Machine Learning Homework #2 README

Joel Kemp, mrjoelkemp@gmail.com

March 11, 2011

File List

The implementation of the regression experiment was achieved using Matlab 7.10.0 (R2010a). Since Matlab's fundamental datatype is a matrix, no external libraries were necessary. All matrix manipulations were performed using built-in Matlab functions.

The following files are included in the homework submission:

1. LinearRegression.m: The main function of the program. This function opens the passed Comma Separated Value (CSV) files and trains a linear regression model of d-parameters that is based and evaluated on the training data. L2 Regularization can also be used by supplying a lambda value. The function prints the training set and testing set mean-squared errors, separated by a comma.

2. regressionWeights.m: This function creates the model (set of weights) of d parameters, where d is the passed degree parameter, about the training data and targets.

3. fitDataToPolynomial.m: This function uses the weights (model) generated by the system to fit passed data to the regression polynomial. The function returns the resulting targets of the data points evaluated on the polynomial.

4. evaluateRegressionModel.m: This function computes the sum-squared-error of a model and optionally performs L2-regularization on the model and returns the error value.

Code Compilation

Matlab code is compilable into executables and standalone applications; however, the process involves purchasing a license for the Matlab Compiler. This creates some difficulties for distributing applications; however, if the end-user has Matlab (or the Matlab Component Runtime discussed later in this section) installed, then the regression system can be run by navigating (in a terminal or command prompt) to the top-level directory of the this assignment's submission folder and typing matlab -nodesktop. The -nodesktop option prevents the entire Matlab java-based environment from loading, only initializing a command window for executing commands.

Within the Matlab Command Window, the end-user can interact with the Linear Regression application by typing:

"LinearRegression training.csv testing.csv d lambda". Of course, the user should supply numeric values for *d* and *lambda*.

Sample interaction with the Matlab Command Window and the Linear Regression application can be seen below:

```
LinearRegression linear-regression.train.csv linear-regression.test.csv 13 0
0.154164, 0.080834

LinearRegression linear-regression.train.csv linear-regression.test.csv 13 0.000035
0.200823, 0.206812

LinearRegression linear-regression.train.csv linear-regression.test.csv 13 0.000035
0.200823, 0.206812
```

If the end-user does not have Matlab installed, then execution of the Linear Regression system can still be achieved by installing the Matlab Component Runtime. This runtime must be of the same version of Matlab that was used to create the code. The runtime for Matlab 7.10.0 has been uploaded to a file sharing site MegaUpload.com and can be downloaded with this link: http://www.megaupload.com/?d=XROGZ7PD

If troubles arise for the end-user, please email the developer at mrjoelkemp@gmail.com

High-level Description of the Task

The task achieved by this system is to generate a regression model (set of optimal weights that define the regression polynomial) of a user-specified number of parameters (polynomial degree) about a set of training and testing data. The training data consists of observed data and target values that will be used to generate the regression polynomial (i.e., model the behavior/trend of the data). The testing data consists of new observations and targets that will be used to test the predictive accuracy of the regression model.

The Training and Testing data files consist of observed data and target values listed in two columns in two separate Comma Separated Value (CSV) files.

The purpose of this polynomial generation is to infer a target value for a new observation (data input outside of the training set; i.e, data not used to train the model) based on the modeling of previously observed data and its target values. Once the model has been generated, the system should compute the Mean Squared Error for both the training and testing sets. These errors are used to judge the accuracy/usefulness of the model.

Within a particular range of polynomial degrees, the generated model suffers from overfitting the training data. This results in a wild fluctuation of the resulting mean squared error for the training and testing sets. This is typically characterized by very large weights (coefficients) in the regression polynomial.

In an attempt to reduce the effects of polynomial overfitting in the model, we employ L2 regularization which utilizes a user-defined lambda value to penalize large weights. With an adequate lambda value, we can greatly improve the model in the situation of an overfitting polynomial degree.

