EE660_HW7_Q1_Hardik_2678294168

November 27, 2021

0.0.1 EE660_HW7_Q1: Hardik Prajapati (2678294168)

[15]:
```
import numpy as np
import pandas as pd
import os
from sklearn.ensemble import AdaBoostClassifier
from sklearn.mixture import GaussianMixture
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt
from sklearn.mixture import GaussianMixture
from scipy.stats import multivariate_normal
```

[2]:
```
#reading files
def read_files(file_name,label_flag):
    root="..\\data\\TL\\"
    filepath=os.path.join(root,file_name)
    df=pd.read_csv(filepath)
    if label_flag==True:
        Xdata=df.iloc[:,:2]
        Ydata=df.iloc[:,2]
        return Xdata,Ydata
    else:
        Xdata=df.iloc[:,:]
        return Xdata
```

[3]:
```
#grab the source training, test data
Xtrain_source,Ytrain_source=read_files("data_source_train.csv",True)
Xtest_source,Ytest_source=read_files("data_source_test.csv",True)

#grab the target labeled, unlabeled & test data
Xlab_target,Ylab_target=read_files("data_target_labeled.csv",True)
Xunlab_target=read_files("data_target_unlabeled.csv",False)
Xtest_target,Ytest_target=read_files("data_target_test.csv",True)
```

[4]:
```
def train_clf(X_train,Y_train,X_test,Y_test):
    clf=AdaBoostClassifier()
    clf=clf.fit(X_train,Y_train)
    Y_pred=clf.predict(X_test)
    score=accuracy_score(Y_test,Y_pred)
    print("Accuracy score: {}".format(score))
    return score,clf
```

[5]:
```
def merge_df(df1,df2):
    df3=pd.concat([df1,df2],axis=0)
    return df3
```

[6]:
```
#Q1.a: Supervised Learning: Source data
print("---xxx---Q1.a: Supervised Learning: Source data---xxx---xxx--")
score_a,clf_a=train_clf(Xtrain_source,Ytrain_source,Xtest_source,Ytest_source)
```

```
---xxx---Q1.a: Supervised Learning: Source data---xxx---xxx--
Accuracy score: 0.86
```

[7]:
```
#Q1.b: report accuracy score of model trained on Source data for target test␣
 ↪data
print("---xxx--Q1.b: TL: Train model on Source data and test on target test␣
 ↪data---xxx---")
Y_pred=clf_a.predict(Xtest_target)
score_b=accuracy_score(Ytest_target,Y_pred)
print("Accuracy score: {}".format(score_b))
```

```
---xxx--Q1.b: TL: Train model on Source data and test on target test data---
xxx---
Accuracy score: 0.78
```

[8]:
```
#Q1.c: Supervised Learning: Target labeled data
print("---xxx---Q1.c: Supervised Learning: Target Labeled data---xxx---")
score_c,clf_c=train_clf(Xlab_target,Ylab_target,Xtest_target,Ytest_target)
```

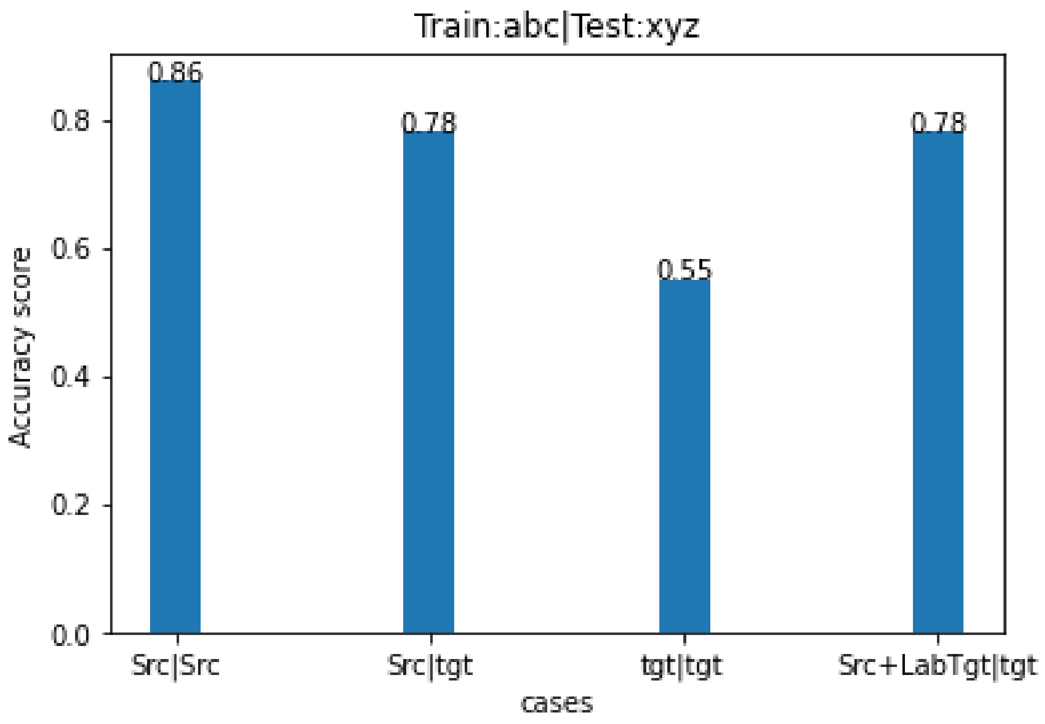
```
---xxx---Q1.c: Supervised Learning: Target Labeled data---xxx---
Accuracy score: 0.55
```

