# ista_solve_hot

April 13, 2021

## 1 Q1

[172]:
```
import numpy as np
import scipy.io as sp
import matplotlib.pyplot as plt
```

[173]:
```
data = sp.loadmat("BreastCancer.mat")
```

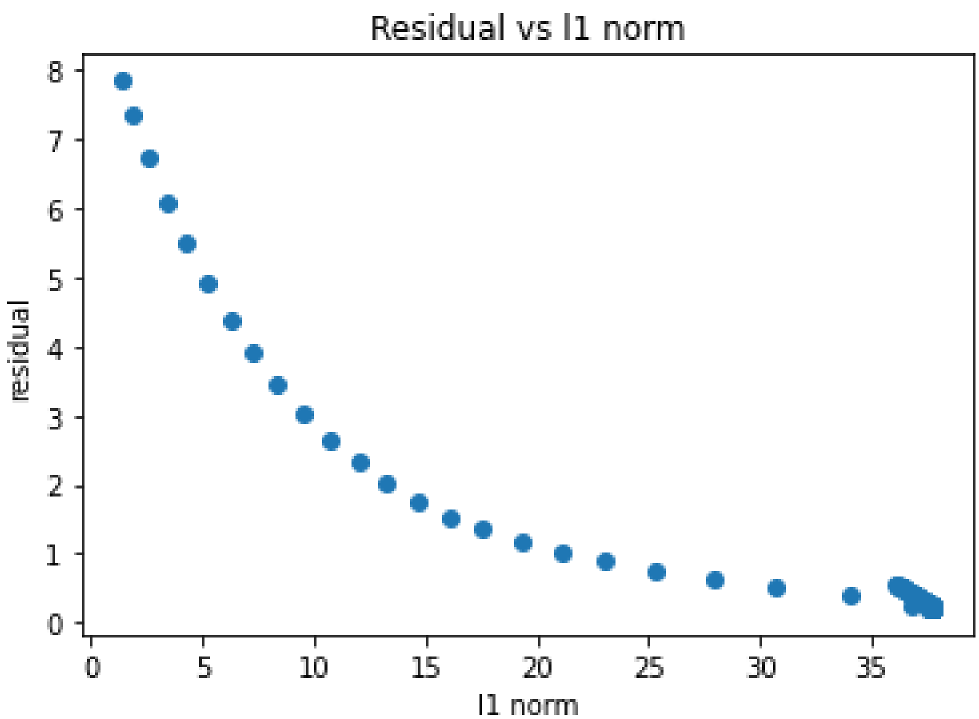
[174]:
```
def ista_solve_hot( A, d, la_array ):
    # ista_solve_hot: Iterative soft-thresholding for multiple values of
    # lambda with hot start for each case - the converged value for the previous
    # value of lambda is used as an initial condition for the current lambda.
    # this function 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
```

### 1.1 (a)

[175]:
```
A = data['X'][:100]
d = data['y'][:100]
lambda_arr = np.logspace(-8,np.log10(20),100)

wmat = ista_solve_hot(A,d,lambda_arr)
```

[176]:
```
plot_x = []
plot_y = []

for i in range(len(wmat[0])):
    w = wmat[:, i:i+1]
    plot_x.append(np.linalg.norm(w,ord=1))
    plot_y.append(np.linalg.norm(A@w-d))

plt.title("Residual vs l1 norm")
plt.xlabel("l1 norm")
plt.ylabel("residual")
plt.scatter(plot_x,plot_y)
```

