

2.0_classification_models

May 26, 2021

1 2.0 Classification

1.1 Supervised classification models for the deduplicated vectorized data

- test train split
- run model, get scores
- plot roc auc for multiple models
- pull top 5000 features based on coef or log proba from chosen model
- make matrix based on above for further feature selection

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

from sklearn import naive_bayes
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, 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 seaborn as sns

import time
import math
import itertools
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

```

```

[ ]: startTime = datetime.now()
print (startTime)

#load input data
path = "./"
path1 = "LR/"
path2 = "NB/"

```

```

[ ]: r=feature_unpickle(path)
XX = r[0]
y = r[3]

```

```
[ ]: #Stratified b/c we have unbalanced classes (more non-transfused than
      ↳transfused),
      # and shuffle b/c the classe are grouped together in the matrix. This function
      ↳will help us to get a balanced, random selection of each class into our test
      ↳and train sections.

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]
```

```
[ ]: m = LabelEncoder()
y_test1 = m.fit_transform(y_test)
m = LabelEncoder()
y_train1 = m.fit_transform(y_train)
```

```
[ ]: 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
```

2 Models

2.0.1 Naive Bayes

```
[ ]: NBmodel = naive_bayes.MultinomialNB()
NBmodel.fit(X_train, y_train)
# save model
with open('NBmodel.pickle', 'wb') as picklefile:
    pickle.dump(NBmodel,picklefile)
#with open('NB_model.pickle', 'rb') as f:
    # NBmodel = pickle.load(f,encoding='latin1')
print ("naive bayes Scoring on test set")
Y_pred = NBmodel.predict(X_test)
cr = classification_report(y_test, Y_pred)
print( cr)
```

```
[ ]: cm = confusion_matrix(y_test, Y_pred)
fig = plt.figure(figsize=(8, 8))
plot = plot_confusion_matrix(cm, classes=['Non','Transfused'], normalize=False)
plt.savefig("NB_confusion_matrix.svg")
plt.show()
print(cm)
```

3 logistic regression

```
[ ]: LRmodel = LogisticRegression(verbose=1)
LRmodel.fit(X_train, y_train)
#save model
with open('LRmodel.pickle', 'wb') as picklefile:
    pickle.dump(LRmodel,picklefile)
#load model
#with open('LRmodel.pickle', 'rb') as f:
    # LRmodel = pickle.load(f,encoding='latin1')

print( "Logistic regression Scoring on test set")
LR_Y_pred = LRmodel.predict(X_test)
cr = classification_report(y_test, LR_Y_pred)
print( cr)
```

```
[ ]: cm = confusion_matrix(y_test, LR_Y_pred)
fig = plt.figure(figsize=(8, 8))
plot = plot_confusion_matrix(cm, classes=['Non','Transfused'], normalize=False)
plt.savefig("LR_confusion_matrix.svg")
plt.show()
print(cm)
```

3.1 Random Forest

```
[ ]: r_forest_model = RandomForestClassifier(n_estimators=10)
r_forest_model.fit(X_train, y_train)
```

```
# save model
with open('r_forest_model.pickle', 'wb') as picklefile:
    pickle.dump(r_forest_model, picklefile)

print ("Random Forest Scoring on test set")
Y_pred = r_forest_model.predict(X_test)
cr = classification_report(y_test, Y_pred)
print (cr)
```

```
[ ]: feats=r[4]
feature_names = [feats[i] for i in r_forest_model.feature_importances_]

feature_importances = pd.DataFrame(r_forest_model.feature_importances_,
                                   index = feature_names,
                                   columns=['importance']).
    ↪sort_values('importance', ascending=False)
```

```
[ ]:
```

```
[ ]: SGDc_model = SGDClassifier(loss='modified_huber', penalty='l2',\
                               alpha=1e-3, random_state=42,\
                               max_iter=100, tol=None, shuffle=True)
    #loss="modified_huber", penalty="l2",max_iter=100, shuffle=True)
SGDc_model.fit(X_train, y_train)
# save model
with open('SGDc_model.pickle', 'wb') as picklefile:
    pickle.dump(SGDc_model, picklefile)

print ("SVM Scoring on test set")
Y_pred = SGDc_model.predict(X_test)
cr = classification_report(y_test, Y_pred)
print (cr)
```

```
[ ]: KNNc_model = KNeighborsClassifier(n_neighbors=5)
KNNc_model.fit(X_train, y_train)
# save model
with open('KNNc_model.pickle', 'wb') as picklefile:
    pickle.dump(KNNc_model, picklefile)

print ("KNN Scoring on test set")
Y_pred = KNNc_model.predict(X_test)
cr = classification_report(y_test, Y_pred)
```

