# Assignment 8

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.

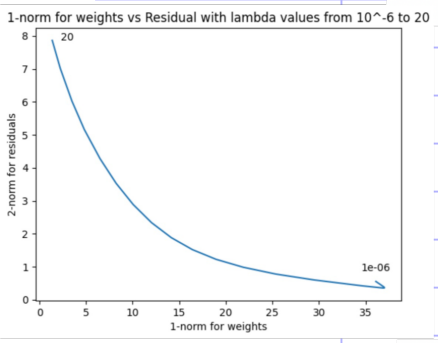
   The goal is to solve the Lasso problem

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

   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 $\lambda$ to explore the tradeoff between data-fitting and sparsity.

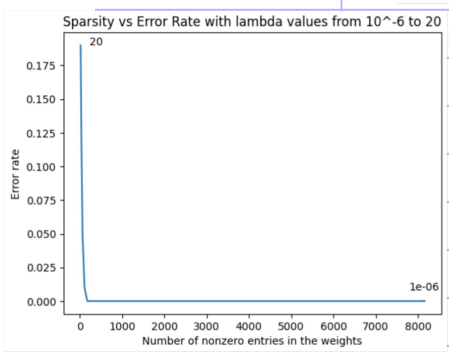
   Scripts that implement iterative soft thresholding via proximal gradient descent to solve the LASSO problem are available. The scripts use a hot start procedure for finding the solution with different values for $\lambda$. The initial guess for the next value of $\lambda$ is the converged solution for the preceding value. This accelerates convergence when subsequent values of $\lambda$ 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 $\lambda$. In other words, create the curve by finding $\boldsymbol{w}^*$ for different $\lambda$, and plotting $\|\boldsymbol{w}^*\|_1$ vs. $\|\boldsymbol{A}\boldsymbol{w}^* - \boldsymbol{d}\|_2$. Experiment with $\lambda$ to find a range that captures the variation from the least-squares solution (small $\lambda$) to the all zeros solution (large $\lambda$). Appropriate values of $\lambda$ may range from $10^{-6}$ to 20, spaced logarithmically. Explain the result.

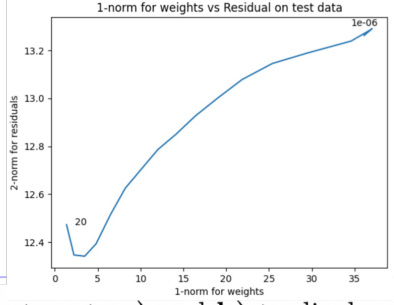



The residual is high with a large lambda because the effect of the regularization step is large. This pulls the weights away from the least squares solution. Instead the weights are smaller and closer to 0, as you can tell from the 1-norm which causes more misclassifications on the training data.

   **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 $\lambda$ 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.




With a higher lambda, we get sparser solutions. However, this means that the weights may be underfitted and thus they predict the training data label worse.
A smaller lambda gives us a more complex model, and we predict the training data well. However, this then means the model may be overfitted.




**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.

From the test data, we start to see the effect of overfitting. Smaller lambdas with higher 1-norm for weights and more non zero entries approximate the training data well but then cause a lot of errors on the test data because they overfit to the training data and cannot handle new data.

**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 $\lambda$ 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 $\lambda$. Note that LASSO and ridge regression will produce different best values for $\lambda$.
- Compute the test error on the final subset of the data for the choice of $\lambda$ 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 ($\lambda$) 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.

$$(A^TA + \lambda I)^{-1}A^Tb = A^T(AA^T + \lambda I)^{-1}b$$

This one will compute faster because we are taking the inverse of the smaller 100x100 matrix.

