Compare FL, SL, GEN, and Logistic regressions (LASSO, Ridge, without penalty - Sklearn)

Original image size is 128 X 128. Here we compress it to 32 x 32

Orignal data has 4 classes from not demented to moderate demented. Here we pick 1000 images from non-demented labelled as "healthy" - 0, and 200 mild-demented images labelled as "sick" - 1 for a binary classification task. The tuning set is 40% of the whole data; training set 40% and test set 20%: 480/480/240, and p = 1024

In [1]:
```
import cv2
import PIL
import matplotlib.pyplot as plt
import numpy as np
import pathlib
```

In [2]:
```
path = 'C:/Users/sswei/Desktop/running time/AD2/'
data_dir = pathlib.Path(path)
```

In [3]:
```
sick = list(data_dir.glob('1/*'))
```

In [4]:
```
healthy = list(data_dir.glob('0/*'))
```

In [5]:
```
len(healthy)
```

Out[5]:
```
1000
```

Compress image size to 32 x 32 pixels (speed up experiments)

In [6]:
```
X1_all = np.vstack([np.asarray(cv2.resize(plt.imread(str(sick[i])), (32, 32))).fl
```

In [7]:
```
y1_all = np.ones(len(sick))
```

In [8]:
```
X0_all = np.vstack([np.asarray(cv2.resize(plt.imread(str(healthy[i])), (32, 32))
```

In [9]:
```
y0_all = np.zeros(len(healthy))
```

Make tuning, train, test sets

In [10]:
```
from sklearn.model_selection import train_test_split
```

In [11]:
```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1_all, y1_all, test_size
```

In [12]:
```
X1_train, X1_val, y1_train, y1_val = train_test_split(X1_train, y1_train, test_si
```

In [13]:
```
X0_train, X0_test, y0_train, y0_test = train_test_split(X0_all, y0_all, test_size
```

In [14]:
```
X0_train, X0_val, y0_train, y0_val = train_test_split(X0_train, y0_train, test_si
```

In [15]:
```
X_train = np.concatenate((X1_train, X0_train))
```

In [16]:
```
y_train  = np.concatenate((y1_train, y0_train))
```

In [17]:
```
X_test = np.concatenate((X1_test, X0_test))
y_test  = np.concatenate((y1_test, y0_test))
```

In [18]:
```
X_val = np.concatenate((X1_val, X0_val))
y_val  = np.concatenate((y1_val, y0_val))
```

normalize each feature to have mean 0, std 1

In [19]:
```
from sklearn import preprocessing
```

In [20]:
```
X_test = preprocessing.StandardScaler().fit(X_test).transform(X_test)
```

In [21]:
```
X_train = preprocessing.StandardScaler().fit(X_train).transform(X_train)
```

In [22]:
```
X_val = preprocessing.StandardScaler().fit(X_val).transform(X_val)
```

Fit graph based models:

Tuning over small grid {'l1': [0, 0.1], 'l2': [0, 0.001, 0.01, 0.05, 0.1, 0.25, 0.5, 1, 2, 5]}

Graph is 2-D grid (natural choice)

In [24]:
```
from signals import *
from skest import *
```

In [25]:
```
D = grid_incidence(32)
```

In [26]:
```
X_val.shape
```

Out[26]:
```
(480, 1024)
```

Tuning

In [27]:
```
naive_cv_logit(Log_FL, X_val, y_val, D)
```

Out[27]:
```
({'l1': 0, 'l2': 0.25}, 169.4143307209015)
```

In [28]:
```
naive_cv_logit(Log_SL, X_val, y_val, D)
```

```
C:\Users\sswei\anaconda3\lib\site-packages\cvxpy\problems\problem.py:1296: User
Warning: Solution may be inaccurate. Try another solver, adjusting the solver s
ettings, or solve with verbose=True for more information.
  warnings.warn(
```

Out[28]:
```
({'l1': 0, 'l2': 2}, 195.77160143852234)
```

In [29]:
```
naive_cv_logit(Log_OUR, X_val, y_val, D)
```

Out[29]:
```
({'l1': 0.1, 'l2': 0.001}, 195.73687195777893)
```

Fitting graph based methods

In [30]:
```
clf1 = Log_FL(0, 0.25, D).fit(X_train, y_train)
```

In [31]:
```
clf2 = Log_SL(0, 2, D).fit(X_train, y_train)
```

In [46]:
```
clf3 = Log_OUR(0.1, 0.001, D).fit(X_train, y_train)
```

Prediction Accuracy and sensitivity

In [47]:
```
def acc(clf):
    return 1 - np.sum(np.abs(y_test - clf.predict(X_test)))/len(y_test)
def sen(clf):
    return 1 - np.sum(np.abs(y_test[y_test == 1] - clf.predict(X_test[y_test == 1
```

FL:

accuracy

In [48]:
```
acc(clf1)
```

Out[48]:
```
0.6916666666666667
```

sensitivity:

In [49]:
```
sen(clf1)
```
Out[49]:
```
0.925
```

SL:

In [50]:
```
acc(clf2)
```
Out[50]:
```
0.7
```

In [51]:
```
sen(clf2)
```
Out[51]:
```
0.9
```

GEN:

In [52]:
```
acc(clf3)
```
Out[52]:
```
0.6916666666666667
```

In [53]:
```
sen(clf3)
```
Out[53]:
```
0.925
```

We may also compare to Logistic regression methods (ridge, lasso, non-penalty)

In [64]:
```
import sklearn.linear_model
```

Without any penalty

In [93]:
```
clf4 = sklearn.linear_model.LogisticRegression(penalty = 'none')
```

In [94]:
```
clf4.fit(X_train, y_train)
```
Out[94]:
```
LogisticRegression(penalty='none')
```

In [95]:
```
acc(clf4)
```
Out[95]:
```
0.9416666666666667
```

In [96]:
```
sen(clf4)
```
Out[96]:
```
0.825
```

lasso

In [97]:
```
clf5 = sklearn.linear_model.LogisticRegression(penalty = 'l1', solver = 'saga')
```

In [98]:
```
clf5.fit(X_train, y_train)
```

```
C:\Users\sswei\anaconda3\lib\site-packages\sklearn\linear_model\_sag.py:328: Co
nvergenceWarning: The max_iter was reached which means the coef_ did not conver
ge
  warnings.warn("The max_iter was reached which means "
```

Out[98]:
```
LogisticRegression(penalty='l1', solver='saga')
```

In [99]:
```
acc(clf5)
```

Out[99]:
```
0.95
```

In [100]:
```
sen(clf5)
```

Out[100]:
```
0.8
```

Ridge

In [102]:
```
clf6 = sklearn.linear_model.LogisticRegression()
```

In [103]:
```
clf6.fit(X_train, y_train)
```

Out[103]:
```
LogisticRegression()
```

In [104]:
```
acc(clf6)
```

Out[104]:
```
0.9666666666666667
```

In [106]:
```
sen(clf6)
```

Out[106]:
```
0.85
```

Conclusions:

We see that graph based method always have better sensitivity but worse accuracy. One reason is: Here I'm using loglik for cross validation scorer instead of using prediction accuracy.

In [ ]:
```

```