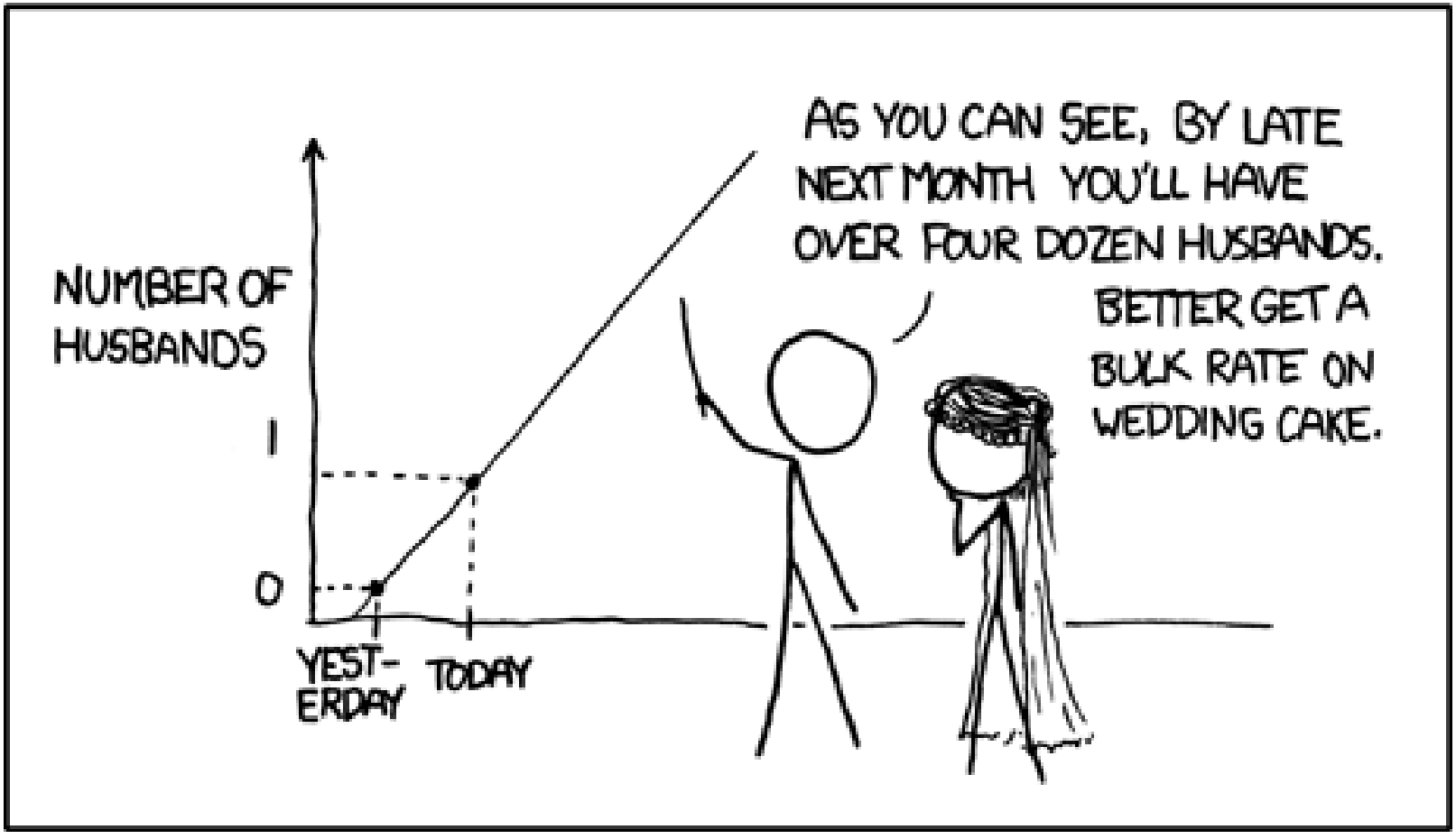
Advanced course in machine learning
582744
Lecture 2

Arto Klami

MY HOBBY: EXTRAPOLATING



Outline

Machine learning process

- A *model* M describes data and typically has some unknown parameters $\boldsymbol{\theta}$
- A *data set* D is some collection of observations we want to model, often $D = \{\mathbf{x}_n, y_n\}_{n=1}^N$ or $D = \{\mathbf{x}_n\}_{n=1}^N$ ($\mathbf{x}$ is input, y is output)
- *Learning* or *model fitting* means choosing the parameters $\boldsymbol{\theta}$ based on the data D
- This is fundamentally an optimization problem: Some loss function $L(D, M(\boldsymbol{\theta}))$ needs to be minimized or maximized over the parameters $\boldsymbol{\theta}$
- Given the solution $\hat{\boldsymbol{\theta}}$ we can make predictions with the model: $p(\tilde{\mathbf{x}}|M, \hat{\boldsymbol{\theta}})$ or $p(\tilde{y}|\tilde{\mathbf{x}}, M, \hat{\boldsymbol{\theta}})$

Machine learning process: example

- Model: $p(y|\mathbf{x}, \boldsymbol{\theta}) = N(\boldsymbol{\theta}^T \mathbf{x}, 1)$
- Data set: $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ...\}$
- Loss function: $L(D, \boldsymbol{\theta}) = \frac{1}{N} \sum_n \|y_n - \boldsymbol{\theta}^T \mathbf{x}_i\|^2$
- Fit: $\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}\mathbf{y}$
- Prediction: $p(y|\tilde{\mathbf{x}}, \hat{\boldsymbol{\theta}}) = N(\hat{\boldsymbol{\theta}}^T \tilde{\mathbf{x}}, 1)$

Loss functions

The loss function $L(D, M(\theta))$

- Depends on the data D and the model/parameters θ
- We can always think of minimizing it; if the problem looks like maximization just put minus sign in front of it
- The loss function defines the goal of the learning task
- Examples:
  $L(y, f(x, \theta)) = \|y - f(x, \theta)\|^2$
  $L(y, f(x, \theta)) = |y - f(x, \theta)| \quad y \in [0, 1]$
  $L(y, f(x, \theta)) = \log p(y|x, \theta)$
  $L(x, f(\theta)) = \log p(x|\theta)$

Minimizing the loss is called *optimization*, and there is a whole field of science that studies this

The most important slide of the course

What are we really interested in?