[176]:
```
<matplotlib.collections.PathCollection at 0x7f51dc63f390>
```

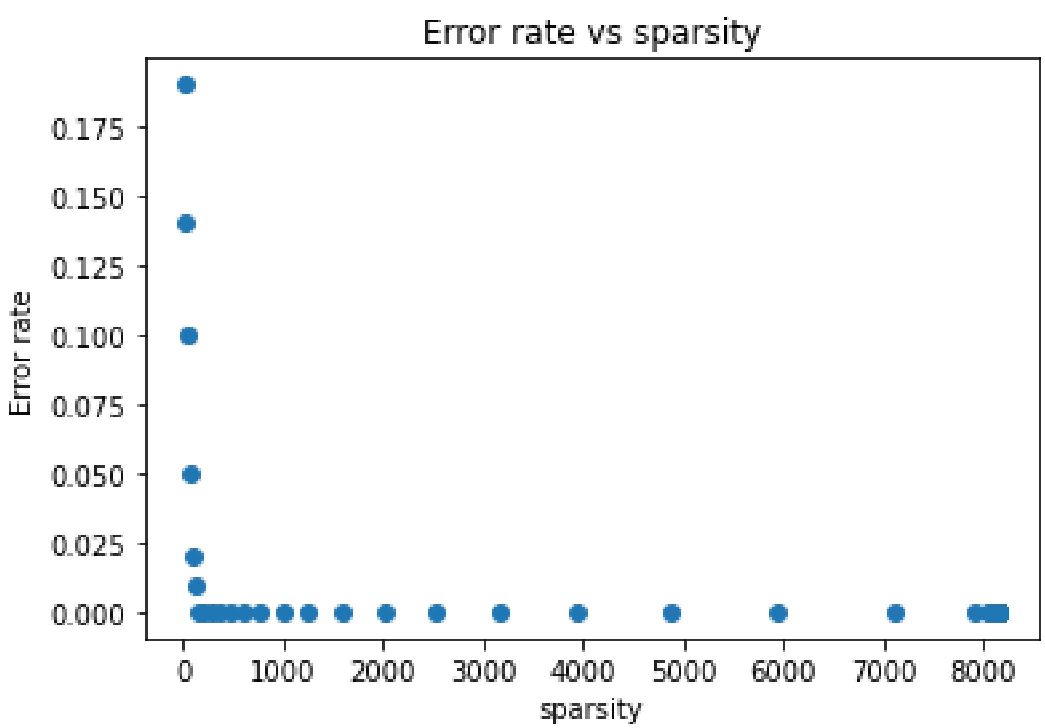


## 1.2 The residual and l1 norm seem to have an inverse relationship

## 1.3 (b)

[177]:
```python
plot_x = []
plot_y = []

for i in range(len(wmat[0])):
    w = wmat[:, i:i+1]
    d_hat = np.sign(A@w)
    error_vec = [0 if m[0]==m[1] else 1 for m in np.hstack((d_hat, d))]
    error_rate = sum(error_vec)/len(error_vec)
    sparsity = 0
    for j in w:
        if j!=0:
            sparsity+=1
    plot_x.append(sparsity)
    plot_y.append(error_rate)

plt.title("Error rate vs sparsity")
plt.xlabel("sparsity")
plt.ylabel("Error rate")
plt.scatter(plot_x,plot_y)
```

[177]:
```
<matplotlib.collections.PathCollection at 0x7f51dc5afd50>
```



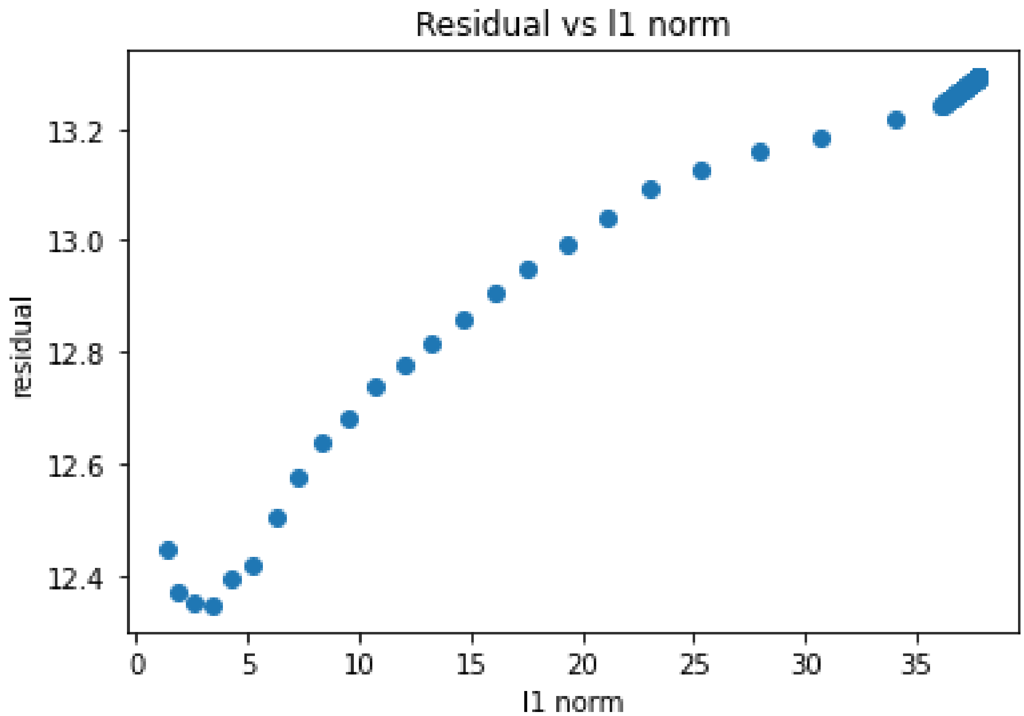
### 1.4 As the sparsity goes up, the error rate goes down. Initially, increasing the sparsity even by a little bit makes the error rate drop drastically. The drop in error rate becomes insignificant beyond a particular sparsity telling that a good sparse solution exists which focuses on only a few significant features

