IntroML - Lecture Notes Week 3

Ruben Schenk, ruben.schenk@inf.ethz.ch

April 6, 2022

1 Model Selection

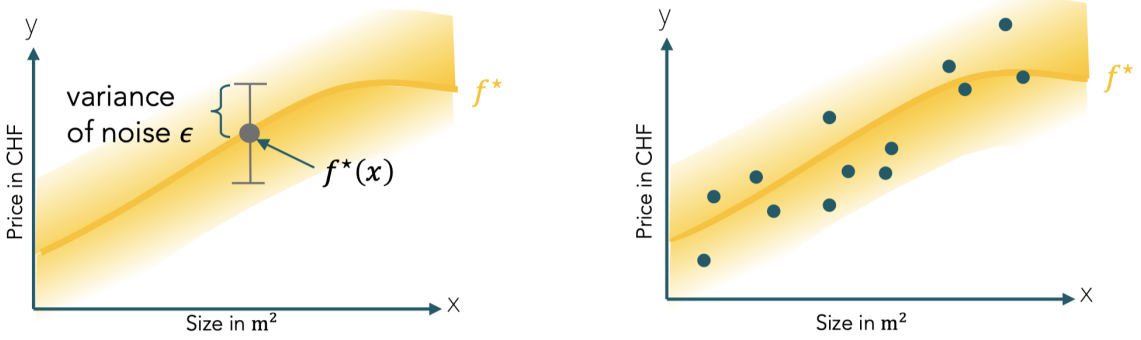
1.1 Introduction

1.1.1 Good Model via Ground Truth

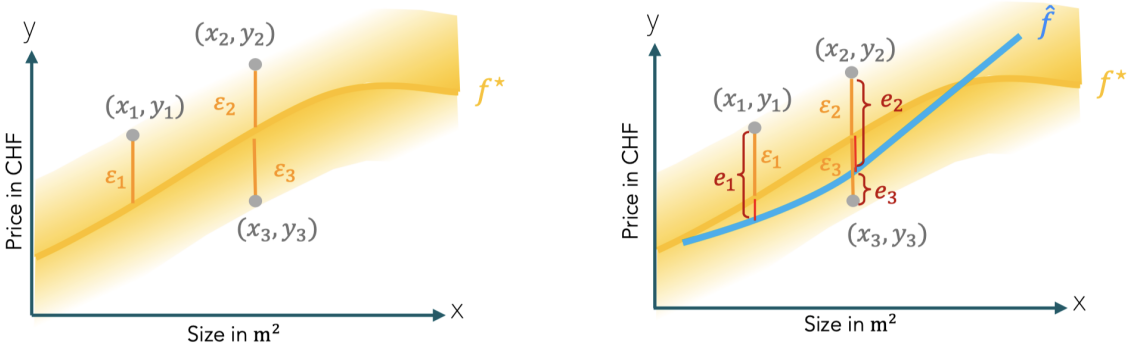
We formalize again that our goal is to get an intuition for how a sample size n can change the prediction performance. What is a good prediction? The *goal standard* is to assume the "true" average price is some *ground truth linear function* $f^*(x) = x^T w^*$. A *goo model* $\hat{f}$ is close to the real average f^*, i.e. has a low **estimation error** $l(\hat{f}(x), f^*(x)) = (\hat{f}(x) - f^*(x))^2$. The average error is then given by $\mathbb{E}_x(\hat{f}(x) - f^*(x))^2$ over all possible houses x weighted by how often they might appear. But we don't know f^*, so we can't compute it!

1.1.2 Prediction Error vs. Estimation Error

In fact, even though we want to predict the average price f^* or w^*, we never actually measure it! Instead, we usually only observe $y_i = f^*(x_i) + \epsilon_i = \langle w^*, x_i \rangle + \epsilon_i$ with noise ϵ_i.



Instead of the estimation error that is predicted compared to the average price $l(\hat{f}(x), f^*(x)) = (\hat{f}(x) - f^*(x))^2$, for each observed sample we can compute the **prediction error** $l(\hat{f}(x), y) = (\hat{f}(x) - y)^2$. For the following figure, we denote $e_i = |\hat{f}(x_i) - y_i|$:



Given the observations $y = f^*(x) + \epsilon = \langle w^*, x \rangle + \epsilon$ we exapnd the square to observations

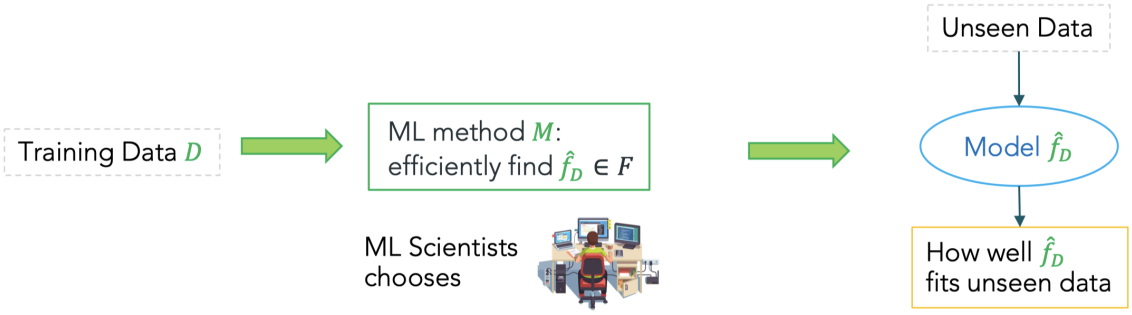
$$l(\hat{f}(x), y) = (\hat{f}(x) - f^*(x) - \epsilon)^2 = (\hat{f}(x) - f^*(x))^2 + \epsilon^2 - 2\epsilon(\hat{f}(x) - f^*(x)).$$

In fact, the **generalized error** (or average prediction error) is given by

$$R(\hat{f}) := \mathbb{E}_{x,y} l(\hat{f}(x), y) = \mathbb{E}_x l(\hat{f}(x), f^*(x)) + \text{irreducible noise}$$

1.2 Objective

We again remind us about the previously shown pipeline:



A more precise notation would be: A **model** $\hat{f}_D = M(D)$ is obtaine dby using a training method M on data set D. The method M could be, for example, minimizing a particular training loss on D using a specified optimization algorithm.

1.3 Approximating Generalization Error

1.3.1 Training Loss

So far, our methods found a minimizer of the training loss $\hat{f}_D = \arg\min_{f \in F} L(f; D) = \frac{1}{n} \sum_{i=1}^n l(f(x_i), y_i)$. However, the training loss $L(\hat{f}_D, D) = \frac{1}{n} \sum_{i=1}^n l(\hat{f}_D(x_i), y_i)$ is in general a *too optimistic* approximation of the generalization error $R(\hat{f}_D) = \mathbb{E}_{x,y} l(\hat{f}_D(x), y)$.

1.3.2 Simple Train vs. Test Split

We would like a proxy for the average error on the data that was "unseen" for $\hat{f}_D$. What we did so far is bad: we were using the training data for both training and evaluation!

The ideas is to *hold out* a part of the available data (called **held-out or test data** D'') and only train on the rest $D = D_{full} - D''$.

Thene, during testing, we approximate the generalization error $\mathbb{E}_{x,y} l(\hat{f}_D(x), y)$ by computing the average **test error** $\frac{1}{|D''|} \sum_{(x,y) \in D''} l(\hat{f}_D(x), y)$ evaluated on the test set D''.

1.3.3 Cross-Validation

In general we end up with a model selection problem. Consider the scenario for regression where we have no prior knowledge that informs us which feature map ϕ to use. Now we also need to determine which feature map ϕ to use!

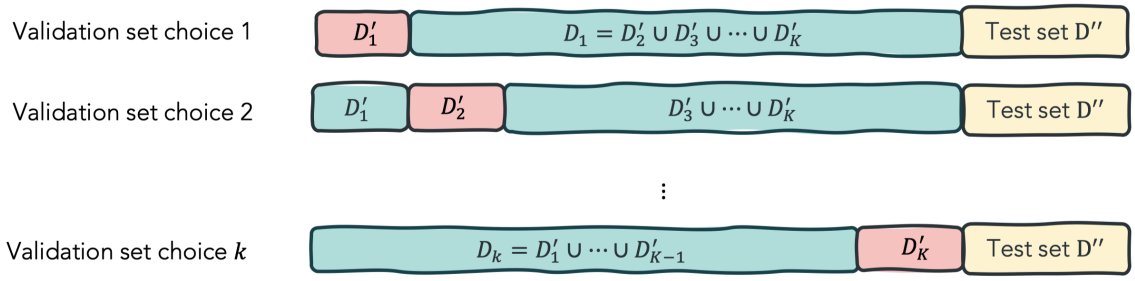
The solution is similar to the previous subchapter. We get more stability if we split D' again in to two sets and then use one to choose the model and one to test the model. I.e., we end up with the following split:

- Training set D, usually 80% or 50%

- Validation set D', usually 10% or 25%
- Test set D'', usually 10% or 25%

Often, the dataset is quite small and it's wasteful to set aside both hold-out validation set and the test set. Instead, with test set D'' fixed, for the rest we choose different validation sets and then average the validation errors.

K-fold cross-validation: The idea from above is formalized in the **k-fold cross-validation split**. We first split D_{full} into D_{rest} and the test set D''. We then split the rest data into K same size validation subsets D'_k. We then perform k splits where the validation set is D'_i and the training set is $D_i = D_{rest} \setminus D'_i$ for $i = 1, ..., k$.



This results in the following algorithm:

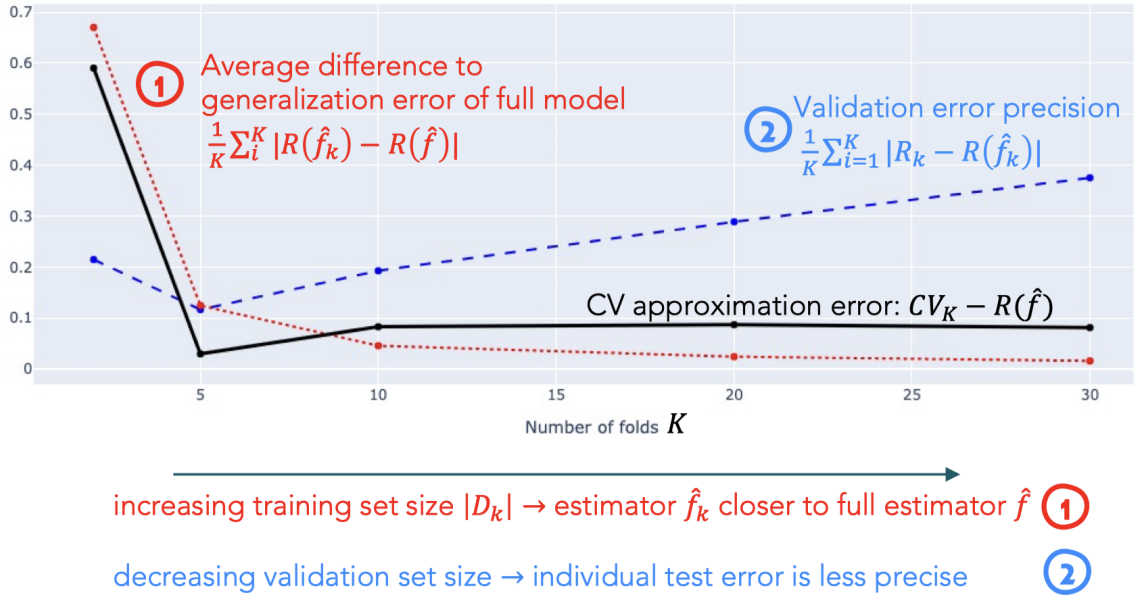
Algorithm: Given a choice of features ϕ, and K folds, do the following steps:

1. For all folds $k = 1, ..., K$
   (a) Compute $\hat{f}_k^{\phi} = M_{\phi}(D_k)$ that minimized the training loss on D_k
   (b) Compute the validation error on fold k: $R_k(\phi) = \frac{1}{|D'_k|} \sum_{(x,\, y) \in D'_k} l(\hat{f}_k^{\phi}(x),\, y)$
2. Compute the **cross-validation error** $CV_K(\phi) = \frac{1}{K} \sum_{i=1}^{K} R_k(\phi)$
3. Model selection: Pick feature ϕ^* with the lowest CV error $CV_K(\phi)$
4. Model training: compute the final model $\hat{f} = \hat{f}^{\phi^*} = M_{\phi^*(D_{rest})}$
5. Model evaluation: estimate generalized error of $\hat{f}$ using D'' (i.e. D_{test})

Note: This general framework works for comparing any methods M and losses l. The aim of the model selection is to find a method such that the generalized error $R(\hat{f})$ is small!

Model selection using CV can work if the CV error of methods M_{ϕ} is close to the generalization error of the method applied to the full dataset. If K is very large, e.g. $K = |D_{rest}|$ (*Leave-one-out CV or LOOCV*), we can get the best approxiamtion of $M_{\phi}(D_{rest})$, since each model $M_{\phi}(D_k)$ computed in the i-th split is very similar to $M_{\phi}(D_{rest})$. However, this is very computationally intensive, since we need to fit $|D_{rest}|$ models per method. In practice we typically choose $K = 5$ or $K = 10$.

Effects of the choice of k: The effect of the choice of K for a fixed method is given in the figure below:



Furthermore, we provide the following notation cheatsheed,for simplicity omitting the dependence on ϕ:

- Validation error on k-th split:

$$R_k = \frac{1}{|D_k'|} \sum_{(x,\, y) \in D_k'} l(\hat{f}_k(x),\, y)$$

- Cross-validation error

$$CV_K = \frac{1}{K} \sum_{i=1}^{K} R_k$$

- Generalization error for any function f

$$R(f) = \mathbb{E}_{x,\, y} l(f(x),\, y)$$

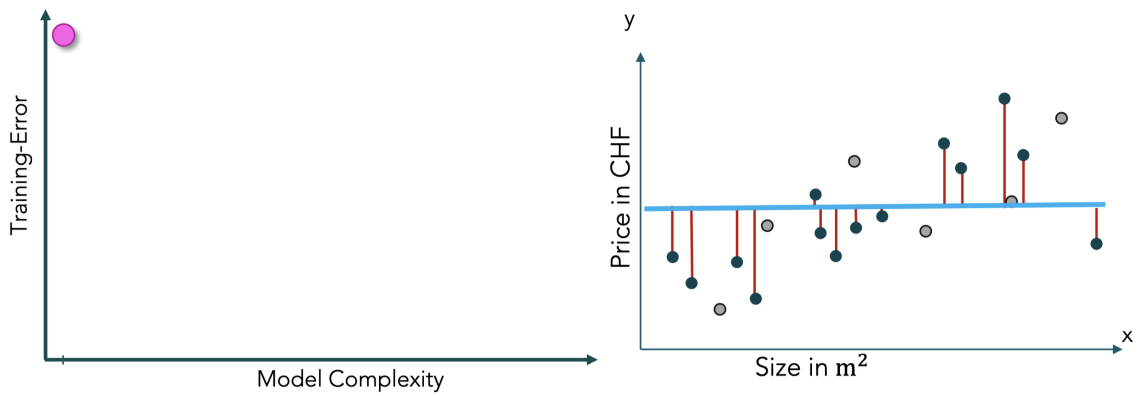
- Full estimator

$$\hat{f} = \arg\min_{f \in F} \sum_{(x,\, y) \in D_{rest}} l(f(x),\, y)$$

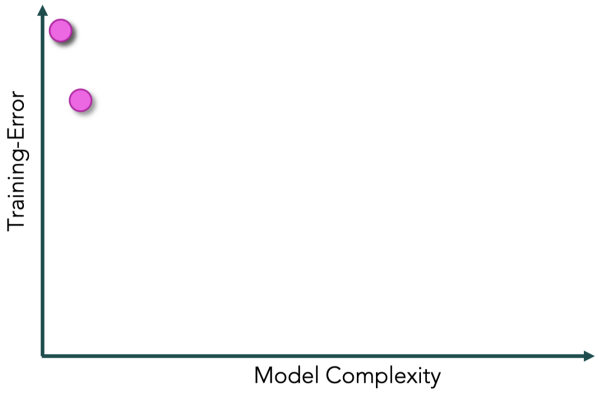
1.4 Increasing Model Complexity

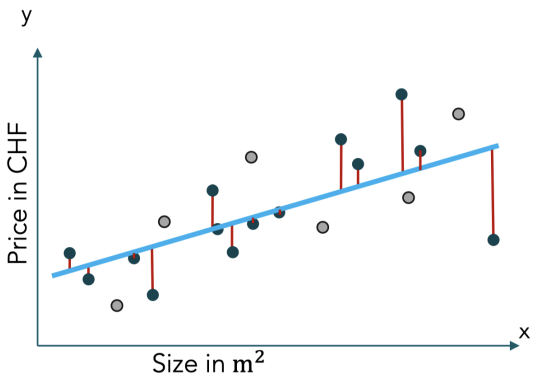
1.4.1 On The Training Error

Simplest function (constant):

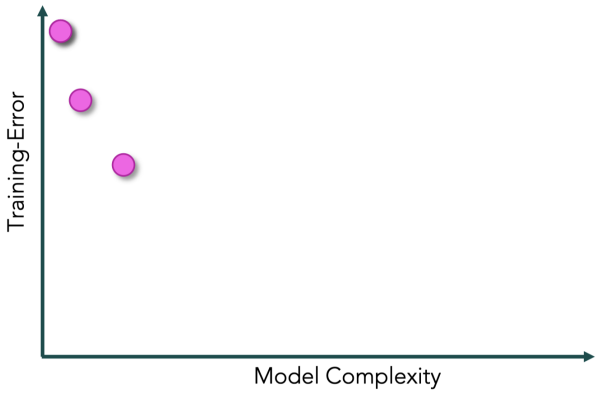


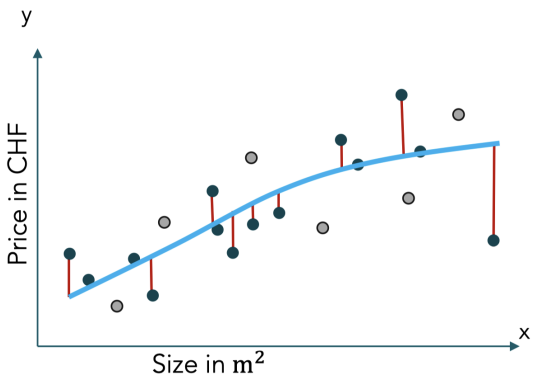
Simple function (linear):



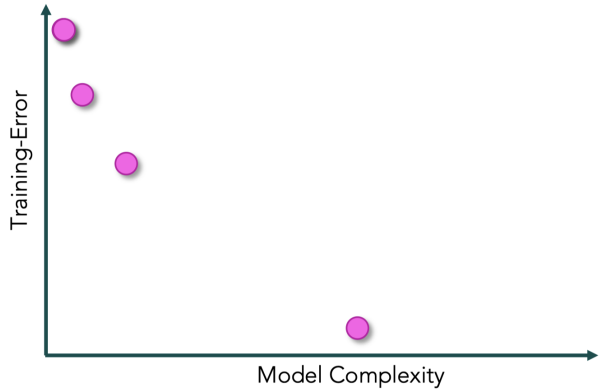


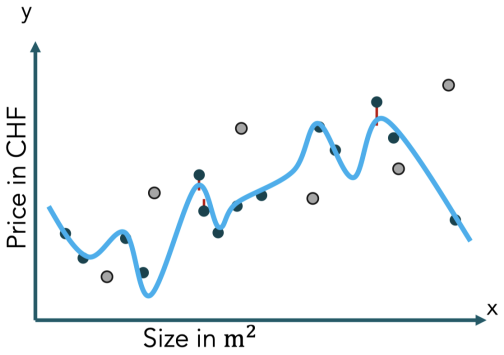
Medium simple function:



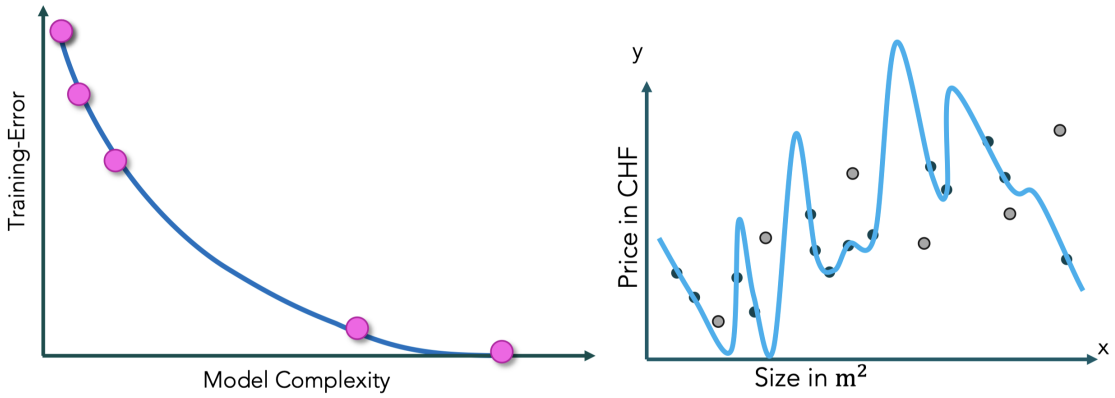


Complex function:



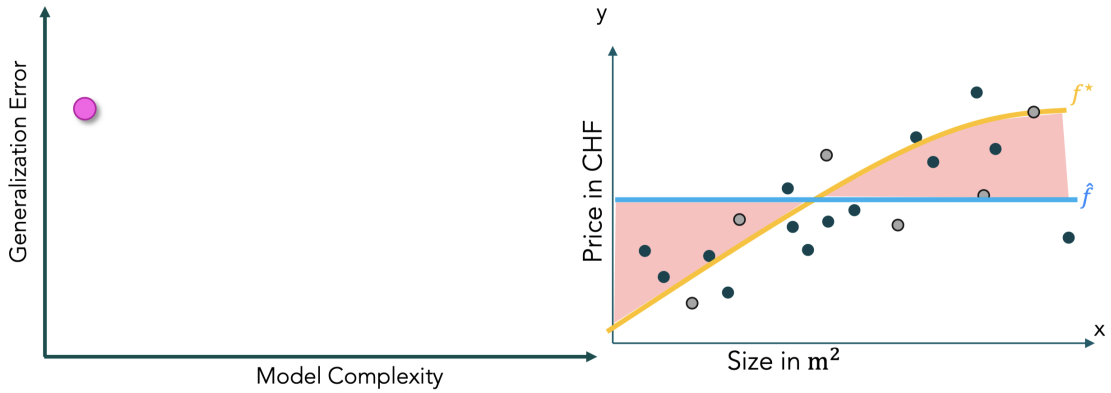


Very complex function:

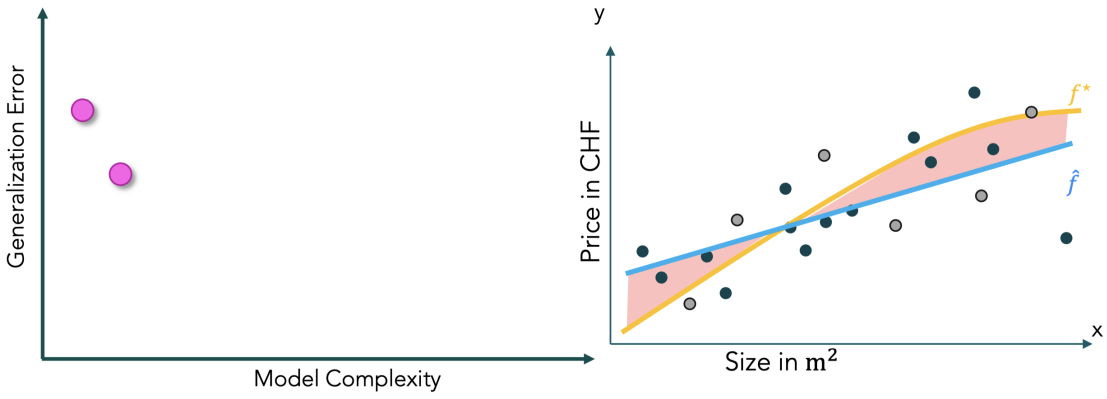


1.4.2 On The Generalization Error

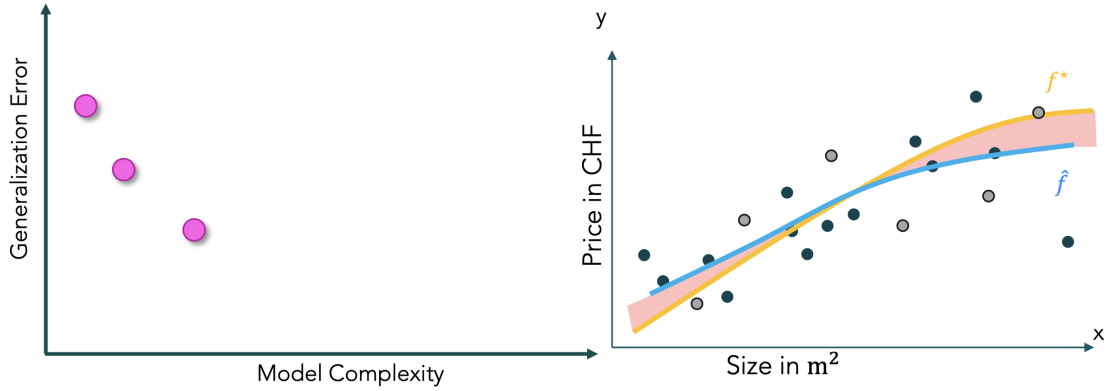
Simplest function (constant):



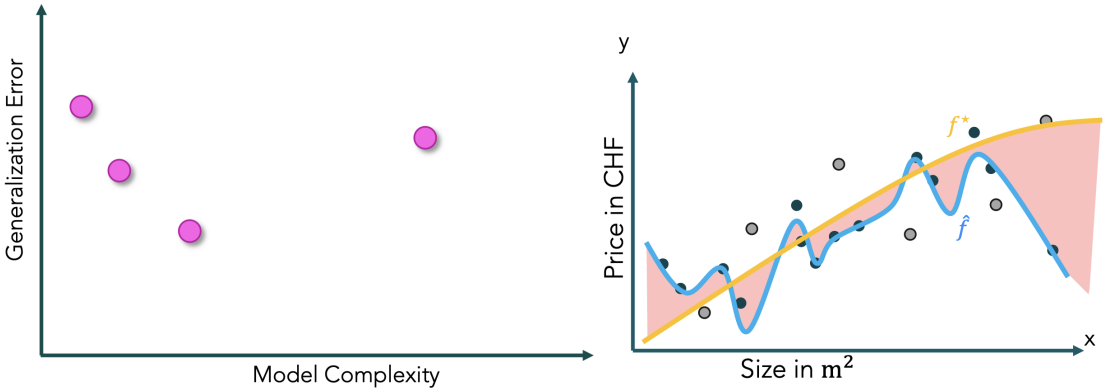
Simple function (linear):



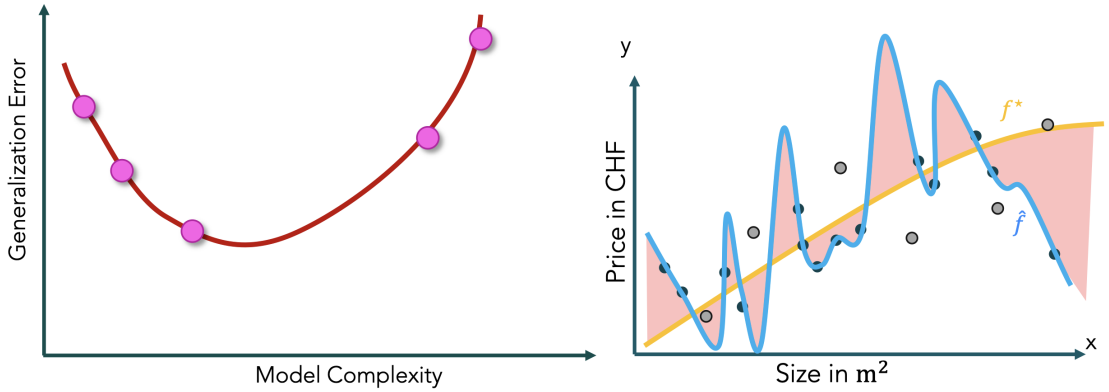
Medium simple function:



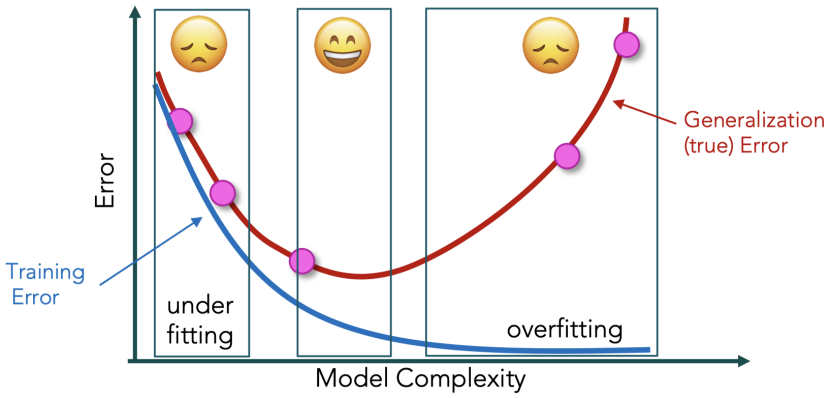
Complex function:



Very complex function:



1.4.3 Phenomenon of Under- and Overfitting



In summary:

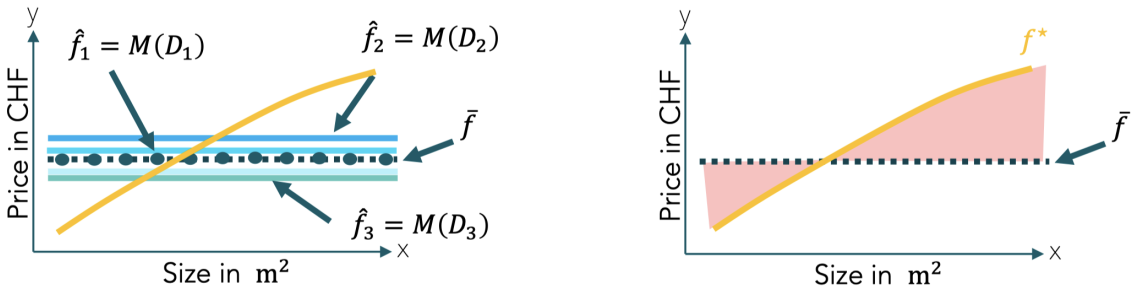
- Underfitting models predict: training data not well, and test data not well
- The "right" model predicts: training data well, and test data well
- Overfitting models predict: training data very well, test data not well

2 Bias-Variance Tradeoff & Regularization

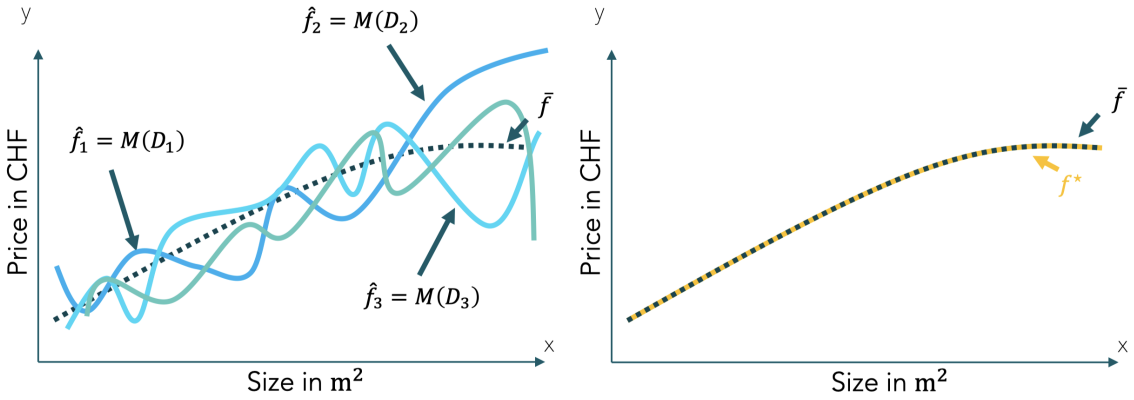
2.1 Bias-Variance Decomposition

2.1.1 Bias

The **bias** of some method M is the distance of the average model $\bar{f} = \frac{1}{K}\sum_{k=1}^{K} \hat{f}_k$ to the ground truth $\mathbb{E}_x(f^*(x) - \bar{f}(x))^2$.

