STAT 383C: Statistical Modeling I **Fall 2015**

Lecture 8 — September 22

Lecturer: Purnamrita Sarkar *Scribe: Evan Ott (proofread by Phaneendra Rao)*

Disclaimer: These scribe notes have been slightly proofread and may have typos etc.

Note: The latex template was borrowed from EECS, U.C. Berkeley.

Table of Contents

8.1 Variable Selection

8.1.1 Linear Test Set Error

For linear models, we often have

$$R_{\text{test}} = R_{\text{test}} + 2k\sigma^2 \tag{8.1}$$

where k is the number of parameters we are estimating and σ^2 is the variance of the ys from

$$Y = X\beta + \varepsilon \tag{8.2}$$
$$\varepsilon_i \sim N(0, \sigma^2) \tag{8.3}$$

If we don't have a linear model like this, we will need a different form of the covariance function (as seen in Equation **??**), but sometimes we use the linear form anyway in practice for convenience (it is also generally still a reasonable estimate of the test set error).

8.1.2 Alternative Estimators

Other ways of estimating the test set error are:

$$\textbf{AIC:}\ -2\log\mathcal{L} + 2d \tag{8.4}$$
$$\textbf{Mallow's } C_p\textbf{:}\ \hat{R}_{\text{tr}} + 2k\hat{\sigma}^2 \ \text{ OR } \ \hat{R}_{\text{tr}} + 2k\hat{\sigma}^2 - n \tag{8.5}$$

Recall that $\hat{R}_{\text{tr}}$ is like the RSS and that we can create an unbiased estimate of $\hat{\sigma}^2$ from data:

$$\hat{\sigma}^2 = \frac{\text{RSS}}{n-k} = \frac{\text{RSS}}{n-p-1} \tag{8.6}$$

We also have the Bayesian information criterion:

$$\textbf{BIC:}\ -2\log\mathcal{L} + \log(n)\cdot d \tag{8.7}$$

BIC, then, has a much larger penalty on the number of dimensions as you increase the size of your data.

For Gaussian noise (as in Equation **??**), we can show that AIC is exactly equal to Mallow's C_p. [this is because the likelihood $\propto \exp\left(-\sum_{i=1}^{n}(y_i - x_i^\intercal\beta)^2\right)$ so the log likelihood $\propto$ RSS].

8.1.3 Best Subset

One problem these estimators help us solve is the **best subset** problem – finding the subset of the covariates that produces the smallest error. However, because there are 2^d potential subsets, this problem is in general intractable. Instead, we need to create a greedy algorithm to find an approximate (local) solution. The following are different algorithms to do this.

Forward step-wise selection

Here we train univariate models for each $X_1, X_2, ..., X_p$ and compute the AIC (or other estimator) for each. We select the model with the smallest AIC (call the corresponding covariate X_*). We now consider all possible bivariate models including X_* and continue this procedure until the AIC only goes up or does not change much.

Backward step-wise selection

This procedure is essentially identical to the previous one, except we will start with a full model (all covariates) and remove one at a time, selecting the best as we go.

Zheng-Loh model selection

Another way of doing this is to consider the z-scores of each covariate. This is the Zheng-Loh method.

We fit the full model (on $X_1, ..., X_p$) and for each one compute the corresponding z:

$$Y \sim N\left(X\beta, \sigma^2 I\right) \tag{8.8}$$

$$\hat{\beta} = \left(X^\intercal X\right)^{-1} X^\intercal Y \tag{8.9}$$

$$\hat{\beta} \sim N\left(\beta, \left(X^\intercal X\right)^{-1}\sigma^2\right) \tag{8.10}$$

and

$$z_i = \frac{\hat{\beta}_i}{\mathrm{SE}\left(\hat{\beta}_i\right)} = \frac{\hat{\beta}_i}{\hat{\sigma}\sqrt{(X^\top X)^{-1}_{ii}}} \tag{8.11}$$

We then order the z-scores:

$$\left|z_{(1)}\right| \geq \left|z_{(2)}\right| \geq \cdots \geq \left|z_{(p)}\right| \tag{8.12}$$

Now, to generate the best subset, we let RSS(j) be the RSS for the top j features (features with z-scores $z_{(1)}$—$z_{(j)}$), and we take the best value of j:

$$j^* = \arg\min_j \left(\text{RSS}(j) + j\hat{\sigma}\log(n)\right) \tag{8.13}$$

This is a greedy algorithm (we take the top j^* features) and penalizes large models (since j is the number of dimensions for a particular model and we multiply this by $\log(n)$ as in the BIC). This does save us quite a bit of computation time.

Note on $\log(n)$ terms

The $\log(n)$ factor in BIC and Zheng-Loh model selection often comes straight from the math to create each criterion but essentially is used to say that "you should have a simple model for large data" – that when you have a large amount of data, the correct model should be apparent and simple. It turns out that AIC (which does not include such a term) is actually not consistent, while BIC is. However, the Kullback-Leibler divergence of the model picked using AIC to the true model converges to zero. And, while BIC is indeed consistent, it often can create models that are far too simple given a large enough data set.

8.2 Recursive Least Squares

This is an example of an *online* algorithm where we maintain a current estimate of $\boldsymbol{\beta}$ based on the data seen so far and update this estimate based on each new datapoint. As it turns out, for linear regression we can do an exact update based on the beautiful Sherman-Morrison-Woodbury formula.

Recall that the matrix product $\boldsymbol{X}^T\boldsymbol{X} = \sum_{i=1}^n \boldsymbol{x}_i\boldsymbol{x}_i^T$, where $\boldsymbol{X}$ is of dimension $n \times k$. If we add one new column $\boldsymbol{x}_{n+1}$, then

$$\boldsymbol{X}_{new}^T\boldsymbol{X}_{new} = \sum_{i=1}^n \boldsymbol{x}_i\boldsymbol{x}_i^T + \boldsymbol{x}_{n+1}\boldsymbol{x}_{n+1}^T = \boldsymbol{X}\boldsymbol{X}^T + \boldsymbol{x}_{n+1}\boldsymbol{x}_{n+1}^T$$

Here $\boldsymbol{X}_{new}$ is a matrix of size $n+1 \times 1+p$ and is a *rank one update* of $\boldsymbol{X}$.

Theorem 8.1. *(Sherman-Morrison-Woodbury)*

$$(\boldsymbol{A} + \boldsymbol{x}\boldsymbol{x}^T)^{-1} = \boldsymbol{A}^{-1} - \frac{\boldsymbol{A}^{-1}\boldsymbol{x}\boldsymbol{x}^T A^{-1}}{1 + \boldsymbol{x}^T A^{-1}\boldsymbol{x}}$$

Recall that $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$.
Let

$$\begin{aligned}
\boldsymbol{P}_n &= (\boldsymbol{X}_{n\times k}^T\boldsymbol{X}_{n\times k})^{-1}. \\
\boldsymbol{q}_n &= \boldsymbol{X}_{n\times k}^T\boldsymbol{y}_{n\times 1} \\
\hat{\beta}_n &= \boldsymbol{P}_n\boldsymbol{q}_n
\end{aligned}$$

Now using the S-M-W formula and simple linear algebra:

$$\begin{aligned} \boldsymbol{P}_{n+1} &= \boldsymbol{P}_n - \frac{\boldsymbol{P}_n \boldsymbol{x}_{n+1} \boldsymbol{x}_{n+1}^T \boldsymbol{P}_n}{1 + \boldsymbol{x}_{n+1}^T \boldsymbol{P}_n \boldsymbol{x}_{n+1}} \\ \boldsymbol{q}_{n+1} &= \boldsymbol{q}_n + \boldsymbol{y}_{n+1} \boldsymbol{x}_{n+1} \end{aligned}$$

Note: the form of the S-M-W actually allows more than rank one updates, and so you can in principle load in a small batch of data-points and calculate the new estimate. However, the formula gets more complex, and requires more computation.

8.3 Cross-Validation

Given a full data set, we typically break it up into three pieces as seen before (see Table **??**).

Train	Validation	Test

Table 8.1. Example of partitioning data into a partition of disjoint subsets for training, validating the model, and the final test.

Since we only use the test set exactly once (otherwise it would not be valid since we would have used it in the construction of our model), we use the validation set instead to check as we try a variety of models as an estimate of the test set error. However, this could produce unnecessary bias (e.g., if the validation set is not representative of the data as a whole), so we often perform **cross-validation**. In this process, we split the non-test data (so, what would have been chosen as "Train" and "Validation" in Table **??**) into k equal-sized segments, as in Table **??**.

1	2	3	4	5

Table 8.2. Splitting up the non-test data into $k = 5$ sets. We'll select one at a time to use as our validation/test data for a model trained on the other 4 sets (here, we select set 1 as the test set).

We will do the normal procedure (train on the "Training" set, test on the "Validation" set), but we will train k different models. Each time, we will leave out one of the k sets of this partition, train on those that remain and test on the one we left out. To estimate the actual test set error, we will average over the error from each model.

Let K be our partition function that takes each element of the data and maps it to a particular set in the partition:

$$K : \{1, ..., n\} \to \{1, ..., k\} \tag{8.14}$$

We will also create the sets themselves for $b \in \{1, ..., k\}$:

$$C_b = \{\ i \mid K(i) = b\} \tag{8.15}$$

Now, let $\hat{f}^{-b}$ be the version of the model trained on the $k-1$ sets in the partition that do not include set b. This is convenient notation for us as it lets us write the form for the error (given loss function L) in relatively compact notation (the more verbose version is also included for comparison).

$$CV(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, \hat{f}^{-K(i)}(x_i)\right) \tag{8.16}$$

$$= \frac{1}{k}\sum_{b=1}^{k}\left[\sum_{i \in C_b} \frac{L\left(y_i, \hat{f}^{-b}(x_i)\right)}{n/k}\right] \tag{8.17}$$

For example, if we take L to be the squared error, we have

$$CV(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}^{-K(i)}(x_i)\right)^2 \tag{8.18}$$

Example for notation

As an example to practice using the notation above, let's say data point x_1 is chosen to be in set C_3 in the partitioned data set (see Table **??**). That would mean that $K(1) = 3$, and that $1 \in C_3$. As such, to create the prediction for what y should be at x_1, we would train our model $\hat{f}$ on $C_1 \cup C_2 \cup C_4 \cup C_5 = (\cup_{i=1}^{5} C_i) - C_3$, which we call $\hat{f}^{-3}$ since we are, in a sense, subtracting C_3 from our training set. So the predicted value for data point x_1 is $\hat{f}^{-3}(x_1)$. This is how we should read Equations **??**–**??**. We can generalize this term by incorporating K: $\hat{f}^{-K(1)}(x_1)$ or more generally $\hat{f}^{-K(i)}(x_i)$ $\forall i \in \{1, ..., n\}$.

C_1	C_2	C_3	C_4	C_5

Table 8.3. Example of a 5-fold partition.

8.3.1 LOOCV

In general, we use $k = 5$ or $k = 10$ as a rule of thumb. However, if we let $k = n$, this is **leave-one-out cross-validation** or **LOOCV**. (While in general, this is computationally terrible, it isn't so bad in our current example of a linear model with Gaussian noise $\rightarrow$ see the homework for this week for an exercise to show this)

In this case, the equation for the cross-validation error (which is an estimate for R_{test}), is

$$LOOCV(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n}\Big(y_i - \hat{f}^{-i}(x_i)\Big)^2 \tag{8.19}$$

because each data point will be in a set of the partition all on its own. This is very much like the training set error – each trained model is only leaving out one data point, so it is very nearly just a model trained on all the non-test data. As such, the bias will be almost identical to the bias of the training set. However, the variance will be high. Why? If we let CV_i be the error computed for data point x_i, then our total error is

$$LOOCV = \frac{\sum_{i=1}^{n} CV_i}{n} \tag{8.20}$$

but this is taking an average so if we compute the variance, we get:

$$\begin{aligned}\text{var}(LOOCV) &= \text{var}\left(\frac{1}{n}\sum_{1=1}^{n} CV_i\right) = \frac{1}{n^2}\text{var}\left(\sum_{1=1}^{n} CV_i\right) \\ &= \frac{1}{n^2}\left(\sum_{i}\text{var}(CV_i) + 2\sum_{i<j}\text{cov}(CV_i, CV_j)\right)\end{aligned} \tag{8.21}$$

But because each prediction $\hat{y}^{-i} = \hat{f}^{-i}(x_i) \approx \hat{f}(x_i) = \hat{y}$, the correlation between errors is high so the covariance terms will contribute significantly to the variance of the LOOCV estimate of the test set error.

This is why we usually use $k = 5$ or $k = 10$. The variance of our test set error estimate will be much smaller, but we do pay a penalty in the bias.

