



Supervised Learning

All slides

October 17, 2024

INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

Boosting

Gaussian Processes

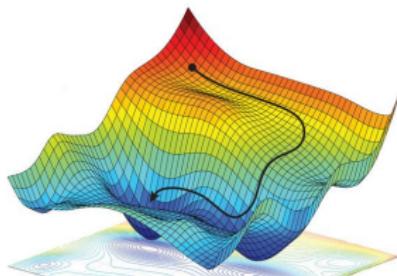


Introduction to Machine Learning

Risk Minimizers



Learning goals



- Know the concepts of the Bayes optimal model (also: risk minimizer, population minimizer)
- Bayes risk
- Consistent learners
- Bayes regret, estimation and approximation error
- Optimal constant model

EMPIRICAL RISK MINIMIZATION

Very often, in ML, we minimize the empirical risk

$$\mathcal{R}_{\text{emp}}(f) = \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)}))$$

where

- each observation $(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{X} \times \mathcal{Y}$ and \mathcal{X}, \mathcal{Y} are the feature and target spaces, respectively,
- $f : \mathcal{X} \rightarrow \mathbb{R}^g, f \in \mathcal{H}$ is a model which maps a feature vector to a numerically encoded element of \mathcal{Y} and \mathcal{H} is the hypothesis space,
- $L : (\mathcal{Y} \times \mathbb{R}^g) \rightarrow \mathbb{R}$ is the loss function which measures the dissimilarity of the model prediction and the true target,
- and we assume that $(\mathbf{x}^{(i)}, y^{(i)}) \stackrel{\text{i.i.d.}}{\sim} \mathbb{P}_{xy}$ where \mathbb{P}_{xy} is the distribution of the data generating process (DGP).

What is the theoretical justification for this procedure?



RISK MINIMIZER

Our goal is to minimize the risk

$$\mathcal{R}_L(f) := \mathbb{E}_{xy}[L(y, f(\mathbf{x}))] = \int L(y, f(\mathbf{x})) d\mathbb{P}_{xy}.$$

for a certain hypothesis $f(\mathbf{x}) \in \mathcal{H}$ and a loss $L(y, f(\mathbf{x}))$.

NB: As \mathcal{R}_L depends on loss L , we sometimes make this explicit with a subscript if needed and omit it in other cases.

Let us assume we are in an “ideal world”:

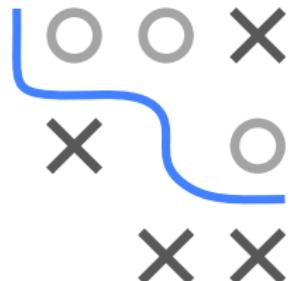
- The hypothesis space \mathcal{H} is unrestricted. We can choose any $f : \mathcal{X} \rightarrow \mathbb{R}^g$.
- We also assume an ideal optimizer; the risk minimization can always be solved perfectly and efficiently.
- We know \mathbb{P}_{xy} .

How should f be chosen?



RISK MINIMIZER / 2

The f with minimal risk across all (measurable) functions is called the **risk minimizer, population minimizer or Bayes optimal model**.



$$\begin{aligned} f^* &= \arg \min_{f: \mathcal{X} \rightarrow \mathbb{R}^g} \mathcal{R}_L(f) = \arg \min_{f: \mathcal{X} \rightarrow \mathbb{R}^g} \mathbb{E}_{xy} [L(y, f(\mathbf{x}))] \\ &= \arg \min_{f: \mathcal{X} \rightarrow \mathbb{R}^g} \int L(y, f(\mathbf{x})) d\mathbb{P}_{xy}. \end{aligned}$$

The resulting risk is called **Bayes risk**

$$\mathcal{R}_L^* = \inf_{f: \mathcal{X} \rightarrow \mathbb{R}^g} \mathcal{R}_L(f)$$

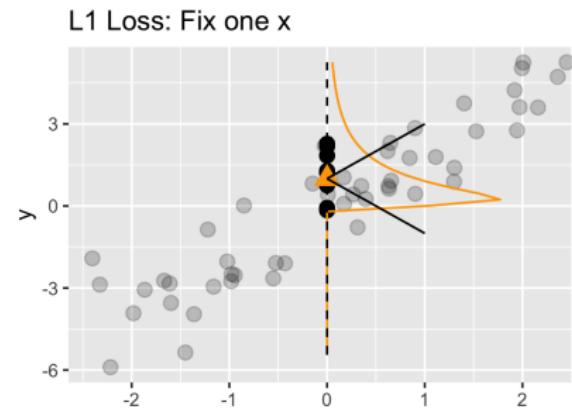
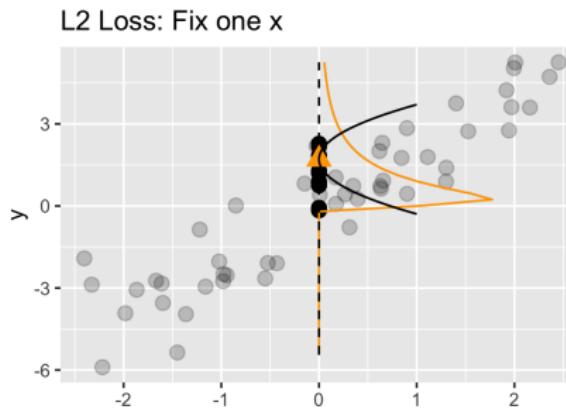
OPTIMAL POINT-WISE PREDICTIONS

To derive the risk minimizer, observe that by the law of total expectation

$$\mathcal{R}_L(f) = \mathbb{E}_{xy} [L(y, f(\mathbf{x}))] = \mathbb{E}_{\mathbf{x}} [\mathbb{E}_{y|x} [L(y, f(\mathbf{x})) \mid \mathbf{x} = \mathbf{x}]] .$$

- We can choose $f(\mathbf{x})$ as we want (unrestricted hypothesis space, no assumed functional form)
- Hence, for a fixed value $\mathbf{x} \in \mathcal{X}$ we can select **any** value c we want to predict. So we construct the **point-wise optimizer**

$$f^*(\mathbf{x}) = \operatorname{argmin}_c \mathbb{E}_{y|x} [L(y, c) \mid \mathbf{x} = \mathbf{x}] \quad \forall \mathbf{x} \in \mathcal{X}.$$



THEORETICAL AND EMPIRICAL RISK

The risk minimizer is mainly a theoretical tool:

- In practice we need to restrict the hypothesis space \mathcal{H} such that we can efficiently search over it.
- In practice we (usually) do not know \mathbb{P}_{xy} . Instead of $\mathcal{R}(f)$, we are optimizing the empirical risk



$$\hat{f} = \arg \min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) = \arg \min_{f \in \mathcal{H}} \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)$$

Note that according to the **law of large numbers** (LLN), the empirical risk converges to the true risk (but beware of overfitting!):

$$\bar{\mathcal{R}}_{\text{emp}}(f) = \frac{1}{n} \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) \xrightarrow{n \rightarrow \infty} \mathcal{R}(f).$$

ESTIMATION AND APPROXIMATION ERROR

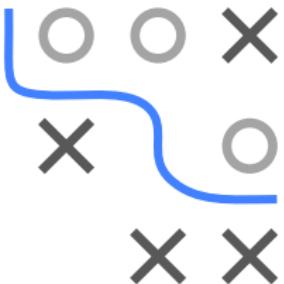
Goal of learning: Train a model \hat{f} for which the true risk $\mathcal{R}_L(\hat{f})$ is close to the Bayes risk \mathcal{R}_L^* . In other words, we want the **Bayes regret**

$$\mathcal{R}_L(\hat{f}) - \mathcal{R}_L^*$$

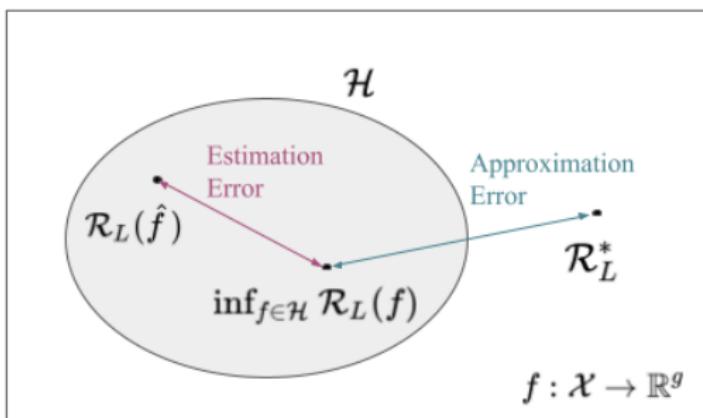
to be as low as possible.

The Bayes regret can be decomposed as follows:

$$\mathcal{R}_L(\hat{f}) - \mathcal{R}_L^* = \underbrace{\left[\mathcal{R}_L(\hat{f}) - \inf_{f \in \mathcal{H}} \mathcal{R}_L(f) \right]}_{\text{estimation error}} + \underbrace{\left[\inf_{f \in \mathcal{H}} \mathcal{R}_L(f) - \mathcal{R}_L^* \right]}_{\text{approximation error}}$$



ESTIMATION AND APPROXIMATION ERROR / 2



- $\mathcal{R}_L(\hat{f}) - \inf_{f \in \mathcal{H}} \mathcal{R}(f)$ is the **estimation error**. We fit \hat{f} via empirical risk minimization and (usually) use approximate optimization, so we usually do not find the optimal $f \in \mathcal{H}$.
- $\inf_{f \in \mathcal{H}} \mathcal{R}_L(f) - \mathcal{R}_L^*$ is the **approximation error**. We need to restrict to a hypothesis space \mathcal{H} which might not even contain the Bayes optimal model f^* .

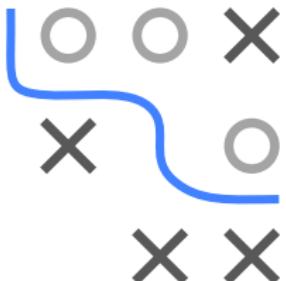
(UNIVERSALLY) CONSISTENT LEARNERS

Consistency is an asymptotic property of a learning algorithm, which ensures the algorithm returns **the correct model** when given **unlimited data**.

Let $\mathcal{I} : \mathbb{D} \rightarrow \mathcal{H}$ be a learning algorithm that takes a training set $\mathcal{D}_{\text{train}} \sim \mathbb{P}_{xy}$ of size n_{train} and estimates a model $\hat{f} : \mathcal{X} \rightarrow \mathbb{R}^g$.

The learning method \mathcal{I} is said to be **consistent** w.r.t. a certain distribution \mathbb{P}_{xy} if the risk of the estimated model \hat{f} converges in probability (\xrightarrow{p}) to the Bayes risk \mathcal{R}^* when n_{train} goes to ∞ :

$$\mathcal{R}(\mathcal{I}(\mathcal{D}_{\text{train}})) \xrightarrow{p} \mathcal{R}_L^* \quad \text{for } n_{\text{train}} \rightarrow \infty.$$



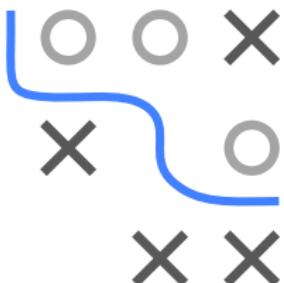
(UNIVERSALLY) CONSISTENT LEARNERS / 2

Consistency is defined w.r.t. a particular distribution \mathbb{P}_{xy} . But since we usually do not know \mathbb{P}_{xy} , consistency does not offer much help to choose an algorithm for a particular task.

More interesting is the stronger concept of **universal consistency**: An algorithm is universally consistent if it is consistent for **any** distribution.

In Stone's famous consistency theorem from 1977, the universal consistency of a weighted average estimator as KNN was proven. Many other ML models have since then been proven to be universally consistent (SVMs, ANNs, etc.).

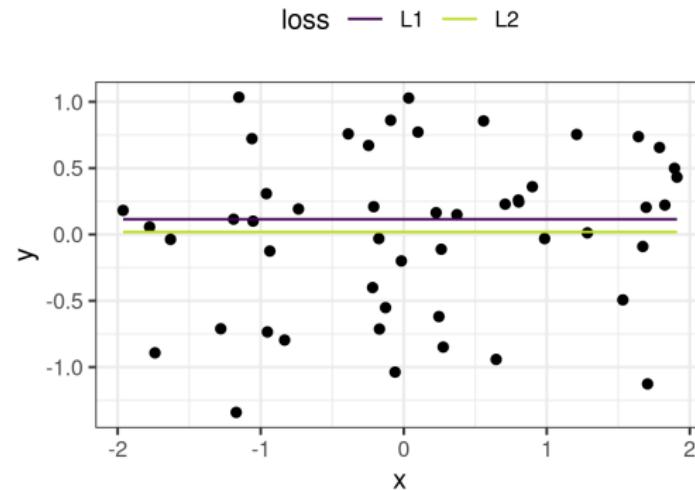
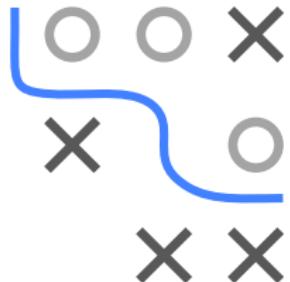
Note that universal consistency is obviously a desirable property - however, (universal) consistency does not tell us anything about convergence rates ...



OPTIMAL CONSTANT MODEL

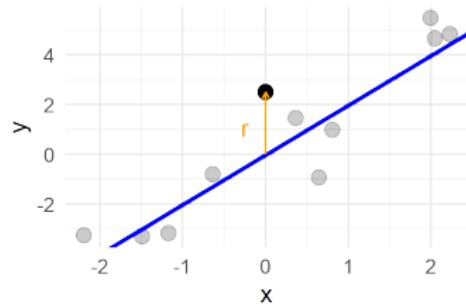
While the risk minimizer gives us the (theoretical) optimal solution, the **optimal constant model** (also: featureless predictor) gives us a computable empirical lower baseline solution.

The constant model is the model $f(\mathbf{x}) = \theta$ that optimizes the empirical risk $\mathcal{R}_{\text{emp}}(\theta)$.



Introduction to Machine Learning

Pseudo-Residuals and Gradient Descent



Learning goals

- Know the concept of pseudo-residuals
- Understand the relationship between pseudo-residuals and gradient descent

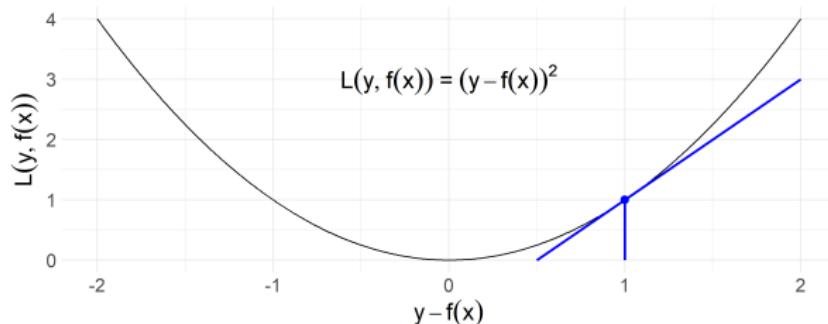
PSEUDO-RESIDUALS

- In regression, residuals are defined as $r := y - f(\mathbf{x})$.
- We further define **pseudo-residuals** as the negative first derivatives of loss functions w.r.t. $f(\mathbf{x})$

$$\tilde{r} := -\frac{\partial L(y, f(\mathbf{x}))}{\partial f(\mathbf{x})}.$$



- This definition also holds for score / probability based classifiers.
- Note that \tilde{r} depends on y and $f(\mathbf{x})$ and L .



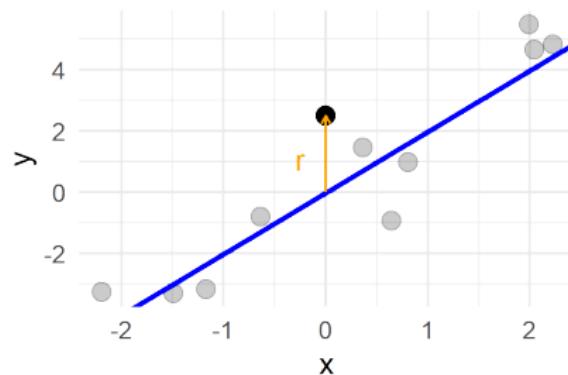
BEST POINT-WISE UPDATE

Assume we have (partially) fitted a model $f(\mathbf{x})$ to data \mathcal{D} .

Assume we could update $f(\mathbf{x})$ point-wise as we like. For a fixed $\mathbf{x} \in \mathcal{X}$, the best point-wise update is the direction of the residual $r = y - f(\mathbf{x})$



$$f(\mathbf{x}) \leftarrow f(\mathbf{x}) + r$$



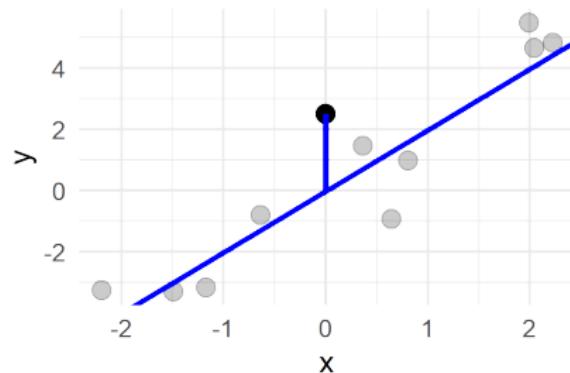
BEST POINT-WISE UPDATE

Assume we have (partially) fitted a model $f(\mathbf{x})$ to data \mathcal{D} .

Assume we could update $f(\mathbf{x})$ point-wise as we like. For a fixed $\mathbf{x} \in \mathcal{X}$, the best point-wise update is the direction of the residual $r = y - f(\mathbf{x})$

$$f(\mathbf{x}) \leftarrow f(\mathbf{x}) + r$$

The point-wise error at this specific \mathbf{x} becomes 0.



APPROXIMATE BEST POINT-WISE UPDATE

When applying gradient descent (GD) to compute a point-wise update of $f(\mathbf{x})$, we would go a step into the direction of the negative gradient

$$f(\mathbf{x}) \leftarrow f(\mathbf{x}) - \frac{\partial L(y, f(\mathbf{x}))}{\partial f(\mathbf{x})}.$$

which is the direction of the pseudo-residual

$$f(\mathbf{x}) \leftarrow f(\mathbf{x}) + \tilde{r}$$



Iteratively stepping towards the direction of the pseudo-residuals is the underlying idea of gradient boosting, which is a learning algorithm that will be covered in a later chapter.

GD IN ML AND PSEUDO-RESIDUALS

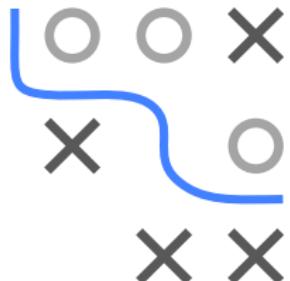
- In GD, we move in the direction of the negative gradient by updating the parameters:

$$\theta^{[t+1]} = \theta^{[t]} - \alpha^{[t]} \cdot \nabla_{\theta} \mathcal{R}_{\text{emp}}(\theta) |_{\theta=\theta^{[t]}}$$

- This can be seen as approximating the unexplained information (measured by the loss) through a model update.
- Using the chain rule:

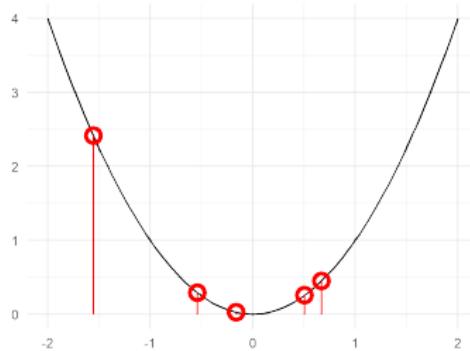
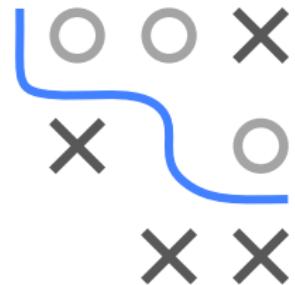
$$\begin{aligned}\nabla_{\theta} \mathcal{R}_{\text{emp}}(\theta) &= \sum_{i=1}^n \frac{\partial L(y^{(i)}, f)}{\partial f} \Bigg|_{f=f(\mathbf{x}^{(i)} | \theta)} \cdot \nabla_{\theta} f(\mathbf{x}^{(i)} | \theta) \\ &= - \sum_{i=1}^n \tilde{r}^{(i)} \cdot \nabla_{\theta} f(\mathbf{x}^{(i)} | \theta).\end{aligned}$$

- Hence the update is determined by a loss-optimal directional change of the model output and a loss-independent derivate of f . This is a very flexible, nearly loss-independent formulation of GD.



Introduction to Machine Learning

Regression Losses: L2 and L1 loss



Learning goals

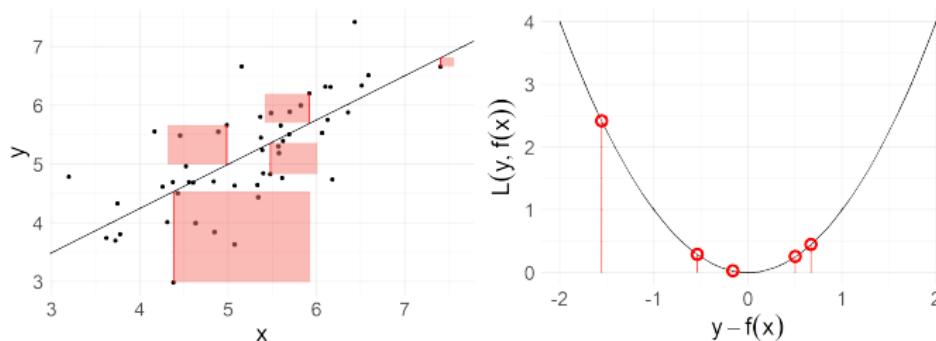
- Derive the risk minimizer of the L2-loss
- Derive the optimal constant model for the L2-loss
- Know risk minimizer and optimal constant model for L1-loss

L2-LOSS

$$L(y, f) = (y - f)^2 \quad \text{or} \quad L(y, f) = 0.5(y - f)^2$$

- Tries to reduce large residuals (if residual is twice as large, loss is 4 times as large), hence outliers in y can become problematic
- Analytic properties: convex, differentiable \Rightarrow gradient no problem in loss minimization

(Warning: $\mathcal{R}_{\text{emp}}(f)$ can still be non-smooth/non-convex due to $f(\mathbf{x})$)



L2-LOSS: OPTIMAL CONSTANT MODEL

Let us consider the (true) risk for $\mathcal{Y} = \mathbb{R}$ and L2-Loss

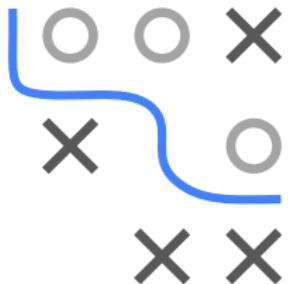
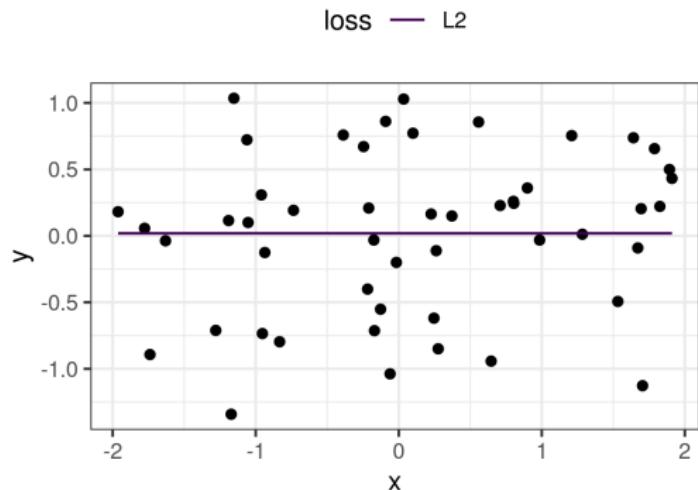
$L(y, f) = (y - f)^2$ with \mathcal{H} restricted to constants. The optimal constant model f_c^* in terms of the theoretical risk is the expected value over y :

$$\begin{aligned}f_c^* &= \arg \min_{c \in \mathbb{R}} \mathbb{E}_{xy} [(y - c)^2] = \arg \min_{c \in \mathbb{R}} \mathbb{E}_y [(y - c)^2] \\&= \arg \min_{c \in \mathbb{R}} \underbrace{\mathbb{E}_y [(y - c)^2]}_{=\text{Var}_y[y-c]=\text{Var}_y[y]} - (\mathbb{E}_y[y] - c)^2 + (\mathbb{E}_y[y] - c)^2 \\&= \arg \min_{c \in \mathbb{R}} \text{Var}_y[y] + (\mathbb{E}_y[y] - c)^2 \\&= \mathbb{E}_y[y]\end{aligned}$$



L2-LOSS: OPTIMAL CONSTANT MODEL

The optimizer \hat{t}_c of the empirical risk is \bar{y} (the empirical mean over $y^{(i)}$), which is the empirical estimate for $\mathbb{E}_y [y]$.



