Advanced Topics in High-Performance Computing

Assignment 1, October 2017

Kamyar Nazeri
Student ID: 100633486

1 Introduction.

In this assignment we train a linear regression model to predict house prices using housing data. We will be using California housing dataset from StatLib. The dataset consists of 20,640 average house values as target variables and 8 features: *average income, housing average, age, average rooms, average bedrooms, population, average occupation, latitude*, and *longitude*.
We employ Ridge Regression (*Tikhonov* regularization) to find the best estimate of the true value of the housing price. We start with a linear model and try to find the best regularization hyper-parameter using cross-validation and plot the corresponding train and test errors. We will compare these results with models using higher degree polynomial of features and finally suggest the best model for predicting housing price for our dataset.

2 Procedure.

We will be using Normal Equation to minimize the least square error and find the best estimate of the true value of the housing price:

$$\underset{\theta}{\operatorname{argmin}} \left(\|\mathbf{y} - \mathbf{X}\theta\|_2^2 + \delta^2 \|\theta\|_2^2 \right)$$

With $\mathbf{X}$ being the design matrix with each row as a data point and each column representing a feature. The first step is to whiten our data; Data whitening is a decorrelation transformation that transforms our features into a set of new features with identity covariance (uncorrelated with unit variances):

$$\mathbf{X}^T_{whitened} = (\mathbf{X}^T\mathbf{X})^{-\frac{1}{2}}\mathbf{X}^T$$

The python code to whiten the data is as follows:

```
X_whiten = sc.linalg.inv(sc.linalg.sqrtm(X.T.dot(X))).dot(X.T).T
```

Next, we add the intercept column to our design matrix to represent the bias:

$$\mathbf{X}_{intercept} = [\mathbf{1} \mid \mathbf{x_1} \mid \mathbf{x_2} \mid ... \mid \mathbf{x_n}] \ \in \ \mathbb{R}^{m\times(n+1)}$$

With m being the number of training examples and $\mathbf{x_1}$ ... $\mathbf{x_n}$ column vectors representing each feature in our dataset.
The python code to add the intercept column is as follows:

```
X_intercept = np.c_[[1] * X_whitened.shape[0], X_whitened]
```

We apply normal equation on the design matrix to find the parameters estimating our data:

$$\theta = (\mathbf{X}^T\mathbf{X} + \delta^2\mathbf{I}_{(n+1)})^{-1}\mathbf{X}^T\mathbf{y}$$

With δ^2 being the regularization term and $\mathbf{I}_{(n+1)}$ an identity matrix. We need to make sure that we do not regularize the bias term, meaning in the equation above we replace $\mathbf{I}_{(n+1)}$ with $\tilde{\mathbf{I}}_{(n+1)}$ with the following structure:

$$\tilde{\mathbf{I}}_{(n+1)} = \operatorname{diag}\{0, 1, 1, ..., 1\} \ \in \ \mathbb{R}^{(n+1)\times(n+1)}$$

In other words, the first entry of the matrix $\tilde{\mathbf{I}}_{(n+1)}$ is **zero**:

$$\theta = \left(\mathbf{X}^T\mathbf{X} + \delta^2 \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}\right)^{-1} \mathbf{X}^T\mathbf{y}$$

The python function to solve the Ridge Regression for θ is as follows:

```
import numpy as np
import scipy as sc

def estimate_theta(X, y, d2):
        m, n = X.shape
        reg = d2 * np.eye(n + 1)     # regularization
        reg[0, 0] = 0                # not regularizing the bias

        # solve the normal equation
        theta = np.linalg.inv(X.T.dot(X) + reg).dot(X.T.dot(y))
        return theta
```

3 Ridge Regression Results

3.1 Regularization Paths:

Our first task is to plot values of θ for different values of δ^2. The range of regularization term is from 10^{-7} to 10^{-1}:

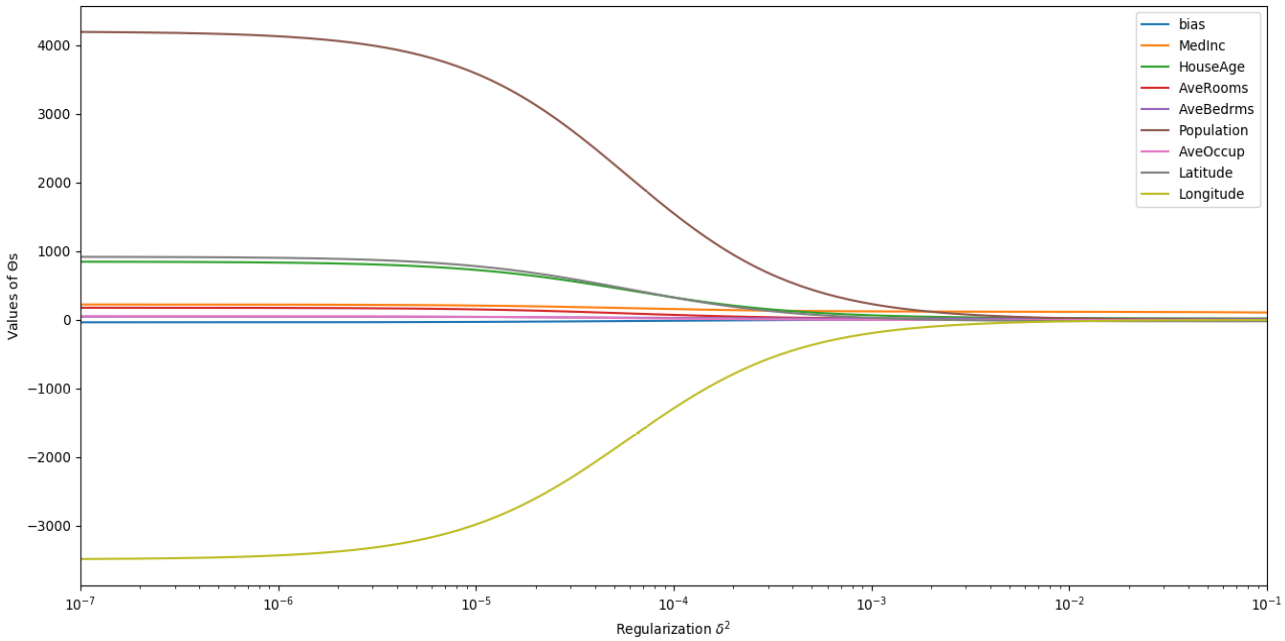

Figure 1: Ridge coefficients as a function of the regularization.

The plot shows that as the regularization term increases, the coefficients are heavily penalized hence become smaller; As a result, the hypothesis will be approximately equal to the bias term.

3.2 Error Analysis:

To calculate error of the Ridge Regression model, we ue Mean-Square-Error, with $y^{(i)}$ being the i^{th} label and $\mathbf{x}^{(i)}$ being the i^{th} data-point:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \theta^T \mathbf{x}^{(i)})^2$$

We split the data into training (80%) and test(20%) sets (80/20 is indeed a good starting point) and calculate MSE for a range of regularization hyper-parameter from 10^{-7} to 10^3:

