CPSC-340 Machine Learning 2012

Homework # 5

NAME:_____

Signature:_____

STD. NUM: _____

1 Regularized linear regression

1.1 Standardize the data.

Download the prostate cancer dataset from the course website. In this prostate cancer study 9 variables – including age, log weight, log cancer volume, etc. – were measured for 97 patients. We will now construct a model to predict the 9th variable a linear combination of the other 8. A description of this dataset appears in the textbook of Hastie et al, freely available on the course website:

"*The data for this example come from a study by Stamey et al. (1989) that examined the correlation between the level of prostate specific antigen (PSA) and a number of clinical measures, in 97 men who were about to receive a radical prostatectomy. The goal is to predict the log of PSA (lpsa) from a number of measurements including log cancer volume (lcavol), log prostate weight lweight, age, log of benign prostatic hyperplasia amount lbph, seminal vesicle invasion svi, log of capsular penetration lcp, Gleason score gleason, and percent of Gleason scores 4 or 5 pgg45.* "

1. First load the data and split it into a response vector (`y`) and a matrix of attributes (`X`),

```
X = np.loadtxt('prostate.data', skiprows=1)
y = X[:,-1]
X = X[:,0:-1]
```

2. Choose the first 50 patients as the training data. The remaining patients will be the test data.

```
ytrain, ytest = y[0:50], y[50:]
Xtrain, Xtest = X[0:50], X[50:]
```

3. Set both variables to have zero mean and standardize the input variables to have unit variance.

```
Xbar = np.mean(Xtrain, axis=0)
Xstd = np.std(Xtrain, axis=0)
ybar = np.mean(ytrain)
ytrain = ytrain - ybar
Xtrain = (Xtrain - Xbar) / Xstd
```

Note: here we are using the "broadcasting" feature of numpy, which allows summation of a vector and a matrix. For example:

```
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> c
```

```
array([7, 8, 9])
>>> a+c
array([[ 8, 10, 12],
       [11, 13, 15]])
```

You will have to be careful when you do this in numpy in order to make the shapes match up as they should. For more details on how to do this properly look up "numpy broadcasting".

Important detail: In this step we are learning the bias θ_0. We will need these exact terms (`Xbar, Xstd, ybar`) for later when we do predictions. Mathematically, what we are saying is that the bias term will be computed separately as follows:

$$\theta_0 = \bar{y} - \bar{\mathbf{x}}^T \widehat{\boldsymbol{\theta}}$$

where $\bar{y}$ is the mean of the elements of the **training data** vector $\mathbf{y}$ and $\bar{\mathbf{x}}^T$ is the vector of 8 means for the input attributes. Note that in this case the 8-dimensional parameter vector $\widehat{\boldsymbol{\theta}}$ includes all the parameters other than the bias term that have been learned with either ridge or lasso. That is, we first learn $\widehat{\boldsymbol{\theta}}$ using standardized data and then proceed to learn θ_0.

When we encounter a new input $\mathbf{x}^\star$ in the test set, we need to standardize it before making a prediction. The actual prediction should be:

$$\widehat{y} = \bar{y} + \sum_{j=1}^{8} \frac{x_j^\star - \bar{x}_j}{\sigma_j} \widehat{\theta}_j$$

where $\bar{x}_j$ and σ_j are the mean and standard deviation of the j-th attribute **obtained from the training data.**

One reason for standardizing the inputs is that we want them to be comparable. Had we had an input much bigger than the other, we would have wanted to apply a different regularizer to it. By standardizing the inputs first, we only need a single scalar regularization coefficient δ^2.

1.2 Ridge regression

We will now construct a model using ridge regression to predict the 9th variable as a linear combination of the other 8.

1. Write code for ridge regression starting from the following skeleton:

   ```
   def ridge(X, y, d2):
       ???
       return theta
   ```

   Compute the ridge regression solutions for a range of regularizers (δ^2). Plot the values of each $\boldsymbol{\theta}$ in the y-axis against δ^2 in the x-axis. This set of plotted values is known as a regularization path. Your plot should look like Figure 1. *Hand in your version of this plot, along with the code you used to generate it.*

2. For each computed value of $\boldsymbol{\theta}$, compute the train and test error. Remember, you will have to standardize your test data with the same means and standard deviations as above (`Xbar, Xstd, ybar`) before you can make a prediction and compute your test error. In other words, to make a prediction do:

   ```
   yhat = ybar + numpy.dot((Xtest - Xbar) / Xstd, theta)
   ```

   Choose a value of δ^2 using cross-validation. What is this value? Show all your intermediate cross-validation steps and the criterion you used to choose δ^2. Plot the train and test errors as a function of δ^2. Your plot should look like Figure 2. *Hand in your version of this plot, along with the code used to generate it.*