L2-LOSS: OPTIMAL CONSTANT MODEL

Proof:

For the optimal constant model f_c^* for the L2-loss $L(y, f) = (y - f)^2$ we solve the optimization problem

$$\arg \min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) = \arg \min_{\theta \in \mathbb{R}} \sum_{i=1}^n (y^{(i)} - \theta)^2.$$



We calculate the first derivative of \mathcal{R}_{emp} w.r.t. θ and set it to 0:

$$\frac{\partial \mathcal{R}_{\text{emp}}(\theta)}{\partial \theta} = -2 \sum_{i=1}^n (y^{(i)} - \theta) \stackrel{!}{=} 0$$

$$\sum_{i=1}^n y^{(i)} - n\theta = 0$$

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^n y^{(i)} =: \bar{y}.$$

L2-LOSS: RISK MINIMIZER

Let us consider the (true) risk for $\mathcal{Y} = \mathbb{R}$ and the L2-Loss

$L(y, f) = (y - f)^2$ with unrestricted $\mathcal{H} = \{f : \mathcal{X} \rightarrow \mathbb{R}^g\}$.

- By the law of total expectation

$$\begin{aligned}\mathcal{R}_L(f) &= \mathbb{E}_{xy} [L(y, f(\mathbf{x}))] = \mathbb{E}_x [\mathbb{E}_{y|x} [L(y, f(\mathbf{x})) \mid \mathbf{x} = \mathbf{x}]] \\ &= \mathbb{E}_x [\mathbb{E}_{y|x} [(y - f(\mathbf{x}))^2 \mid \mathbf{x} = \mathbf{x}]].\end{aligned}$$

- Since \mathcal{H} is unrestricted, at any point $\mathbf{x} = \mathbf{x}$, we can predict any value c we want. The best point-wise prediction is the cond. mean

$$f^*(\mathbf{x}) = \arg \min_c \mathbb{E}_{y|x} [(y - c)^2 \mid \mathbf{x} = \mathbf{x}] \stackrel{(*)}{=} \mathbb{E}_{y|x} [y \mid \mathbf{x}].$$

(*) follows from the derivation of f_c^*

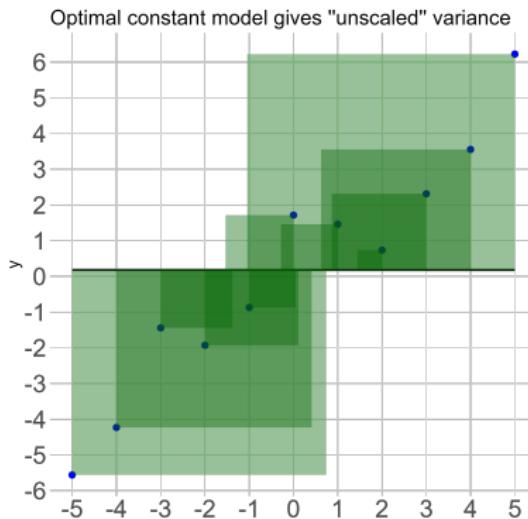


L2 LOSS MEANS MINIMIZING VARIANCE

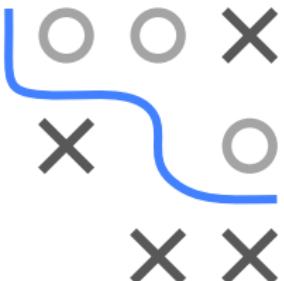
Rethinking what we did in the opt. constant model: We optimized for a constant whose squared distance to all data points is minimal (in sum, or on average). This turned out to be the mean.

What if we calculate the loss of $\hat{\theta} = \bar{y}$? That's $\mathcal{R}_{\text{emp}} = \sum_{i=1}^n (y^{(i)} - \bar{y})^2$.

Average this by $\frac{1}{n}$ or $\frac{1}{n-1}$ to obtain variance.



- Generally, if model yields unbiased predictions,
 $\mathbb{E}_{y | x} [y - f(x) | x] = 0$, using L2-loss means minimizing variance of model residuals
- Same holds for the pointwise construction / conditional distribution considered before

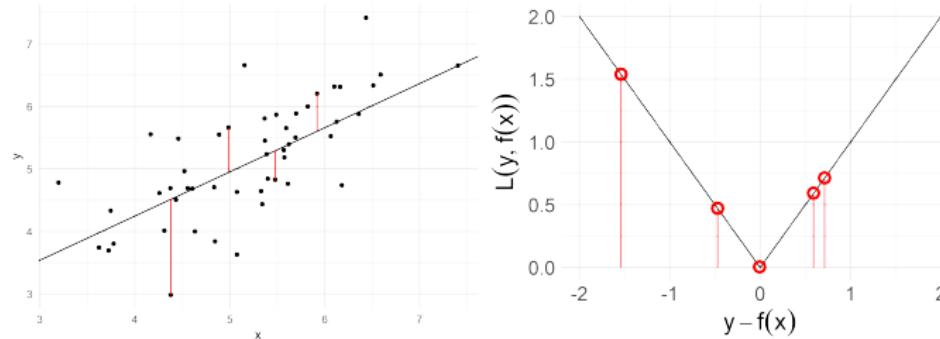


L1-LOSS

The L1 loss is defined as

$$L(y, f) = |y - f|$$

- More robust than L_2 , outliers in y are less problematic.
- Analytical properties: convex, not differentiable for $y = f(x)$ (optimization becomes harder).



L1-LOSS: RISK MINIMIZER

We calculate the (true) risk for the $L1$ -Loss $L(y, f) = |y - f|$ with unrestricted $\mathcal{H} = \{f : \mathcal{X} \rightarrow \mathcal{Y}\}$.

- We use the law of total expectation



$$\mathcal{R}(f) = \mathbb{E}_x [\mathbb{E}_{y|x} [|y - f(\mathbf{x})| \mid \mathbf{x} = \mathbf{x}]] .$$

- As the functional form of f is not restricted, we can just optimize point-wise at any point $\mathbf{x} = \mathbf{x}$. The best prediction at $\mathbf{x} = \mathbf{x}$ is then

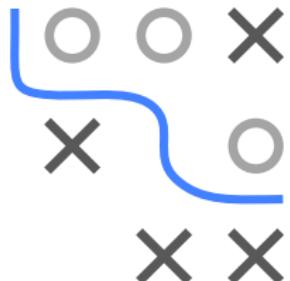
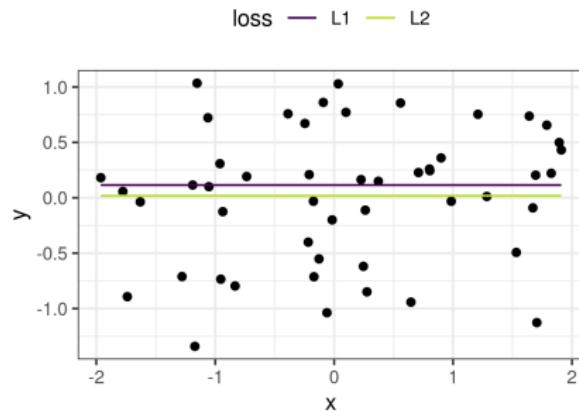
$$f^*(\mathbf{x}) = \arg \min_c \mathbb{E}_{y|x} [|y - c|] = \text{med}_{y|x} [y \mid \mathbf{x}] .$$

L1-LOSS: OPTIMAL CONSTANT MODEL

The optimal constant model in terms of the theoretical risk for the L1 loss is the median over y :

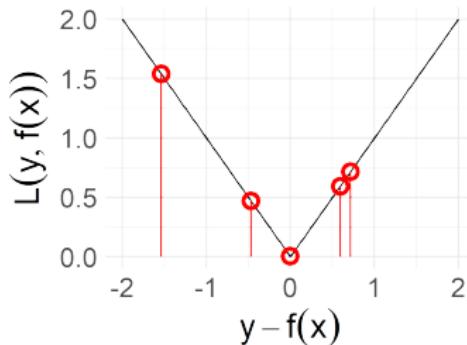
$$f_c^* = \text{med}_{y|x} [y | \mathbf{x}] \stackrel{\text{drop } \mathbf{x}}{=} \text{med}_y [y]$$

The optimizer \hat{f}_c of the empirical risk is $\text{med}(y^{(i)})$ over $y^{(i)}$, which is the empirical estimate for $\text{med}_y [y]$.



Introduction to Machine Learning

L1 Risk Minimizer (Deep-Dive)



Learning goals

- Derive the risk minimizer of the L1-loss
- Derive the optimal constant model for the L1-loss



L1-LOSS: RISK MINIMIZER

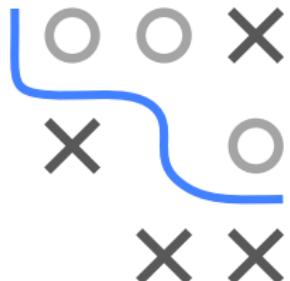
Proof: Let $p(y)$ be the density function of y . Then:

$$\begin{aligned}\arg \min_c \mathbb{E} [|y - c|] &= \arg \min_c \int_{-\infty}^{\infty} |y - c| p(y) dy \\ &= \arg \min_c \int_{-\infty}^c -(y - c) p(y) dy + \int_c^{\infty} (y - c) p(y) dy\end{aligned}$$

We now compute the derivative of the above term and set it to 0

$$\begin{aligned}0 &= \frac{\partial}{\partial c} \left(\int_{-\infty}^c -(y - c) p(y) dy + \int_c^{\infty} (y - c) p(y) dy \right) \\ &\stackrel{* \text{ Leibniz}}{=} \int_{-\infty}^c p(y) dy - \int_c^{\infty} p(y) dy = \mathbb{P}_y(y \leq c) - (1 - \mathbb{P}_y(y \leq c)) \\ &= 2 \cdot \mathbb{P}_y(y \leq c) - 1 \\ \Leftrightarrow 0.5 &= \mathbb{P}_y(y \leq c),\end{aligned}$$

which yields $c = \text{med}_y(y)$.



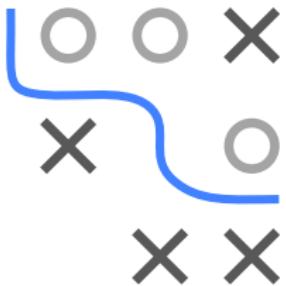
L1-LOSS: RISK MINIMIZER

* Note that since we are computing the derivative w.r.t. the integration boundaries, we need to use Leibniz integration rule

$$\begin{aligned}\frac{\partial}{\partial c} \left(\int_a^c g(c, y) dy \right) &= g(c, c) + \int_a^c \frac{\partial}{\partial c} g(c, y) dy \\ \frac{\partial}{\partial c} \left(\int_c^a g(c, y) dy \right) &= -g(c, c) + \int_c^a \frac{\partial}{\partial c} g(c, y) dy\end{aligned}$$

We get

$$\begin{aligned}& \frac{\partial}{\partial c} \left(\int_{-\infty}^c -(y - c) p(y) dy + \int_c^{\infty} (y - c) p(y) dy \right) \\ &= \frac{\partial}{\partial c} \left(\int_{-\infty}^c \underbrace{-(y - c) p(y)}_{g_1(c,y)} dy \right) + \frac{\partial}{\partial c} \left(\int_c^{\infty} \underbrace{(y - c) p(y)}_{g_2(c,y)} dy \right) \\ &= \underbrace{g_1(c, c)}_{=0} + \int_{-\infty}^c \frac{\partial}{\partial c}(-(y - c)) p(y) dy - \underbrace{g_2(c, c)}_{=0} + \int_c^{\infty} \frac{\partial}{\partial c}(y - c) p(y) dy \\ &= \int_{-\infty}^c p(y) dy + \int_c^{\infty} -p(y) dy.\end{aligned}$$



L1-LOSS: OPTIMAL CONSTANT MODEL

Proof:

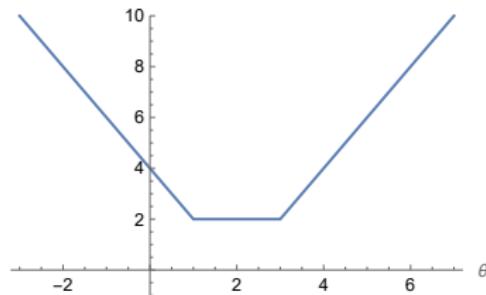
- Firstly note that for $n = 1$ the median $\hat{\theta} = \text{med}(y^{(i)}) = y^{(1)}$ obviously minimizes the emp. risk \mathcal{R}_{emp} using the $L1$ loss.
- Hence let $n > 1$ in the following For $a, b \in \mathbb{R}$, define

$$S_{a,b} : \mathbb{R} \rightarrow \mathbb{R}_0^+, \theta \mapsto |a - \theta| + |b - \theta|$$

Any $\hat{\theta} \in [a, b]$ minimizes $S_{a,b}(\theta)$, because it holds that

$$S_{a,b}(\theta) = \begin{cases} |a - b|, & \text{for } \theta \in [a, b] \\ |a - b| + 2 \cdot \min\{|a - \theta|, |b - \theta|\}, & \text{otherwise.} \end{cases}$$

$$S(\theta) = |a-\theta| + |b-\theta| \text{ for } (a,b)=(3,1)$$

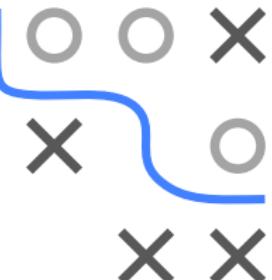


L1-LOSS: OPTIMAL CONSTANT MODEL

W.l.o.g. assume now that all $y^{(i)}$ are sorted in increasing order.

Let us define $i_{\max} = n/2$ for n even and $i_{\max} = (n - 1)/2$ for n odd
and consider the intervals

$$\mathcal{I}_i := [y^{(i)}, y^{(n+1-i)}], i \in \{1, \dots, i_{\max}\}.$$



By construction $\mathcal{I}_{j+1} \subseteq \mathcal{I}_j$ for $j \in \{1, \dots, i_{\max} - 1\}$ and $\mathcal{I}_{i_{\max}} \subseteq \mathcal{I}_i$.
With this, \mathcal{R}_{emp} can be expressed as

$$\begin{aligned}\mathcal{R}_{\text{emp}}(\theta) &= \sum_{i=1}^n L(y^{(i)}, \theta) = \sum_{i=1}^n |y^{(i)} - \theta| \\ &= \underbrace{|y^{(1)} - \theta| + |y^{(n)} - \theta|}_{=S_{y^{(1)}, y^{(n)}}(\theta)} + \underbrace{|y^{(2)} - \theta| + |y^{(n-1)} - \theta| + \dots}_{=S_{y^{(2)}, y^{(n-1)}}(\theta)} + \dots \\ &= \begin{cases} \sum_{i=1}^{i_{\max}} S_{y^{(i)}, y^{(n+1-i)}}(\theta) & \text{for } n \text{ is even} \\ \sum_{i=1}^{i_{\max}} (S_{y^{(i)}, y^{(n+1-i)}}(\theta)) + |y^{((n+1)/2)} - \theta| & \text{for } n \text{ is odd.} \end{cases}\end{aligned}$$

L1-LOSS: OPTIMAL CONSTANT MODEL

From this follows that

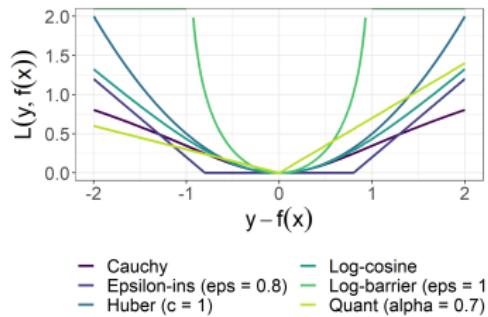
- for “ n is even”: $\hat{\theta} \in \mathcal{I}_{i_{\max}} = [y^{(n/2)}, y^{(n/2+1)}]$ minimizes S_i for all $i \in \{1, \dots, i_{\max}\}$ \Rightarrow it minimizes \mathcal{R}_{emp} ,
- for “ n is odd”: $\hat{\theta} = y^{((n+1)/2)} \in \mathcal{I}_{i_{\max}}$ minimizes S_i for all $i \in \{1, \dots, i_{\max}\}$ and it’s minimal for $|y^{((n+1)/2)} - \theta|$
 \Rightarrow it minimizes \mathcal{R}_{emp} .

Since the median fulfills these conditions, we can conclude that it minimizes the $L1$ loss.



Introduction to Machine Learning

Advanced Regression Losses



Learning goals

- Know the Huber loss
- Know the log-cosh loss
- Know the Cauchy loss
- Know the log-barrier loss
- Know the ϵ -insensitive loss
- Know the quantile loss



ADVANCED LOSS FUNCTIONS

Special loss functions can be used to estimate non-standard posterior components, to measure errors in a custom way or are designed to have special properties like robustness.



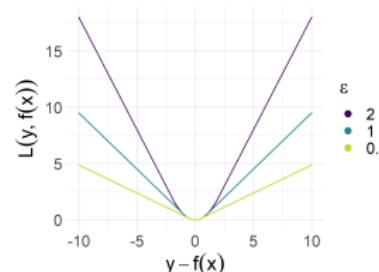
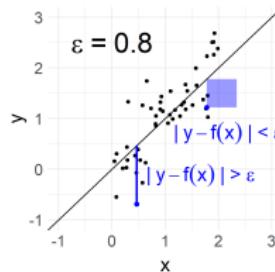
Examples:

- Quantile loss: Overestimating a clinical parameter might not be as bad as underestimating it.
- Log-barrier loss: Extremely under- or overestimating demand in production would put company profit at risk.
- ϵ -insensitive loss: A certain amount of deviation in production does no harm, larger deviations do.

HUBER LOSS

$$L(y, f) = \begin{cases} \frac{1}{2}(y - f)^2 & \text{if } |y - f| \leq \epsilon \\ \epsilon|y - f| - \frac{1}{2}\epsilon^2 & \text{otherwise} \end{cases}, \quad \epsilon > 0$$

- Piece-wise combination of $L1/L2$ to have robustness/smoothness
- Analytic properties: convex, differentiable (once)

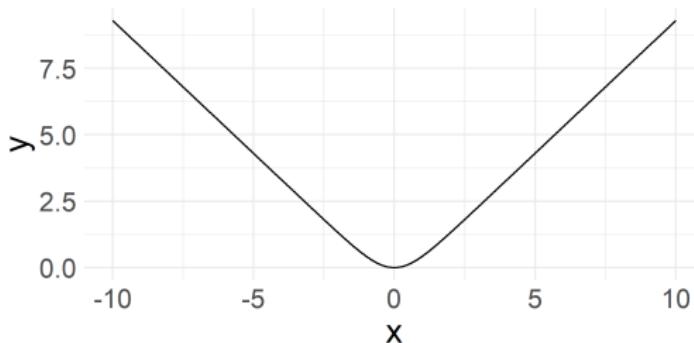
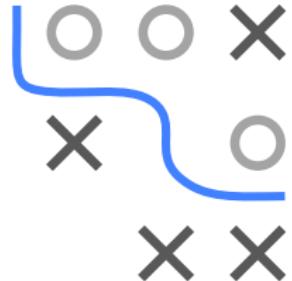


- Risk minimizer and optimal constant do not have a closed-form solution. To fit a model numerical optimization is necessary.
- Solution behaves like **trimmed mean**: a (conditional) mean of two (conditional) quantiles.

LOG-COSH LOSS

$$L(y, f) = \log(\cosh(|y - f|))$$

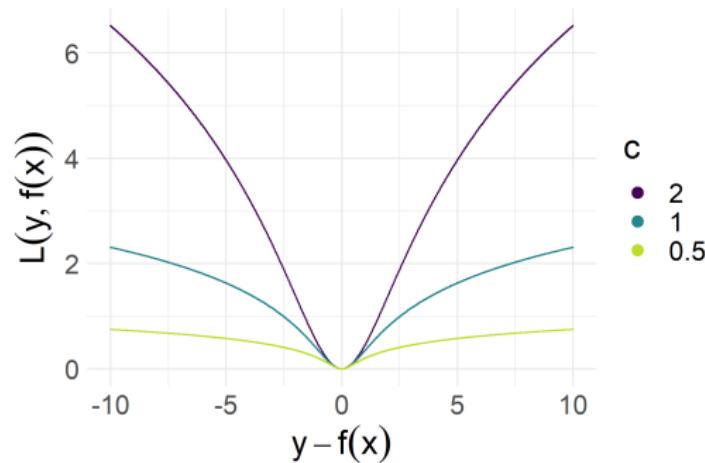
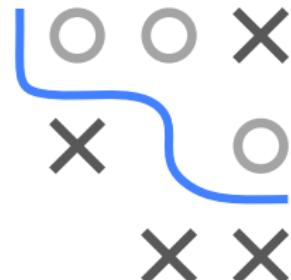
- Logarithm of the hyperbolic cosine of the residual.
- Approximately equal to $0.5(|y - f|)^2$ for small f and to $|y - f| - \log 2$ for large f , meaning it works mostly like $L2$ loss but is less outlier-sensitive.
- Has all the advantages of Huber loss and is, moreover, twice differentiable everywhere.