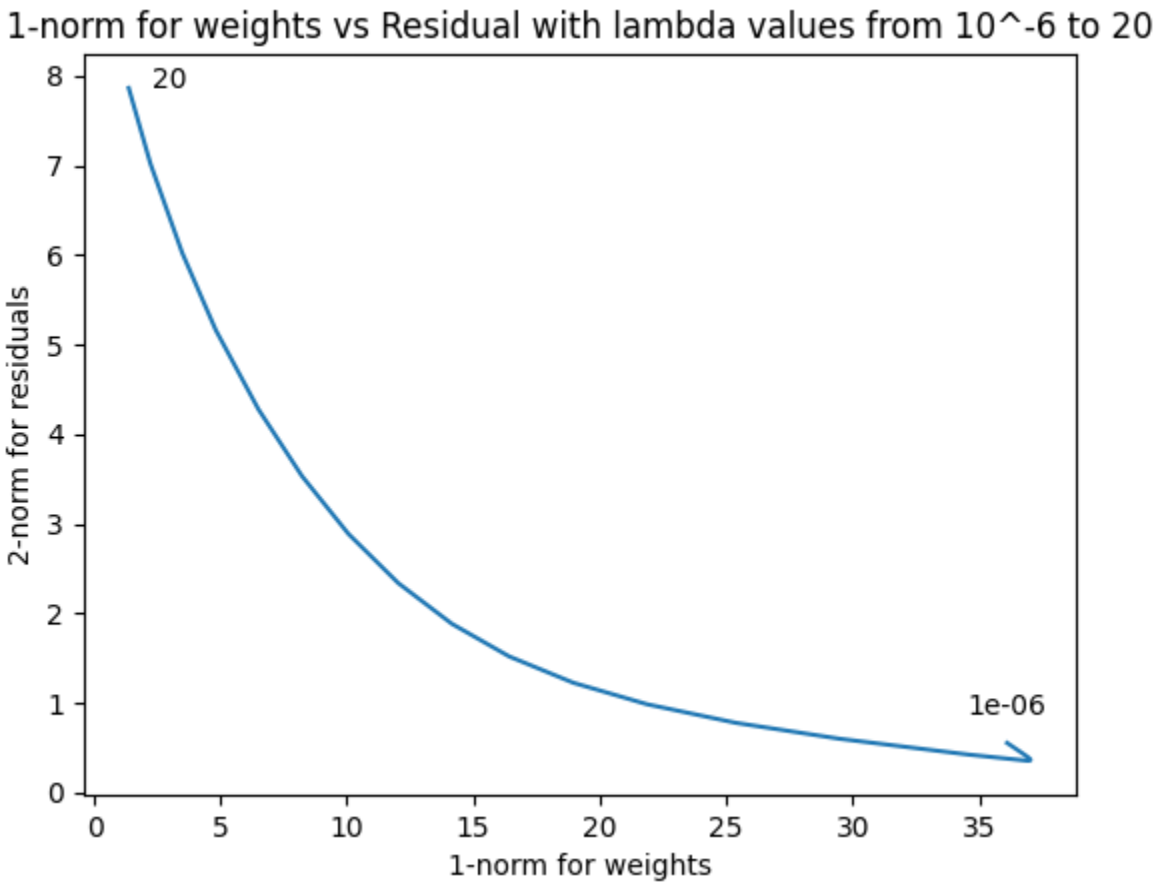
Both squared error and and error rate are slightly worse for ridge regression compared to LASSO. However, especially for misclassification error, the difference is very minute (0.2997 vs 0.3039).
(23.44 vs 26.52)

# Question 1

```
from scipy.io import loadmat
import numpy as np
import math
import matplotlib.pyplot as plt

X = loadmat("BreastCancer.mat")['X']
y = loadmat("BreastCancer.mat")['y']


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


lambda_list = np.logspace(-6, math.log10(20))

X_100 = X[0:100, :]
y_100 = y[0:100]

w_array = ista_solve_hot(X_100, y_100, lambda_list)


#residual = [np.linalg.norm(
    #(np.sign(X_100@w_array[:,i]) - y_100),ord=2) for i in range(50)]
residual = [np.linalg.norm(
    ((X_100@w_array[:,i]).reshape(100,1) - y_100), ord=2) for i in range(50)]
weights = [np.linalg.norm(w_array[:,i], ord=1) for i in range(50)]

print(residual)
print(weights)
```

```
[0.5517470751599132, 0.5436396454039996, 0.5356667729573878, 0.5278257516827031, 0.520113
[36.10905595263041, 36.151745850817264, 36.19371245775491, 36.234962556425, 36.2755214646
```

```
plt.plot(weights, residual)
plt.title("1-norm for weights vs Residual with lambda values from 10^-6 to 20")
plt.xlabel("1-norm for weights")
plt.ylabel("2-norm for residuals")

plt.annotate("1e-06", # text
             (weights[0],residual[0]), # coordinates to position the label
             textcoords="offset points", # how to position the text
             xytext=(0,10), # distance from text to points (x,y)
             ha='center') # horizontal alignment can be left, right or center
plt.annotate("20", # text
             (weights[49],residual[49]), # coordinates to position the label
             textcoords="offset points", # how to position the text
             xytext=(15,0), # distance from text to points (x,y)
             ha='center') # horizontal alignment can be left, right or center

plt.show()
```

