Version 2. Note that in version 1, cv scorer is loglikelihood, here cv scorer is prediction accuarcy

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 version 2 grid gridlogit = {'l1': [0, 0.1, 1], 'l2': [0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 0.75, 1, 2]}

Graph is 2-D grid (natural choice)

```
In [23]: from signals import *
         from skest import *
```

In [24]:
```python
D = grid_incidence(32)
```

In [25]:
```python
X_val.shape
```

Out[25]: (480, 1024)

Tuning

Note that here we are using prediction accuracy as scorer

Caution! GridsearchCV default does not shuffle the data. Here it is necessary to shuffle. ? Seems to make no difference

In [26]:
```python
from sklearn.utils import shuffle
X_val, y_val = shuffle(X_val, y_val)
```

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

```
C:\Users\sswei\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:9
18: UserWarning: One or more of the test scores are non-finite: [       nan 0.6
25       0.69791667 0.6875     0.6875     0.6875
 0.6875     0.6875     0.6875     0.69166667 0.71458333 0.65208333
 0.66041667 0.67708333 0.68958333 0.69583333 0.69791667 0.69375
 0.52916667 0.46666667 0.51875    0.52083333 0.52083333 0.53125
 0.53958333 0.59375    0.625     ]
  warnings.warn(
```

Out[27]: ({'l1': 0.1, 'l2': 0.01}, 207.8515820503235)

In [28]:
```python
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': 0.1, 'l2': 0.2}, 227.5220239162445)

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

Out[29]: ({'l1': 0.1, 'l2': 0}, 234.68088698387146)

Fitting graph based methods

In [31]:
```python
X_train, y_train = shuffle(X_train, y_train)
```

In [32]:
```python
clf1 = Log_FL(0.1, 0.01, D).fit(X_train, y_train)
```

In [33]:
```
clf2 = Log_SL(0.1, 0.2, D).fit(X_train, y_train)
```

In [34]:
```
clf3 = Log_OUR(0.1, 0, D).fit(X_train, y_train)
```

Prediction Accuracy and sensitivity

In [35]:
```
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
```

In [36]:
```
X_test, y_test = shuffle(X_test, y_test)
```

FL:

accuracy

In [37]:
```
acc(clf1)
```

Out[37]: 0.75

sensitivity:

In [38]:
```
sen(clf1)
```

Out[38]: 0.85

SL:

In [39]:
```
acc(clf2)
```

Out[39]: 0.7416666666666667

In [40]:
```
sen(clf2)
```

Out[40]: 0.875

GEN:

In [43]:
```
acc(clf3)
```

Out[43]: 0.6708333333333334

In [44]:
```python
sen(clf3)
```

Out[44]:
```
0.875
```

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

In [45]:
```python
import sklearn.linear_model
```

Without any penalty

In [46]:
```python
clf4 = sklearn.linear_model.LogisticRegression(penalty = 'none')
```

In [47]:
```python
clf4.fit(X_train, y_train)
```

Out[47]:
```
LogisticRegression(penalty='none')
```

In [48]:
```python
acc(clf4)
```

Out[48]:
```
0.9333333333333333
```

In [49]:
```python
sen(clf4)
```

Out[49]:
```
0.775
```

lasso

In [50]:
```python
clf5 = sklearn.linear_model.LogisticRegression(penalty = 'l1', solver = 'saga')
```

In [51]:
```python
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[51]:
```
LogisticRegression(penalty='l1', solver='saga')
```

In [52]:
```python
acc(clf5)
```

Out[52]:
```
0.9375
```

In [53]:
```python
sen(clf5)
```

Out[53]:
```
0.7
```

Ridge

In [54]:
```
clf6 = sklearn.linear_model.LogisticRegression()
```

In [55]:
```
clf6.fit(X_train, y_train)
```

Out[55]:
```
LogisticRegression()
```

In [56]:
```
acc(clf6)
```

Out[56]:
```
0.95
```

In [57]:
```
sen(clf6)
```

Out[57]:
```
0.75
```

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 [ ]:
```

```