[9]:
```
#Q1.d: Concatenate source trainng + target labeled data. Train model on this␣
 ↪and test on target
#test data
print("---xxx---Q1.d: Train: Source train + target Labeled | Test: Target␣
 ↪test---xxx---")
Xtrain_source_labTarget=merge_df(Xtrain_source,Xlab_target)
Ytrain_source_labTarget=merge_df(Ytrain_source,Ylab_target)
score_d,clf_d=train_clf(Xtrain_source_labTarget,Ytrain_source_labTarget,Xtest_target,Ytest_tar
```

```
---xxx---Q1.d: Train: Source train + target Labeled | Test: Target test---xxx---
Accuracy score: 0.78
```

[10]:
```
#Q1.e: Plot bar chart to compare acc score of Q1.a-d
def addlabels(x,y):
    for i in range(len(x)):
        plt.text(i, y[i], y[i], ha = 'center')

x_scale=['Src|Src','Src|tgt','tgt|tgt','Src+LabTgt|tgt']
y_scale=[score_a,score_b,score_c,score_d]
plt.bar(x_scale, y_scale, width=0.2, align='center')
plt.xlabel("cases")
plt.ylabel("Accuracy score")
addlabels(x_scale,y_scale)
plt.title("Train:abc|Test:xyz")
plt.show()
```



We achieved the best accuracy score in case_a, where model was trained on Source data and tested on source test data. This can be argued because of rich data (train) and that test data and train data come from the same distribution.

It perfomed worst when model was trained on target labeled data and tested on target test data. This can be due to having only 2 samples in training data.

Model that was trained on source and tested on target data (case b) performed fairly good maybe because the target domain and source domain share some kind of (known or learnable) relationship. This can also be extended to case d, where additional

training is perfomed on labeled target data. Although it doesn't make much difference because we have only 2 samples in labeled target data.

[11]:
```
def density_estimator(x_data,components):
    gm = GaussianMixture(n_components=components).fit(x_data)
    mean_vector=gm.means_
    covariance_matrix=gm.covariances_
    for i in range(components):
        print("\nMean Vector {} :".format(i+1))
        print("\n",mean_vector[i])
        print("\nCovariance Matrix {} :".format(i+1))
        print("\n",covariance_matrix[i])
    return gm, mean_vector, covariance_matrix
```

[12]:
```
#Q1.f.i: pass the training data (source+lab_target+unlabeled target) through␣
↪density estimator
#and compute the mean vector and Covariance matrix for each component(domain)
X_srcTrain_labTgt_unlabTgt=pd.concat([Xtrain_source,Xlab_target,Xunlab_target])
est1,mean1,covariance1=density_estimator(X_srcTrain_labTgt_unlabTgt,2)
```

```
Mean Vector 1 :

 [1.12453237 0.99180681]

Covariance Matrix 1 :

 [[3.84923391 0.15283634]
 [0.15283634 0.75527211]]

Mean Vector 2 :

 [-1.01280831 -0.33910676]

Covariance Matrix 2 :

 [[ 0.801921    0.36329384]
 [ 0.36329384 19.06863624]]
```

[13]:
```
#Q1.f.ii: pass the training data for source domain through density
#estimator and compute the mean vector and Covariance matrix domain.
est2,mean_src,covariance_src=density_estimator(Xtrain_source,1)
```

```
Mean Vector 1 :

 [1.14916266 1.09731636]

Covariance Matrix 1 :
 [[ 3.70506986 -0.08005567]
 [-0.08005567  0.65845626]]
```