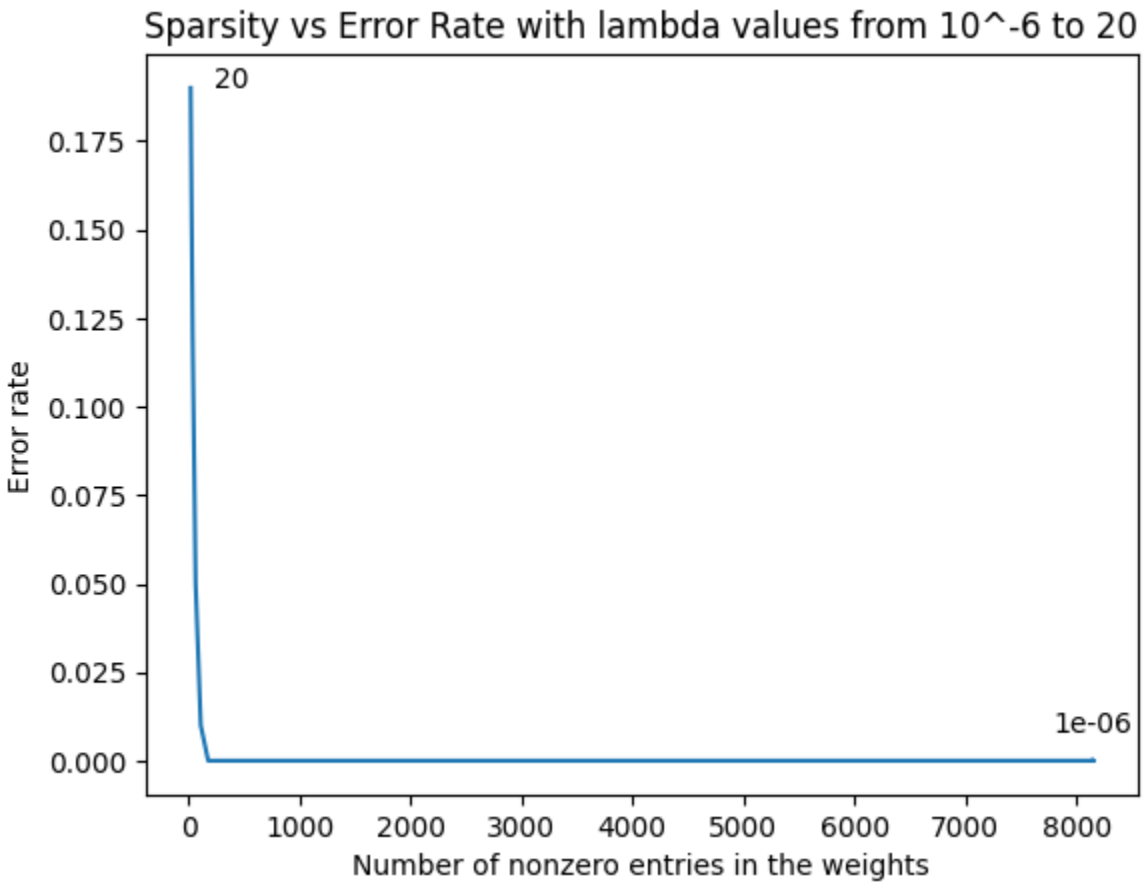



```
def error_rate(X, y, weights):
  err = 0
  y_hat = np.sign(X@weights)
  for i in range(len(y)):
    if (y[i] != y_hat[i]):
      err+=1
  return err/len(y_hat)
```

```
error = [error_rate(X_100, y_100, w_array[:,i]) for i in range(50)]
sparsity = [(abs(w_array[:,i]) > 10**-6).sum() for i in range(50)]

print(error)
print(sparsity)
```

