Statistical Machine Learning

https://cvml.ist.ac.at/courses/SML_W20

Christoph Lampert

IST AUSTRIA
Institute of Science and Technology

Fall Semester 2020/2021
Lecture 5

Overview (tentative)

Date		no.	Topic
Oct 05	Mon	1	A Hands-On Introduction
Oct 07	Wed	2	Bayesian Decision Theory, Generative Probabilistic Models
Oct 12	Mon	3	Discriminative Probabilistic Models
Oct 14	Wed	4	Maximum Margin Classifiers, Generalized Linear Models
Oct 19	Mon	5	Estimators; Overfitting/Underfitting, Regularization, Model Selection
Oct 21	Wed	6	Bias/Fairness, Domain Adaptation
Oct 26	Mon	-	no lecture (public holiday)
Oct 28	Wed	7	Learning Theory I
Nov 02	Mon	8	Learning Theory II
Nov 04	Wed	9	Deep Learning I
Nov 09	Mon	10	Deep Learning II
Nov 11	Wed	11	Unsupervised Learning
Nov 16	Mon	12	project presentations
Nov 18	Wed	13	buffer

Evaluating Predictors

So, you've trained a predictor, $f : \mathcal{X} \to \mathcal{Y}$. How good is it really?

- The loss on the training set, $\hat{\mathcal{R}}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(y^i, f(x^i))$ tells us little about the quality of a learned predictor. Reporting it would be misleading as best.

- Really, we care about the expected loss (=generalization loss),

$$\mathcal{R}(f) = \mathop{\mathbb{E}}_{(x,y)\sim p(x,y)} \ell(y, f(x)).$$

  Unfornately, we cannot compute it, because $p(x, y)$ is unknown.

- In practice, we use a a test set, $\mathcal{D}_{\text{tst}} = \{\, (\bar{x}^1, \bar{y}^1), \dots, (\bar{x}^m, \bar{y}^m) \,\}$, i.e. examples that were not used for training, and compute the test loss

$$\hat{\mathcal{R}}_{\text{tst}}(f) = \frac{1}{m}\sum_{i=1}^{m} \ell(\bar{y}^i, f(\bar{x}^i))$$

  Why is that a good idea? Let's look at $\hat{\mathcal{R}}_{\text{tst}}(f)$ as an **estimator** of $\mathcal{R}(f)$.

Excurse: Estimators

Estimators

An estimator is a rule for calculating an estimate, $\hat{E}(S)$, of a quantity E based on observed data, S. If S is random, then $\hat{E}(S)$ is also random.

Properties of estimators: bias

Let $\hat{E}$ be an estimator of E. We can compute the expected value of the estimate, $\mathbb{E}_S[\hat{E}(S)]$, and define:

$$\mathrm{bias}(\hat{E}) = \mathbb{E}_S[\hat{E}(S)] - E$$

Properties of estimators: unbiasedness

If $\hat{E}$ is an estimator of E, we call it unbiased, if

$$\mathrm{bias}(\hat{E}) = 0 \qquad \text{(i.e. } \underset{S}{\mathbb{E}}[\hat{E}(S)] = E)$$

If $\hat{E}$ is unbiased, we can think of it as a noisy version of E.

Example: Estimating the mean of a Gaussian

Examples: let $S = \{z^1, z^2, \ldots, z^n\}$ be independent samples from $\mathcal{N}(x; \mu, \sigma^2)$. We look at different estimators for μ:

- $\hat{E}(S) = 1$ has bias $1 - \mu$. bias$(\hat{E}) = \mathbb{E}_S \, \hat{E}(S) - \mu = 1 - \mu$

Example: Estimating the mean of a Gaussian

Examples: let $S = \{z^1, z^2, \ldots, z^n\}$ be independent samples from $\mathcal{N}(x; \mu, \sigma^2)$. We look at different estimators for μ:

- $\hat{E}(S) = 1$ has bias $1 - \mu$. bias$(\hat{E}) = \mathbb{E}_S \, \hat{E}(S) - \mu = 1 - \mu$
- $\hat{E}(S) = \frac{1}{n} \sum_{i=1}^{n} z^i$ is unbiased.

Example: Estimating the mean of a Gaussian

Examples: let $S = \{z^1, z^2, \ldots, z^n\}$ be independent samples from $\mathcal{N}(x; \mu, \sigma^2)$. We look at different estimators for μ:

- $\hat{E}(S) = 1$ has bias $1 - \mu$. bias$(\hat{E}) = \mathbb{E}_S\, \hat{E}(S) - \mu = 1 - \mu$
- $\hat{E}(S) = \frac{1}{n}\sum_{i=1}^{n} z^i$ is unbiased.

$$\mathbb{E}_S[\hat{E}(S)] = \mathbb{E}_S[\tfrac{1}{n}\textstyle\sum_i z^i] = \tfrac{1}{n}\textstyle\sum_i \mathbb{E}_S[z^i] = \tfrac{1}{n}\textstyle\sum_i \mu = \mu$$

Example: Estimating the mean of a Gaussian

Examples: let $S = \{z^1, z^2, \ldots, z^n\}$ be independent samples from $\mathcal{N}(x; \mu, \sigma^2)$. We look at different estimators for μ:

- $\hat{E}(S) = 1$ has bias $1 - \mu$. bias$(\hat{E}) = \mathbb{E}_S\, \hat{E}(S) - \mu = 1 - \mu$
- $\hat{E}(S) = \frac{1}{n} \sum_{i=1}^{n} z^i$ is unbiased.

$$\mathbb{E}_S[\hat{E}(S)] = \mathbb{E}_S[\tfrac{1}{n} \textstyle\sum_i z^i] = \tfrac{1}{n} \textstyle\sum_i \mathbb{E}_S[z^i] = \tfrac{1}{n} \textstyle\sum_i \mu = \mu$$

- $\hat{E}(S) = z^1$ is unbiased: $\mathbb{E}_S[\hat{E}(S)] = \mathbb{E}_S[z^1] = \mu$

Example: Estimating the mean of a Gaussian

Examples: let $S = \{z^1, z^2, \dots, z^n\}$ be independent samples from $\mathcal{N}(x; \mu, \sigma^2)$. We look at different estimators for μ:

- $\hat{E}(S) = 1$ has bias $1 - \mu$. bias$(\hat{E}) = \mathbb{E}_S \, \hat{E}(S) - \mu = 1 - \mu$
- $\hat{E}(S) = \frac{1}{n}\sum_{i=1}^{n} z^i$ is unbiased.

$$\mathbb{E}_S[\hat{E}(S)] = \mathbb{E}_S[\tfrac{1}{n}\textstyle\sum_i z^i] = \tfrac{1}{n}\textstyle\sum_i \mathbb{E}_S[z^i] = \tfrac{1}{n}\textstyle\sum_i \mu = \mu$$

- $\hat{E}(S) = z^1$ is unbiased: $\mathbb{E}_S[\hat{E}(S)] = \mathbb{E}_S[z^1] = \mu$
- $\hat{E}(S) = \frac{1}{n} + \frac{1}{n}\sum_{i=1}^{n} z^i$ has bias $\frac{1}{n}$

Example: Stochastic Gradient Descent

Reminder: we wanted to optimize a function that is a sum of (many) terms:

$$f(\theta) = \sum_{j=1}^{n} f_j(\theta)$$

Instead of

$$v := \nabla f(\theta)$$

we use

$$\hat{v} := n \nabla f_i(\theta) \quad \text{with} \quad i \overset{\text{uniformly}}{\sim} \{1, \ldots, n\}$$

Claim: $\hat{v}$ is an unbiased estimator for v.

Example: Stochastic Gradient Descent

Reminder: we wanted to optimize a function that is a sum of (many) terms:

$$f(\theta) = \sum_{j=1}^{n} f_j(\theta)$$

Instead of

$$v := \nabla f(\theta)$$

we use

$$\hat{v} := n \nabla f_i(\theta) \quad \text{with} \quad i \overset{\text{uniformly}}{\sim} \{1, \ldots, n\}$$

Claim: $\hat{v}$ is an unbiased estimator for v.

$$\mathop{\mathbb{E}}_{i}[\hat{v}] = \sum_{i=1}^{n} p(i) \hat{v}[i] = \sum_{i=1}^{n} \frac{1}{n} n \nabla f_i(\theta) = \sum_{i=1}^{n} \nabla f_i(\theta) = \nabla f(\theta)$$

If we get a single, $\hat{E}(S)$, how far is it going to be from its expected value, $\mathbb{E}_S[\hat{E}(S)]$?

Properties of estimators: variance

$$\mathrm{Var}(\hat{E}) = \mathbb{E}_S\Big[\big(\,\hat{E}(S) - \mathbb{E}_S[\hat{E}(S)]\,\big)^2\Big]$$

If $\mathrm{Var}(\hat{E})$ is large, then the estimate for different S differ a lot.

Examples:

Let $S = \{z^1, z^2, \dots, z^n\}$ be independent samples from $\mathcal{N}(x; \mu, \sigma^2)$.
We look at different estimators for μ:

- $\hat{E}(S) = 1$ has variance 0.
- $\hat{E}(S) = \frac{1}{n}\sum_{i=1}^{n} z_i$ has variance $\frac{\sigma^2}{n}$ *(exercise)*
- $\hat{E}(S) = z_1$ has variance σ^2
- $\hat{E}(S) = \frac{1}{n-1}\sum_{i=1}^{n} z_i$ has variance ? *(exercise)*

Bias-Variance Trade-Off

It's good to have small or no bias, and it's good to have small variance.

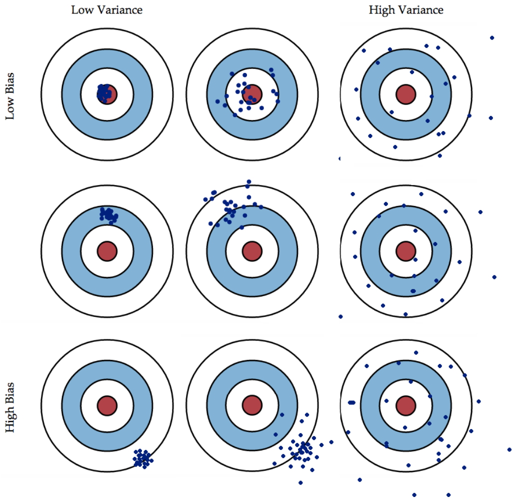


Image: adapted from http://scott.fortmann-roe.com/docs/BiasVariance.html

If you can't have both at the same time, look for a reasonable trade-off.

Consistency

What if we get more and more data, $S_n = \{z_1, \dots, z_n\}$ for $n \to \infty$?

Properties of estimators: consistency

An estimator $\hat{E}$ is called consistent, if

$$\hat{E}(S_n) \to E \qquad \text{for} \quad n \to \infty.$$

Convergence is "in probability", i.e. it means,

$$\lim_{n\to\infty} \Pr\{\ |\hat{E}(S_n) - E| > \epsilon\ \} = 0.$$

Any estimator $\hat{E}$ with bias$(\hat{E}) \overset{n\to\infty}{\to} 0$ and Var$(\hat{E}) \overset{n\to\infty}{\to} 0$ is consistent.