- Training error: The loss for a given data set D
- Generalization error: The loss for a future data set $\tilde{D}$

Machine learning is about optimizing the generalization error!
...but while fitting the model we only have the given data

If not for this discrepancy, the field of optimization would have already solved most of the problems

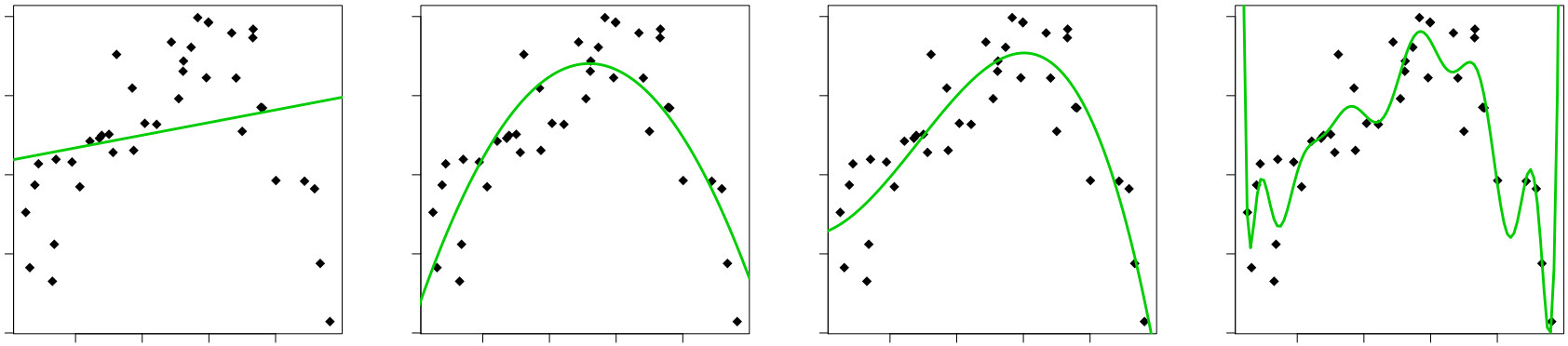
Dreaming of becoming a manager? Always ask "...but how well does it generalize?"

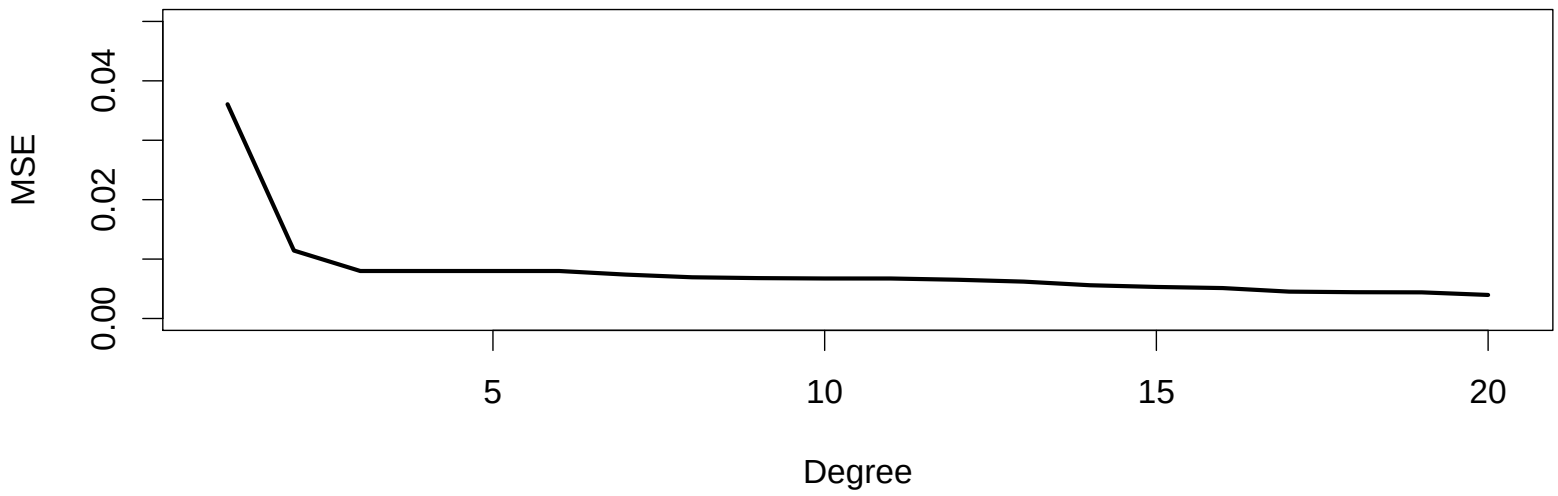
Why are they different?

- The data D is a sample from an uncertain process (that is, it is a random variable)
- Another sample $\tilde{D}$ from the same process would require different parameters $\boldsymbol{\theta}$ to minimize the loss
- Hence: A finite data sample D does not uniquely determine the optimal solution
- Instead, we need to find parameters that are good for the underlying distribution $p(D)$ that generated the data

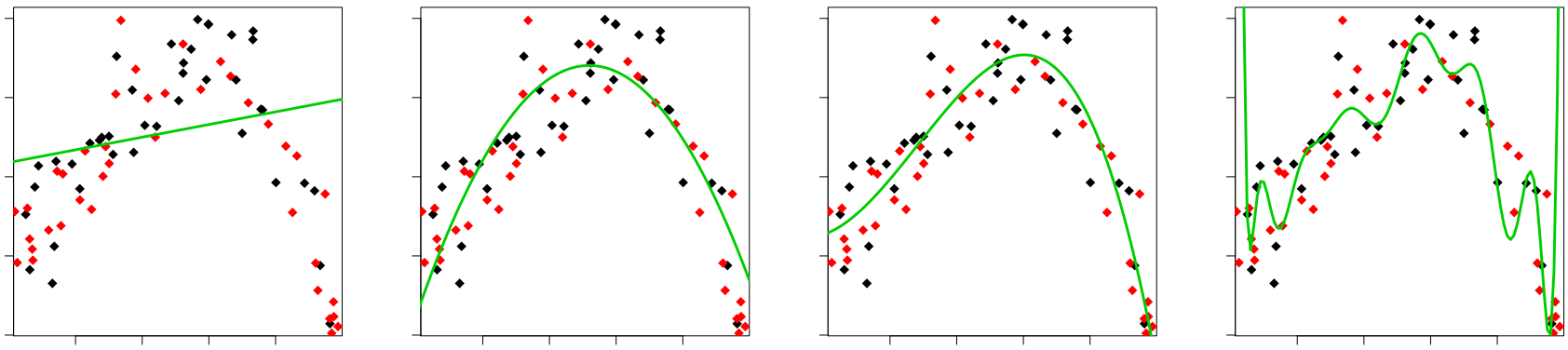
If D is not uncertain then you probably do not need ML