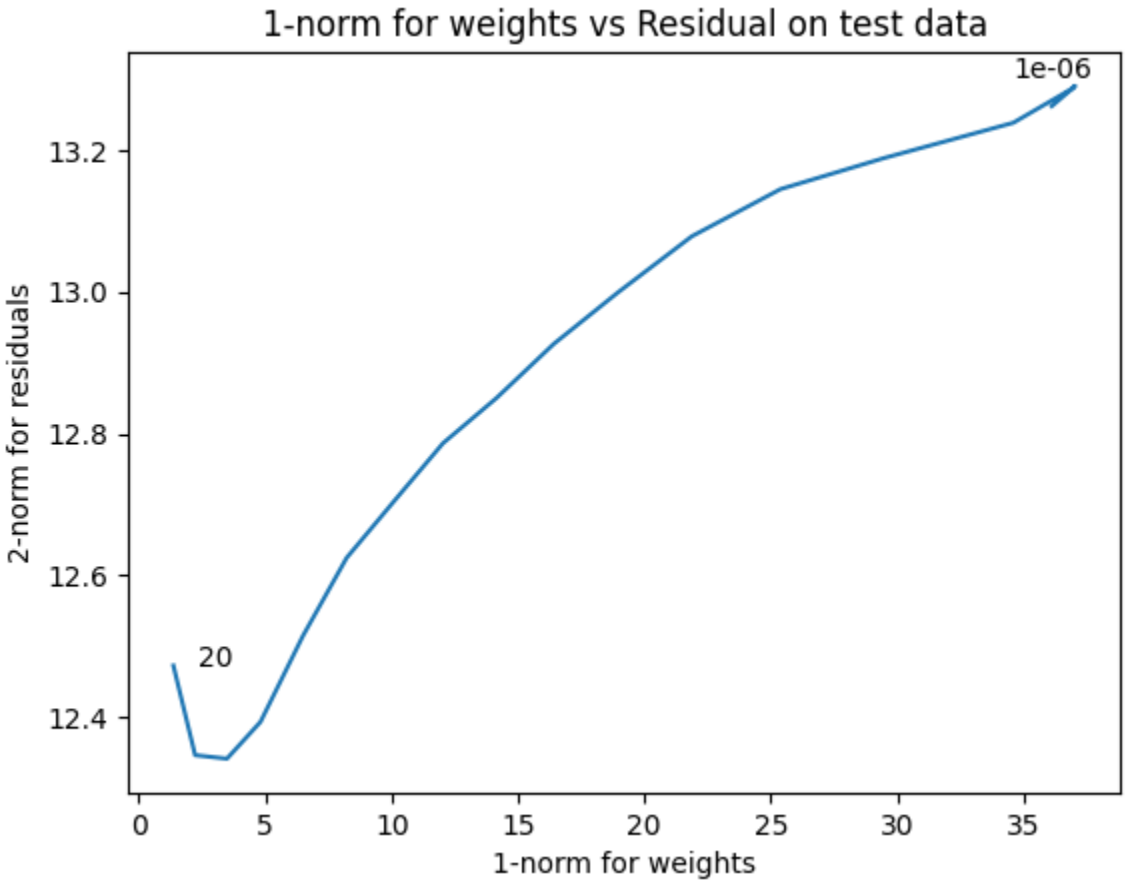
```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
[8137, 8140, 8140, 8139, 8139, 8140, 8141, 8139, 8139, 8139, 8139, 8140, 8140, 8141, 8140
```

```
plt.plot(sparsity, error)
plt.title("Sparsity vs Error Rate with lambda values from 10^-6 to 20")
plt.xlabel("Number of nonzero entries in the weights")
plt.ylabel("Error rate")

plt.annotate(lambda_list[0], # text
             (sparsity[0],error[0]), # coordinates to position the label
             textcoords="offset points", # how to position the text
             xytext=(0,10), # distance from text to points (x,y)
             ha='center') # horizontal alignment can be left, right or center
plt.annotate("20", # text
             (sparsity[49],error[49]), # coordinates to position the label
             textcoords="offset points", # how to position the text
             xytext=(15,0), # distance from text to points (x,y)
             ha='center') # horizontal alignment can be left, right or center

plt.show()
```