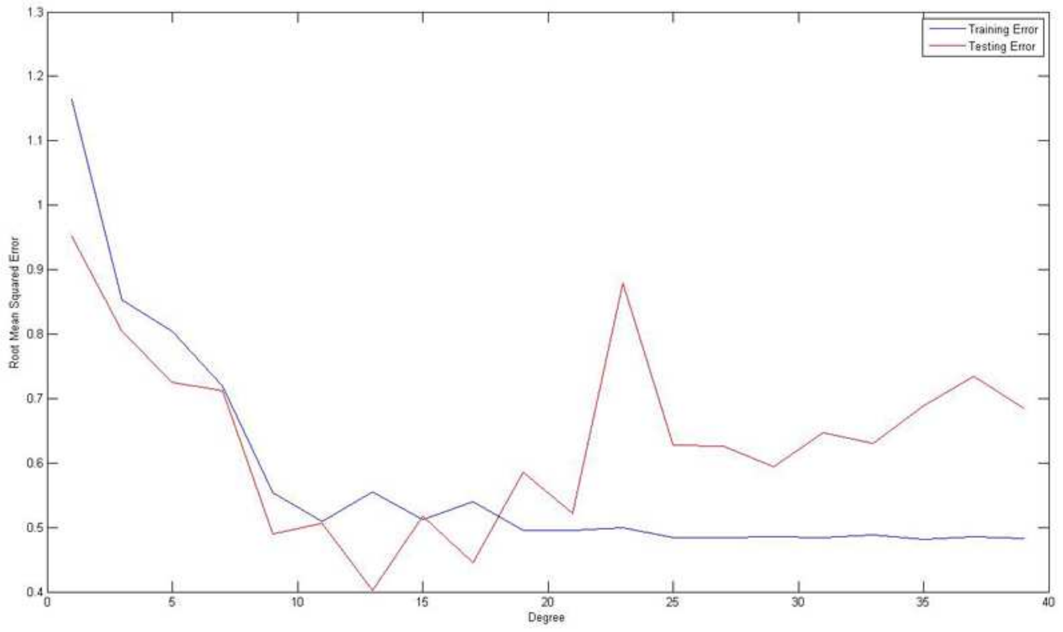
This L2 regularization is realized as an extra term in the computation of the mean squared error for the model.

Experiment with Varying Degree

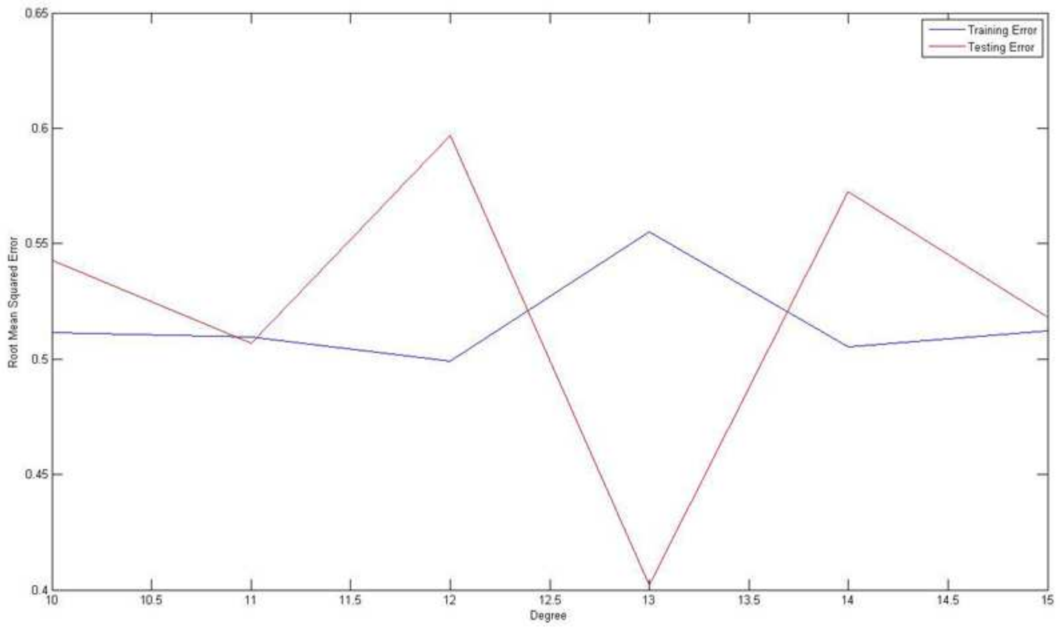
In the table below, we can observe the Training and Testing errors for 20 values of the degree d with no regularization. For the experiment, we have chosen values of d from 1 to 39, in steps of 2.