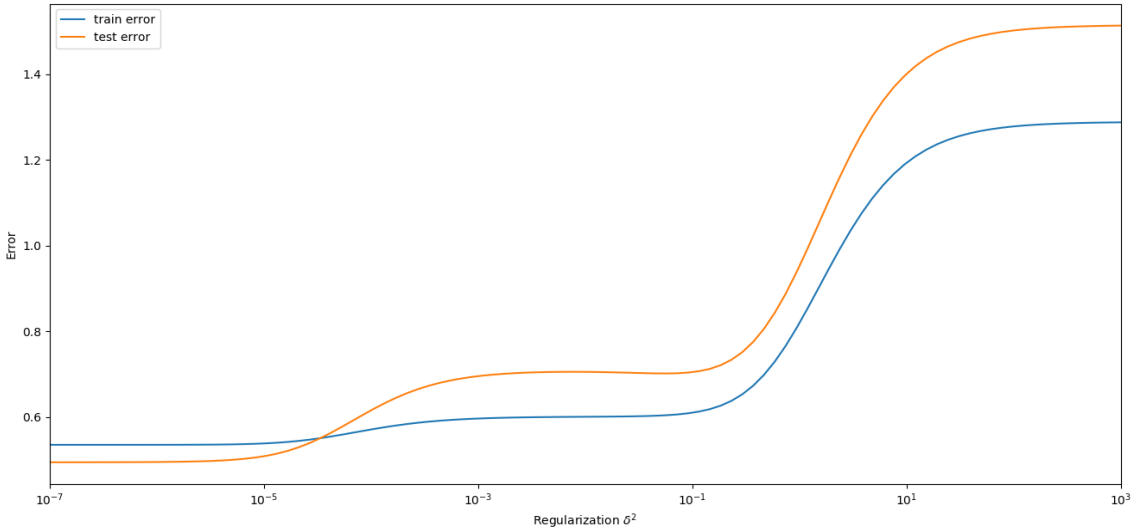

Figure 2: Comparison of test and train error for different values of regularization.

To avoid any element of bias/patterns in the split datasets we shuffle data before training; This can break any possible correlation among the features and the MSE error becomes:

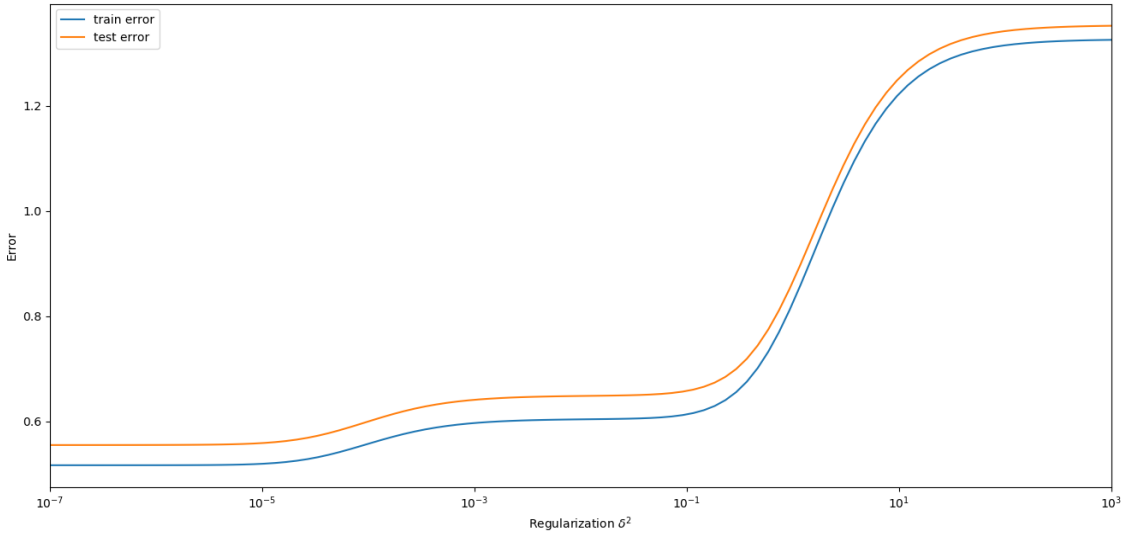

Figure 3: Comparison of test and train error for different values of regularization with shuffling

We use *scikit* built-in functions to shuffle data:

```
X, y = sklearn.utils.shuffle(X, y)
```

To decrease the correlation in data even more and come up with the best cross validation we tried 5-fold cross validation on the shuffled data and took the average from the folds. The test error becomes smoother in the following plot:

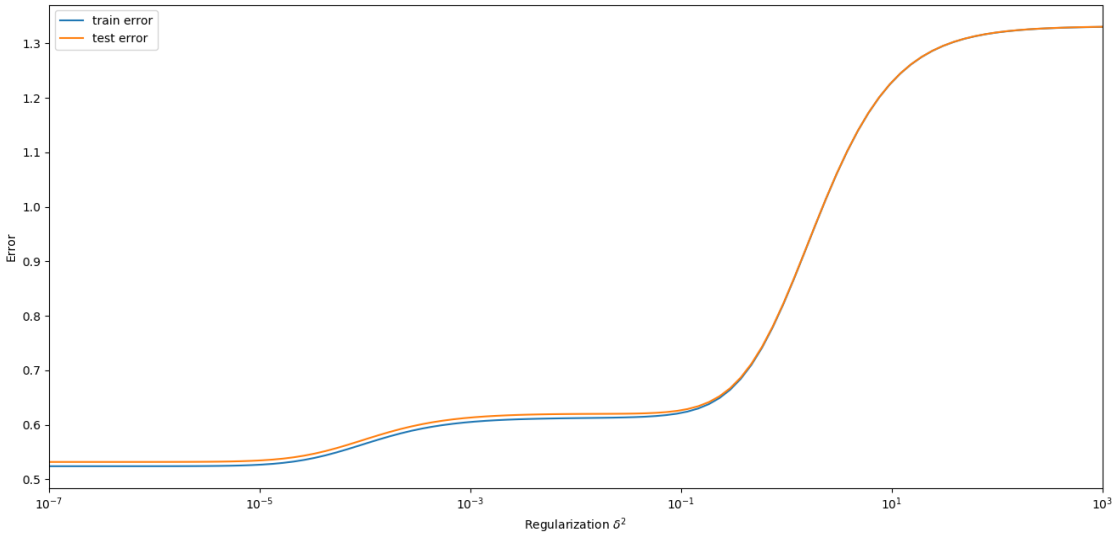

Figure 4: Comparison of test and train error for different values of regularization with shuffling and 5-fold cross validation

The errors graph in *figures 2-4* show that as we increase the value of regularization δ^2, the model becomes high bias and eventually underfits the data, this model is very simple that the hypothesis is almost equal to the bias term.
The best regularization hyper-parameter in our model is any value less than 10^{-5} where the test and train errors are at their minimum. Any regularization value greater than that would have a negative effect on the error. This model would work just fine even without regularization ($\delta^2 = 0$) however, to generalize our model to polynomial regression we chose to include the regularization value.

3.3 Best Model:

To select the best model for our regression problem, we also tried degree 2 and 3 polynomial of features; choosing higher degree polynomials (n > 3) would result in numerical instability when finding the inverse of a matrix in the normal equation with a whitened data.
The following are the same regression model on the *Population* feature in our dataset using first order, degree 2 and 3 polynomials:

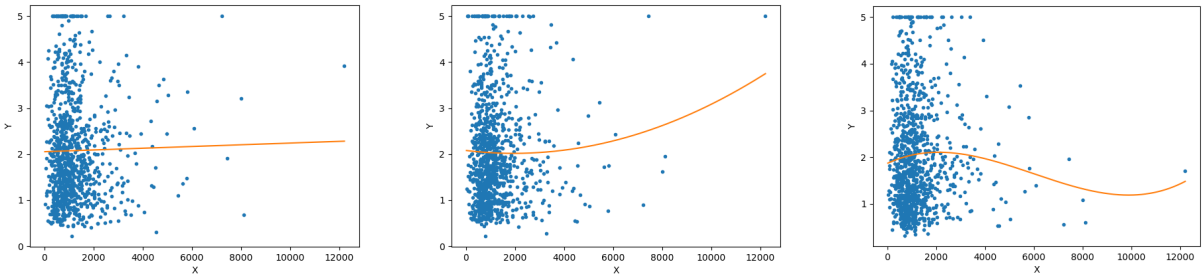

Figure 5: Price of the house as a function of Population. left: 1st order, middle: 2nd degree, right: 3rd degree