## 2 (c)

[178]:
```
A_test = data['X'][101:]
d_test = data['y'][101:]
```

[179]:
```
plot_x = []
plot_y = []

for i in range(len(wmat[0])):
    w = wmat[:, i:i+1]
    plot_x.append(np.linalg.norm(w,ord=1))
    plot_y.append(np.linalg.norm(A_test@w-d_test))

plt.title("Residual vs l1 norm")
plt.xlabel("l1 norm")
plt.ylabel("residual")
plt.scatter(plot_x,plot_y)
```

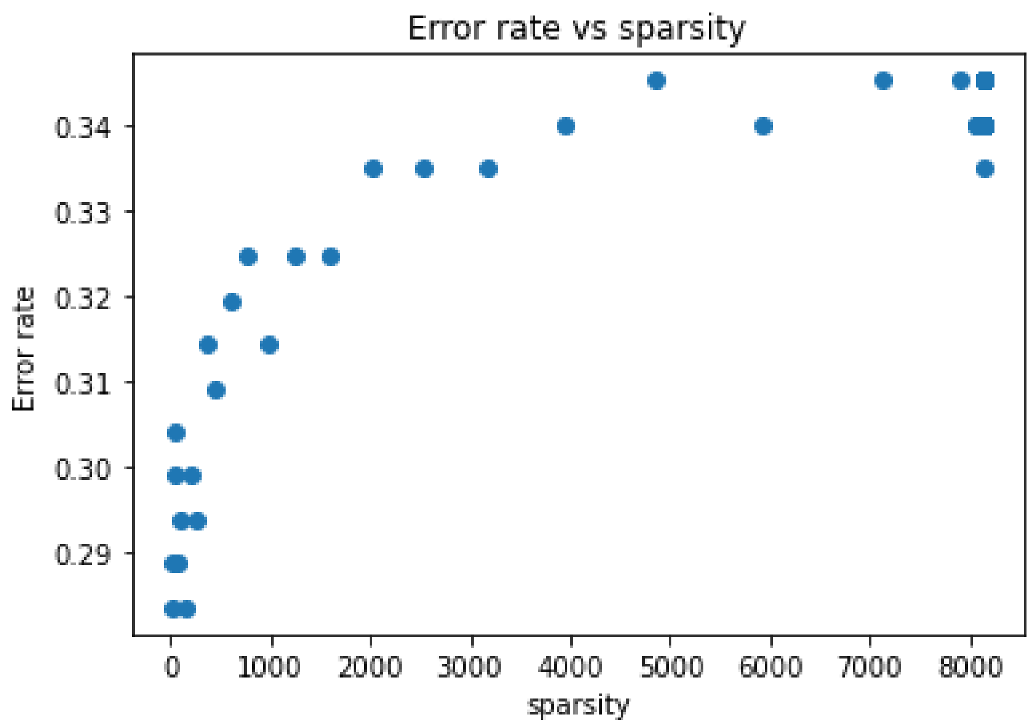
[179]:
```
<matplotlib.collections.PathCollection at 0x7f51dc5a2e90>
```




### 2.1 There seems to be a linear relationship between residual and l1 norm, contrary to the previous graph from our training data