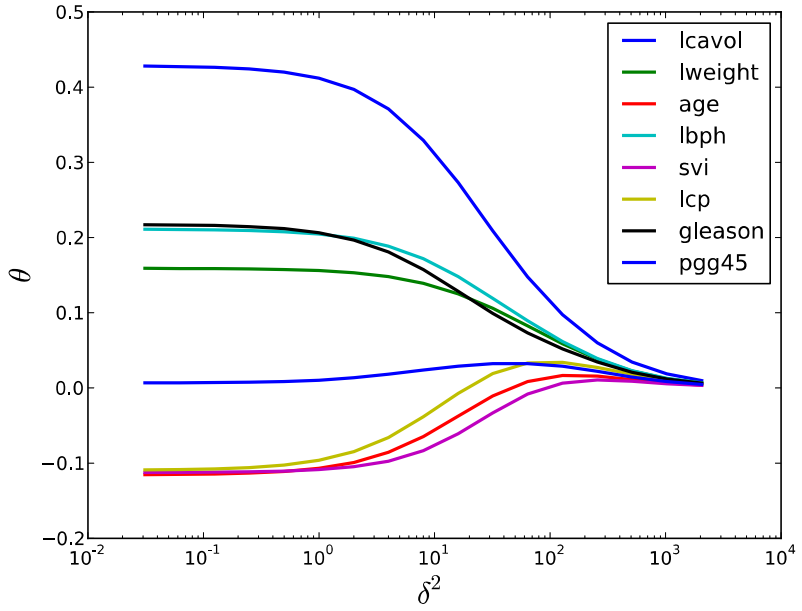

Figure 1: Regularization path for ridge regression.

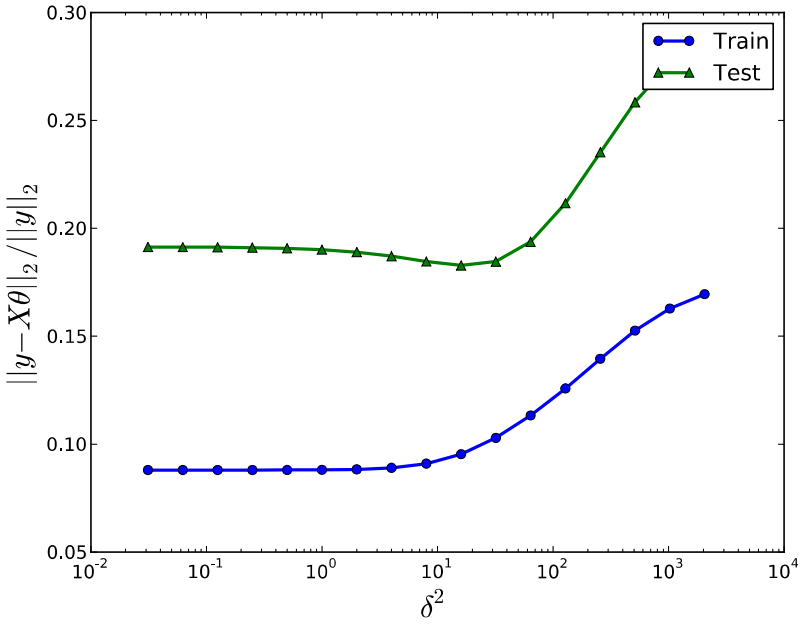

Figure 2: Relative error of the ridge estimator against regularization parameter δ^2.

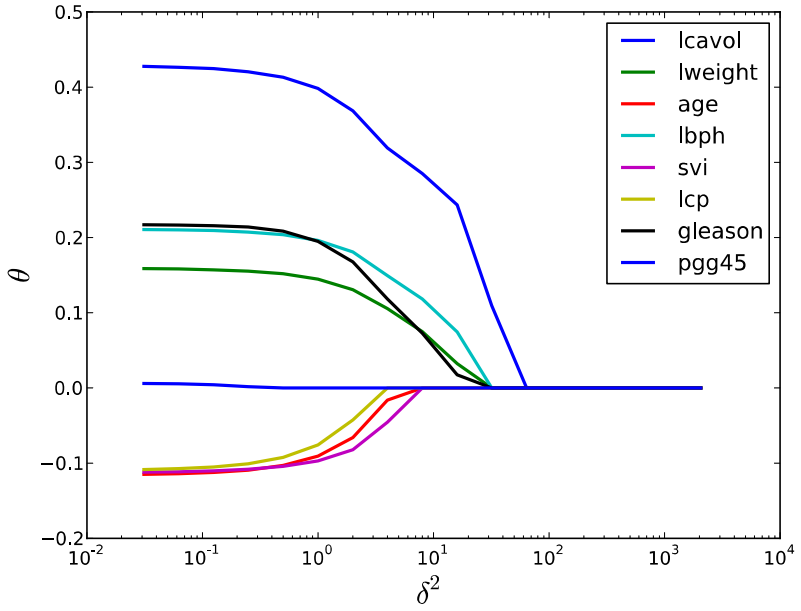

Figure 3: Regularization path for lasso.

