CS 6140: Machine Learning
College of Computer and Information Science
Northeastern University

Instructor: Bilal Ahmed **Scribe: Bilal Ahmed & Virgil Pavlu**

Ridge Regression[1]

1 Introduction

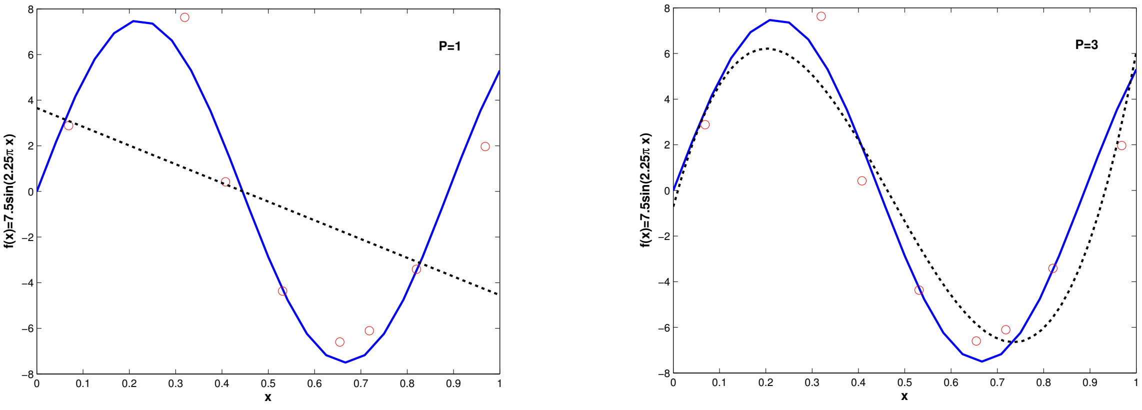
Conisder the polynomial regression task, where the task is to fit a polynomial of a pre-chosen degree to a sample of data comprising of data sampled from the function:

$$f(x) = 7.5\sin(2.5\pi x)$$

to which some random noise is added. So that the training labels can be represented as:

$$t = f(x) + \epsilon \tag{1}$$

where $\epsilon \sim Normal(0, \sigma^2)$ is additive Gaussian noise. The training dataset $\mathcal{D}$ is therefore represented as (x_i, t_i) $\forall i \in \{1, 2, \dots N\}$.

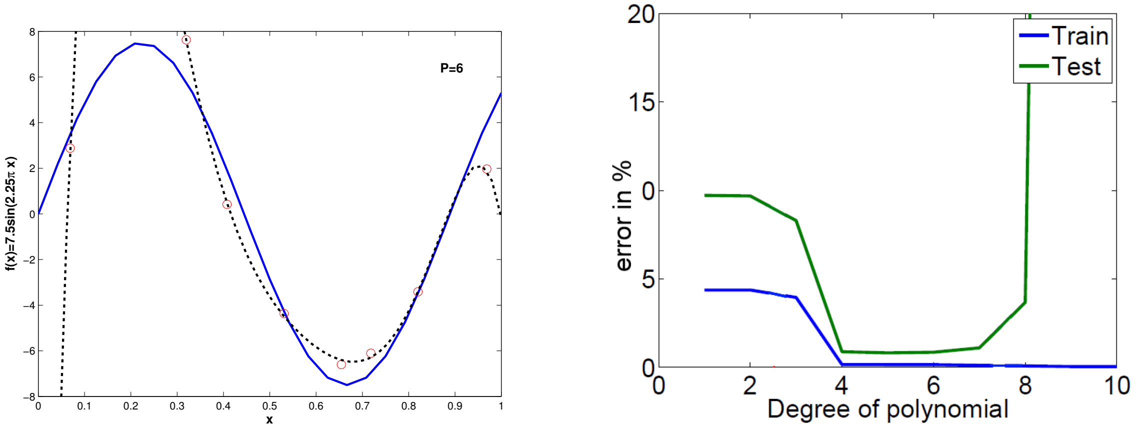


(a)

(b)

Figure 1: Fitting a polynomial to data sampled from $f(x) = 7.5\sin(2.5\pi x)$. The sampled data is shown as red circles, and the underlying function is shown as the solid blue curve, while the fitted functions are shown as the dotted black line. For a degree-1 polynomial shown in (a) we can see that the chosen polynomial class is unable to correctly represent the target function while in (b) the chosen polynomial very closely resembles the shape of the target function even though it does not pass through all the points perfectly.