```
print (cr)
```

```
[ ]: dummy_model = DummyClassifier(strategy='constant', constant=0)
dummy_model.fit(X_train, y_train)
```

4 plot roc curves

```
[ ]: def roc_plot(X_test, y_test):

    def get_roc(model, X_test, y_test):
        y_pred = model.predict(X_test)
        y_pred_proba = model.predict_proba(X_test)
        score = round(model.score(X_test, y_test), 2)
        fpr, tpr, _ = roc_curve(y_test.ravel(), y_pred_proba[:,1])
        roc_auc = auc(fpr, tpr)
        return fpr, tpr, roc_auc, score

    sns.set_style('white')
    sns.set_context("talk")
    fig0 = plt.figure(figsize=(15,8), dpi=100);
    plt.plot([0, 1], [0, 1], lw=2, color = 'black' , linestyle='--')

    fpr1, tpr1, roc_auc1, score1 = get_roc(NBmodel, X_test, y_test)
    plt.plot(fpr1, tpr1, lw=2, color = 'brown', label='Multinomial NB area=%0.
    ↳2f,accuracy={}'.format(score1) % roc_auc1)

    fpr2, tpr2, roc_auc2, score2 = get_roc(LRmodel, X_test, y_test)
    plt.plot(fpr2, tpr2, lw=2, color = 'darkviolet', label='Log Reg area=%0.
    ↳2f,accuracy={}'.format(score2) % roc_auc2)

    fpr5, tpr5, roc_auc5, score5 = get_roc(KNNc_model, X_test, y_test)
    plt.plot(fpr5, tpr5, lw=2, color = 'darkgray', label='KNN area=%0.
    ↳2f,accuracy={}'.format(score5) % roc_auc5)

    fpr3, tpr3, roc_auc3, score3 = get_roc(SGDc_model, X_test, y_test)
    plt.plot(fpr3, tpr3, lw=2, color = 'green', label='SVM area=%0.
    ↳2f,accuracy={}'.format(score3) % roc_auc3)

    fpr4, tpr4, roc_auc4, score4 = get_roc(r_forest_model, X_test, y_test)
    plt.plot(fpr4, tpr4, lw=2, color = 'royalblue', label='Random Forest area=%0.
    ↳2f,accuracy={}'.format(score4) % roc_auc4)

    fpr6, tpr6, roc_auc6, score6 = get_roc(dummy_model, X_test, y_test)
    plt.plot(fpr6, tpr6, lw=2, color = 'black', label='Chance area=%0.
    ↳2f,accuracy={}'.format(score6) % roc_auc6)
```

```

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
#plt.title('ROC {}'.format(model), fontsize=12)
#plt.title('ROC All Models', fontsize=12)
plt.legend(loc="lower right")
plt.savefig(path + "roc_plot.svg")
plt.show()

```

```

[ ]: roc_plot(X_test, y_test1, NBmodel, LRmodel, KNNc_model, SGDC_model,
↳r_forest_model, dummy_model)

```

5 Top 1000+ features (coefs)

- get coefs from the logistic regression model

```

[ ]: t=time.strftime("%Y%m%d%H%M",time.localtime())
#coef = model.coef_.copy()
features = zip(r[1],LRmodel.coef_[0])
top = pd.DataFrame(features)#.sort_values(by=1, ascending=False).head(1500).
↳sort_values(0)
#top.to_html(path + 'top_logit_1500_'+t+'.html')

```

```

[ ]: def get_most_important_features(vectorizer, model, n=5):
    index_to_word = r[1]#{v:k for k,v in r[1]}

    # loop for each class
    classes={}
    for class_index in range(model.coef_.shape[0]):
        word_importances = [(el, index_to_word[i]) for i,el in enumerate(model.
↳coef_[class_index])]
        sorted_coeff = sorted(word_importances, key = lambda x : x[0],
↳reverse=True)
        tops = sorted(sorted_coeff[:n], key = lambda x : x[0])
        bottom = sorted_coeff[-n:]
        classes[class_index] = {
            'tops':tops,
            'bottom':bottom
        }
    return classes

importance = get_most_important_features(r[0], LRmodel, 5000)

```

```
[ ]: def plot_important_words(top_scores, top_words, bottom_scores, bottom_words,
    ↪name):
    y_pos = np.arange(len(top_words))
    top_pairs = [(a,b) for a,b in zip(top_words, top_scores)]
    top_pairs = sorted(top_pairs, key=lambda x: x[1])

    bottom_pairs = [(a,b) for a,b in zip(bottom_words, bottom_scores)]
    bottom_pairs = sorted(bottom_pairs, key=lambda x: x[1], reverse=True)