[180]:
```
plot_x = []
plot_y = []

for i in range(len(wmat[0])):
    w = wmat[:, i:i+1]
    d_hat = np.sign(A_test@w)
    error_vec = [0 if m[0]==m[1] else 1 for m in np.hstack((d_hat, d_test))]
    error_rate = sum(error_vec)/len(error_vec)
    sparsity = 0
    for j in w:
        if j!=0:
            sparsity+=1
    plot_x.append(sparsity)
    plot_y.append(error_rate)

plt.title("Error rate vs sparsity")
plt.xlabel("sparsity")
plt.ylabel("Error rate")
plt.scatter(plot_x,plot_y)
```

[180]: <matplotlib.collections.PathCollection at 0x7f51dc513ad0>




### 2.2 Here, the error rate seems to increase by a lot when increasing sparsity even by a bit. Our model does not seem to be a good fit for the data because taking more features into consideration makes the error rate go up

[ ]:

# Q2_starter

April 13, 2021

## 1 Q2

[26]:
```
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
```

[27]:
```
def ridge( A, d, la_array ):
    n = A.shape[1]
    num_lam = len(la_array)
    X = np.zeros((n, num_lam))
    for i, each_lambda in enumerate(la_array):
        w = A.T@np.linalg.inv(A@A.T + each_lambda*np.eye(len(A@A.T)))@d
        X[:, i:i+1] = w
    return X
```