[1] These lecture notes are intended for in-class use only.



(a)

(b)

Figure 2: (a) Fitting a degree-6 polynomial to data sampled from $f(x) = 7.5\sin(2.5\pi x)$. The sampled data is shown as red circles, and the underlying function is shown as the solid blue curve, while the fitted function is shown as the dotted black line. We can see that the learned function passes through the training data perfectly and will have a very low training error. But can we say anything about how it will behave on out-of-sample (test) data? (b) shows what happens to training and test error when we over and under fit to training data.

Figure-1 shows an example of fitting a degree-1 (straight line) and degree-3 (cubic polynomial), to a training dataset comprising of eight instances. It can be seen in Figure-1a that if we use a degree-1 polynomial as our hypothesis class we are unable to represent the function accurately, it is very intuitive because as compared to the target function our chosen hypothesis class is very simple. This is an example of *underfitting* where the hypothesis class fails to capture the complexities of the target concept. Figure-2a shows that a higher order polynomial (degree-6 in this case) fails to accurately represent the target function even though it perfectly predicts the label of each training instance. This is an example of *overfitting*, where instead of learning the target concept the learned concept aligns itself with the idiosyncrasies (noise) of the observed data. On the other hand by using a cubic polynomial (Figure-1b), we can see that it better rerepsents the target function, while having a higher training error than the degree-6 polynomial.

1.1 Occam's Razor

What happens when we overfit to the training data? Figure-2b, shows that in the case of overfitting, we achieve lower training errors but the error on unseen data (test error) is higher. So, what can we do to avoid overfitting? We have previously seen that in the case of Decision trees, we used pre-pruning and post-pruning to obtain shorter/simpler trees. Preferring simpler hypotheses that may have higher training error, over more complex hypotheses that have lower training error is an example of *Occam's Razor*. William of Occam was an English friar and philosopher who put forward this principle which states that "if one can explain a phenomenon without assuming this or that hypothetical entity, then there is no ground for assuming it i.e. that one should always opt for an explanation in terms of the fewest possible number of causes, factors, or variables". In terms of machine learning this simply implies that we should always prefer the simplest hypothesis that fits our training data, because “simpler hypothesis generalize well”.

How can we apply Occam’s Razor to linear regression? In other words, how can we characterize the complexity of linear regression? Recall, that in the case of decision trees the model complexity was represented by the depth of the tree. A shorter tree therefore, corresponded to a simpler hypothesis. For linear regression, the model complexity is associated with the number of features. As the number of features grow, the model becomes more complex, and is able to capture local structure in data (compare the three plots for the polynomial regression example in Figures 1 and 2), this results in high variance of estimates for feature coefficients i.e., w_i. This means that small changes in the training data will cause large changes in the final coefficient estimates. In order to to apply Occam’s razor to linear regression we need to restrict/constrain the feature coefficients.

2 Regularizing Linear Regression

For linear regression, we can impose a constraint on the feature coefficients so that we penalize the solutions that produce larger estimates. This can be achieved by augmenting the original linear regression objective with a constraint as follows:

$$\begin{aligned} \underset{\mathrm{w}\in\mathbb{R}^m}{\operatorname{argmin}} \quad & \sum_{i=1}^{N}(t_i - \mathrm{w}^T\mathrm{x}_i)^2 \\ \text{subject to:} \quad & \sum_{j=1}^{m} w_i^2 \leq c \end{aligned} \tag{2}$$

