsize=10)
```

```
    plt.yticks(fontsize=7)
    plt.xticks(fontsize=7)
    plt.ylabel('Frequency', fontsize=7)
    plt.savefig(path1 + day + model + str(num) + '_filtered_v1'+ save_name + '.
↪png')
```

[ ]:
```
sscores_df = pd.DataFrame(scores_all)
plot_hist(sscores_df.iloc[0,:], fig_title='Histogram of All Term Scores ('+␣
↪str(num) +' topics) ', \
        save_name='loghist_term_proba', num=num, log_y=True, bins=100)
```

[ ]:
```
plot_hist(sscores_df.iloc[0,:], fig_title='Histogram of All Term Probas ('+␣
↪str(num) +' topics) ', \
        save_name='hist_term_proba', num=num, log_y=False, bins=500)
```

[ ]:
```
scores1.iloc[22,:].plot.hist(
    figsize=(20,15),
    logy=True,
    use_index=True,
    bins=(100),
    title='histogram of terms probability score for topic 22 ('+ str(num) + '␣
↪topics)'
    );
```

[ ]:
```
scores1.iloc[22,:].plot.hist(
    figsize=(20,15),
    logy=False,
    use_index=True,
    bins=(100),
    title='histogram of all 40k terms probability score for topic 22 ('+␣
↪str(num) + ' topics)'
    );
```

## 11 2.0 Assign the Maximum Topic to each document (admission)

[ ]:
```
def open_model( path1, number, model='LDA'):
    """
    Function to open a saved model and the vectorized output from the model.
    """
    # vectorizer
    with open(path1 + '/'+ model + '_' + str(number) +  '/' + day + 'vect_model.
↪pkl', 'rb') as picklefile:
        vec_model =pickle.load(picklefile)
    #vectorized data
```

```
    #with open(path1 + '/'+ model + '_' + str(number) + '/' + day + 'vect_data.
↪pkl', 'rb') as picklefile:
    #    vect_dat=pickle.load(picklefile)

    # topic model
    with open(path1 + '/'+ model + '_' + str(number) + '/' + day + model +␣
↪'_model.pkl', 'rb') as file:
        lda_model=pickle.load(file)
    # topic model results
    with open(path1 + '/'+ model + '_' + str(number) + '/' + day + model +␣
↪'_data.pkl', 'rb') as picklefile:
        mod_dat = pickle.load(picklefile)

    return mod_dat, vec_model, lda_model
```

## 12 2.1 Load the models and results

[ ]:
```
num=45
mod_dat, vect_model, mod = open_model(path1, num, model = 'LDA')
```

## 13 2.2 Assign Max Topic

Get the topic number with the maximum score for each document and save as 'max_topic' column Get the proba/score for that maximum topic for each document and save as 'max_topic_val' column

[ ]:
```
mimic_notes['max_topic'] = mod_dat.argmax(axis=1)
mimic_notes['max_topic_val'] = np.amax(mod_dat, axis=1)

# merge these max cols and notes with all the topic scores
tdf = pd.DataFrame.from_records(mod_dat)
mm = pd.concat([mimic_notes,tdf],axis=1)
mm.head()
```

## 14 2.3 Save

Save the maximum topic for each document/hadm_id

[ ]:
```
all_hadmids = mm.loc[:,['hadm_id','max_topic','max_topic_val']]
all_hadmids.to_pickle(path1  + day + model + str(num) +␣
 ↪'_filtered_v1_max_topic_all_hadmids.pkl')
all_hadmids.to_csv(path1 + day + model + str(num) +␣
 ↪'_filtered_v1_max_topic_all_hadmids.csv')
```

## 15 2.4 Plot

```
[ ]: plot_hist(mm.max_topic_val, fig_title='Histogram of Maximum Topic Coef ('+␣
     ↪str(num) +' topics) ', \
              save_name='_max_topic_all_hadmids', num=num, log_y=False)#, bins=100)
```

## 16 2.5 Select documents for further Review

Get the documents where the max topic for the document is less than the threshold set (.15). This denotes that the document fit into many different topics which indicates that the patient is complex and/or should be reviewed by an expert or clinician to evaluate for adverse events.

Print the length to see how many documents are captured by the threshold. This threshold may need to be adjusted based on the number of topics, or specific dataset. The idea is to capture the 'left tail' of the maximum topic distribution. The amount of that tail that is captured should be based on how many documents the researcher is looking to review.

This list will also include documents that did not fit into any topic (if present) and will have a max topic score that is at the min topic score value (instead of zero, there is typically a constant that is assigned to each topic that is not present in a document). We recommend reviewing these as well.

```
[ ]: len(mm[mm.max_topic_val<=.15])
     amb_topic = mm[mm.max_topic_val<=.15]
     amb_topic = amb_topic.sort_values('max_topic_val',ascending=True)
```

## 17 2.6 Save

Save these hadm_ids with a low maximum topic for SME review

```
[ ]: amb_hadmids = amb_topic.loc[:,['hadm_id','max_topic','max_topic_val']]
     amb_hadmids.to_pickle(path1 + day + model + str(num) +␣
     ↪'_filtered_v1_thresh_15_outlier_hadmids.pkl')
     amb_hadmids.to_csv(path1 + day + model + str(num) +␣
     ↪'_filtered_v1_thresh_15_outlier_hadmids.csv')
```

## 18 2.7 Plot

Plot the distribution of the scores for all documents for a single topic (35)

```
[ ]: mm.iloc[35,4:].plot.bar(figsize=(10,10),fontsize=10, title='probability per␣
     ↪topic '+ str(num));
```

Plot the topic distribution for a single patient (insert an hadm_id where there are xxxxxx)

```
[ ]: one_pt = amb_topic[amb_topic.hadm_id==xxxxxx]
     one_pt
```

[ ]:
```
one_pt.iloc[0,4:].plot.bar(figsize=(10,10),fontsize=10, title='probability per␣
↪topic '+ str(num));
```

[ ]:
```
one_pt1 = amb_topic[amb_topic.hadm_id==xxxxxx]
one_pt1
```

[ ]:
```
one_pt1.iloc[0,4:].plot.bar(figsize=(10,10),fontsize=10, title='probability per␣
↪topic '+ str(num));
```