CS/ECE/ME532 Assignment 8

Note that this assignment can require significant compute time (e.g., over an hour on a modern Mac desktop). You may wish to debug your code on a couple cases before running the full assignment, and then be patient.

1. **Data Fitting vs. Sparsity Tradeoff**. This assignment uses the dataset `BreastCancer.mat` to explore sparse regularization of a least squares problem. The journal article "A gene-expression signature as a predictor of survival in breast cancer" provides background on the role of genes in breast cancer.

   The goal is to solve the Lasso problem

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}\in\mathbb{R}^n} \quad \|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{d}\|_2^2 + \lambda\|\boldsymbol{w}\|_1$$

   Here $\boldsymbol{w}$ is the weight vector applied to the expression levels of 8141 genes and there are 295 patients (feature sets and labels). In this problem we will vary λ to explore the tradeoff between data-fitting and sparsity.

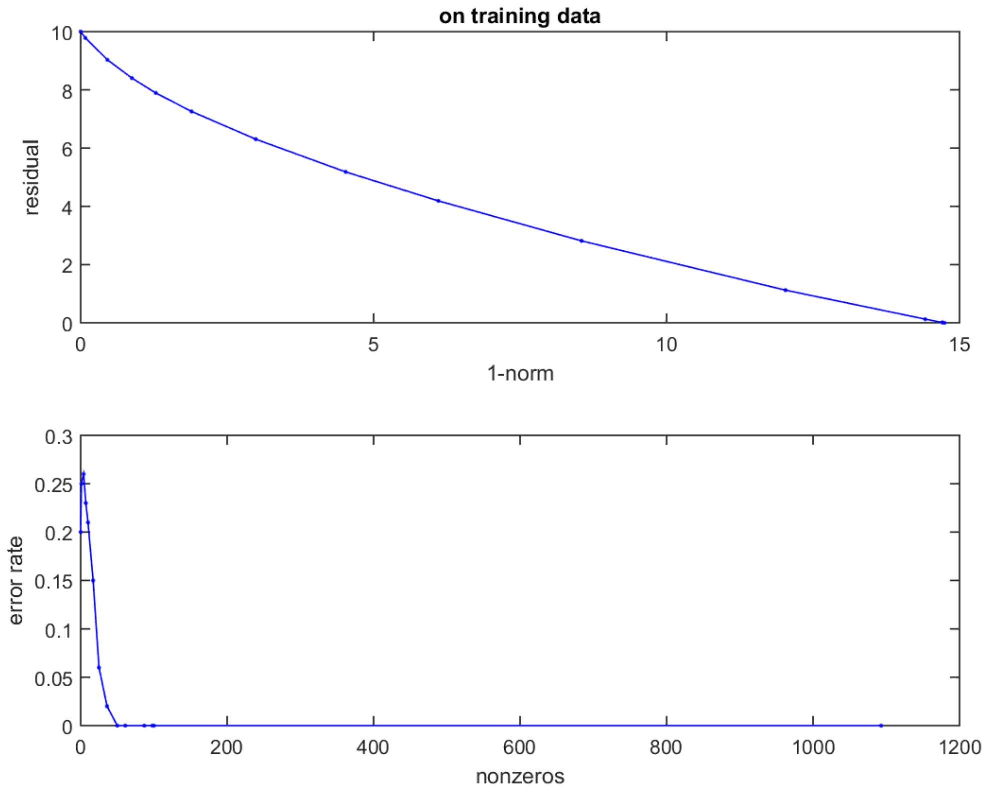
   Scripts that implement iterative soft thresholding via proximal gradient descent to solve the LASSO problem are available. The scripts use a hot start procedure for finding the solution with different values for λ. The initial guess for the next value of λ is the converged solution for the preceding value. This accelerates convergence when subsequent values of λ lead to similar solutions.

   a) Write code to find the optimal weights using only the first 100 patients (first 100 rows). Create a plot with the residual $\|\boldsymbol{A}\boldsymbol{w}^* - \boldsymbol{d}\|_2$ on the vertical-axis and $\|\boldsymbol{w}^*\|_1$ on the horizontal-axis, parameterized by λ. In other words, create the curve by finding $\boldsymbol{w}^*$ for different λ, and plotting $\|\boldsymbol{w}^*\|_1$ vs. $\|\boldsymbol{A}\boldsymbol{w}^* - \boldsymbol{d}\|_2$. Experiment with λ to find a range that captures the variation from the least-squares solution (small λ) to the all zeros solution (large λ). Appropriate values of λ may range from 10^{-6} to 20, spaced logarithmically. Explain the result.