d	Training	Testing
1	0.677431	0.452259
3	0.364047	0.322739
5	0.323823	0.262304
7	0.258486	0.254042
9	0.152913	0.119617
11	0.129880	0.128428
13	0.154164	0.080834
15	0.131195	0.134133
17	0.145646	0.099397
19	0.122979	0.171850
21	0.122621	0.136416
23	0.124666	0.387015
25	0.117507	0.196487
27	0.117450	0.195554
29	0.117850	0.176212
31	0.117072	0.208796
33	0.119291	0.198533
35	0.115751	0.236722
37	0.117774	0.269894
39	0.116458	0.233875

To better observe the approximate value of degree d where over-fitting occurs on the training data, we present the following plot of the Root-Mean-Squared (RMS) Training and Testing Errors versus the varying Degree, d.



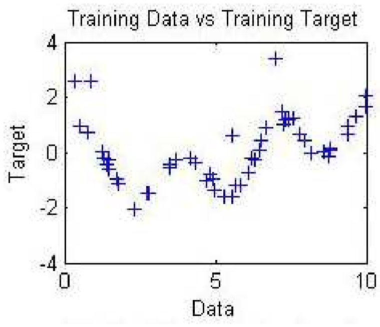
If we focus on the range where the testing set error plummets toward zero, namely when $10 < d < 15$, we obtain the following photo:



From the above figure, we can conclude that over-fitting of the data occurs when $d = 13$.

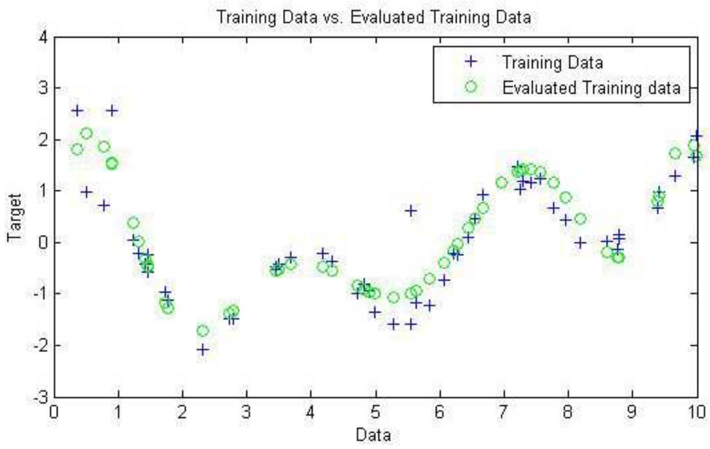
Experiment Discussion

The original data plot of the observations (data) vs the target values can be seen from the following figure:

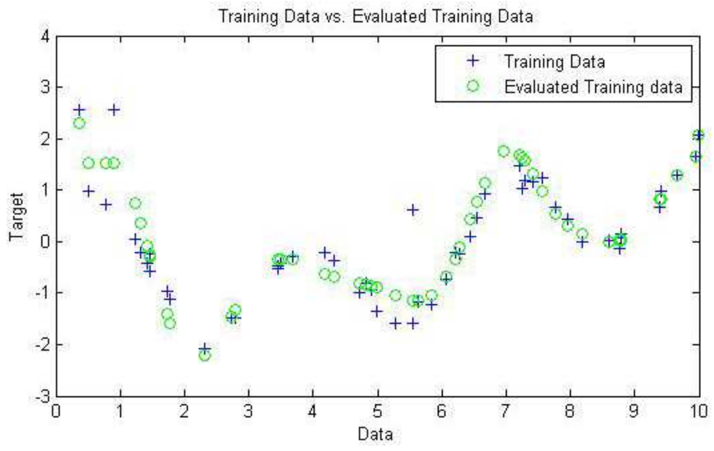


The underlying function generating the data resembles a variation of $\cos(x)$, as the curve starts at 1 and periodically oscillates with noise.