CAUCHY LOSS

$$L(y, f) = \frac{c^2}{2} \log \left(1 + \left(\frac{|y - f|}{c} \right)^2 \right), \quad c \in \mathbb{R}$$

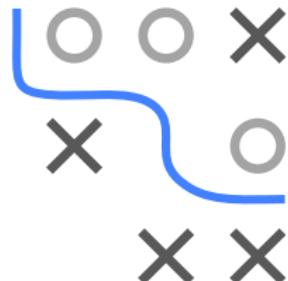
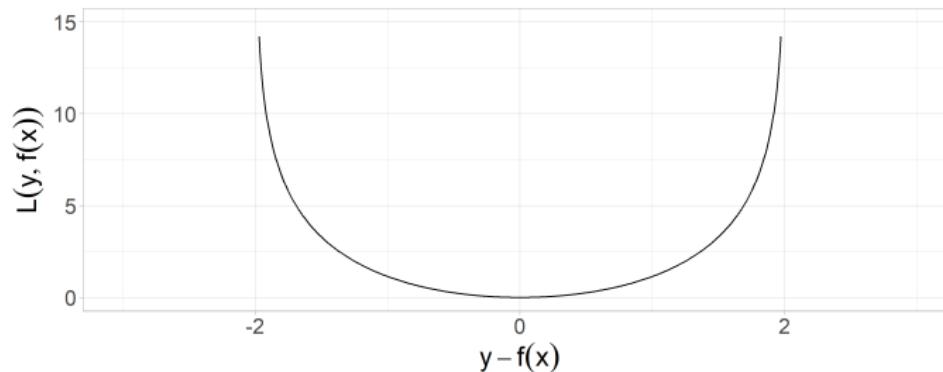
- Particularly robust toward outliers (controllable via c).
- Analytic properties: differentiable, but not convex!



LOG-BARRIER LOSS

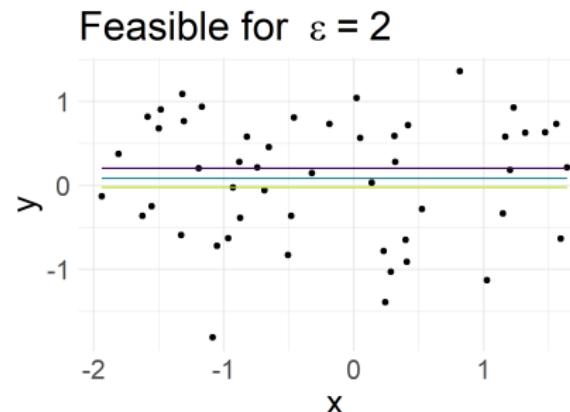
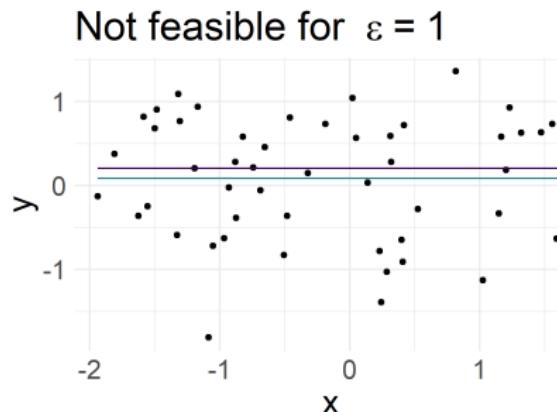
$$L(y, f) = \begin{cases} -\epsilon^2 \cdot \log\left(1 - \left(\frac{|y-f|}{\epsilon}\right)^2\right) & \text{if } |y-f| \leq \epsilon \\ \infty & \text{if } |y-f| > \epsilon \end{cases}$$

- Behaves like L_2 loss for small residuals.
- We use this if we don't want residuals larger than ϵ at all.
- No guarantee that the risk minimization problem has a solution.
- Plot shows log-barrier loss for $\epsilon = 2$:



LOG-BARRIER LOSS

- Note that the optimization problem has no (finite) solution if there is no way to fit a constant where all residuals are smaller than ϵ .



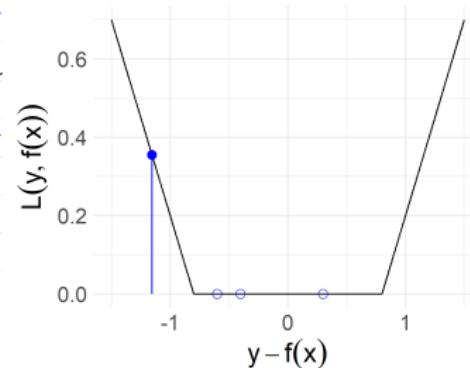
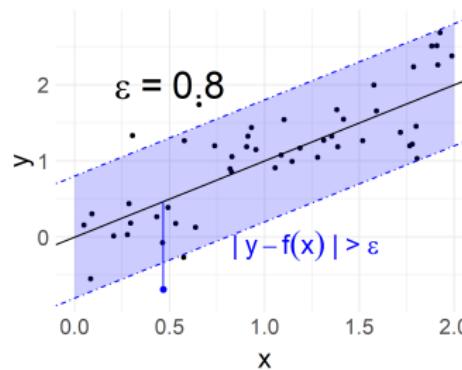
Loss — L1 — L2 — log-barrier



ϵ -INSENSITIVE LOSS

$$L(y, f) = \begin{cases} 0 & \text{if } |y - f| \leq \epsilon \\ |y - f| - \epsilon & \text{otherwise} \end{cases}, \quad \epsilon \in \mathbb{R}_+$$

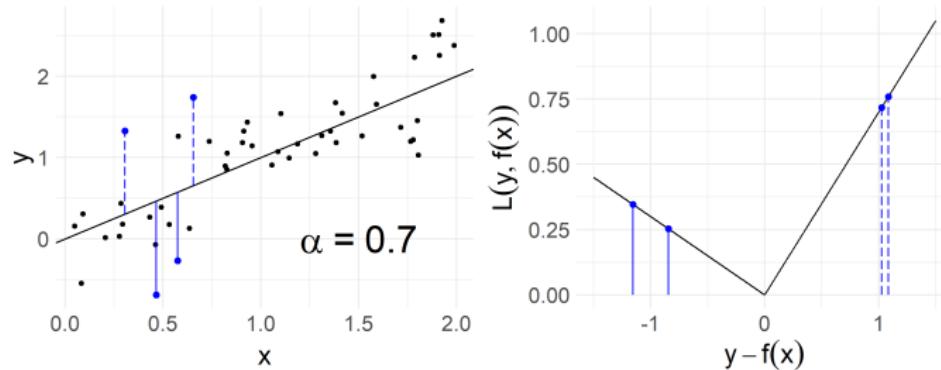
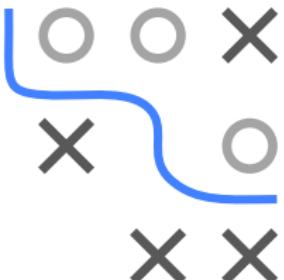
- Modification of L_1 loss, errors below ϵ accepted without penalty.
- Used in SVM regression.
- Properties: convex and not differentiable for $y - f \in \{-\epsilon, \epsilon\}$.



QUANTILE LOSS / PINBALL LOSS

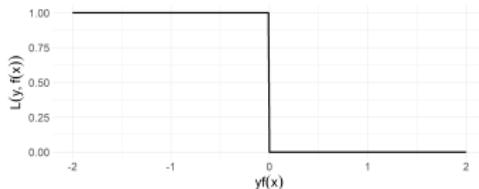
$$L(y, f) = \begin{cases} (1 - \alpha)(f - y) & \text{if } y < f \\ \alpha(y - f) & \text{if } y \geq f \end{cases}, \quad \alpha \in (0, 1)$$

- Extension of $L1$ loss (equal to $L1$ for $\alpha = 0.5$).
- Weights either positive or negative residuals more strongly.
- $\alpha < 0.5$ ($\alpha > 0.5$) penalty to over-estimation (under-estimation)
- Risk minimizer is (conditional) α -quantile (median for $\alpha = 0.5$).



Introduction to Machine Learning

0-1-Loss



Learning goals

- Derive the risk minimizer of the 0-1-loss
- Derive the optimal constant model for the 0-1-loss



0-1-LOSS

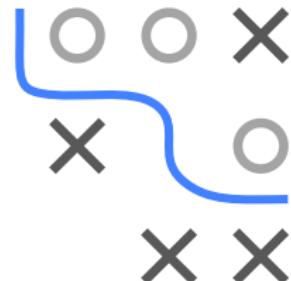
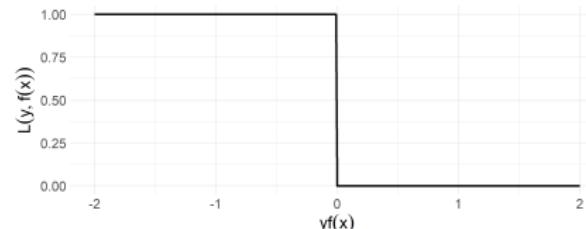
- Let us first consider a discrete classifier $h(\mathbf{x}) : \mathcal{X} \rightarrow \mathcal{Y}$.
- The most natural choice for $L(y, h(\mathbf{x}))$ is the 0-1-loss

$$L(y, h(\mathbf{x})) = \mathbb{1}_{\{y \neq h(\mathbf{x})\}} = \begin{cases} 1 & \text{if } y \neq h(\mathbf{x}) \\ 0 & \text{if } y = h(\mathbf{x}) \end{cases}$$

- For the binary case ($g = 2$) we can express the 0-1-loss for a scoring classifier $f(\mathbf{x})$ based on the margin $\nu := yf(\mathbf{x})$

$$L(y, f(\mathbf{x})) = \mathbb{1}_{\{\nu < 0\}} = \mathbb{1}_{\{yf(\mathbf{x}) < 0\}}.$$

- Analytic properties: Not continuous, even for linear f the optimization problem is NP-hard and close to intractable.

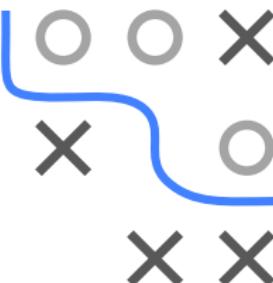


0-1-LOSS: RISK MINIMIZER

By the law of total expectation we can in general rewrite the risk as
(this all works for the multiclass case with 0-1)

$$\begin{aligned}\mathcal{R}(f) &= \mathbb{E}_{xy} [L(y, f(\mathbf{x}))] = \mathbb{E}_x [\mathbb{E}_{y|x} [L(y, f(\mathbf{x}))]] \\ &= \mathbb{E}_x \left[\sum_{k \in \mathcal{Y}} L(k, f(\mathbf{x})) \mathbb{P}(y = k | \mathbf{x}) \right],\end{aligned}$$

with $\mathbb{P}(y = k | \mathbf{x})$ the posterior probability for class k . For the binary case we denote $\eta(\mathbf{x}) := \mathbb{P}(y = 1 | \mathbf{x})$ and the expression becomes



$$\mathcal{R}(f) = \mathbb{E}_x [L(1, \pi(\mathbf{x})) \cdot \eta(\mathbf{x}) + L(0, \pi(\mathbf{x})) \cdot (1 - \eta(\mathbf{x}))].$$

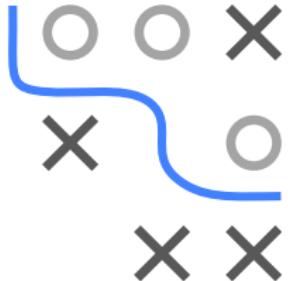
0-1-LOSS: RISK MINIMIZER / 2

We compute the point-wise optimizer of the above term for the 0-1-loss (defined on a discrete classifier $h(\mathbf{x})$):

$$\begin{aligned} h^*(\mathbf{x}) &= \arg \min_{l \in \mathcal{Y}} \sum_{k \in \mathcal{Y}} L(k, l) \cdot \mathbb{P}(y = k \mid \mathbf{x} = \mathbf{x}) \\ &= \arg \min_{l \in \mathcal{Y}} \sum_{k \neq l} \mathbb{P}(y = k \mid \mathbf{x} = \mathbf{x}) \\ &= \arg \min_{l \in \mathcal{Y}} 1 - \mathbb{P}(y = l \mid \mathbf{x} = \mathbf{x}) \\ &= \arg \max_{l \in \mathcal{Y}} \mathbb{P}(y = l \mid \mathbf{x} = \mathbf{x}), \end{aligned}$$

which corresponds to predicting the most probable class.

Note that sometimes $h^*(\mathbf{x}) = \arg \max_{l \in \mathcal{Y}} \mathbb{P}(y = l \mid \mathbf{x} = \mathbf{x})$ is referred to as the **Bayes optimal classifier** (without closer specification of the loss function used).



0-1-LOSS: RISK MINIMIZER / 3

The Bayes risk for the 0-1-loss (also: Bayes error rate) is

$$\mathcal{R}^* = 1 - \mathbb{E}_x \left[\max_{l \in \mathcal{Y}} \mathbb{P}(y = l \mid \mathbf{x}) \right].$$



In the binary case ($g = 2$) we can write risk minimizer and Bayes risk as follows:

$$h^*(\mathbf{x}) = \begin{cases} 1 & \eta(\mathbf{x}) \geq \frac{1}{2} \\ 0 & \eta(\mathbf{x}) < \frac{1}{2} \end{cases}$$

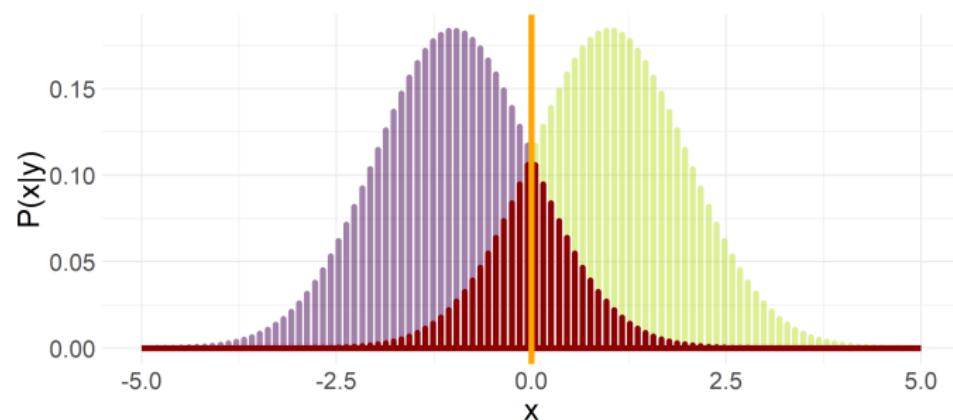
$$\mathcal{R}^* = \mathbb{E}_x [\min(\eta(\mathbf{x}), 1 - \eta(\mathbf{x}))] = 1 - \mathbb{E}_x [\max(\eta(\mathbf{x}), 1 - \eta(\mathbf{x}))].$$

0-1-LOSS: RISK MINIMIZER / 4

Example: Assume that $\mathbb{P}(y = 1) = \frac{1}{2}$ and

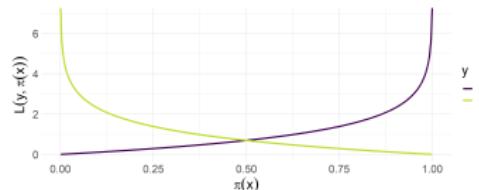
$$\mathbb{P}(x | y) = \begin{cases} \phi_{\mu_1, \sigma^2}(x) & \text{for } y = 0 \\ \phi_{\mu_2, \sigma^2}(x) & \text{for } y = 1 \end{cases}$$

The decision boundary of the Bayes optimal classifier is shown in orange and the Bayes error rate is highlighted as red area.



Introduction to Machine Learning

Bernoulli Loss



Learning goals

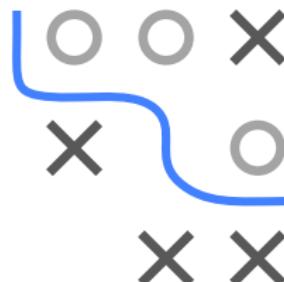
- Know the Bernoulli loss and related losses (log-loss, logistic loss, Binomial loss)
- Derive the risk minimizer
- Derive the optimal constant model
- Understand the connection between log-loss and entropy splitting



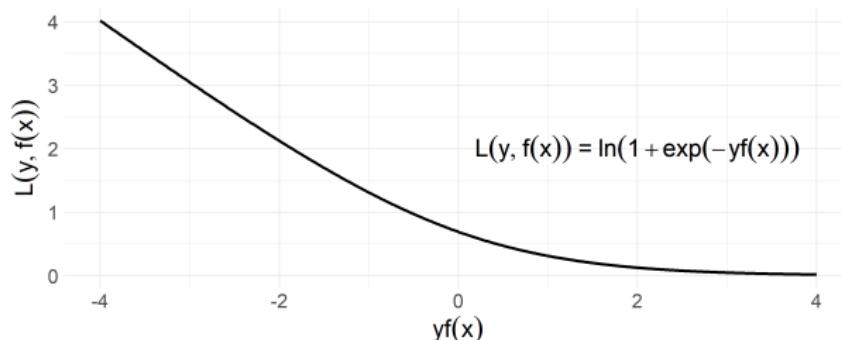
BERNOULLI LOSS

$$L(y, f(\mathbf{x})) = \log(1 + \exp(-y \cdot f(\mathbf{x}))) \quad \text{for } y \in \{-1, +1\}$$

$$L(y, f(\mathbf{x})) = -y \cdot f(\mathbf{x}) + \log(1 + \exp(f(\mathbf{x}))) \quad \text{for } y \in \{0, 1\}$$



- Two equivalent formulations for different label encodings
- Negative log-likelihood of Bernoulli model, e.g., logistic regression
- Convex, differentiable

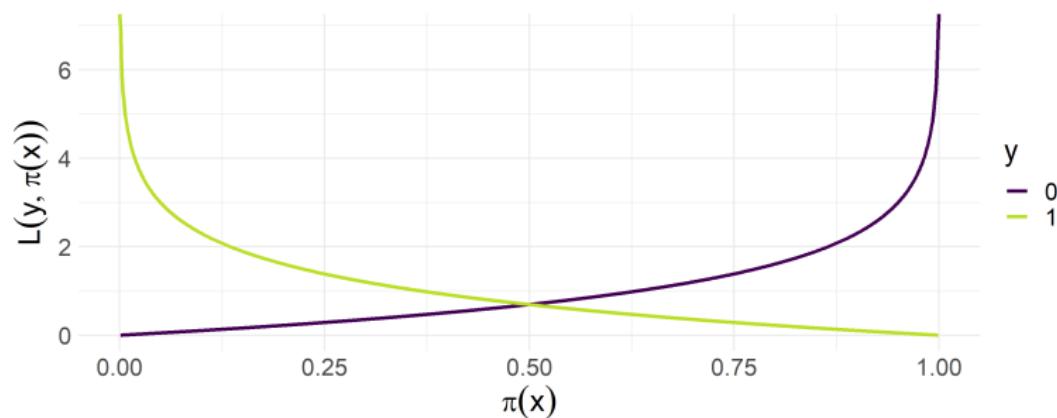


BERNOULLI LOSS ON PROBABILITIES

If scores are transformed into probabilities by the logistic function

$\pi(\mathbf{x}) = (1 + \exp(-f(\mathbf{x})))^{-1}$ (or equivalently if $f(x) = \log\left(\frac{\pi(\mathbf{x})}{1-\pi(\mathbf{x})}\right)$ are the log-odds of $\pi(\mathbf{x})$), we arrive at another equivalent formulation of the loss, where y is again encoded as $\{0, 1\}$:

$$L(y, \pi(\mathbf{x})) = -y \log (\pi(\mathbf{x})) - (1 - y) \log (1 - \pi(\mathbf{x})).$$



BERNOULLI LOSS: RISK MINIMIZER

The risk minimizer for the Bernoulli loss defined for probabilistic classifiers $\pi(\mathbf{x})$ and on $y \in \{0, 1\}$ is

$$\pi^*(\mathbf{x}) = \eta(\mathbf{x}) = \mathbb{P}(y = 1 \mid \mathbf{x} = \mathbf{x}).$$

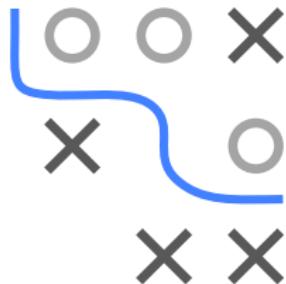
Proof: We can write the risk for binary y as follows:

$$\mathcal{R}(f) = \mathbb{E}_x [L(1, \pi(\mathbf{x})) \cdot \eta(\mathbf{x}) + L(0, \pi(\mathbf{x})) \cdot (1 - \eta(\mathbf{x}))],$$

with $\eta(\mathbf{x}) = \mathbb{P}(y = 1 \mid \mathbf{x})$ (see chapter on the 0-1-loss for more details).

For a fixed \mathbf{x} we compute the point-wise optimal value c by setting the derivative to 0:

$$\begin{aligned} \frac{\partial}{\partial c} (-\log c \cdot \eta(\mathbf{x}) - \log(1 - c) \cdot (1 - \eta(\mathbf{x}))) &= 0 \\ -\frac{\eta(\mathbf{x})}{c} + \frac{1 - \eta(\mathbf{x})}{1 - c} &= 0 \\ \frac{-\eta(\mathbf{x}) + \eta(\mathbf{x})c + c - \eta(\mathbf{x})c}{c(1 - c)} &= 0 \\ c &= \eta(\mathbf{x}). \end{aligned}$$



BERNOULLI LOSS: RISK MINIMIZER / 2

The risk minimizer for the Bernoulli loss defined on $y \in \{-1, 1\}$ and scores $f(\mathbf{x})$ is the point-wise log-odds:

$$f^*(\mathbf{x}) = \log\left(\frac{\mathbb{P}(y | \mathbf{x} = \mathbf{x})}{1 - \mathbb{P}(y | \mathbf{x} = \mathbf{x})}\right).$$

The function is undefined when $P(y | \mathbf{x} = \mathbf{x}) = 1$ or $P(y | \mathbf{x} = \mathbf{x}) = 0$, but predicts a smooth curve which grows when $P(y | \mathbf{x} = \mathbf{x})$ increases and equals 0 when $P(y | \mathbf{x} = \mathbf{x}) = 0.5$.

Proof: As before we minimize

$$\begin{aligned}\mathcal{R}(f) &= \mathbb{E}_{\mathbf{x}} [L(1, f(\mathbf{x})) \cdot \eta(\mathbf{x}) + L(-1, f(\mathbf{x})) \cdot (1 - \eta(\mathbf{x}))] \\ &= \mathbb{E}_{\mathbf{x}} [\log(1 + \exp(-f(\mathbf{x})))\eta(\mathbf{x}) + \log(1 + \exp(f(\mathbf{x}))) (1 - \eta(\mathbf{x}))]\end{aligned}$$



BERNOULLI LOSS: RISK MINIMIZER / 3

For a fixed \mathbf{x} we compute the point-wise optimal value c by setting the derivative to 0:

$$\frac{\partial}{\partial c} \log(1 + \exp(-c))\eta(\mathbf{x}) + \log(1 + \exp(c))(1 - \eta(\mathbf{x})) = 0$$

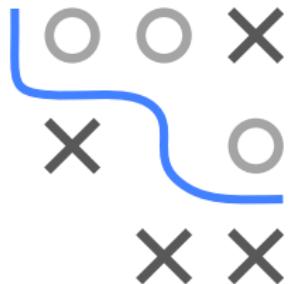
$$-\frac{\exp(-c)}{1 + \exp(-c)}\eta(\mathbf{x}) + \frac{\exp(c)}{1 + \exp(c)}(1 - \eta(\mathbf{x})) = 0$$

$$-\frac{\exp(-c)}{1 + \exp(-c)}\eta(\mathbf{x}) + \frac{1}{1 + \exp(-c)}S(1 - \eta(\mathbf{x})) = 0$$

$$-\eta(\mathbf{x}) + \frac{1}{1 + \exp(-c)} = 0$$

$$\eta(\mathbf{x}) = \frac{1}{1 + \exp(-c)}$$

$$c = \log \left(\frac{\eta(\mathbf{x})}{1 - \eta(\mathbf{x})} \right)$$

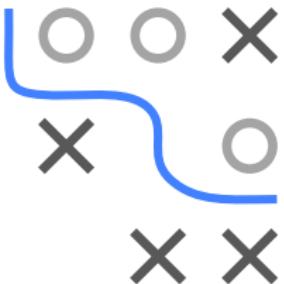


BERNOULLI: OPTIMAL CONSTANT MODEL

The optimal constant probability model $\pi(\mathbf{x}) = \theta$ w.r.t. the Bernoulli loss for labels from $\mathcal{Y} = \{0, 1\}$ is:

$$\hat{\theta} = \arg \min_{\theta} \mathcal{R}_{\text{emp}}(\theta) = \frac{1}{n} \sum_{i=1}^n y^{(i)}$$

Again, this is the fraction of class-1 observations in the observed data. We can simply prove this again by setting the derivative of the risk to 0 and solving for θ .



BERNOULLI: OPTIMAL CONSTANT MODEL / 2

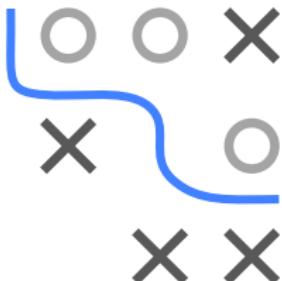
The optimal constant score model $f(\mathbf{x}) = \theta$ w.r.t. the Bernoulli loss labels from $\mathcal{Y} = \{-1, +1\}$ or $\mathcal{Y} = \{0, 1\}$ is:

$$\hat{\theta} = \arg \min_{\theta} \mathcal{R}_{\text{emp}}(\theta) = \log \frac{n_+}{n_-} = \log \frac{n_+/n}{n_-/n}$$

where n_- and n_+ are the numbers of negative and positive observations, respectively.

This again shows a tight (and unsurprising) connection of this loss to log-odds.

Proving this is also a (quite simple) exercise.



BERNOULLI-LOSS: NAMING CONVENTION

We have seen three loss functions that are closely related. In the literature, there are different names for the losses:

$$L(y, f(\mathbf{x})) = \log(1 + \exp(-yf(\mathbf{x}))) \quad \text{for } y \in \{-1, +1\}$$

$$L(y, f(\mathbf{x})) = -y \cdot f(\mathbf{x}) + \log(1 + \exp(f(\mathbf{x}))) \quad \text{for } y \in \{0, 1\}$$



are referred to as Bernoulli, Binomial or logistic loss.

$$L(y, \pi(\mathbf{x})) = -y \log(\pi(\mathbf{x})) - (1 - y) \log(1 - \pi(\mathbf{x})) \quad \text{for } y \in \{0, 1\}$$

is referred to as cross-entropy or log-loss.

We usually refer to all of them as **Bernoulli loss**, and rather make clear whether they are defined on labels $y \in \{0, 1\}$ or $y \in \{-1, +1\}$ and on scores $f(\mathbf{x})$ or probabilities $\pi(\mathbf{x})$.

BERNOULLI LOSS MIN = ENTROPY SPLITTING

When fitting a tree we minimize the risk within each node \mathcal{N} by risk minimization and predict the optimal constant. Another approach that is common in literature is to minimize the average node impurity $\text{Imp}(\mathcal{N})$.

Claim: Entropy splitting $\text{Imp}(\mathcal{N}) = - \sum_{k=1}^g \pi_k^{(\mathcal{N})} \log \pi_k^{(\mathcal{N})}$ is equivalent to minimize risk measured by the Bernoulli loss.

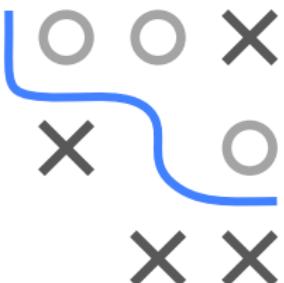
Note that $\pi_k^{(\mathcal{N})} := \frac{1}{n_{\mathcal{N}}} \sum_{(\mathbf{x}, y) \in \mathcal{N}} [y = k]$.

Proof: To prove this we show that the risk related to a subset of observations $\mathcal{N} \subseteq \mathcal{D}$ fulfills

$$\mathcal{R}(\mathcal{N}) = n_{\mathcal{N}} \text{Imp}(\mathcal{N}),$$

where $\mathcal{R}(\mathcal{N})$ is calculated w.r.t. the (multiclass) Bernoulli loss

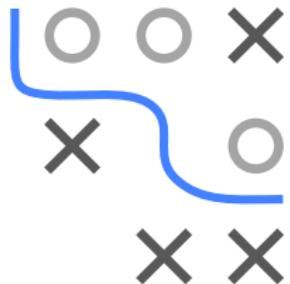
$$L(y, \pi(\mathbf{x})) = - \sum_{k=1}^g [y = k] \log (\pi_k(\mathbf{x})).$$



BERNOULLI LOSS MIN = ENTROPY SPLITTING / 2

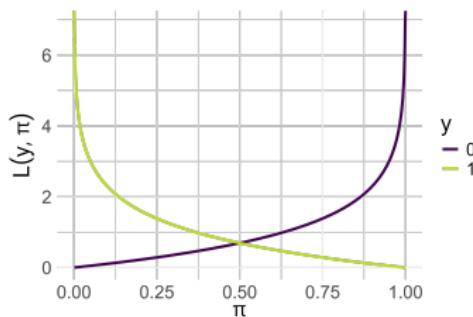
$$\begin{aligned}\mathcal{R}(\mathcal{N}) &= \sum_{(\mathbf{x}, y) \in \mathcal{N}} \left(- \sum_{k=1}^g [y = k] \log \pi_k(\mathbf{x}) \right) \\ &\stackrel{(*)}{=} - \sum_{k=1}^g \sum_{(\mathbf{x}, y) \in \mathcal{N}} [y = k] \log \pi_k^{(\mathcal{N})} \\ &= - \sum_{k=1}^g \log \pi_k^{(\mathcal{N})} \underbrace{\sum_{(\mathbf{x}, y) \in \mathcal{N}} [y = k]}_{n_{\mathcal{N}} \cdot \pi_k^{(\mathcal{N})}} \\ &= - n_{\mathcal{N}} \sum_{k=1}^g \pi_k^{(\mathcal{N})} \log \pi_k^{(\mathcal{N})} = n_{\mathcal{N}} \text{Imp}(\mathcal{N}),\end{aligned}$$

where in $(*)$ the optimal constant per node $\pi_k^{(\mathcal{N})} = \frac{1}{n_{\mathcal{N}}} \sum_{(\mathbf{x}, y) \in \mathcal{N}} [y = k]$ was plugged in.



Introduction to Machine Learning

Advanced Risk Minimization Logistic regression (Deep-Dive)



Learning goals

- Derive the gradient of the logistic regression
- Derive the Hessian of the logistic regression
- Show that the logistic regression is a convex problem

LOGISTIC REGRESSION: RISK PROBLEM

Given $n \in \mathbb{N}$ observations $(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{X} \times \mathcal{Y}$ with $\mathcal{X} = \mathbb{R}^d, \mathcal{Y} = \{0, 1\}$ we want to minimize the following risk

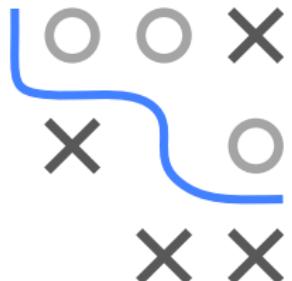
$$\mathcal{R}_{\text{emp}} = -\sum_{i=1}^n y^{(i)} \log(\pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})) + (1 - y^{(i)}) \log(1 - \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta}))$$

with respect to $\boldsymbol{\theta}$ where the probabilistic classifier

$$\pi(\mathbf{x}^{(i)} | \boldsymbol{\theta}) = s(f(\mathbf{x}^{(i)} | \boldsymbol{\theta})),$$

the sigmoid function $s(f) = \frac{1}{1 + \exp(-f)}$ and the score $f(\mathbf{x}^{(i)} | \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}$.

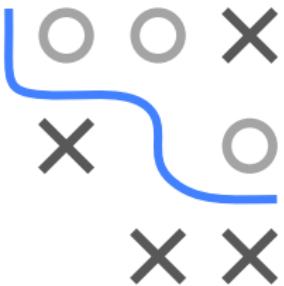
NB: Note that $\frac{\partial}{\partial f} s(f) = s(f)(1 - s(f))$ and $\frac{\partial f(\mathbf{x}^{(i)} | \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = (\mathbf{x}^{(i)})^\top$.



LOGISTIC REGRESSION: GRADIENT

We find the gradient of logistic regression with the chain rule, s.t.,

$$\begin{aligned}\frac{\partial}{\partial \theta} \mathcal{R}_{\text{emp}} &= - \sum_{i=1}^n \frac{\partial}{\partial \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})} y^{(i)} \log(\pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})) \frac{\partial \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \\ &\quad \frac{\partial}{\partial \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})} (1 - y^{(i)}) \log(1 - \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})) \frac{\partial \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ &= - \sum_{i=1}^n \frac{y^{(i)}}{\pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})} \frac{\partial \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} - \frac{1 - y^{(i)}}{1 - \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})} \frac{\partial \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ &= - \sum_{i=1}^n \left(\frac{y^{(i)}}{\pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})} - \frac{1 - y^{(i)}}{1 - \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})} \right) \frac{\partial s(f(\mathbf{x}^{(i)} | \boldsymbol{\theta}))}{\partial f(\mathbf{x}^{(i)} | \boldsymbol{\theta})} \frac{\partial f(\mathbf{x}^{(i)} | \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ &= - \sum_{i=1}^n \left(y^{(i)}(1 - \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})) - (1 - y^{(i)})\pi(\mathbf{x}^{(i)} | \boldsymbol{\theta}) \right) (\mathbf{x}^{(i)})^\top.\end{aligned}$$



LOGISTIC REGRESSION: GRADIENT

$$= \sum_{i=1}^n \left(\pi(\mathbf{x}^{(i)} | \boldsymbol{\theta}) - y^{(i)} \right) \left(\mathbf{x}^{(i)} \right)^\top$$

$$= (\pi(\mathbf{X} | \boldsymbol{\theta}) - \mathbf{y})^\top \mathbf{X}$$



where $\mathbf{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)})^\top \in \mathbb{R}^{n \times d}$, $\mathbf{y} = (y^{(1)}, \dots, y^{(n)})^\top$, $\pi(\mathbf{X} | \boldsymbol{\theta}) = (\pi(\mathbf{x}^{(1)} | \boldsymbol{\theta}), \dots, \pi(\mathbf{x}^{(n)} | \boldsymbol{\theta}))^\top \in \mathbb{R}^n$.

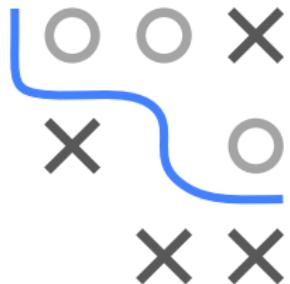
\implies The gradient $\nabla_{\theta} \mathcal{R}_{\text{emp}} = \left(\frac{\partial}{\partial \theta} \mathcal{R}_{\text{emp}} \right)^{\top} = \mathbf{x}^{\top} (\pi(\mathbf{x} | \theta) - \mathbf{y})$

This formula can now be used in gradient descent and its friends.

LOGISTIC REGRESSION: HESSIAN

We find the Hessian via differentiation, s.t.,

$$\begin{aligned}\nabla_{\theta}^2 \mathcal{R}_{\text{emp}} &= \frac{\partial^2}{\partial \theta^T \partial \theta} \mathcal{R}_{\text{emp}} = \frac{\partial}{\partial \theta^T} \sum_{i=1}^n \left(\pi(\mathbf{x}^{(i)} | \boldsymbol{\theta}) - y^{(i)} \right) (\mathbf{x}^{(i)})^T \\ &= \sum_{i=1}^n \mathbf{x}^{(i)} \left(\pi(\mathbf{x}^{(i)} | \boldsymbol{\theta}) (1 - \pi(\mathbf{x}^{(i)} | \boldsymbol{\theta})) \right) (\mathbf{x}^{(i)})^T \\ &= \mathbf{X}^T \mathbf{D} \mathbf{X}\end{aligned}$$



where $\mathbf{D} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with diagonal

$$\left(\pi(\mathbf{x}^{(1)} | \boldsymbol{\theta}) (1 - \pi(\mathbf{x}^{(1)} | \boldsymbol{\theta})), \dots, \pi(\mathbf{x}^{(n)} | \boldsymbol{\theta}) (1 - \pi(\mathbf{x}^{(n)} | \boldsymbol{\theta})) \right).$$

Can now be used in Newton-Raphson and other 2nd order optimizers.

LOGISTIC REGRESSION: CONVEXITY

Finally, we check that logistic regression is a convex problem:

We define the diagonal matrix $\bar{\mathbf{D}} \in \mathbb{R}^{n \times n}$ with diagonal

$$\left(\sqrt{\pi(\mathbf{x}^{(1)} | \theta)(1 - \pi(\mathbf{x}^{(1)} | \theta))}, \dots, \sqrt{\pi(\mathbf{x}^{(n)} | \theta)(1 - \pi(\mathbf{x}^{(n)} | \theta))} \right)$$

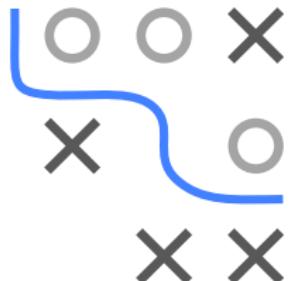
which is possible since π maps into $(0, 1)$.

With this, we get for any $\mathbf{w} \in \mathbb{R}^d$ that

$$\mathbf{w}^\top \nabla_\theta^2 \mathcal{R}_{\text{emp}} \mathbf{w} = \mathbf{w}^\top \mathbf{X}^\top \bar{\mathbf{D}}^\top \bar{\mathbf{D}} \mathbf{X} \mathbf{w} = (\bar{\mathbf{D}} \mathbf{X} \mathbf{w})^\top \bar{\mathbf{D}} \mathbf{X} \mathbf{w} = \|\bar{\mathbf{D}} \mathbf{X} \mathbf{w}\|_2^2 \geq 0$$

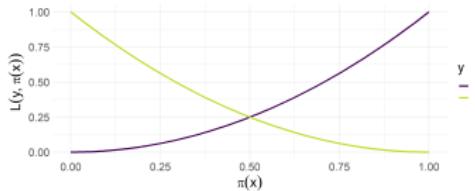
since obviously $\mathbf{D} = \bar{\mathbf{D}}^\top \bar{\mathbf{D}}$.

$\Rightarrow \nabla_\theta^2 \mathcal{R}_{\text{emp}}$ is positive semi-definite $\Rightarrow \mathcal{R}_{\text{emp}}$ is convex.



Introduction to Machine Learning

Brier Score



Learning goals

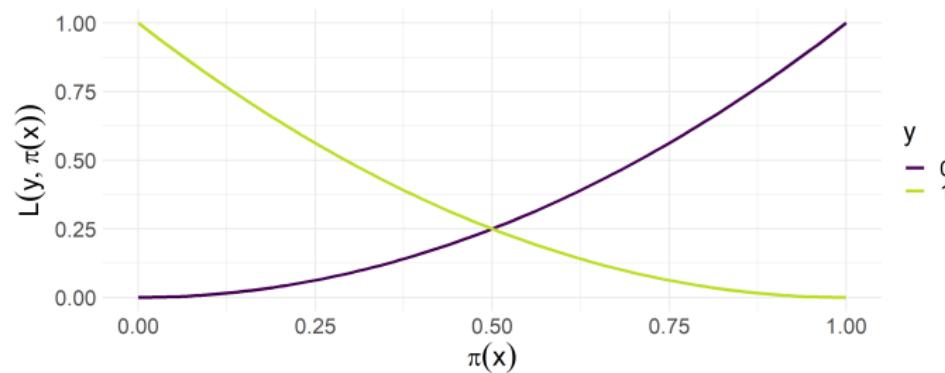
- Know the Brier score
- Derive the risk minimizer
- Derive the optimal constant model
- Understand the connection between Brier score and Gini splitting



BRIER SCORE

The binary Brier score is defined on probabilities $\pi(\mathbf{x}) \in [0, 1]$ and 0-1-encoded labels $y \in \{0, 1\}$ and measures their squared distance (L2 loss on probabilities).

$$L(y, \pi(\mathbf{x})) = (\pi(\mathbf{x}) - y)^2$$



BRIER SCORE: RISK MINIMIZER

The risk minimizer for the (binary) Brier score is

$$\pi^*(\mathbf{x}) = \eta(\mathbf{x}) = \mathbb{P}(y | \mathbf{x} = \mathbf{x}),$$

which means that the Brier score will reach its minimum if the prediction equals the “true” probability of the outcome.

The risk minimizer for the multiclass Brier score is

$$\pi^*(\mathbf{x}) = \mathbb{P}(y = k | \mathbf{x} = \mathbf{x}).$$

Proof: We only show the proof for the binary case. We need to minimize

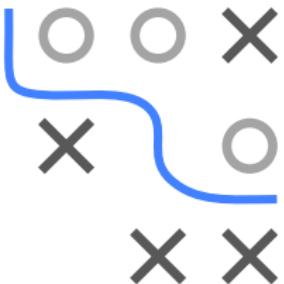
$$\mathbb{E}_{\mathbf{x}} [L(1, \pi(\mathbf{x})) \cdot \eta(\mathbf{x}) + L(0, \pi(\mathbf{x})) \cdot (1 - \eta(\mathbf{x}))],$$



BRIER SCORE: RISK MINIMIZER

which we do point-wise for every \mathbf{x} . We plug in the Brier score

$$\begin{aligned} & \arg \min_c L(1, c)\eta(\mathbf{x}) + L(0, c)(1 - \eta(\mathbf{x})) \\ = & \arg \min_c (c - 1)^2\eta(\mathbf{x}) + c^2(1 - \eta(\mathbf{x})) \\ = & \arg \min_c (c - \eta(\mathbf{x}))^2. \end{aligned}$$

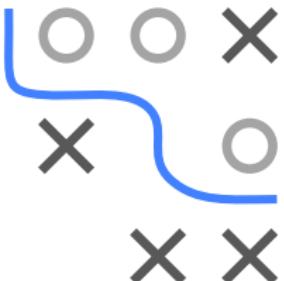


The expression is minimal if $c = \eta(\mathbf{x}) = \mathbb{P}(y = 1 \mid \mathbf{x} = \mathbf{x})$.

BRIER SCORE: OPTIMAL CONSTANT MODEL

The optimal constant probability model $\pi(\mathbf{x}) = \theta$ w.r.t. the Brier score for labels from $\mathcal{Y} = \{0, 1\}$ is:

$$\begin{aligned}\min_{\theta} \mathcal{R}_{\text{emp}}(\theta) &= \min_{\theta} \sum_{i=1}^n (y^{(i)} - \theta)^2 \\ \Leftrightarrow \frac{\partial \mathcal{R}_{\text{emp}}(\theta)}{\partial \theta} &= -2 \cdot \sum_{i=1}^n (y^{(i)} - \theta) = 0 \\ \hat{\theta} &= \frac{1}{n} \sum_{i=1}^n y^{(i)}.\end{aligned}$$



This is the fraction of class-1 observations in the observed data.
(This also directly follows from our L_2 proof for regression).

Similarly, for the multiclass brier score the optimal constant is

$$\hat{\theta}_k = \frac{1}{n} \sum_{i=1}^n [y = k].$$

BRIER SCORE MINIMIZATION = GINI SPLITTING

When fitting a tree we minimize the risk within each node \mathcal{N} by risk minimization and predict the optimal constant. Another approach that is common in literature is to minimize the average node impurity $\text{Imp}(\mathcal{N})$.

Claim: Gini splitting $\text{Imp}(\mathcal{N}) = \sum_{k=1}^g \pi_k^{(\mathcal{N})} (1 - \pi_k^{(\mathcal{N})})$ is equivalent to the Brier score minimization.

Note that $\pi_k^{(\mathcal{N})} := \frac{1}{n_{\mathcal{N}}} \sum_{(\mathbf{x}, y) \in \mathcal{N}} [y = k]$

Proof: We show that the risk related to a subset of observations $\mathcal{N} \subseteq \mathcal{D}$ fulfills

$$\mathcal{R}(\mathcal{N}) = n_{\mathcal{N}} \text{Imp}(\mathcal{N}),$$

where Imp is the Gini impurity and $\mathcal{R}(\mathcal{N})$ is calculated w.r.t. the (multiclass) Brier score

$$L(y, \pi(\mathbf{x})) = \sum_{k=1}^g ([y = k] - \pi_k(\mathbf{x}))^2.$$



BRIER SCORE MINIMIZATION = GINI SPLITTING

$$\mathcal{R}(\mathcal{N}) = \sum_{(\mathbf{x}, y) \in \mathcal{N}} \sum_{k=1}^g ([y = k] - \pi_k(\mathbf{x}))^2 = \sum_{k=1}^g \sum_{(\mathbf{x}, y) \in \mathcal{N}} \left([y = k] - \frac{n_{\mathcal{N}, k}}{n_{\mathcal{N}}} \right)^2,$$

by plugging in the optimal constant prediction w.r.t. the Brier score ($n_{\mathcal{N}, k}$ is defined as the number of class k observations in node \mathcal{N}):

$$\hat{\pi}_k(\mathbf{x}) = \pi_k^{(\mathcal{N})} = \frac{1}{n_{\mathcal{N}}} \sum_{(\mathbf{x}, y) \in \mathcal{N}} [y = k] = \frac{n_{\mathcal{N}, k}}{n_{\mathcal{N}}}.$$

We split the inner sum and further simplify the expression

$$\begin{aligned} &= \sum_{k=1}^g \left(\sum_{(\mathbf{x}, y) \in \mathcal{N}: y=k} \left(1 - \frac{n_{\mathcal{N}, k}}{n_{\mathcal{N}}} \right)^2 + \sum_{(\mathbf{x}, y) \in \mathcal{N}: y \neq k} \left(0 - \frac{n_{\mathcal{N}, k}}{n_{\mathcal{N}}} \right)^2 \right) \\ &= \sum_{k=1}^g n_{\mathcal{N}, k} \left(1 - \frac{n_{\mathcal{N}, k}}{n_{\mathcal{N}}} \right)^2 + (n_{\mathcal{N}} - n_{\mathcal{N}, k}) \left(\frac{n_{\mathcal{N}, k}}{n_{\mathcal{N}}} \right)^2, \end{aligned}$$

since for $n_{\mathcal{N}, k}$ observations the condition $y = k$ is met, and for the remaining $(n_{\mathcal{N}} - n_{\mathcal{N}, k})$ observations it is not.



BRIER SCORE MINIMIZATION = GINI SPLITTING

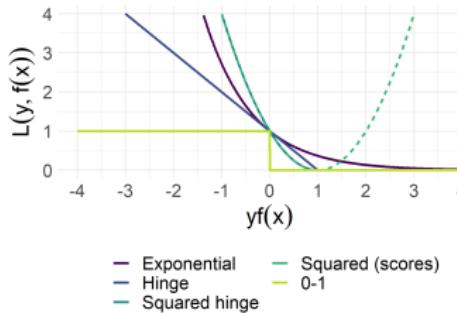
We further simplify the expression to

$$\begin{aligned}\mathcal{R}(\mathcal{N}) &= \sum_{k=1}^g n_{\mathcal{N},k} \left(\frac{n_{\mathcal{N}} - n_{\mathcal{N},k}}{n_{\mathcal{N}}} \right)^2 + (n_{\mathcal{N}} - n_{\mathcal{N},k}) \left(\frac{n_{\mathcal{N},k}}{n_{\mathcal{N}}} \right)^2 \\ &= \sum_{k=1}^g \frac{n_{\mathcal{N},k}}{n_{\mathcal{N}}} \frac{n_{\mathcal{N}} - n_{\mathcal{N},k}}{n_{\mathcal{N}}} (n_{\mathcal{N}} - n_{\mathcal{N},k} + n_{\mathcal{N},k}) \\ &= n_{\mathcal{N}} \sum_{k=1}^g \pi_k^{(\mathcal{N})} \cdot \left(1 - \pi_k^{(\mathcal{N})} \right) = n_{\mathcal{N}} \text{Imp}(\mathcal{N}).\end{aligned}$$



Introduction to Machine Learning

Advanced Classification Losses



Learning goals

- Know the (squared) hinge loss
- Know the L_2 loss defined on scores
- Know the exponential loss
- Know the AUC loss

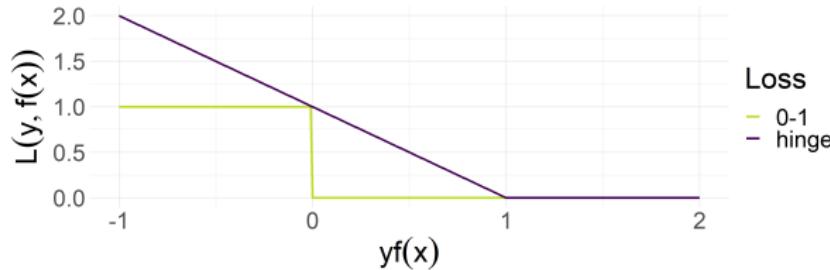


HINGE LOSS

- The intuitive appeal of the 0-1-loss is set off by its analytical properties ill-suited to direct optimization.
- The **hinge loss** is a continuous relaxation that acts as a convex upper bound on the 0-1-loss (for $y \in \{-1, +1\}$):

$$L(y, f(\mathbf{x})) = \max\{0, 1 - yf(\mathbf{x})\}.$$

- Note that the hinge loss only equals zero for a margin ≥ 1 , encouraging confident (correct) predictions.
- It resembles a door hinge, hence the name:

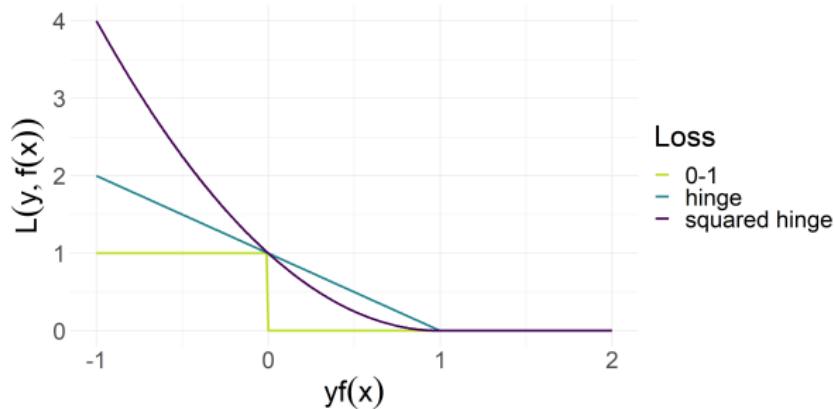


SQUARED HINGE LOSS

- We can also specify a **squared** version for the hinge loss:

$$L(y, f(\mathbf{x})) = \max\{0, (1 - yf(\mathbf{x}))\}^2.$$

- The *L2* form punishes margins $yf(\mathbf{x}) \in (0, 1)$ less severely but puts a high penalty on more confidently wrong predictions.
- Therefore, it is smoother yet more outlier-sensitive than the non-squared hinge loss.

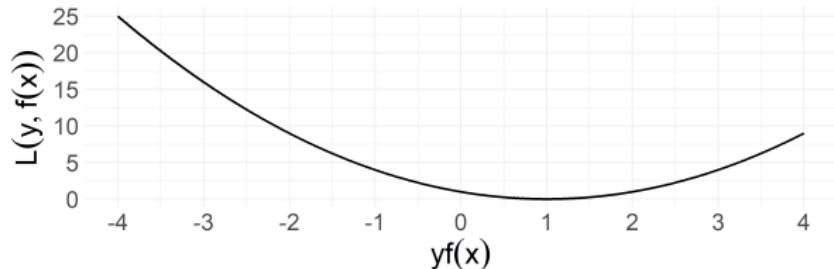
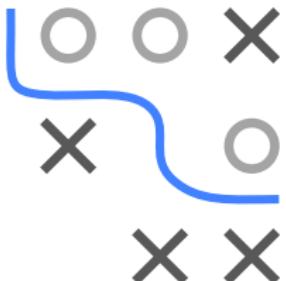


SQUARED LOSS ON SCORES

- Analogous to the Brier score defined on probabilities we can specify a **squared loss on classification scores** (again, $y \in \{-1, +1\}$, using that $y^2 \equiv 1$):

$$\begin{aligned} L(y, f(\mathbf{x})) &= (y - f(\mathbf{x}))^2 = y^2 - 2yf(\mathbf{x}) + (f(\mathbf{x}))^2 = \\ &= 1 - 2yf(\mathbf{x}) + (yf(\mathbf{x}))^2 = (1 - yf(\mathbf{x}))^2 \end{aligned}$$

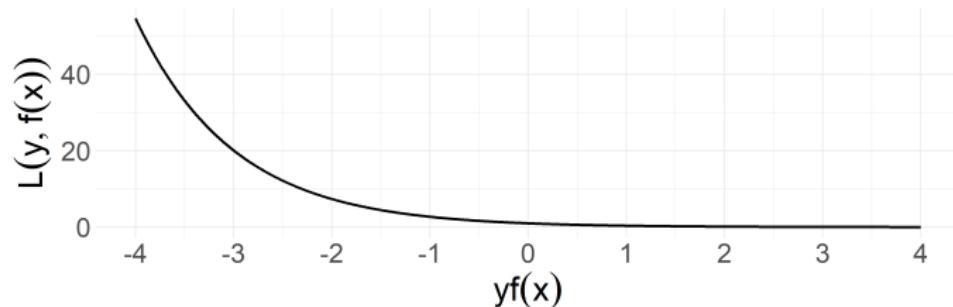
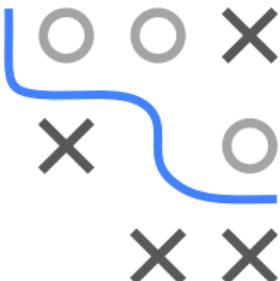
- This loss behaves just like the squared hinge loss for $yf(\mathbf{x}) < 1$, but is zero only for $yf(\mathbf{x}) = 1$ and actually increases again for larger margins (which is in general not desirable!)



CLASSIFICATION LOSSES: EXPONENTIAL LOSS

Another possible choice for a (binary) loss function that is a smooth approximation to the 0-1-loss is the **exponential loss**:

- $L(y, f(\mathbf{x})) = \exp(-yf(\mathbf{x}))$, used in AdaBoost.
- Convex, differentiable (thus easier to optimize than 0-1-loss).
- The loss increases exponentially for wrong predictions with high confidence; if the prediction is right with a small confidence only, there, loss is still positive.
- No closed-form analytic solution to (empirical) risk minimization.



CLASSIFICATION LOSSES: AUC-LOSS

- Often AUC is used as an evaluation criterion for binary classifiers.
- Let $y \in \{-1, +1\}$ with n_- negative and n_+ positive samples.
- The AUC can then be defined as

$$AUC = \frac{1}{n_+} \frac{1}{n_-} \sum_{i:y^{(i)}=1} \sum_{j:y^{(j)}=-1} [f^{(i)} > f^{(j)}]$$

- This is not differentiable w.r.t f due to $[f^{(i)} > f^{(j)}]$.
- But the indicator function can be approximated by the distribution function of the triangular distribution on $[-1, 1]$ with mean 0.
- However, direct optimization of the AUC is numerically more difficult, and might not work as well as using a common loss and tuning for AUC in practice.

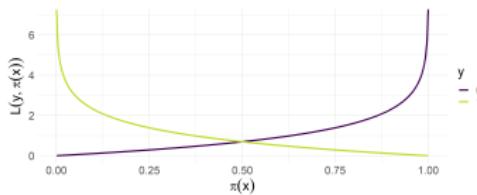


Introduction to Machine Learning

Optimal constant model for the empirical log loss risk (Deep-Dive)



Learning goals

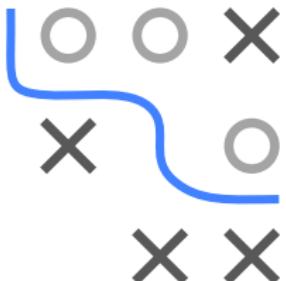


- Derive the optimal constant model for the binary empirical log loss risk
- Derive the optimal constant model for the empirical multiclass log loss risk

BINARY LOG LOSS: EMP. RISK MINIMIZER

Given $n \in \mathbb{N}$ observations $y^{(1)}, \dots, y^{(n)} \in \mathcal{Y} = \{0, 1\}$ we want to determine the optimal constant model for the empirical log loss risk.

$$\arg \min_{\theta \in (0,1)} \mathcal{R}_{\text{emp}} = \arg \min_{\theta \in (0,1)} - \sum_{i=1}^n y^{(i)} \log(\theta) + (1 - y^{(i)}) \log(1 - \theta).$$



BINARY LOG LOSS: EMP. RISK MINIMIZER

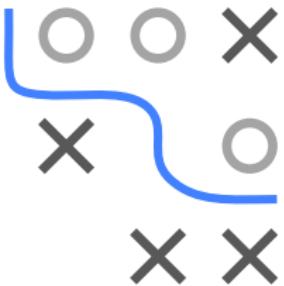
The minimizer can be found by setting the derivative to zero, i.e.,

$$\frac{d}{d\theta} \mathcal{R}_{\text{emp}} = - \sum_{i=1}^n \frac{y^{(i)}}{\theta} - \frac{1-y^{(i)}}{1-\theta} \stackrel{!}{=} 0$$

$$\iff - \sum_{i=1}^n y^{(i)}(1-\theta) - \theta(1-y^{(i)}) \stackrel{!}{=} 0$$

$$\iff - \sum_{i=1}^n (y^{(i)} - \theta) \stackrel{!}{=} 0$$

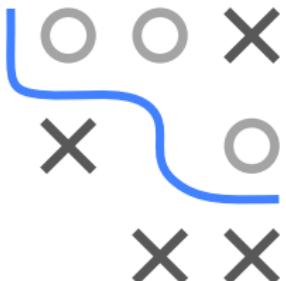
$$\Rightarrow \hat{\theta} = \frac{1}{n} \sum_{i=1}^n y^{(i)} \in (0, 1) \checkmark (\text{assuming both labels occur}).$$



MULTICLASS LOG LOSS: EMP. RISK MINIMIZER

Given $n \in \mathbb{N}$ observations $y^{(1)}, \dots, y^{(n)} \in \mathcal{Y} = \{1, \dots, g\}$ with $g \in \mathbb{N}_{>1}$ we want to determine the optimal constant model $\theta \in (0, 1)^g$ for the empirical log loss risk

$$\begin{aligned}\arg \min_{\theta \in (0,1)^g} \mathcal{R}_{\text{emp}} &= \arg \min_{\theta \in (0,1)^g} - \sum_{i=1}^n \sum_{j=1}^g \mathbb{1}_{\{y^{(i)}=j\}} \log(\theta_j) \\ \text{s.t. } &\sum_{j=1}^g \theta_j = 1.\end{aligned}$$

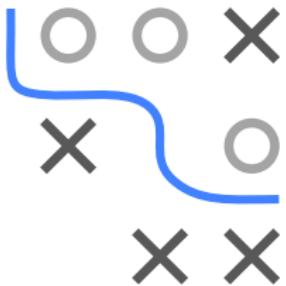


We can solve this constrained optimization problem by plugging the constraint into the risk (we could also use Lagrange multipliers), i.e., we replace θ_g (this is an arbitrary choice) such that $\theta_g = 1 - \sum_{j=1}^{g-1} \theta_j$.

MULTICLASS LOG LOSS: EMP. RISK MINIMIZER

With this, we find the equivalent optimization problem

$$\begin{aligned}\arg \min_{\theta \in (0,1)^{g-1}} \mathcal{R}_{\text{emp}} &= \arg \min_{\theta \in (0,1)^{g-1}} - \sum_{i=1}^n \sum_{j=1}^{g-1} \mathbb{1}_{\{y^{(i)}=j\}} \log(\theta_j) \\ &\quad + \mathbb{1}_{\{y^{(i)}=g\}} \log \left(1 - \sum_{j=1}^{g-1} \theta_j \right) \\ \text{s.t. } & \sum_{j=1}^{g-1} \theta_j < 1.\end{aligned}$$



For $j \in \{1, \dots, g-1\}$, the j -th partial derivative of our objective

$$\begin{aligned}\frac{\partial}{\partial \theta_j} \mathcal{R}_{\text{emp}} &= - \sum_{i=1}^n \mathbb{1}_{\{y^{(i)}=j\}} \frac{1}{\theta_j} - \mathbb{1}_{\{y^{(i)}=g\}} \frac{1}{1 - \sum_{j=1}^{g-1} \theta_j} \\ &= -\frac{n_j}{\theta_j} + \frac{n_g}{\theta_g}\end{aligned}$$

where n_k with $k \in \{1, \dots, g\}$ is the number of label k in y and we assume that $n_k > 0$.

MULTICLASS LOG LOSS: EMP. RISK MINIMIZER

For the minimizer, it must hold for $j \in \{1, \dots, g - 1\}$ that

$$\begin{aligned} \frac{\partial}{\partial \theta_j} \mathcal{R}_{\text{emp}} &\stackrel{!}{=} 0 \\ \iff -n_j \theta_g + n_g \theta_j &\stackrel{!}{=} 0 \\ \Rightarrow \sum_{j=1}^{g-1} (-n_j \theta_g + n_g \theta_j) &\stackrel{!}{=} 0 \\ \iff -(n - n_g) \theta_g + n_g (1 - \theta_g) &\stackrel{!}{=} 0 \\ \iff -n \theta_g + n_g &\stackrel{!}{=} 0 \\ \Rightarrow \hat{\theta}_g = \frac{n_g}{n} &\in (0, 1) \checkmark \\ \Rightarrow \forall j \in \{1, \dots, g - 1\} : \quad \hat{\theta}_j = \frac{\hat{\theta}_g n_j}{n_g} = \frac{n_j}{n} &\in (0, 1) \checkmark. \\ \left(\Rightarrow \sum_{j=1}^{g-1} \hat{\theta}_j = 1 - \hat{\theta}_g = 1 - \frac{n_g}{n} < 1 \checkmark \right) \end{aligned}$$



CONVEXITY

Finally, we check that we indeed found a minimizer by showing that \mathcal{R}_{emp} is convex for the multiclass case (binary is a special case of this):

The Hessian of \mathcal{R}_{emp}

$$\nabla_{\theta}^2 \mathcal{R}_{\text{emp}} = \begin{pmatrix} \frac{n_1}{\theta_1^2} & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{n_{g-1}}{\theta_{g-1}^2} \end{pmatrix}$$



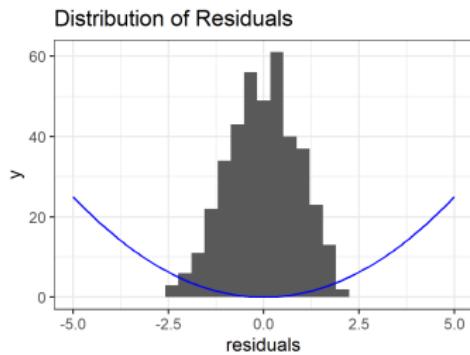
is positive definite since all its eigenvalues

$$\lambda_j = \frac{n_j}{\theta_j^2} > 0 \quad \forall j \in \{1, \dots, g-1\}.$$

From this, it follows that \mathcal{R}_{emp} is (strictly) convex.

Introduction to Machine Learning

Maximum Likelihood Estimation vs. Empirical Risk Minimization



Learning goals

- Understand the connection between maximum likelihood and risk minimization
- Learn the correspondence between a Gaussian error distribution and the L2 loss

MAXIMUM LIKELIHOOD

Let's consider regression from a maximum likelihood perspective.

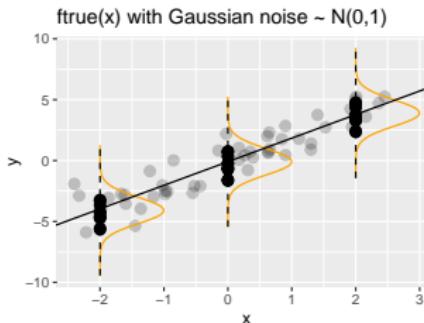
Assume:

$$y | \mathbf{x} \sim p(y | \mathbf{x}, \theta)$$



Common case: true underlying relationship f_{true} with additive noise:

$$y = f_{\text{true}}(\mathbf{x}) + \epsilon$$



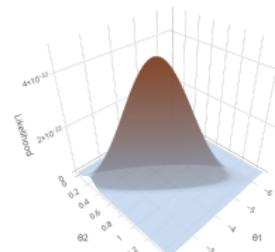
where f_{true} has params θ and ϵ a RV that follows some distribution \mathbb{P}_ϵ , with $\mathbb{E}[\epsilon] = 0$. Also, assume $\epsilon \perp\!\!\!\perp \mathbf{x}$.

MAXIMUM LIKELIHOOD

From a statistics / maximum-likelihood perspective, we assume (or we pretend) we know the underlying distribution $p(y | \mathbf{x}, \theta)$.

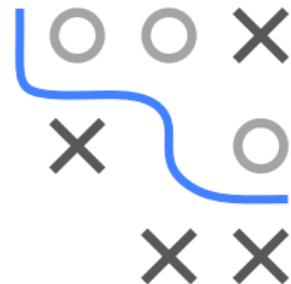
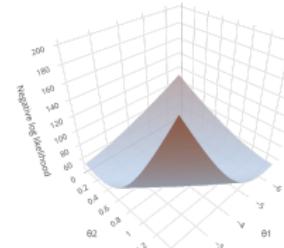
- Then, given i.i.d data $\mathcal{D} = ((\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}))$ from \mathbb{P}_{xy} the maximum-likelihood principle is to maximize the **likelihood**

$$\mathcal{L}(\theta) = \prod_{i=1}^n p(y^{(i)} | \mathbf{x}^{(i)}, \theta)$$



or equivalently to minimize the **negative log-likelihood**

$$-\ell(\theta) = - \sum_{i=1}^n \log p(y^{(i)} | \mathbf{x}^{(i)}, \theta)$$



MAXIMUM LIKELIHOOD

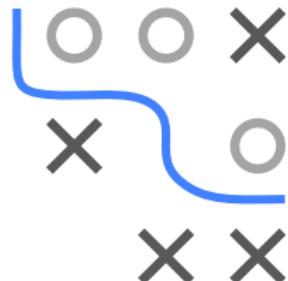
From an ML perspective we assume our hypothesis space corresponds to the space of the (parameterized) f_{true} .

- Simply define neg. log-likelihood as **loss function**

$$L(y, f(\mathbf{x} \mid \theta)) := -\log p(y \mid \mathbf{x}, \theta)$$

- Then, maximum-likelihood = ERM

$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \theta\right)\right)$$



- NB: When we are only interested in the minimizer, we can ignore multiplicative or additive constants.
- We use \propto as “proportional up to multiplicative and additive constants”

GAUSSIAN ERRORS - L2-LOSS

Assume $y = f_{\text{true}}(\mathbf{x}) + \epsilon$ with additive Gaussian errors, i.e.
 $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$. Then

$$y \mid \mathbf{x} \sim N(f_{\text{true}}(\mathbf{x}), \sigma^2)$$

The likelihood is then

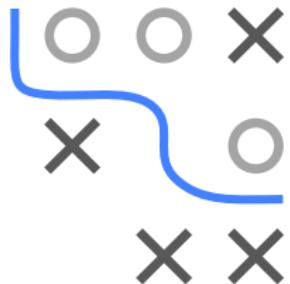


$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= \prod_{i=1}^n p\left(y^{(i)} \mid f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right), \sigma^2\right) \\ &\propto \prod_{i=1}^n \exp\left(-\frac{1}{2\sigma^2} \left(y^{(i)} - f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)^2\right)\end{aligned}$$

GAUSSIAN ERRORS - L2-LOSS

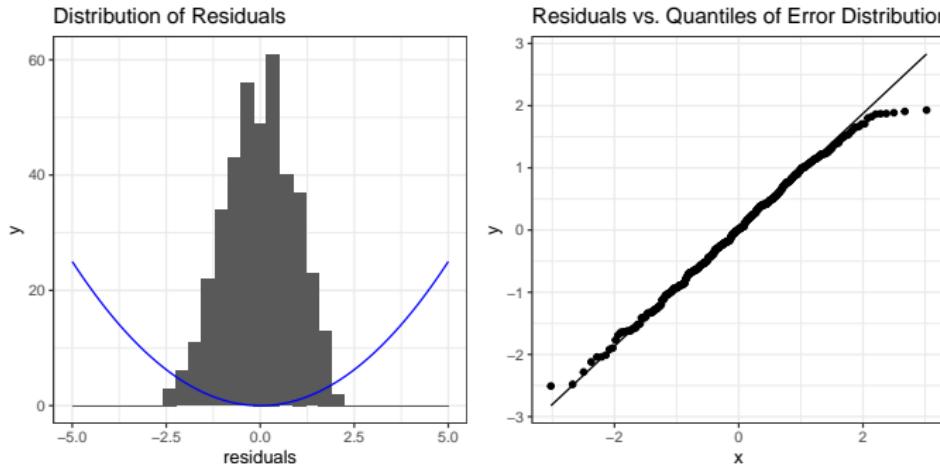
Easy to see: minimizing neg. negative log-likelihood with Gaussian errors is the same as ERM with $L2$ -loss:

$$\begin{aligned}-\ell(\theta) &= -\log (\mathcal{L}(\theta)) \\&= -\log \left(\prod_{i=1}^n \exp \left(-\frac{1}{2\sigma^2} \left(y^{(i)} - f(\mathbf{x}^{(i)} | \theta) \right)^2 \right) \right) \\&\propto \sum_{i=1}^n \left(y^{(i)} - f(\mathbf{x}^{(i)} | \theta) \right)^2\end{aligned}$$



GAUSSIAN ERRORS - L2-LOSS

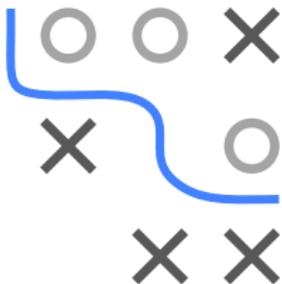
- We simulate data $y \mid \mathbf{x} \sim \mathcal{N}(f_{\text{true}}(\mathbf{x}), 1)$ with $f_{\text{true}} = 0.2 \cdot \mathbf{x}$
- Let's plot empirical errors as histogram, after fitting our model with $L2$ -loss
- Q-Q-plot compares empirical residuals vs. theoretical quantiles of Gaussian



DISTRIBUTIONS AND LOSSES

- For every error distribution \mathbb{P}_ϵ we can derive an equivalent loss function, which leads to the same point estimator for the parameter vector θ as maximum-likelihood. Formally,
 - $\hat{\theta} \in \arg \max_{\theta} \mathcal{L}(\theta) \implies \hat{\theta} \in \arg \min_{\theta} -\log(\mathcal{L}(\theta))$
- **But:** The other way around does not always work: We cannot derive a corresponding pdf or error distribution for every loss function – the Hinge loss is one prominent example, for which some probabilistic interpretation is still possible however, see

► Sollich, 1999.



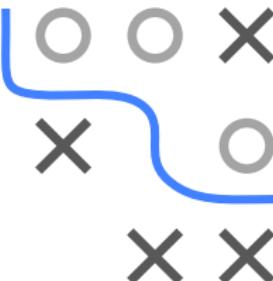
DISTRIBUTIONS AND LOSSES

When does the reverse direction hold?

- If we can write the loss as $L(y, f(\mathbf{x})) = L(y - f(\mathbf{x})) = L(r)$ for $r \in \mathbb{R}$, then minimizing $L(y - f(\mathbf{x}))$ is equivalent to maximizing a conditional log-likelihood $\log(p(y - f(\mathbf{x}|\theta))$) if
 - $\log(p(r))$ is affine trafo of L (undoing the \propto):

$$\log(p(r)) = a - bL(r), \quad a \in \mathbb{R}, b > 0$$

- p is a pdf (non-negative and integrates to one)

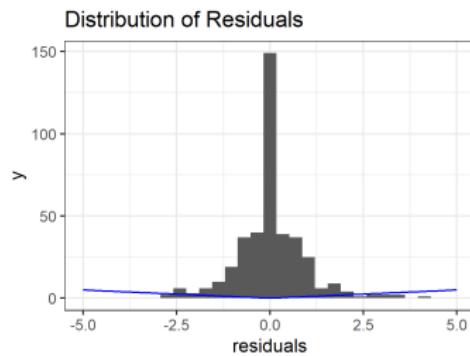


Thus, a loss L corresponds to MLE under *some* distribution if there exist $a \in \mathbb{R}$, $b > 0$ such that

$$\int_{\mathbb{R}} \exp(a - bL(r)) dr = 1$$

Introduction to Machine Learning

Maximum Likelihood Estimation vs. Empirical Risk Minimization



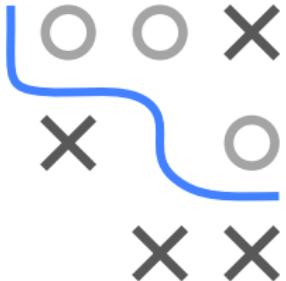
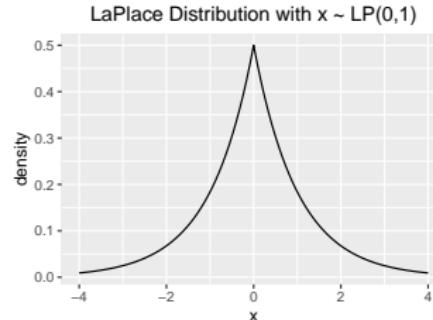
Learning goals

- Correspondence between Laplace errors and L1 loss
- Correspondence between Bernoulli targets and the Bernoulli / log loss

LAPLACE ERRORS - L1-LOSS

Let's consider Laplacian errors ϵ now, with density:

$$\frac{1}{2\sigma} \exp\left(-\frac{|\epsilon|}{\sigma}\right), \sigma > 0.$$



Then

$$y = f_{\text{true}}(\mathbf{x}) + \epsilon$$

also follows Laplace distrib. with mean $f(\mathbf{x}^{(i)} | \theta)$ and scale σ .

LAPLACE ERRORS - L1-LOSS

The likelihood is then

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= \prod_{i=1}^n p\left(y^{(i)} \mid f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right), \sigma\right) \\ &\propto \exp\left(-\frac{1}{\sigma} \sum_{i=1}^n \left|y^{(i)} - f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right|\right).\end{aligned}$$



The negative log-likelihood is

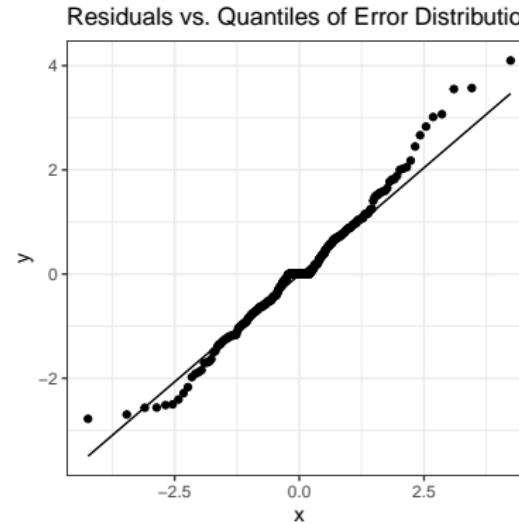
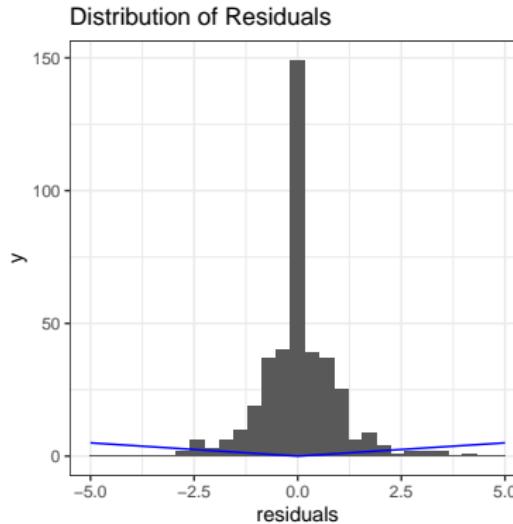
$$-\ell(\boldsymbol{\theta}) \propto \sum_{i=1}^n \left|y^{(i)} - f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right|.$$

MLE for Laplacian errors = ERM with L1-loss.

- Some losses correspond to more complex or less known error densities, like the Huber loss ► Meyer, 2021
- Huber density is (unsurprisingly) a hybrid of Gaussian and Laplace

LAPLACE ERRORS - L1-LOSS

- We simulate data $y \mid \mathbf{x} \sim \text{Laplacian}(f_{\text{true}}(\mathbf{x}), 1)$ with $f_{\text{true}} = 0.2 \cdot \mathbf{x}$.
- We can plot the empirical error distribution, i.e. the distribution of the residuals after fitting a regression model w.r.t. $L1$ -loss.
- With the help of a Q-Q-plot we can compare the empirical residuals vs. the theoretical quantiles of a Laplacian distribution.



MAXIMUM LIKELIHOOD IN CLASSIFICATION

Let us assume the outputs y to be Bernoulli-distributed, i.e.

$$y \mid \mathbf{x} \sim \text{Ber}(\pi_{\text{true}}(\mathbf{x})).$$

The negative log likelihood is



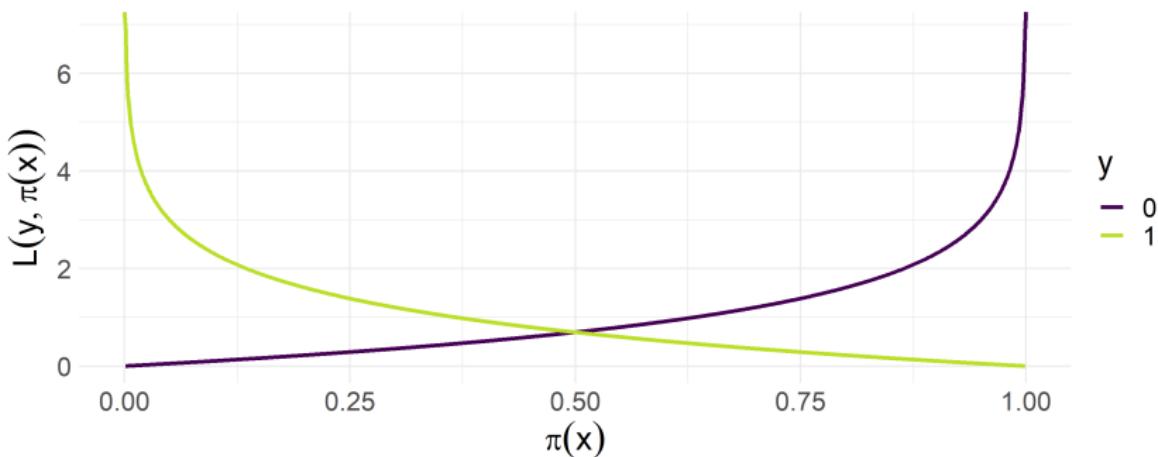
$$\begin{aligned}-\ell(\boldsymbol{\theta}) &= -\sum_{i=1}^n \log p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta}) \\&= -\sum_{i=1}^n \log \left[\pi(\mathbf{x}^{(i)})^{y^{(i)}} \cdot (1 - \pi(\mathbf{x}^{(i)}))^{(1-y^{(i)})} \right] \\&= \sum_{i=1}^n -y^{(i)} \log[\pi(\mathbf{x}^{(i)})] - (1 - y^{(i)}) \log[1 - \pi(\mathbf{x}^{(i)})].\end{aligned}$$

MAXIMUM LIKELIHOOD IN CLASSIFICATION

This gives rise to the following loss function

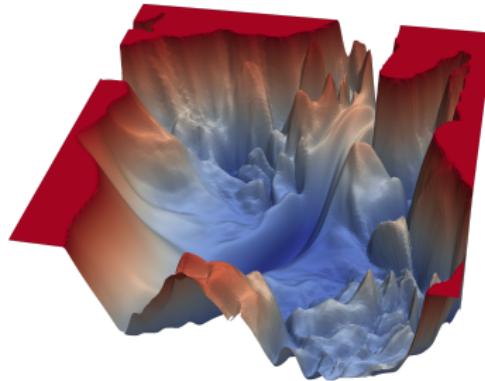
$$L(y, \pi(\mathbf{x})) = -y \log(\pi(\mathbf{x})) - (1 - y) \log(1 - \pi(\mathbf{x})), \quad y \in \{0, 1\}$$

which we introduced as **Bernoulli** loss.



Introduction to Machine Learning

Properties of Loss Functions



Learning goals

- Statistical properties
- Robustness
- Numerical properties
- Some fundamental terminology



THE ROLE OF LOSS FUNCTIONS

Why should we care about the choice of the loss function $L(y, f(\mathbf{x}))$?

- **Statistical** properties: choice of loss implies statistical assumptions about the distribution of $y \mid \mathbf{x} = \mathbf{x}$ (see *maximum likelihood estimation vs. empirical risk minimization*).
- **Robustness** properties: some loss functions are more robust towards outliers than others.
- **Numerical** properties: the computational complexity of

$$\arg \min_{\theta \in \Theta} \mathcal{R}_{\text{emp}}(\theta)$$

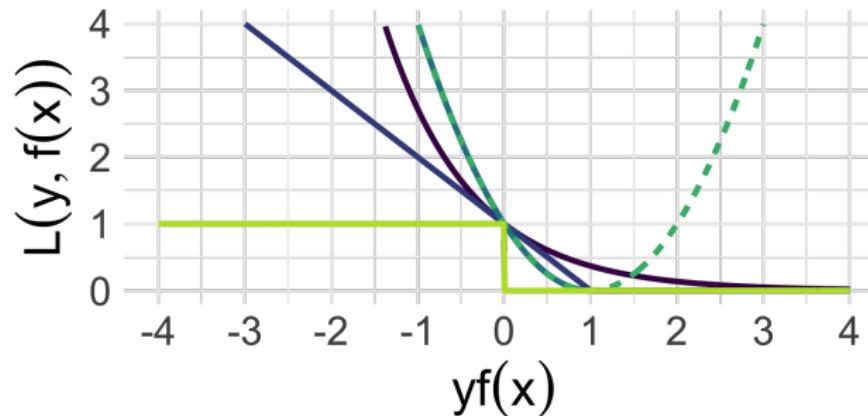
is influenced by the choice of the loss function.



SOME BASIC TERMINOLOGY

Classification losses are usually expressed in terms of the **margin**:

$$\nu := y \cdot f(\mathbf{x}).$$

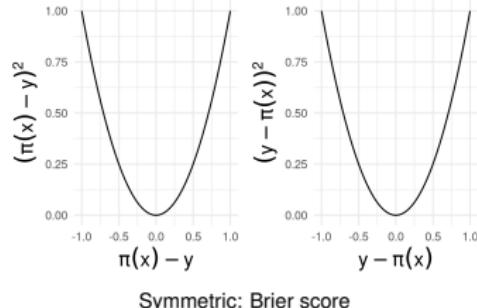
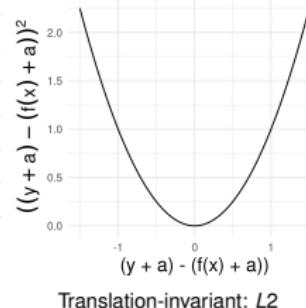
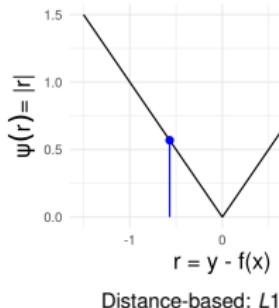
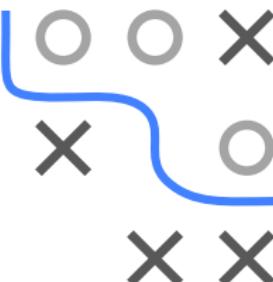


- Exponential
- Hinge
- Squared hinge
- Squared (scores)
- 0-1



SOME BASIC TERMINOLOGY

- Regression losses often only depend on the **residuals** $r := y - f(\mathbf{x})$.
- Losses are called **symmetric** if $L(y, f(\mathbf{x})) = L(f(\mathbf{x}), y)$.
- A loss is **translation-invariant** if $L(y + a, f(\mathbf{x}) + a) = L(y, f(\mathbf{x}))$, $a \in \mathbb{R}$.
- A loss is called **distance-based** if
 - it can be written in terms of the residual, i.e., $L(y, f(\mathbf{x})) = \psi(r)$ for some $\psi : \mathbb{R} \rightarrow \mathbb{R}$, and
 - $\psi(r) = 0 \Leftrightarrow r = 0$.

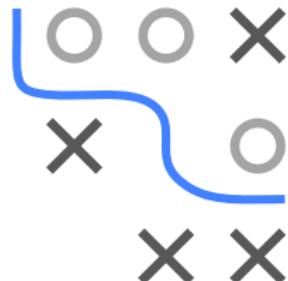
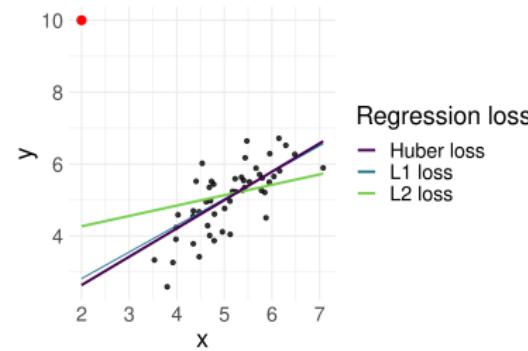
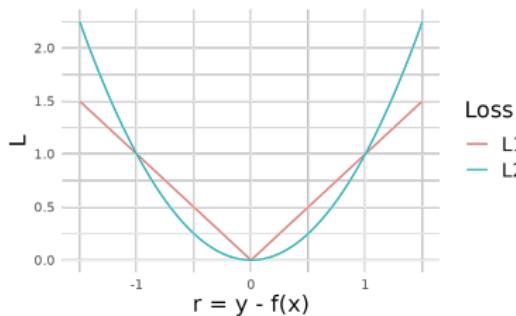


ROBUSTNESS

Outliers (in y) have large residuals $r = y - f(\mathbf{x})$. Some losses are more affected by large residuals than others. If loss goes up superlinearly (e.g. L2) it is not robust, linear (L1) or even sublinear losses are more robust.

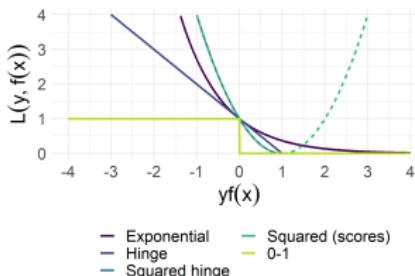
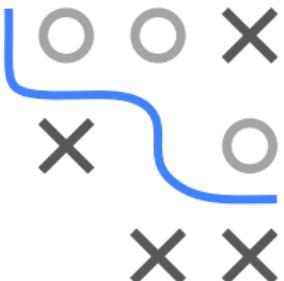
$y - \hat{f}(\mathbf{x})$	$L1$	$L2$	Huber ($\epsilon = 5$)
1	1	1	0.5
5	5	25	12.5
10	10	100	37.5
50	50	2500	237.5

As a consequence, a model is less influenced by outliers than by "inliers" if the loss is **robust**.
Outliers e.g. strongly influence L2.



NUMERICAL PROPERTIES: SMOOTHNESS

- **Smoothness** of a function is a property measured by the number of continuous derivatives.
- Derivative-based optimization requires smoothness of the risk $\mathcal{R}_{\text{emp}}(\theta)$
 - If loss is unsmooth, we might have to use derivative-free optimization (or worse, in case of 0-1)
 - Smoothness of $\mathcal{R}_{\text{emp}}(\theta)$ not only depends on L , but also requires smoothness of $f(\mathbf{x})$!



Squared loss, exponential loss and squared hinge loss are continuously differentiable.
Hinge loss is continuous but not differentiable.
0-1 loss is not even continuous.

NUMERICAL PROPERTIES: CONVEXITY

- A function $\mathcal{R}_{\text{emp}}(\theta)$ is convex if

$$\mathcal{R}_{\text{emp}}\left(t \cdot \theta + (1 - t) \cdot \tilde{\theta}\right) \leq t \cdot \mathcal{R}_{\text{emp}}(\theta) + (1 - t) \cdot \mathcal{R}_{\text{emp}}(\tilde{\theta})$$

$$\forall t \in [0, 1], \theta, \tilde{\theta} \in \Theta$$

(strictly convex if the above holds with strict inequality).

- In optimization, convex problems have a number of convenient properties. E.g., all local minima are global.

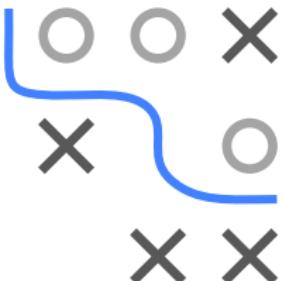
→ strictly convex function has at most **one** global min
(uniqueness).

- For $\mathcal{R}_{\text{emp}} \in \mathcal{C}^2$, \mathcal{R}_{emp} is convex iff Hessian $\nabla^2 \mathcal{R}_{\text{emp}}(\theta)$ is psd.

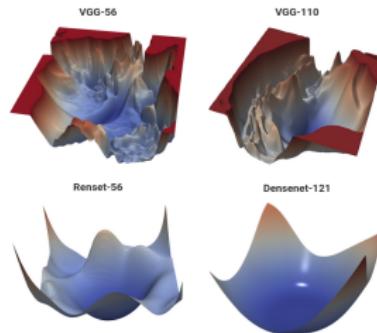


NUMERICAL PROPERTIES: CONVEXITY

- Convexity of $\mathcal{R}_{\text{emp}}(\theta)$ depends both on convexity of $L(\cdot)$ (given in most cases) and $f(\mathbf{x} \mid \theta)$ (often problematic).
- If we model our data using an exponential family distribution, we always get convex losses
 - For $f(\mathbf{x} \mid \theta)$ linear in θ , linear/logistic/softmax/poisson/... regression are convex problems (all GLMs)!



Li et al., 2018: *Visualizing the Loss Landscape of Neural Nets*. The problem on the bottom right is convex, the others are not (note that very high-dimensional surfaces are coerced into 3D here).

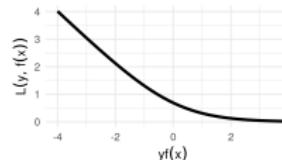


NUMERICAL PROPERTIES: CONVERGENCE

In case of **complete separation**, optimization might even fail entirely, e.g.:

- Margin-based loss that is strictly monotonically decreasing in $y \cdot f$, e.g., **Bernoulli loss**:

$$L(y, f(\mathbf{x})) = \log(1 + \exp(-yf(\mathbf{x})))$$



- f linear in θ , e.g., **logistic regression** with $f(\mathbf{x} | \theta) = \theta^\top \mathbf{x}$
- Data perfectly separable by our learner, so we can find θ :

$$y^{(i)} f(\mathbf{x}^{(i)} | \theta) = y^{(i)} \theta^\top \mathbf{x}^{(i)} > 0 \quad \forall \mathbf{x}^{(i)}$$

- Can now construct a strictly better θ

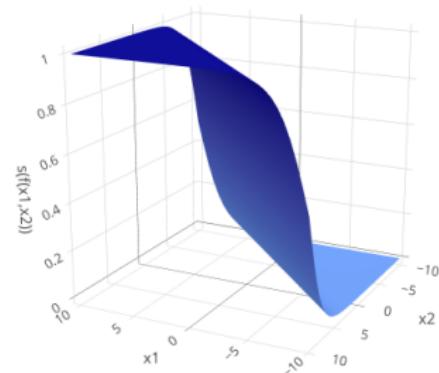
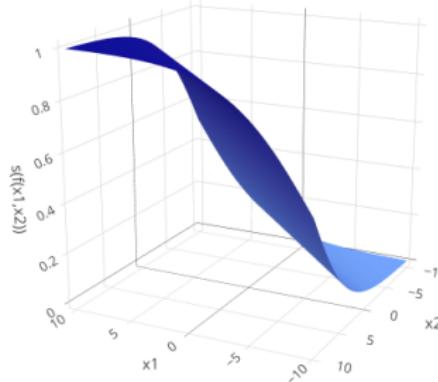
$$\mathcal{R}_{\text{emp}}(2 \cdot \theta) = \sum_{i=1}^n L\left(2y^{(i)} \theta^\top \mathbf{x}^{(i)}\right) < \mathcal{R}_{\text{emp}}(\theta)$$

- As $\|\theta\|$ increases, sum strictly decreases, as argument of L is strictly larger
- We can iterate that, so there is no local (or global) optimum, and no numerical procedure can converge



NUMERICAL PROPERTIES: CONVERGENCE

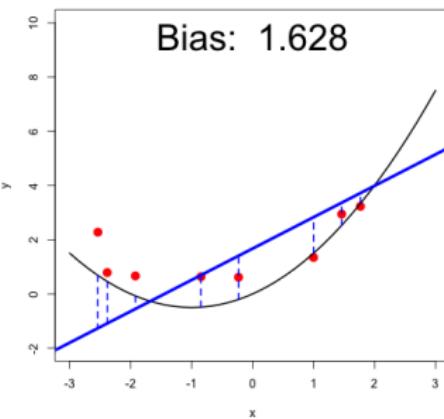
- Geometrically, this translates to an ever steeper slope of the logistic/softmax function, i.e., increasingly sharp discrimination:



- In practice, data are seldomly linearly separable and misclassified examples act as counterweights to increasing parameter values.
- Besides, we can use **regularization** to encourage convergence to robust solutions.

