DSP3_Classification_final_dpmin4

August 26, 2020

[1]:
```
import pandas as pd
import numpy as np
import sklearn
from sklearn.linear_model import LogisticRegressionCV
import seaborn as sns
%matplotlib inline
import matplotlib.pyplot as plt
```

[2]:
```
data_links_biarcs0 = ['https://dspass3ngrams.s3.amazonaws.com/csv/output_dpmin4.csv',
  'https://dspass3ngrams.s3.amazonaws.com/csv/output_dpmin5.csv',
  'https://dspass3ngrams.s3.amazonaws.com/csv/output_dpmin6.csv']

data_links_biarcs1 = ['https://dspass3ngrams.s3.amazonaws.com/csv/output2_dpmin6.csv.
  ↪zip',
                     'https://dspass3ngrams.s3.amazonaws.com/csv/output2_dpmin5.csv.zip',
                     'https://dspass3ngrams.s3.amazonaws.com/csv/output2_dpmin4.csv.zip']

data_links = data_links_biarcs1
```

[3]:
```
# !wget https://dspass3ngrams.s3.amazonaws.com/csv/output2_dpmin6.csv.zip
# !wget https://dspass3ngrams.s3.amazonaws.com/csv/output2_dpmin5.csv.zip
!wget https://dspass3ngrams.s3.amazonaws.com/csv/output2_dpmin4.csv.zip

#TODO: generify this
```

```
--2020-08-21 14:30:12--
https://dspass3ngrams.s3.amazonaws.com/csv/output2_dpmin4.csv.zip
Resolving dspass3ngrams.s3.amazonaws.com (dspass3ngrams.s3.amazonaws.com)…
52.216.133.147
Connecting to dspass3ngrams.s3.amazonaws.com
(dspass3ngrams.s3.amazonaws.com)|52.216.133.147|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 5705401 (5.4M) [application/zip]
Saving to: output2_dpmin4.csv.zip

output2_dpmin4.csv. 100%[===================>]   5.44M  3.58MB/s    in 1.5s
```

```
2020-08-21 14:30:14 (3.58 MB/s) - output2_dpmin4.csv.zip saved
[5705401/5705401]
```

[4]:
```
!unzip \*.zip

!ls
```

```
Archive:  output2_dpmin4.csv.zip
  inflating: output2_dpmin4.csv
DSP3_Classification.ipynb       output2_dpmin4.csv
DSP3_Classification_final.ipynb output2_dpmin4.csv.zip
```

[5]:
```
# _df = pd.read_csv ('output2_dpmin{0}.csv'.format(4))
# print(df.head())
```

[6]:
```
# words[4]['Word1']
```

We run this with 3 *DPMIN* Values: 4, 5, 6

[7]:
```
checks = [4]
```

[8]:
```
X = {}
Y = {}
words = {}
embd_X = {}

for i in checks:

  df = pd.read_csv ('output2_dpmin{0}.csv'.format(i))
  feature_len = len(df.columns)-3 #3 because we don't care about: Word1, Word2 and True␣
  ↪Label

  X_df = df[['Feature_'+str(j)  for j in range (feature_len)]]
  Y_df = df[['True Label']]
  words[i] = df[['Word1','Word2']]

  X[i] = np.squeeze(X_df.to_numpy())
  Y[i] = np.squeeze(Y_df.to_numpy())
```

