2.3_classification_vocabs

May 26, 2021

1 2.3 Classification Vocabulary for Transfused

- run classification using Logistic Regression (l1 penalty) and filter out low
- load cleaned Naive Bayes vocab from 2.1.3
- load cleaned Logistic Regression (l2 penalty) from 2.2.3
- run chi2 on full vocab and filter for terms p=<.05
- put together in one df and save

[ ]:
```
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.preprocessing import scale, LabelEncoder

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

from sklearn import naive_bayes
from sklearn.linear_model import  LogisticRegression
from sklearn.feature_selection import SelectKBest, chi2

from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import classification_report, make_scorer, accuracy_score

import matplotlib
import matplotlib.patches as mpatches
import matplotlib.cm as cm
import matplotlib.pyplot as plt
%matplotlib inline

import time
import math
from datetime import datetime
import sys

import numpy as np
import pickle
import pandas as pd

from scipy.sparse import csr_matrix, vstack
```

```
from importlib_metadata import version
```

[ ]:
```
libraries = ['pandas','numpy','scikit-learn', 'scipy','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))
```

[ ]:
```
def feature_unpickle(path):
    mat = []
    for i in range(0,10):
        with open(path+'textfeatures_mat'+str(i+1)+'.pickle', 'rb') as f:
            mat.append(pickle.load(f, encoding='latin1'))
    mat=vstack(mat)

    q=[mat]
    with open(path+'textfeatures_vocab.pickle', 'rb') as f:
        vocab=pickle.load(f, encoding='latin1')
        q.append(vocab)
    with open(path+'textfeatures_id.pickle', 'rb') as f:
        ids=pickle.load(f, encoding='latin1')
        q.append(ids)
    with open(path+'textfeatures_source.pickle', 'rb') as f:
        source=pickle.load(f, encoding='latin1')
        q.append(source)
    return q
```

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

2 I. Data Preparation

2.0.1 Prepare Train/Test

[ ]:
```
def shuffle_split(X, y):

    rs = StratifiedShuffleSplit(n_splits=1, random_state=42, test_size=0.25,␣
↪train_size=None)

    for train_index, test_index in rs.split(X,y):
        print('TRAIN:', train_index, "TEST:", test_index)

    X_train = X[train_index,:]
    X_test = X[test_index,:]
```

```
    y_train = y[train_index]
    y_test = y[test_index]

    return X_train, X_test, y_train, y_test
```

[ ]:
```
r=feature_unpickle(path)
XX = r[0] # sparse document-term. matrix
y = r[3] # label (transfused/ non-transfused)

vocab = r[1]
hadmids = r[2] # hadm_ids for each
```

[ ]:
```
# make train test split
X_train, X_test, y_train, y_test = shuffle_split(XX, y)
```

3 II. Models

[ ]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.summer):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=30)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, fontsize=20)
    plt.yticks(tick_marks, classes, fontsize=20)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center",
                 color="white" if cm[i, j] < thresh else "black", fontsize=40)

    plt.tight_layout()
    plt.ylabel('True label', fontsize=30)
    plt.xlabel('Predicted label', fontsize=30)

    return plt
```

3.1 Logistic Regression

- l2 penalty was calculated already in step 2.0, only need to redo for l1

- decision_function will give us confidence intervals per sample in predict

[ ]:
```
def train_LR(X_train, X_test, y_train, y_test, penalty = ['l1', 'l2']):

    LRmodel = LogisticRegression(verbose=1, penalty = penalty)
    LRmodel.fit(X_train, y_train)

    print( "Logistic regression Scoring on test set")

    LR_pred = LRmodel.predict(X_test)
    cr = classification_report(y_test, LR_pred)

    print (cr)

    cm = confusion_matrix(y_test, LR_pred)
    fig = plt.figure(figsize=(8, 8))
    plot = plot_confusion_matrix(cm, classes=['Non','transfused'],␣
 ↪normalize=False)

    plt.show()

    return LRmodel, LR_pred
```

[ ]:
```
def full_LR(X, y, penalty = ['l1', 'l2']):

    LRmodel = LogisticRegression(verbose=1, penalty = penalty)
    LRmodel.fit(X, y)

    print( "Logistic regression Scoring on test set")

    LR_pred = LRmodel.predict(X)
    cr = classification_report(y, LR_pred)

    print (cr)

    cm = confusion_matrix(y, LR_pred)
    fig = plt.figure(figsize=(8, 8))
    plot = plot_confusion_matrix(cm, classes=['non','transfsused'],␣
 ↪normalize=False)

    plt.show()

    return LRmodel, LR_pred
```

[ ]:
```
lr_l2_model, lr_12_pred = train_LR(X_train, X_test, y_train, y_test, 'l2')
```

[ ]:
```
# to train logistic regression on train/test/split
lr_l1_model, lr_l1_pred = train_LR(X_train, X_test, y_train, y_test, 'l1')
```

[ ]:
```
# to train logistic regression on entire dataset (no train/test split)
lr_l1_full_model, lr_l1_full_pred = full_LR(XX, y, 'l1')
```

[ ]:
```
lr_l2_full_model, lr_l2_full_pred = full_LR(XX, y, 'l2')
```

4 III. Create Vocab

[ ]:
```
# logistic regression top features
def get_vocab_LR(model, matrix_vocab, thresh):
    features = zip(matrix_vocab, model.coef_[0]) # zip vocabulary and model␣
 ↪betas
    df = pd.DataFrame(features).sort_values(by=1, ascending=False) # tail