Introduction to Machine Learning

Advanced Risk Minimization: Bias-Variance Decomposition



Learning goals

- Understand how to decompose the generalization error of a learner into
 - bias of the learner
 - variance of the learner
 - inherent noise in the data

BIAS-VARIANCE DECOMPOSITION

Let us take a closer look at the generalization error of a learning algorithm \mathcal{I}_L . This is the expected error of an induced model $\hat{f}_{\mathcal{D}_n}$, on training sets of size n , when applied to a fresh, random test observation.

$$GE_n(\mathcal{I}_L) = \mathbb{E}_{\mathcal{D}_n \sim \mathbb{P}_{xy}^n, (\mathbf{x}, y) \sim \mathbb{P}_{xy}} \left(L(y, \hat{f}_{\mathcal{D}_n}(\mathbf{x})) \right) = \mathbb{E}_{\mathcal{D}_n, xy} \left(L(y, \hat{f}_{\mathcal{D}_n}(\mathbf{x})) \right)$$

We therefore need to take the expectation over all training sets of size n , as well as the independent test observation.

We assume that the data is generated by

$$y = f_{\text{true}}(\mathbf{x}) + \epsilon,$$

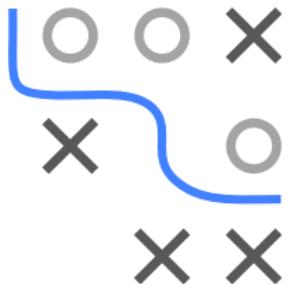
with zero-mean homoskedastic error $\epsilon \sim (0, \sigma^2)$ independent of \mathbf{x} .



BIAS-VARIANCE DECOMPOSITION

$$GE_n(\mathcal{I}_L) =$$

$$\underbrace{\sigma^2}_{\text{Variance of the data}} + \underbrace{\mathbb{E}_{xy} \left[\text{Var}_{\mathcal{D}_n} \left(\hat{f}_{\mathcal{D}_n}(\mathbf{x}) \mid \mathbf{x}, y \right) \right]}_{\text{Variance of learner at } (\mathbf{x},y)} + \underbrace{\mathbb{E}_{xy} \left[\left(f_{\text{true}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}_n} \left(\hat{f}_{\mathcal{D}_n}(\mathbf{x}) \right) \right)^2 \mid \mathbf{x}, y \right]}_{\text{Squared bias of learner at } (\mathbf{x},y)}$$



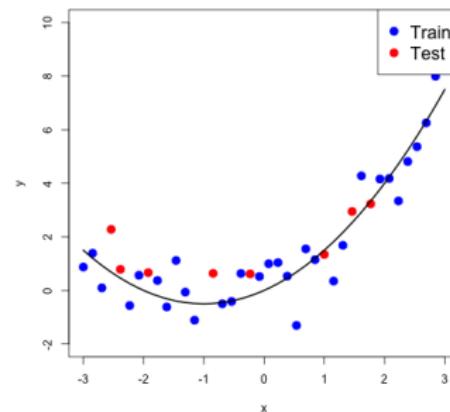
- ➊ The first term expresses the variance of the data. This is pure **noise** in the data. Also called Bayes, intrinsic or irreducible error. No matter what we do, we will never get below this error.
- ➋ The second term expresses, on average, how much $\hat{f}_{\mathcal{D}_n}(\mathbf{x})$ fluctuates around test points if we vary the training data. Expresses also the learner's tendency to learn random things irrespective of the real signal (overfitting).
- ➌ The third term says how much we are "off" on average at test locations (underfitting). Models with high capacity typically have low **bias** and *vice versa*.

BIAS-VARIANCE DECOMPOSITION

Illustration: Let us consider the following example. We will generate a dataset using the following model :

$$y = x + \frac{x^2}{2} + \epsilon, \quad \epsilon \sim N(0, 1)$$

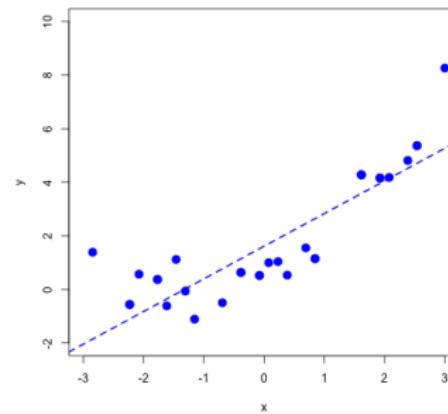
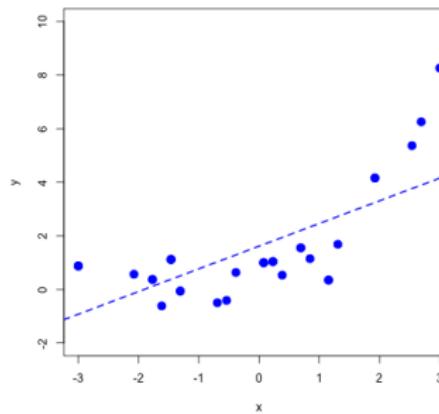
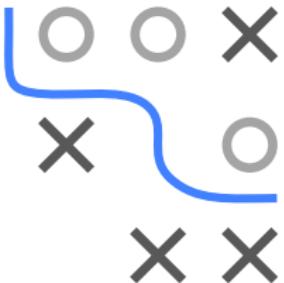
The data is then split into a training set and a test set.



BIAS-VARIANCE DECOMPOSITION

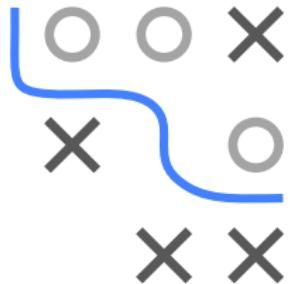
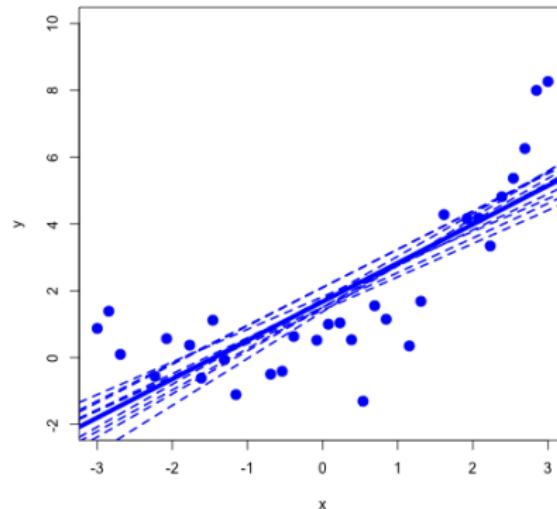
To obtain estimates for the bias and variance, we will train several models by sampling with replacement from the training data. This is commonly known as **bootstrapping**.

First, we train several (low capacity) linear models (polynomial of degree $d = 1$).



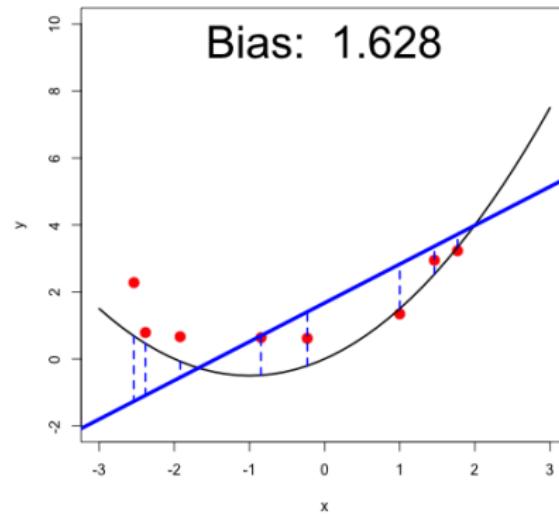
BIAS-VARIANCE DECOMPOSITION

By creating several models, we obtain the average model over different samples of the training dataset.



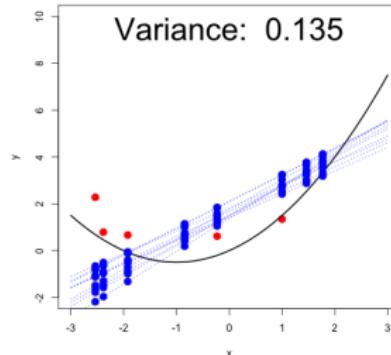
BIAS-VARIANCE DECOMPOSITION

We can now estimate the (squared) bias, by computing the average squared difference between the average model and the true model, at the test point locations.



BIAS-VARIANCE DECOMPOSITION

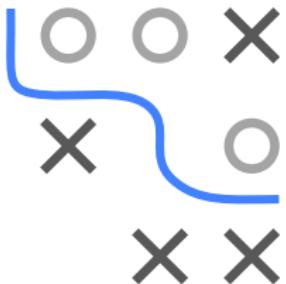
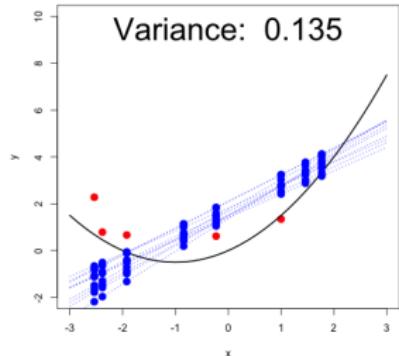
We compute the average variance of the predictions of the models we trained at the test point locations.



$$GE_n(\mathcal{I}_L) \approx 1 + 1.628 + 0.135 = 2.763$$

- The biggest component of the generalization error is the bias.
- Computing the MSE in the usual way for each model, via L2 loss, and then averaging over models gives rise to nearly the same value, as expected

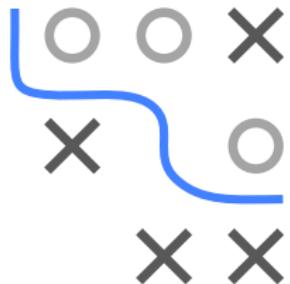
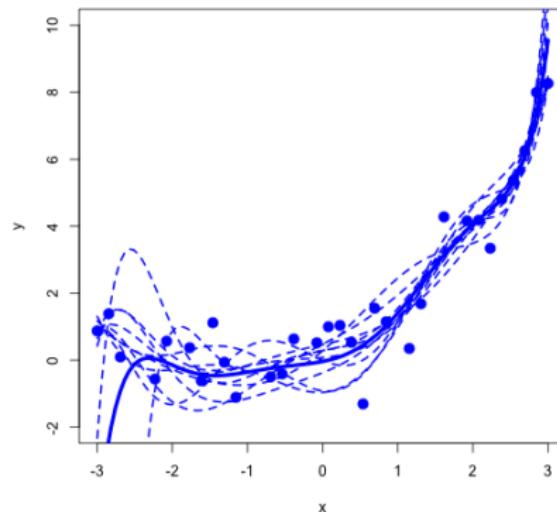
BIAS-VARIANCE DECOMPOSITION



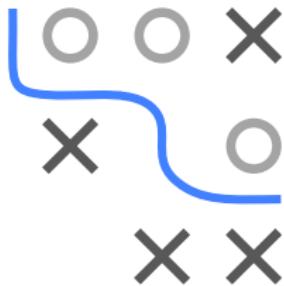
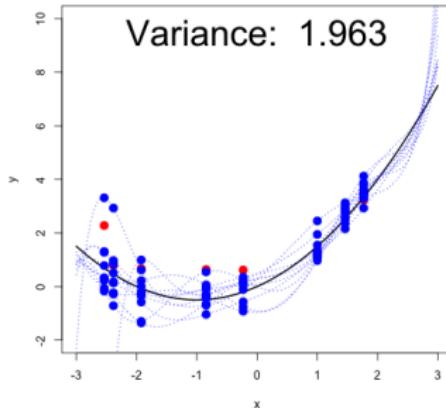
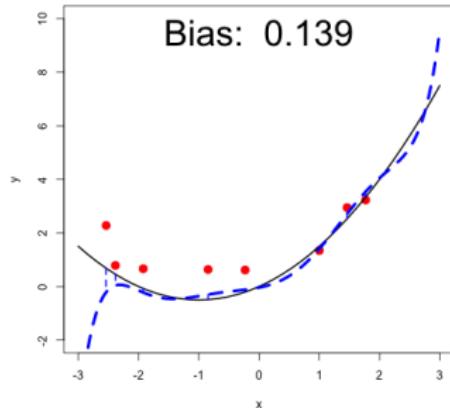
- We can now check whether this alternative computation of the GE is correct
- So, we simply compute the MSE in the standard fashion for each model
- So for each model we compute the L2 loss at each data point, then average
- Then we average these MSEs over all models
- Result = 2.72, would be closer if we average over more models and test points

BIAS-VARIANCE DECOMPOSITION

We will repeat the same procedure, but use a high-degree polynomial ($d = 7$) with more capacity.



BIAS-VARIANCE DECOMPOSITION

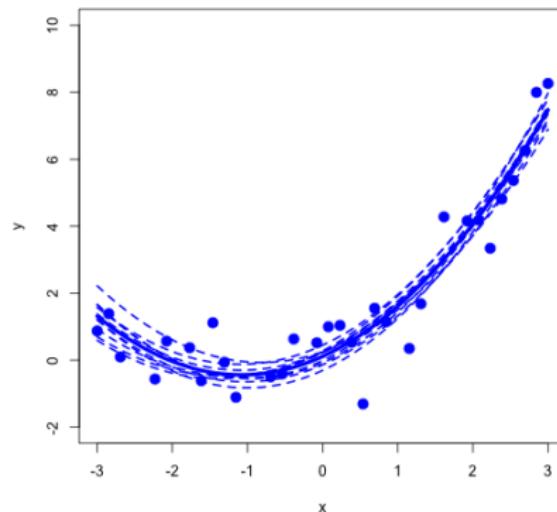


$$GE_n(\mathcal{I}_L) \approx 1 + 0.139 + 1.963 = 3.102$$

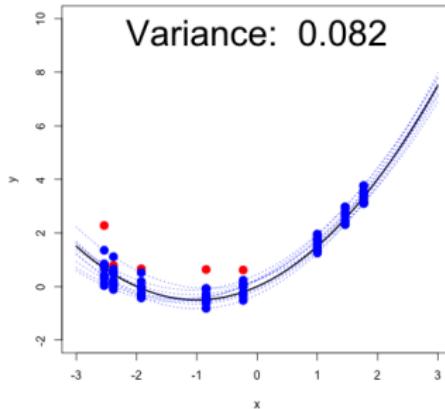
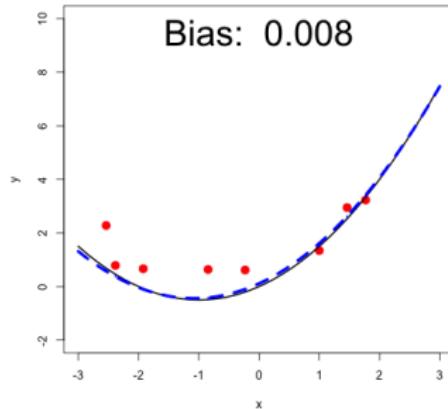
- The generalization error is higher than before
- Even though the bias is lower, the variance of the learner is higher.

BIAS-VARIANCE DECOMPOSITION

What happens if we use a model with the same complexity as the true model (quadratic polynomial)?



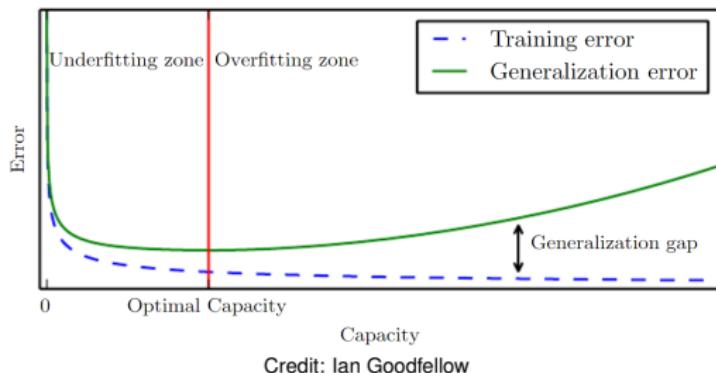
BIAS-VARIANCE DECOMPOSITION



$$GE_n(\mathcal{I}_L) \approx 1 + 0.008 + 0.082 = 1.091$$

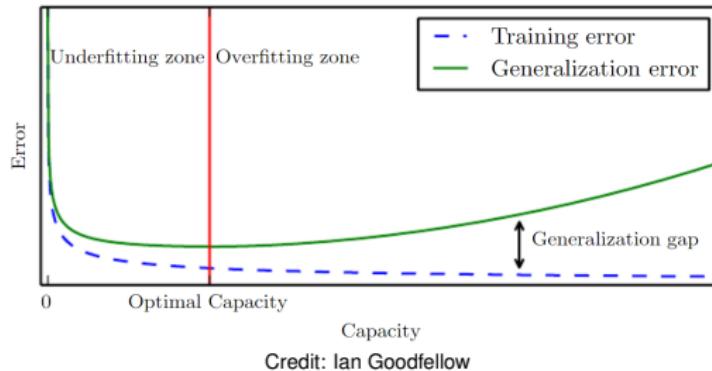
- The generalization error is the lowest at this complexity.
- The variance of the data acts as a lower bound.

CAPACITY AND OVERFITTING



- The performance of a learner depends on its ability to
 - ➊ **fit** the training data well
 - ➋ **generalize** to new data
- Failure of the first point is called **underfitting**
- Failure of the second item is called **overfitting**

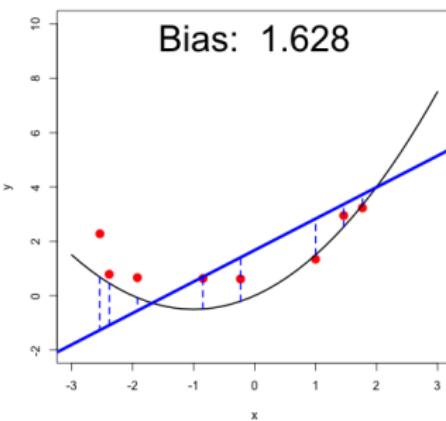
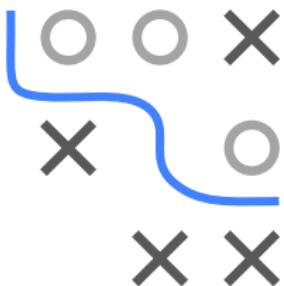
CAPACITY AND OVERFITTING



- The tendency of a model to underfit/overfit is a function of its capacity, determined by the type of hypotheses it can learn.
- Usually, low bias means high capacity, which in turn means a higher chance of overfitting
- Low-bias models usually have also higher variance
- For such models, regularization (we discuss later) is essential
- Even for correctly specified models, the generalization error is lower-bounded by the irreducible noise σ^2

Introduction to Machine Learning

Bias-Variance Decomposition (Deep-Dive)



Learning goals

- Understand how to decompose the generalization error of a learner into
 - Bias of the learner
 - Variance of the learner
 - Inherent noise in the data

BIAS-VARIANCE DECOMPOSITION

Let us take a closer look at the generalization error of a learning algorithm \mathcal{I}_L . This is the expected error of an induced model $\hat{f}_{\mathcal{D}_n}$, on training sets of size n , when applied to a fresh, random test observation.

$$GE_n(\mathcal{I}_L) = \mathbb{E}_{\mathcal{D}_n \sim \mathbb{P}_{xy}^n, (\mathbf{x}, y) \sim \mathbb{P}_{xy}} \left(L(y, \hat{f}_{\mathcal{D}_n}(\mathbf{x})) \right) = \mathbb{E}_{\mathcal{D}_n, xy} \left(L(y, \hat{f}_{\mathcal{D}_n}(\mathbf{x})) \right)$$

We therefore need to take the expectation over all training sets of size n , as well as the independent test observation.

We assume that the data is generated by

$$y = f_{\text{true}}(\mathbf{x}) + \epsilon,$$

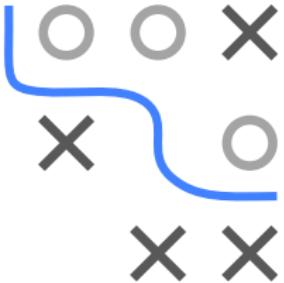
with zero-mean homoskedastic error $\epsilon \sim (0, \sigma^2)$ independent of \mathbf{x} .



BIAS-VARIANCE DECOMPOSITION

By plugging in the $L2$ loss $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$ we get

$$\begin{aligned} GE_n(\mathcal{I}_L) &= \mathbb{E}_{\mathcal{D}_n, xy} \left(L \left(y, \hat{f}_{\mathcal{D}_n}(\mathbf{x}) \right) \right) = \mathbb{E}_{\mathcal{D}_n, xy} \left((y - \hat{f}_{\mathcal{D}_n}(\mathbf{x}))^2 \right) \\ &\stackrel{\text{LIE}}{=} \mathbb{E}_{xy} \left[\underbrace{\mathbb{E}_{\mathcal{D}_n} \left((y - \hat{f}_{\mathcal{D}_n}(\mathbf{x}))^2 \mid \mathbf{x}, y \right)}_{(*)} \right] \end{aligned}$$



Let us consider the error $(*)$ conditioned on one fixed test observation (\mathbf{x}, y) first. (We omit the $\mid \mathbf{x}, y$ for better readability for now.)

$$\begin{aligned} (*) &= \mathbb{E}_{\mathcal{D}_n} \left((y - \hat{f}_{\mathcal{D}_n}(\mathbf{x}))^2 \right) \\ &= \underbrace{\mathbb{E}_{\mathcal{D}_n} (y^2)}_{=y^2} + \underbrace{\mathbb{E}_{\mathcal{D}_n} (\hat{f}_{\mathcal{D}_n}(\mathbf{x})^2)}_{(1)} - 2 \underbrace{\mathbb{E}_{\mathcal{D}_n} (y \hat{f}_{\mathcal{D}_n}(\mathbf{x}))}_{(2)} \end{aligned}$$

by using the linearity of the expectation.