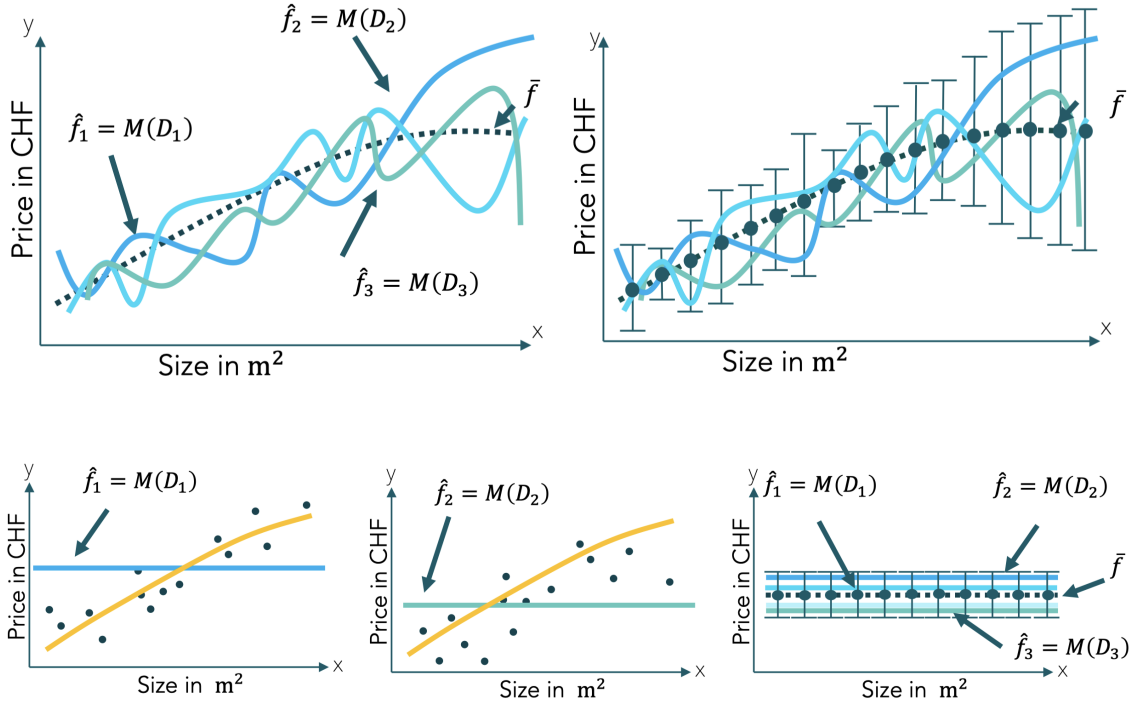


It is important to note that complex functions can model the ground truth well on average, as shown by the following figure:



2.1.2 Variance

The **variance** of some method M is the average distance of individual models to the average model, i.e. $\mathbb{E}_x \frac{1}{K} \sum_{k=1}^{K} (\hat{f}_k(x) - \bar{f}(x))^2$. For complex models, the varaince is large, whereas it is small for simple models:



2.1.3 Conclusion

In summary:

- Bias = "average - ground truth"
  - Drives Underfitting
  - Arises even for noiseless data
- Variance = "individual - average"
  - Drives overfitting
  - Especially for complex models

2.2 Regularization

2.2.1 Different Methods

Given a setting with linear regression with $1d$ polynomial features $\phi(x) = (1, x, x^2, ..., x^m)$. We observe that complex models use too many features! But how can we force it to use fewer?

Ideally we have that $\arg\min_{w \in \mathbb{R}^d} ||y - \Phi w||^2$ is such that $||w||_0 \leq k$. I.e. we need to search for the best subset of size k, however this is combinatorial infeasible for large d.

The option we choose is to only use a few of these features and encourage sparsity via limiting the l_1-norm through **lasso regression:**

$$\arg\min_{w \in \mathbb{R}^d} ||y - \Phi w||^2 \quad s.t. \, ||w||_1 \leq R \Leftrightarrow \arg\min_{w \in \mathbb{R}^d} ||y - \Phi w||^2 + \lambda ||w||_1$$

Another option would be to choose models with limited l_2-norm through **ridge regression:**

$$\arg\min_{w \in \mathbb{R}^d} ||y - \Phi w||^2 \quad s.t. \, ||w||_2 \leq R \Leftrightarrow \hat{w}_\lambda = \arg\min_{w \in \mathbb{R}^d} ||y - \Phi w||^2 + \lambda ||w||_2^2,$$

where R is given and equivalent for some λ. The *closed-form solution* is simple by finding the stationary point: $\hat{w}_\lambda = (X^T X + \lambda I)^{-1} X^T y$

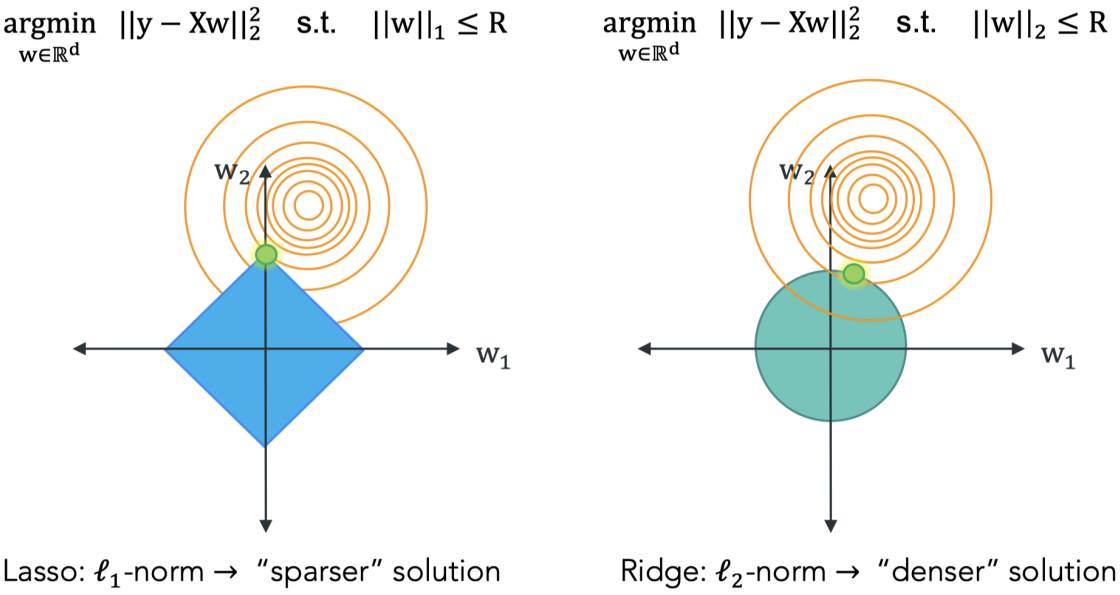
2.2.2 Intuition Behind the Lasso

The first inuition can be given with small calculations. Take for example $w_{sparse} = (1, 0, 0, ..., 0)$ vs. some dense vector $w_{dense} = \frac{1}{\sqrt{d}}(1, 1, ..., 1)$. For the same l_2 norm, the vectors with the smallest l_1 norm are sparse

