$$y_i = g(\vec{x}) + \underbrace{\underbrace{h^*(\vec{x}) - g(\vec{x})}_{\text{estimation}} + \underbrace{\underbrace{h^*(\vec{x}) - f(\vec{x})}_{\text{misspecification}} + \underbrace{t(\vec{z}) - f(\vec{x})}_{\text{ignorance}}}_{\varepsilon(\vec{x})}}_{e(\vec{x})}$$

errors: estimation misspecification ignorance

$e_i := y_i - \hat{y}_i = y_i - g(\vec{x}_i)$ $i = 1 \ldots n$, rows of $\mathbb{D}$.

These e_1, ..., e_n are called "in-sample residuals" because they come from D (the sample). Thus, SSE, RMSE, R^2, SSR are all "in-sample".

Thus, they can all be "fake". Why? Because we saw that if p goes up, i.e. we make up new x-vectors, we can get any answer we want. We can make R^2 arbitrarily high and RMSE arbitrarily low. Thus, e_1, ..., e_n are not honest estimates of our prediction errors / prediction accuracy.

Imagine

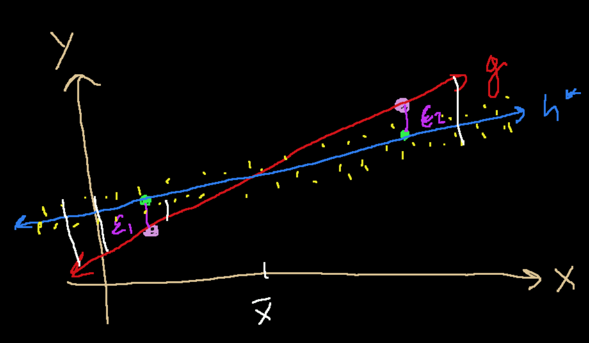
$$\begin{array}{ll} \mathbb{D} & \text{past} \\ \hline \mathbb{D}_* & \text{future} \end{array} \Big\downarrow \text{time}$$

A good way to honestly estimate prediction errors / prediction accuracy is to compute $\vec{e}_* = \vec{y}_* - \hat{\vec{y}}_*$ i.e. on data you did not use to build your model g.
$\hat{\vec{y}}_*$ cannot be overfit.

For this to be true, we need another assumption, "stationarity" i.e. that t(z-vec) stays the same and the relationships between the z's and the x's remain the same which means the function f doesn't change with time t. An example of a non-stationary relationship is stock prices explained by some variables z that are causal at one moment.

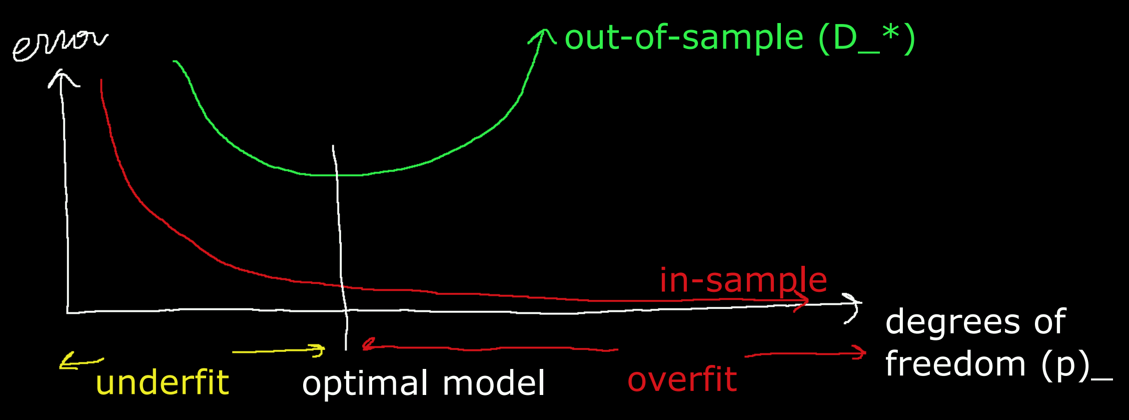
What is underfitting? Not learning as much as you can given your collection of x's.

Overfitting in one dimension (p=1) with two observations (n=2).



$y = \underset{\sim}{h^*(x)} + \underset{\sim}{\varepsilon}$

Overfitting is fitting the epsilons which you know is a bad idea since epsilon = misspecification error + error due to ignorance. Fitting any degree of epsilon leads to a poorer model. Overfitting doesnt' change h^* or f, it only changes g => it's error in estimation.



We need out of sample (honest) metrics:

$$\text{oosRMSE} = \sqrt{\frac{1}{n_* - (p+1)}\,\text{oosSSE}}$$

$$\text{oosSSE} = \sum_{i=1}^{n_*} e_{*_i}^2 \qquad = \sqrt{\frac{1}{n_*}\,\text{oosSSE}}$$

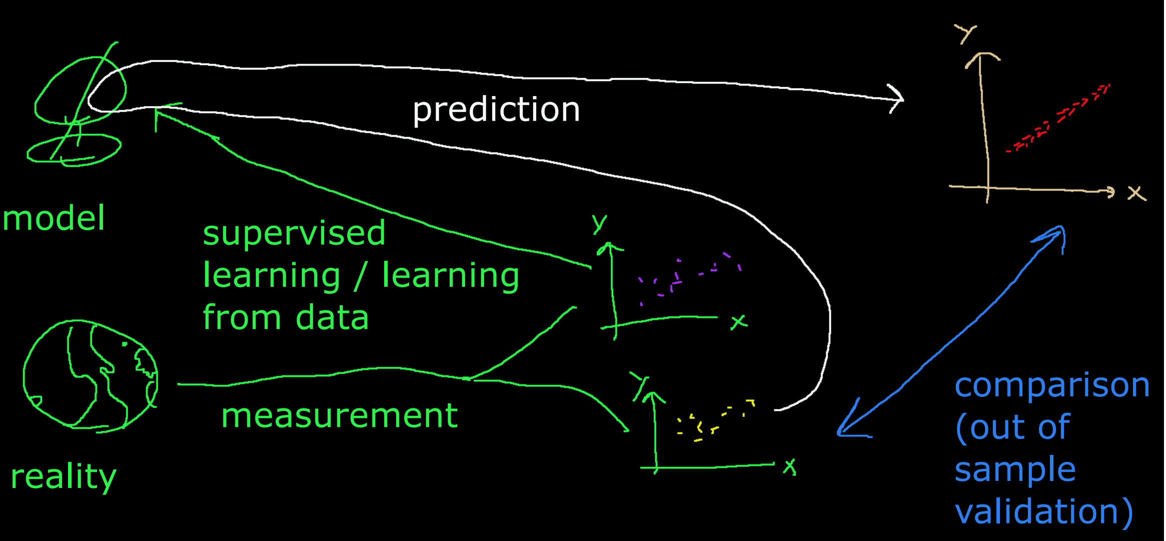
$$\text{oosR\^{}2} = 1 - \frac{\text{oosSSE}}{SST_*} \qquad SST_* = \sum (y_{*_i} - \bar{y}_*)^2$$

oos error metrics require computation on data from the future (which you don't have). So how can we possibly compute oos error metrics? So... assume stationarity and partition our original data set into:

$$\mathbb{D} = \mathbb{D}_{train} \cup \mathbb{D}_{test}$$

Then use D_train the way we've been using D all along which is use it to create your model: g = $\mathcal{A}(\mathbb{D}_{train}, \mathcal{H})$

and then use D_test to compute the honest oos error metrics to inform us of our model's predictive accuracy in the future
(i.e. give us an estimate of our generalization error). Two issues:
(1) what proportion of the data do you use for training?
(2) how to do the split?
(1) really has no answer but people generally use 80% or 90%.
The usual notation is to specify a K where 1 / K := proportion in the test set. So defaults are K = 5 or 10.
(2) usually done randomly in case there's a time trend in D.
n_train < n. Is there a cost to this? Yes, estimation error increases and thus, the generalization error estimate (oos error metrics) will look a tad worse than they will be when using your model (since the model you will use in the future will be built with all D, all n). So your oos error is conservative; the real oos error will be better.


prediction
model
supervised learning / learning from data
measurement
reality
y
x
y
x
Y
x
comparison (out of sample validation)

$h^*(\vec{x}) - g(\vec{x}) = X\vec{\beta} - X\vec{b} = X(\vec{\beta} - \vec{b})$.

Estimation error in OLS.
As b-vector diverges from the beta-vector, the estimation error goes up. One way to measure it is via $\|\vec{\beta} - \vec{b}\|^2$

In the above procedure, there is only one oos validation comparison. If you see that you overfit by seeing that e.g. RMSE << oosRMSE, it is too late! You can't correct the model honestly anymore. So... this is a big problem and we need to solve it.

But first, let's talk about reducing misspecification error.