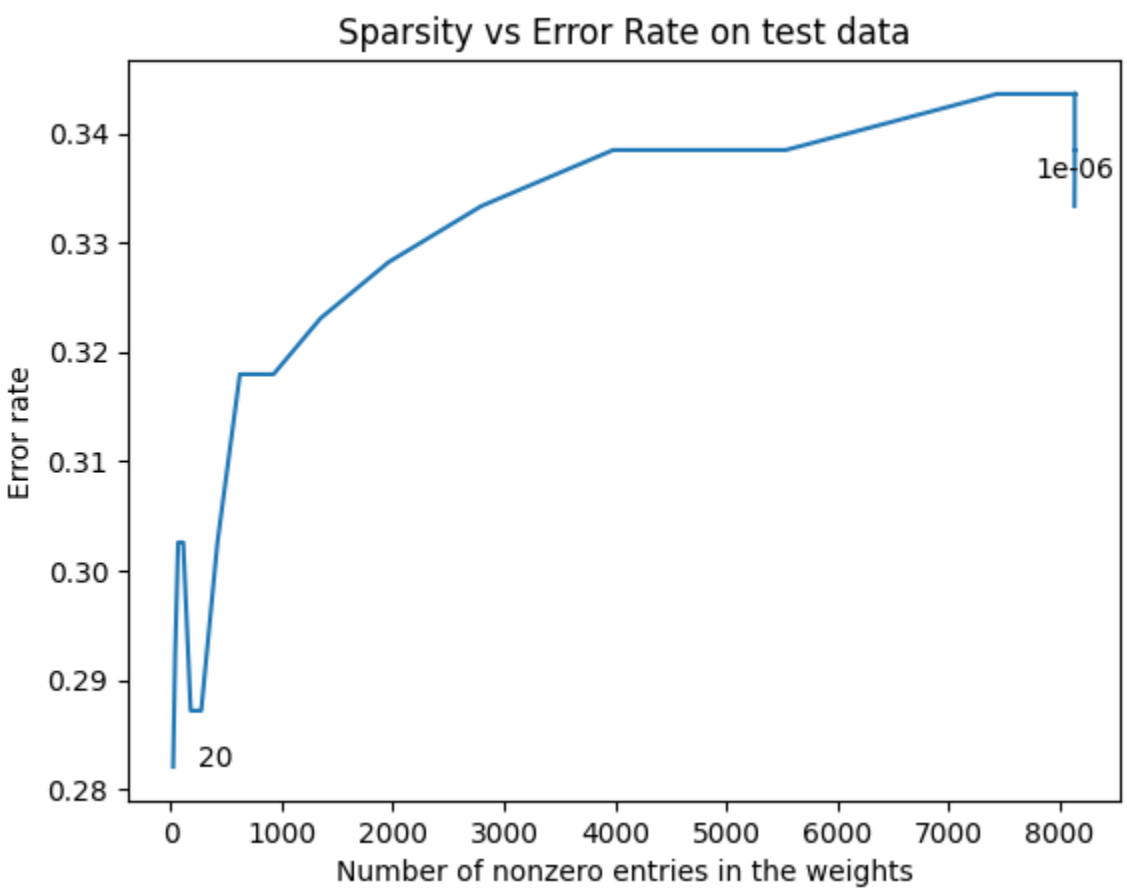
## Part c)

```
X_test = X[100:, :]
y_test = y[100:]

residual_test = [np.linalg.norm(
    ((X_test@w_array[:,i]).reshape(195,1) - y_test),ord=2) for i in range(50)]

plt.plot(weights, residual_test)
plt.title("1-norm for weights vs Residual on test data")
plt.xlabel("1-norm for weights")
plt.ylabel("2-norm for residuals")
plt.annotate("1e-06", # text
             (weights[0],residual_test[0]), # coordinates to position the label
             textcoords="offset points", # how to position the text
             xytext=(0,10), # distance from text to points (x,y)
             ha='center') # horizontal alignment can be left, right or center
plt.annotate("20", # text
             (weights[49],residual_test[49]), # coordinates to position the label
             textcoords="offset points", # how to position the text
             xytext=(15,0), # distance from text to points (x,y)
             ha='center') # horizontal alignment can be left, right or center
plt.show()
```




```
error_test = [error_rate(X_test, y_test, w_array[:,i]) for i in range(50)]
```

```
plt.plot(sparsity, error_test)
plt.title("Sparsity vs Error Rate on test data")
plt.xlabel("Number of nonzero entries in the weights")
plt.ylabel("Error rate")

plt.annotate(lambda_list[0], # text
             (sparsity[0],error_test[0]), # coordinates to position the label
             textcoords="offset points", # how to position the text
             xytext=(0,10), # distance from text to points (x,y)
             ha='center') # horizontal alignment can be left, right or center
plt.annotate("20", # text
             (sparsity[49],error_test[49]), # coordinates to position the label
             textcoords="offset points", # how to position the text
             xytext=(15,0), # distance from text to points (x,y)
             ha='center') # horizontal alignment can be left, right or center

plt.show()
```