Proof... follows from later observations

Back to machine learning

Test set loss as an estimator of the risk

Is the test set loss

$$\hat{\mathcal{R}}_{\mathsf{tst}}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(\, y_i, f(x_i))$$

a good estimator of

$$\mathcal{R}(f) = \mathop{\mathbb{E}}_{(x,y)\sim p(x,y)} \ell(\, y, f(x)\,)$$

Test set loss as an estimator of the risk

Is the test set loss

$$\hat{\mathcal{R}}_{\text{tst}}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(\, y_i, f(x_i))$$

a good estimator of

$$\mathcal{R}(f) = \mathop{\mathbb{E}}_{(x,y)\sim p(x,y)} \ell(\, y, f(x)\,)$$

Yes, if we use the right data:

Test error as an unbiased estimator

Let ℓ be a bounded loss function, i.e. $\ell(y,\bar{y}) \in [0, M]$ for some $M > 0$. If the test set data $\mathcal{D}_{\text{tst}} = \{\,(x^1, y^1), \dots, (x^m, y^m)\,\}$ is sampled i.i.d. from the distribution $p(x, y)$, and f was chosen independently of them, then $\hat{\mathcal{R}}_{\text{tst}}(f)$ is an unbiased and consistent estimator of $\mathcal{R}(f)$:

Otherwise? Things might go wrong (exercise).

Proof: unbiasedness

- $\mathcal{D}$ is a set of random variables, $(X^1, Y^1), \dots, (X^m, Y^m) \in \mathcal{X} \times \mathcal{Y}$.
- All $(X^1, Y^1), \dots, (X^m, Y^m)$ are independent with distribution p.
- For fixed functions f, ℓ, chosen independently of $\mathcal{D}$

$$\ell(Y^1, f(X^1)),\ \dots\ ,\ \ell(Y^m, f(X^m))$$

are independent (real-valued) random variables

$$\begin{aligned}
\mathop{\mathbb{E}}_{\mathcal{D} \overset{i.i.d.}{\sim} p} \hat{\mathcal{R}}_{\mathsf{tst}}(f) &= \mathop{\mathbb{E}}_{(X^1,Y^1),\dots,(X^m,Y^m)\sim p} \frac{1}{m} \sum_{i=1}^{m} \ell(\,Y^i, f(X^i)\,) \\
&= \frac{1}{m} \sum_{i=1}^{m} \mathop{\mathbb{E}}_{(X^1,Y^1),\dots,(X^m,Y^m)\sim p} \ell(\,Y^i, f(X^i)\,) \\
&= \frac{1}{m} \sum_{i=1}^{m} \mathop{\mathbb{E}}_{(X^i,Y^i)\sim p} \ell(\,Y^i, f(X^i)\,) \\
&= \frac{1}{m} \sum_{i=1}^{m} \mathop{\mathbb{E}}_{(X,Y)\sim p} \ell(\,Y, f(X)\,) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{R}(f) = \mathcal{R}(f) \qquad \square
\end{aligned}$$

Learning from Data

In the real world, $p(x, y)$ is unknown, but we have a training set $\mathcal{D}$.

Definition

Given a training set $\mathcal{D}$, we call it

- a **generative probabilistic approach**:
  if we use $\mathcal{D}$ to build a model $\hat{p}(x, y)$ of $p(x, y)$, and then define

$$f(x) := \operatorname*{\mathbf{argmin}}_{y \in \mathcal{Y}} \mathop{\mathbb{E}}_{\bar{y} \sim \hat{p}(x, \bar{y})} \ell(\bar{y}, y).$$

- a **discriminative probabilistic approach**:
  if we use $\mathcal{D}$ to build a model $\hat{p}(y|x)$ of $p(y|x)$ and define

$$f(x) := \operatorname*{\mathbf{argmin}}_{y \in \mathcal{Y}} \mathop{\mathbb{E}}_{\bar{y} \sim \hat{p}(\bar{y}|x)} \ell(\bar{y}, y).$$

- a **decision theoretic approach**: if we use $\mathcal{D}$ to directly seach for a classifier f.

Definition (Empirical Risk Minimization)

Given a training set $\mathcal{D} = \{\, (x^1, y^1), \dots, (x^n, y^n) \,\}$, we call it **empirical risk minimization (ERM)**, if we find a classifier by minimizing the empirical risk:

$$f := \operatorname*{\mathbf{argmin}}_{h \in \mathcal{H}} \hat{\mathcal{R}}(h) \qquad \text{for} \quad \hat{\mathcal{R}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(\, y^i, f(x^i))$$

where $\mathcal{H} \subset \{h : \mathcal{X} \to \mathcal{Y}\}$ is called the hypothesis set.

Unfortunately, ERM is often **NP-hard**, including for 0/1-loss [Marcotte, Savard. 1992]

→ for all practically relevant problem sizes, we don't solve it exactly but find an approximate solution by minimizing a surrogate loss function instead. e.g.

- hinge loss: $\mathcal{L}(y, t) = \mathbf{max}\{1 - yt, 0\}$ → support vector machine
- binary logistic loss: $\mathcal{L}(y, t) = \log(1 + \exp(-yt))$ → logistic regression

Empirical Risk Minimization

What we want:

1) first choose $f : \mathcal{X} \to \mathcal{Y}$, then observe $\mathcal{D} = \{(x^1, y^1), \dots, (x^n, y^n)\}$:

$$\hat{\mathcal{R}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(\,y^i, f(x^i)) \quad \text{unbiased, consistent estimator of } \mathcal{R}(f)$$

What we do:

2) first observe $\mathcal{D} = \{(x^1, y^1), \dots, (x^n, y^n)\}$, then choose f based on $\mathcal{D}$:

$$\hat{\mathcal{R}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(\,y^i, f(x^i)) \qquad \text{not an unbiased estimator of } \mathcal{R}(f)$$

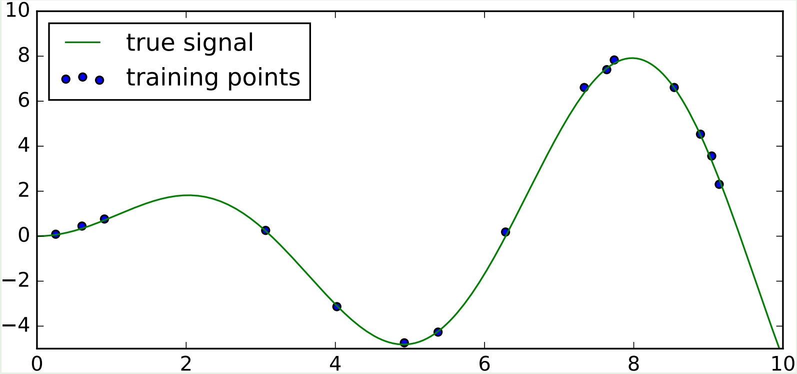
- $\ell_i := \ell(\,y^i, f(x^i))$ are not independent RVs, law of large numbers does not apply

Why would it make sense to do 2), if what we want is 1)?

The relation between training and generalization loss

Relation between training loss and generalization loss

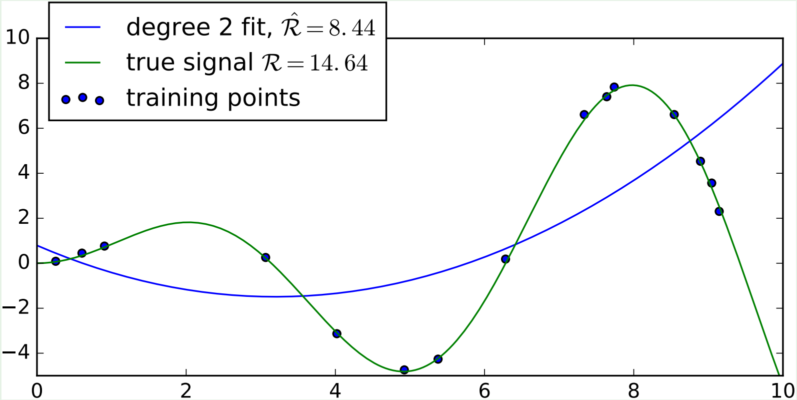
Example: 1D curve fitting



training points

Relation between training loss and generalization loss

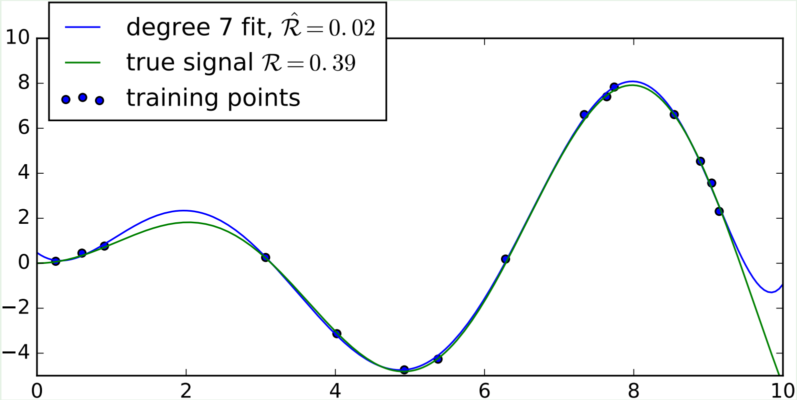
Example: 1D curve fitting



best learned polynomial of degree 2: large $\hat{\mathcal{R}}$, large $\mathcal{R}$

Relation between training loss and generalization loss

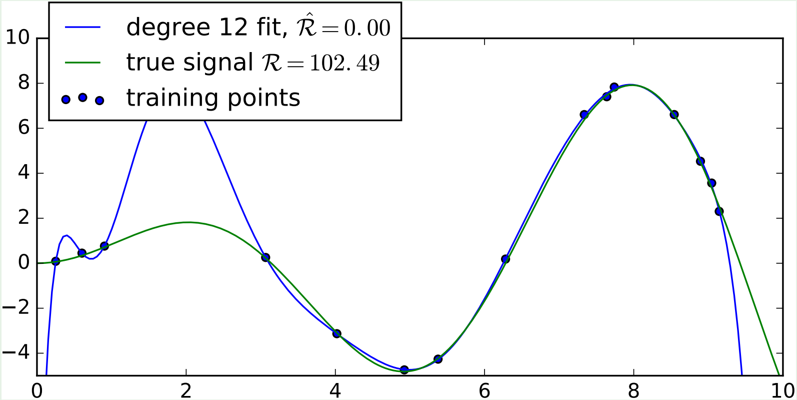
Example: 1D curve fitting



best learned polynomial of degree 7: small $\hat{\mathcal{R}}$, small $\mathcal{R}$

Relation between training loss and generalization loss

Example: 1D curve fitting



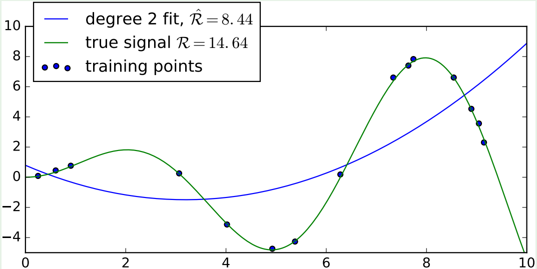
best learned polynomial of degree 12: small $\hat{\mathcal{R}}$, large $\mathcal{R}$

We found a model f_{θ^*} by minimizing the training error $\hat{\mathcal{R}}$.

Q: Will its generalization error, $\mathcal{R}$, be small?

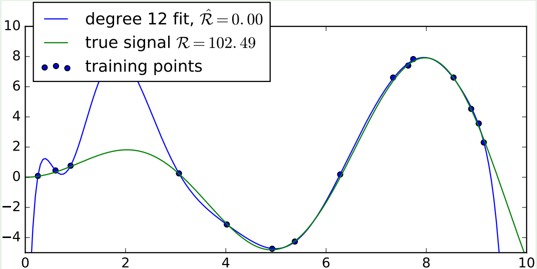
A: **Unfortunately, that is not guaranteed.**

Underfitting/Overfitting



Underfitting

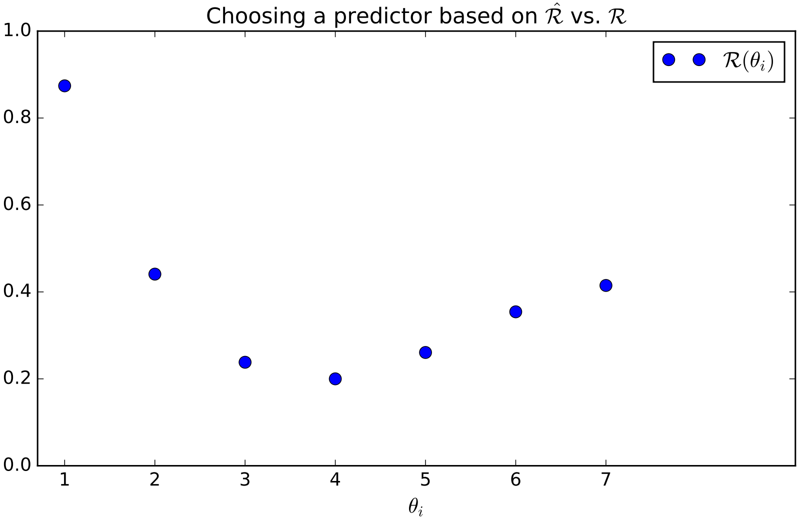
(to some extend) detectable from $\hat{\mathcal{R}}$



Overfitting

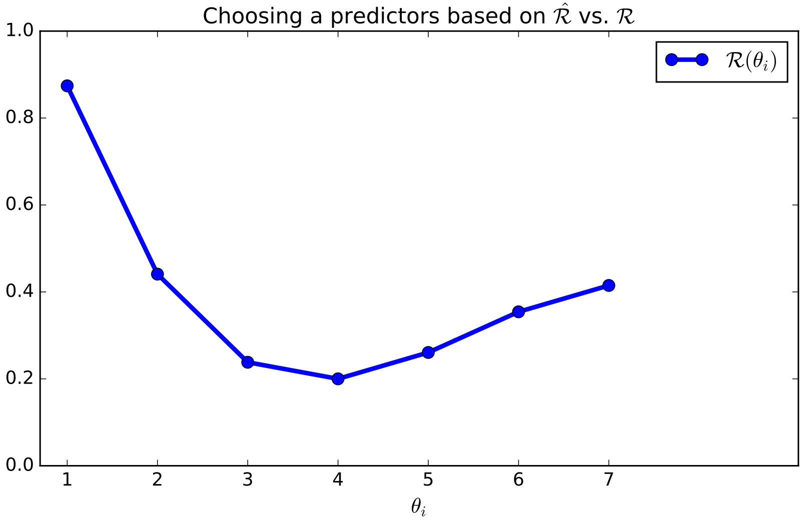
not detectable from $\hat{\mathcal{R}}$!

Where does overfitting come from?



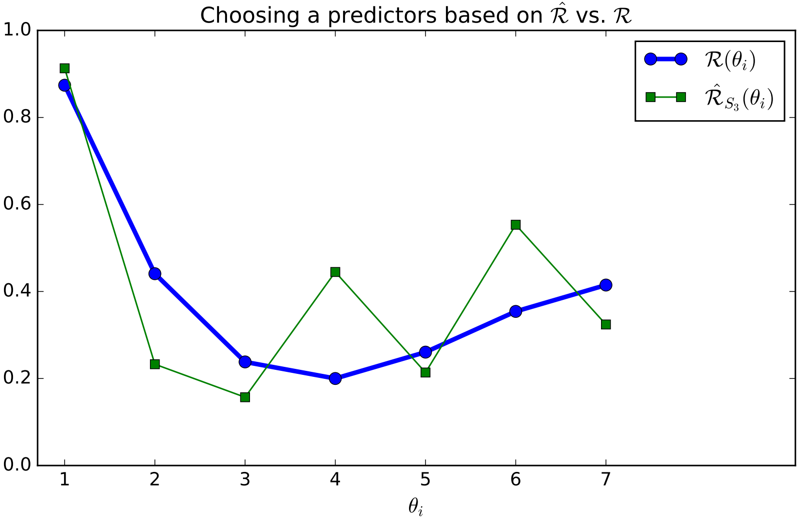
generalization error $\mathcal{R}$ for 7 different predictors (x-axis)

Where does overfitting come from?



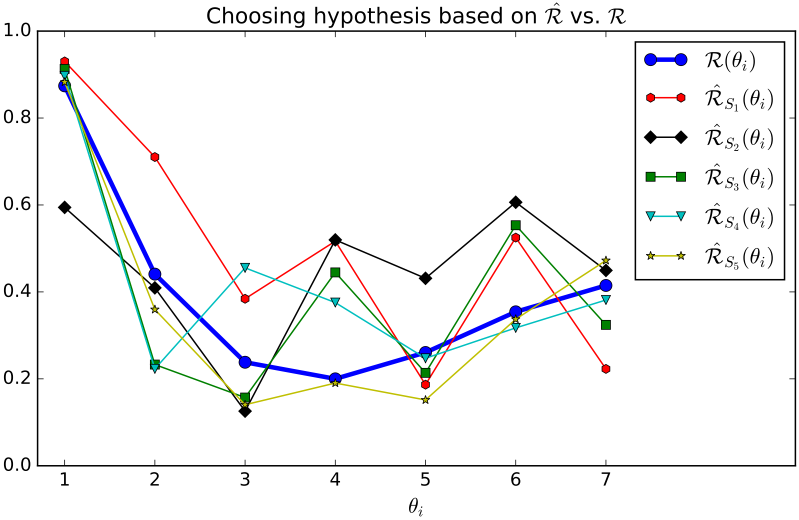
generalization error $\mathcal{R}$ for 7 different predictors (x-axis)

Where does overfitting come from?



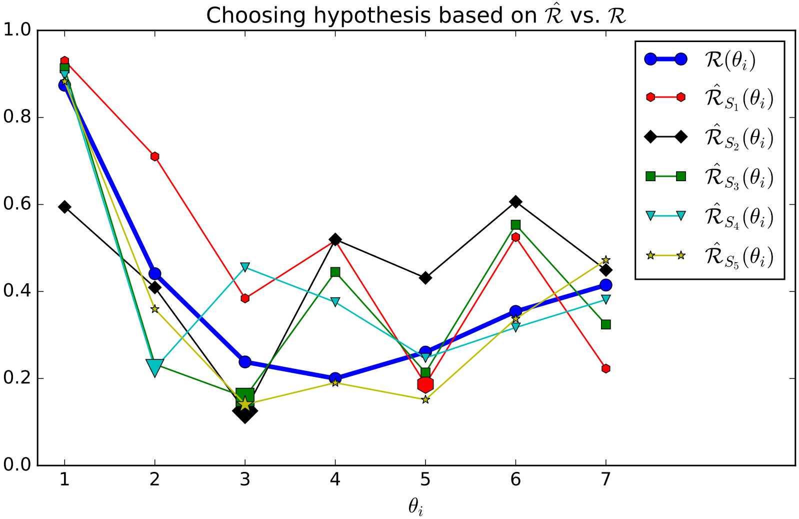
training error $\hat{\mathcal{R}}$ for a training set, S

Where does overfitting come from?



training errors $\hat{\mathcal{R}}$ for 5 possible training sets $(S_1, \dots, X_5)$

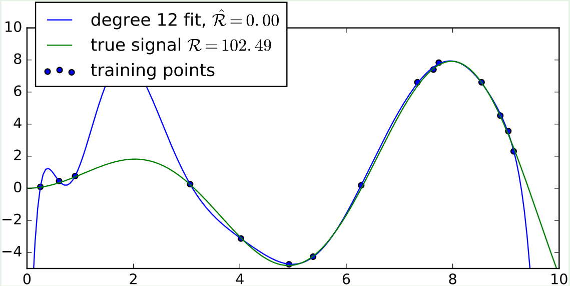
Where does overfitting come from?



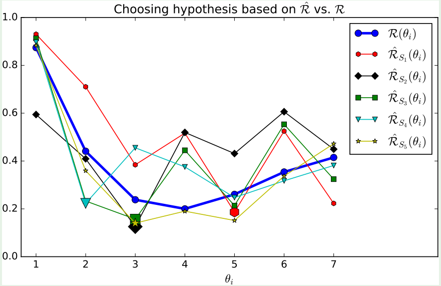
model with smallest training error might have high generalization error → overfitting

Preventing Overfitting

Reminder: Overfitting



overfitting

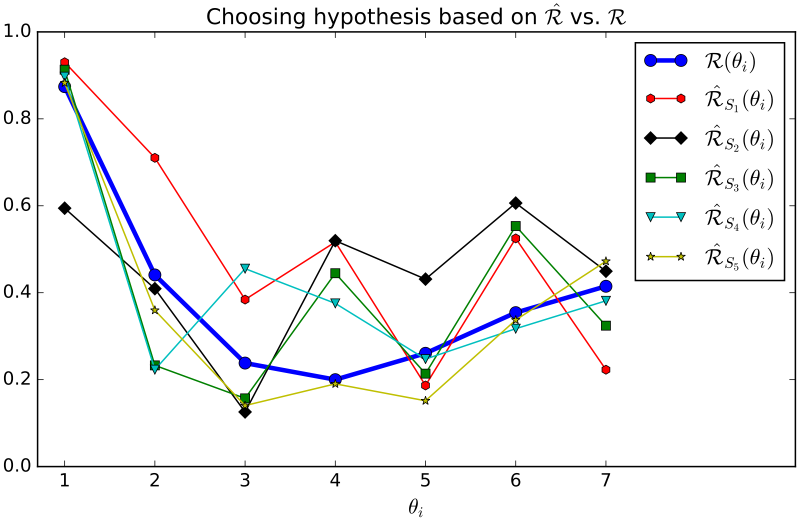


$\hat{\mathcal{R}}$ **vs.** $\mathcal{R}$

How can we prevent overfitting when learning a model?