From the analysis of the first figure, namely, the Training and Testing errors under a varying polynomial degree d, we can see that for increasing degrees, the polynomial generates results at least as good as the results when approximately $d = 9$. This point is further strengthened by observing the almost unnoticeable change in the fitted curve from the following two figures.



The above figure shows the Training and Testing errors for a degree-9 polynomial being used to fit the training data.



The above figure shows the Training and Testing errors for a degree-39 polynomial being used to fit the training data. Note that the resulting polynomial looks very similar (almost identical) to the polynomial of degree 9.

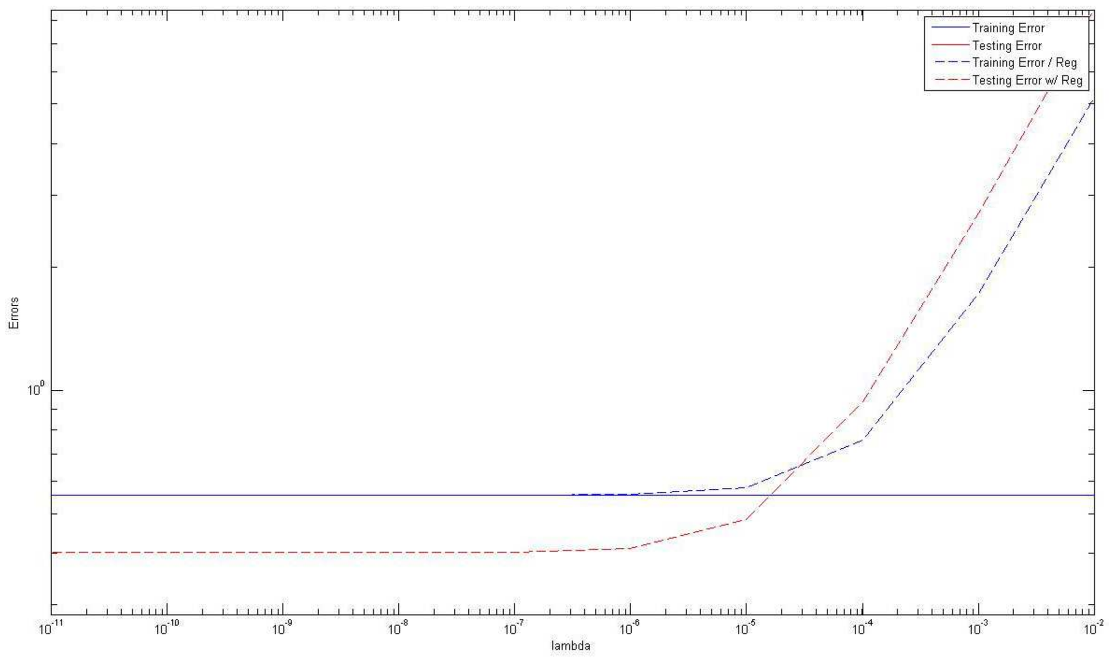
Regularization Experiment

The next phase of the experiment involves varying lambda values and analyzing the resulting training and testing set errors (namely, RMS errors) to find the optimal lambda value that attempts to minimize the effects of overfitting. For this experiment, we have chosen lambdas within the range: [0.00001 0.0001] in steps of 0.000001. Of course, this range is narrowed down from empirical studies of lambda values very slowly approaching 1. Only a subset of these values will be given with the resulting training and testing RMS errors as seen below:

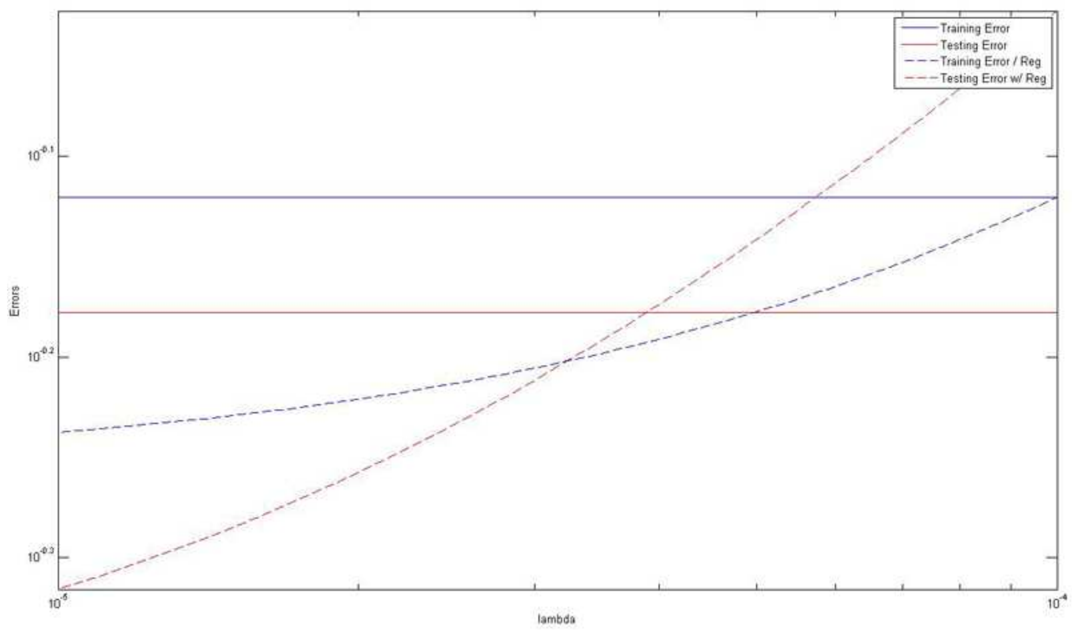
λ	Training	Testing
0.000030	0.194157	0.188815
0.000031	0.195490	0.192414
0.000032	0.196823	0.196014
0.000033	0.198156	0.199613
0.000034	0.199489	0.203213
0.000035	0.200823	0.206812
0.000036	0.202156	0.210411
0.000037	0.203489	0.214011
0.000038	0.204822	0.217610
0.000039	0.206155	0.221209
0.000040	0.207488	0.224809
0.000041	0.208821	0.228408
0.000042	0.210154	0.232007
0.000043	0.211487	0.235607
0.000044	0.212820	0.239206
0.000045	0.214154	0.242806

From the above table, it can be seen that when $\lambda = 0.000035$, the resulting training and testing set RMS errors are very close, with a error difference of $|ERMS_{Train} - ERMS_{Test}| = 0.0059$. In turn, this result means that $\lambda = 0.000035$ is the optimal lambda value for reducing the effects of model overfitting with a 13-degree polynomial.

This result can be approximated from viewing the effects of varying lambda on the 13-degree polynomial in the figure below:



Focusing in on the region where the two regularized errors seemingly intersect, we obtain the following figure:



Of course, in a logarithmic-space plot, visually ascertaining the optimal lambda value is rather difficult. But the plot shows the closeness of the testing and training set errors (with regularization) that can be attributed to the optimal lambda value observed previously.

Regularization Discussion

The experiment yielded quite surprising results. The potential range of "good" lambda values was unknown before-hand. By overfitting the model, the resulting polynomial coefficients became much larger than imagined. However, in retrospect, it could have been somewhat expected that the norm of a 13-degree (14 weight coefficients including w_0) polynomial's weight vector would be quite large. To potentially nullify the effects of the huge weights, very miniscule values of lambda were necessary.

Initial tests involved observing the influence of lambda when the parameter assumed values larger than 1. This proved to be very unhelpful in reducing the effect of the weights – in fact, the weights were amplified with $\lambda > 2$. This rightly follows from the lambda term $\frac{\lambda}{2}||\vec{w}||^2$ in the L2 regularization.

Hence, significantly smaller values needed to be supplied for lambda. The settled range (presented in the previous section) was obtained by experimenting with exponentially decreasing values of lambda.

In summary, the regularization experiment greatly clarified the detrimental effects of model overfitting on the usefulness of a model and its related error evaluation.