    top_words = [a[0] for a in top_pairs]
    top_scores = [a[1] for a in top_pairs]

    bottom_words = [a[0] for a in bottom_pairs]
    bottom_scores = [a[1] for a in bottom_pairs]

    fig = plt.figure(figsize=(10, 20))

    plt.subplot(121)
    plt.barh(y_pos, bottom_scores, alpha=0.5)
    plt.title('Non-Transfused', fontsize=20)
    plt.yticks(y_pos, bottom_words, fontsize=14)
    plt.suptitle('Key words', fontsize=16)
    plt.xlabel('Importance', fontsize=20)

    plt.subplot(122)
    plt.barh(y_pos, top_scores, alpha=0.5)
    plt.title('Transfused', fontsize=20)
    plt.yticks(y_pos, top_words, fontsize=14)
    plt.suptitle(name, fontsize=16)
    plt.xlabel('Importance', fontsize=20)

    plt.subplots_adjust(wspace=0.8)
    plt.savefig(path + path1 + "top_bottom_plot.svg")
    plt.show()

    top_scores_p = [a[0] for a in importance[0]['tops']][4955:-1]
    top_words_p = [a[1] for a in importance[0]['tops']][4955:-1]
    bottom_scores_p = [a[0] for a in importance[0]['bottom']][4955:-1]
    bottom_words_p = [a[1] for a in importance[0]['bottom']][4955:-1]

    plot_important_words(top_scores_p, top_words_p, bottom_scores_p,
    ↪bottom_words_p, "Most important words for relevance")
```

```
[ ]: bottom_coef = pd.
    ↪DataFrame(columns=['vocab', 'coef'])#, data=[top_words, top_scores])
    bottom_coef['vocab'] = [a[1] for a in importance[0]['bottom']]
    bottom_coef['coef'] = [a[0] for a in importance[0]['bottom']]
```



```
bottom_coef.sort_values('coef',ascending=True).head(200)
```

```
[ ]: top_scores = [a[0] for a in importance[0]['tops']]
top_words = [a[1] for a in importance[0]['tops']]

top_coef = pd.DataFrame(columns=['vocab','coef'])#,data=[top_words,top_scores])
top_coef['vocab'] = top_words
top_coef['coef'] = top_scores
```

```
[ ]: print(top_coef.shape)
top_coef.sort_values('coef').tail(200)
```

```
[ ]: top_coef = top_coef.sort_values(by='coef',axis=0,ascending=False)
top_coef.to_csv(path + path1 + 'top_logit_coef_5000.csv')

print ('create matrix for clustering')
top_idx=[r[1].index(k) for k in top_coef['vocab'].values]

r0= r[0].sorted_indices()

tmat = r0.T[top_idx]

print ('done making matrix')

#below is input for 'Remove transfusion terms, Collapse dupes into largest_
↳n-gram.ipynb'

out=[tmat,top_coef]
with open('logits_top_5000_matrix.pickle','wb') as f:
    pickle.dump(out,f)
```

5.1 TOP 5000 features according to log probability

- for the naive Bayes model

```
[ ]: t=time.strftime("%Y%m%d%H%M",time.localtime())

prob = pd.DataFrame(NBmodel.feature_log_prob_)
prob=prob.transpose()
prob.columns=NBmodel.classes_
prob['vocab']=pd.Series(r[1])
prob['ratio']=prob['transfusion']/prob['control']
prob= prob.drop(['transfusion', 'control'], axis=1)

top_all = prob.sort_values(by='ratio')
top = prob.sort_values(by='ratio').head(5000)
print(top_all.shape)
```

```
top_all.head()

top1k = prob.sort_values(by='ratio').head(1000)
```

```
[ ]: # save terms and ratios and model

top_all.to_pickle('NB_terms_ratio_all.pkl')

with open( 'NB_model.pickle','wb') as f:
    pickle.dump(NBmodel,f)
top.to_csv('top_5000_feat_'+t+'.csv', columns=['vocab','ratio'], index=False)
top.to_pickle('top_5000_feat.pkl')

[ ]: print (prob.sort_values(by='ratio').head(500))
```

6 create matrix for modeling

- saves results into a sparse matrix for unsupervised models (LDA, etc)

```
[ ]: # top 5k
top_idx=[r[1].index(k) for k in top['vocab'].values]
r0= r[0].sorted_indices()
tmat = r0.T[top_idx]
tmat.size

[ ]: #below is input for 'Remove transfusion terms, Collapse dupes into largest_
↳n-gram.ipynb'
out=[tmat,top]
with open('NB_5000_matrix.pickle','wb') as f:
    pickle.dump(out,f)
```