Regularization and Cross Validation

Ryan P. Adams

These notes are meant to supplement the readings and should not be considered a substitute for doing the assigned readings in Bishop.

There is a deep sense in which machine learning is sort of impossible: we're trying to make interesting statements about data we've never seen before! That's what *generalization* is all about — build a model on one set of *training* data and hope that it makes effective predictions on a different set of *test* data. The Kaggle practicals get at this directly: you have both inputs and outputs for a training set and you are asked to make predictions on data for which you only have the inputs. The only reason this is possible is because we're willing to constrain the kinds of things that can be learned from data, e.g., that sensible regressions are linear weightings of basis functions or that classification boundaries are somewhat smooth as a function of input features.

These constraints are what we call an *inductive bias* — the assumptions we're willing to make about the world in order to be able to learn and make predictions on unseen data. Generally speaking, there are two kinds of inductive bias: restriction biases and preference biases. A restriction bias is a *limitation* on what things are possible. For example, we might allow quadratic polynomials for our regressors but forbid cubic and higher-order polynomials. This restricts the kinds of functions we can model, but enables us to avoid considering functions that very rapidly change their outputs in response to small changes in the input. A *preference* bias is a softer version of this in which we come up with a kind of ordering over the models and penalize those models that have more complexity. Ridge regression is an example of this: "complexity" here is represented as weight vectors with large L_2 norm, which would lead to functions with large slopes. We would prefer flatter, better behaved functions, all else being equal. In machine learning, this is called *regularization* and in statistics it is sometimes called *shrinkage*: don't necessarily fit the data as well as you can, but make the model more regular, or shrink the parameters towards a value like zero. This also comes up as *penalized maximum likelihood*, in which we apply a regularization penalty to prevent us from finding the true maximum likelihood parameters in the event that they would be pulled toward an undesirable region of the parameter space. Finally, this is exactly what priors are about in the Bayesian setting: a way to penalize models according to *a priori* beliefs about what parameters might be sensible. These ideas are all intricately related and there are few real differences in the assumptions made by Bayesian models with MAP inference, penalized maximum likelihood, or various regularized fitting procedures.

So, we have to choose ultimately between flexibility in our models and efficient use of data. Put simply, flexible models generally take more data to fit. In machine learning this is referred to as the *bias-variance tradeoff*, and Bishop provides a nice discussion in 3.2. Here *bias* means statistical bias, which you can think of as deviations between your model and the data based on assumptions encoded via a restriction or preference bias. For example, if you try to fit a linear model to data from a cubic polynomial, your solutions will be biased in the sense that the expectation of the fit

cannot recover the truth. On the other hand, *variance* is a measure of how the peculiarities of this particular data set influence what you learn. Decreasing the bias is appealing because you get more flexible models that can learn more things, but the bias-variance tradeoff means that you pay for it in sensitivity to noise that you may not care about. Bishop's discussion is worth a close read and Figure 3.5 is a good illustration; other books such as Elements of Statistical Learning also discuss this fundamental topic in detail.

When you encounter real data in the world and try to learn models that generalize well, the phenomena of bias and variance come up as *underfitting* and *overfitting*, respectively. Underfitting is where you have not introduced enough capacity and flexibility into your model and so you can't learn the important properties of the data, e.g., you use a quadratic basis for quartic data. Overfitting is often a much bigger problem, on the other hand, which is where the model is so flexible that it learns peculiarities of the training data that are really just noise. This is very frustrating because it can appear during training that your model is learning very well, but then when applied to test data, it makes poor predictions.

Cross Validation

Trying to find the balance between underfitting and overfitting so that you get good generalization is one of the main challenges of machine learning. In practice, it means trying to figure out how to set "knobs" of your learning procedure whose effect may be unintuitive, such a the size of an L_1 or L_2 penalty, the number of basis functions in a regression, the C parameter in a support vector machine, or the number of topics in a topic model. We would ideally like to set these to get optimal generalization, but of course we don't have the test data available to us yet. Instead, we often use something called a *validation set*.

Consider a predictive task in which we have 10,000 training examples and we're going to be asked to make predictions later on some test data. We build a nice and flexible model and train on the 10,000 data. It might get great training error, but how do we know it didn't overfit and memorize a bunch of silly dataset-specific noise? If we have regularization parameters, how do we set them? A very common approach is to remove a subset of the training data and call it a validation set. These data are not used for training, but are used to evaluate how well the model generalizes to data it hasn't seen before. They can then be used to determine these other "hyperparameters" such as the penalty in ridge regression. For example, we might take 2,000 of the 10,000 data and set them aside as a validation set. Then we would train our model on the other 8,000 and ask it to make predictions on the data we set aside. If the performance on the validation set is good, then we can feel comfortable that we are generalizing well.

A more involved procedure is called *cross validation* and it makes somewhat better use of the data. In cross validation, one takes the training set and divides it up into a set of K *folds*. Then, training is done K times, holding out a different fold as the validation set each time. Finally, generalization can be assessed by averaging performance across the K folds. The idea is to make sure that the validation set isn't accidentally special in some way. A typical value of K is five or ten. To be very concrete, imagine that we had our 10,000 data from before. To perform five-fold cross validation we would chop these data up into five disjoint sets of 2,000 data each. Call these

five sets A, B, C, D, and E. We would then iterate through five training procedures:

$$\text{Train on } B \cup C \cup D \cup E \text{ validate on } A.$$
$$\text{Train on } A \cup C \cup D \cup E \text{ validate on } B.$$
$$\text{Train on } A \cup B \cup D \cup E \text{ validate on } C.$$
$$\text{Train on } A \cup B \cup C \cup E \text{ validate on } D.$$
$$\text{Train on } A \cup B \cup C \cup D \text{ validate on } E.$$

Finally, the five validation performances would be averaged to assess generalization.

There are other variants, including randomized cross validation in which the folds are chosen randomly (and so data may appear multiple times in the validation set), and *leave-one-out cross validation* in which K is chosen to be the same size as the data set.

Also note that in fully Bayesian models, cross validation may not be necessary or sensible. The marginal likelihood is a common and coherent way to perform model selection in this case. Bishop 3.4 provides an excellent discussion of this topic.