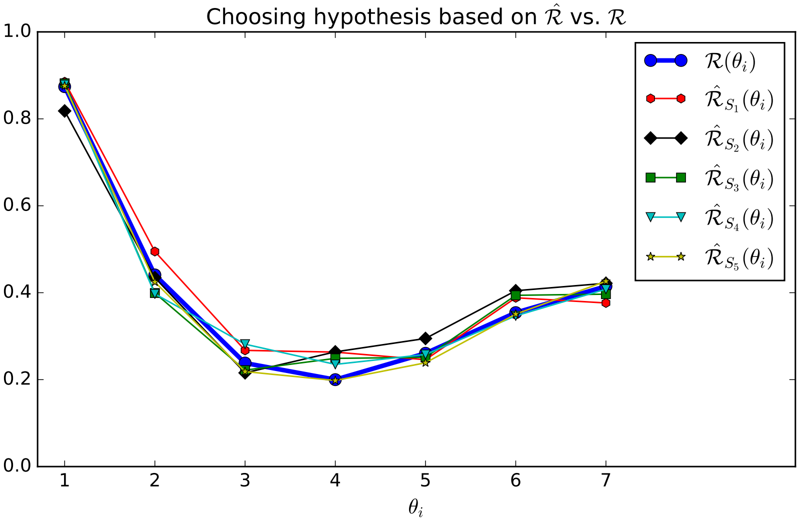
Preventing overfitting

Preventing overfitting 1) larger training set



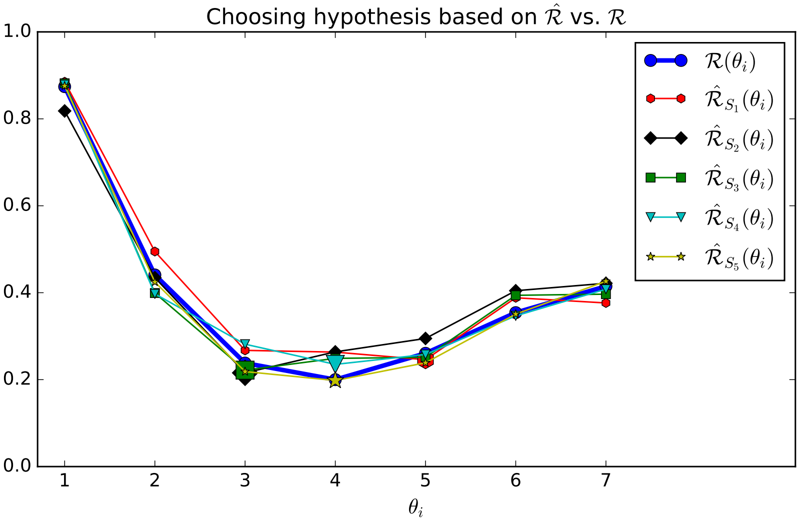
larger training set → smaller variance of $\hat{\mathcal{R}}$

Preventing overfitting 1) larger training set



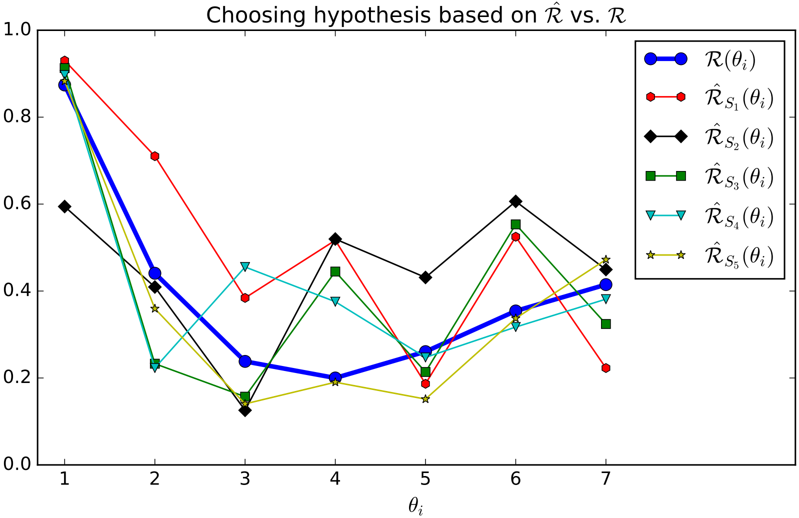
lower probability that $\hat{\mathcal{R}}$ differs strongly from $\mathcal{R}$

Preventing overfitting 1) larger training set

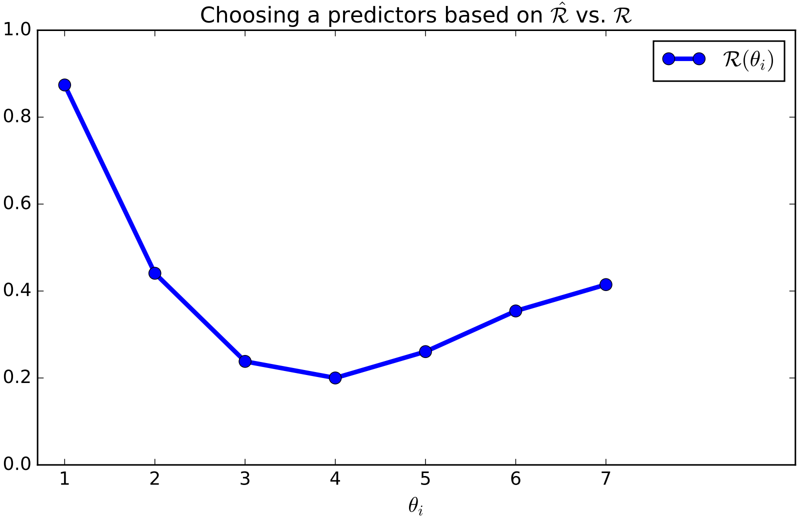


lower probability that $\hat{\mathcal{R}}$ differs strongly from $\mathcal{R}$ → overfitting less likely

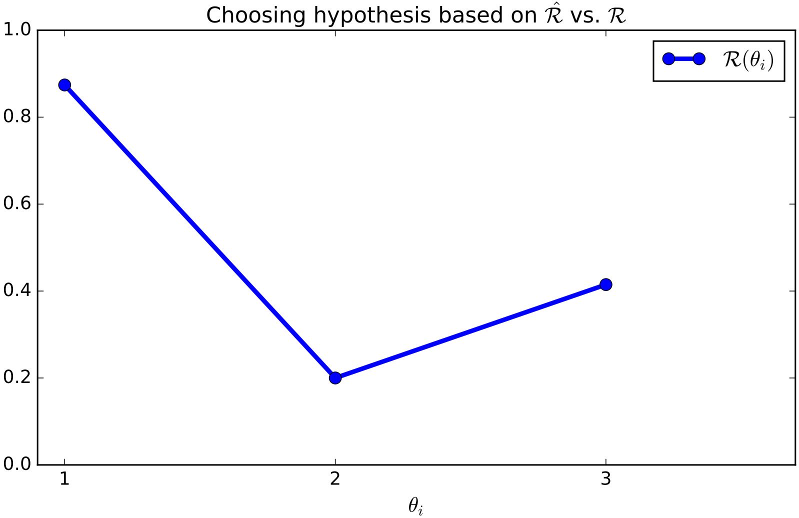
Preventing overfitting 2) reduce the number of hypotheses



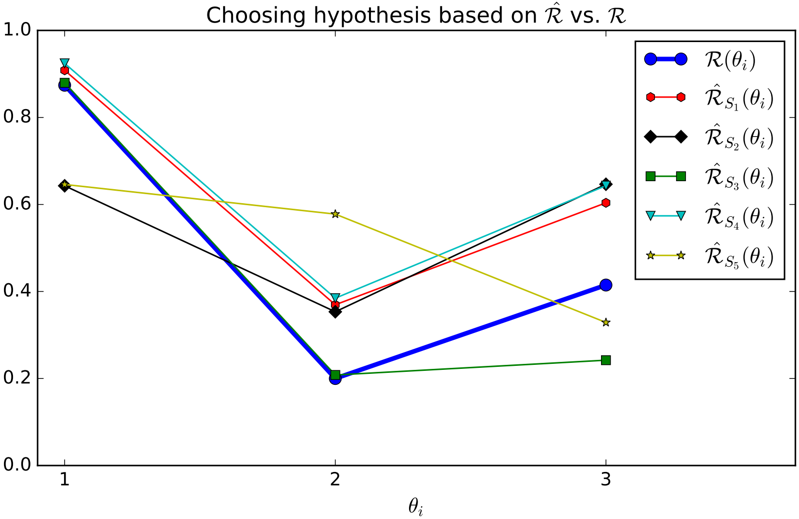
Preventing overfitting 2) reduce the number of hypotheses



Preventing overfitting 2) reduce the number of hypotheses

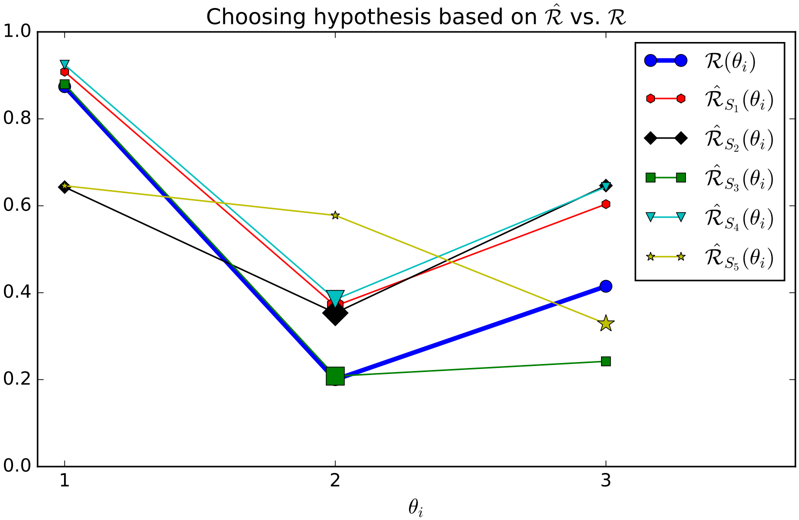


Preventing overfitting 2) reduce the number of hypotheses



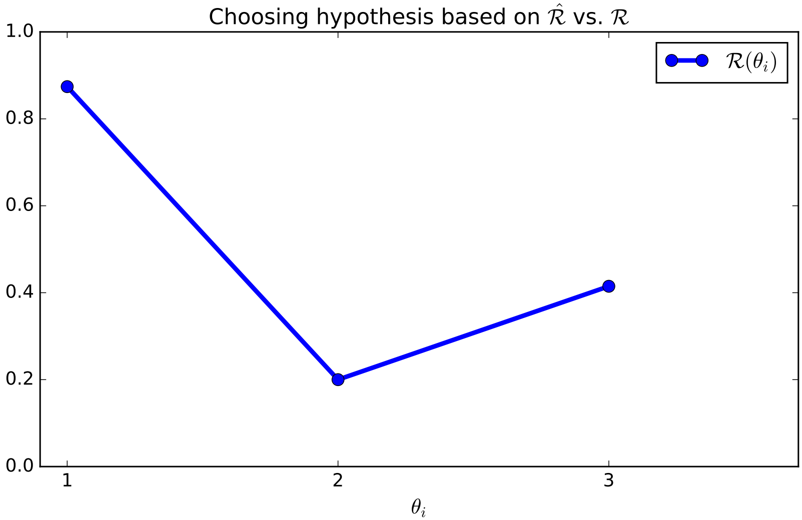
fewer models → lower probability of a model with small $\hat{\mathcal{R}}$ but high $\mathcal{R}$

Preventing overfitting 2) reduce the number of hypotheses

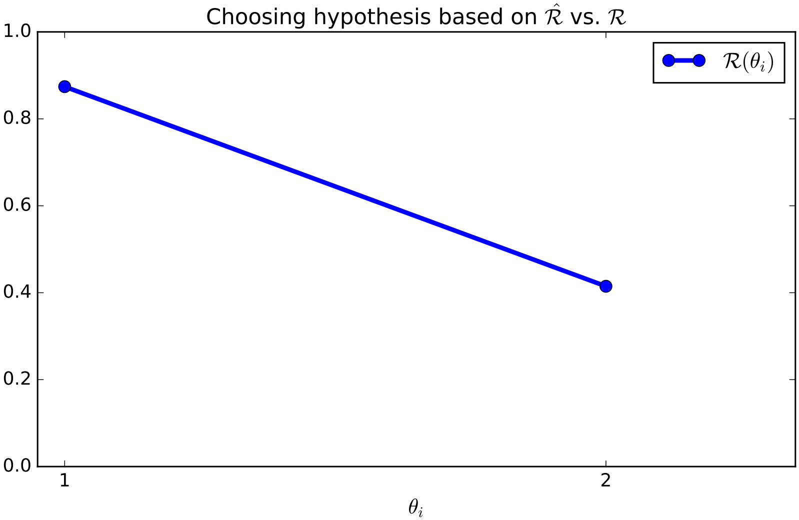


fewer models → lower probability of a model with small $\hat{\mathcal{R}}$ but high $\mathcal{R}$

But: danger of underfitting

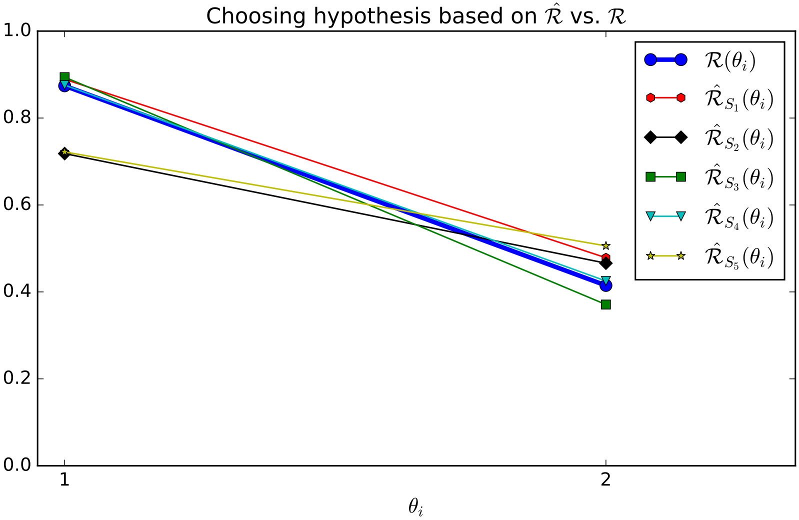


But: danger of underfitting



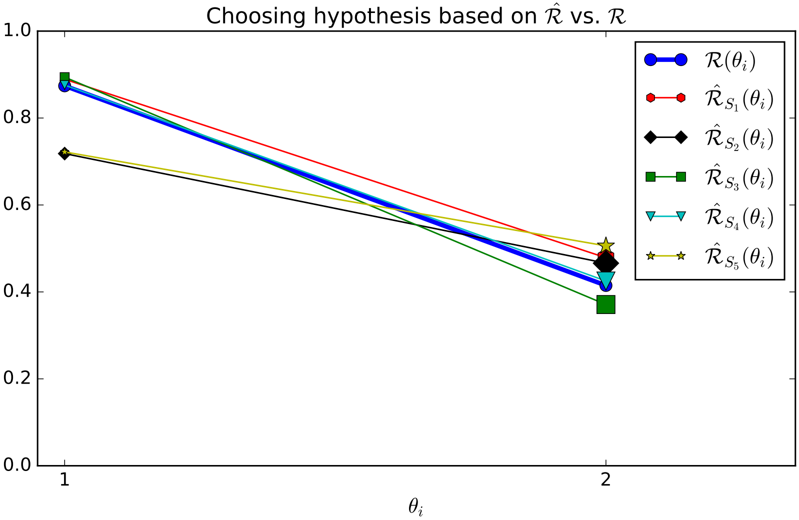
to few models select to from → danger that no model with low $\mathcal{R}$ is left!

But: danger of underfitting



all models have high train (and test) error → **Underfitting!**

But: danger of underfitting



all models have high train (and test) error $\rightarrow$ **Underfitting!**

Overfitting happens when . . .

- there are too many models to choose from
  (not strictly true: there's usually infinitely many models anyway)

- the models we search over are too "flexible", so they fit not only the signal but also noise
  (not strictly true: the models themselves are not "flexible" at all)

- the models have too many free parameters
  (not strictly true: even models with very few parameters can overfit)

How to avoid overfitting?

- 1) Use a model class that
  - is "as simple as possible", but still contains a model with low $\hat{\mathcal{R}}$

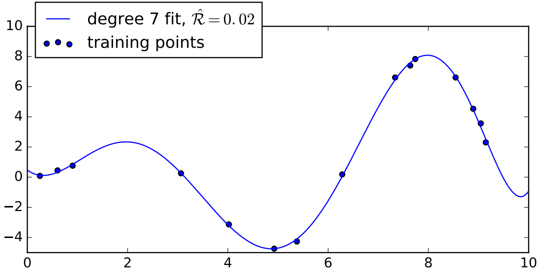
- 2) Use a training algorithm that
  - implicitly has a preference for "simple" rather than complex models,

Regularization

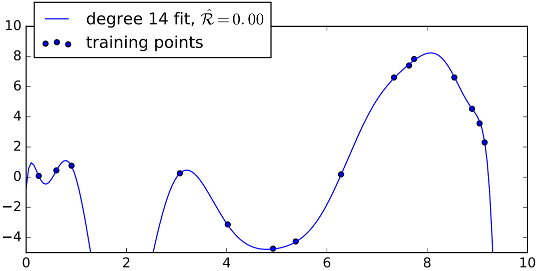
Regularization

Models with big difference between $\hat{\mathcal{R}}$ and $\mathcal{R}$ are often **extreme cases**:

- a large number of model parameters
- large values of the model parameters
- for polynomials: high degree , etc.



coeffs: $\theta_i \in [-2.4, 4.6]$

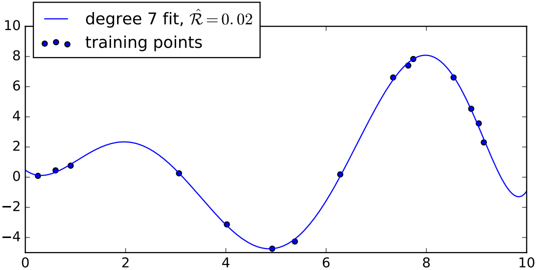


coeffs: $\theta_i \in [-1312.5, 1136.6]$

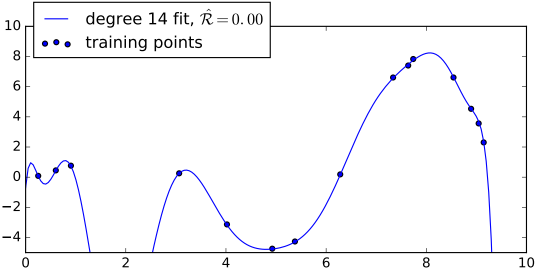
Regularization

Models with big difference between $\hat{\mathcal{R}}$ and $\mathcal{R}$ are often **extreme cases**:

- a large number of model parameters
- large values of the model parameters
- for polynomials: high degree ,   etc.



coeffs: $\theta_i \in [-2.4, 4.6]$



coeffs: $\theta_i \in [-1312.5, 1136.6]$

Regularization: avoid overfitting by preventing extremes to occur

- explicit regularization (changing the objective function)
- implicit regularization (modifying the optimization procedure)

Explicit regularization

Add a regularization term (=regularizer) to the empirical risk that gives large values to extreme parameter choices. The minimization will be discouraged away from such choices.

Regularized risk minimization

Take a training set, $S = \{(x^1, y^1), \dots, (x^n, y^n)\}$, find θ^* by solving,

$$\min_{\theta} J_\lambda(\theta) \quad \text{with} \quad J_\lambda(\theta) = \underbrace{\sum_{i=1}^{n} \ell(y^i, f_\theta(x^i))}_{\text{empirical risk}} + \underbrace{\lambda\Omega(\theta)}_{\text{regularizer}}$$

e.g. with $\quad \Omega(\theta) = \|\theta\|_{L^2}^2 = \sum_j \theta_j^2 \qquad$ or $\qquad \Omega(\theta) = \|\theta\|_{L^1} = \sum_j |\theta_j|$

Optimization looks for model with small empirical risk, but also small absolute values of the model parameters.

Explicit regularization

Add a regularization term (=regularizer) to the empirical risk that gives large values to extreme parameter choices. The minimization will be discouraged away from such choices.

Regularized risk minimization

Take a training set, $S = \{(x^1, y^1), \dots, (x^n, y^n)\}$, find θ^* by solving,

$$\min_{\theta} J_\lambda(\theta) \quad \text{with} \quad J_\lambda(\theta) = \underbrace{\sum_{i=1}^{n} \ell(y^i, f_\theta(x^i))}_{\text{empirical risk}} + \underbrace{\lambda\Omega(\theta)}_{\text{regularizer}}$$

$$\text{e.g. with} \quad \Omega(\theta) = \|\theta\|_{L^2}^2 = \sum_j \theta_j^2 \qquad \text{or} \qquad \Omega(\theta) = \|\theta\|_{L^1} = \sum_j |\theta_j|$$

Regularization (hyper)parameter $\lambda \geq 0$: trade-off between both.

- $\lambda = 0$: (unregularized) empirical risk minimization → risk of overfitting
- $\lambda \to \infty$: all parameters 0 → risk of underfitting

Explicit regularization

Add a regularization term (=regularizer) to the empirical risk that gives large values to extreme parameter choices. The minimization will be discouraged away from such choices.