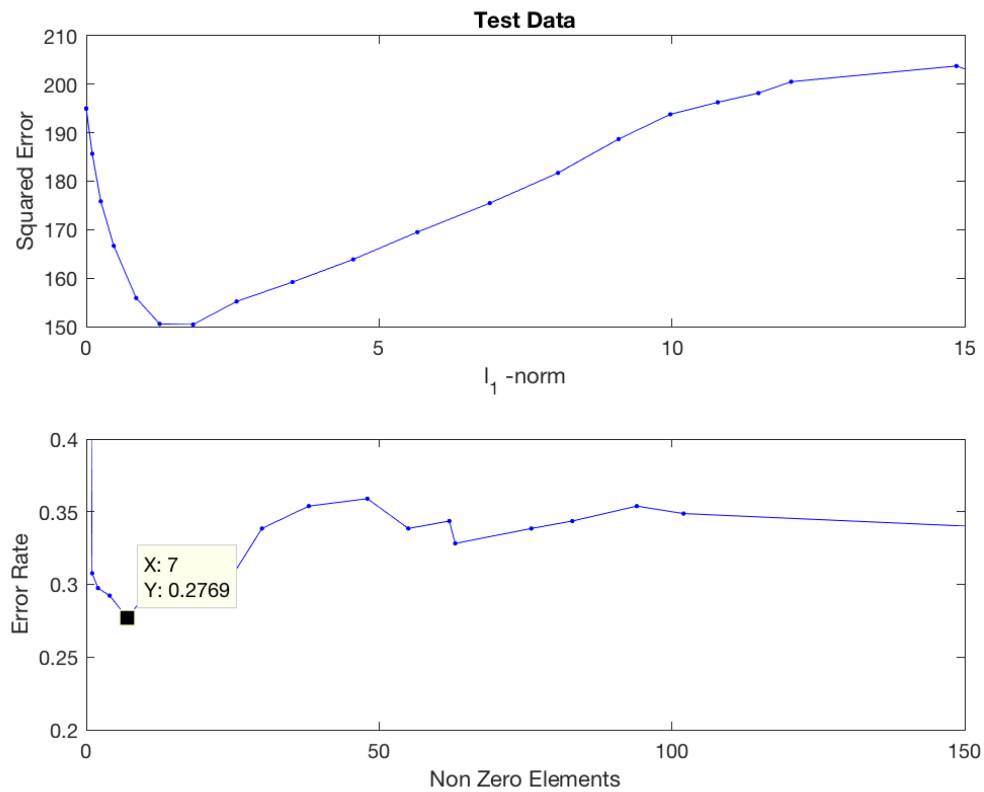
   b) Next use your solutions from part **a)** to plot the error rate on the vertical-axis versus the sparsity on the horizontal-axis as λ varies. Define the error rate as the number of incorrect predictions divided by the total number of predictions and the sparsity as the number of nonzero entries in $\boldsymbol{w}^*$. For this purpose, we'll say an entry w_i is nonzero if $|w_i| > 10^{-6}$. Calculate the error rate using the training data, the data used to find the optimal weights. Explain the result.

   c) Repeat parts **a)** and **b)** to display the residual and error rate, respectively using validation or test data, rows 101-295 of the data matrix, that is, the data not used to design the optimal classifier. Again, explain what you see in each plot.

SOLUTION: Code for solving this (and the next) problem is provided at the end of this document. Here are the plots for parts **a)**, **b)**, and **c)**:



The residual vs 1-norm looks convex as expected. And we see that the more we penalize the 1-norm, the more we encourage sparsity.



This time something curious happens; there is a sweet spot where the estimator achieves both a good error rate as well as a sparse solution! The minimum error rate occurs with seven nonzero weights, which is *very* sparse considering we have over 8000 features. It is possible we could find better solutions if we considered more values for λ.