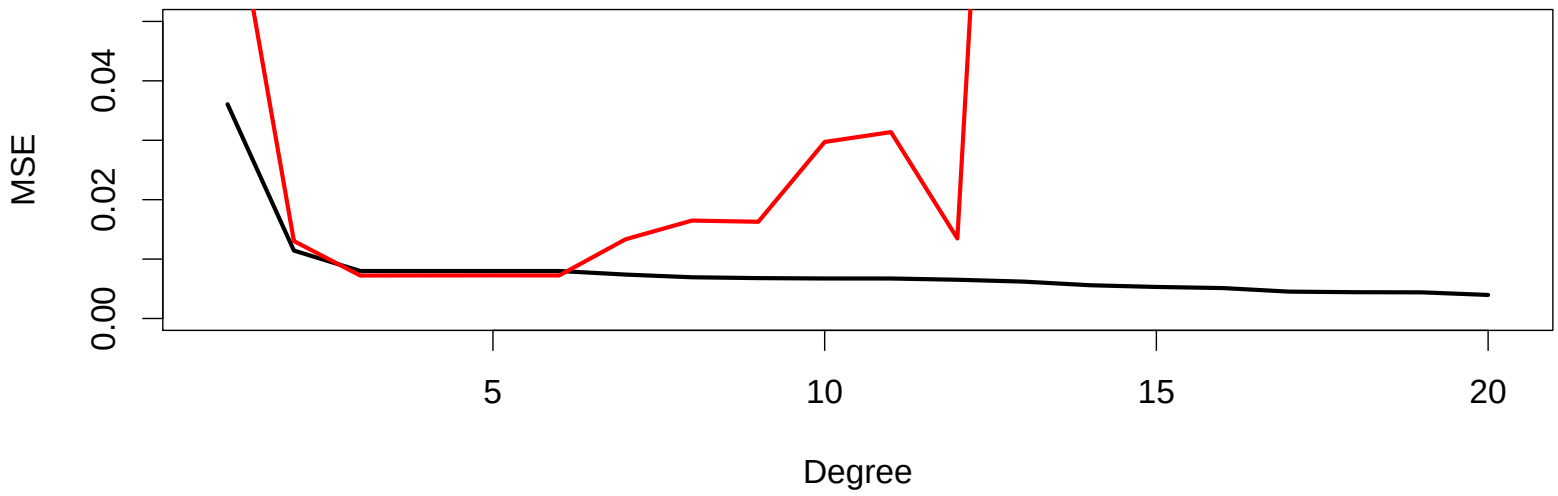
Illustration



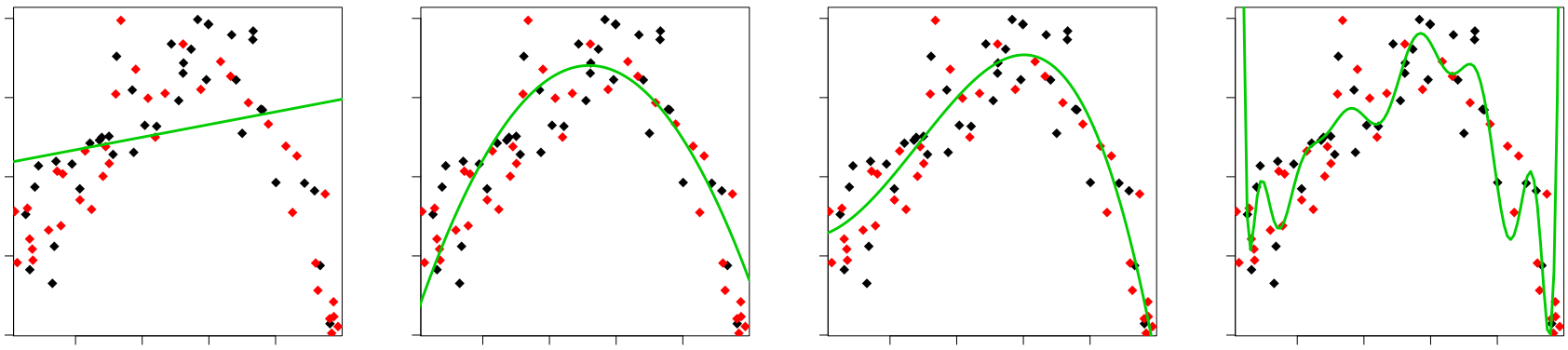


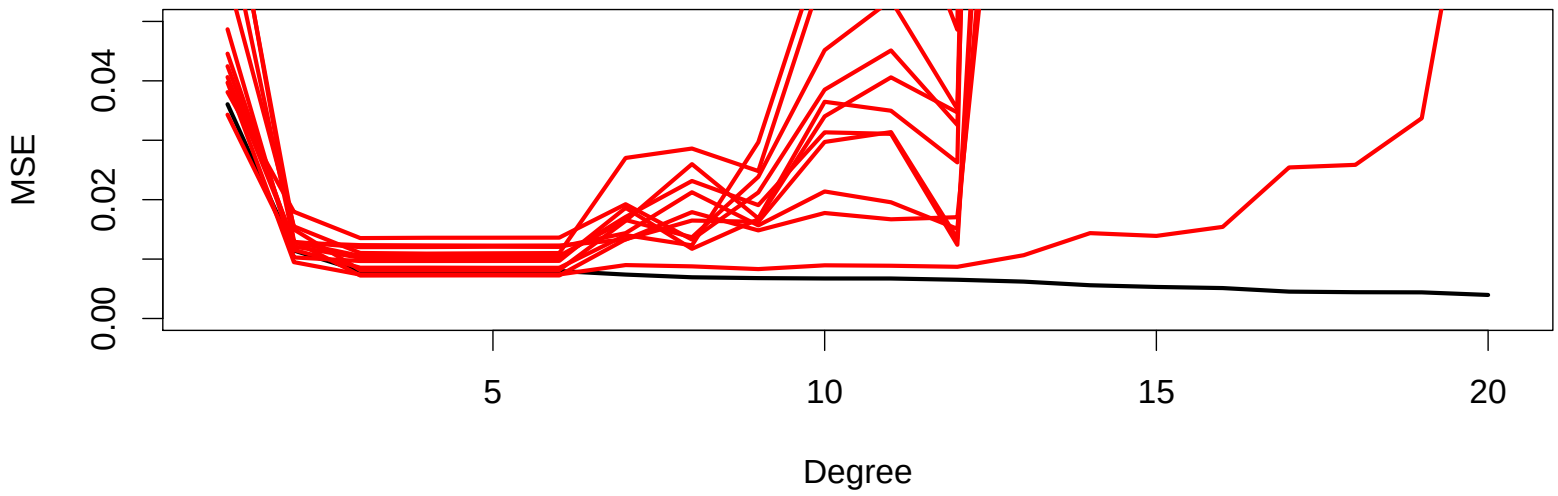
Illustration





Illustration





Approaching the generalization error

- Formal definitions
- How to estimate it? Validation
- How to prevent it? Regularization and averaging

Risk: the expected loss

For simplicity of notation, let us consider classification problems where we output some y for each $\mathbf{x}$, and denote by $\delta(D)$ the classifier that makes the decisions

The *risk* is defined as the expected loss over the distribution generating the data points:

$$R(\delta) = E_{p(y,\mathbf{x})}[L(y, \delta(\mathbf{x}))] = \int_{\mathbf{x},y} L(y, \delta(\mathbf{x}))p(y, \mathbf{x})d\mathbf{x}dy$$

Simple, right? But we do not know $p(y, \mathbf{x})$...

Risk formulations

$$R(\delta) = E_{p(y,\mathbf{x})}[L(y, \delta(\mathbf{x}))] = \int_{\mathbf{x},y} L(y, \delta(\mathbf{x}))p(y, \mathbf{x})d\mathbf{x}dy$$

Statistical learning theory:
Treat the data generating process as truly unknown, and try to somehow bound the risk, for example by considering the worst-case scenario