Regularized risk minimization

Take a training set, $S = \{(x^1, y^1), \dots, (x^n, y^n)\}$, find θ^* by solving,

$$\min_\theta J_\lambda(\theta) \quad \text{with} \quad J_\lambda(\theta) = \underbrace{\sum_{i=1}^{n} \ell(y^i, f_\theta(x^i))}_{\text{empirical risk}} + \underbrace{\lambda\Omega(\theta)}_{\text{regularizer}}$$

$$\text{e.g. with} \quad \Omega(\theta) = \|\theta\|_{L^2}^2 = \sum_j \theta_j^2 \qquad \text{or} \qquad \Omega(\theta) = \|\theta\|_{L^1} = \sum_j |\theta_j|$$

Examples:

- Ridge Regression: $\quad \mathbf{min}_w \quad \lambda\|w\|^2 + \sum_i (\langle w, x^i\rangle - y^i)^2$
- Logistic Regression: $\quad \mathbf{min}_w \quad \lambda\|w\|^2 + \sum_i \log(1 + e^{-y^i\langle w, x^i\rangle})$
- SVM: $\quad \mathbf{min}_w \quad \frac{1}{2}\|w\|^2 + C\sum_i \mathbf{max}\{0, 1 - y^i\langle w, x^i\rangle\} \qquad \leftarrow \qquad C \hat{=} \frac{1}{2\lambda}$

Regularization as Trading Off Bias and Variance

Training error, $\hat{\mathcal{R}}$, is a noise estimate of the generalization error, $\mathcal{R}$

- original risk $\hat{\mathcal{R}}$ is unbiased, but variance can be huge
- regularization introduces a bias, but reduces variance
- for $\lambda \to \infty$, the variance goes to 0, but the bias gets very big

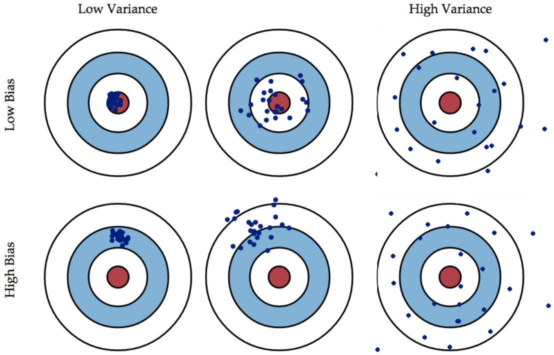


Image: adapted from http://scott.fortmann-roe.com/docs/BiasVariance.html

Regularization as Trading Off Bias and Variance

Training error, $\hat{\mathcal{R}}$, is a noise estimate of the generalization error, $\mathcal{R}$

- original risk $\hat{\mathcal{R}}$ is unbiased, but variance can be huge
- regularization introduces a bias, but reduces variance
- for $\lambda \to \infty$, the variance goes to 0, but the bias gets very big

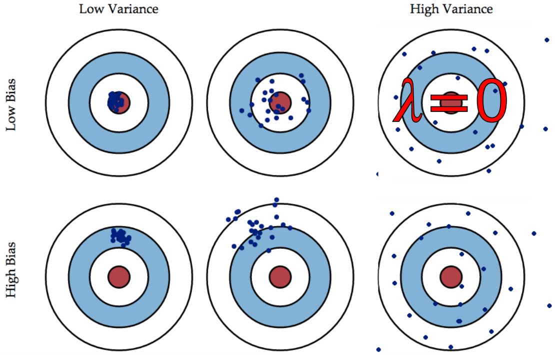


Image: adapted from http://scott.fortmann-roe.com/docs/BiasVariance.html

Regularization as Trading Off Bias and Variance

Training error, $\hat{\mathcal{R}}$, is a noise estimate of the generalization error, $\mathcal{R}$

- original risk $\hat{\mathcal{R}}$ is unbiased, but variance can be huge
- regularization introduces a bias, but reduces variance
- for $\lambda \to \infty$, the variance goes to 0, but the bias gets very big

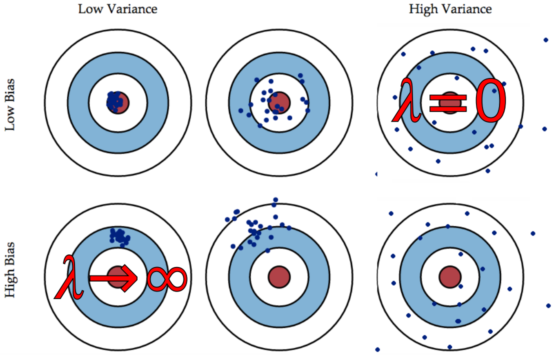


Image: adapted from http://scott.fortmann-roe.com/docs/BiasVariance.html

Regularization as Trading Off Bias and Variance

Training error, $\hat{\mathcal{R}}$, is a noise estimate of the generalization error, $\mathcal{R}$

- original risk $\hat{\mathcal{R}}$ is unbiased, but variance can be huge
- regularization introduces a bias, but reduces variance
- for $\lambda \to \infty$, the variance goes to 0, but the bias gets very big

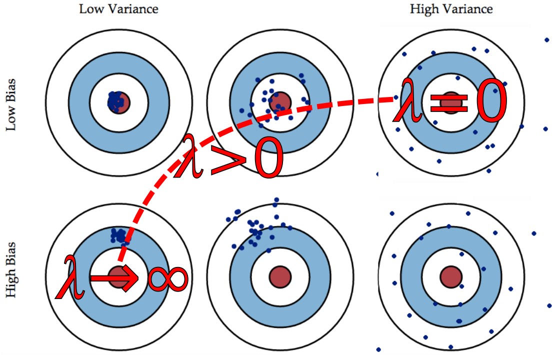


Image: adapted from http://scott.fortmann-roe.com/docs/BiasVariance.html

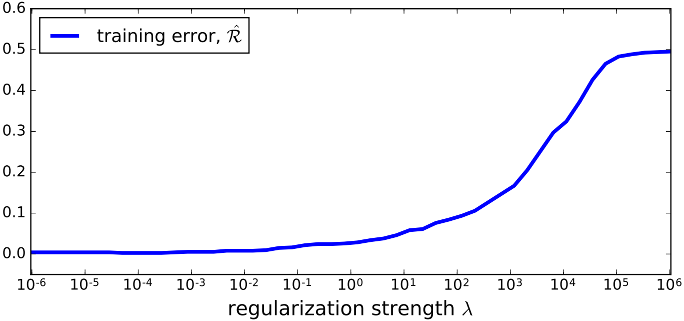
Example: regularized least-squares classifier

$$\min_{w} J_\lambda(w) \quad \text{for} \quad J_\lambda(w) = \sum_{i=1}^{n} (w^\top x^i - y^i)^2 + \lambda \|w\|^2$$

Example: regularized least-squares classifier

$$\min_w J_\lambda(w) \quad \text{for} \quad J_\lambda(w) = \sum_{i=1}^{n} (w^\top x^i - y^i)^2 + \lambda \|w\|^2$$

Train/test error for classifier $c(x) = \operatorname{sign}\langle w, x\rangle$ with w obtained by minimizing J_λ with varying amounts of regularization:

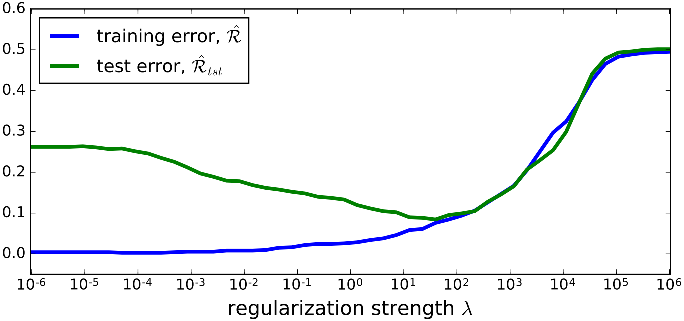


eye dataset: 737 examples for training, 736 examples for evaluation

Example: regularized least-squares classifier

$$\min_{w} J_\lambda(w) \quad \text{for} \quad J_\lambda(w) = \sum_{i=1}^{n} (w^\top x^i - y^i)^2 + \lambda \|w\|^2$$

Train/test error for classifier $c(x) = \operatorname{sign}\langle w, x\rangle$ with w obtained by minimizing J_λ with varying amounts of regularization:

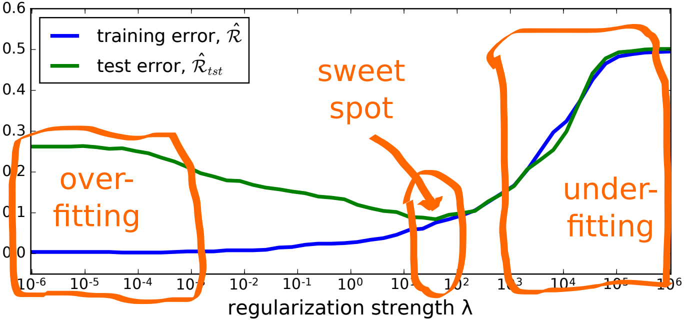


eye dataset: 737 examples for training, 736 examples for evaluation

Example: regularized least-squares classifier

$$\min_{w} J_\lambda(w) \quad \text{for} \quad J_\lambda(w) = \sum_{i=1}^{n} (w^\top x^i - y^i)^2 + \lambda \|w\|^2$$

Train/test error for classifier $c(x) = \operatorname{sign}\langle w, x \rangle$ with w obtained by minimizing J_λ with varying amounts of regularization:



eye dataset: 737 examples for training, 736 examples for evaluation

Implicit regularization

Numerical optimization is performed iteratively, e.g. gradient descent

Gradient descent optimization

```
initialize $\theta^{(0)}$
**for** $t = 1, 2, \dots$
    $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta_t \nabla_\theta J(\theta^{(t-1)})$        ($\eta_t \in \mathbb{R}$ is some stepsize rule)
**until convergence**
```

Implicit regularization methods modify these steps, e.g.

- early stopping
- weight decay
- data augmentation/jittering
- dropout

Implicit regularization: early stopping

Gradient descent optimization with early stopping

```
initialize $\theta^{(0)}$
for $t = 1, 2, \ldots, T$        ($T \in \mathbb{N}$ is number of steps)
    $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta_t \nabla_\theta J(\theta^{(t-1)})$
```

Implicit regularization: early stopping

Gradient descent optimization with early stopping

```
initialize $\theta^{(0)}$
for $t = 1, 2, \dots, T$        ($T \in \mathbb{N}$ is number of steps)
    $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta_t \nabla_\theta J(\theta^{(t-1)})$
```

Early stopping: stop optimization before convergence

- idea: if parameters are update only a small number of time, they might not reach extreme values
- T hyperparameter controls trade-off:
  - large T: parameters approach risk minimizer → risk of overfitting
  - small T: parameters stay close to initialization → risk of underfitting

Implicit regularization: weight decay

Gradient descent optimization with weight decay

```
initialize $\theta^{(0)}$
**for** $t = 1, 2, \ldots$
    $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta_t \nabla_\theta J(\theta^{(t-1)})$
    $\theta^{(t)} \leftarrow \gamma\theta^{(t)}$        for, e.g., $\gamma = 0.99$
**until convergence**
```

Implicit regularization: weight decay

Gradient descent optimization with weight decay

```
initialize $\theta^{(0)}$
for $t = 1, 2, \dots$
    $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta_t \nabla_\theta J(\theta^{(t-1)})$
    $\theta^{(t)} \leftarrow \gamma\theta^{(t)}$        for, e.g., $\gamma = 0.99$
until convergence
```

Weight decay:

Multiply parameters with a constant smaller than 1 in each iteration

- two 'forces' in parameter update:
  - $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta_t \nabla_\theta J(\theta^{(t-1)})$
    pull towards empirical risk minimizer → risk of overfitting
  - $\theta^{(t)} \leftarrow \gamma\theta^{(t)}$ pulls towards 0 → risk of underfitting
- convergence: both effects cancel out → trade-off controlled by η_t, γ

Note: essentially same effect as explicit regularization with $\Omega = \frac{\gamma}{2}\|\theta\|^2$

Implicit regularization: data augmentation (="jittering", "virtual samples")

Gradient descent optimization with data augmentation

```
initialize $\theta^{(0)}$
for $t = 1, 2, \dots$
    for $i = 1, \dots, n$:
        $\tilde{x}^i \leftarrow$ randomly perturbed version of $x^i$
    set $\tilde{J}(\theta) = \sum_{i=1}^{n} \ell(y^i, f_\theta(\tilde{x}^i))$
    $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta_t \nabla_\theta \tilde{J}(\theta^{(t-1)})$
until convergence
```

