Assignment 1 - Ridge Regression and Model Selection

Course: *Advanced Machine Learning*

October 29, 2018

1 Teammembers and Contributions

- **Bernd Menia**, *01316129*: Report, Part Programming
- **Maximilian Samsinger**, *01115383*: Main Programming

The difference in our contributions is because Bernd just only started working with Machine Learning, while Maximilian has already prior knowledge in the subject. Thus we worked together, but the main programming work for this assignment was done by Maximilian. Apart from that due to the size of the code we will only list the important parts and explain them in detail with the results.

2 Prerequisites and Creation of Subsets

Our data points are denoted by N which we set to 30. There was no mathematical reason for this, but by testing our code with multiple values for N we found 30 to be a fitting number for this assignment. The nonlinear function $f(x) = \sin(2 * \pi * x)$ can be seen in Figure 1. We chose a sine function because it is simplistic and let's us demonstrate the linear regression easily. The blue curve corresponds to the initial sine function whereas the orange curve depicts the calculated values including the randomized errors.

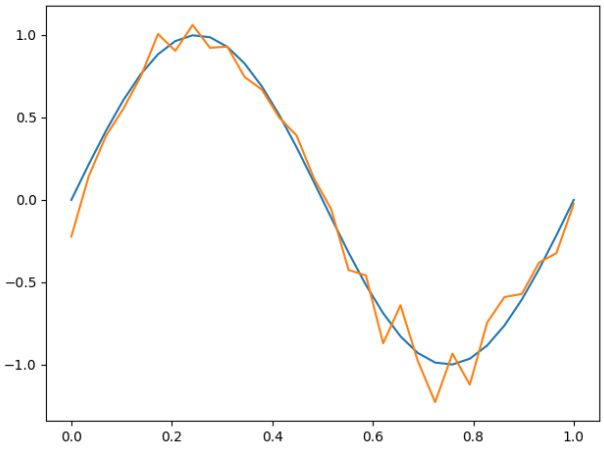

Figure 1: Initial sine function
Only $\sin(2 * \pi * x)$
Including randomized errors

The next step is to divide our data points into training- and test sets. We chose to use $\frac{N}{K}, where K = 5$ of all data points rounded down. There is no rule how the points have to be distributed. This also means that we can theoretically distribute the points like we want. Let's assume we have a vector x that holds all x-values for the data-

and test points. We could take the first 2 elements of the vector for the trainings set and the rest for the test set. However this would most likely be very bad because if the elements are ordered we have a biased test set. Taking the first 2 elements of $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]$ would give $[1, 2]$ which is heavily biased towards lower x-values, or rather the left side of the function. In that case the probability is high that the error function produces a high result.

A simple solution would be to order the vector and then re-do the explained procedure. However even by doing this there still is the possibility that the points are not evenly distributed and biased. Randomizing x could still produce the result $[1, 2, 5, 3, 12, 7, 8, 4, 10, 9, 2, 6, 11]$ in which case the trainings points would again be $[1, 2]$.

There is another solution that fixes this problem without much trouble. First we have to order our vector x and then divide it into $\frac{N}{K}$ subsets. For example if $N = 12$ and $K = 3$ then we would split x into $\frac{12}{3} = 4$ as equally big parts as possible. For x this would result in $[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]$. The second step is to randomize each bucket. This could for example result in $[[3, 1, 2], [5, 4, 6], [7, 8, 9], [11, 10, 12]]$. Finally we can take 1 element out of each bucket as our test points and the rest as our trainings points. Doing this we can guarantee that there is no bias towards any part of the curve because the buckets are evenly distributed over the x-values.

3 K-Fold Cross-Validation

Doing this is also convenient because we can use the subsets for our K-Fold Cross-Validation. Since we have $\frac{N}{K} = I$ subsets we iterate I times over all elements in the following way: We start at $i = 0$ up to $i = I - 1$ and take the $i - th$ subset as our test points. All other subsets correspond to the training points for each iteration respectively (i.e. one fold). So taking our example from before, $[[3, 1, 2], [5, 4, 6], [7, 8, 9], [11, 10, 12]]$ we can produce 4 different folds. If $i = 0$ then we would take $[3, 1, 2]$ as our test points and $[5, 4, 6], [7, 8, 9], [11, 10, 12]$ as our training points. If $i = 1$ then our test points would correspond to [5,4,6], whereas the training points would be $[3, 1, 2], [7, 8, 9], [11, 10, 12]$ and so on. Finally we simply take the average of all folds as our result. How this is done will be explained in the next section.

