L19: Cross Validation & Uncertainty

If there was no noise in data, then data mining would be too easy, and a mainly solved topic.

What is noise in data? There are several main classes of noise, and modeling these can be as important as modeling the structure in data.

- **Spurious readings.** These are data points that could be anywhere, and are sometimes ridiculously far from where the real data should have been. With small data sets, these often pruned by hand. With large sensed datasets, these need to be automatically dealt with. In high dimensions they are often impossible to physically "see."
- **Measurement error.** This is a small amount of error that can occur on *all* data points and in *every* dimension. It can occur because a sensor is not accurate enough, a rounding or approximation error occurs before the data reaches you, or a truncation error in a conversion to an abstract data type. This type of data noise may have little effect on the structure you are trying to measure, but can violate noiseless assumptions and should be understood.
- **Background data.** These data are from something other than what you are trying to measure. This could be unlabeled data (like unrated movies on Netflix) or people who don't have a disease you are trying to monitor. The problem is that it sometimes gets mixed in, and indistinguishable, front he actual data of the phenomenon you are trying to monitor.

We will deal with this noise in 3 ways. First *cross validation* that helps one choose parameters, and also shows the influence of noise. Second is directly considering *uncertainty in data*, and modeling its effect on the output. Third is about outliers, and is discussed later.

19.1 Cross Validation

Many algorithms we discuss have a parameter. Probably all algorithms that deal effectively with noise have some parameter. The k in k-means clustering, the degree of a polynomial in polynomial linear regression, the number of singular values to use in PCA, the regularization parameter s in Lasso. How should we choose this parameter?

We discussed many techniques. They key principal we emphasized was to reduce the problem to a simple and robust enough one that we could eye-ball it. Then we could hand choose the right parameter. If this seems un-satisfying, a more automated way is cross-validation.

The basic idea is simple: Divide P (randomly) into two sets $Q, R \subset P$ so $Q \cap R =$ and $Q \cup R = P$. Let Q be the training set, and R be the test set.

Then we run our algorithm to build a model on Q with parameter α: $M(Q, \alpha)$, and then evaluate the error $L_R(M(Q, \alpha))$ on R.

For instance, lets say the problem is to find a robust linear regression A using Tikhinov Regularization minimizing the *loss function*:

$$L_P(A, \alpha) = \|P_y - P_X A\|_2 + \alpha \|A\|_2.$$

We we solve $A(Q, \alpha) = (Q_X^T Q_X + \alpha^2)^{-1} Q_X^T Q_y$ and then evaluate $L_R(A(Q, \alpha), 0)$. We can then search for the minimum value of $L_R(A(Q, \alpha), 0)$ as a function of α. This gives us a good choice of α.

The idea behind this is that P is drawn from some underlying distribution μ, which we don't have access to. So we need to pretend we could draw more points from the true μ. Instead of creating points out of the vapor, we use some points R from P. But it is cheating to also train on these points so we need to keep the training set Q and test set R separate.

Leave One Out Cross-Validation. The above simple test set / training set problems gives too much emphasis to the arbitrary split of R and Q. A less arbitrary way is to always let the test set R be a single point p. This way we are essentially using as much data as possible to train $Q_p = P \setminus p$, and still getting a test. We can then repeat this for each points $p \in P$, and take the α which is best on average.

This can be too expensive (it requires $|P|$ constructions of our model). So sometimes we leave some k (perhaps $= |P|/10$) points out each time and get $|P|/k$ different training / test set splits where each p is tested once.

Or we can take out some k number of test points at random and repeat this t times, for a sufficiently large t.

Testing on the Training Set (or the full set) can lead to ***over-fitting*** where a model too complex is found, but seems to fit the data really well.

Bootstrapping. A downside of Leave-One-Out (LOO) cross validation (and more so its more efficient variants) is that it reduces the size of the test set P artificially. If we have $|P|$ data points, we should be able to use all of them. More over with LOO (which only decreases to $|P|$ points), we can only perform $|P|$ trials to average over. Perhaps we would like to do more to improve out estimate.

Bootstrapping (Efron 1979) chooses a set $Q \subset P$ **with replacement** of size $|P|$. So it has exactly the same size as P itself. Then it trains on Q to get $M(Q)$ and then also test on Q as $L(M(Q), Q)$ to get an estimate of the error.

It then repeats this several times (maybe 1000 of $10{,}000$ times) with new Q choose with replacement from P.

This not only can be used for cross-validation, but is even more powerful (and designed for) understanding the effect of the noise in the data (with respect to how it is chosen from an imaginary μ) on the model M.

For instance, one may find that the mean (as M) has much more variance than the median (as M) and be inclined to use the median instead of the mean.

19.2 Uncertain Data

Sometimes a data set P is presented to a user with known uncertainty in it. Perhaps the way it was sensed indicates some distribution μ_i on the true value of each point $i \in P$. Or perhaps each object $p_i \in P$ is sensed multiple times so we have a representation $p_i = \{p_{i,1}, p_{i,2}, \ldots, p_{i,k}\}$. Thus each "point" $p \in P$ is actually represented as several (lets say k) different possible points. The common practice is then to treat p_i as a probability distribution μ_i where

$$\mu_i : \mathbf{Pr}[p_i = x] = \begin{cases} 1/k & \text{if } x = p_{i,j} \in p_i \\ 0 & \text{otherwise.} \end{cases}$$

Lets be clear on the difference between this model and the above model. Here we trust that each point exists, but it is drawn from an individual distribution μ_i (these are often assumed independent, but can have covariance). In the above model, we assume that *all* points are drawn from a single distribution μ.

So how should we understand this uncertainty? Instead of a single model, we should have a *distribution of models*. This is most easily seen with a single point model, such as the mean, for one-dimensional data. In this can be seen as a distribution f_P on the point that is the mean. In particular, let $f_P : \mathbb{R} \to \mathbb{R}^+$ represent the probability that its input is the mean. This is a bit hard to understand since when each μ_i is continuous (like a Gaussian) then all values $f_P(x) = 0$. It is easier to think of the cumulative density function $F_P(x) = \int_{z=-\infty}^{x} f_P(x)dx$.

Calculating F_P exactly can be quite challenging, but it is easy to approximate. We say g is an ε-*quantization* of F_P if for all $x \in \mathbb{R}$ we have $|g(x) - F_P(x)| \leq \varepsilon$. We can create such a function g as

Algorithm 19.2.1. This generated an ε-quantization with probability at least $1 - \delta$ since each m_j is a random sample from f_P. Then we can apply a Chernoff-Hoeffding bound.

Algorithm 19.2.1 Monte Carlo (ε, δ)-Quantization

for $j = 1$ **to** $t = O((1/\varepsilon)^2 \log(1/\delta))$ **do**
  for all $p_i \in P$ **do**
    Choose some $q_{i,j} \sim \mu_i$.
  Let $Q_j = \{q_{1,j}, q_{2,j}, \ldots, q_{n,j}\}$.
  Calculate estimator m_j from Q_j.
Let $\mathcal{M} = \{m_1, \ldots, m_t\}$.
Return $g(x) = |\{m_j \in \mathcal{M} \mid m_j \leq x\}|/t$.

This representation of g is actually quite easy to work with since it is a step function defined by $t = O((1/\varepsilon)^2 \log(1/\delta))$ points. This value t is *independent of $|P|$, the estimator (as long as it is a one-dimensional point), and all μ_i*! However, t can be quite large if ε is small. For instance if $\varepsilon = 0.01$ (allowing 1% error), and $\delta = 0.001$, then $t = 30{,}000$.

We can compress g with the same error bounds by sorting the points $\mathcal{M}$ and taking only one out of every t/s in the sorted order where $s = 1/\varepsilon$. Now the size is much smaller, where for $\varepsilon = 0.01$ only $s = 100$ points are needed.

A challenging open problem of the data is to extend this sort of approximate representation of uncertain output distributions for more complicated models and estimators.