Implicit regularization: data augmentation (="jittering", "virtual samples")

Gradient descent optimization with data augmentation

```
initialize $\theta^{(0)}$
for $t = 1, 2, \dots$
    for $i = 1, \dots, n$:
        $\tilde{x}^i \leftarrow$ randomly perturbed version of $x^i$
    set $\tilde{J}(\theta) = \sum_{i=1}^{n} \ell(y^i, f_\theta(\tilde{x}^i))$
    $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta_t \nabla_\theta \tilde{J}(\theta^{(t-1)})$
until convergence
```

Data augmentation: use randomly perturbed examples in each iteration

- idea: a good model should be robust to small changes of the data
- simulate (infinitely-)large training set → hopefully less overfitting

  (also possible: just create large training set of jittered examples in the beginning)
- problem: coming up with reasonable perturbations needs *domain knowledge*

Implicit regularization: dropout

Gradient descent optimization with dropout

```
initialize $\theta^{(0)}$
for $t = 1, 2, \dots$
    $\tilde{\theta} \leftarrow \theta^{(t-1)}$ with a random fraction $p$ of values set to $0$, e.g. $p = \frac{1}{2}$
    $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta_t \nabla_\theta J(\tilde{\theta})$
until convergence
```

Implicit regularization: dropout

Gradient descent optimization with dropout

```
initialize $\theta^{(0)}$
for $t = 1, 2, \ldots$
    $\tilde{\theta} \leftarrow \theta^{(t-1)}$ with a random fraction $p$ of values set to $0$, e.g. $p = \frac{1}{2}$
    $\theta^{(t)} \leftarrow \theta^{(t-1)} - \eta_t \nabla_\theta J(\tilde{\theta})$
until convergence
```

Dropout: every time we evaluate the model, a random subset of its parameters are set to zero.

- aims for model with low empirical risk even if parameters are missing
- idea: no single parameter entry can become 'too important'
- similar to data augmentation, but without need for domain knowledge about x's
- overfitting vs. underfitting tradeoff controlled by p

Regularization

Often, more than one regularization techniques are combined, e.g.

Explicit regularization: e.g. *"elastic net"*

- $\Omega(\theta) = \alpha\|\theta\|_{L^2}^2 + (1-\alpha)\|\theta\|_{L^1}$

Explicit/implicit regularization: e.g. large-scale support vector machines

- $\Omega(\theta) = \|\theta\|_{L^2}^2$, early stopping, data augmentation

Implicit regularization: e.g. deep networks

- early stopping, weight decay, dropout, potentially data augmentation

Summary

Regularization can prevent overfitting

Intuition: avoid "extreme" models, e.g. very large parameter values

Explicit Regularization: modify object function

Implicit Regularization: change optimization procedure

Regularization introduces additional (hyper)parameters

How much of a regularization method to apply is a free parameter, often called *regularization constant*. The optimal values are problem-specific.

Choosing between models/methods/parameters

Predictor Training (idealized)

```
input training data $\mathcal{D}_{\text{trn}}$
input learning procedure $A$
  $g \leftarrow A[\mathcal{D}]$    (apply $A$ with $\mathcal{D}$ as training set)
output resulting predictor $g : \mathcal{X} \to \mathcal{Y}$
```

Predictor Evaluation

```
input trained predictor $g : \mathcal{X} \to \mathcal{Y}$
input test data $\mathcal{D}_{\text{tst}}$
  apply $g$ to $\mathcal{D}_{\text{tst}}$ and measure performance $\hat{\mathcal{R}}_{\text{tst}}(g)$
output performance estimate $\hat{\mathcal{R}}_{\text{tst}}(g)$
```

Predictor Training (idealized)

```
input training data $\mathcal{D}_{\text{trn}}$
input learning procedure $A$
   $g \leftarrow A[\mathcal{D}]$   (apply $A$ with $\mathcal{D}$ as training set)
output resulting predictor $g : \mathcal{X} \to \mathcal{Y}$
```

Predictor Evaluation

```
input trained predictor $g : \mathcal{X} \to \mathcal{Y}$
input test data $\mathcal{D}_{\text{tst}}$
   apply $g$ to $\mathcal{D}_{\text{tst}}$ and measure performance $\hat{\mathcal{R}}_{\text{tst}}(g)$
output performance estimate $\hat{\mathcal{R}}_{\text{tst}}(g)$
```

Remark: In commercial applications, this is realistic:

- given some training set one builds a single system,
- one deploys it to the customers,
- the customers use it on their own data, and complain if disappointed

In research, one typically has no customer, but only a fixed amount of data to work with.

If given only one dataset, $\mathcal{D}$, one *simulates* the train/test protocol.

Classifier Training and Evaluation

```
input data $\mathcal{D}$
input learning method $A$
   split $\mathcal{D} = \mathcal{D}_{\text{trn}} \dot\cup \mathcal{D}_{\text{tst}}$ disjointly
   set aside $\mathcal{D}_{\text{tst}}$ to a safe place        // do not look at it
   $g \leftarrow A[\mathcal{D}_{\text{trn}}]$                        // learn a predictor from $\mathcal{D}_{\text{trn}}$
   apply $g$ to $\mathcal{D}_{\text{tst}}$ and measure performance $\hat{\mathcal{R}}_{\text{tst}}(g)$
output performance estimate $\hat{\mathcal{R}}_{\text{tst}}(g)$
```

If given only one dataset, $\mathcal{D}$, one *simulates* the train/test protocol.

Classifier Training and Evaluation

```
input data $\mathcal{D}$
input learning method $A$
  split $\mathcal{D} = \mathcal{D}_{\text{trn}} \dot\cup \mathcal{D}_{\text{tst}}$ disjointly
  set aside $\mathcal{D}_{\text{tst}}$ to a safe place        // do not look at it
  $g \leftarrow A[\mathcal{D}_{\text{trn}}]$                        // learn a predictor from $\mathcal{D}_{\text{trn}}$
  apply $g$ to $\mathcal{D}_{\text{tst}}$ and measure performance $\hat{\mathcal{R}}_{\text{tst}}(g)$
output performance estimate $\hat{\mathcal{R}}_{\text{tst}}(g)$
```

Remark. $\mathcal{D}_{\text{tst}}$ should be as small as possible, to keep $\mathcal{D}_{\text{trn}}$ as big as possible, but large enough to be convincing.

- small datasets: 50%/50%,   larger datasets: 80%/20%, or 90%/10%

If given only one dataset, $\mathcal{D}$, one *simulates* the train/test protocol.

Classifier Training and Evaluation

```
input data $\mathcal{D}$
input learning method $A$
    split $\mathcal{D} = \mathcal{D}_{\text{trn}} \,\dot{\cup}\, \mathcal{D}_{\text{tst}}$ disjointly
    set aside $\mathcal{D}_{\text{tst}}$ to a safe place      // do not look at it
    $g \leftarrow A[\mathcal{D}_{\text{trn}}]$                  // learn a predictor from $\mathcal{D}_{\text{trn}}$
    apply $g$ to $\mathcal{D}_{\text{tst}}$ and measure performance $\hat{\mathcal{R}}_{\text{tst}}(g)$
output performance estimate $\hat{\mathcal{R}}_{\text{tst}}(g)$
```

Remark. $\mathcal{D}_{\text{tst}}$ should be as small as possible, to keep $\mathcal{D}_{\text{trn}}$ as big as possible, but large enough to be convincing.

- small datasets: 50%/50%,   larger datasets: 80%/20%, or 90%/10%

Warning: $\mathcal{D}_{\text{tst}}$ is "use once": it must not be used for any decisions in building the predictor, only to evaluate it at the very end.

Classifier Training and Evaluation

```
input data $\mathcal{D}$
input learning method $A$
  split $\mathcal{D} = \mathcal{D}_{\text{trn}} \dot\cup \mathcal{D}_{\text{tst}}$ disjointly
  set aside $\mathcal{D}_{\text{tst}}$ to a safe place        // do not look at it
  $g \leftarrow A[\mathcal{D}_{\text{trn}}]$                        // learn a predictor from $\mathcal{D}_{\text{trn}}$
  apply $g$ to $\mathcal{D}_{\text{tst}}$ and measure performance $R_{\text{tst}}$
output performance estimate $R_{\text{tst}}$
```

In practice we often want more: not just train a classifier and evaluate it, but

- select the best algorithm out of multiple ones,
- select the best (hyper)parameters for a training algorithm.

We simulate the classifier evaluation step during the training procedure. This needs (at least) one additional data split:

Training and Selecting between Multiple Models

```
input data $\mathcal{D}$
input set of method $\mathcal{A} = \{A_1, \dots, A_K\}$
  split $\mathcal{D} = \mathcal{D}_{\text{trnval}} \dot\cup \mathcal{D}_{\text{tst}}$ disjointly
  set aside $\mathcal{D}_{\text{tst}}$ to a safe place (do not look at it)

  split $\mathcal{D}_{\text{trnval}} = \mathcal{D}_{\text{trn}} \dot\cup \mathcal{D}_{\text{val}}$ disjointly
  for all models $A_i \in \mathcal{A}$ do
    $g_i \leftarrow A_i[\mathcal{D}_{\text{trn}}]$
    apply $g_i$ to $\mathcal{D}_{\text{val}}$ and measure performance $E_{\text{val}}(A_i)$
  end for
  pick best performing $A_i$

  (optional) $g_i \leftarrow A_i[\mathcal{D}_{\text{trnval}}]$    // retrain best method on larger dataset
  apply $g_i$ to $\mathcal{D}_{\text{tst}}$ and measure performance $R_{\text{tst}}$
output performance estimate $R_{\text{tst}}$
```

How to split? For example $\frac{1}{3} : \frac{1}{3} : \frac{1}{3}$ or $70\% : 10\% : 20\%$.

Discussion.

- Each algorithm is trained on $\mathcal{D}_{\text{trn}}$ and evaluated on disjoint $\mathcal{D}_{\text{val}}$ ✓
- You select a predictor based on $\mathcal{R}_{\text{val}}$ (its performance on $\mathcal{D}_{\text{val}}$), only afterwards $\mathcal{D}_{\text{tst}}$ is used. ✓
- $\mathcal{D}_{\text{tst}}$ is used to evaluate the final predictor and nothing else. ✓

Discussion.

- Each algorithm is trained on $\mathcal{D}_{\text{trn}}$ and evaluated on disjoint $\mathcal{D}_{\text{val}}$ ✓
- You select a predictor based on $\mathcal{R}_{\text{val}}$ (its performance on $\mathcal{D}_{\text{val}}$), only afterwards $\mathcal{D}_{\text{tst}}$ is used. ✓
- $\mathcal{D}_{\text{tst}}$ is used to evaluate the final predictor and nothing else. ✓

Problems.

- small $\mathcal{D}_{\text{val}}$ is bad: $\mathcal{R}_{\text{val}}$ could be bad estimate of g_A's true performance, and we might pick a suboptimal method.
- large $\mathcal{D}_{\text{val}}$ is bad: $\mathcal{D}_{\text{trn}}$ is much smaller than $\mathcal{D}_{\text{trnval}}$, so the classifier learned on $\mathcal{D}_{\text{trn}}$ might be much worse than necessary.
- retraining the best model on $\mathcal{D}_{\text{trnval}}$ might overcome that, but it comes at a risk: just because a model worked well when trained on $\mathcal{D}_{\text{trn}}$, this does not mean it'll also work well when trained on $\mathcal{D}_{\text{trnval}}$.