0.1 Machine Learning

0.1.1 Preprocessing

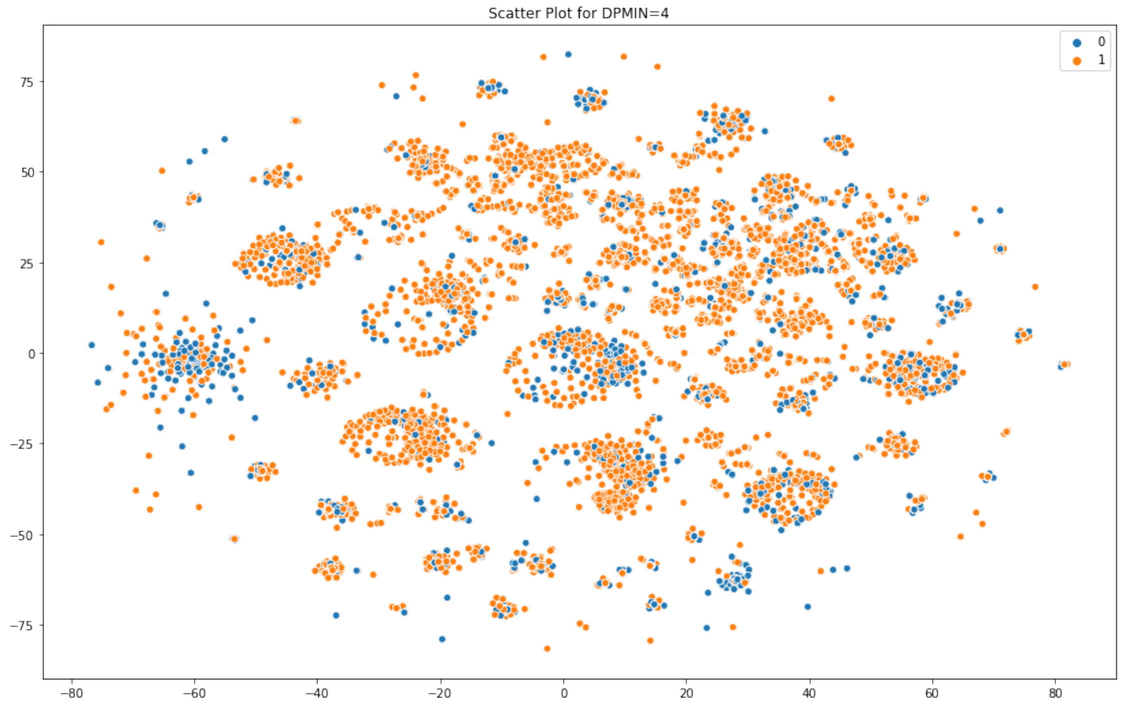
[9]:
```
from sklearn.decomposition import PCA

K = 100 # the new dimesnion size - we reducing to

for i in checks:
  pca = PCA(n_components = K)
  embd_X[i] = pca.fit_transform(X[i])
  total_var = pca.explained_variance_ratio_.cumsum()[-1]
  print('Total variance retrained with this PCA: {0}%'.format(total_var*100))
```

```
Total variance retrained with this PCA: 70.7462314695531%
```

Then, we will visualize the datapoints

[10]:

```
from sklearn.manifold import TSNE

for i in checks:

  np.random.seed(33)
  tsne = TSNE(n_components=2).fit_transform(embd_X[i]) #TSNE is good for visualising scatter plots
  plt.figure(figsize=(16,10))
  sns.scatterplot(tsne[:,0],tsne[:,1],hue=Y[i])
  plt.title('Scatter Plot for DPMIN={}'.format(i))
```



Doesn't seem like it can be easily seperated, at least with no kernel.

0.1.2 Training

We will train with logistic regression and with $CV = 10$

[11]:

```
clf_logic = {}
for i in checks:
    tmp_clf = LogisticRegressionCV(cv = 10,class_weight='balanced',max_iter=10_000).
    ↪fit(embd_X[i],Y[i])
    print('i-th score: ', tmp_clf.score(embd_X[i],Y[i]))
    clf_logic[i] = tmp_clf
```

```
i-th score:  0.6457220504170181
```

0.1.3 Analysis

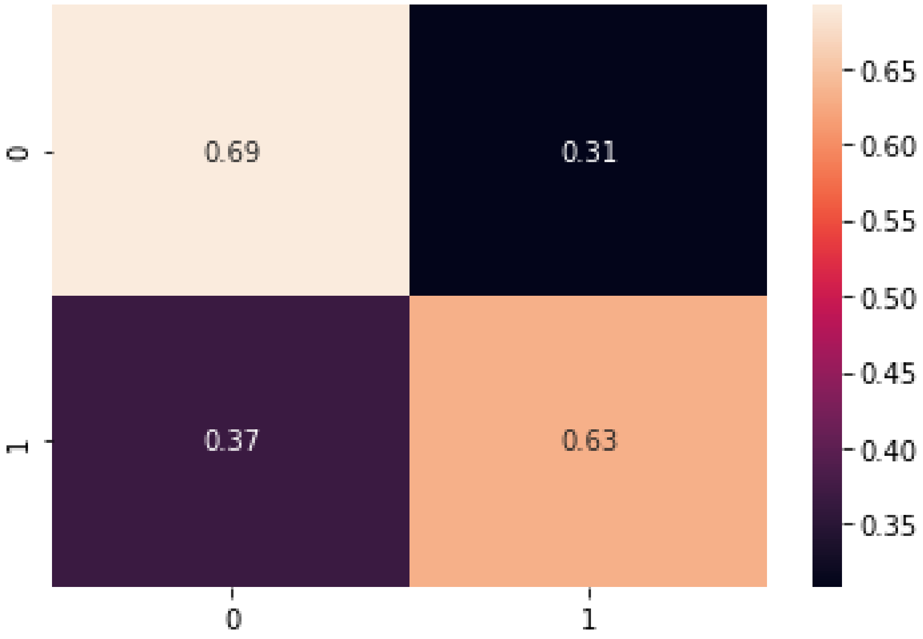
[12]:

```
from sklearn.metrics import precision_recall_fscore_support, confusion_matrix, roc_auc_score

def analyse_clf (clf,i):

    y_pred = np.round(clf.predict(embd_X[i]))
    y_true = Y[i]

    prf =  list(precision_recall_fscore_support( y_true, y_pred,average='binary'))[:-1]
    prf.append(roc_auc_score(y_true,y_pred))
    conf_mat = confusion_matrix(y_true, y_pred,normalize='true') #also normlize according␣
    ↪to the truth


    print('Precision: {0}\nRecall: {1}\nF1_Score:{2}\n ROC: {3}'.format(*prf))
    print('------------------------------------')
    plt.figure()
    sns.heatmap(conf_mat,annot=True)
    # row is true, column is predicted
```