[14]:
```
#Q1.f.ii: pass the training data for target domain(labeled+unlabeled) through␣
 ↪density
#estimator and compute the mean vector and Covariance matrix domain.
X_labTgt_unlabTgt=pd.concat([Xlab_target,Xunlab_target])
est3,mean_tgt,covariance_tgt=density_estimator(X_labTgt_unlabTgt,1)
```

```
Mean Vector 1 :

 [-1.05830525 -0.45760988]

Covariance Matrix 1 :

 [[ 0.76377201  0.30418792]
 [ 0.30418792 19.02092692]]
```

For my results, both the above methods yield similar results for mean vector and covariance matrix for both domain. The only concern being, in the first method (simultaneous computation) it's difficult to identify which parameters constitute to source and target domain respectively.

[71]:
```
def pdf_clc(xdata,mean_vectors,covariance_matrix):
    y = multivariate_normal.pdf(xdata, mean=mean_vectors, cov=covariance_matrix)
    return y
```

[72]:
```
#compute pdf of training data from target and source using the parameters␣
 ↪computed above
#and then divide them to calulate weights
def␣
 ↪weights_pdf(train_data,mean_target,covariance_target,mean_source,covariance_source):
 ↪
    p_t=pdf_clc(train_data,mean_target,covariance_target)
    p_s=pdf_clc(train_data,mean_source,covariance_source)
    weights=np.divide(p_t, p_s)
    return weights
```

[73]:
```
#Q1.g: Computing the weights
Xtrain_final=pd.concat([Xtrain_source,Xlab_target])
Ytrain_final=pd.concat([Ytrain_source,Ylab_target])
weights_sample=weights_pdf(Xtrain_final,mean_tgt[0],covariance_tgt[0],mean_src[0],covariance_s
weights_sample
```