    top = df.head(thresh)

    return top
```

5 Load NB Vocab

[ ]:
```
# to load the NB top 5k terms that have been cleaned (transfused terms removed,␣
 ↪phrases concatenated)
NB_mat = pd.read_csv(path + 'NB_top_4879_terms_only_dist.csv')
NB_mat.head()
```

6 Load LR l2 vocab

[ ]:
```
# to load the NB top 5k terms that have been cleaned (transfused terms removed,␣
 ↪phrases concatenated)
lr_l2_mat = pd.read_csv(path + 'LR_top_5000_terms_only.csv')
lr_l2_mat.head()
```

[ ]:
```
# get vocab from all three models

NBvocab = NB_mat#.loc[:,'feature']
LRvocab_l2 = lr_l2_mat#.loc[:,'feature']
LRvocab_l1 = get_vocab_LR(lr_l1_full_model, vocab, 5000)
LRvocab_l1.head()
```

[ ]:
```
LRvocab_l1[1].describe()
```

7 filtering

[ ]:
```
# getting rid of lasso coefs less than 0
# does not actually get rid of anything b/c coefs are high for this␣
↪classification
# increasing to .2 b/c it's in 70%ish IQR
LRvocab_l1 = LRvocab_l1[LRvocab_l1[1]>0.2]
LRvocab_l1.shape
```

[ ]:
```
LRvocab_l1.rename(columns = {0: 'vocab', 1: 'LR_l1_coef'},inplace = True)
LRvocab_l2.rename(columns = {'feature': 'vocab', 'coef':␣
↪'LR_l2_coef','total_hadmids':'LR_l2_total_hadmids','total_count_freq':
↪'LR_l2_total_count_freq'},inplace = True)
NBvocab.rename(columns = {'feature': 'vocab','ratio': 'NB_ratio',␣
↪'total_hadmids':'NB_total_hadmids','total_count_freq':
↪'NB_total_count_freq'}, inplace = True)
```

7.0.1 Select K Best

chi squared test on full dataset (not 10k vocab)
gives pval for every single feature (b/c we run on full vocab)
most are nan

[ ]:
```
selector = SelectKBest(chi2)
selector.fit(XX, y) # fit to entire datase
scores = selector.pvalues_ # get all the features p values
df = pd.DataFrame(scores) # put scores into df
df['vocab'] = vocab # match to vocab from matrix
df.dropna(inplace = True) # drop nulls - most of the select k best = null
df.rename(columns = {0: 'chi2_pval_p_05'},inplace = True)
df = df[df['chi2_pval_p_05'] <= 0.05] # keeping only p values less than or␣
↪equal to 0.05
df.shape
```

7.0.2 Joining Vocab & Chi2

Combine vocabulary from the top coefficients from logistic regression (ridge and lasso), Naive Bayes, and select K Best

[ ]:
```
del alldf
```

[ ]:
```
alldf = NBvocab.merge(LRvocab_l2, on = 'vocab', how ='outer') # merge NB, ridge␣
↪LR
alldf = alldf.merge(LRvocab_l1, on = 'vocab', how = 'outer') # merge lasso LR
alldf = alldf.merge(df, on ='vocab', how = 'outer') # merge chi2 significant
#alldf.drop(columns = [0, 1], inplace = True)
alldf.isnull().sum()
```

[ ]:
```
alldf.head()
```

[ ]:
```
print(len(alldf))
```

[ ]:
```
alldf.to_csv(path + 'final_classification_features.csv')
```

7.1 Save Model

[ ]:
```
def save_model(model,
               y_test,
               Y_pred,
               path,
               modeltype = ['LR', 'NB', 'RF'],
               save_model = False,
               matrix_vocab = vocab,
               thresh = 5000):


    # creating base path for saving all pieces
    basepath = path + modeltype + '/' + modeltype

    if save_model == True:
        # saving model
        with open(basepath + '.pickle', 'wb') as picklefile:
            pickle.dump(model, picklefile)

    # remaking and saving classification report as dataframe
    cr = classification_report(y_test, Y_pred, output_dict=True)
    cr = pd.DataFrame(cr).transpose()
    cr.to_csv(basepath + '_classification_report.csv')

    # remake and save confusion matrix
    cm = confusion_matrix(y_test, Y_pred)
    #fig = plt.figure(figsize=(8, 8))
    plot = plot_confusion_matrix(cm, classes=['pre','post'], normalize=False)
    plt.savefig(basepath + '_confusion_matrix.svg')

    # save vocab
    if modeltype == 'LR':
        topwords = get_vocab(model, matrix_vocab, thresh)
        topwords.to_csv(basepath + '_topwords.csv')

    #return
```

[ ]:
```
save_model(LRmodel, y_test, LR_pred,
                      path = path,
                      modeltype = 'LR', save_model = False)
```