[13]:

```
for i in checks:
    analyse_clf(clf_logic[i],i)
```

```
Precision: 0.8675787517946l2
Recall: 0.6315627689659412
F1_Score:0.7309922795033266
 ROC: 0.66132905981788
------------------------------------
```

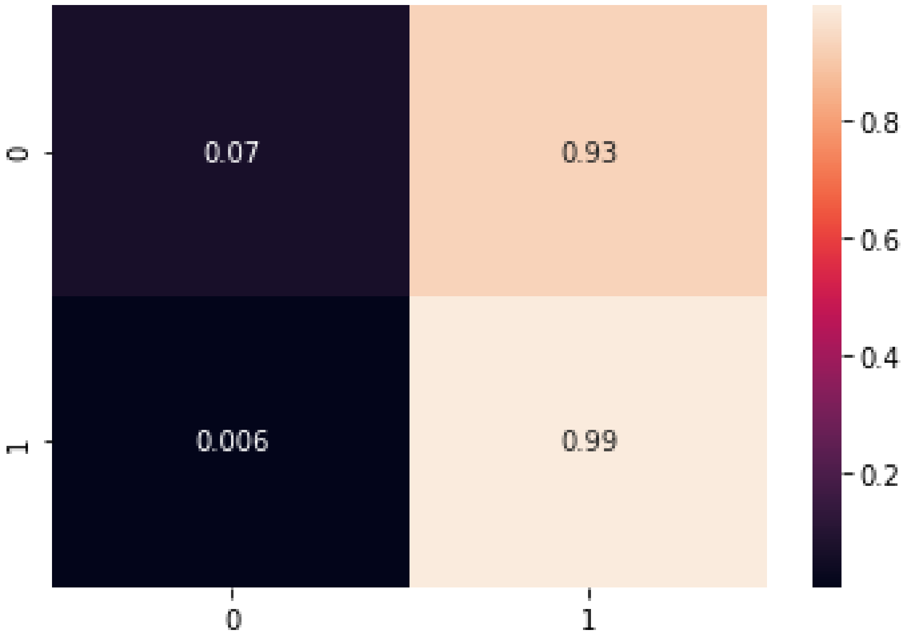


Now let us try *SVC*

[14]:
```
scoring = ['precision', 'recall','accuracy','f1','roc_auc']
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

from sklearn.model_selection import cross_validate
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score

from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

best_models = {}

for i in checks:

  tmp_clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
  # tmp_clf.fit(embd_X[i],Y[i])

  scores = cross_validate(tmp_clf, embd_X[i], Y[i],␣
↪scoring=scoring,return_estimator=True,cv=10)
  print("**********{}**********".format(i))
```

```
    # print(scores)

    best_model_ndx = np.argmax(scores['test_roc_auc'])
    print
    best_models[i] = scores['estimator'][best_model_ndx]

    analyse_clf(best_models[i],i)
```

```
***********4**********
Precision: 0.7739326057821175
Recall: 0.9940366408459363
F1_Score:0.8702836535873837
 ROC: 0.531789794024229
-------------------------------------
```