4 Calculation of the Results

With that in mind it is now time to calculate the actual results. To dampen the effect of high amplitudes, due to possible outliers, we included a regularization factor, λ, in our calculations. The range of our λ values is divided into 300 elements, ranging from -37 up to -1 in equal steps. For each lambda we calculate the training- and test errors, as well as the coefficients of our resulting functions with the K-Fold Cross Validation as seen in section 3. All resulting values then get sorted. From there on we chose 3 specific curves to showcase the effect of **underfitting**, **overfitting** and **optimal fitting**. For the underfitting curve we chose the λ value were the error for the training set was the highest.

On the contrary the overfitting curve is represented by the λ value that has the lowest training error, but a comparatively high test error. Finally the optimal fitting curve is simply the curve with the lowest combined training error and test error, i.e. the best approximation to our initial function. All three curves can be seen in Figure 2, together with the actual underlying function, the randomized data points and the initial sine function from which we approximated our results.

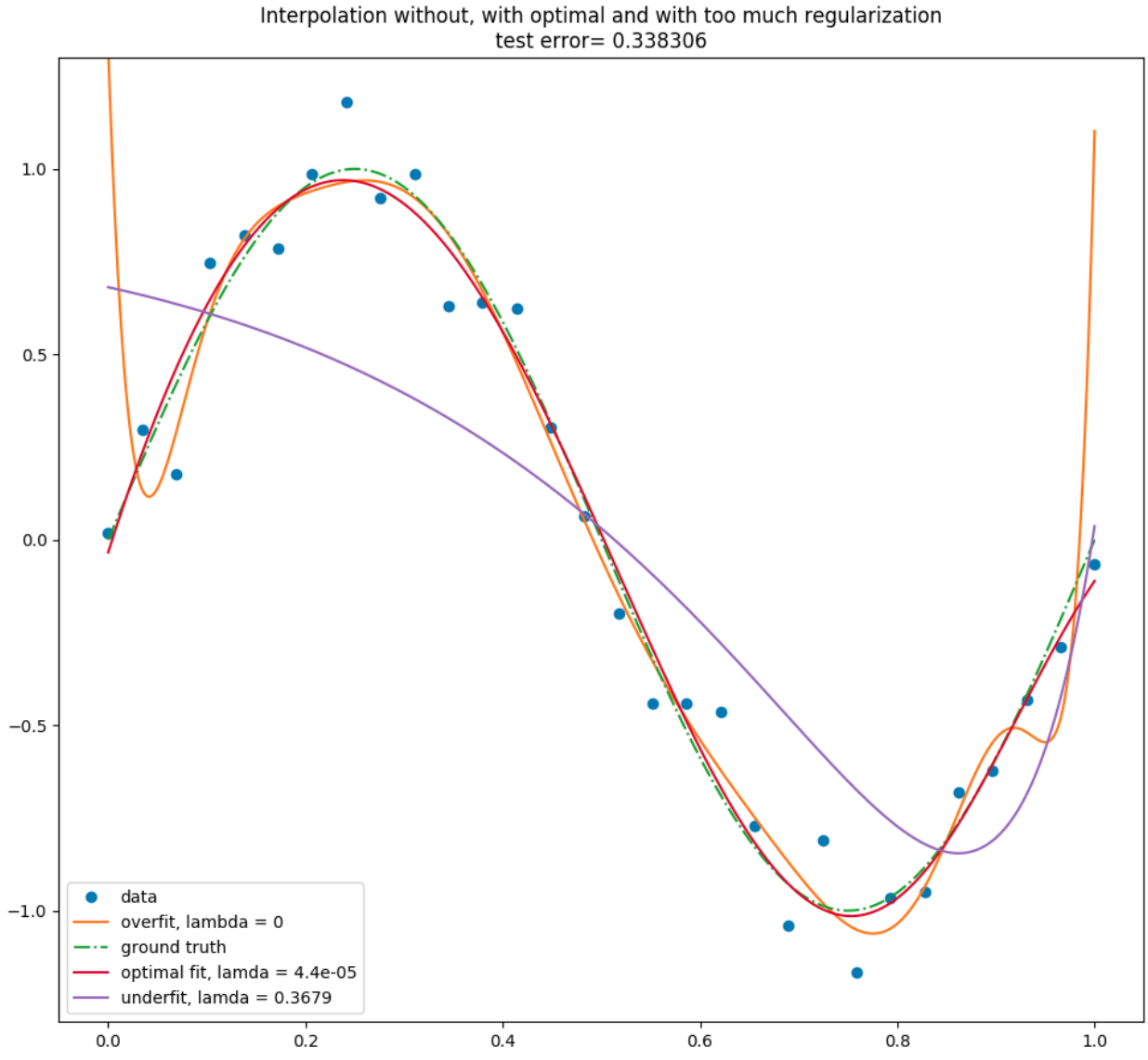

Figure 2: Various results with different λ values.