The objective function is exactly the same as before for linear regression: minimize the sum of squares errors over the training data. However, we have an added constraint on the feature weights that restricts them form growing abnormally large. By formulating the Lagrangian of Equation-2 we get the following optimization problem:

$$\mathrm{w}_{ridge} = \underset{\mathrm{w}\in\mathbb{R}^m}{\operatorname{argmin}} \; J(w) = \sum_{i=1}^{N}(t_i - \mathrm{w}^T\mathrm{x}_i)^2 + \lambda\sum_{j=1}^{m} w_i^2 \tag{3}$$

for $\lambda \geq 0$. There is a one-to-one correspondence between λ and c in Equation 2. For a two dimensional case i.e., $\mathrm{w} \in \mathbb{R}^2$, we can visualize the optimization problem as shown in Figure-3.

We know, that:

$$\sum_{j=1}^{m} w_i^2 = \mathrm{w}^T\mathrm{w} = \|\mathrm{w}\|_2^2$$

therefore, we can re-write the problem as:

$$\mathrm{w}_{ridge} = \underset{\mathrm{w}\in\mathbb{R}^m}{\operatorname{argmin}} \; J(w) = \sum_{i=1}^{N}(t_i - \mathrm{w}^T\mathrm{x}_i)^2 + \lambda\|\mathrm{w}\|_2^2 \tag{4}$$

Including an L2 penalty term in the optimization problem is also known as *Tirkhonov* regularization.

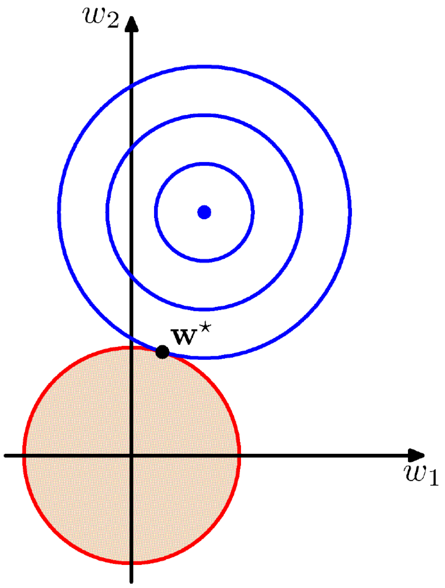


Figure 3: The red circle represents the constraints imposed on the individual coefficients, while the contour plot shows the sum of squares objective. w* is the ridge regression solution.

2.1 An Analytical Solution

We can get an analytical solution similar to how we derived the normal equations. Before we proceed we need to address the fact that the intercept or w_0 is part of the weight vector that we are optimizing. If we place a constraint on w_0 then essentially our solution will change if we re-scale the t_i's. By convention, we are going to drop the constant feature (all ones) from the design matrix, and also center each feature i.e., subtract the mean from the feature values. Let Z be the centered data matrix and let t be the vector of observed target values (generated according to Equation 1).

Then,

$$\begin{aligned}
\nabla_w J(\mathbf{w}) &= \nabla_w E^T E = \nabla_w (Z\mathbf{w} - \mathbf{t})^T (Z\mathbf{w} - \mathbf{t}) + \nabla_w \lambda \mathbf{w}^T \mathbf{w} \\
&= \nabla_w (\mathbf{w}^T Z^T Z\mathbf{w} - \mathbf{w}^T Z^T \mathbf{t} - \mathbf{t}^T Z\mathbf{w} + \mathbf{t}^T \mathbf{t}) + \lambda \nabla_w \mathbf{w}^T \mathbf{w} \\
&= \nabla_w (\mathbf{w}^T Z^T Z\mathbf{w} - \mathbf{w}^T Z^T \mathbf{t} - \mathbf{w}^T Z^T \mathbf{t} + \mathbf{t}^T \mathbf{t}) + \lambda \nabla_w \mathbf{w}^T \mathbf{w} \\
&= \nabla_w (\mathbf{w}^T Z^T Z\mathbf{w} - 2\mathbf{w}^T Z^T \mathbf{t} + \mathbf{t}^T \mathbf{t}) + \lambda \nabla_w \mathbf{w}^T \mathbf{w} \\
&= 2(Z^T Z\mathbf{w} - Z^T \mathbf{t}) + 2\lambda \mathbf{w} \\
&= 2Z^T Z\mathbf{w} + 2\lambda I\mathbf{w} - 2Z^T \mathbf{t} \\
&= 2(Z^T Z + \lambda I)\mathbf{w} - 2Z^T \mathbf{t}
\end{aligned}$$

where, in the third step we have $(AB)^T = B^T A^T$.

