CS 189 Introduction to Machine Learning
Fall 2017 Note 3

1 Regression and hyperparameters

Recall the supervised regression setting in which we attempt to learn a mapping $f : \mathbb{R}^d \to \mathbb{R}$ from labeled examples $\{(\vec{x}_i, y_i)\}_{i=1}^n$, where $y_i \approx f(\vec{x}_i)$. The typical strategy in regression is to pick some parametric model and then fit the parameters of the model to the data. Here we consider a model of the form

$$f_{\vec{\alpha}}(\vec{x}) = \sum_{j=1}^{\overset{\text{model order}}{p}} \underbrace{\alpha_j}_{\text{weights}} \underbrace{\phi_j(\vec{x})}_{\text{features}}$$

The functions ϕ_j compute **features** of the input $\vec{x}$. They are sometimes called **basis functions** because our model $f_{\vec{\alpha}}$ is a linear combination of them. In the simplest case, we could just use the components of $\vec{x}$ as features (i.e. $\phi_j(\vec{x}) = x_j$) but in future lectures it will sometimes be helpful to disambiguate the features of an example from the example itself, so we encourage this way of thinking now.

Once we've fixed a model, we can the optimal value of $\vec{\alpha}$ by minimizing some cost function L that measures how poorly the model fits the data:

$$\min_{\vec{\alpha}} L(\{\vec{x}_i, y_i\}_{i=1}^n, \vec{\alpha}) \overset{e.g.}{=} \min_{\vec{\alpha}} \sum_{i=1}^n (f_{\vec{\alpha}}(\vec{x}) - y_i)^2$$

Observe that the model order p is not one of the decision variables being optimized in the minimization above. For this reason p is called a **hyperparameter**. We might say more specifically that it is a **model hyperparameter**, since it determines the structure of the model.

Let's consider another example. **Ridge regression** is like ordinary least squares except that we add an ℓ^2 penalty on the parameters $\vec{\alpha}$:

$$\min_{\vec{\alpha}} \|A\vec{\alpha} - \vec{b}\|^2 + \lambda \|\vec{\alpha}\|_2^2$$

The solution to this optimization problem is obtained by solving the linear system

$$(A^T A + \lambda I)\vec{\alpha} = A^T \vec{b}$$

The regularization weight λ is also a hyperparameter, as it is fixed during the minimization above. However λ, unlike the previously discussed hyperparameter p, is not a part of the model. Rather, it is an aspect of the optimization procedure used to fit the model, so we say it is an **optimization hyperparameter**. Hyperparameters tend to fall into one of these two categories.

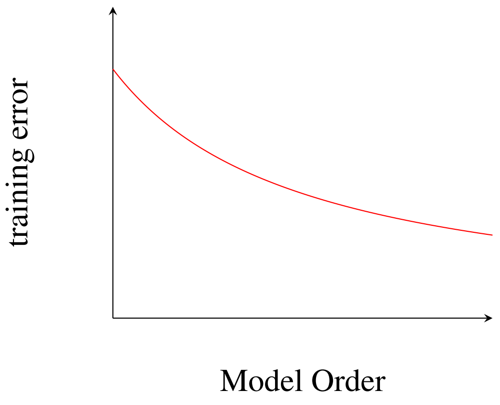
2 Kinds of error

Let us define the **mean-squared training error** (or just **training error**) as

$$\widehat{\text{MSE}}_{\text{train}}(\vec{\alpha}) = \frac{1}{n}\sum_{i=1}^{n}(f_{\vec{\alpha}}(\vec{x}_i) - y_i)^2$$

This is a measure of how well the model fits the data.

Note that as we add more features, training error can only decrease. This is because the optimizer can set $\alpha_j = 0$ if feature j cannot be used to reduce training error.

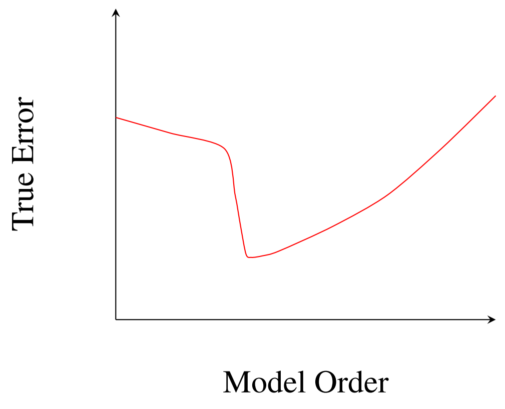


But training error is not exactly the kind of error we care about. In particular, training error only reflects the data in the training set, whereas we really want the model to perform well on new points sampled from the same data distribution as the training data. With this motivation we define the true error as

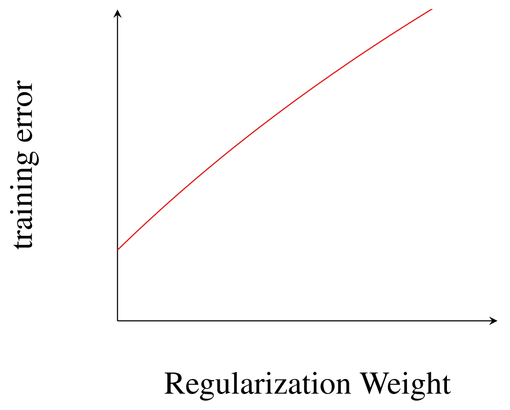
$$\text{MSE}(\vec{\alpha}) = \mathbb{E}[(f_{\vec{\alpha}}(\vec{x}) - y)^2]$$

where the expectation is taken over all pairs $(\vec{x}, y)$ from the data distribution. We generally cannot compute the true error because we do not have access to the data distribution, but we will see later that we can estimate it.

Adding more features tends to reduce true error as long as the additional features are useful predictors of the output. However, if we keep adding features, these begin to fit noise in the training data instead of the true signal, causing true error to actually increase. This phenomenon is known as **overfitting**.



The regularization hyperparameter λ has a somewhat different effect on training error. Observe that if $\lambda = 0$, we recover the exact OLS problem, which is directly minimizing the training error. As λ increases, the optimizer places less emphasis on the training error and more emphasis on reducing the magnitude of the parameters. This leads to a degradation in training error as λ grows:

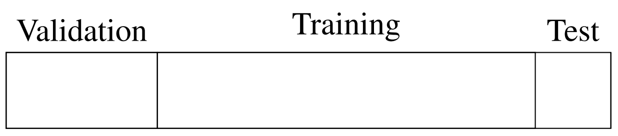


3 Validation

We've given some examples of hyperparameters and mentioned that they are not determined by the data-fitting optimization procedure. How, then, should we choose the values of p and λ?

3.1 Validation Error

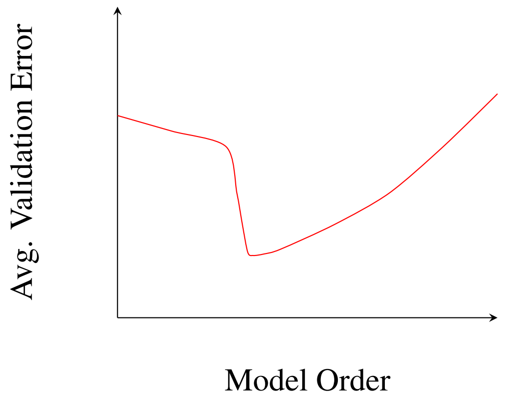
The key idea is to set aside some portion (say 30%) of the training data, called the **validation set**, which is *not* allowed to be used when fitting the model:



We can use this validation set to estimate the true error, as in

$$\widehat{\text{MSE}}_{\text{validate}}(\vec{\alpha}) = \frac{1}{m}\sum_{k=1}^{m}(f_\lambda(\vec{x}_{v,k}) - y_{v,k})^2 \approx \mathbb{E}[(f_\lambda(\vec{x}) - y)^2]$$

where f_λ is the model obtained by solving the regularized problem with value λ. The validation error tracks the true error reasonably well as long as m is large:



Note that this approximation only works because the validation set is disjoint from the training set. If we trained and evaluated the model using the same data points, the estimate would be very biased, since the model has been constructed specifically to perform well on those points.

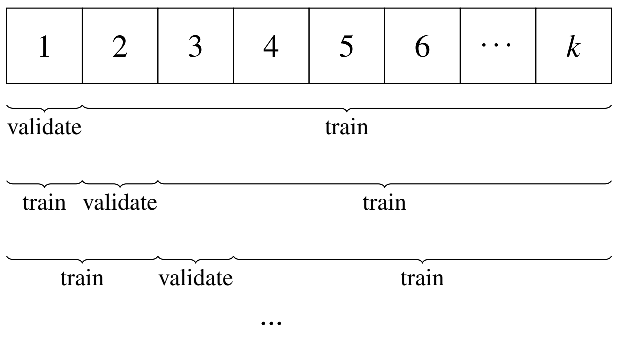
Now that we can estimate the true error, we have a simple method for choosing hyperparameter values: simply try a bunch of configurations of the hyperparameters and choose the one that yields the lowest validation error.

3.2 Cross-validation

Setting aside a validation set works well, but comes at a cost, since we cannot use the validation data for training. Since having more data generally improves the quality of the trained model, we may prefer not to let that data go to waste, especially if we have little data to begin with and/or collecting more data is expensive. Cross-validation is an alternative to having a dedicated validation set.

k-fold cross-validation works as follows:

1. Shuffle the data and partition it into k equally-sized (or as equal as possible) blocks.
2. For $i = 1, \dots, k$,
    - Train the model on all the data except block i.
    - Evaluate the model (i.e. compute the validation error) using block i.



3. Average the k validation errors; this is our final estimate of the true error.

Note that this process (except for the shuffling and partitioning) must be repeated for every hyper-parameter configuration we wish to test. This is the principle drawback of k-fold cross-validation as compared to using a held-out validation set – there is roughly k times as much computation required. This is not a big deal for the relatively small linear models that we've seen so far, but it can be prohibitively expensive when the model takes a long time to train, as is the case in the Big Data regime or when using neural networks.