Learning Curve

Like we said before, LOOCV has low bias but high variance. But lets come to the point of bias. Often the error estimated using 5/10 fold CV is biased upwards. A simple way of understanding this is by thinking about the learning curve. If we plot $1 -$ error versus n (where our population is $n = 1000$), we might expect the following plot (see Figure **??**).

What this says is that if we only train on a fifth of the data, the slope of the learning curve is still high – this means that as we increase n we are getting still a lot of useful new information to reduce the error. As we approach n though, each additional data point does not decrease error very much – the slope is very small. If we use 2 fold cross validation, then our training data within CV wont give a very good model and as a result the prediction error will biased up.

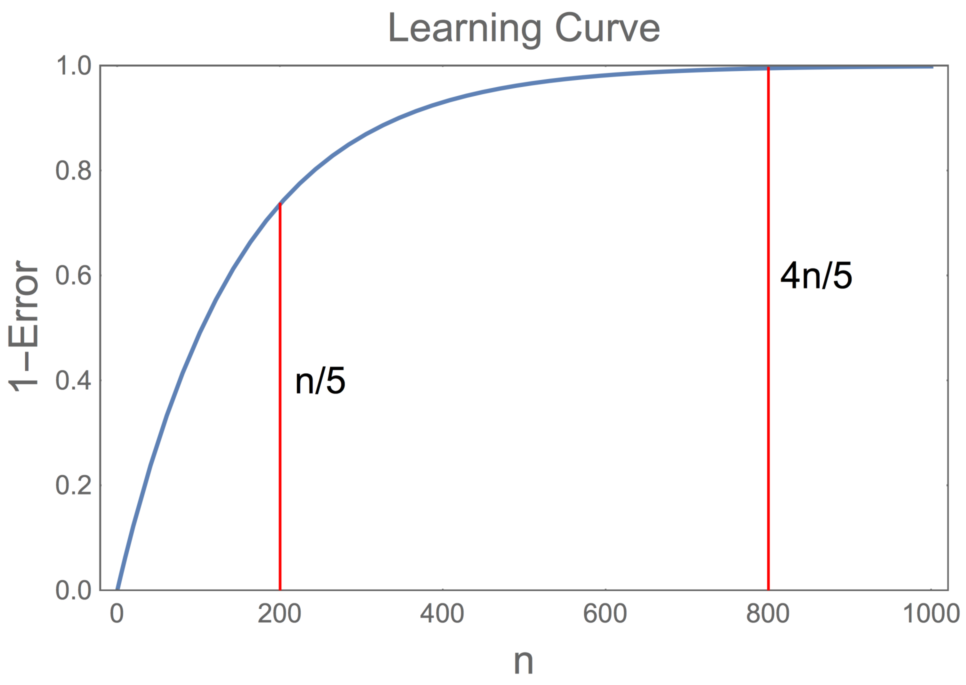

Figure 8.1. 1-error as a function of number of data points used in the training set (n for the population is 1000 in this image). This indicates the decrease in slope at $4n/5$ relative to $n/5$.

8.3.2 Estimating Test Set Error

We now must consider how to estimate the test set error using cross-validation. Say I have an algorithm which picks the 100 most correlated features with the response and then trains LS with those features. I want to find out the prediction error of this method.

Here's one potential algorithm for doing so (think about it as you read):

1. Pick the best features (say u), taking the (for example) 100 most correlated with y.
2. Divide the data into k chunks, report the k-fold cross-validation error on all these covariates

Does this work? **No!** You've already looked at all the data to develop your choice of covariates – that means you cheated. Instead, we must do it this way:

1. Divide data into k chunks and use cross-validation
2. For each model trained on $k - 1$ chunks, pick out the best covariates
3. Take the average of the errors for each trained model and report this as your estimated test set error

The point is that you cannot use the response variables in the test data to pick the best covariates first and then do cross validation which uses the test data for calculating prediction error. This is cheating. However if you wanted to clean the data where you look only at the X's that you can do before CV.

At this point a big confusion is: but now I have 10 models, since each of the folds picked out different sets of covariates. What do I do with all these different models? Do I somehow average them? Do I pick the best one? Does it matter?

We'll potentially talk about this another time, but essentially the answer is — it does not matter. We are not yet doing model selection— we are just estimating the prediction error of our method using CV. More in the next class.