Setting the derivative to zero, we obtain

$$(Z^T Z + \lambda I)\mathbf{w} = Z^T \mathbf{t} \text{ or } \mathbf{w}_{ridge} = (Z^T Z + \lambda I)^{-1} Z^T \mathbf{t}$$

This solution corresponds to a particular λ, which is a hyper-parameter that we have chosen before hand. Algorithm-1 shows the steps involved in estimating the ridge regression solution for a given training set.

```
**Data**: Training Data:$(x_i, t_i)$ $i \in \{1, 2, \dots, N\}$
**Result**: Ridge Regression Feature Estimates: $w^*_{ridge}$
center the data: $z_{ij} = x_{ij} - \bar{x}_j$;
set the intercept: $w_0 = \bar{t} = \frac{1}{N}\sum_{i=1}^{N} t_i$;
choose $\lambda$;
estimate solution vector: $\hat{\mathbf{w}}_{ridge} = (Z^T Z + \lambda I)^{-1} Z^T \mathbf{t}$;
```

Algorithm 1: The gradient descent algorithm for minimizing a function.

3 Generalizing Ridge Regression

We can cast the objective function of ridge regression into a more general framework as follows:

$$\hat{\mathbf{w}}_{reg} = \underset{\mathrm{w} \in \mathbb{R}^m}{\text{argmin}} \ \text{Loss}(x, w) + \|\mathbf{w}\|_q$$

where, q represents the q^{th} vector norm. Some of the regularizers corresponding to different values of q are shown in Figure-4. The loss function for Ridge regression (for centered features) is given as:

$$\text{Loss}(x, w) = \sum_{i=1}^{N} (t_i - w_0 - \sum_{j=1}^{m} w_j x_{ij})^2$$

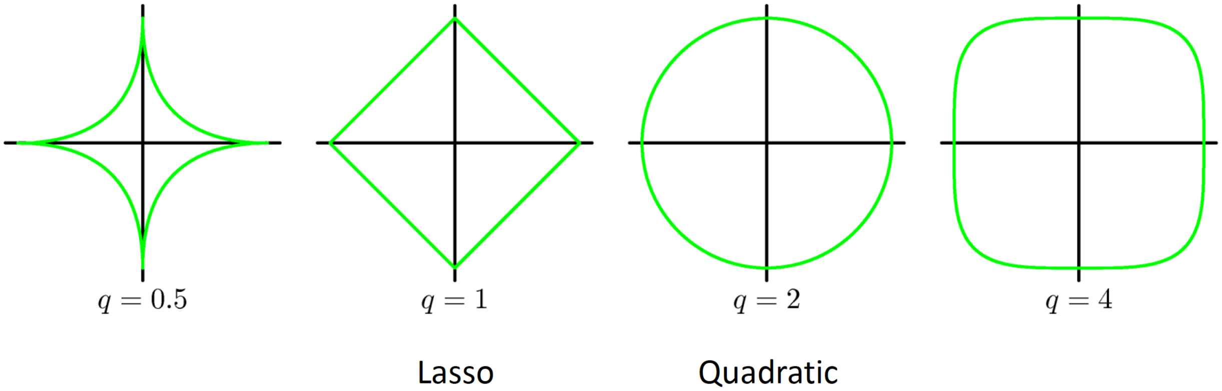


Figure 4: Regularizers corresponding to different values of q. When $q = 1$ we get the Lasso penalty which produces sparse solution by driving most of the coefficients to zero. Lasso can be used for performing feature selection, but cannot be analytically solved instead we need to resort to approximations. $q = 2$ produces the Trikhonov regularization resulting in Ridge regression.