# Question 2

```
## 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']


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

def ridge(A, d, la_array):
    dim = A.shape[0]
    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.transpose()@np.linalg.inv(
            A@A.transpose() + each_lambda*np.identity(dim))@d
        X[:, i:i+1] = w
    return X

def error_rate(X, y, weights):
    err = 0
    y_hat = np.sign(X@weights)
    for i in range(len(y_hat)):
      if (y[i] != y_hat[i]):
        err+=1
    return err/len(y_hat)
```

```
##  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
lambda_list = np.logspace(-6, math.log10(20))

misclass_L = np.zeros(cases)
misclass_2 = np.zeros(cases)
squared_error_L = np.zeros(cases)
squared_error_2 = np.zeros(cases)

# 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))

    # H
```

```
    # Use training data to learn classifier
    W_L = ista_solve_hot(At,bt,lambda_list) #LASSO
    W_2 = ridge(At, bt, lambda_list) #ridge regression

    # Find best lambda value using first validation set
    error_L = [error_rate(Av1, bv1, W_L[:,i]) for i in range(50)]
    error_2 = [error_rate(Av1, bv1, W_2[:,i]) for i in range(50)]
    # index of best weights
    best_L = np.argsort(error_L)[0]
    best_2 = np.argsort(error_2)[0]

    # performance on second validation set
    best_error_L = error_rate(Av2, bv2, W_L[:,best_L])
    best_error_2 = error_rate(Av2, bv2, W_2[:,best_2])
    misclass_L[j] = best_error_L
    misclass_2[j] = best_error_2

    squared_L = np.linalg.norm(
        ((Av2@W_L[:,best_L]).reshape(len(bv2),1) - bv2),ord=2)**2
    squared_2 = np.linalg.norm(
        ((Av2@W_2[:,best_2]).reshape(len(bv2),1) - bv2),ord=2)**2
    squared_error_L[j] = squared_L
    squared_error_2[j] = squared_2
```

```
 30 30 235
 30 30 235
 30 30 235
 30 30 235
 30 29 236
 29 29 237
 29 29 237
 29 30 236
```

```
print("For LASSO:")
print("Average misclassification error:", np.average(misclass_L))
print("Average squared error:", np.average(squared_error_L))
print("\n")
print("For ridge regression:")
print("Average misclassification error:", np.average(misclass_2))
print("Average squared error:", np.average(squared_error_2))
```

```
For LASSO:
Average misclassification error: 0.2997126436781609
Average squared error: 23.440022647942314


For ridge regression:
Average misclassification error: 0.3038793103448276
Average squared error: 26.52090910565953
```