2. Now compare the performance of the LASSO and ridge regression for the breast cancer dataset using the following steps:

- Randomly split the set of 295 patients into ten subsets of size 29-30.
- Use the data in eight of the subsets to find a solution to the Lasso optimization above and to the ridge regression problem

$$\min_{\boldsymbol{w}} \quad \|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{d}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2 \ .$$

  Repeat this for a range of λ values to obtain a set of solutions $\boldsymbol{w}_\lambda$.

- Compute the prediction error using each $\boldsymbol{w}_\lambda$ on **one** of the remaining two of the ten subsets. Use the solution that has the smallest prediction error to find the best λ. Note that LASSO and ridge regression will produce different best values for λ.

- Compute the test error on the final subset of the data for the choice of λ that minimizes the prediction error. Compute both the squared error and the error rate.

Repeat this process for different subsets of eight training, one tuning (λ) and one testing subsets, and compute the average squared error and average number of misclassifications across all different subsets.

Note that you should use the identity derived in Problem 1 of the Activity 5.2 in order to speed the computation of ridge regression.

SOLUTION: Code for solving the entire problem is provided on subsequent pages. Note, in this scenario it is *much* more efficient to compute:

$$(\boldsymbol{A}^T\boldsymbol{A} + \lambda\boldsymbol{I})^{-1}\boldsymbol{A}^T\boldsymbol{d} = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{d}$$

since the expression on the right requires inverting a 200×200 matrix while the one on the left requires inverting a 8141×8141 matrix. We can further accelerate the task by pre-computing the SVD of $\boldsymbol{A}$.

The error rate for LASSO is 0.3001 while ridge regression gives a slightly worse result of 0.3039. The residual squared error for LASSO is 26.49 while ridge regression gives a slightly worse result of 26.52. Note that although the improvements with LASSO are modest, they are obtain using a much, much sparser set of weights, i.e., using much, much fewer features.

In [1]:
```
import numpy as np
from scipy.io import loadmat
from matplotlib import pyplot as plt
```

In [2]:
```
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 previo
us
    # 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
```

2)

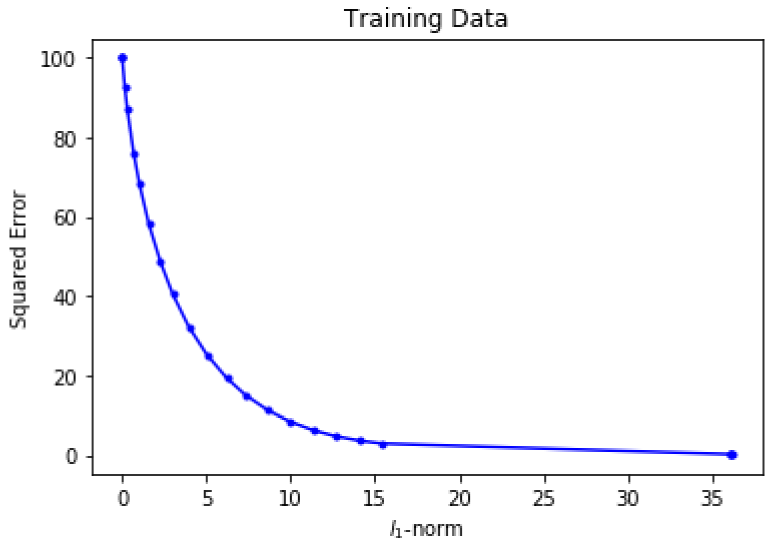
Evaluate results for all lambda

In [3]:
```python
data = loadmat('BreastCancer.mat')
X = data['X']
y = data['y']

At = X[:100, :]
bt = y[:100, :]

Av = X[100:, :]
bv = y[100:, :]

lam_vals = [1e-6, 1e-4, 1e-2, 1e-1]
lam_vals = np.hstack((lam_vals, np.logspace(0,2,num=20)))

number = lam_vals.shape[0]

W = ista_solve_hot(At,bt,lam_vals);

err = []
res = []
norm = []
nonz = []

errv = []
resv = []

for i in range(number):
    err.append(np.mean(np.sign(At@W[:,i:i+1])!=bt))
    res.append(np.linalg.norm(At@W[:,i:i+1]-bt)**2)
    norm.append(np.linalg.norm(W[:,i], 1))
    nonz.append(np.sum(abs(W[:,i])>1e-8))

    errv.append(np.mean(np.sign(Av@W[:,i:i+1])!=bv))
    resv.append(np.linalg.norm(Av@W[:,i:i+1]-bv)**2)
```

