Lecture 8: Model selection and structural risk minimization

Akshay Krishnamurthy
akshay@cs.umass.edu

September 28, 2017

1 Recap

Last time we saw our first real learning algorithm (that wasn't obviously ERM), namely kernel regression. Recall the estimator took the form

$$\hat{\eta}(x) = \frac{\sum_{i=1}^n Y_i K(\|X_i - x\|/h)}{\sum_{i=1}^n K(\|X_i - x\|/h)}$$

where the data $(X_i, Y_i) \in \mathcal{X} \times \mathcal{Y}$ and $K : \mathbb{R} \to \mathbb{R}$ is a smoothing kernel. We showed that if the true regression function was Lipschitz continuous and the bandwidth parameter h is chosen appropriately, then we can estimate the regression function in integrated MSE at rate of $O(n^{-2/(d+2)})$. The critical parameter was the bandwidth. How do we choose the bandwidth in practice? More generally, how do we decide what family of hypotheses we should use for learning?

2 Cross Validation

The approach that is most popular in practice is cross validation. The idea is to split the dataset into two groups, a training set and a validation set, and then learn a bunch of predictors on the training set and then choose the predictor that does the best on the validation set. Since the validation set was not used for the training process, there is minimal danger of overfitting (except if used improperly, which we'll get to below). Or more formally, the validation risk concentrates sharply to the true risk, with no need for uniform convergence.

Proposition 1. *Let h be some predictor obtained by training on a dataset S and let V be an iid dataset of size n. Assume the loss function is in $[0, 1]$. Then for any δ, with probability at least $1 - \delta$*

$$|\hat{R}_V(h) - R(h)| \le \sqrt{\frac{\log(2/\delta)}{n}}.$$

Remember that we usually get a bound like $O(\sqrt{\frac{d \log(1/\delta)}{n}})$ for the ERM of a class with VC-dimension d. Thus the risk on the validation set is a more precise estimate of the true risk than what we can even hope for, so there is essentially no loss.

In cross-validation we typically train a bunch of models, all on the training data, and then look at their validation risk, choosing the model with the lowest validation risk. This method is justified by the above proposition and the union bound, since we only need concentration on a few hypotheses on the validation set.

Example 1 (Kernel Regression)**.** *In kernel regression, we can try a bunch of bandwidth parameters $\mathcal{H} = \{h_1, \ldots, h_N\}$. If we split the data in half, then we estimate the MSE on the second half of the data by*

$$\frac{2}{n} \sum_{i=n/2+1}^{n/2} (\hat{\eta}_h(X_i) - Y_i)^2.$$

In cross validation we would take the best of these, but observe that we only required concentration of N risk estimates, which is usually negligible compared to the MSE of any one of the estimators. We can also try different smoothing kernels.

Example 2 (Regularized Regression)**.** *Another common example of this is with regularization or with bounds on the function class we are optimizing over. Remember that the Rademacher complexity of the class of linear functions grows with the bound on the norm of the weight vectors. Hence you might want to do linear regression over norm balls of different sizes and then do cross validation to choose one. This is strikingly similar to the regularized ERM*

$$\hat{w}_\lambda = \operatorname{argmin} \hat{R}_n(w) + \lambda \|w\|_2^2.$$

Then we can optimize for many different values of λ and do cross validation over these.

K-fold and Leave-One-Out. Two things that are common in practice, but less well-studied theoretically are k-fold and leave one out cross validation. In the former, instead of splitting the data in half, you split the data into k groups, run your algorithm on $k-1$ of the groups and evaluate the ri sk on the last group. You repeat this where you leave out each of the groups, giving you k numbers that you average together. Leave-one-out is the extreme version of this where $k = n$. The idea is that this is more robust, since you are training on more of the data, and you are still averaging in the risk estimate. The challenge is that now you are re-using the data in different training and validation folds. So the terms in the averages are not independent. There is some work on this, but understanding k-fold CV is still fairly open.

On the other hand, from a theoretical perspective there is little to be said, since we know that the train/validation split affects your excess risk by a factor of at most 2, which seems fine theoretically.

Example 3. *We actually analyzed a leave-one-out estimator in a recent paper on nonparametric estimation. At a high level, we wanted to estimate the entropy of a continuous distribution, and we only wanted to make mild smoothness assumption, so we're in the nonparametric setting. The data-splitting approach here is motivated by re-writing the entropy*

$$H(p) = -\int p(x)\log p(x)dx = -\mathbb{E}_{x\sim p}[\log p(x)].$$

So the data-splitting approach uses half the data to build a density estimator $\hat{p}_{1:n/2}$ and the other half to estimate the expectation:

$$\hat{H}_{split} = \frac{-2}{n}\sum_{i=n/2+1}^{n} \log \hat{p}_{1:n/2}(x_i).$$

The leave-one-out estimator uses all but one sample to estimate the density $\hat{p}_{-i}$ and evaluates on x_i, but then averages all of these:

$$\hat{H}_{LOO} = \frac{-1}{n}\sum_{i=1}^{n} \log \hat{p}_{-i}(x_i)$$

Actually to analyze estimators like this properly it seems you have to use Efron-Stein, since there are many correlated terms, but the conditional variances are all small. You could also try to use McDiarmid/bounded differences, but it will give you a polynomially worse rate.

2.1 Adaptive Data Analysis

One thing that you do need to be careful about is how you use your validation data. In all of the above, I have been careful to say that you choose all your hypotheses up front looking at the training data however you want, and then you evaluate all of them on the validation data. If you interleave choosing hypotheses and evaluating on the validation set, you risk overfitting to the validation set. This has been a major issue in scientific investigation, where it is common to test many hypotheses, chosen adaptively, on a single, expensive-to-collect, dataset. As a result, this has come to be known as *adaptive data analysis*, which is a thriving research area within machine learning.

As just a simple example, let's see what can go wrong and how you can easily overfit the validation data. Let's work with the 0/1 loss and assume that you have n validation points. Instead of asking for the validation error of a predictor, you simply supply a list of predictions $u \in \{0,1\}^n$ and you recieve back $\frac{1}{n}\sum_{i=1}^{n} \ell_{0/1}(y_i, u_i)$ where y_i are the true labels for the examples in the validation set. The so called "boosting attack" is as follows:

1. Select $u_1, \ldots, u_k \in \{0,1\}^n$ with uniform random entries.
2. Query the validation set on these k vectors to get $\ell_1, \ldots, \ell_k$.
3. Set $I = \{i \in [k] \mid \ell_i \leq 1/2\}$
4. Set $u_i^\star = \mathrm{maj}(\{u_i \mid i \in I\})$ for each coordinate i.

Claim 2. *The boosting attack finds a vector $u^\star \in \{0,1\}^n$ such that $\hat{R}_V(u^\star) \leq 1/2 - \Omega(\sqrt{k/n})$ with high probability.*

Proof sketch. The idea behind the proof is that conditionally on having validation error $\leq 1/2$ the probability that $u_i \neq y_i$ is slightly smaller than 1/2. Thus when you take majority of many items that are positively biased, you tend to actually get the right thing. □

The issue here is that you can actually learn about the validation labels very quickly by making well-informed queries to the validation set. This is how you overfit something, and the sequential nature here lets you overfit much more quickly than if we issued queries in a batch. Avoiding this issue, and other related ones, is the subject of adaptive data analysis. At a very high level, an idea that has been popular in that line of work is to introduce some small amount of noise into the reported validation error, which limits how much information about that dataset leaks into your next choice.

3 Structural Risk Minimization

In the more general setting, imagine we have a possibly infinite sequence of hypothesis classes $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots,$ at our disposal. Further assume that each $\mathcal{H}_i$ has the uniform convergence property, but possibly with different functions $n_{\mathcal{H}_i}^{\mathrm{UC}}$. As usual, if n is big, we should choose a large class, which corresponds to i big, but if n is small, we should choose a smaller class to better balance estimation and approximation. Structural risk minimization is a way to basically do this for free, and get the best of both worlds.

The first observation is that if we weight the classes appropriately by taking a union bound, we can get a convergence result that is fairly similar to the uniform convergence results we have used in the past. This is a form of non-uniform learnability. Let us define

$$\epsilon_i(n, \delta) = \min\{\epsilon \mid n_{\mathcal{H}_i}^{\mathrm{UC}}(\epsilon, \delta) \leq n\}$$

to be the smallest error that you can achieve in the uniform convergence bound with n samples and failure probability δ. We will also need a weighting function $w : \mathbb{N} \to [0,1]$ such that $\sum_i w(i) \leq 1$. This function should be chosen so that $w(i)$ encodes how much we believe the best hypothesis is in $\mathcal{H}_i$. Some choices that encode minimal beliefs are

1. If there are finitely many classes, say just N. Then set $w(i) = 1/N$.
2. Otherwise set $w(i) = \frac{6}{\pi^2 n^2}$ which is a convergence sequence. In the nested setting, this expresses a very mild preference for smaller clases.

The basic lemma underlying the Structural Risk Minimization procedure is now easy for us to prove if we work through the definitions and use the union bound.

Lemma 3. *For every $\delta \in (0,1)$ and distribution $\mathcal{D}$, with probability at least $1 - \delta$, the following bound holds simultaneously for all $i \in \mathbb{N}$ and all $h \in \mathcal{H}_i$*

$$|R(h) - \hat{R}(h)| \leq \epsilon_i(n, w(i)\delta)$$

Proof. The proof is a simple consequence of the union bound. We know that for class $\mathcal{H}_i$ we can achieve error $\epsilon_i(n, w(i)\delta)$ using n samples and with failure probability $w(i)\delta$. Repeating this for every class, the total failure probability is $\sum_i w(i)\delta \leq \delta$ which proves the result. □

Structural Risk Minimization. This motivates SRM, which is a form of bound-minimization algorithm.

1. For each i compute $\hat{h}_i = \mathrm{argmin}_{h \in \mathcal{H}_i} \hat{R}(h)$, the ERM for class i.

2. Compute $\hat{i} = \text{argmin}_{i \in \mathbb{N}} R(\hat{h}_i) + \epsilon_i(n, w(i)\delta)$.

3. Return $\hat{h}_{\hat{i}}$.

The procedure does something fairly obvious. First compute the ERM for every class, then find the class that has the best empirical risk, and has a reasonable generalization bound. Then return the ERM for that class. This procedure has a very nice property in that it is simultaneously competitive with all of the hypothesis classes. In statistics this is known as an oracle inequality.

Theorem 4. *For any δ and any distribution $\mathcal{D}$, with probability $1-\delta$ the output of the SRM procedure satisfies*

$$R(\hat{h}_{\hat{i}}) \le \min_{k \in \mathbb{N}} \left\{ \min_{h \in \mathcal{H}_k} R(h) + 2\epsilon_k(n, w(k)\delta) \right\}$$

Proof. First by Lemma 3 we know that with probability at least $1-\delta$

$$\forall k, h \in \mathcal{H}_k, |R(h) - \hat{R}(h)| \le \epsilon_k(n, w(k), \delta)$$

Now fix some class k and let us consider the excess risk between $\hat{h}_{\hat{i}}$ and $h_k^\star = \text{argmin}_{h \in \mathcal{H}_k} R(h)$.

$$\begin{aligned} R(\hat{h}_{\hat{i}}) - R(h_k^\star) &= R(\hat{h}_{\hat{i}}) - \hat{R}(\hat{h}_{\hat{i}}) + \hat{R}(\hat{h}_{\hat{i}}) - \hat{R}(h_k^\star) + \hat{R}(h_k^\star) - R(h_k^\star) \\ &\le \epsilon_{\hat{i}}(n, w(\hat{i})\delta) + \epsilon_k(n, w(k)\delta) + \hat{R}(\hat{h}_{\hat{i}}) - \hat{R}(h_k^\star). \end{aligned}$$

This bound is what we usually do, but we have this extra $\epsilon_{\hat{i}}$ term, which could be much bigger than what we want. We also have this difference in empirical risks, and to handle this we'll have to use the second minimization in the SRM. We know that $\hat{R}(h_k^\star) \ge \hat{R}(\hat{h}_k)$ but moreover we know that $\hat{i}$ was chosen to minimize a particular bound! This gives

$$\hat{R}(\hat{h}_{\hat{i}}) - \hat{R}(h_k^\star) \le \hat{R}(\hat{h}_{\hat{i}}) - \hat{R}(\hat{h}_k) \pm \epsilon_{\hat{i}}(n, w(\hat{i})\delta) \pm \epsilon_k(n, w(k)\delta) \le -\epsilon_{\hat{i}}(n, w(\hat{i})\delta) + \epsilon_k(n, w(k)\delta)$$

Combining this with above proves the final bound. $\square$

Let us instantiate the bound with VC classes. Recall that a VC class of dimension d satisifes the uniform convergence property with $\epsilon(n, \delta) = O(\sqrt{\frac{d \log(n/d)}{n}} + \sqrt{\frac{\log(1/\delta)}{n}}$. Above, If we set $w(k) = \frac{6}{\pi^2 k^2}$ then we get the oracle inequality

$$R(\hat{h}_{\hat{i}}) = \min_k \left\{ \min_{h \in \mathcal{H}_k} R(h) + O\left(\sqrt{\frac{d_k \log(n/d_k)}{n}} + \sqrt{\frac{\log(k/\delta)}{n}} \right) \right\}$$

If d_k is growing (think $d_k = k$), then we basically pay nothing for the model selection component here. Thus we can basically do as well as if we knew the best class before hand. This is quite powerful.

Remark 5. *While SRM is a simple approach that is theoretically justified, you should always be wary when concentration inequalities are used algorithmically like they are here. The reason is that this bakes in worst case assumptions about the problem and prevents you from adapting to favorable conditions. For example, we can no longer exploit the $R(h^\star)$-style bound we proved in homework 1 using this algorithm (but you can exploit it with a different algorithm), or if at some i the problem is realizable, then you should be getting a $O(1/n)$ rate but it is hard to exploit that algorithmically.*

More generally, the performance of this algorithm is tied to the deviation bound that you can prove – if you can prove a better deviation bound then you get a better algorithm. This is unsatisfying since you'd like the algorithm and the analysis to be somewhat detached from each other.