11 Cross Validation

Suppose we need to choose between two possible ways to fit a model to some dataset. For example, we might want to compare the performance of different classification algorithms on a given dataset. Or, in the case of a regression problem, we might consider different classes of basis functions (e.g., polynomials or RBFs).

Sometimes we want to compare the performance of two different versions of the same type of model, but with different settings of a hyper-parameter. For example, when solving a regression problem one might need to specify the number of RBF functions, or the the order of the polynomials. In regularized regression, there is a regularization parameter that controls the weighting of data and smoothness terms in the objective; choosing a good value of this weight is often difficult to do a priori, but it often has a major impact on the quality of the learned model.

In classification algorithms there are a number of key parameter choices that might be essential to training a good model. Examples include the neighborhood size for a k-NN classifier (i.e., k). As we discussed above, a larger value of k tends to smooth the decision boundaries, which may yield better predictions in the face of noise in the training data. In the case of decision trees (and decision forests), one needs to decide on the depth of the trees. Although one may apply the recursive learning procedure down to leaves that only contain one training point, this is often prone to small amounts of error in training data, and tends to overfit. It is therefore common to prune the tree back from the leaves, but how far should one prune? Indeed, many algorithms have one or more parameters like this which are necessary to specify.

What's the right measure of performance? How should we choose between two learned models? Simply measuring how well different models (or different hyper-parameter values) fit the data would mean that we always try to fit the data as closely as possible. The best method for fitting the data is simply to memorize it in big look-up table. However, fitting the data is no guarantee that we will be able to **generalize** to new measurements. For example, higher-order polynomials will always fit data as well or better than low-order polynomials. Indeed, an $N-1$ degree polynomial will fit N data points exactly (to within numerical error). So just fitting the data as well as possible usually produces models with many parameters, which do not generalize well to new inputs in almost all cases of interest.

11.1 Hold-Out Validation

One general solution is to evaluate models by testing them on new data (the "validation set"), distinct from the training set. This measures how **predictive** the model is. That is, is it useful in new situations? More generally, we often wish to obtain empirical estimates of performance. This can be useful for finding errors in implementation, comparing competing models and learning algorithms, and detecting over- or under-fitting in a learned model. In particular, it is widely used to determine model parameters that might otherwise to hard to determine, such as k in k-NN classification, or the bandwidth σ in RBF regression.

In the simplest method, we first partition our data randomly into a "training set" and a "validation set." Let k be the unknown model parameter. We pick a set of range of possible values for k (e.g., $k = 1, ..., 5$). For each possible value of k, we learn a model with that k on the training set,

and compute that model's error on the validation set. For example, the validation error might be just the squared-error, $\sum_i (\, y_i - f(x_i)\,)^2$. We then pick the k with the smallest validation error. The same idea can be applied if we have multiple parameters (e.g., k and the σ in the k-NN weighting function), however, the number of possible combinations of k and σ we have to try becomes large very quickly. So this is only feasible with a relatively small number of hyper-parameters.

The other problem with this approach is that the models are fit with smaller training sets. So we only get good results if our initial training set is rather large. If large amounts of data are expensive or impossible to obtain this can be a serious problem. In that case one might opt to use K-fold cross validation.

11.2 K-Fold Cross Validation

In K-fold cross-validation we randomly partition the training data into K sets of equal size and run the learning algorithm K times. Each time, a different one of the K sets is deemed the validation set, and is therefore held out during learning; i.e., the model is trained on the remaining $K-1$ sets. The quality of the model is determined by averaging the validation errors across the K models. We can then pick the model (or hyper-parameter values) that gave the lowest score. We then fix those hyper-parameters, or the type of model, and fit a single model using all the data.

With small datasets, a good choice for K is N, where N is the number of points in the training set. In other words, we learn as many models as there are data points, each time using all but one data point. In this way we use as much data as possible to learn all the models while holding out one point each time, which is then used to test the model predictions. This is called **leave-one-out cross-validation**, abbreviated LOOCV.

The main benefit of LOOCV is that is can be used for smaller datasets, and it's simple to implement. The downside is that it can be extremely time consuming because one has to fit as many models as one has datapoints. For linear regression problems, interestingly, it turns out that one can use LOOCV by fitting just one model. Let MSE_i is the squared error in the prediction of the ith input for a model learned with all but the ith data-point. Then the LOOCV score function measures the average error over the N models (where N is the size of the training dataset), as

$$CV \;=\; \frac{1}{N} \sum_{i=i1}^{N} MSE_i \tag{11.1}$$

For LS linear regression, or basis function regression, it can be shown that CV can be computed from a single model that was fit using the entire dataset. In this case the LS solution has the form $\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$ and the predicted values of the model are given by $\hat{\mathbf{y}} = X\mathbf{w} = H\mathbf{y}$, where $H = X(X^T X)^{-1} X^T$. Then, the CV score can be expressed as

$$CV \;=\; \frac{1}{N} \sum_{i=1}^{N} \left(\frac{y_i - \hat{y}_i}{1 - h_i} \right)^2 \tag{11.2}$$

where y_i and $\hat{y}_i$ are the ground truth and predicted outputs, and h_i is the ith diagonal entry of H. Here, h_i is often called the leverage of the ith point as it is related to the influence of the ith point on the LS model fit.

11.3 Issues with Cross Validation

Cross validation is a very simple and empirical way of comparing models. However, there are a number of issues to keep in mind:

- The method can be very time-consuming, since many training runs may be needed. For models with more than a few parameters, cross validation may be too inefficient to be useful.
- Because a reduced dataset is used for training, there must be sufficient training data so that all relevant phenomena of the problem exist in both the training data and the test data.
- It is safest to use a random partition, to avoid the possibility that there are unmodeled correlations in the data. For example, if the data was collected over a period of a week, it is possible that data from the beginning of the week has a different structure than the data later in the week.
- Because cross-validation finds a minimum of an objective function, over- and under-fitting may still occur, although it is much less likely. For example, if the test set is very small, it may prefer a model that fits the random pattern in the test data.

Aside:
Testing machine learning algorithms is very much like testing scientific theories: scientific theories must be predictive, or, that is, falsifiable. Scientific theories must also describe plausible models of reality, whereas machine learning methods need only be useful for making decisions. However, statistical inference and learning first arose as theories of scientific hypothesis testing, and remain closely related today.

One of the most famous examples is the case of planetary motion. Prior to Newton, astronomers described the motion of the planets through onerous tabulation of measurements — essentially, big lookup tables. These tables were not especially predictive, and needed to updated constantly. Newton's equations of motion, which could describe the motion of the planets with only a few simple equations, were vastly simpler and yet also more effective at predicting motion, and became the accepted theories of motion.

However, there remained some anomolies. Two astronomers, John Couch Adams and Urbain Le Verier, thought that these discrepancies might be due to a new, as-yet-undiscovered planet. Using techniques similar to modern regression, but with laborious hand-calculation, they independently deduced the position, mass, and orbit of the new planet. By observing in the predicted directions, astronomers were indeed able to observe a new planet, which was later named Neptune. This provided powerful validation for their models.

Incidentally, Adams was an undergraduate working alone when he began his investigations.

Reference: http://en.wikipedia.org/wiki/Discovery_of_Neptune