Bayesian approach:
Assume we know the correct model but are just unsure of the parameters. Then we can average over the parameters conditional on the data:
$$R(\delta) = E_{p(\boldsymbol{\theta}|D)}[L(y, \delta(\mathbf{x}))] = \int_{\boldsymbol{\theta}} L(y, \delta(\mathbf{x}))p(y, \mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}|D)d\boldsymbol{\theta}$$

The book is largely based on the latter, but does not very clearly state its limitation: The expectation is wrong if the model is wrong

Empirical risk minimization

$$R(\delta) = E_{p(y,\mathbf{x})}[L(y, \delta(\mathbf{x}))] = \int_{\mathbf{x},y} L(y, \delta(\mathbf{x}))p(y, \mathbf{x})d\mathbf{x}dy$$

The training error is called "empirical risk" and simply plugs in the observations; $1/N$ weight for each of the N training samples: $R_e \approx \frac{1}{N}\sum_n L(y_n, \delta(\mathbf{x}_n))$

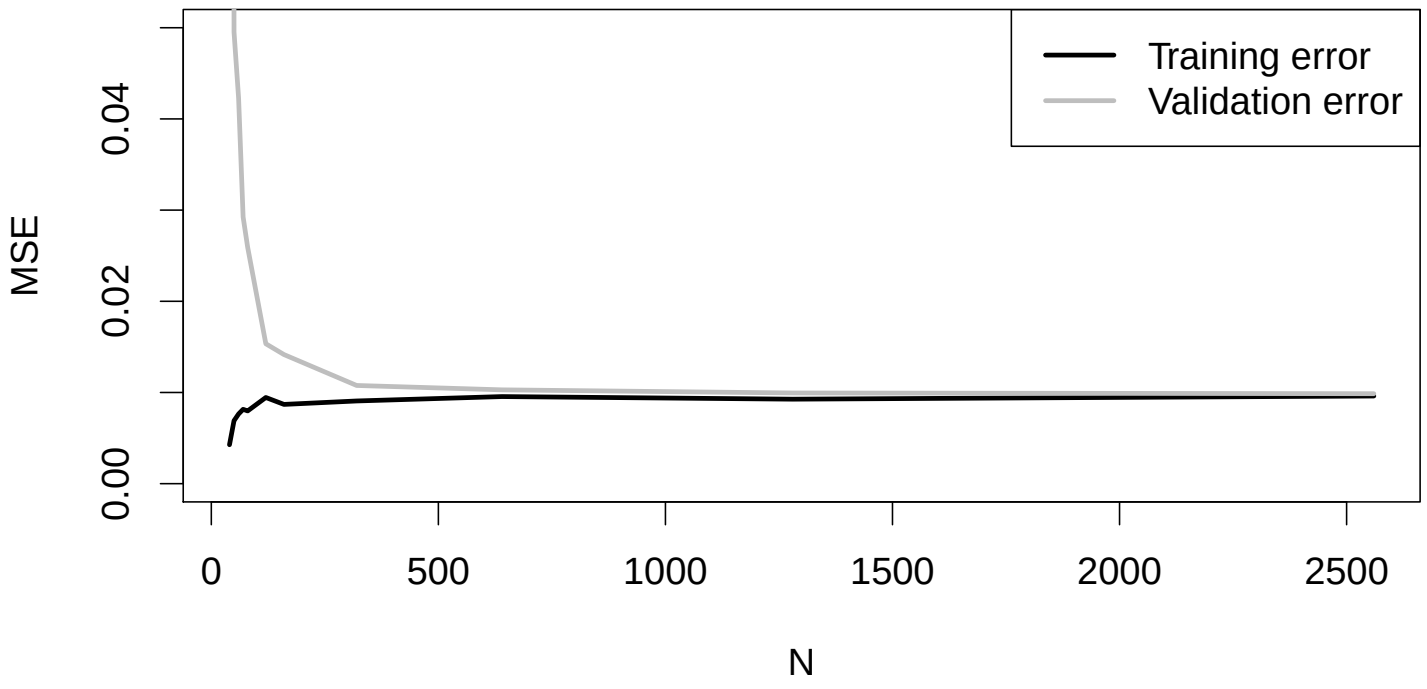
A *consistent estimator* gives the right solution by minimizing the empirical risk when $N \to \infty$

Estimating the risk

Instead of using the empirical error on the training data, we can get an unbiased (but typically high-variance) estimate on the expected risk by looking at the error on data samples not used for training

$R_v \approx \frac{1}{N} \sum_n L(y_n, \delta(\mathbf{x}_n))$ for $\{(x_n, y_n)\}$ not used in training

Consistency



Cross-validation error

A practical way of getting a lower variance estimate

- Split the data randomly into K folds
- Estimate θ based on samples in $K-1$ folds
- Evaluate the error on the remaining one
- Estimate the risk as $\frac{1}{K} R_{v(k)}$, where $R_{v(k)}$ is the empirical risk for the kth fold

Leave-one-out validation: $K = N$, always leaving out one sample at a time

Risk minimization

Now we can recognize models with high risk, but how do we avoid it?
Three standard ways:

- Early-stopping: Keep on optimizing only as long as the estimated risk goes down
- Regularize the loss function: Modified optimization problem that is less likely to overfit
- Average predictions over multiple models

The first covered in exercises

Regularized risk minimization

Idea: Limiting the complexity should help avoiding overfitting

Optimize $R(\delta) + \lambda C(\delta)$ instead, where $C(\delta)$ somehow measures the complexity

We also need to set λ to control the amount of regularization. This is typically done by cross-validation; we estimate the risk for each possible choice of λ and pick the best

Statistical learning theory

(One) formal justification for regularization comes from *statistical learning theory*: Prove an upper bound for the risk over all possible data generating distributions $p(D)$

For binary classification, Hoeffding's inequality states that
$p(\max_\delta |R_e(\delta) - R(\delta)| > \epsilon) \leq 2|\mathcal{H}|e^{-2N\epsilon^2}$
where $|\mathcal{H}|$ is the size of the hypothesis space

Statistical learning theory

What is $|\mathcal{H}|$?

- The number of possible models in a finite set
- ...but usually we have infinitely many
- Vapnik-Chevronenkis (VC) dimension: The maximum number of points that can be arranged so that we make no mistakes in binary classification
- *Probably appxorimately correct* (PAC) if we can find a function that has low empirical risk and the hypothesis space is small
- In practice: Computing VC dimension is hard and the bound can still be loose

Just remember the intuition: If the hypothesis space is small, the empirical risk is probably quite close to the true risk. If not, we probably underestimated the risk.

Typical regularizers

Most regularizers control the norm of the parameters:

- ℓ_2: The squared norm
- ℓ_1: The sum of the absolute elements
- ℓ_∞: The largest element

...while some attempt to directly measure the complexity:

- Bayesian/Akaike information criterion: degrees of freedom
- Minimum description length (MDL): See *Information theoretic modeling*

Bootstrap

Another alternative is to create multiple data sets D_b, estimate the parameters for each, and then average the predictions over those: $\frac{1}{B}|y - p(y|\mathbf{x}, \boldsymbol{\theta}_b)|$

Ideally we would draw the fake data sets D_b from the true unknown process, but in practice we re-sample from the current set: Pick N samples with replacement

Probabilistic modeling

A *generative model* is a model that is written as a collection of probability distributions, describing a process generating the data

$$
\begin{aligned}
p(y|\mathbf{x}, \boldsymbol{\theta}) &= N(\boldsymbol{\theta}^T \mathbf{x}, \sigma^2) \\
p(\mathbf{x}) &= \text{Uniform}(-1, 1) \\
p(\boldsymbol{\theta}) &= N(0, \mathbf{I}) \\
p(\sigma^2) &= \text{Ga}(0.1, 0.1)
\end{aligned}
$$

Characterized by the joint probability
$p(y, \mathbf{x}, \boldsymbol{\theta}, \sigma^2) = \prod_n \left[p(y|\mathbf{x}, \boldsymbol{\theta})p(\mathbf{x})\right] p(\boldsymbol{\theta})p(\sigma^2)$

Probabilistic modeling

Now consider the logarithmic loss $L = \log p(\cdot)$, which factorizes into $\log p(y|\mathbf{x}, \boldsymbol{\theta}) + + \log p(\mathbf{x}) + \log p(\boldsymbol{\theta}) + \log p(\sigma^2)$

- $p(y|\mathbf{x}, \boldsymbol{\theta})p(\mathbf{x})$ is the *likelihood*, quantifying how well the model fits the data
- $p(\boldsymbol{\theta})p(\sigma^2)$ is the *prior distribution*, which controls the complexity (in some way)

The book presents most methods from this perspective, even though some of them were not originally developed as probabilistic models

Probabilistic terminology

- Empirical risk minimization is called *maximum likelihood* (ML) estimation since we only care about the likelihood part
- If we include also the prior, corresponding to regularized risk minimization, then we are searching for *maximum a posteriori* (MAP) estimate

- *Evidence / marginal likelihood*:
  $p(y, \mathbf{x}) = \int_{\boldsymbol{\theta}, \sigma^2} p(y|\boldsymbol{\theta}, \sigma^2, \mathbf{x}) p(\mathbf{x}) p(\boldsymbol{\theta}, \sigma^2) d\boldsymbol{\theta} d\sigma^2$
- *Posterior distribution*: $p(\boldsymbol{\theta}, \sigma^2|\mathbf{x}, y) = \frac{p(y|\boldsymbol{\theta}, \sigma^2, \mathbf{x}) p(\mathbf{x}) p(\boldsymbol{\theta}, \sigma^2)}{p(y, \mathbf{x})}$
  summarizes everything we know about the parameters

Expected risk for Bayesian inference

The risk was defined as
$R(\delta) = E_{p(y,\mathbf{x})}[L(y, \delta(\mathbf{x}))] = \int_{\mathbf{x},y} L(y, \delta(\mathbf{x}))p(y, \mathbf{x})d\mathbf{x}dy,$
but the problem was we did not know the data distribution

...but now we do, since we constructed a model for it. We just do not know the parameter values and hence need to average over them
$R(\delta) = E_{p(\boldsymbol{\theta}|D)}[L(y, \delta(\mathbf{x}))] = \int_{\boldsymbol{\theta}} L(y, \delta(\mathbf{x}))p(y, \mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}|D)d\boldsymbol{\theta}$

We now longer have the problem of unknown data generating distribution, but have replaced it with the assumption that we know the correct model!

On Bayesian inference vs optimization

Bayesian inference is about averaging predictions over the posterior distribution $p(\boldsymbol{\theta}|D)$

- Finding $p(\boldsymbol{\theta}|D)$ is actually not an optimization problem; it is given directly by the Bayes' rule and all we need is algebraic manipulation and integration
- It is typically not tractable, so in practice we still end up with a learning problem
- Markov chain Monte Carlo: Design a stochastic process that draws samples from $p(\boldsymbol{\theta}|D)$
- Variational inference: Find an approximation $q(\boldsymbol{\theta}|\boldsymbol{\psi}) \approx p(\boldsymbol{\theta}|D)$, optimize $\boldsymbol{\psi}$ to minimize the distance – this is again an optimization problem!

We might return to these near the end of the course if time permits

Unified view

- Empirical risk minimization $\approx$ maximum likelihood estimation
- Regularized risk minimization $\approx$ maximum a posteriori estimation
- (Bootstrap $\approx$ full Bayesian inference)
- (Leave-one-out cross-validation $\approx$ marginal likelihood)

Most (all?) regularized loss functions correspond to MAP estimation for some probabilistic model

ML should be about full posterior inference, but in practice we often do RRM/MAP, trusting that we regularize well enough

Optimization

ML is about solving the optimization problem $L(D, \boldsymbol{\theta}) + R(\boldsymbol{\theta})$
...so we need some optimization tools

Two kinds of problems: Convex and non-convex
...or "easy" and hard

Constrained or un-constrained; we mostly consider the former on this course

Convexity

- A convex function is a happy function
- Multiple definitions: secants are above the curve, the second derivatives are non-negative
- Strongly convex has positive second derivatives (above some ϵ)
- If $f(\cdot)$ and $g(\cdot)$ are convex then $af(\cdot) + bg(\cdot)$ is convex if $a, b \geq 0$
- ...and $\max f(\cdot) + g(\cdot)$ is convex
- ...and $g(f(\cdot))$ is convex if $g(\cdot)$ is also non-decreasing (think of $\exp(\cdot)$)
- Smooth (differentiable) vs non-smooth: Things are always easier for the former

Convex losses in ML

- Least squares, many regularizers
- Convexified losses: Hinge loss instead of binary classification error
- Every smooth function in a local neighborhood; think of Taylor expansion
- Parts of more complex cost functions: often the cost is convex with respect to some parameters if the others are kept constant
- Very often in the form $\sum_n f(\mathbf{x}_n|\boldsymbol{\theta}) + g(\boldsymbol{\theta})$, where $f(\cdot)$ and $g(\cdot)$ are convex

Convex optimization

Convex optimization problems are kind of easy: We can follow the gradients to reach the minimum

- Simple gradient-descent algorithm is enough, but needs to be implemented properly
- Convex optimization studies the convergence rates (and other theoretical properties) of different kinds of algorithms
- The ML community can cherry-pick the most robust techniques

Gradients in optimization

$$\nabla L(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial L(\boldsymbol{\theta})}{\partial \theta_1} \\ \vdots \\ \frac{\partial L(\boldsymbol{\theta})}{\partial \theta_k} \end{bmatrix}$$

- High-school math: The gradient is zero at the optimum
- Already solves for example least-squares regression, but for most problems we cannot find the optimum analytically
- Instead, we need iterative algorithms

Gradient descent

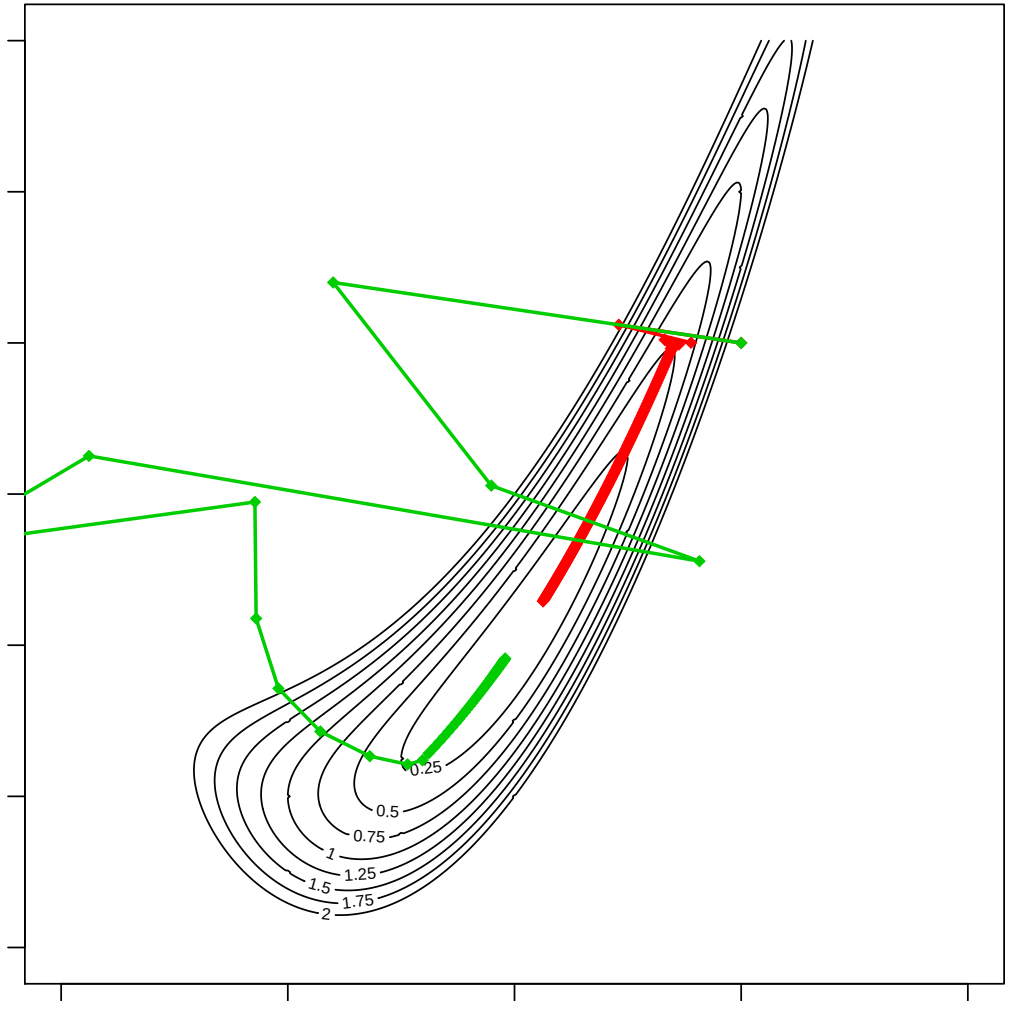
- Gradient descent: Iteratively replace the solution with one that is a bit towards (against) the gradient direction: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla L(D, \boldsymbol{\theta})$
- For convex cost functions guaranteed to find the optimal solution
- The critical detail: Setting the step-size α