Leave-one-out Evaluation (for a single model/algorithm)

```
input algorithm $A$
input loss function $\ell$
input data $\mathcal{D}$        (trnval part only: test part set aside earlier)
  for all $(x^i, y^i) \in \mathcal{D}$ do
    $g^{\neg i} \leftarrow A[\ \mathcal{D} \setminus \{(x^i, y^i)\}\ ]$    // $\mathcal{D}_{\text{trn}}$ is $\mathcal{D}$ with $i$-th example removed
    $r^i \leftarrow \ell(y^i, g^{\neg i}(x^i))$              // $\mathcal{D}_{\text{val}} = \{(x^i, y^i)\}$, disjoint to $\mathcal{D}_{\text{trn}}$
  end for
output $\mathcal{R}_{\text{loo}} = \frac{1}{n}\sum_{i=1}^{n} r^i$    (average leave-one-out risk)
```

Properties.

- Each r^i is a unbiased (but high variance) estimate of the risk $\mathcal{R}(g^{\neg i})$
- $\mathcal{D} \setminus \{(x^i, y^i)\}$ is almost the same as $\mathcal{D}$, so we can hope that each $g^{\neg i} \approx g = A[\mathcal{D}]$.
- Therefore, R_{loo} can be expected a good estimate of $\mathcal{R}(g)$

Problem:

- slow, trains n times on $n-1$ examples instead of once on n
- $\mathcal{R}_{\text{loo}}$ is not the qualify of any individual, but an average over many

Compromise: use fixed number of small $\mathcal{D}_{\text{val}}$

K-fold Cross Validation (CV)

```
input algorithm $A$, loss function $\ell$, data $\mathcal{D}$ (trnval part)
    split $\mathcal{D} = \dot{\bigcup}_{k=1}^{K} \mathcal{D}_k$ into $K$ equal sized disjoint parts
    for $k = 1, \dots, K$ do
        $g^{\neg k} \leftarrow A[\,\mathcal{D} \setminus \mathcal{D}_k\,]$
        $r^k \leftarrow \frac{1}{|\mathcal{D}_k|} \sum_{(x,y)\in\mathcal{D}_k} \ell(y^i, g^{\neg k}(x))$
    end for
output $R_{K\text{-CV}} = \frac{1}{K} \sum_{k=1}^{n} r^k$    ($K$-fold cross-validation risk)
```

Observation.

- for $K = |\mathcal{D}|$ same as leave-one-out error.
- approximately k times increase in runtime.
- most common: $k = 10$ or $k = 5$.

Problem: training sets overlap, so the error estimates are correlated.
Exception: $K = 2$

5×2 Cross Validation (5×2-CV)

```
input algorithm $A$, loss function $\ell$, data $\mathcal{D}$ (trnval part)
  for $k = 1, \dots, 5$ do
    Split $\mathcal{D} = \mathcal{D}_1 \,\dot{\cup}\, \mathcal{D}_2$
    $g_1 \leftarrow A[\mathcal{D}_1]$,
    $r_1^k \leftarrow$ evaluate $g_1$ on $\mathcal{D}_2$
    $g_2 \leftarrow A[\mathcal{D}_2]$,
    $r_2^k \leftarrow$ evaluate $g_2$ on $\mathcal{D}_1$
    $r^k \leftarrow \frac{1}{2}(r_k^1 + r_k^2)$
  end for
output $\mathcal{R}_{5\times 2} = \frac{1}{5}\sum_{k=1}^{5} r^k$
```

Observation.

- 5×2-CV is really the average of 5 runs of 2-fold CV
- very easy to implement: shuffle the data and split into halfs
- within each run the training sets are disjoint and the classifiers g_1 and g_2 are independent

Problem: training sets are smaller than in 5- or 10-fold CV.

Other methods of evaluation

Classification with Imbalanced Classes

If classes are imbalanced accuracy might not tell us much:

- $p(y=-1) = 0.99$, $p(y=+1) = 0.01$ → "always no" is 99% correct
- there might not be any better non-constant classifier

Three "solutions":

1. balancing the dataset
   - use only subset of the majority class to balance data (5:1, or 1:1)
2. reweighting
   - multiple loss in optimization with class-dependent constant C_{y_i},

$$\frac{1}{|\mathcal{D}_+|}\sum_{(x_i,y_i)\in\mathcal{D}_+}^{n} \ell(\,y_i, f(x_i)\,) + \frac{1}{|\mathcal{D}_-|}\sum_{(x_i,y_i)\in\mathcal{D}_-}^{n} \ell(\,y_i, f(x_i)\,) + \Omega(f)$$

3. treat as a retrieval problem instead of classification

Classifiers for Information Retrieval Tasks

Some classification tasks are really rather retrieval tasks, e.g.

- database lookup: is an entry x relevant $(y = 1)$ or not $(y = -1)$?

A typical property:

- prediction is performed on a fixed database
- we have access to all elements of the test set at the same time (not one by one)
- positives $(y = 1)$ are important, negative $(y = -1)$ are a nuisanse
- we don't need all decisions, getting a few correct positives are enough

Classifiers for Information Retrieval Tasks

Some classification tasks are really rather retrieval tasks, e.g.

- database lookup: is an entry x relevant $(y = 1)$ or not $(y = -1)$?

A typical property:

- prediction is performed on a fixed database
- we have access to all elements of the test set at the same time (not one by one)
- positives $(y = 1)$ are important, negative $(y = -1)$ are a nuisanse
- we don't need all decisions, getting a few correct positives are enough

Procedure:

- for a classifier $g(x) = \operatorname{sign} f(x)$ with $f(x) : \mathcal{X} \to \mathbb{R}$ (e.g., $f(x) = \langle w, x \rangle$), interpret $f(x) \in \mathbb{R}$ as its confidence.
- To produce K positives we return the test samples of highest confidence.
- Alternatively, decide by $g_\theta(x) = \operatorname{sign}(\, f(x) - \theta \,)$, for suitably chosen cutoff threshold θ.

Evaluating Retrieval Systems

Retrieval quality is often measure in terms of **precision** and **recall**:

Definition (Precision, Recall, F-Score)

For $\mathcal{Y} = \{\pm 1\}$, let $g : \mathcal{X} \to \mathcal{Y}$ a decision function and $\mathcal{D} = \{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$ be a *database*.

Then we define

$$\textit{precision}(g) = \frac{\textit{number of test samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of test samples with } g(x^j) = 1}$$

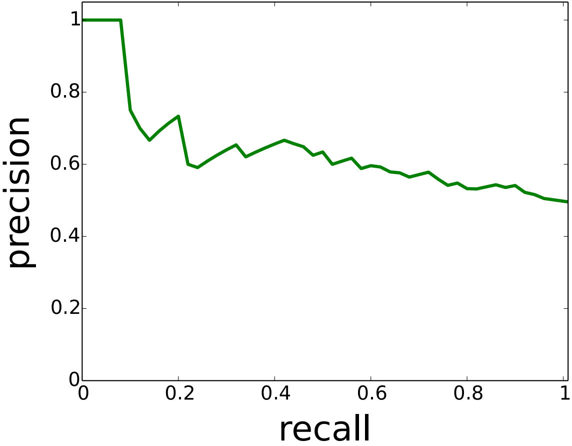
$$\textit{recall}(g) = \frac{\textit{number of test samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of test samples with } y^j = 1}$$

$$F1\textit{-score}(g) = 2\frac{\textit{precision}(g) \cdot \textit{recall}(g)}{\textit{precision}(g) + \textit{recall}(g)}$$

Evaluating Retrieval Systems

For different cutoff thresholds, θ, we obtain different precision and recall values.

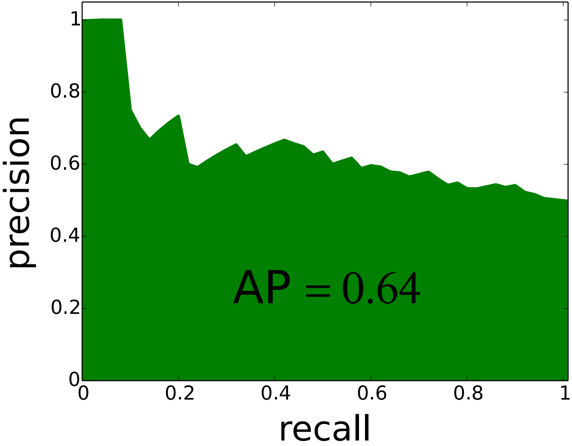
They are summarized by a **precision-recall curve**:



Evaluating Retrieval Systems

For different cutoff thresholds, θ, we obtain different precision and recall values.

They are summarized by a **precision-recall curve**:



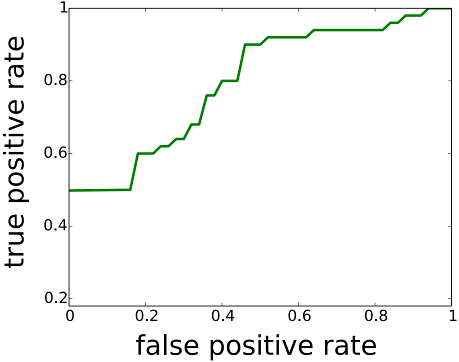
- If pressured, summarize into one number: **average precision**.
- Curve/value depends on class ratio: higher values for more positives

A similar role in different context:

Receiver Operating Characteristic (ROC) Curve

true-positive-rate $$\textit{TPR}(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of samples with } y^j = 1}$$

false-positive-rate $$\textit{FPR}(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = -1}{\textit{number of samples with } y^j = -1}$$

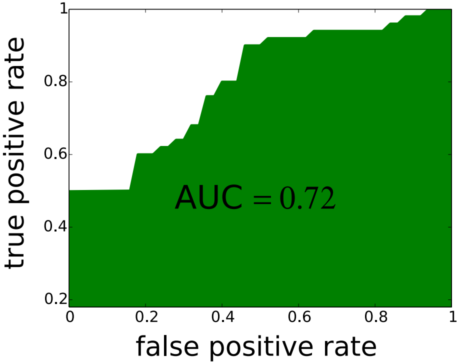


A similar role in different context:

Receiver Operating Characteristic (ROC) Curve

true-positive-rate $$TPR(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of samples with } y^j = 1}$$

false-positive-rate $$FPR(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = -1}{\textit{number of samples with } y^j = -1}$$

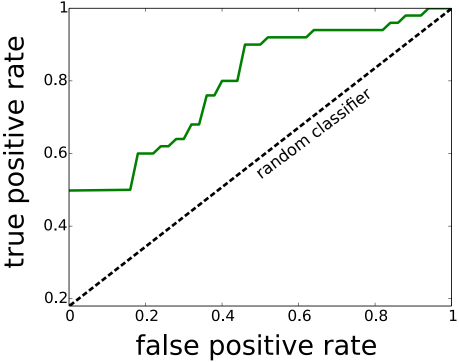


A similar role in different context:

Receiver Operating Characteristic (ROC) Curve

true-positive-rate $$TPR(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of samples with } y^j = 1}$$

false-positive-rate $$FPR(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = -1}{\textit{number of samples with } y^j = -1}$$


true positive rate
false positive rate
random classifier