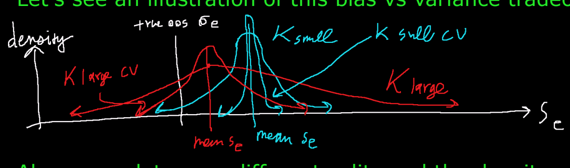
How do we pick K? Is there a tradeoff between small and large K values? There is one main tradeoff:

(1) If K is large => n_test is small => oos est. of performance is highly variable because its an estimate with very little data (principle from Stat 101). But n_train is almost n which means the oos est. of performance is not as biased.
(2) If is K is small => n_train is small => oos est. of performance will be biased in the direction of bad performance because n_train < n and the final model will be trained on all n. But oos est. of performance is less variable.

Let's see an illustration of this bias vs variance tradeoff:



Above we plot many different splits and the density of the resulting oos estimate.

What values can K take to get even splits?

$$\frac{1}{K} := \frac{n_{test}}{n}, \quad n_{test} \in \left\{1, 2, \ldots, \frac{n}{2}, \ldots \frac{n}{3}, \ldots, \frac{n}{5}, \ldots, n-1\right\}$$
$$\Downarrow \; K = n / n_{test}$$

But n_test cannot be 0 or n because then there wouldn't be a training/test split.

$$K \in \left\{n, \frac{n}{2}, \frac{n}{3}, \ldots 10, \ldots, 5, 3, 2\right\}$$

most common values

What situation does K = n represent? n_test = 1 and n_train = n-1! In this case it is obvious you must cross validate! This type of cross validation is called "leave one out cross validation" (LOOCV). Downside: highly variable and highly computationally expensive.

I don't believe there is any theory on the "optimal K". I believe it's greater than 2 and less than n. This is why defaults are 3, 5, 10.

Given the sam p raw features, there are (1) many transformations and interactions where one can augment the design matrix, (2) many different algorithms. So you get many different models. Let's call the number of models M:

$$g_1(\vec{x}), g_2(\vec{x}), \ldots, g_M(\vec{x})$$

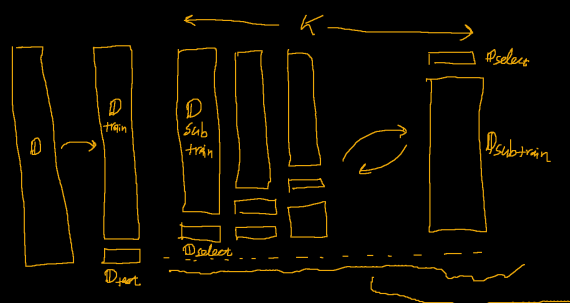
Since all models for real phenomena are "approximate", there is no "correct" model. But you still need to choose one to use! This is the fundamental problem in science and statistics of "model selection". There are whole textbooks on this problem.

For example, with just p = 1 and just OLS:

$$g_1(x) = b_0 + b_1 x$$
$$g_2(x) = b_0 + b_1 x + b_2 x^2$$
$$g_3(x) = b_0 + b_1 \ln(x)$$
$$\vdots$$

The model section procedure we'll discuss is merely to pick the one with lowest oos error. This is an obvious idea.

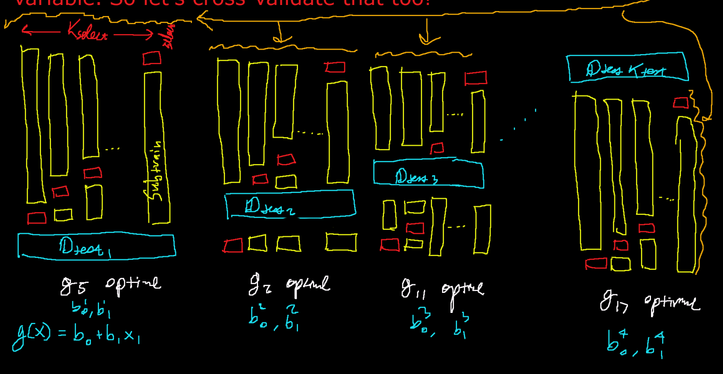
The K should be picked carefully and you should cross validate. Why? If the oos error is very variable and you're picking the model with the mininum value... you may just picking a random model.



In the D_subtrain, you fit / train / build all M models. Then you access the errors of all M models on D_select. (D_select is kind of like the D_test for the model selection only). Then you do the K-fold split crossing and assess the error of each of the M models. Then you pick the one with the best performance (m_*).

To get an estimate of the m_* model you picked, you can fit it on all D_train and then assess its oos performance on D_test.

But the oos estimate you get from one D_test can be highly variable. So let's cross-validate that too!



This is called "nested resampling".

How to pick K_test and K_select? No one knows. It's the same bias-variance tradeoff consideration but for different decisions.

In each of the outer resamplings we locate a different m_*! So what does the final oos error really mean? It is the error on the "meta-algorithm" that selects the best of M models. That's a very fancy algorithm curlyA.

How do you compute g_final? I'll just run the model selection procedure on all of D. It will pick one m_* and that will be g_final.

This model selection procedure has many uses. We just discussed one i.e.

(I) Among M models, select the best one.

We will study two more:

(II) Greedy Forward Stepwise Model Building.

Step A: Using the p_raw features, build a transformed set of features. These can be polynomials, logs, interactions, etc. For example:

$$x_1, \ldots, x_p, x_1^2, \ldots, x_p^2, \ln(x_1), \ldots, \ln(x_p), x_1 x_2, \ldots x_1 x_p, x_2 x_3, \ldots, x_2 x_p, \text{etc.}$$

This set of p_tr features can be enormous! Much bigger than the n observations you have originally.

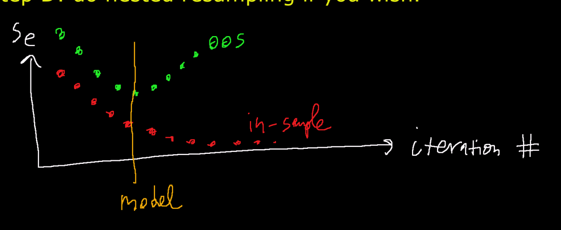
Step B: Begin with the model y = b_0 i.e. the null model.

Step 1: For each of the p_tr, try each adding each one to the model and see which one is the best according to in-sample performance (that's the greedy part). Then add it to the model.

Step 2: Using the model from step 1, return to step 1 and try every remaining feature and take the one provides the best gain in in-sample performance. STOP when the oos error (using D_select) goes UP!

Step C: compute oos error on the final model using D_test.

Step D: do nested resampling if you wish.



(III) Hyperparemeter selection

Recall the SVM with the Vapnik function: $E = AHE + \lambda \|\vec{w}\|^2$

How do we pick lambda??

Step 1: Create a grid of possible lambdas (hyperparameters) e.g.

$$\lambda_{grid} = \{.001, .01, .1, 1, 10, 100\}$$

Step 2: Each of the lambdas (hyperparameters) provides a different model. Now just use model procedure (I). Use nested resampling to pick the best model and provide an oos estimate of its performance.