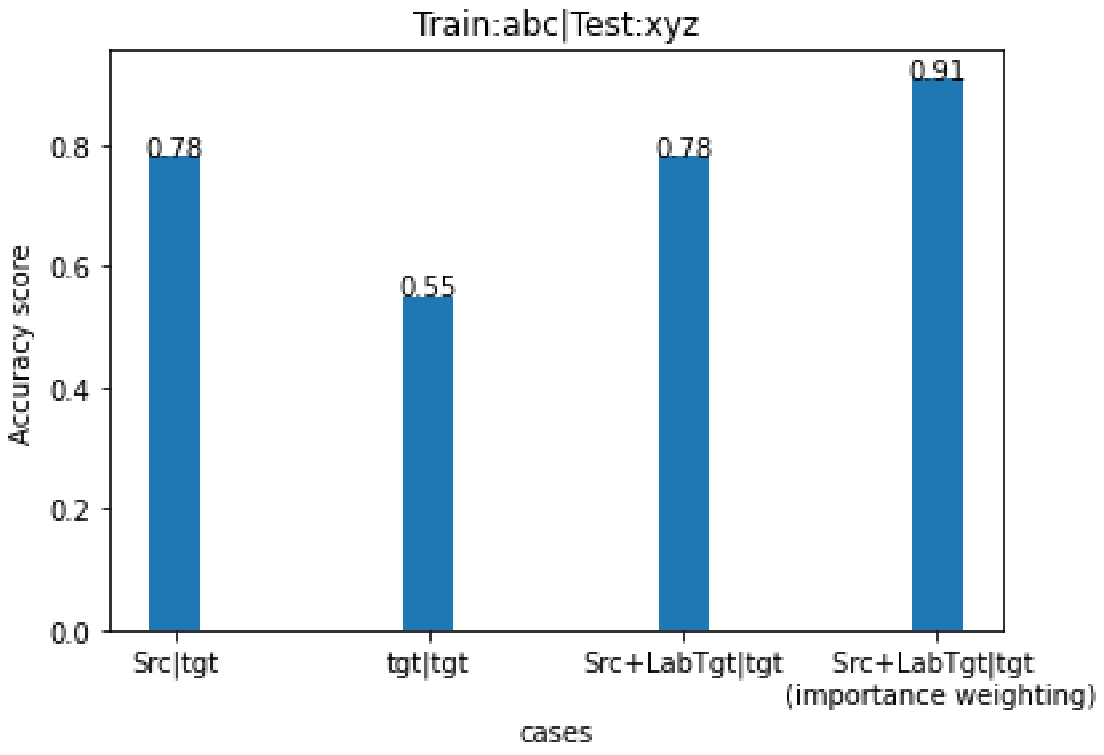
[73]:
```
array([2.55417987e-06, 2.49917893e-03, 2.44482601e-02, 5.72293292e-03,
       5.11235980e-01, 8.92213735e-01, 8.31277160e-08, 1.07347219e-04,
       3.20747937e-04, 8.47464934e-01, 4.55549772e-02, 7.46360586e-06,
       5.93283455e-02, 4.34169244e-01, 2.92829874e-01, 1.32782829e-03,
       1.66799479e-03, 8.78171840e-01, 1.46441571e-06, 1.17951128e-05,
       2.70913055e-02, 7.49497531e-01, 2.98626956e-02, 1.13169880e-03,
       3.41244122e-07, 9.47247290e-02, 4.60045090e-07, 1.14482242e+00,
       8.69914696e-09, 1.03776693e+00, 5.22289658e-13, 1.97664814e-07,
       1.09848313e-01, 2.30051194e-03, 7.45243252e-05, 8.97345510e-01,
       3.11647963e-01, 1.09008064e-03, 7.27340978e-02, 3.56195453e-03,
       6.84248099e-01, 3.52573059e-09, 7.60484792e-01, 2.45114805e-15,
       2.19066045e-03, 9.16201234e-01, 4.51994329e-01, 4.15602894e-02,
       4.15352767e-02, 5.70065449e-01, 2.50501342e-05, 1.57970654e-01,
       8.19792716e-03, 2.41817641e-07, 1.24975565e+00, 7.12243547e-03,
       2.81318954e-04, 8.02150614e-01, 4.14526210e-03, 6.64370958e-01,
       1.35616598e-02, 1.03183569e-03, 3.01846018e-04, 1.89939010e+00,
       3.41926616e-03, 5.40202312e-01, 6.65282585e-03, 6.82644609e-01,
       1.95167926e-07, 6.18798710e-04, 4.24388432e-02, 9.61215895e-04,
       4.70857297e-04, 7.15906914e-03, 2.06874251e-01, 5.56830115e-03,
       5.05800136e-02, 2.57280474e-02, 1.53050463e-01, 4.76228451e-01,
       5.75463462e-04, 1.11707869e+00, 2.16443410e-01, 5.92094122e-01,
       6.74795832e-01, 3.33612584e-01, 3.96602713e-01, 1.24651735e+00,
       1.51555476e-05, 1.16285836e-02, 1.99791080e+00, 2.90020366e-04,
       1.10406992e+00, 8.65693502e-04, 8.29632936e-07, 1.21579320e-03,
       7.09862883e-01, 6.68880959e-03, 7.07758337e-02, 8.10399724e-06,
       1.30540551e+10, 1.24382861e+24])
```

[74]:
```
#Q1.g: training the model with importance weighting and testing on target test␣
↪data.
clf=AdaBoostClassifier()
clf=clf.fit(Xtrain_final,Ytrain_final,sample_weight=weights_sample)
Y_pred=clf.predict(Xtest_target)
score=accuracy_score(Ytest_target,Y_pred)
print("--xxx---Q1.g: Transfer Learning: Train: Source train + target Labeled |␣
↪Test: Target test---xxx--")
print("Accuracy score: {}".format(score))
```

```
--xxx---Q1.g: Transfer Learning: Train: Source train + target Labeled | Test:
Target test---xxx--
Accuracy score: 0.91
```

[77]:
```
#Q1.h: comparing the accuracy scores of cases b-d with g.
def addlabels(x,y):
    for i in range(len(x)):
        plt.text(i, y[i], y[i], ha = 'center')

x_scale=['Src|tgt','tgt|tgt','Src+LabTgt|tgt','Src+LabTgt|tgt \n (importance␣
↪weighting)']
y_scale=[score_b,score_c,score_d,score]
plt.bar(x_scale, y_scale, width=0.2, align='center')
plt.xlabel("cases")
plt.ylabel("Accuracy score")
addlabels(x_scale,y_scale)
plt.title("Train:abc|Test:xyz")
plt.show()
```



It can be clearly observed that Adaboost with importance weighting performed the best achieving a high accuracy score of 91%. On introducing sample weights as additional paramter to case d, amplified the importance of target labeled data compared to source train data. This can be viewed as compensation of drawing samples according to source joint PDF AND not according to joint PDF of target domain.

The comparision of rest of the cases (b,c) follows the same explanation given in section Q1.e