5 Error Ratio

The ratio between the test- and training errors can be seen in Figure 3. Note that we have taken the natural logarithm of the λ's so that we can properly display all values in a small plot. As we can see the more we go to the left of our graph, i.e. the λ values get smaller, the overfitting gets increased drastically. The error for the training set stays almost the same from about -8 to -37, however from -20 downwards the test errors skyrocket. This represents overfitting of our initial function and can also be seen as the orange curve in Figure 2. The bump between -20 and -10 is interesting, but (I don't know why this exists). Figure 3 also shows us underfitting on the right side of the plot. From about -8 up to 0 we can again see that the test errors skyrocket. However in contrast to overfitting, due to the nature of underfitting curves, the training errors also increase. This happens because we try to fit a lower order function to a higher order one of which it simply cannot follow the ups and downs.

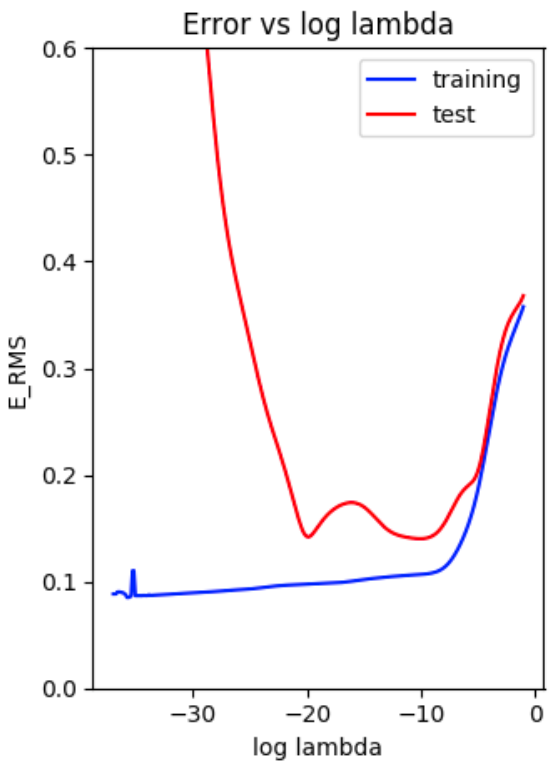

Figure 3: Ratio between the test- and training errors

Finally it has to be noted that we use randomized functions to test our program. So the λ values of the curves and the curves themselves may change each run. For example another run of the program yields the results as seen in Figure 4.

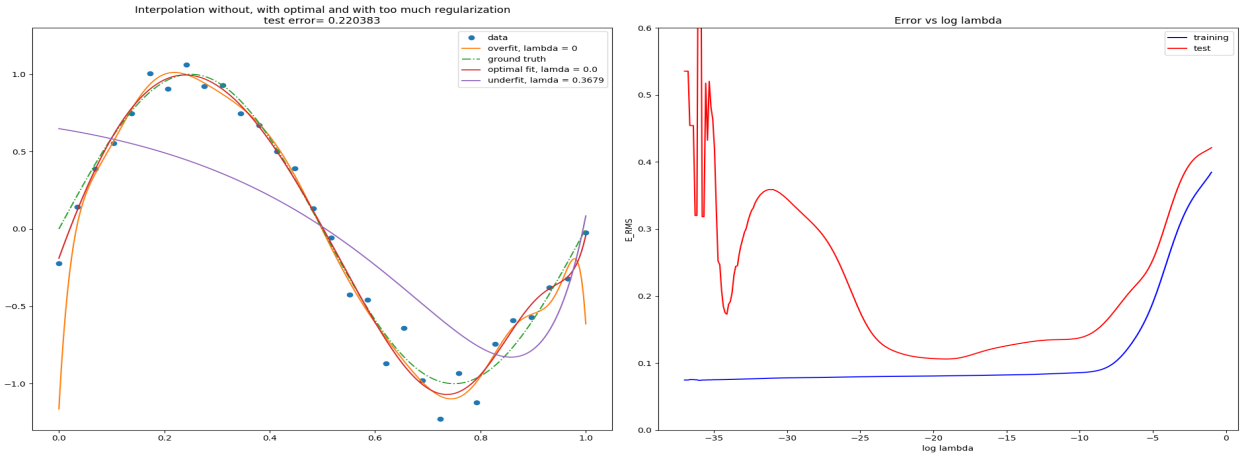

Figure 4: Another randomized run through with various results.