[28]:
```
## Breast Cancer LASSO Exploration
## Prepare workspace
from scipy.io import loadmat
import numpy as np
X = loadmat("BreastCancer.mat")['X']
y = loadmat("BreastCancer.mat")['y']

##  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
avg_sqd_error_lasso = 0
avg_sqd_error_ridge = 0
avg_error_rate_ridge = 0
avg_error_rate_lasso = 0
lam_vals = np.logspace(-8,np.log10(20),100)

# Loop over various cases
for j in range(cases):
    print("Iteration: ", j+1)
    # 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("Set sizes")
    print("v1: ", len(v1_ind), "v2: ", len(v2_ind), "training: ", len(trn_ind))
    # Use training data to learn classifier
    wmat_lasso = ista_solve_hot(At,bt,lam_vals)
    wmat_ridge = ridge(At,bt,lam_vals)
    lam_lasso = None
    w_opt_lasso = None
    min_error_lasso = -1
    lam_ridge = None
    w_opt_ridge= None
    min_error_ridge = -1
    # Find best lambda value using first validation set
    for i in range(len(wmat_lasso[0])):
        w_lasso = wmat_lasso[:, i:i+1]
        d_hat_lasso = np.sign(Av1@w_lasso)
        error_vec_lasso = [0 if m[0]==m[1] else 1 for m in np.
↪hstack((d_hat_lasso, bv1))]
        error_rate_lasso = sum(error_vec_lasso)/len(error_vec_lasso)
        if min_error_lasso == -1 or error_rate_lasso<min_error_lasso:
            min_error_lasso = error_rate_lasso
            lam_lasso = lam_vals[i]
            w_opt_lasso = w_lasso

        w_ridge = wmat_ridge[:, i:i+1]
        d_hat_ridge = np.sign(Av1@w_ridge)
        error_vec_ridge = [0 if m[0]==m[1] else 1 for m in np.
↪hstack((d_hat_ridge, bv1))]
        error_rate_ridge = sum(error_vec_ridge)/len(error_vec_ridge)
        if min_error_ridge == -1 or error_rate_ridge<min_error_ridge:
            min_error_ridge = error_rate_ridge
            lam_ridge = lam_vals[i]
            w_opt_ridge = w_ridge

    # Evaluate performance on second validation set
    # and accumulate performance metrics over all cases partitions
    print("Best lambda for LASSO: ", lam_lasso)
    print("Best lambda for ridge regression: ", lam_ridge)
```

```
    d_hat_lasso = np.sign(Av2@w_opt_lasso)
    error_vec_lasso = [0 if m[0]==m[1] else 1 for m in np.hstack((d_hat_lasso,␣
↪bv2))]
    error_rate_lasso = sum(error_vec_lasso)/len(error_vec_lasso)
    avg_error_rate_lasso += error_rate_lasso
    squared_error_lasso = np.linalg.norm(d_hat_lasso-bv2)**2
    avg_sqd_error_lasso += squared_error_lasso
    print("Error rate lasso: ", error_rate_lasso)
    print("Squared error lasso: ", squared_error_lasso)

    d_hat_ridge = np.sign(Av2@w_opt_ridge)
    error_vec_ridge = [0 if m[0]==m[1] else 1 for m in np.hstack((d_hat_ridge,␣
↪bv2))]
    error_rate_ridge = sum(error_vec_ridge)/len(error_vec_ridge)
    avg_error_rate_ridge += error_rate_ridge
    squared_error_ridge = np.linalg.norm(d_hat_ridge-bv2)**2
    avg_sqd_error_ridge += squared_error_ridge
    print("Error rate ridge: ", error_rate_ridge)
    print("Squared error ridge: ", squared_error_ridge)

    print("\n")

avg_error_rate_ridge /= cases
avg_sqd_error_ridge /= cases
avg_error_rate_lasso /= cases
avg_sqd_error_lasso /= cases

print("####################################")
print("Average error rate lasso: ", avg_error_rate_lasso)
print("Average squared error lasso: ", avg_sqd_error_lasso)
print("Average error rate ridge: ", avg_error_rate_ridge)
print("Average squared error ridge: ", avg_sqd_error_ridge)
```

```
Iteration:  1
Set sizes
v1:  30 v2:  30 training:  235
Best lambda for LASSO:  3.5434932692979846
Best lambda for ridge regression:  1e-08
Error rate lasso:  0.43333333333333335
Squared error lasso:  51.99999999999999
Error rate ridge:  0.43333333333333335
Squared error ridge:  51.99999999999999


Iteration:  2
Set sizes
```

```
v1:  30 v2:  30 training:  235
Best lambda for LASSO:  16.109430878355187
Best lambda for ridge regression:  1e-08
Error rate lasso:  0.26666666666666666
Squared error lasso:  32.00000000000001
Error rate ridge:  0.23333333333333334
Squared error ridge:  28.000000000000004


Iteration:  3
Set sizes
v1:  30 v2:  30 training:  235
Best lambda for LASSO:  3.5434932692979846
Best lambda for ridge regression:  1e-08
Error rate lasso:  0.4666666666666667
Squared error lasso:  56.0
Error rate ridge:  0.43333333333333335
Squared error ridge:  51.99999999999999


Iteration:  4
Set sizes
v1:  30 v2:  30 training:  235
Best lambda for LASSO:  6.780801107819854
Best lambda for ridge regression:  1e-08
Error rate lasso:  0.13333333333333333
Squared error lasso:  16.0
Error rate ridge:  0.2
Squared error ridge:  23.999999999999996


Iteration:  5
Set sizes
v1:  30 v2:  29 training:  236
Best lambda for LASSO:  1.4915314679609128
Best lambda for ridge regression:  1e-08
Error rate lasso:  0.3103448275862069
Squared error lasso:  36.0
Error rate ridge:  0.3103448275862069
Squared error ridge:  36.0


Iteration:  6
Set sizes
v1:  29 v2:  29 training:  237
Best lambda for LASSO:  16.109430878355187
Best lambda for ridge regression:  1e-08
Error rate lasso:  0.13793103448275862
```

```
Squared error lasso:  16.0
Error rate ridge:  0.1724137931034483
Squared error ridge:  20.000000000000004


Iteration:  7
Set sizes
v1:  29 v2:  29 training:  237
Best lambda for LASSO:  20.000000000000004
Best lambda for ridge regression:  1e-08
Error rate lasso:  0.5172413793103449
Squared error lasso:  60.00000000000001
Error rate ridge:  0.4482758620689655
Squared error ridge:  51.99999999999999


Iteration:  8
Set sizes
v1:  29 v2:  30 training:  236
Best lambda for LASSO:  3.955000701930882e-07
Best lambda for ridge regression:  1e-08
Error rate lasso:  0.2
Squared error lasso:  23.999999999999996
Error rate ridge:  0.2
Squared error ridge:  23.999999999999996


####################################
Average error rate lasso:  0.3081896551724138
Average squared error lasso:  36.5
Average error rate ridge:  0.3038793103448276
Average squared error ridge:  36.0
```

[ ]: