CS 433 Assignment 1: Linear Regression

Eric Stevens

Introduction:

In this assignment I have used **Python 2.7.13**. This assignment also relies on the **numpy** library. To run the script type '**python ass1.py**'. The single python file includes everything for the assignment aside from this write up. They python file is organized with a set of helper functions at the top, some of which sove entire parts of the assignment. The part of the file that runs when you run the script is at the bottom. It will output simple results of running the data.

Parts 1 & 2:

This part of the assignment involves loading data from a text file into two matrices and then running an operation on the two matrices to generate a third matrix. To do this, we rely on two helper functions. The **load_matrix()** function takes an input of a filename and returns the **X** and **Y** matrices. **X** is a matrix of feature data except for the last column which if filled with ones. **Y** is a single column matrix filled with target data. From there the **X** and **Y** matrices are fed as parameters into the **weight_vector()** function. This function will return **W**, the weight vector. The weight vector is the solution to:

$$W = (X^T X)^{-1} X^T Y$$

Optimal:

```
W =  [[ -1.01137046e-01]
      [  4.58935299e-02]
      [ -2.73038670e-03]
      [  3.07201340e+00]
      [ -1.72254072e+01]
      [  3.71125235e+00]
      [  7.15862492e-03]
      [ -1.59900210e+00]
      [  3.73623375e-01]
      [ -1.57564197e-02]
      [ -1.02417703e+00]
      [  9.69321451e-03]
      [ -5.85969273e-01]
      [  3.95843212e+01]]
```

Parts 3:

In order to calculate the sum squared error (SSE) the parameters calculated in parts one and two are fed into the helper function **sum_sq_err()**. This function performs the matrix operations needed to calculate the SSE and returns it as **E** where:

$$E = (Y - XW)^T (Y - XW)$$

Following the calculation of the SSE of the weight vector applied to the training data the variables **Xt** and **Yt** are loaded with the testing data using the **load_matrix()** function. Then **Xt**, **Yt**, and **W** are fed into the **sum_sq_err()** function and return the variable **Et**, the SSE when applying the W calculated from the training data and applying it to the test data.

Training Data With Dummy SSE: 9561.1912899766994
Test Data With Dummy SSE: 1675.2309659480629

Parts 4:

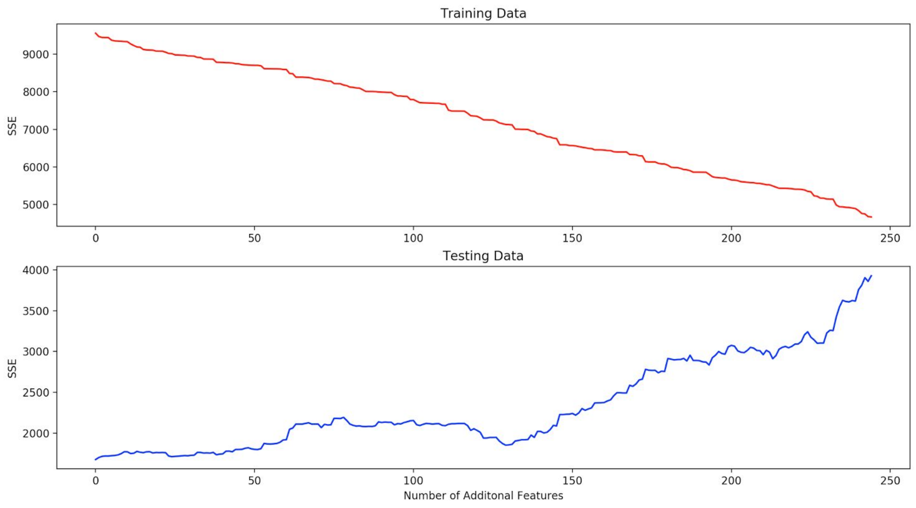
The objective in this section is to repeat the same process as above using training data that does not include a dummy variable. Our **X** from above is fed into a function called **no_dummy()** which returns the matrix without its dummy column to a variable **X_nd**. This process is repeated with **Xt** to return the testing data with no dummy variable to **Xt_nd**. **W_nd** is then calculated by feeding **X_nd** and **Y** into the **weight_vector()** function. From there we can calculate the SSEs of the training and test data. The SSE of the training data is **E_nd**, and is calculated by feeding **X_nd, Y,** and **W_nd** into the function **sum_sq_err()**. The SSE of the testing data**, Et_nd**, is calculated by feeding **Xt_nd, Yt,** and **W** into the **sum_sq_err()** function.

Training Data Without Dummy SSE: 10598.0572458
Test Data Without Dummy SSE: 1797.625625

For both the training and the testing data, the lack of a dummy variable causes the SSE to go up. This is because the lack of a dummy variable forces the hyper plane to cross the origin instead of having an offset in the error domain.

Parts 5:

In this the goal is to see how adding junk features affects the value of the calculated weight vector. For this purpose a function, **feature_relationships()**, has been written. **feature_realationship()** returns three lists. The first list **num_feat** is the number of features added and will serve as the x axis on the plots. The second list **List_SSE** is a list of the SSE values of the training data corresponding to the number of features added in **num_feat**. The third list **List_SSEt** is a list of the SSE values of the testing data corresponding to the number of features added in **num_feat**. **feature_relationships()** works by opening new instances of the training and test data and looping **W** and **Wt** calculations after continually concatenating random columns in different ranges onto **X** and **Xt**. The following graph is the result of running this operation.



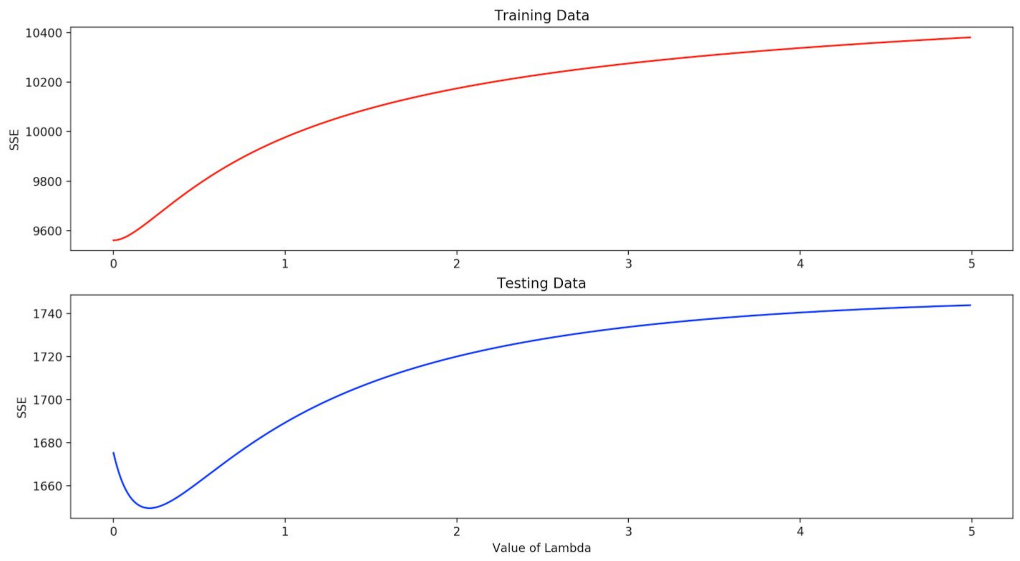
As can be see from the results adding random extra features makes the training data SSE decrease. This is overfitting. The random features in can be used as extra degrees of freedom when fitting the weight vector to the training data. However, when The calculated weight matrix is applied to the testing data in causes a large increase in the SSE. This is because the random data never actually had correlation with the target values.

Parts 6:

For part 6, the **weight_vector()** function is replaced by the **weight_vector_variant()**. Its inputs and outputs operate the same was as **weight_vector()** except there is an extra input parameter, **lamb**. Return value **W** is calculated as follows.

$$W = (X^T X + \lambda I)^{-1} X^T Y$$

In order to obtain an array of SSE values based varying values of lambda the **weight_vector_variant()** function is embedded in a for loop within the function **lambda_relationships()**. This function takes an array of lambda values called **lambda_list** as a parameter and returns a list of SSEs for the training data and a list of SSEs for the testing data; these are **List_SSE** and **List_SSEt** respectively.



Parts 7:

For the training data the SSE seems to continually increase with the value of lambda. The SSE for the testing data is more complicated. The SSE drops as the value of λ increases initially and bottoms out when λ is around 0.21.

Parts 8:

The objective function has a regularization factor which causes a different result based on the magnitudes of the weight vector. Since our weight vector is a vector with now complex values, the provided equation can be rewritten as follows.

$$\sum_{i=1}^{N}(y_i - w^T x_i)^2 + \lambda \sum_{j=1}^{M} w_j^2$$

Since the training data will mold the weight vector to itself, it is obvious that as λ increases the the value of the regularization vector and thus the error will increase. However, when the we are using the same weight vector on a different set of data the magnitudes of the weight vector becomes influential. The value of λ allows modifies the amount of influence that the weight coefficients have on the the error result. Essentially, the addition of the regularization factor encourages **W** to utilize values that are small. If large values of lambda are introduced in an effort to fit training data more precisely than the objective function we are trying to minimize will increase dramatically.