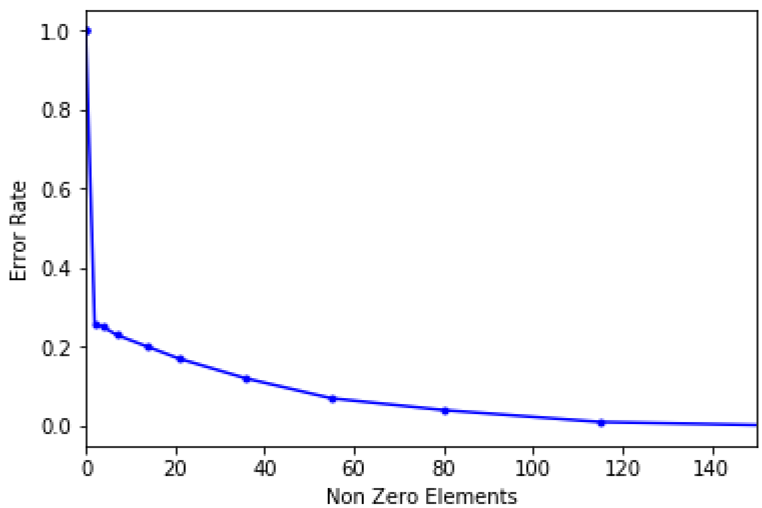
Display results

In [4]:

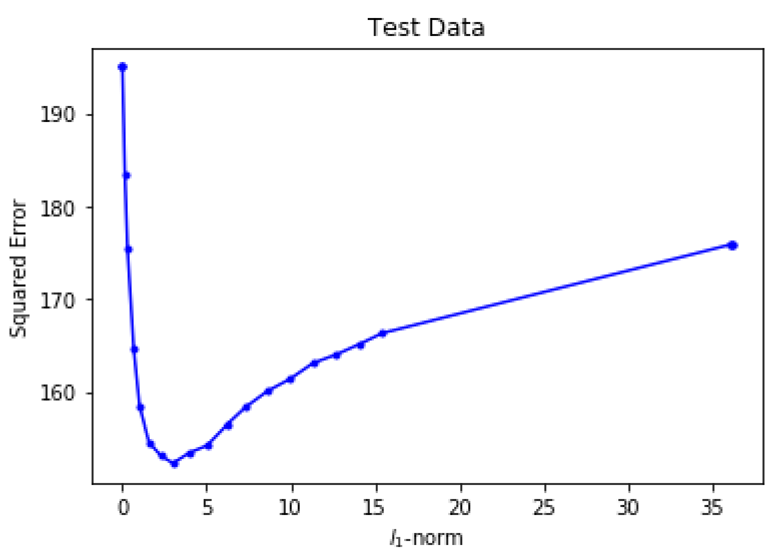
```
plt.figure()
plt.plot(norm, res, 'b.-')
plt.xlabel('$l_1$-norm');
plt.ylabel('Squared Error');
plt.title('Training Data')
plt.show()

plt.figure()
plt.plot(nonz,err, 'b.-')
plt.xlim([0,150])
plt.xlabel('Non Zero Elements')
plt.ylabel('Error Rate')
plt.show()

plt.figure()
plt.plot(norm, resv, 'b.-')
plt.xlabel('$l_1$-norm');
plt.ylabel('Squared Error');
plt.title('Test Data')
plt.show()

plt.figure()
plt.plot(nonz, errv, 'b.-')
plt.xlim([0,150])
plt.xlabel('Non Zero Elements')
plt.ylabel('Error Rate')
plt.show()
```



