Assignment 1 - Programming Project: Ridge Regression and Model Selection

October 27, 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.

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 Partitioning

Exercise 1: Choose a nonlinear function $\mathbf{f}(\mathbf{x})$ *of a scalar variable* $\mathbf{x}$*. Generate a data set of* $\mathbf{N}$ *data points* $\mathbf{t_n} = \mathbf{f}(\mathbf{x_n}) + \epsilon$*, where is drawn from a normal distribution with variance.*

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

The next step is to divide our data points into training- and test sets. We chose to use $\frac{1}{5}$ 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 $k = \frac{1}{5}$ 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 $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ would give $[0, 1]$ 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.

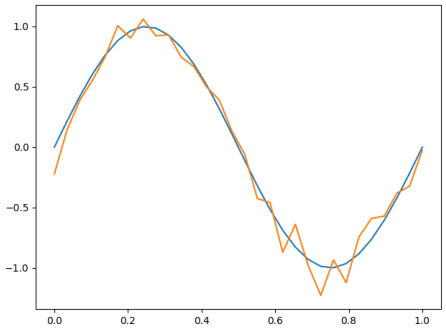


Figure 1: Initial sine function

Only $\sin(2 * \pi * x)$

Including randomized errors

A simple solution would be to order the vector and then take 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 $[0, 1, 5, 3, 7, 8, 4, 9, 2, 6]$ in which case the trainings points would again be [0,1].

There is another solution that fixes this problem without much trouble. First we have to order our vector x and then partition it into n buckets. For example if $n = 2$ then we would split x into 2 as equally big parts as possible. For x this would result in $[[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]]$. We chose n to be $k = \frac{1}{5}$ of our data points. The second step is to randomize each bucket. This could for example result in $[[3, 2, 4, 1, 0], [6, 9, 8, 5, 7]]$. 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.

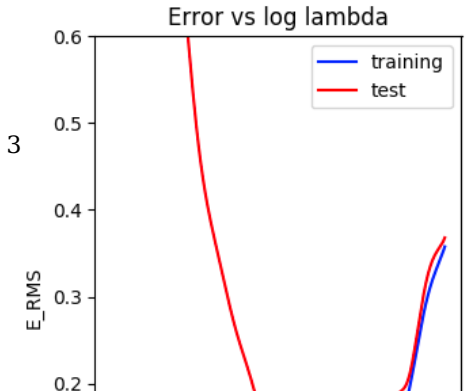
3 K-Fold Cross-Validation

Doing this is also convenient because we can use the buckets for our K-Fold Cross-Validation. Since our buckets have k elements we iterate k times over all buckets. In each iteration we take the $k-th$ element of each bucket and put them into a new bucket b. This new bucket then serves as one fold. k of all elements in the fold are then put into the test set (should always be 1 due to how the buckets got created) and the rest are training points.

4 Calculate 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. All resulting values 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. (In graph X this value corresponds to Y. Allthough since we use randomized functions the lambda values may change each run). On the contrary the overfitting curve is represented by the λ value that has the lowest training error, but comparatively high test error. Finally the optimal fitting curve is simply the curve with the lowest combined training error and test error. All three curves can be seen in Figure X, together with the actual underlying function, the randomized data points and the initial sine function from which we approximated our results. All of the described curves and values are depicted in Figure 2.

The ratio between the test- and training errors can be seen in Figure 3. Note that we have taken



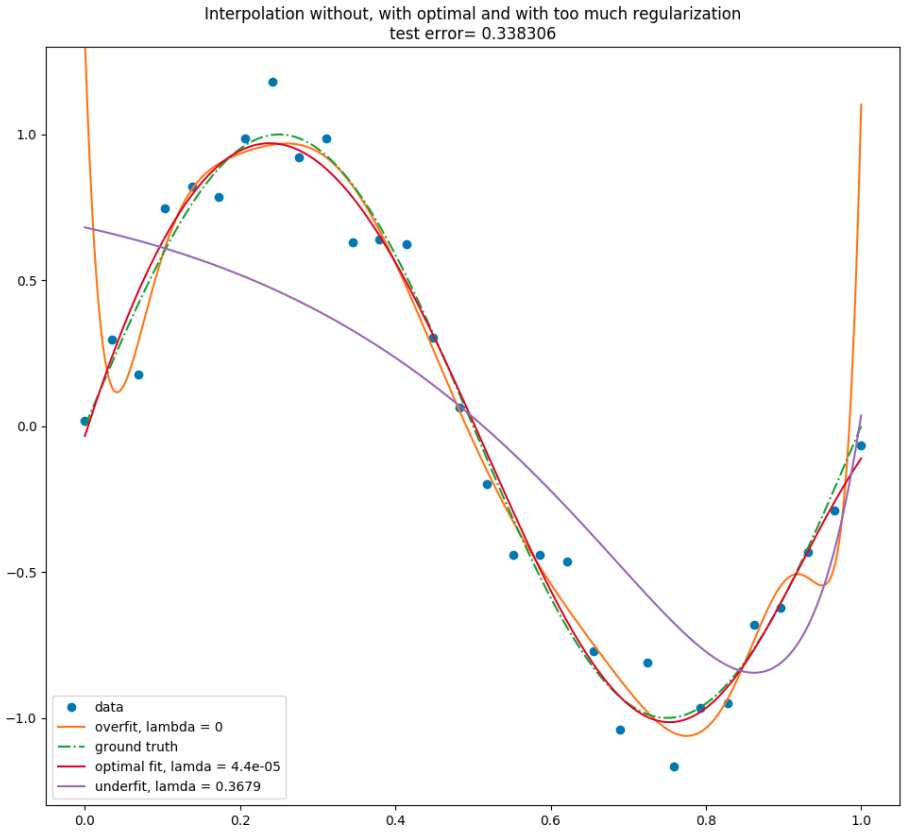


Figure 2

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 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.