1.3 Lasso

We will now implement the Lasso and try this code out on the prostate cancer data.

1. Implement the coordinate descent (aka "shooting" method) for solving Lasso. Pseudo-code for this solver is given in the slides. You should start with the following skeleton code:

```
def lasso(X, y, d2):
    theta_ = np.zeros(k)
    theta = ridge(X, y, d2)
    while np.sum(np.abs(theta-theta_)) > 1e-5:
        theta_ = theta.copy()
        ???
    return theta
```

2. Find the solutions and generate the two plots above, but now using this new Lasso solver. Your two plots should look like Figures 3 and 4. Once again choose δ^2 by cross-validation and write down all your steps. What is the optimal value of δ^2? Which parameters (and hence inputs) are active for this choice of regularization parameter? That is, which are the non-zero thetas?

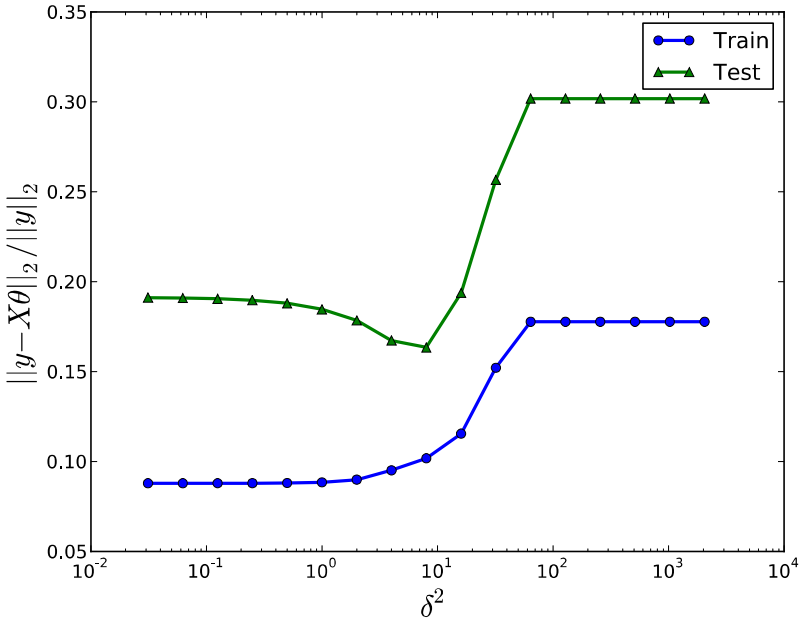

Figure 4: Relative error of the lasso estimator against regularization parameter δ^2.

2 Twitter sentiment classification

In this question, you will be given data consisting of a large number of tweets and be asked to build a classifier which will determine the sentiment of any new tweets we give you as either positive or negative.

In particular, the data we give you will contain one million tweets and approximately ten thousand binary features. The data will be a sparse matrix (i.e. most of its elements will be zero) created using methods from the package `scipy.sparse`. Once you have the datafile `tweets.mtx` (via a link at the end of this document) you can load it using the following commands:

```
import scipy.io
data = scipy.io.mmread('tweets.mtx').tocsr()
```

This just loads the data and converts it into Compressed Sparse Row (CSR) format. This format is just a more efficient way to access the data, but for more details see the `scipy.sparse` documentation (again, linked to at the end of this document).

The format of the data is also pretty basic: each row coincides with one tweet where the first column is the label of that tweet (i.e. positive sentiment or negative) and the remaining columns are the binary features. We can split this up into our standard input/label matrices via:

```
X = data[:,1:]
y = data[:,0]
```

You may also want to treat `y` as a standard binary vector, which we can do via:

```
y = np.array(y.todense()).flatten()
```

However, while `y` is small enough that this should be fine, **do not do this** to `X`! A fully dense `X` will probably take up more memory than your computer can handle.

At this point we should probably also mention what you can do with sparse matrices. These objects are treated as matrices, so if we wanted to take the dot-product between `y` and `X` we could write this as just