The 3rd degree polynomial feature might be able to fit the data much better by smoothing the regression curve; however, we need to regularize it well enough to prevent overfitting. The following plot shows the 3rd degree polynomial test/train MSE for different values of regularization:

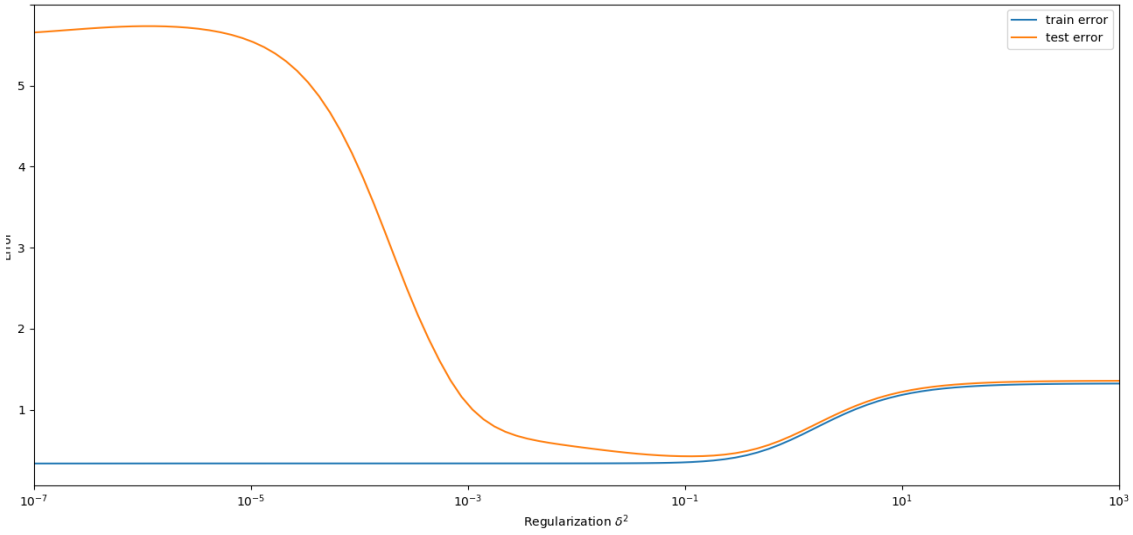

Figure 6: Comparison of test and train error for different values of regularization with 3rd degree polynomial features

The errors graph in *figure 6* suggests that although with higher degree polynomial the error is relatively smaller compared with the first order features, the system is more prone to overfitting. Regularization becomes necessary with polynomial regression, otherwise the test error becomes large and the model becomes high variance. The best regularization value in the model above (3rd degree) is almost 10^{-1}.
This simple experiment shows that as long as we regularize, higher degree models could outperform linear models in terms of mean-square error.

4 Conclusion.

Linear regression attempts to draw a straight line that best minimizes the residual sum of squares between the data points and the predictions. However, this model does not generalize well and is prone to overfitting, especially with higher degree polynomial of features(*figure 6*, $\delta^2 < 10^{-1}$). With Ridge Regression we penalize a least squares by shrinking the values of the coefficients; Using regularization, the slope of the regression line tend to be more stable and the variance smaller (*figure 6*, $\delta^2 = 10^{-1}$).

Considering the values of θ (*figure 1*) one can conclude that the most important feature in our regression model is *Population*. This is based on the idea that when a range of independent features are all on the same scale, the most important features should have the highest coefficients (θ) in the model. In our model, features with the most impact on the housing price are: *Population, Latitude, HouseAge, MedInc, AveRooms, AveOccup, AveBedrms*, and *Longitude* respectively.