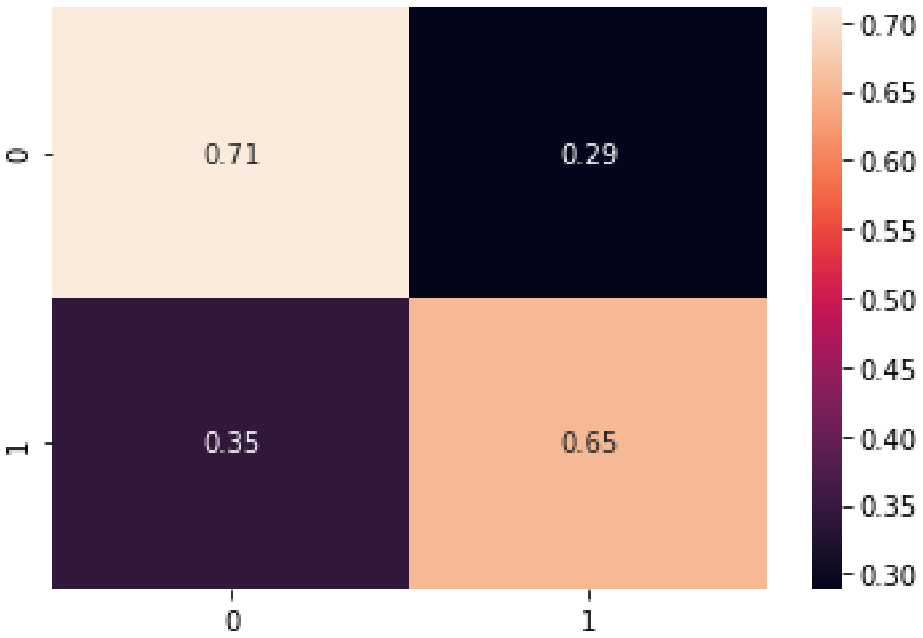
Let us try with *balanced* class weight

[15]:

```
best_models_balanced = {}

for i in checks:

    tmp_clf = make_pipeline(StandardScaler(), SVC(gamma='auto',class_weight='balanced'))
    # tmp_clf.fit(embd_X[i],Y[i])

    scores = cross_validate(tmp_clf, embd_X[i], Y[i],␣
    ↪scoring=scoring,return_estimator=True,cv=10)
    print("***********{}**********".format(i))
```

```
  # print(scores)

  best_model_ndx = np.argmax(scores['test_roc_auc'])
  print
  best_models_balanced[i] = scores['estimator'][best_model_ndx]

  analyse_clf(best_models_balanced[i],i)

best_clf = best_models_balanced[4]
```

```
**********4**********
Precision: 0.8787528868360277
Recall: 0.6549858600762326
F1_Score:0.75054596688975
 ROC: 0.6826938756645938
-------------------------------------
```



[16]:

```
from collections import Counter

for i in checks:
  print("**********{}**********".format(i))

  counted_vals = Counter(Y[i])
  zero_label_percent = counted_vals[0] / (counted_vals[0] + counted_vals[1])
  one_label_percent = counted_vals[1] / (counted_vals[0] + counted_vals[1])
  print("No-hypernyms",zero_label_percent)
print("hypernyms",one_label_percent)
```

```
***********4**********
No-hypernyms 0.2378408771436604
hypernyms 0.7621591228563396
```

[17]:
```
# We supposed to play with the examples now
embd_X
```

[17]:
```
{4: array([[-0.10029389, -0.13539825, -0.384451  , ...,  0.00271945,
          0.0013594 ,  0.00175056],
        [-0.19144404,  0.10003238, -0.28997174, ...,  0.00328079,
         -0.00307037,  0.00269231],
        [-0.29098437, -0.05802356,  0.19640241, ...,  0.04034176,
          0.04514644, -0.02322879],
        ...,
        [-0.13210339, -0.15700779, -0.32373864, ...,  0.03877869,
          0.01598212,  0.01070661],
        [-0.40883516, -0.59908823,  0.2890192 , ...,  0.00262066,
          0.0015617 , -0.00371533],
        [-0.36605547,  0.68059671, -0.0910479 , ...,  0.00324655,
          0.00232376, -0.00320101]])}
```

[18]:
```
# extract Bad and Good Examples:

TP, FP, TN, FN = [], [], [], []
got_enough_examples = False
predictions = best_clf.predict(embd_X[4])
counter = 0

for w1,w2,pred,true in zip(words[4]['Word1'],words[4]['Word2'],predictions,Y[4]):
  if true == 0:
    if pred == 0:
      TN.append((w1,w2))
    if pred == 1:
      FP.append((w1,w2))
  if true == 1:
    if pred == 0:
      FN.append((w1,w2))
    if pred == 1:
      TP.append((w1,w2))
```

[18]:
```

```