In [23]:
```
## 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']
```

In [24]:
```
# global variables
la_num = 20
lam_vals = np.logspace(-6, 2, num=la_num)
```

In [25]:
```
def ista_solve_hot( A, d, la_array ):
    # ista_solve_hot: Iterative soft-thresholding for multiple values
    # lambda with hot start for each case - the converged value for th
    # value of lambda is used as an initial condition for the current
    # 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, n
            X[:, [i]] = w
            if np.linalg.norm(w - w_old) < tol:
                break
    return X

def ridge_many(A,d,la_array):
    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):
        w = A.T@np.linalg.inv(A@A.T+each_lambda*np.identity(A.shape[0]
        X[:, [i]] = w
    return X
```

In [26]:
```
def find_err_rate(X, y, w):
    err = 0
    y_head = X @ w
    for i in range(len(y)):
        if (y[i] != np.sign(y_head[i])):
            err+=1
    err_rate = err / len(y)
    return err_rate


def findOptLamdaIndx(X, y, W):
    err_rates = np.zeros(la_num)
    for i in range(la_num):
        w = W[:, [i]]
        err_rates[i] = find_err_rate(X, y, w);
    lamda_opt_indx = np.argsort(err_rates)[0]
    return lamda_opt_indx
```

In [37]:
```
##  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

# each row of holdoutindices denotes the partitions that are held out
# 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 loopin
# through the various training, validation, and test partitions

err_rates_LASSO = np.zeros(cases)
sq_error_LASSO = np.zeros(cases)

err_rates_RR = np.zeros(cases)
sq_error_RR = np.zeros(cases)
```

LASSO

In [35]:

```
# 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,setind

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

    # 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))
    # Use training data to learn classifier
    W = ista_solve_hot(At,bt,lam_vals) # W across a range of lamdas

    # Find best lambda value using first validation set
    lamda_opt_indx = findOptLamdaIndx(Av1, bv1, W)
    w_opt = W[:, [lamda_opt_indx]]

    # evaluate performance on second validation set, and accumulate pe
    err_rate = find_err_rate(Av2, bv2, w_opt)
    err_rates_LASSO[j] = err_rate
    sq_error = np.linalg.norm(Av2 @ w_opt - bv2, ord=2)
    sq_error_LASSO[j] = sq_error

print("Average error rate for LASSO is: ", np.average(err_rates_LASSO)
print("Average squred error for LASSO is: ", np.average(sq_error_LASSO
```

```
Average error rate for LASSO is:  0.3001436781609196
Average squred error for LASSO is:  5.129196576172512
```

Ridge Regression

In [38]:
```
# 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,setind

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

    # 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))
    # Use training data to learn classifier
    W = ridge_many(At,bt,lam_vals) # W across a range of lamdas

    # Find best lambda value using first validation set
    lamda_opt_indx = findOptLamdaIndx(Av1, bv1, W)
    w_opt = W[:, [lamda_opt_indx]]

    # evaluate performance on second validation set, and accumulate pe
    err_rate = find_err_rate(Av2, bv2, w_opt)
    err_rates_RR[j] = err_rate
    sq_error = np.linalg.norm(Av2 @ w_opt - bv2, ord=2)
    sq_error_RR[j] = sq_error

print("Average error rate for Ridge Regression is: ", np.average(err_r
print("Average squred error for Ridge Regression is: ", np.average(sq_
```

```
Average error rate for Ridge Regression is:  0.30833333333333335
Average squred error for Ridge Regression is:  4.962352789866199
```