$$||w_{dense}||_1 = \sqrt{d} >> ||w_{sparse}||_1 = 1,$$

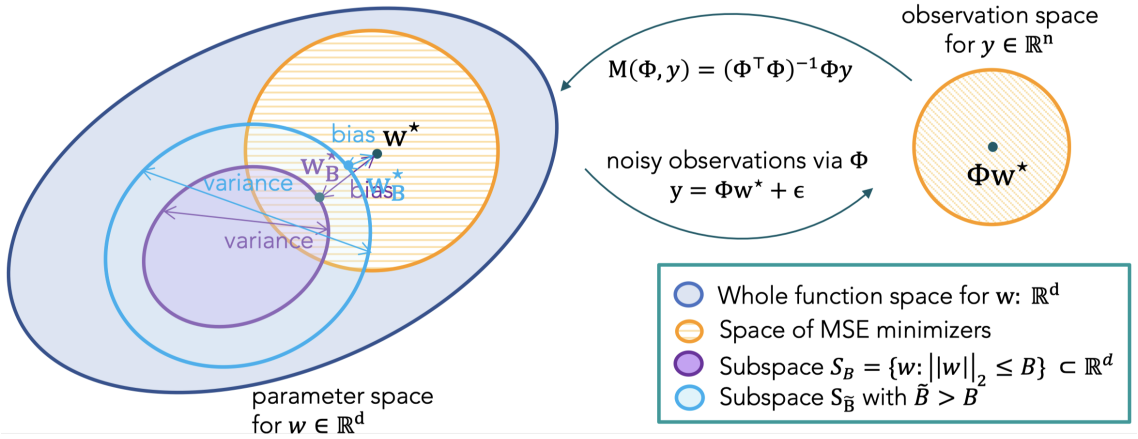
whereas vectors with the largest l_1 norm are dense

$$||w_{dense}||_1 = \sqrt{d} >> ||w_{dense}||_2 = 1.$$

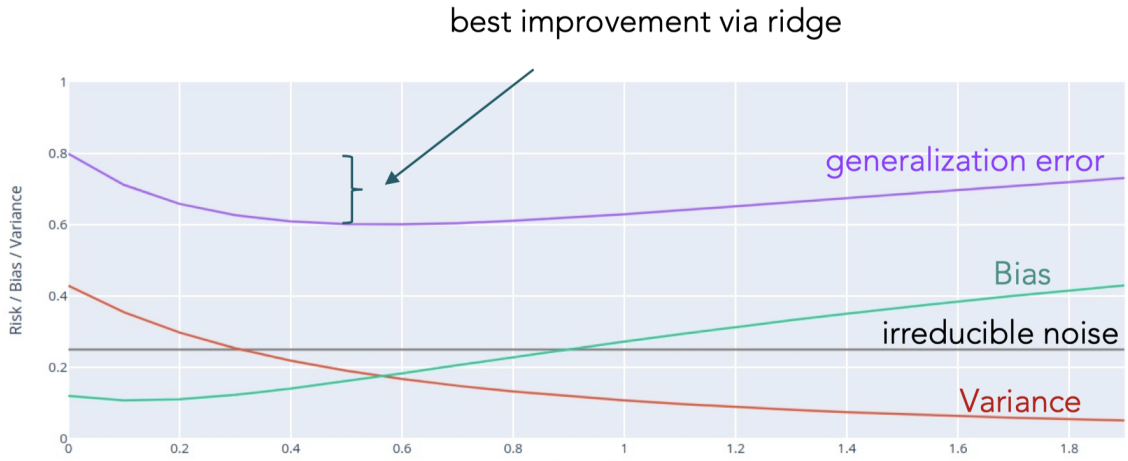


2.2.3 Bias and Variance as Function of Regularization Strength

The variance shrinks and the bias increases with model complexity. The following figure shows varying model complexity reflected via sizes of ellipses for $n > d$:



We might also display the bias and variance as a function of λ:



2.2.4 Cross-Validation to Determine Regulariztaion Strength

Instead of feature maps. the question now becomes how to optimally choose λ.

The regularized estimator $\hat{f}^\lambda = \langle \hat{w}_\lambda; \cdot \rangle = M_\lambda(D)$ is the minimizer of the regularized training loss $M_\lambda(D) = \arg\min_f L_0(f; D) + \lambda||f||$ with $L_0(f, D) = \frac{1}{|D|}\sum_{(x,y)\in D} l(f(x), y)$.

The algorithm is the same as before:

Algorithm: Given a choice of features λ, and K folds, do the following steps:

1. For all folds $k = 1, ..., K$
   (a) Compute $\hat{f}_k^\lambda = M_\lambda(D_k) = \arg\min_f L_0(f; D_k) + \lambda||f||$
   (b) Compute validation loss on fold k : $R_k(\lambda) = \frac{1}{|D_k'|}\sum_{(x,y)\in D_k'} l(\hat{f}_k^\lambda)(x), y$
2. Compute cross-validation error $CV_K(\lambda) = \frac{1}{K}\sum_{i=1}^K R_k(\lambda)$
3. Model selection: Pick λ with lowest error $CV_k(\lambda)$
4. Model training: Compute final model $\hat{f} = \hat{f}^\lambda = M_\lambda(D_{rest})$
5. Model evaluation: Estimate generalized error using D_{test}