Q2_starter

April 18, 2022

[7]:
```python
## Breast Cancer LASSO Exploration
## Prepare workspace
from scipy.io import loadmat
import numpy as np
import math
X = loadmat("BreastCancer.mat")['X']
y = loadmat("BreastCancer.mat")['y']
```

[8]:
```python
def ista_solve_hot( A, d, la_array ):
    # ista_solve_hot: Iterative soft-thresholding for multiple values of
    # lambda with hot start for each case - the converged value for the previous
    # value of lambda is used as an initial condition for the current lambda.
    # this function solves the minimization problem
    # Minimize |Ax-d|_2^2 + lambda*|x|_1 (Lasso regression)
    # using iterative soft-thresholding.
    max_iter = 10**4
    tol = 10**(-3)
    tau = 1/np.linalg.norm(A,2)**2
    n = A.shape[1]
    w = np.zeros((n,1))
    num_lam = len(la_array)
    X = np.zeros((n, num_lam))
    for i, each_lambda in enumerate(la_array):
        for j in range(max_iter):
            z = w - tau*(A.T@(A@w-d))
            w_old = w
            w = np.sign(z) * np.clip(np.abs(z)-tau*each_lambda/2, 0, np.inf)
            X[:, i:i+1] = w
            if np.linalg.norm(w - w_old) < tol:
                break
    return X
lam_vals = []
i = 1e-6
while i < 19:
    lam_vals.append(i)
    i += math.log(2)
lam_vals.append(20)
```

[78]:

```
##  10-fold CV

# each row of setindices denotes the starting an ending index for one
# partition of the data: 5 sets of 30 samples and 5 sets of 29 samples
setindices =␣
↪[[1,30],[31,60],[61,90],[91,120],[121,150],[151,179],[180,208],[209,237],[238,266],[267,295

# each row of holdoutindices denotes the partitions that are held out from
# the training set
holdoutindices = [[1,2],[2,3],[3,4],[4,5],[5,6],[7,8],[9,10],[10,1]]

cases = len(holdoutindices)

# be sure to initiate the quantities you want to measure before looping
# through the various training, validation, and test partitions
#
#
#
errors = []
W = []

# Loop over various cases
for j in range(cases):
    # row indices of first validation set
    v1_ind = np.
↪arange(setindices[holdoutindices[j][0]-1][0]-1,setindices[holdoutindices[j][0]-1][1])

    # row indices of second validation set
    v2_ind = np.
↪arange(setindices[holdoutindices[j][1]-1][0]-1,setindices[holdoutindices[j][1]-1][1])

    # row indices of training set
    trn_ind = list(set(range(295))-set(v1_ind)-set(v2_ind))

    # define matrix of features and labels corresponding to first
    # validation set
    Av1 = X[v1_ind,:]
    bv1 = y[v1_ind]

    # define matrix of features and labels corresponding to second
    # validation set
    Av2 = X[v2_ind,:]
    bv2 = y[v2_ind]

    # define matrix of features and labels corresponding to the
    # training set
    At = X[trn_ind,:]
```

```
    bt = y[trn_ind]

#     print(len(v1_ind), len(v2_ind), len(trn_ind))
    w_case = ista_solve_hot(At,bt,lam_vals) # w matrix of w's for each lambda
    error_case = []
    for i in range(len(lam_vals)):
        error_case.append( (np.sign(Av1@w_case[:,i]) - bv1).sum() /len(bv1))
    errors.append(error_case)
    W.append(w_case)

# Use training data to learn classifier
# W = ista_solve_hot(At,bt,lam_vals)
```

[79]:
```
# Find best lambda value using first validation set, then evaluate
# performance on second validation set, and accumulate performance metrics
# over all cases partitions
# We search for the best lambda
min_case_idx = None
min_error = None
min_lambda = None
for i in range(len(errors)): # iterate through each case => i=0 is case of [1,2]
    change_lambda = False
    for j in range(len(errors[i])):
        if min_error == None or errors[i][j] <= min_error:
            min_error = errors[i][j]
            min_lambda = lam_vals[j]
            change_lambda = True
    if change_lambda:
        min_case_idx = i
```

[80]:
```
# Calculate with the best case of houlout and lambda
v1_ind = np.
 ↪arange(setindices[holdoutindices[i][0]-1][0]-1,setindices[holdoutindices[i][0]-1][1])

v2_ind = np.
 ↪arange(setindices[holdoutindices[i][1]-1][0]-1,setindices[holdoutindices[i][1]-1][1])

trn_ind = list(set(range(295))-set(v1_ind)-set(v2_ind))

Av1 = X[v1_ind,:]
bv1 = y[v1_ind]

Av2 = X[v2_ind,:]
bv2 = y[v2_ind]

At = X[trn_ind,:]
bt = y[trn_ind]
```

```python
w = ista_solve_hot(At,bt,[min_lambda])

y1_pred = np.sign(Av1@w) - bv1
y2_pred = np.sign(Av2@w) - bv2

y_pred = np.concatenate((y1_pred, y2_pred))

misclassifications = 0
for i in y_pred:
    if i != 0:
        misclassifications += 1

error_rate = misclassifications / len(y_pred)
squared_error = (np.linalg.norm(Av1@w - bv1))**2 + (np.linalg.norm(Av2@w -␣
↪bv2))**2

print("Error rate: " , error_rate)
print("Squared error: ", squared_error)
```

```
Error rate:  0.3898305084745763
Squared error:  55.86164056908563
```

[99]:
```python
length = len(At.transpose())
lambdas_matrix = np.zeros((length,length))
np.fill_diagonal(lambdas_matrix, min_lambda)
```

[101]:
```python
w_ridge = np.linalg.inv(At.transpose()@At + lambdas_matrix)@At.transpose()@bt
w_ridge
```

[101]:
```
array([[-0.00472367],
       [ 0.00706112],
       [ 0.00881967],
       …,
       [-0.00279695],
       [-0.00814373],
       [-0.00346693]])
```

[105]:
```python
y1_pred_ridge = np.sign(Av1@w_ridge) - bv1
y1_pred_ridge = np.sign(Av2@w_ridge) - bv2

y_pred_ridge = np.concatenate((y1_pred_ridge, y1_pred_ridge))

misclassifications_ridge = 0
for i in y_pred_ridge:
    if i != 0:
        misclassifications_ridge += 1
```

```python
error_rate_ridge = misclassifications_ridge / len(y_pred_ridge)
squared_error_ridge = (np.linalg.norm(Av1@w_ridge - bv1))**2 + (np.linalg.
 ↪norm(Av2@w_ridge - bv2))**2

print("Error rate: " , error_rate_ridge)
print("Squared error: ", squared_error_ridge)
```

```
Error rate:  0.2
Squared error:  62.012381589953506
```

We can see that the error rate is lower if we use ridge regression but the squared error is higher

[ ]:

```python

```