Introduction to Overfitting.

Oriol Pujol

Introduction to Machine Learning

2014

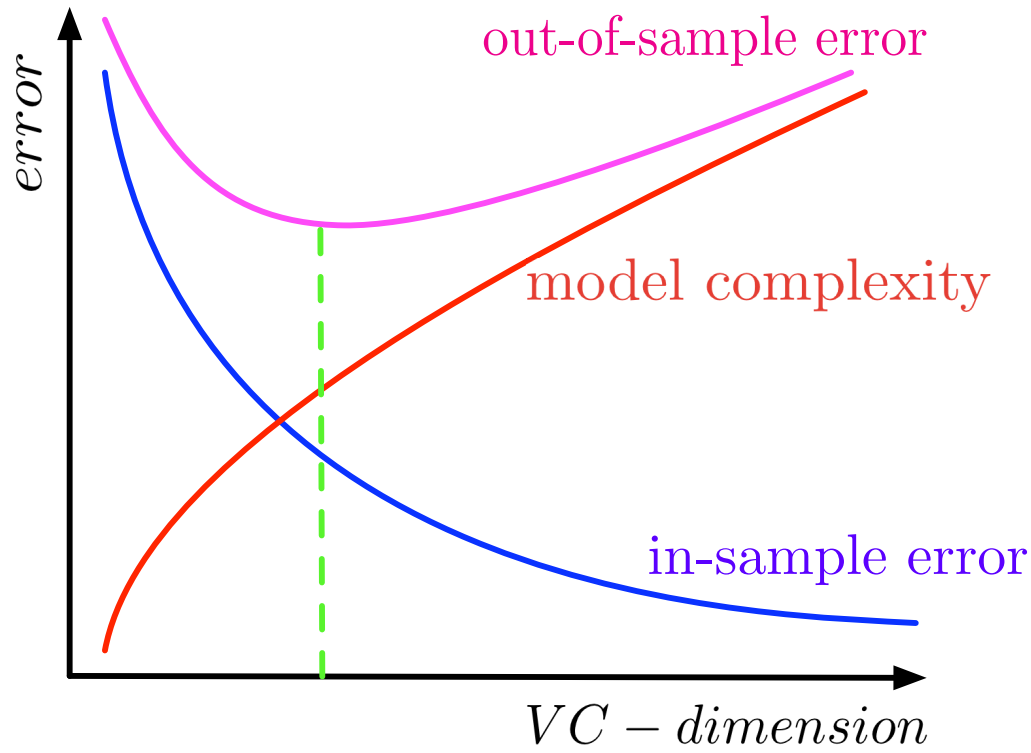
Acknowledgements

This work is inspired in the courses of T. Jaakkola, M. Collins, L. Kaelbling, and T. Poggio at MIT, Andrew Ng at Stanford, Y. Abu-Mostafa at CalTech, E. Xing at CMU, and all my mentors and people who made me realize Machine Learning is one of my passions.

Outline

- Non-linear models: decision trees
- Overfitting
- Bias and variance
- Regularization
- Validation

Learning theory.



Up to this moment, we have seen only linear models. This means very small complexity.

Our first non-linear model: Decision tree

Decision trees

Ad hoc classic model inspired in divide and conquer paradigm. The basic idea is to divide the input space and fit a model to each piece.
We have to answer two questions:

- How do we divide space?
- What piecewise model/value to use?

We will see

- Regression trees
- Classification trees

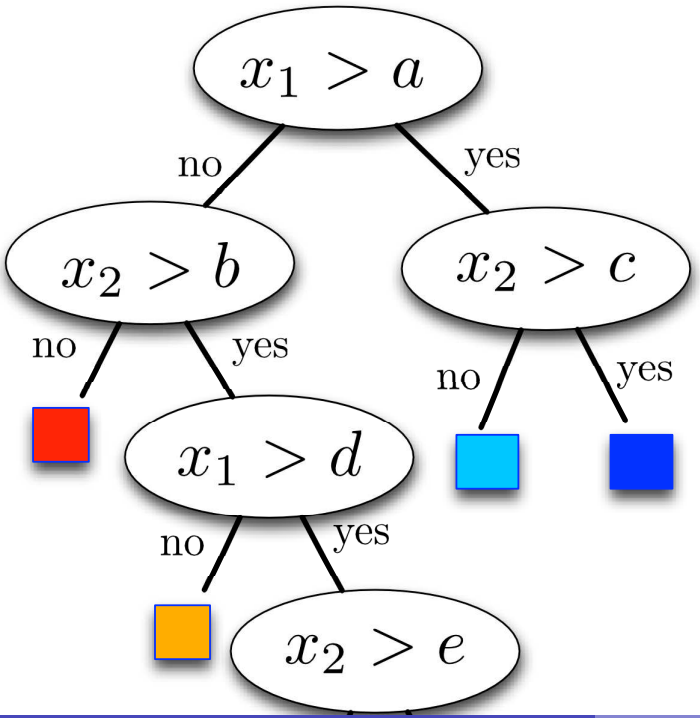
Regression trees

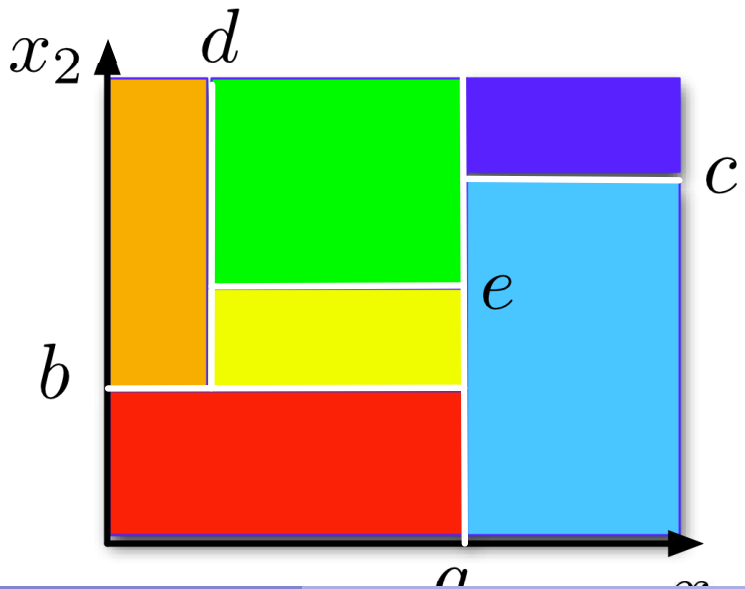
Regression trees

Goal: Approximate a general function according to the data using a piece-wise function.

Elements:

- Split using axis-parallel hyperplanes.
- Fit some function/value to the pieces (e.g. constant value).





Regression trees

Ideally, optimize splits to find the tree (T) that minimizes the overall fitting cost:

$$\underset{T}{\text{minimize}} \quad \sum_{i=1}^{N} (T(x_i) - y_i)^2 + (\lambda \cdot \#\text{nodes})$$

It is computationally intractable, so we use greedy approaches.

Regression trees

Ideally, optimize splits to find the tree (T) that minimizes the overall fitting cost:

$$\underset{T}{\text{minimize}} \quad \sum_{i=1}^{N} (T(x_i) - y_i)^2 + (\lambda \cdot \#\text{nodes})$$

It is computationally intractable, so we use greedy approaches.

Building the tree recursively:

Initialize: List of nodes to split with their corresponding data: $N = \{\mathcal{N}_{0,\mathcal{D}}\}$

1. Remove a node $\mathcal{N}_{\cdot,\mathcal{D}_\mathcal{N}}$ from N and split it – find j, s, the variable index and split point that

$$\underset{j,s}{\text{minimize}} \sum_{\mathbf{x}_u \in \{\mathbf{x} \in \mathcal{D}_\mathcal{N} | x_j \leqslant s\}} (y_u - \bar{y}(\mathbf{x}_u))^2 + \sum_{\mathbf{x}_v \in \{\mathbf{x} \in \mathcal{D}_\mathcal{N} | x_j > s\}} (y_v - \bar{y}(\mathbf{x}_v))^2$$

2. $N = N \bigcup \textbf{MakeNodes}(\mathcal{N}_{\cdot,\mathcal{D}_\mathcal{N}}, < j, s >)$
3. **while** $|N| \neq 0$: **go to 1**

Regression trees

Each call considers $\mathcal{O}(d \cdot N)$ splits each requiring $\mathcal{O}(N)$ work.
The stopping criterion is that $|\mathcal{D}_{\mathcal{N}}| < k$, where k is a small constant value or the approximation of the data for that node is "good enough".

Classification trees

Representation

We have a tree, where:

- At each internal node we test a value of one feature (binary test).
- The leaves make the class prediction.

What is great about decision trees?

- Trees are easy for humans to interpret.
- Decision trees can express any function of the input attributes. Red flag!!! Easy to over fit.
- There can be more than one tree that fits the same data.

Classification tree splitting criterion

Minimize a measure of impurity of the classification at the leaves.

Classification tree splitting criterion

Minimize a measure of impurity of the classification at the leaves.
Value at m leaf – the fraction of samples belonging to the majority class:

$$K(m) = \arg\max_k \hat{P}_{m,k} = \arg\max_k \frac{|\{y = k | (\mathbf{x}, y) \in \mathcal{D}_{\mathcal{N}_m}\}|}{|\mathcal{D}_{\mathcal{N}_m}|}$$

Measuring node impurity

- Misclassification error: $Q_m(T) = 1 - \hat{P}_{m,k(m)}$
- Gini index $\sum_k \hat{P}_{m,k}(1 - \hat{P}_{m,k})$
- Cross-entropy $-\sum_k \hat{P}_{m,k} log \hat{P}_{m,k}$

Information theory in a nutshell

Let us suppose we have a binary random variable (e.g. a biased coin toss), with probability distribution:

$P(X=0)$	$P(X=1)$
0.2	0.8

The *surprise/information* of each value of X is

$$S(X = x_i) = -\log_2 P(X = x_i)$$

This value (*surprise*) turns out to be equal to the number of bits of *information* that need to be transmitted to a recipient who knows the probabilities of the results.

Entropy

Given a discrete random variable $X \in \{x_1, \ldots, x_n\}$, the *entropy* of X ($H(X)$) is defined as

$$H(X) = -\sum_{i=1}^{n} P(X = x_i) \log_2 P(X = x_i)$$

Intuitively, this is the average surprise of one "trial" of X (one coin toss). Entropy can be viewed as a measure of uncertainty. In machine learning terms, "entropy characterizes the impurity of a an arbitrary collection of examples."

Entropy is the expected information . In the binary case we can characterize it using a frequentist approach: Let p_+,p_- be the proportion of positive and negative examples, respectively. The entropy of a set is computed as:

$$H(\mathcal{D}) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

Mutual Information

Information gain (aka mutual information) between two random variables X and Y, defined as

$$I(X;Y) = H(X) - H(X|Y)$$

is the reduction in entropy of one of the variables by knowing the other. In our case, we have two random variables, the outcome/class (Y) and the features (X). Thus, mutual information accounts for the reduction in the uncertainty of the class from knowing the value of a particular attribute, and vice-versa. (It is symmetric)
The conditional entropy $H(X|Y)$ of X given Y:

$$H(X|Y) = -\sum_{j \in \mathrm{Val}(y)} P(y=j) \log_2 H(X|y=j)$$

$$H(X|y=j) = -\sum_{i=1}^{N} P(x=i|y=j) \log_2 P(x=i|y=j)$$

Decision trees

In the binary case (p_+,p_- are the proportion of positive and negative examples, respectively):

$$H(\mathcal{D}) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

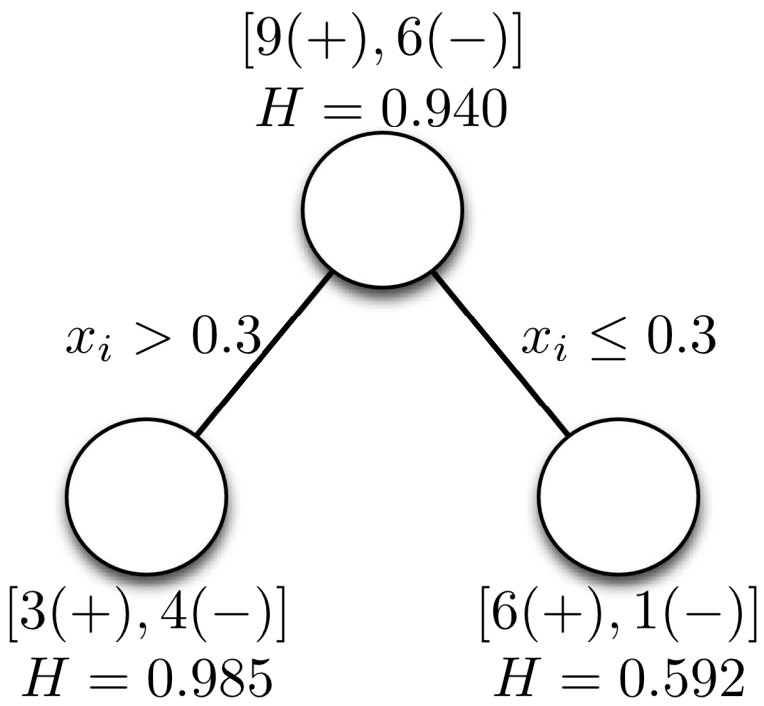
Information gain:

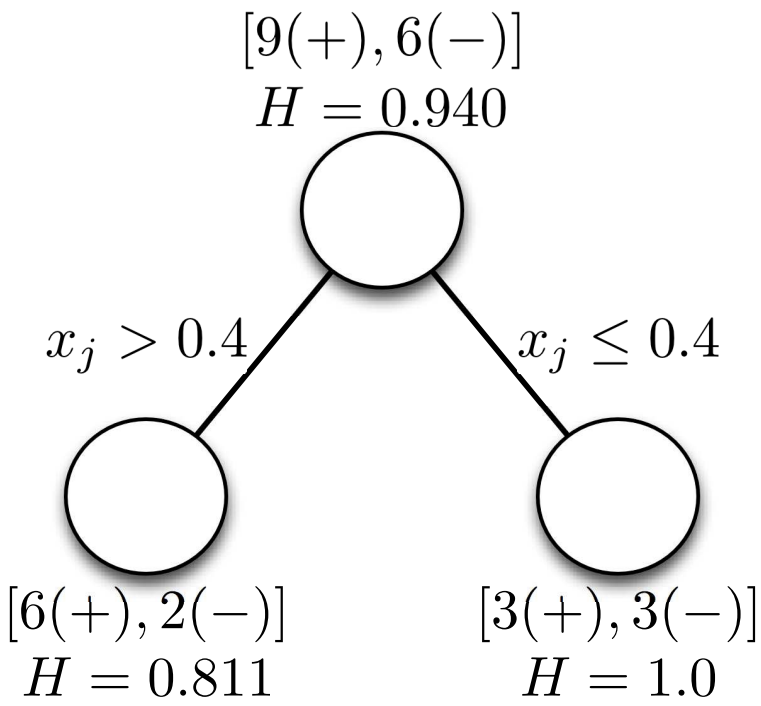
$$G = entropy(\mathcal{D}_p) - \sum_{i=1}^{k} \frac{n_i}{n} entropy(i)$$

Parent node p is split into k partitions, n_i is the number of samples in partition i.

- Information gain, measures the reduction in entropy achieved because of the split. Choose the split that achieve most reduction (maximizes gain).
- Tends to prefer splits that result in a large number of partitions all of them pure.

Decision trees: Entropy





$$G('x_i < 0.3') = H_{\text{root}} - H$$

$$= 0.940 - (7/14)0.985 - (7/14)0.592$$

$$= 0.151$$

$$G('x_j < 0.4') = H_{\text{root}} - H$$

$$= 0.940 - (8/14)0.811 - (6/14)0.1.0$$

$$= 0.048$$

Inspired in Tom Mitchell's Machine Learning example.

Classification trees

Observations:

- Typically use entropy to grow the tree.
- Trees tend to have very high variance: small changes in data result in large changes in hypothesis.
- Easy to handle multi-class, different loss functions, etc.

Trees and overfitting

Given enough time we can fit the complete training set perfectly, this includes also noise.

How can we avoid overfitting?

- Stop growing when data split is not statistically significant.
- Grow the full tree, then post-prune.

How to select the "best" tree:

- Measure performance over separate validation data set.
- Add complexity penalty to performance measure.

A simple way to prune: reduced-error pruning.

- Split data into training and validation set
- Create a candidate tree on your training data
- Do until further pruning is harmful:
  1. Evaluate impact on validation set of pruning each possible node (plus those below it)
  2. Greedily remove the one that most improves validation set accuracy

Working with real data.

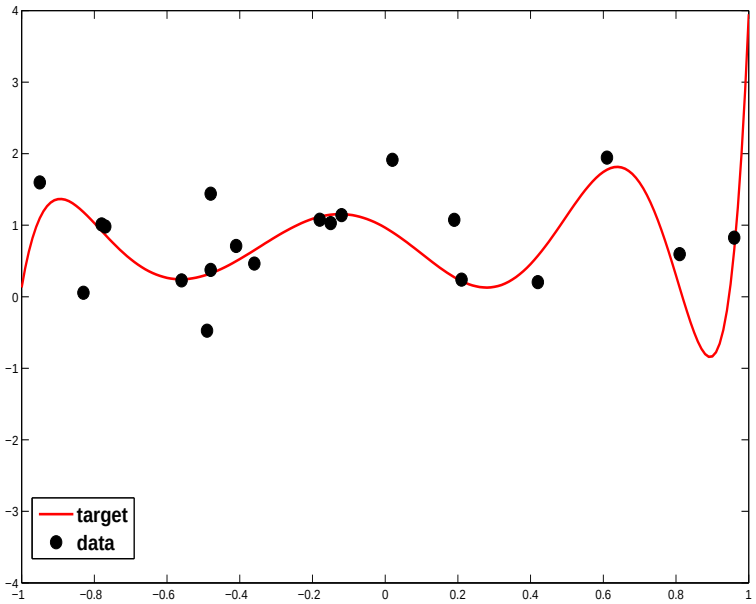
Decision trees work by testing a hypothesis and splitting according to the resulting test. But...

- Features with multiple discrete values
  - Construct a multiway split.
  - Test one value versus all the others. Build as many boolean features as values has the attribute. Each new feature takes value 1 if the example has a particular value. This strategy is useful for any classifier.
  - Group values into disjoint subsets. Select a bipartition of the set of values. Figure out which groups are the best. Figuring this out is a hard problem.
- Real-valued features: Consider a threshold split using each observed value of the feature.

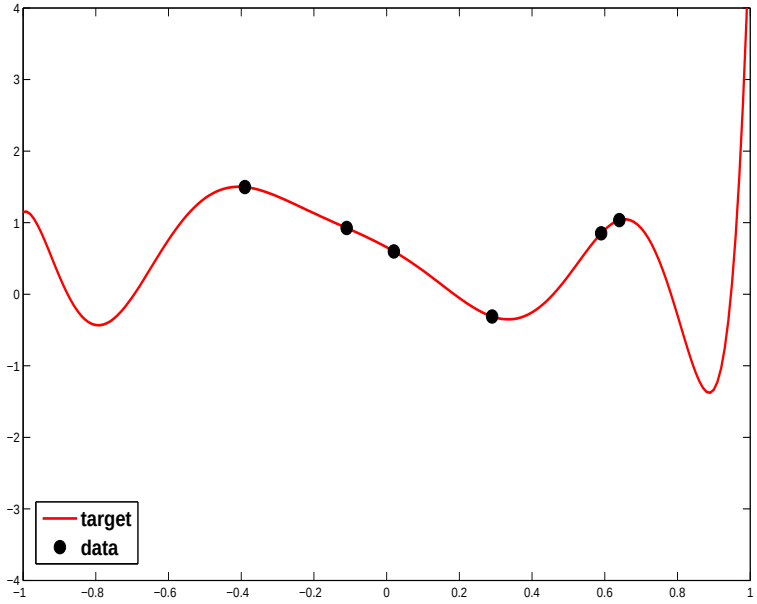
Case studies

We have seen before the over fitting effect. Let us retake the question here.

Case A: sampling from a target function + noise

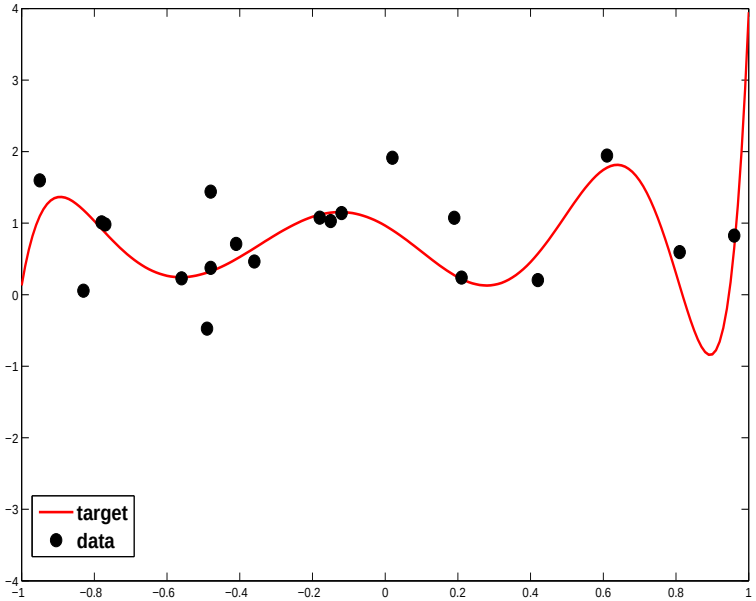


Case B: sampling from a target function

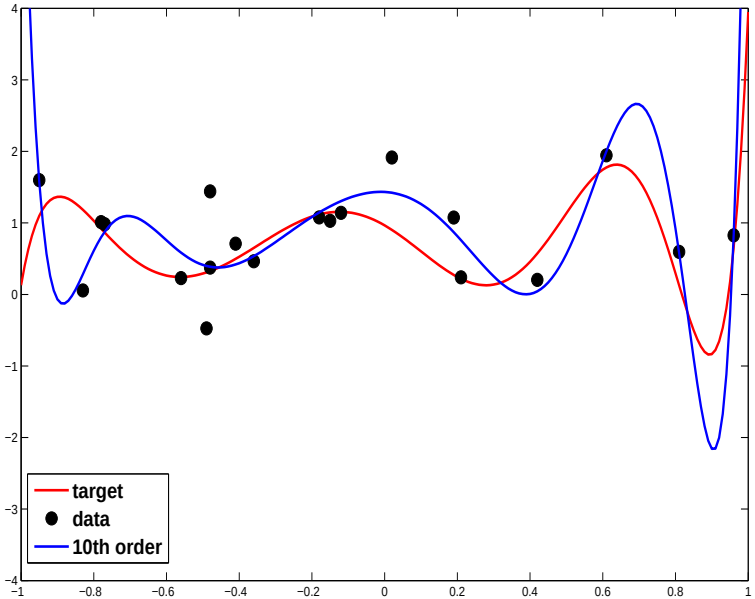


Noisy target

Case A: sampling from a target function (10th order polynomial) + noise



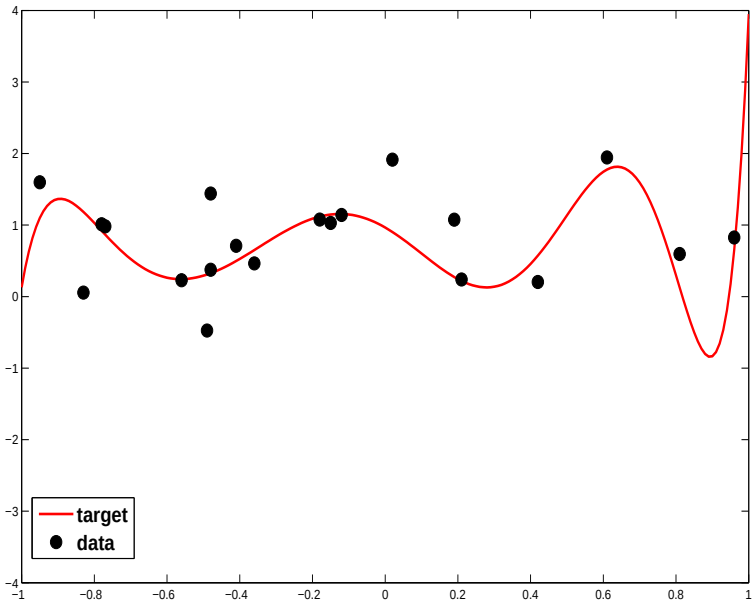
Case A.1: Fit a 10th order polynomial



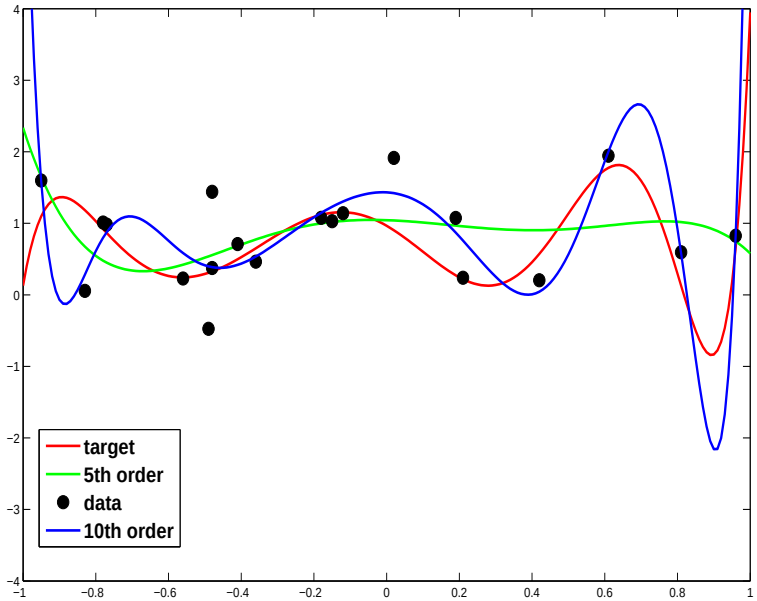
$E_{in} \downarrow \quad E_{out} \uparrow$

Noisy target

Case A: sampling from a target function (10th order polynomial) + noise



Case A.2: Fit a 5th order polynomial

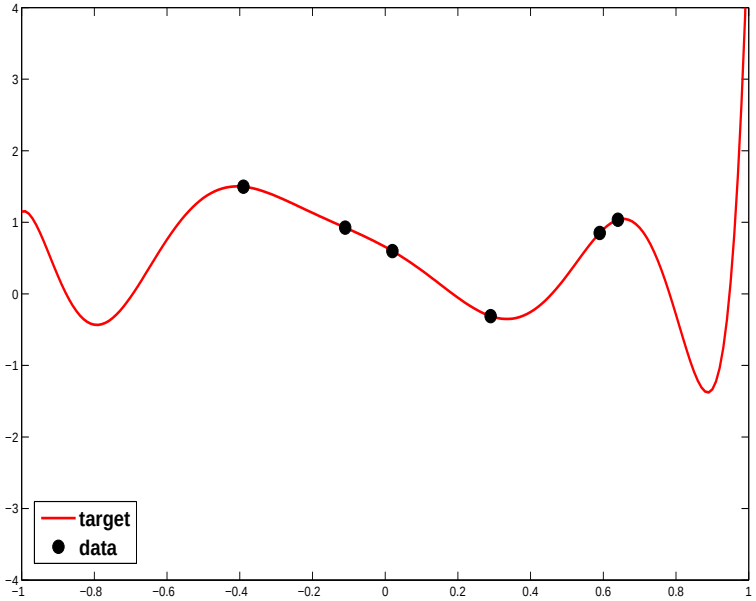


$$E_{in} \downarrow \quad E_{out} \uparrow$$

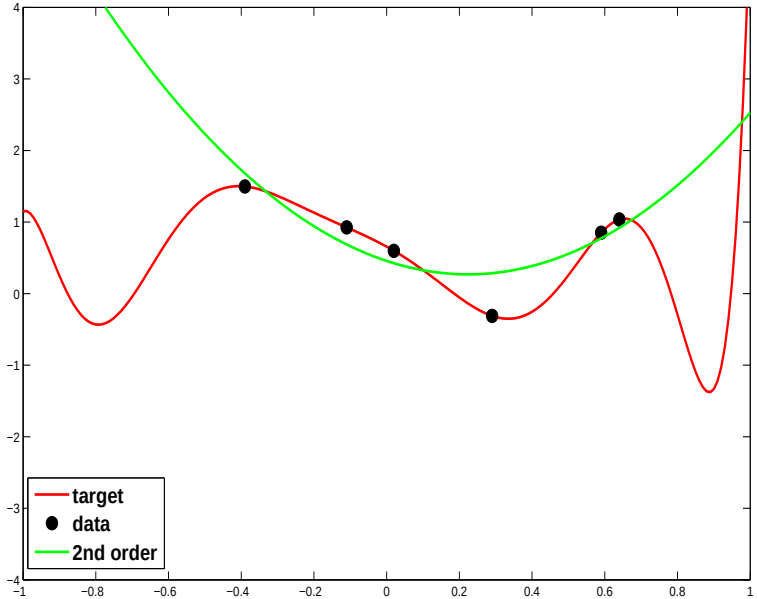
$$(E_{in} \leqslant E_{in} \leqslant E_{out} \leqslant E_{out})$$

Noiseless target

Case B: Sampling from a target function (10th order polynomial) without noise



Case B.1: Let us fit a 2nd order polynomial.



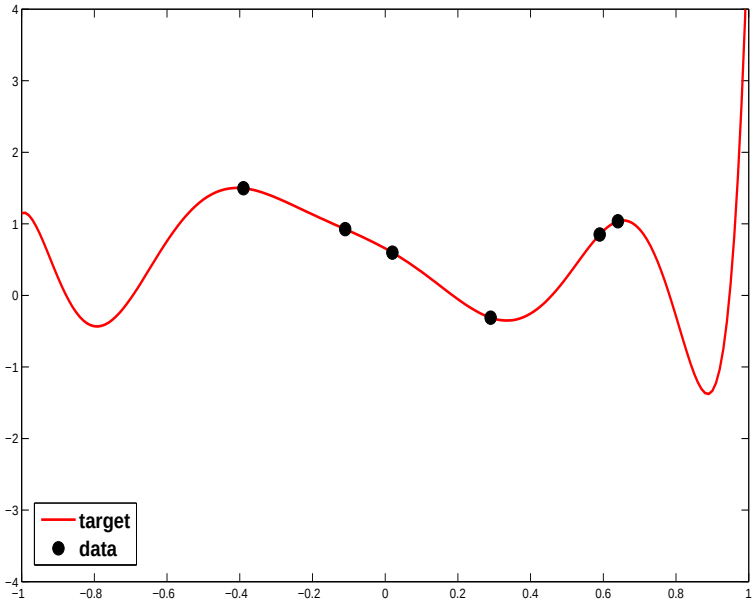
$E_{in} \downarrow \quad E_{out} \uparrow$

But ... I know there is no noise.

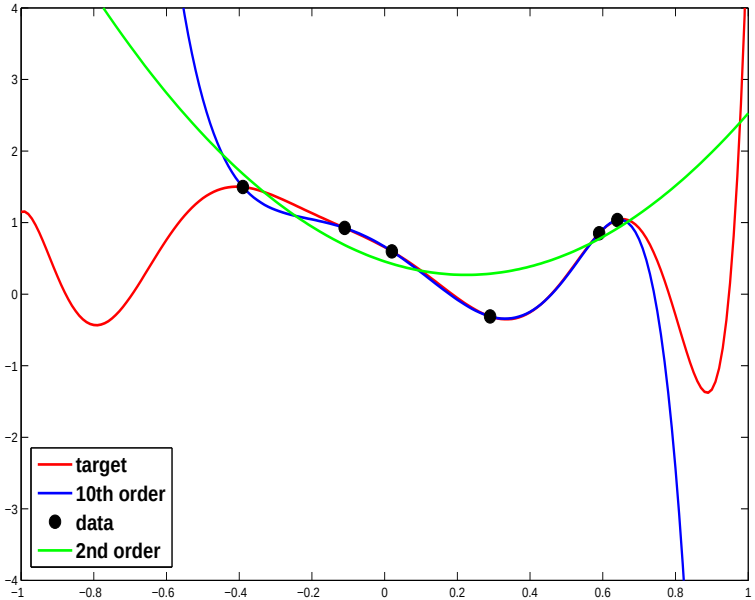
Let us fit a 10th order polynomial.

Noiseless target

Case B: Sampling from a target function (10th order polynomial) without noise



Case B.2: Let us fit a 10th order polynomial.



$E_{in} \downarrow\downarrow \quad E_{out} \uparrow\uparrow$

Wait a minute!

I already know target function is 10th order polynomial, and fitting a 10th order polynomial worsens my situation with respect the restricted model!

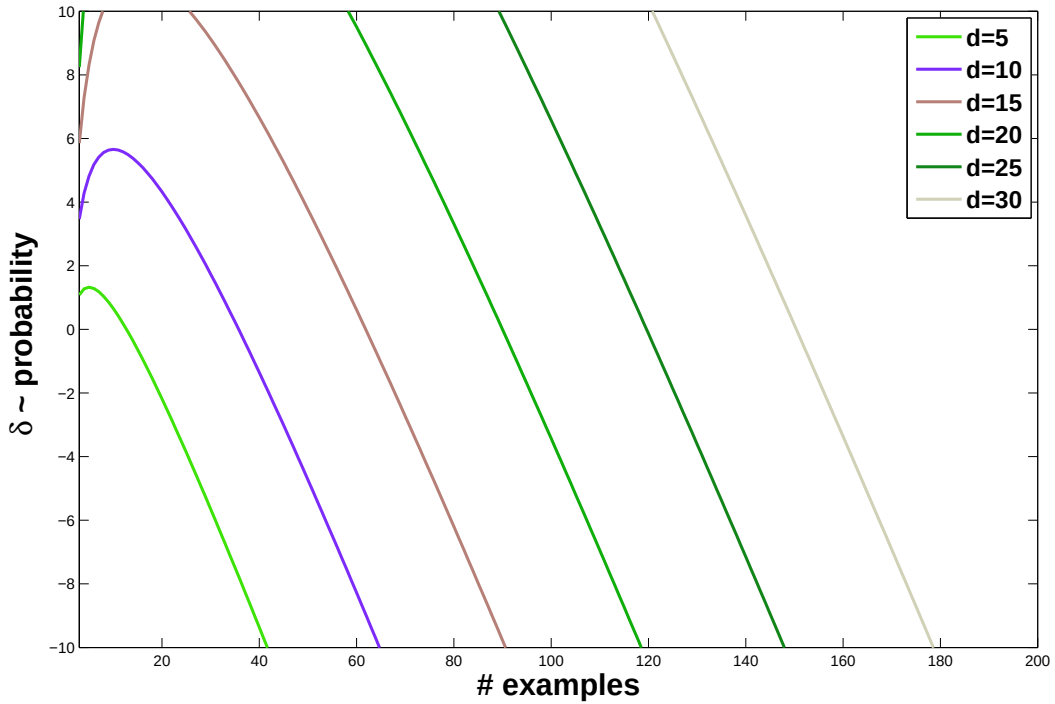
Observations

What is happening here?

We are trying to match target complexity instead of available data resources

Remember the VC dimension,

$$\mathbb{P}[\max_{h \in \mathcal{H}} |E_{in}(h) - E_{out}(h)| > \epsilon] \leqslant \delta \quad \propto \quad N^d e^{-N}$$



Observations

Observations

- The lines are close to linear below 0 $\implies \mathbb{P} \leqslant 1$.
- Given a certain level of δ the number of examples are proportional to the VC-dimension. Bigger VC-dimension you will need more samples.

Rules of thumb

For reasonable ϵ and δ the following observation holds:

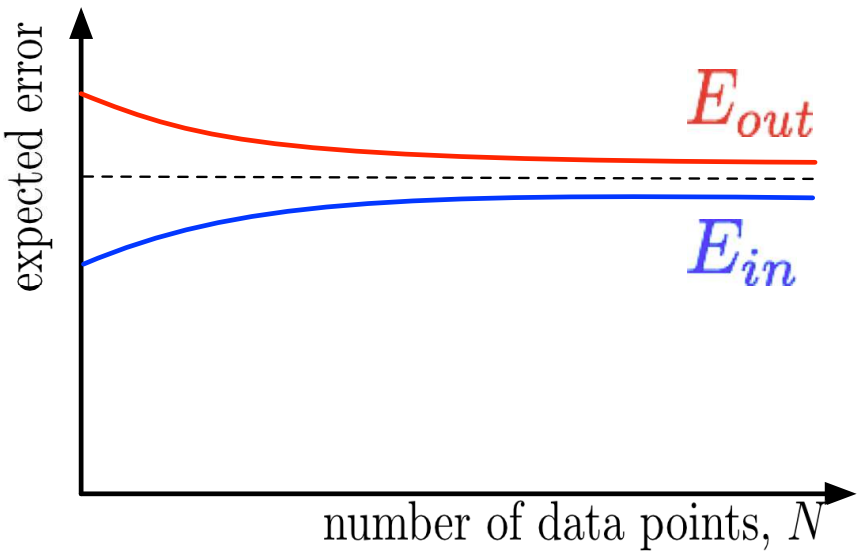
$$N \geqslant 10 d_{VC}$$

Remember than the VC-dimension can be regarded as the number of actual degrees of freedom in your model.

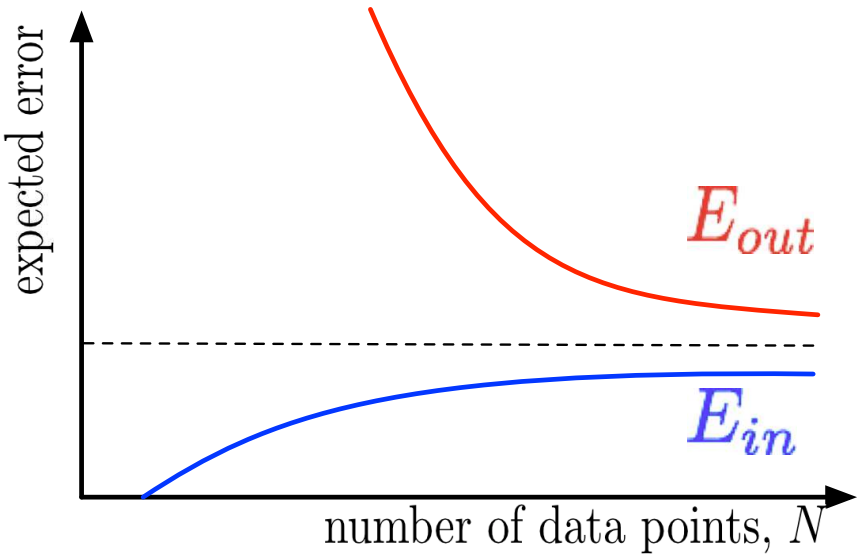
So, what happened in the fitting examples?

Learning curves

Low order model:



Higher order model:



Two sources of over fitting

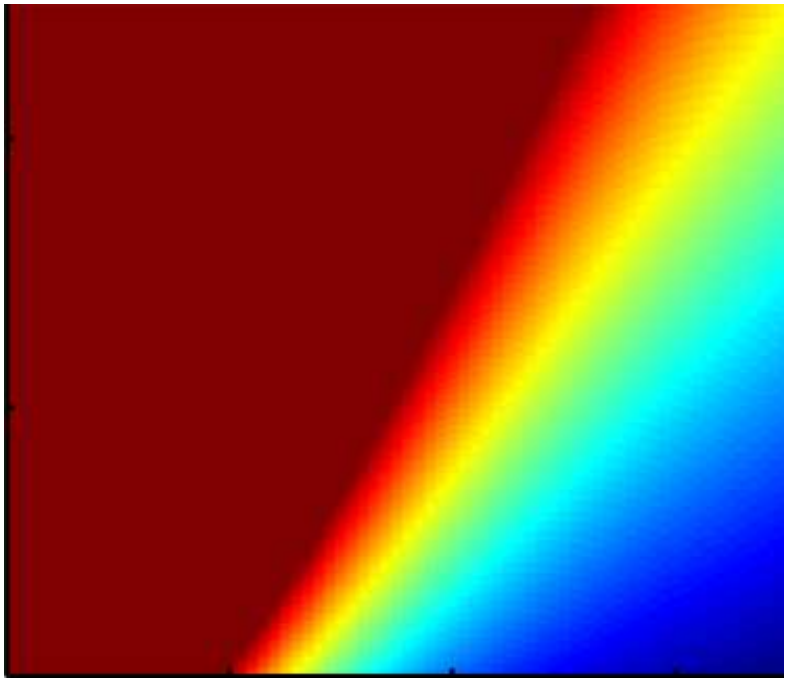
In order to visualize the two intuitive sources of over fitting we may actually construct an experiment to evaluate this. On one hand we can measure how over fitting occurs when comparing the energy of the noise (σ^2) and the model complexity (order of a polynomial) against the number of examples N (abscissa).
In order to measure over fitting we will use to different order polynomials: 2nd order and 10th order. If changing from 2nd to 10th gets us in trouble then we are in front of over fitting. Otherwise we will be under fitting.
Thus our measurement of over fitting can be

$$E_{out}^{10th} - E_{out}^{2nd}$$

Experiment from Y. Abu-Mostafa at CalTech slides

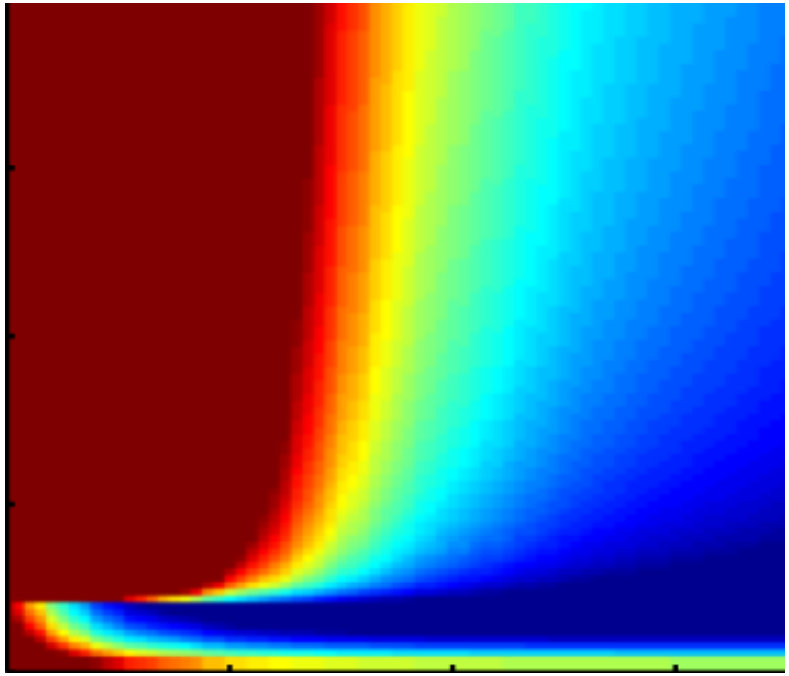
Two sources of over fitting

Noise with respect to the number of samples (abscissa). Blue and red correspond to negative and positive values, respectively

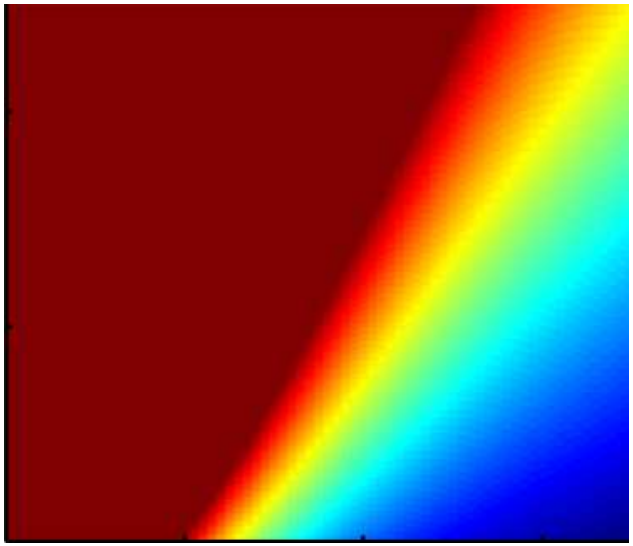


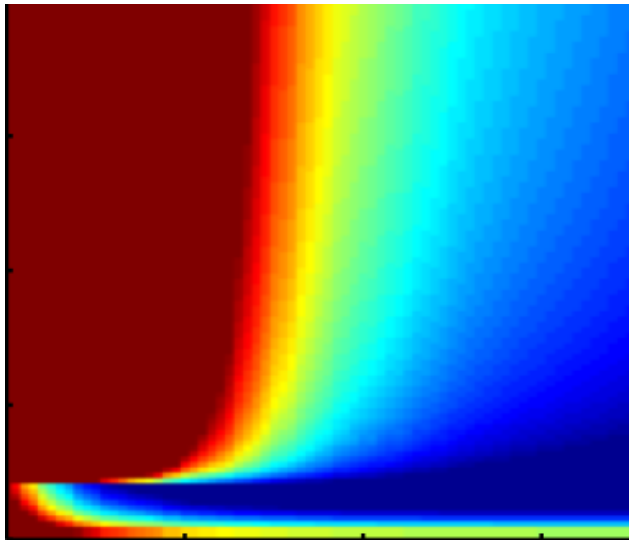
Experiment from Y. Abu-Mostafa at CalTech slides

Model complexity with respect to the number of samples (abscissa). Blue and red correspond to negative and positive values, respectively



Two sources of over fitting





Observations

- number of data points ↑ overfitting ↓
- stochastic noise ↑ overfitting ↑
- "deterministic" noise ↑ overfitting ↑

Experiment from Y. Abu-Mostafa at CalTech slides

Renaming deterministic noise

The notion of "deterministic" noise represents the part of the target function g that can not be captured by $\mathcal{H}$. This is basically $h^*(x) - g(x)$.

Let us analyze how the expected error in a least squares loss regression is decomposed assuming we have some noise.

Consider a noisy target function $y = g(x) + \epsilon, \quad \mathbb{E}[\epsilon] = 0$

Renaming deterministic noise

We want to characterize,

$$\mathbb{E}_{\mathcal{D},x}[(y - f(x))^2] = \mathbb{E}_{\mathcal{D},x,\epsilon}[(g(x) + \epsilon - f(x))^2] =$$

Renaming deterministic noise

We want to characterize,

$$\mathbb{E}_{\mathcal{D},x}[(y - f(x))^2] = \mathbb{E}_{\mathcal{D},x,\epsilon}[(g(x) + \epsilon - f(x))^2] =$$

Let us add the expected value of our hypothesis $f(\bar{x})$

$$= \mathbb{E}_{\mathcal{D},x,\epsilon}[(g(x) + \epsilon - f(x) + f(\bar{x}) - f(\bar{x}))^2] =$$

Renaming deterministic noise

We want to characterize,

$$\mathbb{E}_{\mathcal{D},x}[(y - f(x))^2] = \mathbb{E}_{\mathcal{D},x,\epsilon}[(g(x) + \epsilon - f(x))^2] =$$

Let us add the expected value of our hypothesis $f(\bar{x})$

$$= \mathbb{E}_{\mathcal{D},x,\epsilon}[(g(x) + \epsilon - f(x) + f(\bar{x}) - f(\bar{x}))^2] =$$

rearranging

$$= \mathbb{E}_{\mathcal{D},x,\epsilon}[((g(x) - f(\bar{x})) + (f(\bar{x}) - f(x)) + \epsilon)^2] =$$

Renaming deterministic noise

We want to characterize,

$$\mathbb{E}_{\mathcal{D},x}[(y - f(x))^2] = \mathbb{E}_{\mathcal{D},x,\epsilon}[(g(x) + \epsilon - f(x))^2] =$$

Let us add the expected value of our hypothesis $f(\bar{x})$

$$= \mathbb{E}_{\mathcal{D},x,\epsilon}[(g(x) + \epsilon - f(x) + f(\bar{x}) - f(\bar{x}))^2] =$$

rearranging

$$= \mathbb{E}_{\mathcal{D},x,\epsilon}[((g(x) - f(\bar{x})) + (f(\bar{x}) - f(x)) + \epsilon)^2] =$$

$$= \mathbb{E}_{\mathcal{D},x,\epsilon}[((g(x) - f(\bar{x}))^2 + (f(\bar{x}) - f(x))^2 + \epsilon^2 + \text{lots of cross-terms})] =$$

Renaming deterministic noise

We want to characterize,

$$\mathbb{E}_{\mathcal{D},x}[(y - f(x))^2] = \mathbb{E}_{\mathcal{D},x,\epsilon}[(g(x) + \epsilon - f(x))^2] =$$

Let us add the expected value of our hypothesis $f(\bar{x})$

$$= \mathbb{E}_{\mathcal{D},x,\epsilon}[(g(x) + \epsilon - f(x) + f(\bar{x}) - f(\bar{x}))^2] =$$

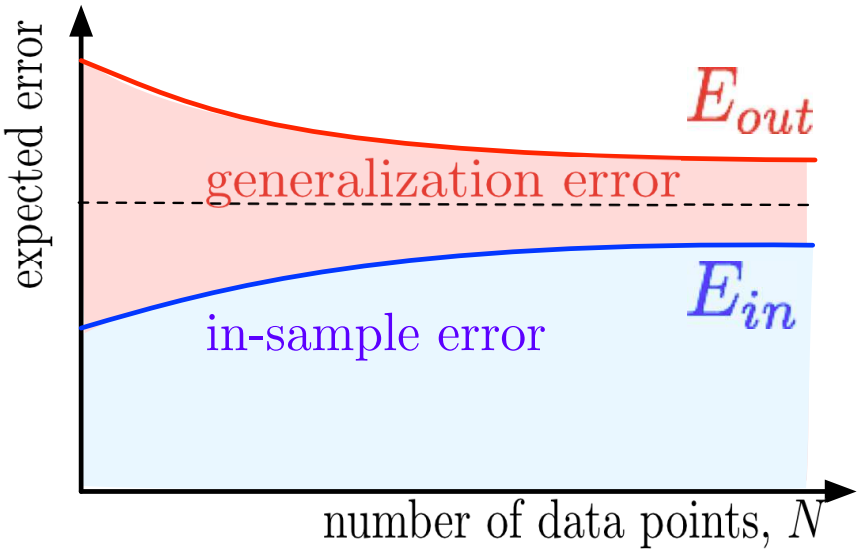
rearranging

$$= \mathbb{E}_{\mathcal{D},x,\epsilon}[((g(x) - f(\bar{x})) + (f(\bar{x}) - f(x)) + \epsilon)^2] =$$

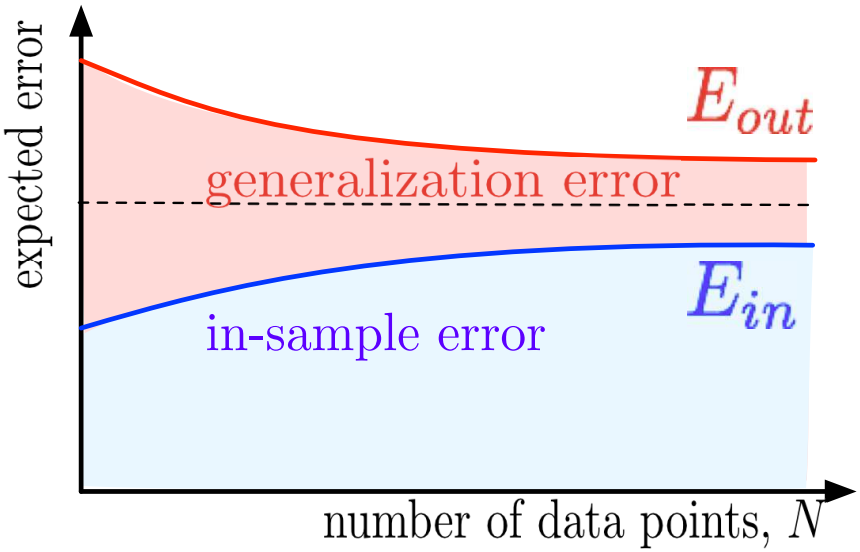
$$= \mathbb{E}_{\mathcal{D},x,\epsilon}[((g(x) - f(\bar{x}))^2 + (f(\bar{x}) - f(x))^2 + \epsilon^2 + \text{lots of cross-terms})] =$$

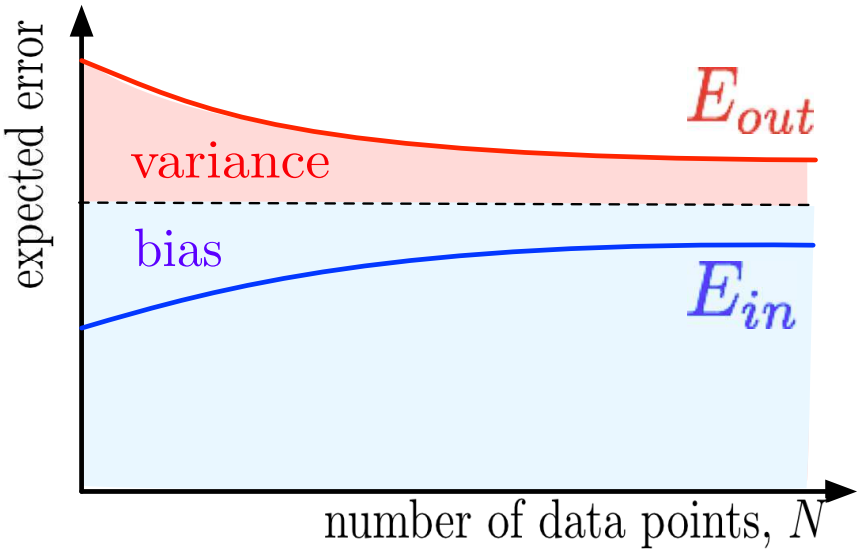
$$= \underbrace{\mathbb{E}_{\mathcal{D},x}[(f(\bar{x}) - f(x))^2]}_{\text{variance}} + \underbrace{\mathbb{E}_{x}[((g(x) - f(\bar{x}))]^2}_{\text{bias}} + \underbrace{\mathbb{E}_{\epsilon}[\epsilon^2]}_{\text{stochastic noise}}$$

Learning curves: VC versus Bias-variance

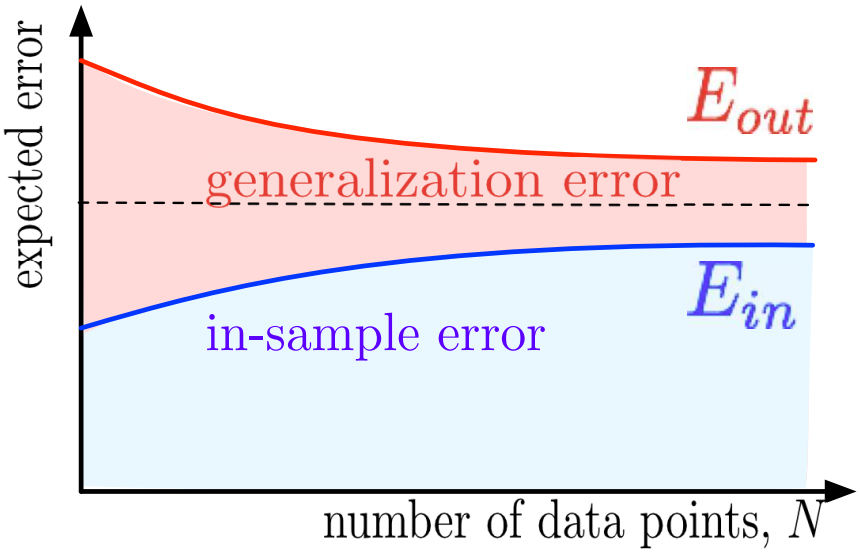


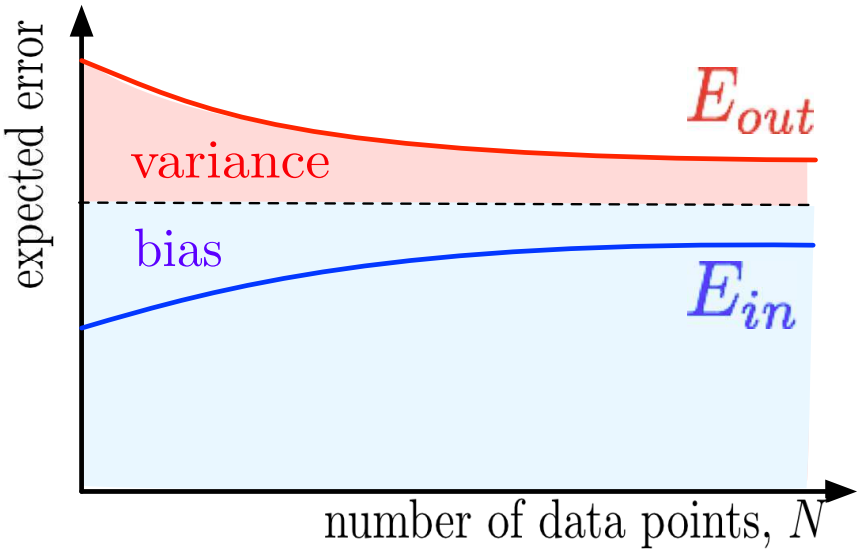
Learning curves: VC versus Bias-variance





Learning curves: VC versus Bias-variance



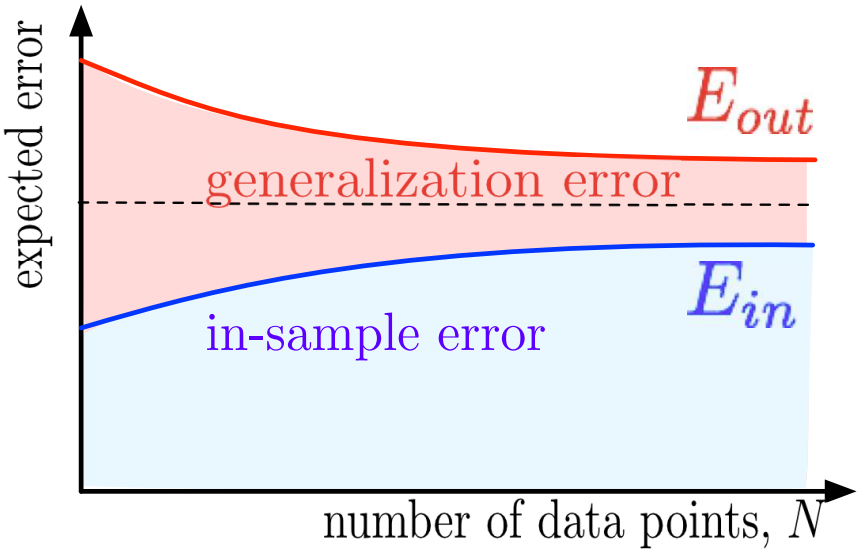


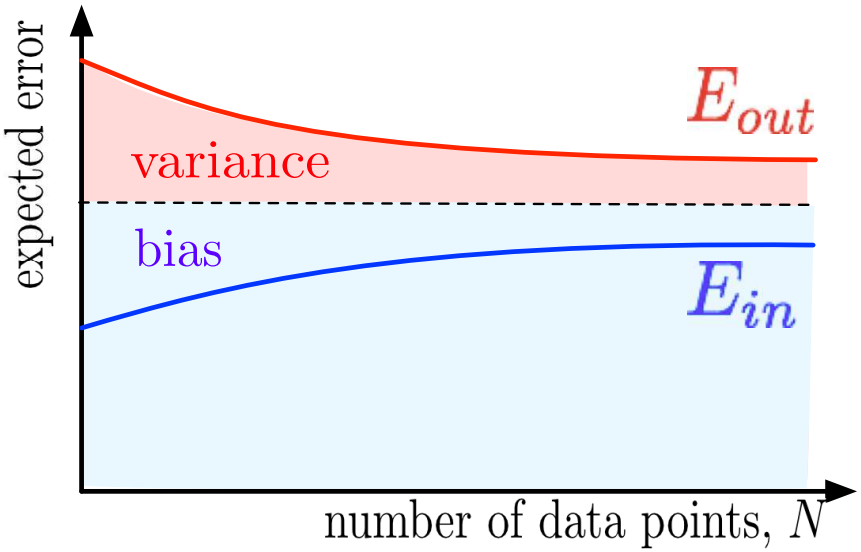
Observations (I)

- The bias shows how the best (or close to the best)[a] approximation behaves.
- In bias-variance we consider approximation overall and in vc approach we see approximation in-sample

[a] the best can be outside of the hypothesis set

Learning curves: VC versus Bias-variance





Observations (II)

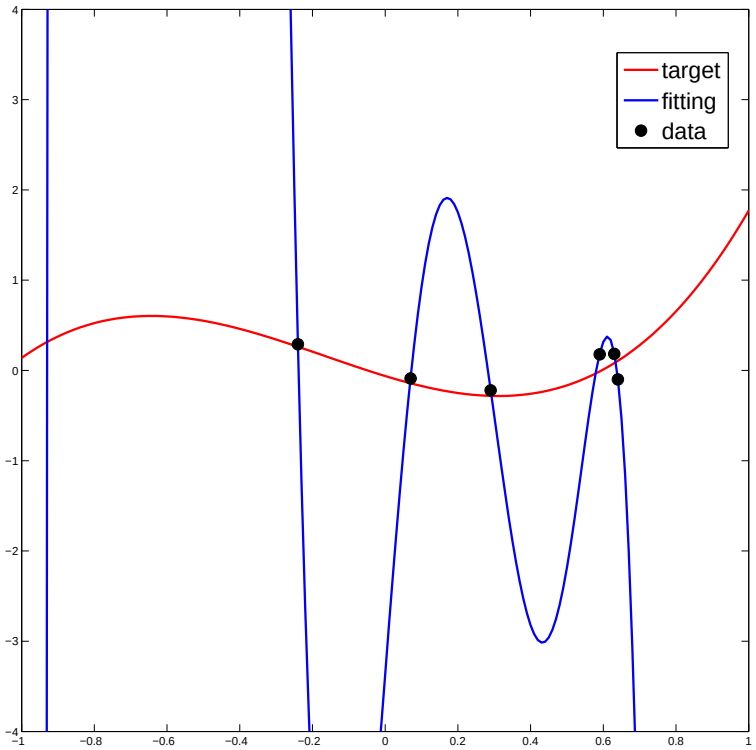
- variance and generalization error consider the uncertainty of hypothesis and show how much do I lose going for out-of-sample.
- More complex models can represent a wide variety of hypothesis and thus have small bias, but higher variance – small changes in the data set can generate big changes in the estimate.

Two cures for overfitting

1. **Regularization:** Putting the brakes.
2. **Validation:** Check the bottom line and check it does not over fit.

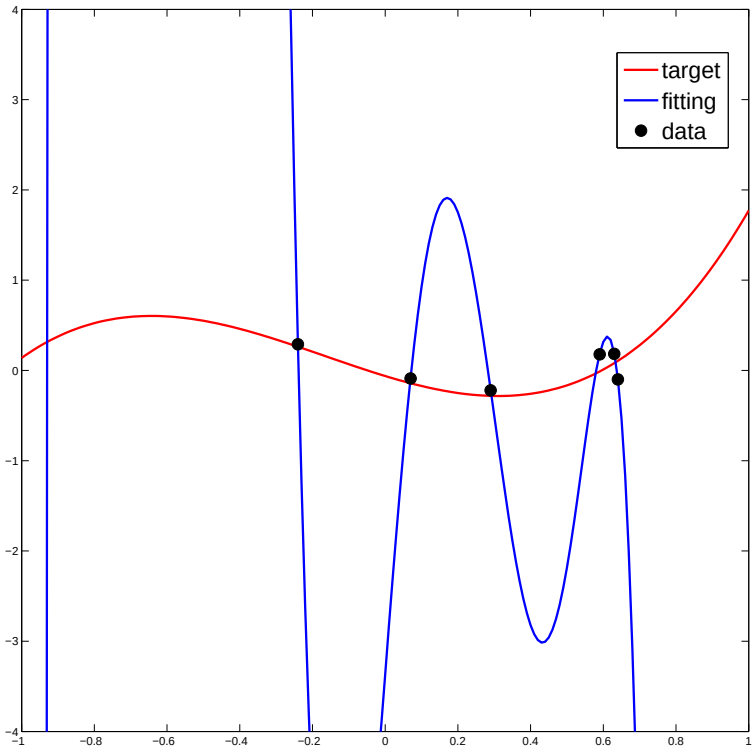
Introduction to regularization

free fit

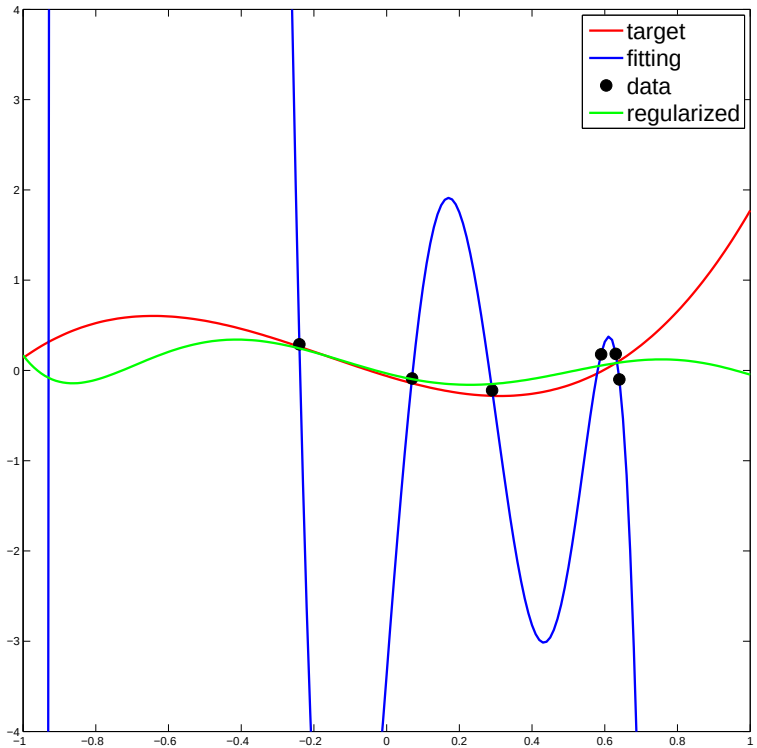


Introduction to regularization

free fit



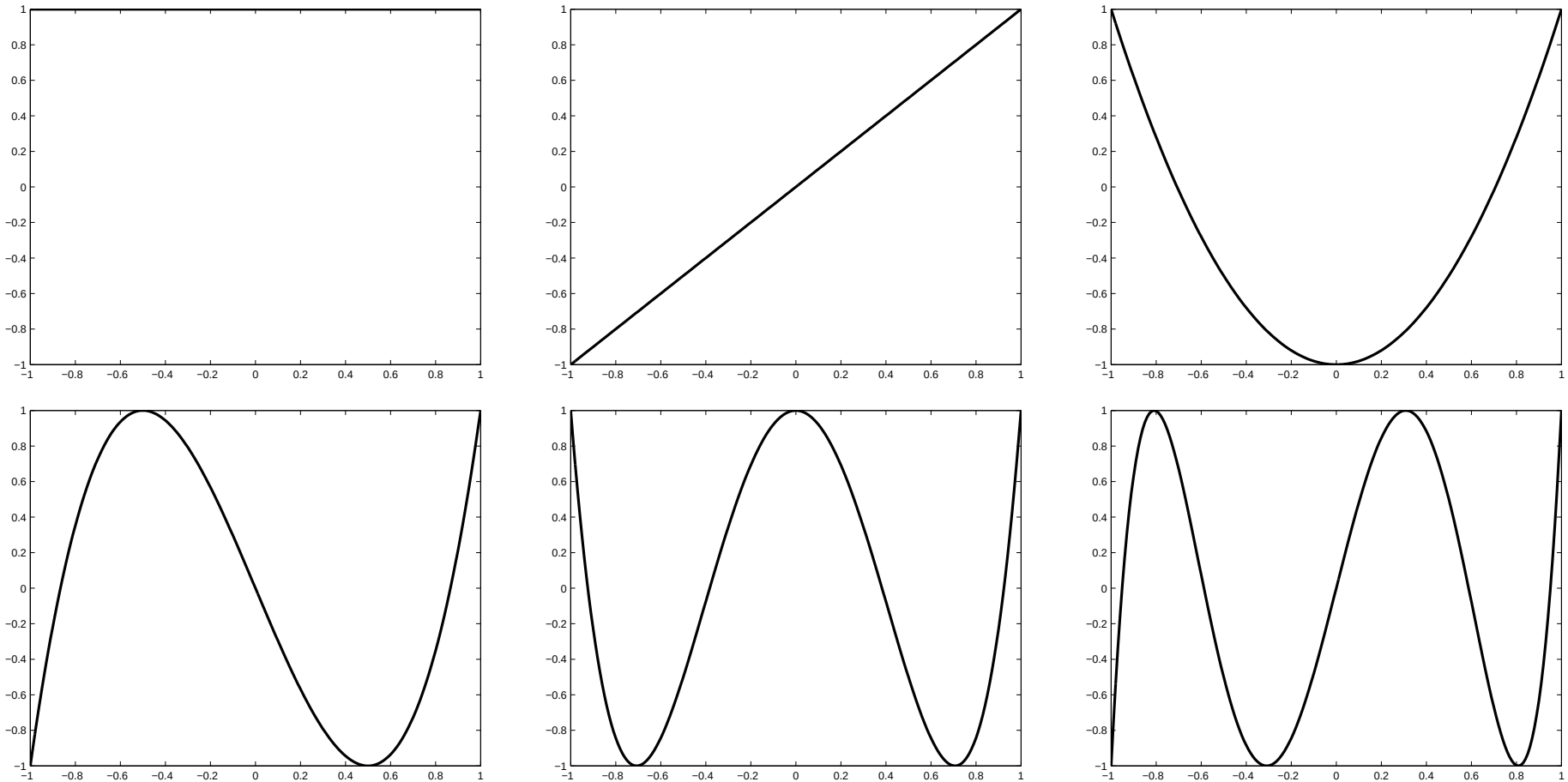
restrained fit



Introduction to regularization

Consider the problem of fitting a function using some polynomial basis

Chebyschev orthogonal polynomial basis



Introduction to regularization

Consider the problem of fitting a function using a linear combination of some polynomial basis $\{p^i\}$ $i = 0 \dots k$ where $p^i(x)$ is an i-th order polynomial.
The approximating model is :

$$\mathbf{p}(x) = \sum_{i=0}^{k} w_i p^i(x)$$

Observe that this is a linear model of the polynomial basis, thus we can use all the machinery from the former lecture.

Introduction to regularization

Given a set of data pairs $(x_j, y_j) \rightarrow ((p^0(x_j), \ldots, p^k(x_j)), y_j)$, the in-sample error using a least squares loss function is

$$E_{in} = \frac{1}{N} \sum_{j=1}^{N} (\sum_{i=0}^{k} w_i p^i(x_j) - y_j)^2 = \frac{1}{N} (P\mathbf{w} - \mathbf{y})^T (P\mathbf{w} - \mathbf{y})$$

and its solution is known

$$\mathbf{w}^* = (P^T P)^{-1} P^T \mathbf{y}$$

Introduction to regularization

We have seen constrained models before ... $\mathcal{H}^{2nd}$ is a constrained version of $\mathcal{H}^{10th}$ where $w_i = 0,\ i > 3$ (hard constraint).

Soft constraint:

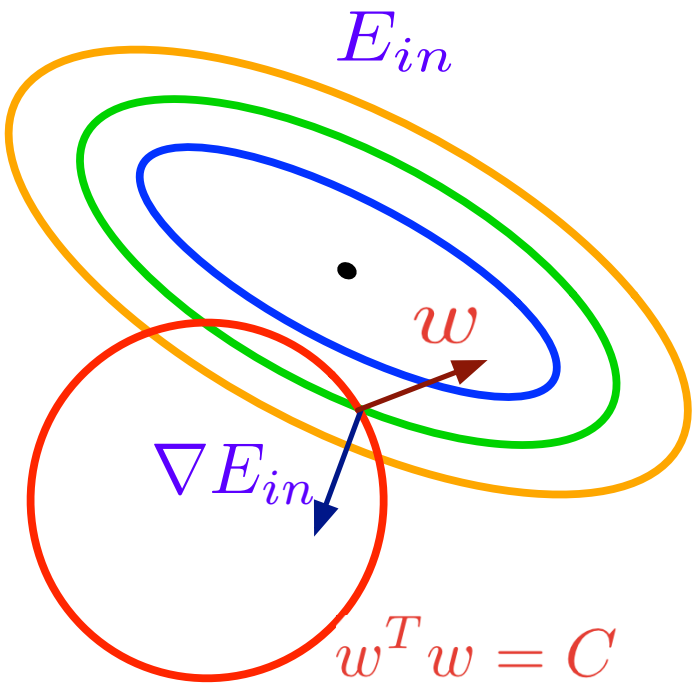
$$\sum_{i=0}^{k} w_i^2 \leqslant C$$

thus, the problem becomes

$$\begin{array}{ll} \text{minimize} & \frac{1}{N}(P\mathbf{w} - \mathbf{y})^T(P\mathbf{w} - \mathbf{y}) \\ \text{subject to} & \mathbf{w}^T\mathbf{w} \leqslant C \end{array}$$

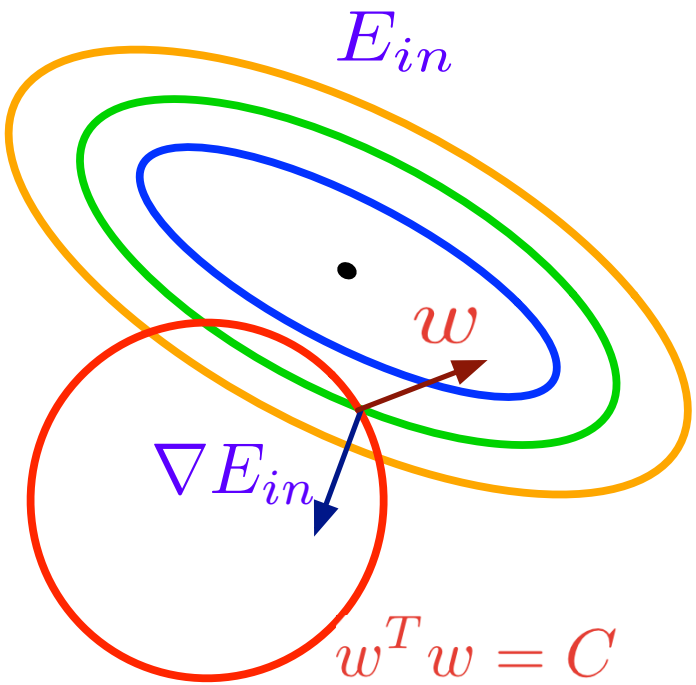
Introduction to regularization

minimize $\frac{1}{N}(P\mathbf{w}-\mathbf{y})^T(P\mathbf{w}-\mathbf{y})$
subject to $\mathbf{w}^T\mathbf{w} \leqslant C$



Introduction to regularization

minimize $\frac{1}{N}(P\mathbf{w}-\mathbf{y})^T(P\mathbf{w}-\mathbf{y})$
subject to $\mathbf{w}^T\mathbf{w} \leqslant C$



The minimum is achieved when

$$\nabla E_{in}(\mathbf{w}^*) \propto -\mathbf{w}^*$$

Introduction to regularization

For the sake of simplicity we may choose

$$\nabla E_{in}(\mathbf{w}^*) = -2\frac{\lambda}{N}\mathbf{w}^*$$

$$\nabla E_{in}(\mathbf{w}^*) + 2\frac{\lambda}{N}\mathbf{w}^* = 0$$

which corresponds to the necessary and sufficient condition for the following unconstrained minimization problem

$$\text{minimize} \quad E_{in}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$$

(we expected this result since this corresponds to the Lagrangian of a constrained problem)

Introduction to regularization

For the sake of completion, let us find the constrained solution

$$\text{minimize} \quad E_{aug} = E_{in}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w} \rightarrow \nabla E_{aug}(w) = 0$$

$$\nabla\Big(\frac{1}{N}(P\mathbf{w} - \mathbf{y})^T(P\mathbf{w} - \mathbf{y}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}\Big) = 0$$

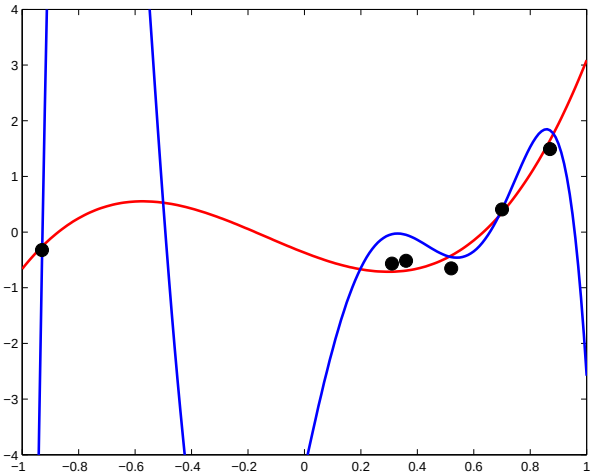
$$P^T(P\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w} = 0$$

$$P^TP\mathbf{w} + \lambda\mathbf{w} = P^T\mathbf{y}$$

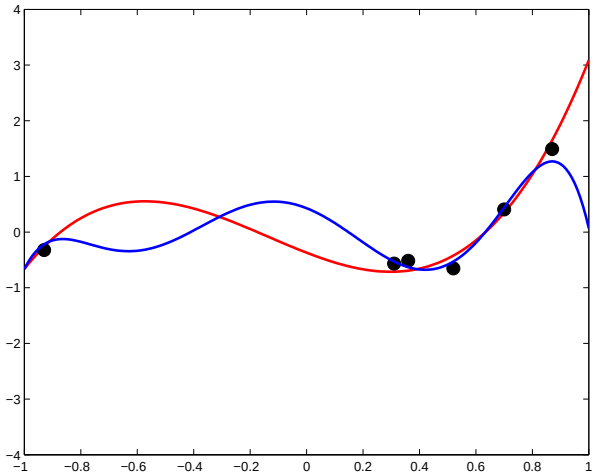
$$(P^TP + \lambda I)\mathbf{w} = P^T\mathbf{y}$$

$$\mathbf{w}^* = (P^TP + \lambda I)^{-1}P^T\mathbf{y}$$

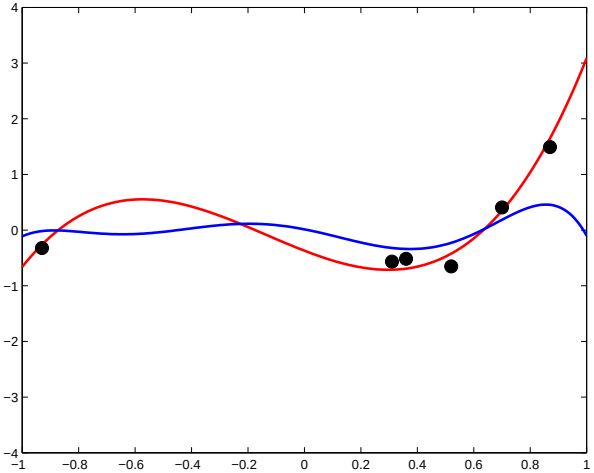
Introduction to regularization



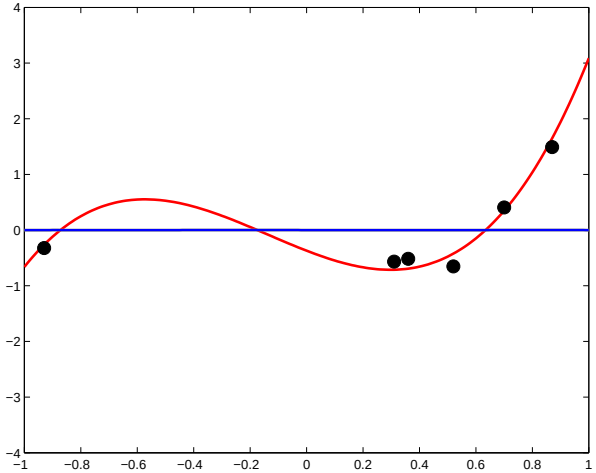
$(\lambda = 0)$



$(\lambda = 0.5)$



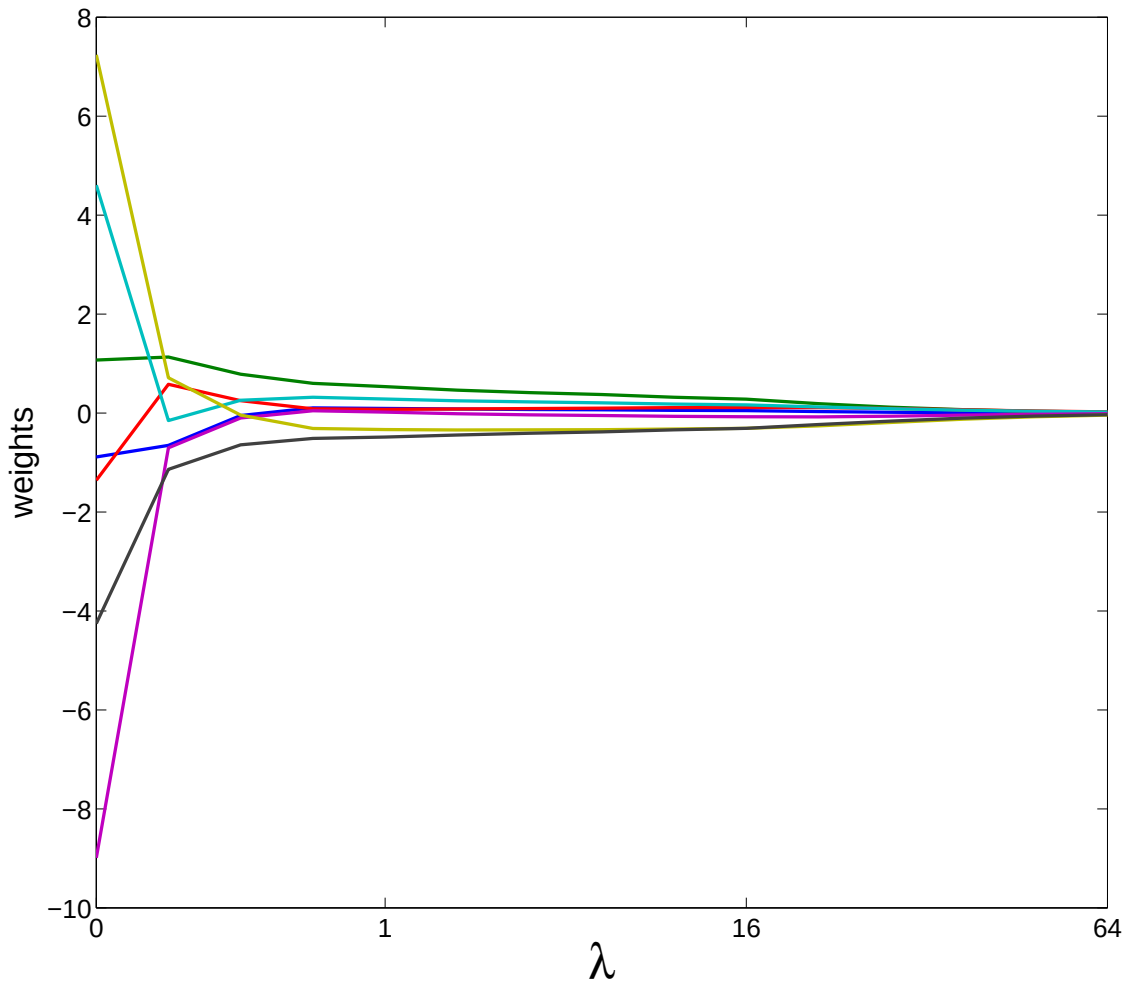
$(\lambda = 10)$



$(\lambda = 1000)$

Introduction to regularization

As λ increases the weights decay and approach zero.



Introduction to regularization. In general:

Data: $(\mathbf{x}_1, y_1) \dots (\mathbf{x}_n, y_n) \in \mathbb{R}^N \times \mathbb{R}$
Estimate: $f : \mathbb{R}^N \to \mathbb{R}$

Hypothesis space $\mathcal{H}$.

$$f^* = \arg\min_{f \in \mathcal{H}} \frac{1}{n} \sum_i \mathcal{L}(f(x_i), y_i) + \Omega(f)$$

fit to data + complexity penalty

Occam's razor: "simpler" hypotheses are better.
$\Omega(f)$ incorporates our simplicity assumptions.
If we fail with the complexity penalty we still have λ (fixed by validation)

Regularization on trees

Regularization encodes a notion to complexity to be minimized. Recall the regression tree optimization

$$\underset{T}{\text{minimize}} \quad \sum_{i=1}^{N} (T(x_i) - y_i)^2 + (\lambda \cdot \#\text{nodes})$$

In that formulation our notion of complexity is encoded as "the number of nodes in the tree".
Since we can not optimize the function directly we can **regularize by pruning** after building the tree.

Regularization on trees

Consider all trees T that can be obtained by pruning the original one (replacing some node(s) by the average of all data below).
Cost complexity:

$$C_\lambda(T) = \sum_{m=1}^{|T|} \sum_{\mathbf{x}_u \in \mathcal{D}_{\mathcal{N}_m}} (y_u - \bar{y}(\mathbf{x}_u))^2 + \lambda |T|$$

where m ranges over the leaves of T.
For a fixed λ find a T that minimizes $C_\lambda(T)$ by "weakest-link" pruning: remove the bottom-level split that minimizes the increase in overall variance.

Regularization on trees

Consider all trees T that can be obtained by pruning the original one (replacing some node(s) by the average of all data below).
Cost complexity:

$$C_\lambda(T) = \sum_{m=1}^{|T|} \sum_{\mathbf{x}_u \in \mathcal{D}_{\mathcal{N}_m}} (y_u - \bar{y}(\mathbf{x}_u))^2 + \lambda |T|$$

where m ranges over the leaves of T.
For a fixed λ find a T that minimizes $C_\lambda(T)$ by "weakest-link" pruning: remove the bottom-level split that minimizes the increase in overall variance.
In classification trees: use typically entropy to grow the tree and misclassification to prune.

Regularization on trees

Consider all trees T that can be obtained by pruning the original one (replacing some node(s) by the average of all data below).
Cost complexity:

$$C_\lambda(T) = \sum_{m=1}^{|T|} \sum_{\mathbf{x}_u \in \mathcal{D}_{\mathcal{N}_m}} (y_u - \bar{y}(\mathbf{x}_u))^2 + \lambda |T|$$

where m ranges over the leaves of T.
For a fixed λ find a T that minimizes $C_\lambda(T)$ by "weakest-link" pruning: remove the bottom-level split that minimizes the increase in overall variance.
In classification trees: use typically entropy to grow the tree and misclassification to prune.

How should we pick λ?

Support vector machines ... AGAIN!

Remember the unconstrained equivalent of the primal SVM formulation,

$$\text{minimize} \quad \|a\|_2 + \lambda \sum_{i=1}^{N} (1 - y_i(a^T x_i + b))_+$$

Observe that we can interpret this as a regularization formulation where $\|a\|_2$ is the stabilization / penalty parameter (inverse of the geometric margin). This tells us that when no margin constraints are violated (zero loss), the regularization helps us to select the solution with the largest geometric margin.

Validation

$$\underbrace{E_{out}(h)}_{\text{validation estimates this quantity}} = E_{in}(h) + \underbrace{\text{overfit penalty}}_{\text{regularization estimates this quantity}}$$

Validation

Given an out-of-sample point $(\mathbf{x}, y)$, the error is $e(h(\mathbf{x}), y)$ (e.g. squared error, binary error). Then,

$$\mathbb{E}[e(h(\mathbf{x}, y)] = E_{out}(h)$$

$$\text{var}[e(h(\mathbf{x}), y)] = \sigma^2$$

Validation

Consider a validation set of K points, the error is

$$E_{val}(h) = \sum_{k=1}^{K} e(h(\mathbf{x}_k), y_k)$$

$$\mathbb{E}[E_{val}(h)] = \sum_{k=1}^{K} \mathbb{E}[e(h(\mathbf{x}_k), y_k)] = E_{out}(h)$$

$$\text{var}[E_{val}(h)] = \frac{1}{K^2} \sum_{k=1}^{K} \text{var}[e(h(\mathbf{x}_k), y_k)] = \frac{\sigma^2}{K}$$

$$E_{val}(h) = E_{out}(h) \pm \mathcal{O}\left(\frac{1}{\sqrt{K}}\right)$$

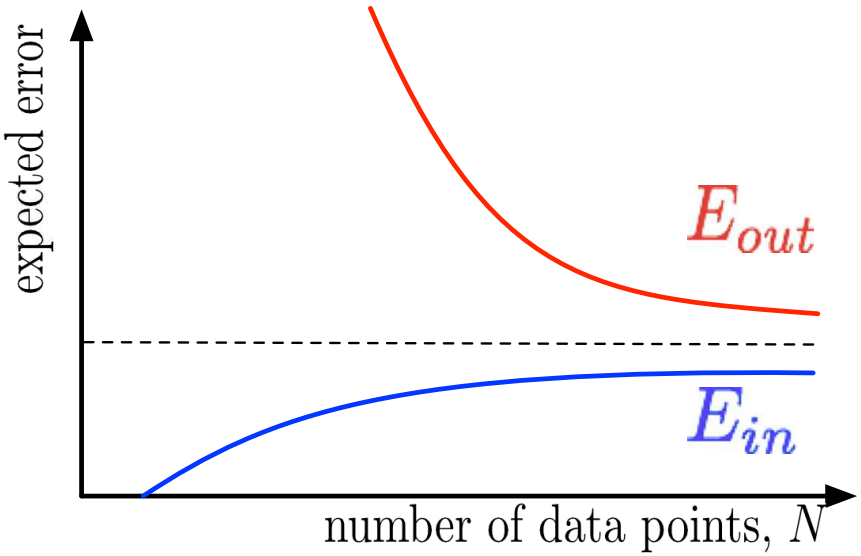
Validation

Given a data set $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$,

$\underbrace{K \text{ points}}_{\mathcal{D}_{val}} \rightarrow$ validation $\quad \underbrace{N - K \text{ points}}_{\mathcal{D}_{train}} \rightarrow$ training

Small $K \implies$ bad estimate

Large $K \implies$?



$\leftarrow K$

Validation

- Let $f_{\mathcal{D}}$ be the trained model given a dataset $\mathcal{D}$.
- Our goal is to approximate $E_{out}(f_{\mathcal{D}})$.
- We divide the training set in $\mathcal{D} = \mathcal{D}_{val} \cup \mathcal{D}_{train}$.
- E_{val} corresponds to the unbiased estimation of $E_{out}(f_{\mathcal{D}_{train}})$.
- Our hypothesis is to to consider $E_{out}(f_{\mathcal{D}}) \approx E_{out}(f_{\mathcal{D}_{train}})$

Observation

As stated, this process is exactly the same process used for testing.

Validation

Observations

- For exploitation we want to use a model trained on the complete dataset, $f_{\mathcal{D}}$.
- If K is small, we have large variance on the estimation E_{val}. It is an unreliable measure of how well the method will perform.
- If K is large, E_{val} is a reliable estimate of $E_{out}(f_{\mathcal{D}_{train}})$, but $f_{\mathcal{D}_{train}}$ is a very bad model. It deviates a lot from $f_{\mathcal{D}}$.

Validation

Observations

- For exploitation we want to use a model trained on the complete dataset, $f_{\mathcal{D}}$.
- If K is small, we have large variance on the estimation E_{val}. It is an unreliable measure of how well the method will perform.
- If K is large, E_{val} is a reliable estimate of $E_{out}(f_{\mathcal{D}_{train}})$, but $f_{\mathcal{D}_{train}}$ is a very bad model. It deviates a lot from $f_{\mathcal{D}}$.

Variances trade-off

The value K of the validation set trades-off the variance of the estimation of E_{val} with the variance of the model we are training $f_{\mathcal{D}_{train}}$.

Validation

Observations

- For exploitation we want to use a model trained on the complete dataset, $f_{\mathcal{D}}$.
- If K is small, we have large variance on the estimation E_{val}. It is an unreliable measure of how well the method will perform.
- If K is large, E_{val} is a reliable estimate of $E_{out}(f_{\mathcal{D}_{train}})$, but $f_{\mathcal{D}_{train}}$ is a very bad model. It deviates a lot from $f_{\mathcal{D}}$.

Variances trade-off

The value K of the validation set trades-off the variance of the estimation of E_{val} with the variance of the model we are training $f_{\mathcal{D}_{train}}$.

Rule of thumb

$$K = \frac{N}{5}$$

Difference between validation and test

Test: $\mathcal{D}_{test}$ is **only** used to estimate $E_{out}(f_{\mathcal{D}})$.
Validation: $\mathcal{D}_{val}$ is used to make decisions, e.g. finding the value of the regularization parameter λ.

Attention

Given a set, if an estimate of $E_{out}(f_{\mathcal{D}})$ using this set affects the learning process, it is no longer a test set.

Difference between validation and test

Test: $\mathcal{D}_{test}$ is **only** used to estimate $E_{out}(f_{\mathcal{D}})$.
Validation: $\mathcal{D}_{val}$ is used to make decisions, e.g. finding the value of the regularization parameter λ.

Attention

Given a set, if an estimate of $E_{out}(f_{\mathcal{D}})$ using this set affects the learning process, it is no longer a test set.

Test: Gives an unbiased estimation.
Validation: Gives an optimistic biased estimation (part of the knowledge in the set is transferred to the training process, so the training will fit it).

Validation and model selection.

The main use of validation is for model selection.

Model selection

1. Divide the training set into validation and training: $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{val}$.
2. Train the hypotheses using $\mathcal{D}_{train}$.
3. Estimate the out-of-sample error of each hypothesis using $\mathcal{D}_{val}$.
4. Select the model / parameters with minimum error.
5. Train the selected model using the complete dataset $\mathcal{D}$

The problem of K

We expect to approximate $E_{out}(f_{\mathcal{D}})$

$$E_{out}(f_{\mathcal{D}}) \underset{\text{(small } K\text{)}}{\approx} E_{out}(f_{\mathcal{D}_{train}}) \underset{\text{(large } K\text{)}}{\approx} E_{val}(f_{\mathcal{D}_{train}})$$

Cross-validation

Cross-validation is a scheme for having a large training set $\mathcal{D}_{train}$ while allowing a reliable estimate $E_{val}(f_{\mathcal{D}_{train}})$.

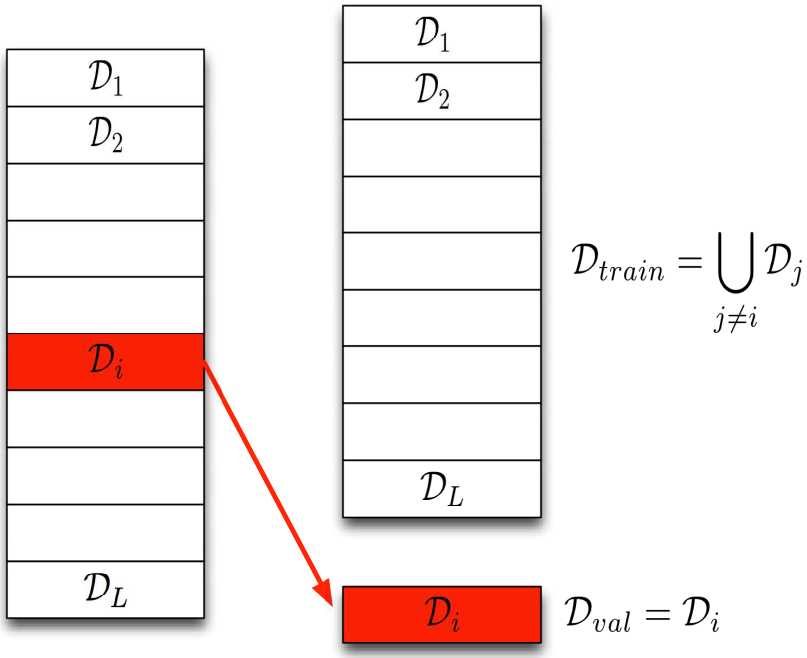
It is based on the observation that $E_{val} = \mathbb{E}[e_{val}]$, where e_{val} is the validation error in a small validation set.

Cross validation error:

$$E_{val} = \sum_{i=1}^{L} e_{val,i}$$

Cross-validation

1. **for** $i = 1..L$:



   1.
   2. Train the model on $\mathcal{D}_{train}$ and evaluate $e_{val}(\mathcal{D}_i)$

   endfor

2. $E_{val} = \sum_{i=1}^{L} e_{val}(\mathcal{D}_i)$

if $L = N \implies K = 1$ the process is called **Leave-one-out**. Otherwise, the method is called L-**fold cross-validation**.

A question to think about...

A case

Say that you are writing an article on a new classification method with regularization and you have to compare two classification models. You want to report the error rate of both methods. However, both methods have to tune the regularization parameter. How do you split the data given to you? What do you use for model selection? And for reporting the out-of-sample error?

Wrapping up the lecture

- A non-linear model, i.e. decision trees, is introduced.
- We analyze the problem of overfitting.
- The bias-variance trade-off is shown.
- Two cures to the overfitting problem are described: regularization and validation.