Linear Regression - Programming Assignment

NAME:_____

Signature:_____

_____

1 Regularized linear regression

1.1 Data standardization

The goal of standardization is to make the d input attributes of the data comparable. If we were predicting the price or real estate in Vancouver using as input attributes the number of rooms and the price of the property 3 years ago ($d = 2$), then we would run into trouble because the number of rooms is typically in the range 1 to 4, while the price 3 years ago is probably in the range 500,000 to 5,000,000. We need to place these two attributes on the same scale so that applying the same regularizer to their weights makes sense.

To achieve this, we standardize the input data by ensuring it has zero mean and unit variance. We can do this by computing

$$x_{ij} \quad = \quad \frac{x_{ij} - \overline{x}_j}{\sigma_j}, \tag{1}$$

where $\overline{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$ is the empirical mean of the j'th attribute, and $\sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \overline{x}_j)^2$ is the empirical variance. Here, $i = 1, \ldots, n$ denotes the index over the data and $j = 1, \ldots, d$ is the index over input attributes.

The subtraction of the mean is not necessary for datasets that are already known to be zero mean, such as acoustic time series. Likewise, for gray-scale images we know that each attribute (pixel) has the same range of possible values, so we can skip the variance normalization. If the attributes do not have the same range of values, then the variance rescaling can prove to be very useful. For example if the first attribute is the number of people who entered a building, the second attribute the temperature of the building, and the third attribute the number of rooms in the building, then these three numbers are likely not to be in the same scale. Variance rescaling places them on a reasonable comparative scale.

The parameters $\overline{x}_j$ and σ_j^2 should be considered part of the model. So at test time, we should center with respect to $\overline{x}_j$ and scale with respect to σ_j, rather than centering and scaling the test set with respect to the mean and standard deviation of the test set. To see why, imagine the test set consisted of a single example; then scaling by the test set standard deviation, instead of the value derived from the training set, would clearly be wrong, since the standard deviation of a single data point is not defined. All this will be made more clear in the following section.

1.2 Load and 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 the training step, we will learn the bias θ_0 separately. We do this because it makes not sense to apply regularization to the bias θ_0. Recall that the purpose of regularization is to get rid of unwanted input attributes.

We will need the terms (`Xbar, Xstd, ybar`) 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 ridge regression. 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:

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

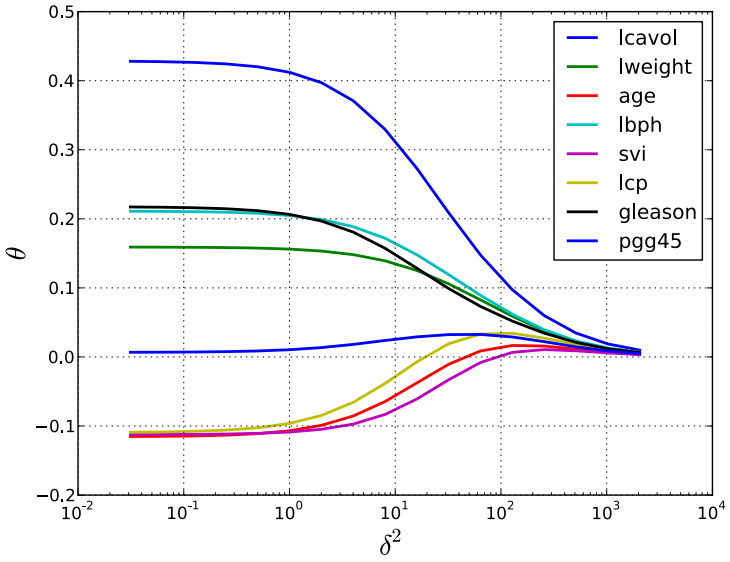

Figure 1: Regularization path for ridge regression.

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.

1.3 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)
```

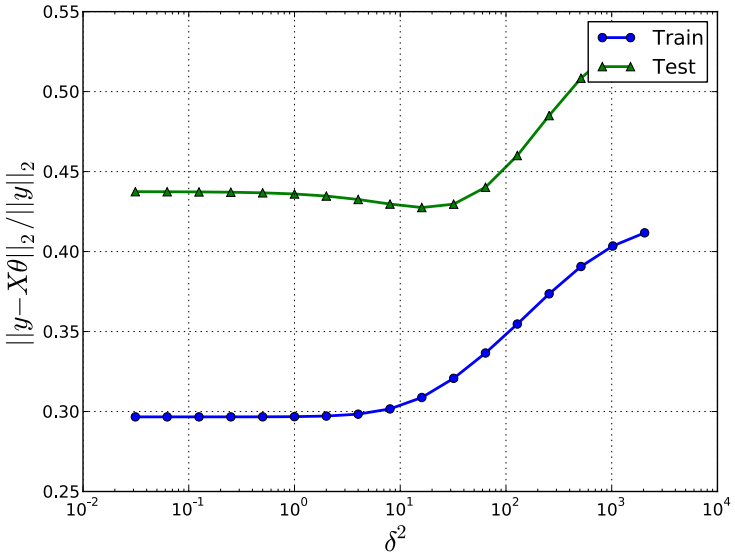

Figure 2: Relative error of the ridge estimator against regularization parameter δ^2.

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.*

Submission

SubmissionLink:
https://drive.google.com/drive/folders/1BkhuZ7KuF8sln2bFDvWo2XvdUKTl0TNr?usp=sharing

Please zip all the source codes & documentation under 1 zip file with following file format: <Firstname>_<Lastname>_Week8-RidgeRegression.zip

The zip file should be encrypted by a password, then please send this password to email: quang.tran@aioz.io