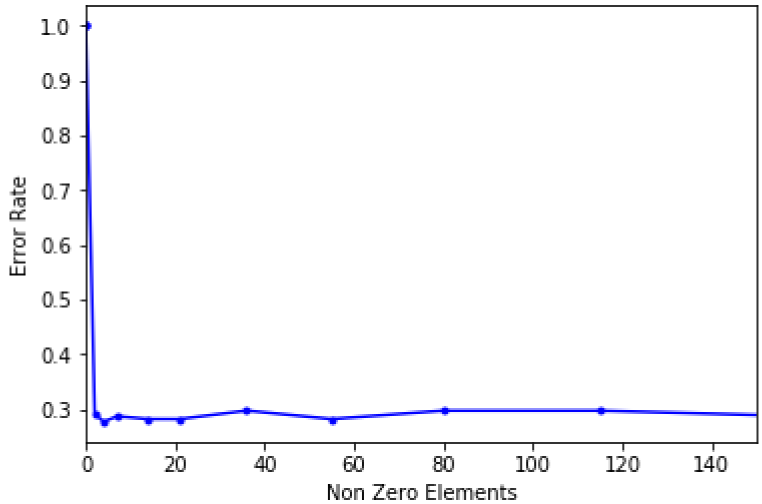
Training Data
Squared Error
l_1-norm



Error Rate
Non Zero Elements



Test Data
Squared Error
l_1-norm



3)

Use 10-fold CV with l_1 regularization

In [6]:
```python
setindices = np.asarray([[0,30],[30,60],[60,90],[90,120],[120,150],
    [150,179],[179,208],[208,237],[237,266],[266,295]])

holdoutindices = np.asarray([[0,1],[1,2],[2,3],[3,4],[4,5],
                             [5,6],[6,7],[7,8],[8,9],[9,0]])

cases = 10
index = np.asarray(range(295))

errv2_l1 = np.zeros(cases)
resv2_l1 = np.zeros(cases)
for j in range(cases):
    v1_ind = index[setindices[holdoutindices[j,0],0]:
                  setindices[holdoutindices[j,0],1]]
    v2_ind = index[setindices[holdoutindices[j,1],0]:
                  setindices[holdoutindices[j,1],1]]
    trn_ind = np.setdiff1d(np.setdiff1d(index, v1_ind), 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, :]

    W = ista_solve_hot(At,bt,lam_vals)

    Bhatv1 = np.sign(Av1@W) # for finding lambda

    errv1 = np.zeros(number)
    for i in range(number):
        errv1[i] = np.mean(Bhatv1[:,i:i+1]!=bv1)
    min_ind = np.argmin(errv1)

    errv2_l1[j] = np.mean(np.sign(Av2@W[:,min_ind:min_ind+1])!=bv2)
    resv2_l1[j] = np.linalg.norm(Av2@W[:,min_ind:min_ind+1]-bv2)**2

err10fold_l1 = np.mean(errv2_l1)
res10fold_l1 = np.mean(resv2_l1)

print("err10fold_l1 =", err10fold_l1)
print("res10fold_l1 =", res10fold_l1)
```

```
err10fold_l1 = 0.31862068965517243
res10fold_l1 = 24.433411185456592
```

Use 10-fold CV with l_2 regularization

In [7]:
```
errv2_l2 = np.zeros(cases)
resv2_l2 = np.zeros(cases)
for j in range(cases):
    v1_ind = index[setindices[holdoutindices[j,0],0]:
                   setindices[holdoutindices[j,0],1]]
    v2_ind = index[setindices[holdoutindices[j,1],0]:
                   setindices[holdoutindices[j,1],1]]
    trn_ind = np.setdiff1d(np.setdiff1d(index, v1_ind), 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, :]

    W2 = np.zeros((At.shape[1],number))

    errv1 = np.zeros(number)
    for i in range(number):
        W2[:, i:i+1] = At.T@np.linalg.inv(At@At.T+lam_vals[i]*np.eye(At.shape[
0]))@bt
        errv1[i] = np.mean(Av1@W2[:, i:i+1]!=bv1)
    min_ind = np.argmin(errv1)

    errv2_l2[j] = np.mean(np.sign(Av2@W2[:,min_ind:min_ind+1])!=bv2)
    resv2_l2[j] = np.linalg.norm(Av2@W2[:,min_ind:min_ind+1]-bv2)**2

err10fold_l2 = np.mean(errv2_l2)
res10fold_l2 = np.mean(resv2_l2)

print("err10fold_l2 =", err10fold_l2)
print("res10fold_l2 =", res10fold_l2)
```

```
err10fold_l2 = 0.3120689655172414
res10fold_l2 = 28.00152732609226
```

In [ ]: