F-elNet

October 3, 2019

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
import pandas as pd
import datetime
from IPython.display import SVG
from sklearn.model_selection import KFold
from sklearn import metrics
from sklearn.linear_model import LogisticRegression

from itertools import cycle
from sklearn.linear_model import lasso_path, enet_path
```

[2]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
```

[3]:
```
SMALL_SIZE = 10
MEDIUM_SIZE = 15
BIGGER_SIZE = 18

# font = {'family' : 'monospace',
#         'weight' : 'bold',
#         'size'   : 'larger'}

#plt.rc('font', **font)  # pass in the font dict as kwargs
plt.rc('font', size=MEDIUM_SIZE,family='normal',weight='normal')          #␣
↪controls default text sizes
plt.rc('axes', titlesize=MEDIUM_SIZE,)     # fontsize of the axes title
```

```
plt.rc('axes', labelsize=MEDIUM_SIZE,)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=MEDIUM_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=MEDIUM_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE,titleweight='bold')  # fontsize of the␣
 ↪figure title
#plt.rc('xtick', labelsize=15)
#plt.rc('ytick', labelsize=15)
```

[4]:
```
np.random.seed(42)
print(str(datetime.datetime.now()))
```

2019-10-02 18:45:23.400627

[5]:
```
NBname='_F-elNet'
N_folds=4
np.random.seed(100)
kf = KFold(n_splits=N_folds, shuffle=False)

rm='res_x.npy'
rl='res_y.npy'
rdata=np.load(os.path.abspath(rm))
rlabels=np.load(os.path.abspath(rl))

sm='sen_x.npy'
sl='sen_y.npy'
sdata=np.load(os.path.abspath(sm))
slabels=np.load(os.path.abspath(sl))

fmtim='testim_x.npy'
fltim='testim_y.npy'
testim=np.load(os.path.abspath(fmtim))
tlabelsim=np.load(os.path.abspath(fltim))

fmtb='testb_x.npy'
fltb='testb_y.npy'
testb=np.load(os.path.abspath(fmtb))
tlabelsb=np.load(os.path.abspath(fltb))

# =================
# Do once!
# =================
sen_batch = np.random.RandomState(seed=45).permutation(sdata.shape[0])
bins = np.linspace(0, 200, 41)
digitized = np.digitize(sen_batch, bins,right=False)
# ================
```

```
# ================================
# # FINAL TRAIN
# ================================
train_idx_k=np.random.permutation(rdata.shape[0])
s_x=sdata[np.isin(digitized,train_idx_k+1)]
s_y=slabels[np.isin(digitized,train_idx_k+1)]
r_x=np.
 ↪concatenate((rdata[train_idx_k],rdata[train_idx_k],rdata[train_idx_k],rdata[train_idx_k],rd
r_y=np.
 ↪concatenate((rlabels[train_idx_k],rlabels[train_idx_k],rlabels[train_idx_k],rlabels[train_i

f_train_x, f_train_y = np.concatenate((s_x,r_x)), np.concatenate((s_y,r_y))
train_shuf_idx = np.random.permutation(f_train_x.shape[0])
x_train, y_train = f_train_x[train_shuf_idx], f_train_y[train_shuf_idx]

# x_better_test=x_test.reshape(x_test.shape[0],x_test.shape[1])
# y_better_test=y_test.reshape(y_test.shape[0],y_test.shape[1])
# y_better_test=y_better_test[:,1]

x_better_train=x_train.reshape(x_train.shape[0],x_train.shape[1])
y_better_train=y_train.reshape(y_train.shape[0],y_train.shape[1])
y_better_train=y_better_train[:,1]

xb_better_test=testb.reshape(testb.shape[0],testb.shape[1])
yb_better_test=tlabelsb.reshape(tlabelsb.shape[0],tlabelsb.shape[1])
yb_better_test=yb_better_test[:,1]

xim_better_test=testim.reshape(testim.shape[0],testim.shape[1])
yim_better_test=tlabelsim.reshape(tlabelsim.shape[0],tlabelsim.shape[1])
yim_better_test=yim_better_test[:,1]

# l1rat=0.4
# C=0.61

l1rat=0.6
C=0.81

LR = LogisticRegression(C=C, tol=0.01, penalty='elasticnet', solver='saga',␣
 ↪n_jobs=-1, l1_ratio=l1rat)
LR.fit(x_better_train,y_better_train)

# y_pred = regE.predict(xb_better_test)

# mname1='f1'+NBname+'.h5'
# mname2='f2'+NBname+'.h5'
# regE1.save(mname1)
# regE2.save(mname2)
```

```
# # ======================
# # ONLY FOR CROSS-VAL
# # ======================

# i=0
# # logistic=[]

# l1rat=0.5

# acc1=[]
# acc2=[]
# acc3=[]

# # callbacks = [EarlyStopping(monitor='val_loss', patience=10),
# #              ModelCheckpoint(filepath='best_model'+NBname+'.h5',␣
 ↪monitor='val_loss', save_best_only=True)]



# for train_idx_k, val_idx_k in kf.split(rdata):
#     print ("Running Fold", i+1, "/", N_folds)
#     f= open('perform_'+str(i+1)+NBname+'.txt', "a")
#     # ===============================
#     # select train
#     # ===============================

#     s_train_x=sdata[np.isin(digitized,train_idx_k+1)]
#     s_train_y=slabels[np.isin(digitized,train_idx_k+1)]
#     r_train_x=np.
 ↪concatenate((rdata[train_idx_k],rdata[train_idx_k],rdata[train_idx_k],rdata[train_idx_k],rd
#     r_train_y=np.
 ↪concatenate((rlabels[train_idx_k],rlabels[train_idx_k],rlabels[train_idx_k],rlabels[train_i

#     # ===============================
#     # select val
#     # ===============================

#     s_val_x=sdata[np.isin(digitized,val_idx_k+1)]
#     s_val_y=slabels[np.isin(digitized,val_idx_k+1)]

#     r_val_x=np.
 ↪concatenate((rdata[val_idx_k],rdata[val_idx_k],rdata[val_idx_k],rdata[val_idx_k],rdata[val_
#     r_val_y=np.
 ↪concatenate((rlabels[val_idx_k],rlabels[val_idx_k],rlabels[val_idx_k],rlabels[val_idx_k],rl

#     # ===============================
```

```
#     # concatenate F_train/val_x/y
#     # ================================

#     f_train_x, f_train_y = np.concatenate((s_train_x,r_train_x)), np.
↪concatenate((s_train_y,r_train_y))
# #     train_shuf_idx = np.random.permutation(f_train_x.shape[0])
# #     F_train_x, F_train_y = f_train_x[train_shuf_idx],␣
↪f_train_y[train_shuf_idx]

#     f_val_x, f_val_y = np.concatenate((s_val_x,r_val_x)), np.
↪concatenate((s_val_y,r_val_y))
# #     val_shuf_idx = np.random.permutation(f_val_x.shape[0])
# #     F_val_x, F_val_y = f_val_x[val_shuf_idx], f_val_y[val_shuf_idx]

#     # ================================
#     # shuffle just because we can?
#     # ================================
#     train_shuf_idx = np.random.permutation(f_train_x.shape[0])
#     x_train_CV, y_train_CV = f_train_x[train_shuf_idx],␣
↪f_train_y[train_shuf_idx]

#     val_shuf_idx = np.random.permutation(f_val_x.shape[0])
#     x_val_CV, y_val_CV = f_val_x[val_shuf_idx], f_val_y[val_shuf_idx]

#     x_better_val=x_val_CV.reshape(x_val_CV.shape[0],x_val_CV.shape[1])
#     y_better_val=y_val_CV.reshape(y_val_CV.shape[0],y_val_CV.shape[1])
#     y_better_val=y_better_val[:,1]

#     x_better_train=x_train_CV.reshape(x_train_CV.shape[0],x_train_CV.shape[1])
#     y_better_train=y_train_CV.reshape(y_train_CV.shape[0],y_train_CV.shape[1])
#     y_better_train=y_better_train[:,1]

#     f.write('start of ' + str(i+1) + ' fold\n')

#     l1rat=0.4
#     C=0.61

#     clf_en_LR = LogisticRegression(C=C, tol=0.01, penalty='elasticnet',␣
↪solver='saga', n_jobs=-1, l1_ratio=l1rat)
#     regE=clf_en_LR.fit(x_better_train,y_better_train)
#     y_pred = regE.predict(x_better_val)
#     acc1.append(metrics.accuracy_score(y_pred,y_better_val))


#     l1rat=0.4
#     C=0.61
```

```
#     clf_en_LR = LogisticRegression(C=C, tol=0.01, penalty='elasticnet',␣
↪solver='saga', n_jobs=-1, l1_ratio=l1rat)
#     regE=clf_en_LR.fit(x_better_train,y_better_train)
#     y_pred = regE.predict(x_better_val)
#     acc2.append(metrics.accuracy_score(y_pred,y_better_val))
# #     for x in range(5):

# #         l1rat=x*0.2
# #         f.write('l1_ratio is set to ' + str(l1rat) + '\n')
# #         for y in range(5):
# #             C=1 +y*1.5
# #             clf_l1_LR = LogisticRegression(C=C, penalty='l1', tol=0.01,␣
↪solver='saga',n_jobs=-1)
# #             clf_l2_LR = LogisticRegression(C=C, penalty='l2', tol=0.01,␣
↪solver='saga',n_jobs=-1)
# #             clf_en_LR = LogisticRegression(C=C, tol=0.01,␣
↪penalty='elasticnet', solver='saga', n_jobs=-1, l1_ratio=l1rat)

# #             reg1=clf_l1_LR.fit(x_better_train, y_better_train)
# #             reg2=clf_l2_LR.fit(x_better_train, y_better_train)
# #             regE=clf_en_LR.fit(x_better_train,y_better_train)

# #             y_pred = reg1.predict(x_better_val)
# #             acc1.append(metrics.accuracy_score(y_pred,y_better_val))
# #             y_pred = reg2.predict(x_better_val)
# #             acc2.append(metrics.accuracy_score(y_pred,y_better_val))
# #             y_pred = regE.predict(x_better_val)
# #             acc3.append(metrics.accuracy_score(y_pred,y_better_val))

# #         f.writelines(map('{}\t{}\t{}\n'.format, acc1, acc2, acc3))
# #         acc1=[]
# #         acc2=[]
# #         acc3=[]

#     i=i+1
#     f.write('done for ' + str(i) + ' fold\n')
#     f.write(str(datetime.datetime.now()))
#     f.close()
```

[5]:
```
LogisticRegression(C=0.81, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=0.6, max_iter=100,
                   multi_class='warn', n_jobs=-1, penalty='elasticnet',
                   random_state=None, solver='saga', tol=0.01, verbose=0,
                   warm_start=False)
```

[ ]:
```
```

[51]:

```
# =============================================================================
# # DO NOT UNCOMMENT UNTIL THE END; DECLARES FUNCTION FOR AN UNBIASED TEST
# =============================================================================

def plot_auc(aucies,fprs,tprs, last):
    #plt.figure(figsize=(13,13))
    plt.figure(figsize=(11,11))
    plt.plot([0, 1], [0, 1], 'k--')
    for i in range(len(aucies)):
        st='model_'+str(i+1)+' '
        if i==0:
            st='Balanced'
        else:
            st='Imbalanced'
        plt.plot(fprs[i], tprs[i], label='{} (AUC= {:.3f})'.
 ↪format(st,aucies[i]),linewidth=1.5)

    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title('ROC curve: LR')
    plt.legend(loc='best')

    figname='ROC'+last+'.png'
    plt.savefig(figname,dpi=500)
```

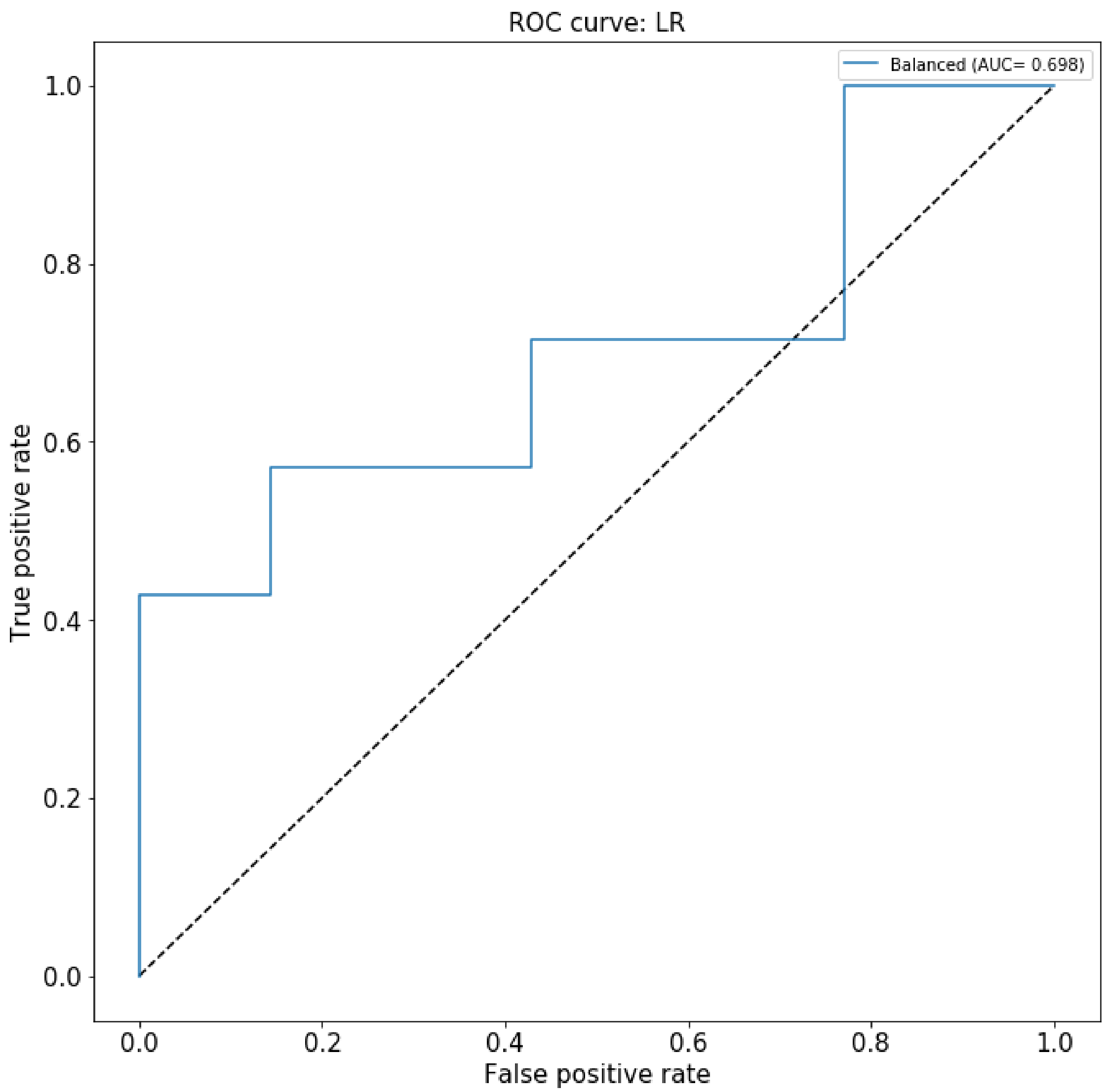
[52]:

```
# =============================================================================
# # THIS IS THE UNBIASED TEST; DO NOT UNCOMMENT UNTIL THE END
# =============================================================================

fpr_x=[]
tpr_x=[]
thresholds_x=[]
auc_x=[]

pre_S=[]
rec_S=[]
f1_S=[]
kap_S=[]
acc_S=[]
mat_S=[]
```

1 BALANCED TESTING

[53]:
```
NBname='_F-elNetb'

# xb_better_test=testb.reshape(testb.shape[0],testb.shape[1])
# yb_better_test=tlabelsb.reshape(tlabelsb.shape[0],tlabelsb.shape[1])
# yb_better_test=yb_better_test[:,1]

y_pred = LR.predict_proba(xb_better_test)
fpr_0, tpr_0, thresholds_0 = roc_curve(yb_better_test, y_pred[:,1])
fpr_x.append(fpr_0)
tpr_x.append(tpr_0)
thresholds_x.append(thresholds_0)
auc_x.append(auc(fpr_0, tpr_0))

testby=yb_better_test

# predict probabilities for testb set
yhat_probs = LR.predict_proba(xb_better_test)
# predict crisp classes for testb set
yhat_classes = LR.predict(xb_better_test)
# # reduce to 1d array

#testby1=tlabels[:,1]
#yhat_probs = yhat_probs[:, 0]
#yhat_classes = yhat_classes[:, 0]

# accuracy: (tp + tn) / (p + n)
acc_S.append(accuracy_score(testby, yhat_classes))
#print('Accuracy: %f' % accuracy_score(testby, yhat_classes))

#precision tp / (tp + fp)
pre_S.append(precision_score(testby, yhat_classes))
#print('Precision: %f' % precision_score(testby, yhat_classes))

#recall: tp / (tp + fn)
rec_S.append(recall_score(testby, yhat_classes))
#print('Recall: %f' % recall_score(testby, yhat_classes))

# f1: 2 tp / (2 tp + fp + fn)
f1_S.append(f1_score(testby, yhat_classes))
#print('F1 score: %f' % f1_score(testby, yhat_classes))

# kappa
kap_S.append(cohen_kappa_score(testby, yhat_classes))
#print('Cohens kappa: %f' % cohen_kappa_score(testby, yhat_classes))
```

```
# confusion matrix
mat_S.append(confusion_matrix(testby, yhat_classes))
#print(confusion_matrix(testby, yhat_classes))

with open('perform'+NBname+'.txt', "w") as f:

    f.writelines("AUC \t Accuracy \t Precision \t Recall \t F1 \t Kappa\n")
    f.writelines(map("{}\t{}\t{}\t{}\t{}\t{}\n".format, auc_x, acc_S, pre_S,␣
↪rec_S, f1_S, kap_S))
    for x in range(len(fpr_x)):
        f.writelines(map("{}\n".format, mat_S[x]))
        f.writelines(map("{}\t{}\t{}\n".format, fpr_x[x], tpr_x[x],␣
↪thresholds_x[x]))

# ============================================================================
# # THIS IS THE UNBIASED testb; DO NOT UNCOMMENT UNTIL THE END
# ============================================================================

plot_auc(auc_x,fpr_x,tpr_x,NBname)
```



1.1 to see which samples were correctly classified …

[54]:
```
yhat_probs[yhat_probs[:,1]>=0.5,1]
```

[54]:
```
array([0.91791241, 0.91791241, 0.57811525, 0.9519404 , 0.61222333,
       0.91791241, 0.61979184, 0.94451295, 0.57811525, 0.57811525,
       0.91791241, 0.77234803, 0.57811525, 0.56077694, 0.68372015,
       0.94451295, 0.9519404 , 0.9519404 , 0.94451295, 0.94451295,
       0.57811525, 0.9519404 , 0.9519404 , 0.94451295, 0.6505654 ,
       0.91791241])
```

[55]:
```
yhat_probs[:,1]>=0.5
```

[55]: array([False, False, True, True, True, False, False, True, True,
        False, False, False, False, True, False, False, False, True,
         True, False, False, True, False, False, False, True, True,
        False, True, False, True, False, True, False, False, False,
        False, False, False, False, True, True, True, False, False,
        False, True, True, False, False, True, True, False, False,
        False, True, False, True, False, False, True, False, False,
         True, False, True, False, False, False, False])

[56]:
```
yhat_classes
```

[56]: array([0., 0., 1., 1., 1., 0., 0., 1., 1., 0., 0., 0., 0., 1., 0., 0., 0.,
        1., 1., 0., 0., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0.,
        0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 1.,
        1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 1., 0., 1., 0., 0.,
        0., 0.], dtype=float32)

[57]:
```
testby
```

[57]: array([0., 1., 1., 1., 1., 0., 0., 1., 0., 1., 0., 0., 0., 1., 1., 0., 0.,
        0., 1., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 1., 1., 0., 1.,
        0., 0., 0., 0., 1., 1., 0., 1., 1., 1., 0., 0., 1., 1., 0., 1., 1.,
        1., 0., 0., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 1., 1., 0., 1.,
        0., 0.], dtype=float32)

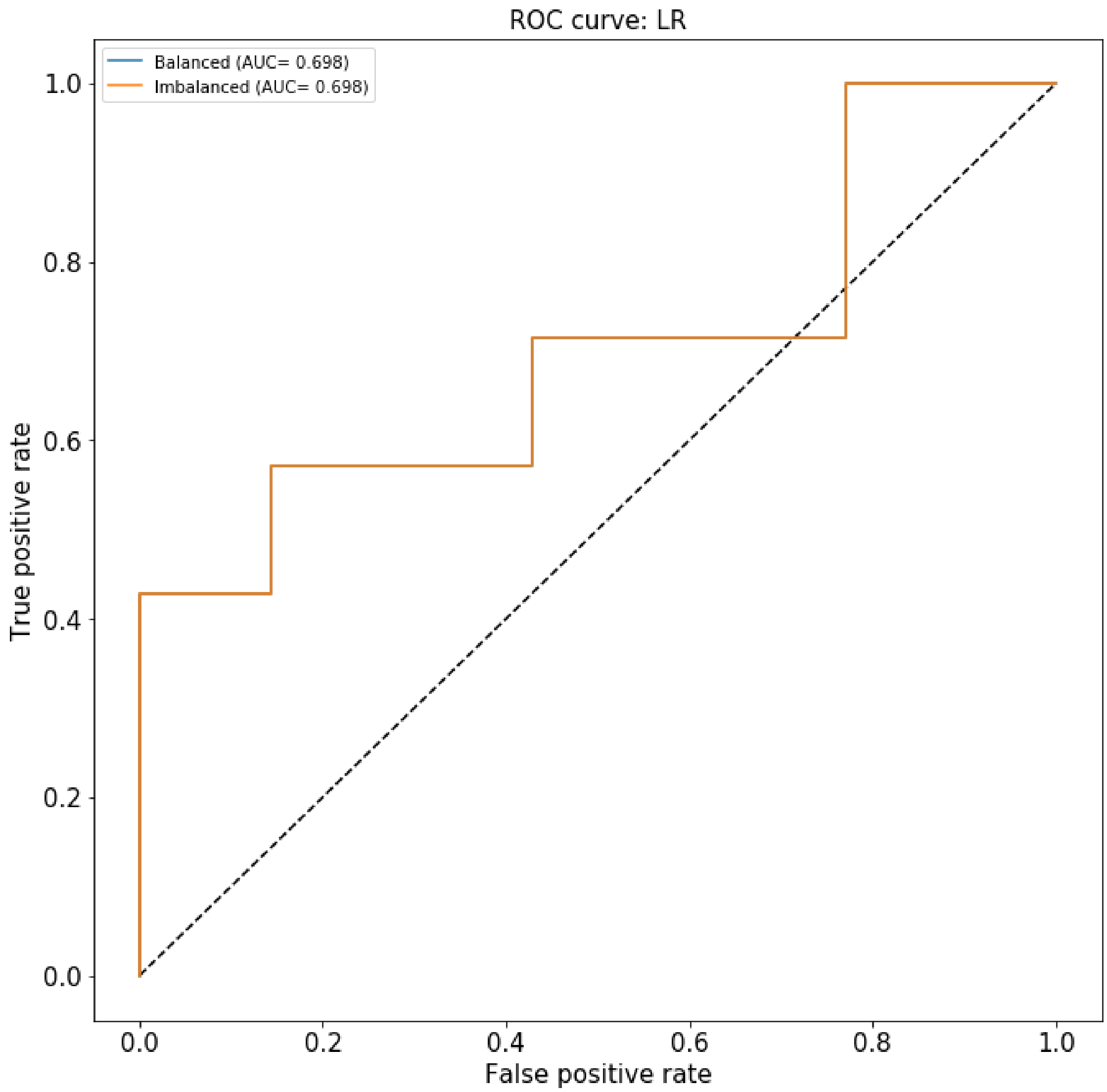
[ ]:

2 IMBALANCED TESTING

[58]:
```
NBname='_F-elNetim'

# xim_better_test=testim.reshape(testim.shape[0],testim.shape[1])
# yim_better_test=tlabelsim.reshape(tlabelsim.shape[0],tlabelsim.shape[1])
# yim_better_test=yim_better_test[:,1]


y_pred = LR.predict_proba(xim_better_test)#.ravel()
fpr_0, tpr_0, thresholds_0 = roc_curve(yim_better_test, y_pred[:,1])
fpr_x.append(fpr_0)
tpr_x.append(tpr_0)
thresholds_x.append(thresholds_0)
auc_x.append(auc(fpr_0, tpr_0))

testim=xim_better_test
```

```
# predict probabilities for testim set
yhat_probs = LR.predict_proba(testim)
# predict crisp classes for testim set
yhat_classes = LR.predict(testim)
# reduce to 1d array
testimy=tlabelsim[:,1]

#testimy1=tlabels[:,1]
#yhat_probs = yhat_probs[:, 0]
#yhat_classes = yhat_classes[:, 0]

# accuracy: (tp + tn) / (p + n)
acc_S.append(accuracy_score(testimy, yhat_classes))
#print('Accuracy: %f' % accuracy_score(testimy, yhat_classes))

#precision tp / (tp + fp)
pre_S.append(precision_score(testimy, yhat_classes))
#print('Precision: %f' % precision_score(testimy, yhat_classes))

#recall: tp / (tp + fn)
rec_S.append(recall_score(testimy, yhat_classes))
#print('Recall: %f' % recall_score(testimy, yhat_classes))

# f1: 2 tp / (2 tp + fp + fn)
f1_S.append(f1_score(testimy, yhat_classes))
#print('F1 score: %f' % f1_score(testimy, yhat_classes))

# kappa
kap_S.append(cohen_kappa_score(testimy, yhat_classes))
#print('Cohens kappa: %f' % cohen_kappa_score(testimy, yhat_classes))

# confusion matrix
mat_S.append(confusion_matrix(testimy, yhat_classes))
#print(confusion_matrix(testimy, yhat_classes))

with open('perform'+NBname+'.txt', "w") as f:
    f.writelines("##THE TWO LINES ARE FOR BALANCED AND IMBALALANCED TEST\n")
    f.writelines("#AUC \t Accuracy \t Precision \t Recall \t F1 \t Kappa\n")
    f.writelines(map("{}\t{}\t{}\t{}\t{}\t{}\n".format, auc_x, acc_S, pre_S,␣
↪rec_S, f1_S, kap_S))
    f.writelines("#TRUE_SENSITIVE \t TRUE_RESISTANT\n")
    for x in range(len(fpr_x)):
        f.writelines(map("{}\n".format, mat_S[x]))
        #f.writelines(map("{}\t{}\t{}\n".format, fpr_x[x], tpr_x[x],␣
↪thresholds_x[x]))
    f.writelines("#FPR \t TPR \t THRESHOLDs\n")
    for x in range(len(fpr_x)):
```

```
        #f.writelines(map("{}\n".format, mat_S[x]))
        f.writelines(map("{}\t{}\t{}\n".format, fpr_x[x], tpr_x[x],␣
↪thresholds_x[x]))
        f.writelines("#NEXT\n")
# =============================================================================
# # THIS IS THE UNBIASED testim; DO NOT UNCOMMENT UNTIL THE END
# =============================================================================

plot_auc(auc_x,fpr_x,tpr_x,NBname)
```



2.1 to see which samples were correctly classified …

[59]:
```
yhat_probs[yhat_probs[:,1]>=0.5,1]
```

[59]:
```
array([0.77234803, 0.61979184, 0.9519404 , 0.56077694, 0.6505654 ,
       0.57811525, 0.91791241, 0.94451295, 0.61222333, 0.68372015])
```

[60]:
```
yhat_probs[:,1]>=0.5
```

[60]:
```
array([False, False,  True, False, False, False, False,  True, False,
        True,  True, False,  True, False, False, False, False, False,
       False,  True, False, False, False, False, False, False, False,
        True, False, False,  True, False, False,  True, False, False,
        True, False, False, False, False, False])
```

[61]:
```
yhat_classes
```

[61]:
```
array([0., 0., 1., 0., 0., 0., 0., 1., 0., 1., 1., 0., 1., 0., 0., 0., 0.,
       0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 1.,
       0., 0., 1., 0., 0., 0., 0., 0.], dtype=float32)
```

[62]:
```
testimy
```

[62]:
```
array([0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

[ ]:
```

```

3 MISCELLANEOUS

[40]:
```
mat_S
```

[40]:
```
[array([[29,  6],
        [15, 20]]), array([[29,  6],
        [ 3,  4]])]
```

[41]:
```
auc_x
```

[41]:
```
[0.6979591836734693, 0.6979591836734693]
```

4 END OF TESTING

[42]:
```python
print(str(datetime.datetime.now()))
```

```
2019-10-03 15:09:52.177842
```

[ ]:
```python

```