Gradient descent: step-size

How do we set the step size?

- What happens when we get it wrong? Example
- Small step size always works, but is often ridiculously slow

GD with fixed step-size



Red: small step-size, green: large step-size

Gradient descent: Step size

- Fixed value: Small is slow, large overshoots
- Line search: Accurate but slow, often not worth it
- Backtracking: Try with large value, divide by two if the cost did not decrease (enough) $f(\boldsymbol{\theta} + \alpha \nabla f) \leq f(\boldsymbol{\theta}) - \beta\alpha\nabla f$
- Adaptive: Modify the previous length by some rules, typically increasing it if the cost dropped and reverting back to some small value if we overshot

Gradient descent: stochastic

For cost functions of the form
$L = \frac{1}{N}\sum_n f(\mathbf{x}_n|\boldsymbol{\theta}) + g(\boldsymbol{\theta}) = \mathbb{E}[f(\mathbf{x}|\boldsymbol{\theta})] + g(\boldsymbol{\theta})$

- we can change the order of expectation and differentiation to get $\nabla\mathbb{E}[f(\cdot)] = \mathbb{E}[\nabla f(\cdot)]$
- ...and can estimate the expectation based on a subset of the data points (or just one of them) – this speeds things up a lot if N is large
- Results in *stochastic gradient descent* where we simply use gradients computed based on that subset
- *Robbins-Monro* conditions for the step size:
  $\sum_{t=1}^{\infty}\alpha_t = \infty$ and $\sum_{t=1}^{\infty}\alpha_t^2 \leq \infty$

Gradient descent: Step-size for SGD

For example $\alpha_t = (\alpha_0 + t)^{-\beta}$ satisfies the conditions for $\beta \in (0.5, 1]$ satisfies the R-M conditions

In practice you should use some adative rate instead

- Adagrad: Per-parameter step-size scaled by $1/\sqrt{\sum_t g_t^2}$
- Adam, Adadelta and RMSProp are other similar techniques; it is enough for you to know one of these

SGD with Adagrad or some alternative solves most of the "big data" problems – the complexity does not depend on the amount of data

Can we do better?

Gradient is an intuitive direction because it points towards the steepest descent. However, it need not be optimal

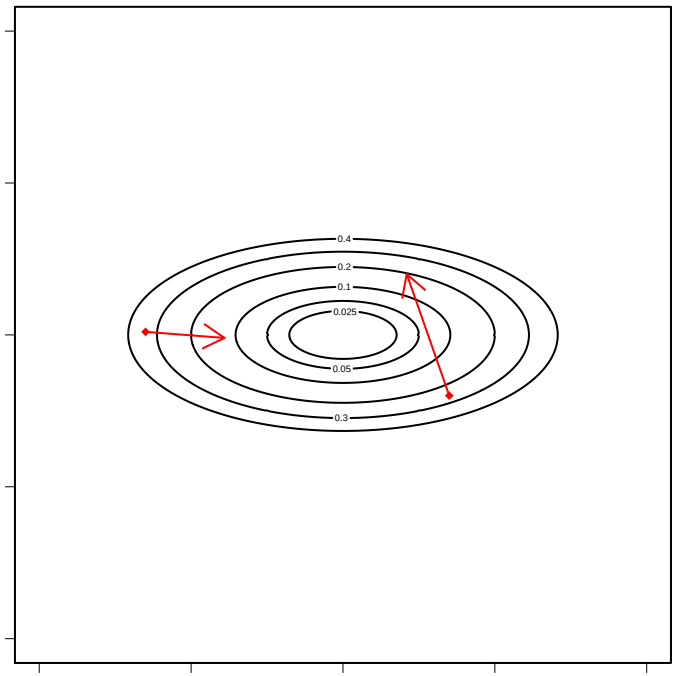
Instead of gradients, we can go towards any direction $\mathbf{d}$ that decreases the cost ($\mathbf{d}^T \nabla L < 0$). Often we can find better directions than the gradient

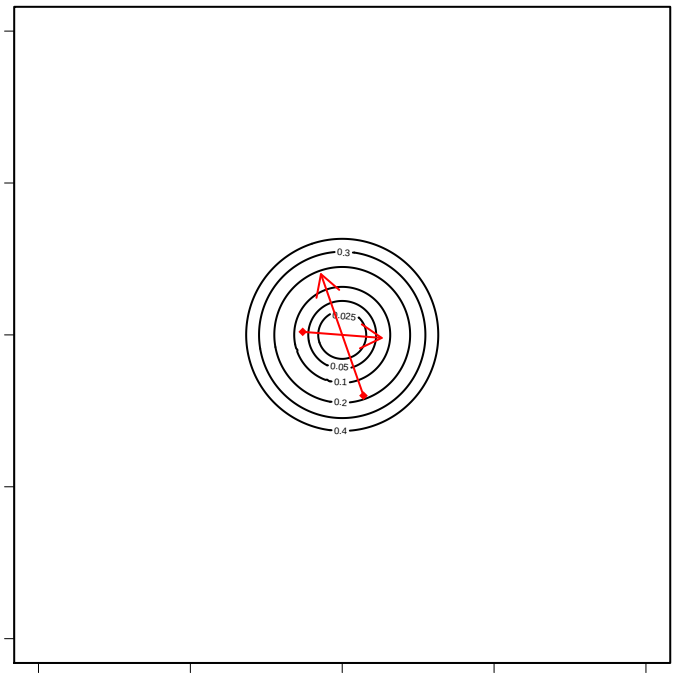
Coordinate descent

Pick one dimension at a time and perform one-dimensional optimization in that direction
$\theta_i = \arg\min f(\boldsymbol{\theta} + \alpha \mathbf{e}_i)$

Can be useful with non-smooth functions, and is in general surprisingly efficient

Geometry example



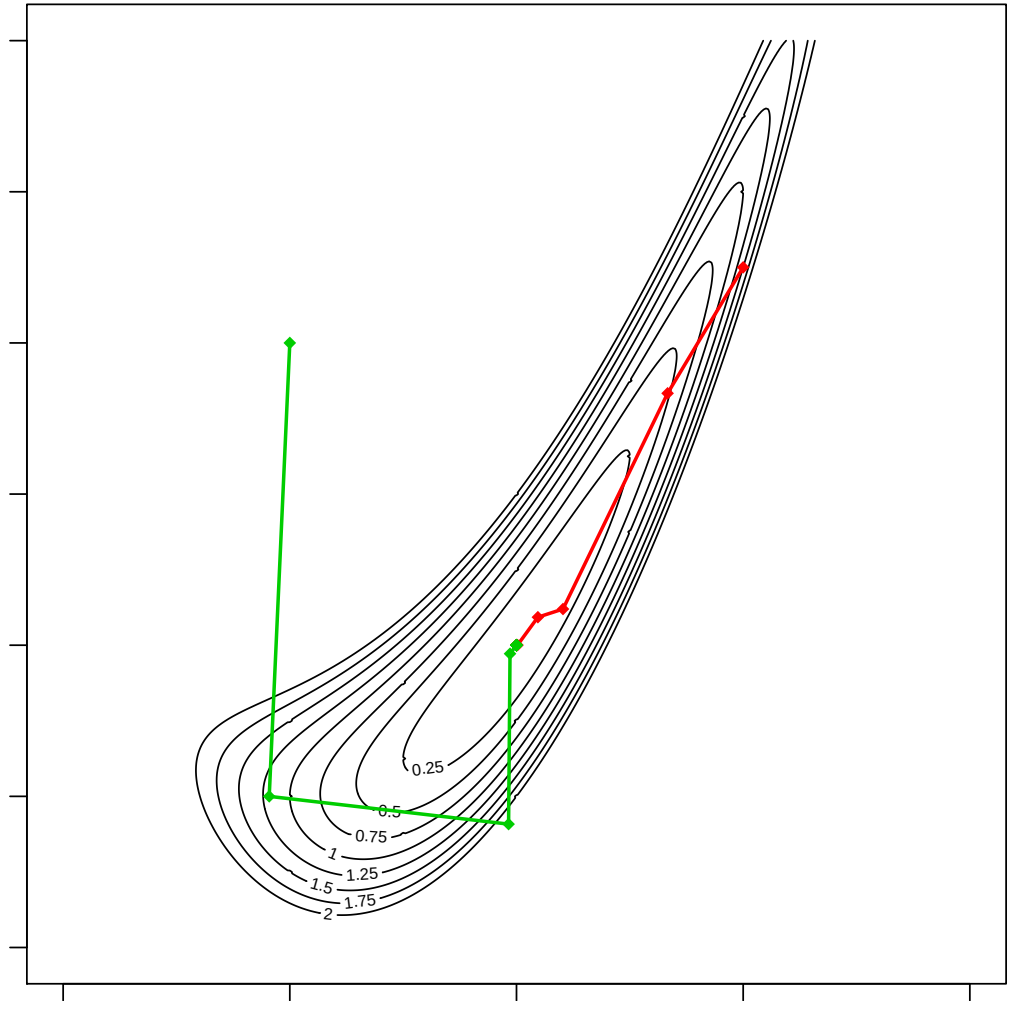


Gradient descent: second-order

Newton's method: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{H}(\boldsymbol{\theta})^{-1}\nabla f(\boldsymbol{\theta})$, where $H_{i,j} = \frac{\partial^2 H}{\partial \theta_i \partial \theta_j}$

- Fast but requires quite a bit of computation; we need the second derivatives and we have to invert the Hessian
- Can also be fraqile
- Does the geometric transformation locally
- Quasi-Newton methods: BFGS etc approximate the inverse of the Hessian based on the gradients, requiring less computation and memory – standard optimization libraries usually use these by default

Newton's method



...with two starting points; both converge quickly

Conjugate gradients

- Conjugate gradient algorithms modify the gradient direction based on the previous gradients
- Optimal for quadratic functions $\boldsymbol{\theta}^T \mathbf{A} \boldsymbol{\theta}$, converge in D steps
- Requires (sufficiently) exact line-search

Non-convex optimization

- The problem: We have multiple local optima and might miss the global optimal solution
- The cheap way: Just use convex optimization techniques and hope for the best
- ...and perhaps buy a few more lottery tickets by trying again with random initializations
- Momentum in gradient descent, simulated annealing, genetic algorithms, convexified loss functions, ...

On this course, we are happy with convex optimization techniques, but will frequently use them only for subproblems

Constrained optimization

- A big field in itself
- We will only need one simple trick: Projecting back to the constrained set
- Peform some gradient-based update, then project the solution back to set feasible set

ML definitions

- "Field of study that gives computers the ability to learn without being explicitly programmed" (A. Samuel, 1959)
- "...a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty" (K. Murphy)
- ML is optimization for loss functions that are expectations over unknown data generating processes (A. Klami)