BIAS-VARIANCE DECOMPOSITION

$$(*) = \mathbb{E}_{\mathcal{D}_n} \left((y - \hat{f}_{\mathcal{D}_n}(\mathbf{x}))^2 \right) = y^2 + \underbrace{\mathbb{E}_{\mathcal{D}_n} (\hat{f}_{\mathcal{D}_n}(\mathbf{x})^2)}_{(1)} - 2 \underbrace{\mathbb{E}_{\mathcal{D}_n} (y \hat{f}_{\mathcal{D}_n}(\mathbf{x}))}_{(2)} =$$

Using that $\mathbb{E}(z^2) = \text{Var}(z) + \mathbb{E}^2(z)$, we see that

$$= y^2 + \text{Var}_{\mathcal{D}_n} (\hat{f}_{\mathcal{D}_n}(\mathbf{x})) + \mathbb{E}_{\mathcal{D}_n}^2 (\hat{f}_{\mathcal{D}_n}(\mathbf{x})) - 2y \mathbb{E}_{\mathcal{D}_n} (\hat{f}_{\mathcal{D}_n}(\mathbf{x}))$$

Plug in the definition of y

$$= f_{\text{true}}(\mathbf{x})^2 + 2\epsilon f_{\text{true}}(\mathbf{x}) + \epsilon^2 + \text{Var}_{\mathcal{D}_n} (\hat{f}_{\mathcal{D}_n}(\mathbf{x})) + \mathbb{E}_{\mathcal{D}_n}^2 (\hat{f}_{\mathcal{D}_n}(\mathbf{x})) - 2(f_{\text{true}}(\mathbf{x}) + \epsilon) \mathbb{E}_{\mathcal{D}_n} (\hat{f}_{\mathcal{D}_n}(\mathbf{x}))$$

Reorder terms and use the binomial formula

$$= \epsilon^2 + \text{Var}_{\mathcal{D}_n} (\hat{f}_{\mathcal{D}_n}(\mathbf{x})) + \left(f_{\text{true}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}_n} (\hat{f}_{\mathcal{D}_n}(\mathbf{x})) \right)^2 + 2\epsilon \left(f_{\text{true}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}_n} (\hat{f}_{\mathcal{D}_n}(\mathbf{x})) \right)$$



BIAS-VARIANCE DECOMPOSITION

$$(*) = \epsilon^2 + \text{Var}_{\mathcal{D}_n}(\hat{f}_{\mathcal{D}_n}(\mathbf{x})) + \left(f_{\text{true}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}_n}(\hat{f}_{\mathcal{D}_n}(\mathbf{x})) \right)^2 + 2\epsilon \left(f_{\text{true}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}_n}(\hat{f}_{\mathcal{D}_n}(\mathbf{x})) \right)$$

Let us come back to the generalization error by taking the expectation over all fresh test observations $(\mathbf{x}, y) \sim \mathbb{P}_{xy}$:

$$\begin{aligned} GE_n(\mathcal{I}_L) &= \underbrace{\sigma^2}_{\text{Variance of the data}} + \underbrace{\mathbb{E}_{xy} \left[\text{Var}_{\mathcal{D}_n}(\hat{f}_{\mathcal{D}_n}(\mathbf{x}) | \mathbf{x}, y) \right]}_{\text{Variance of learner at } (\mathbf{x}, y)} \\ &+ \underbrace{\mathbb{E}_{xy} \left[\left(\left(f_{\text{true}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}_n}(\hat{f}_{\mathcal{D}_n}(\mathbf{x})) \right)^2 | \mathbf{x}, y \right) \right]}_{\text{Squared bias of learner at } (\mathbf{x}, y)} + \underbrace{0}_{\text{As } \epsilon \text{ is zero-mean and independent}} \end{aligned}$$



INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

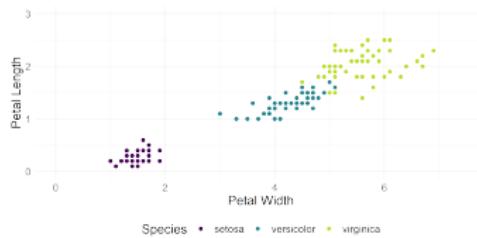
Boosting

Gaussian Processes



Introduction to Machine Learning

Multiclass Classification and Losses



Learning goals

- Know what multiclass means and which types of classifiers exist
- Know the MC 0-1-loss
- Know the MC brier score
- Know the MC logarithmic loss

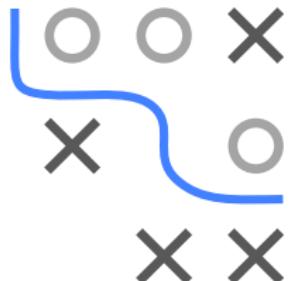
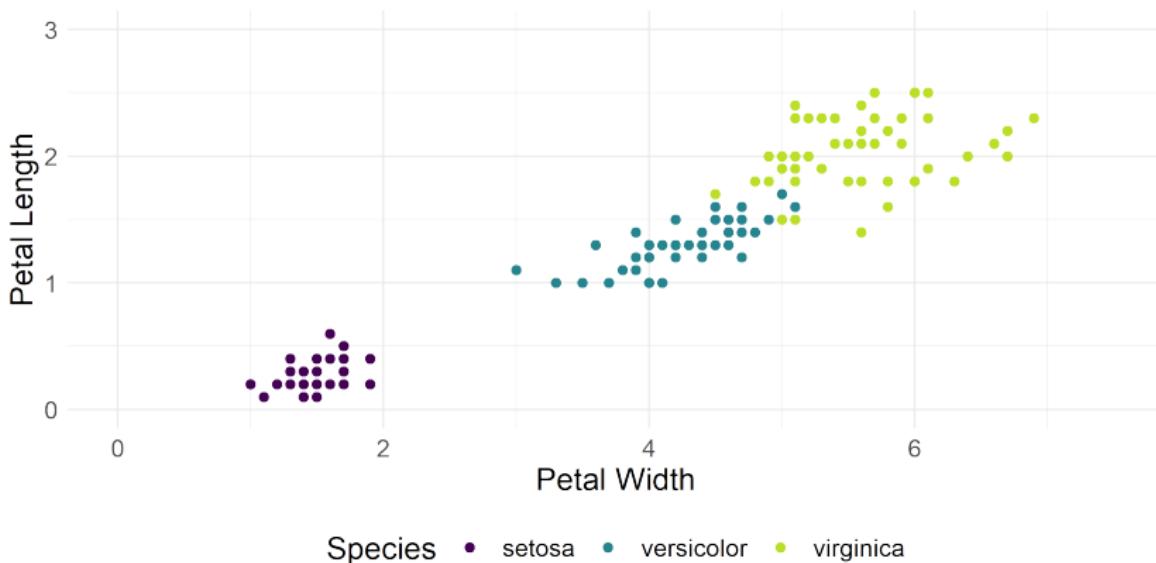


MULTICLASS CLASSIFICATION

Scenario: Multiclass classification with $g > 2$ classes

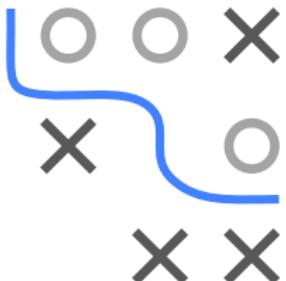
$$\mathcal{D} \subset (\mathcal{X} \times \mathcal{Y})^n, \mathcal{Y} = \{1, \dots, g\}$$

Example: Iris dataset with $g = 3$



REVISION: RISK FOR CLASSIFICATION

Goal: Find a model $f : \mathcal{X} \rightarrow \mathbb{R}^g$, where g is the number of classes, that minimizes the expected loss over random variables $(\mathbf{x}, y) \sim \mathbb{P}_{xy}$



$$\mathcal{R}(f) = \mathbb{E}_{xy}[L(y, f(\mathbf{x}))] = \mathbb{E}_{\mathbf{x}} \left[\sum_{k \in \mathcal{Y}} L(k, f(\mathbf{x})) \mathbb{P}(y = k | \mathbf{x} = \mathbf{x}) \right]$$

The optimal model for a loss function $L(y, f(\mathbf{x}))$ is

$$\hat{f}(\mathbf{x}) = \arg \min_{f \in \mathcal{H}} \sum_{k \in \mathcal{Y}} L(k, f(\mathbf{x})) \mathbb{P}(y = k | \mathbf{x} = \mathbf{x}).$$

Because we usually do not know \mathbb{P}_{xy} , we minimize the **empirical risk** as an approximation to the **theoretical** risk

$$\hat{f} = \arg \min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) = \arg \min_{f \in \mathcal{H}} \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right).$$

TYPES OF CLASSIFIERS

- We already saw losses for binary classification tasks. Now we will consider losses for **multiclass classification** tasks.
- For multiclass classification, loss functions will be defined on
 - vectors of scores

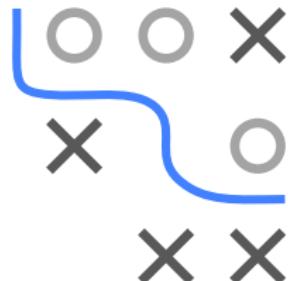
$$f(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_g(\mathbf{x}))$$

- vectors of probabilities

$$\pi(\mathbf{x}) = (\pi_1(\mathbf{x}), \dots, \pi_g(\mathbf{x}))$$

- hard labels

$$h(\mathbf{x}) = k, k \in \{1, 2, \dots, g\}$$



ONE-HOT ENCODING

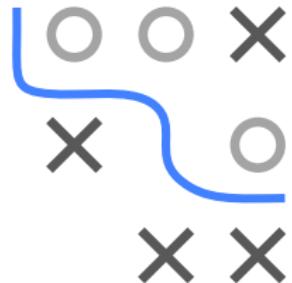
- Multiclass outcomes y with classes $1, \dots, g$ are often transformed to g binary (1/0) outcomes using

$$\text{with } \mathbb{1}_{\{y=k\}} = \begin{cases} 1 & \text{if } y = k \\ 0 & \text{otherwise} \end{cases}$$

- One-hot encoding does not lose any information contained in the outcome.

Example: Iris

Species	Species.setosa	Species.versicolor	Species.virginica
versicolor	0	1	0
virginica	0	0	1
versicolor	0	1	0
versicolor	0	1	0
setosa	1	0	0
setosa	1	0	0



0-1-Loss



0-1-LOSS

We have already seen that optimizer $\hat{h}(\mathbf{x})$ of the theoretical risk using the 0-1-loss

$$L(y, h(\mathbf{x})) = \mathbb{1}_{\{y \neq h(\mathbf{x})\}}$$

is the Bayes optimal classifier, with

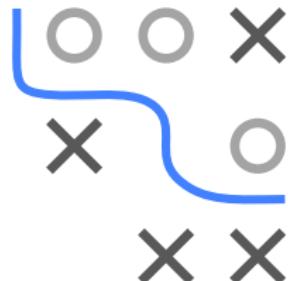
$$\hat{h}(\mathbf{x}) = \arg \max_{l \in \mathcal{V}} \mathbb{P}(y = l \mid \mathbf{x} = \mathbf{x})$$

and the optimal constant model (featureless predictor)

$$h(\mathbf{x}) = k, k \in \{1, 2, \dots, g\}$$

is the classifier that predicts the most frequent class $k \in \{1, 2, \dots, g\}$ in the data

$$h(\mathbf{x}) = \text{mode} \left\{ y^{(i)} \right\}.$$



MC Brier Score



MC BRIER SCORE

The (binary) Brier score generalizes to the multiclass Brier score that is defined on a vector of class probabilities $(\pi_1(\mathbf{x}), \dots, \pi_g(\mathbf{x}))$

$$L(y, \pi(\mathbf{x})) = \sum_{k=1}^g (\mathbb{1}_{\{y=k\}} - \pi_k(\mathbf{x}))^2.$$

Optimal constant prob vector $\pi(\mathbf{x}) = (\theta_1, \dots, \theta_g)$:

$$\boldsymbol{\theta} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^g, \sum \theta_k = 1} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) \quad \text{with} \quad \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \left(\sum_{i=1}^n \sum_{k=1}^g (\mathbb{1}_{\{y^{(i)}=k\}} - \theta_k)^2 \right)$$

We solve this by setting the derivative w.r.t. θ_k to 0

$$\frac{\partial \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}{\partial \theta_k} = -2 \cdot \sum_{i=1}^n (\mathbb{1}_{\{y^{(i)}=k\}} - \theta_k) = 0 \Rightarrow \hat{\pi}_k(\mathbf{x}) = \hat{\theta}_k = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{y^{(i)}=k\}},$$

being the fraction of class- k observations.

NB: We naively ignored the constraints! But since $\sum_{k=1}^g \hat{\theta}_k = 1$ holds for the minimizer of the unconstrained problem, we are fine. Could have also used Lagrange multipliers!



MC BRIER SCORE

Claim: For $g = 2$ the MC Brier score is exactly twice as high as the binary Brier score, defined as $(\pi_1(\mathbf{x}) - y)^2$.

Proof:

$$L(y, \pi(x)) = \sum_{k=0}^1 (\mathbb{1}_{\{y=k\}} - \pi_k(\mathbf{x}))^2$$

For $y = 0$:

$$\begin{aligned} L(y, \pi(x)) &= (1 - \pi_0(\mathbf{x}))^2 + (0 - \pi_1(\mathbf{x}))^2 = (1 - (1 - \pi_1(\mathbf{x})))^2 + \pi_1(\mathbf{x})^2 \\ &= \pi_1(\mathbf{x})^2 + \pi_1(\mathbf{x})^2 = 2 \cdot \pi_1(\mathbf{x})^2 \end{aligned}$$

For $y = 1$:

$$\begin{aligned} L(y, \pi(x)) &= (0 - \pi_0(\mathbf{x}))^2 + (1 - \pi_1(\mathbf{x}))^2 = (-(1 - \pi_1(\mathbf{x})))^2 + (1 - \pi_1(\mathbf{x}))^2 \\ &= 1 - 2 \cdot \pi_1(\mathbf{x}) + \pi_1(\mathbf{x})^2 + 1 - 2 \cdot \pi_1(\mathbf{x}) + \pi_1(\mathbf{x})^2 \\ &= 2 \cdot (1 - 2 \cdot \pi_1(\mathbf{x}) + \pi_1(\mathbf{x})^2) = 2 \cdot (1 - \pi_1(\mathbf{x}))^2 = 2 \cdot (\pi_1(\mathbf{x}) - 1)^2 \end{aligned}$$

$$L(y, \pi(x)) = \begin{cases} 2 \cdot \pi_1(\mathbf{x})^2 & \text{for } y = 0 \\ 2 \cdot (\pi_1(\mathbf{x}) - 1)^2 & \text{for } y = 1 \end{cases} = 2 \cdot (\pi_1(\mathbf{x}) - y)^2$$



Logarithmic Loss



LOGARITHMIC LOSS (LOG-LOSS)

The generalization of the Binomial loss (logarithmic loss) for two classes is the multiclass **logarithmic loss / cross-entropy loss**:

$$L(y, \pi(x)) = - \sum_{k=1}^g \mathbb{1}_{\{y=k\}} \log (\pi_k(\mathbf{x})) ,$$

with $\pi_k(\mathbf{x})$ denoting the predicted probability for class k .

Optimal constant prob vector $\pi(\mathbf{x}) = (\theta_1, \dots, \theta_g)$:

$$\pi_k(\mathbf{x}) = \theta_k = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{y^{(i)}=k\}},$$

being the fraction of class- k observations.

Proof: Exercise.

In the upcoming section we will see how this corresponds to the (multinomial) **softmax regression**.



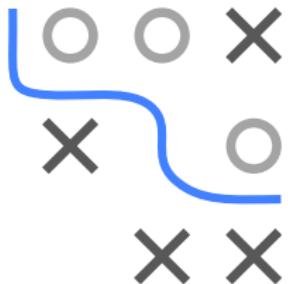
LOGARITHMIC LOSS (LOG-LOSS)

Claim: For $g = 2$ the log-loss is equal to the Bernoulli loss, defined as

$$L_{0,1}(y, \pi_1(\mathbf{x})) = -y\log(\pi_1(\mathbf{x})) - (1-y)\log(1 - \pi_1(\mathbf{x}))$$

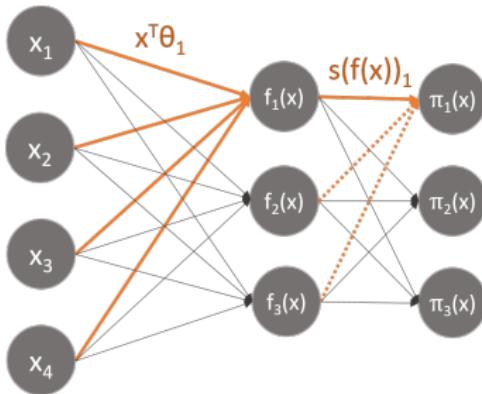
Proof:

$$\begin{aligned} L_{0,1}(y, \pi_1(\mathbf{x})) &= -y\log(\pi_1(\mathbf{x})) - (1-y)\log(1 - \pi_1(\mathbf{x})) \\ &= -y\log(\pi_1(\mathbf{x})) - (1-y)\log(\pi_0(\mathbf{x})) \\ &= -\mathbb{1}_{\{y=1\}}\log(\pi_1(\mathbf{x})) - \mathbb{1}_{\{y=0\}}\log(\pi_0(\mathbf{x})) \\ &= -\sum_{k=0}^1 \mathbb{1}_{\{y=k\}} \log(\pi_k(\mathbf{x})) = L(y, \pi(\mathbf{x})) \end{aligned}$$



Introduction to Machine Learning

Multiclass Classification Softmax Regression



Learning goals

- Know softmax regression
- Understand that softmax regression is a generalization of logistic regression

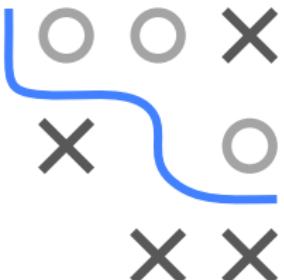
FROM LOGISTIC REGRESSION ...

Remember **logistic regression** ($\mathcal{Y} = \{0, 1\}$): We combined the hypothesis space of linear functions, transformed by the logistic function $s(z) = \frac{1}{1+\exp(-z)}$, i.e.

$$\mathcal{H} = \left\{ \pi : \mathcal{X} \rightarrow \mathbb{R} \mid \pi(\mathbf{x}) = s(\boldsymbol{\theta}^\top \mathbf{x}) \right\},$$

with the Bernoulli (logarithmic) loss:

$$L(y, \pi(\mathbf{x})) = -y \log (\pi(\mathbf{x})) - (1 - y) \log (1 - \pi(\mathbf{x})).$$



Remark: We suppress the intercept term for better readability. The intercept term can be easily included via $\boldsymbol{\theta}^\top \tilde{\mathbf{x}}$, $\boldsymbol{\theta} \in \mathbb{R}^{p+1}$, $\tilde{\mathbf{x}} = (1, \mathbf{x})$.

... TO SOFTMAX REGRESSION

There is a straightforward generalization to the multiclass case:

- Instead of a single linear discriminant function we have g linear discriminant functions

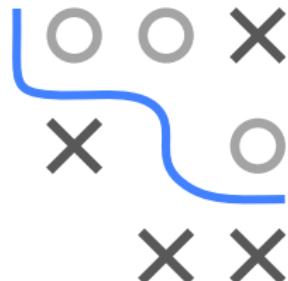
$$f_k(\mathbf{x}) = \boldsymbol{\theta}_k^\top \mathbf{x}, \quad k = 1, 2, \dots, g,$$

each indicating the confidence in class k .

- The g score functions are transformed into g probability functions by the **softmax** function $s : \mathbb{R}^g \rightarrow \mathbb{R}^g$

$$\pi_k(\mathbf{x}) = s(f(\mathbf{x}))_k = \frac{\exp(\boldsymbol{\theta}_k^\top \mathbf{x})}{\sum_{j=1}^g \exp(\boldsymbol{\theta}_j^\top \mathbf{x})},$$

instead of the **logistic** function for $g = 2$. The probabilities are well-defined: $\sum \pi_k(\mathbf{x}) = 1$ and $\pi_k(\mathbf{x}) \in [0, 1]$ for all k .



... TO SOFTMAX REGRESSION

- The softmax function is a generalization of the logistic function.
For $g = 2$, the logistic function and the softmax function are equivalent.
- Instead of the **Bernoulli** loss, we use the multiclass **logarithmic loss**

$$L(y, \pi(\mathbf{x})) = - \sum_{k=1}^g \mathbb{1}_{\{y=k\}} \log (\pi_k(\mathbf{x})).$$

- Note that the softmax function is a “smooth” approximation of the arg max operation, so $s((1, 1000, 2)^T) \approx (0, 1, 0)^T$ (picks out 2nd element!).
- Furthermore, it is invariant to constant offsets in the input:

$$s(f(\mathbf{x}) + \mathbf{c}) = \frac{\exp(\theta_k^\top \mathbf{x} + c)}{\sum_{j=1}^g \exp(\theta_j^\top \mathbf{x} + c)} = \frac{\exp(\theta_k^\top \mathbf{x}) \cdot \exp(c)}{\sum_{j=1}^g \exp(\theta_j^\top \mathbf{x}) \cdot \exp(c)} = s(f(\mathbf{x}))$$



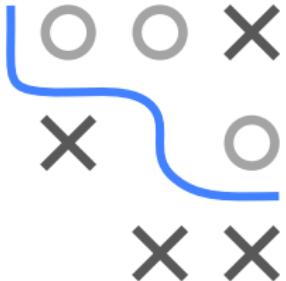
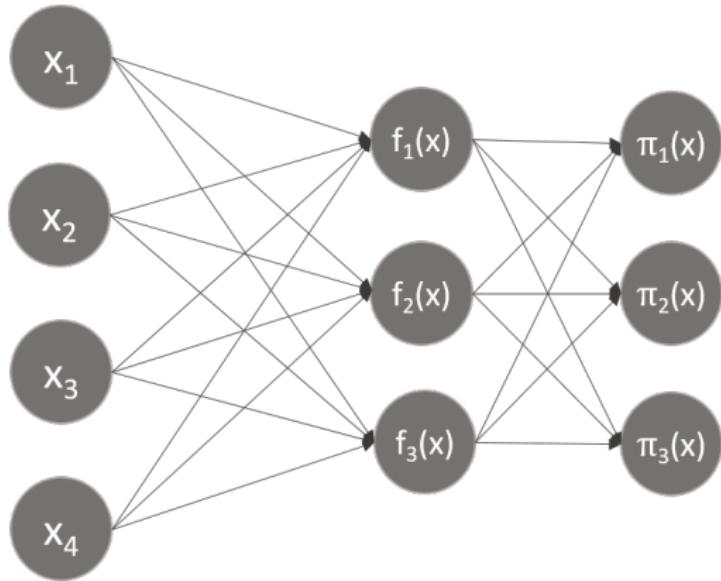
LOGISTIC VS. SOFTMAX REGRESSION

	Logistic Regression	Softmax Regression
\mathcal{Y}	$\{0, 1\}$	$\{1, 2, \dots, g\}$
Discriminant fun.	$f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x}$	$f_k(\mathbf{x}) = \boldsymbol{\theta}_k^\top \mathbf{x}, k = 1, 2, \dots, g$
Probabilities	$\pi(\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^\top \mathbf{x})}$	$\pi_k(\mathbf{x}) = \frac{\exp(\boldsymbol{\theta}_k^\top \mathbf{x})}{\sum_{j=1}^g \exp(\boldsymbol{\theta}_j^\top \mathbf{x})}$
$L(y, \pi(\mathbf{x}))$	Bernoulli / logarithmic loss $-y \log(\pi(\mathbf{x})) - (1 - y) \log(1 - \pi(\mathbf{x}))$	Multiclass logarithmic loss $-\sum_{k=1}^g [y = k] \log(\pi_k(\mathbf{x}))$



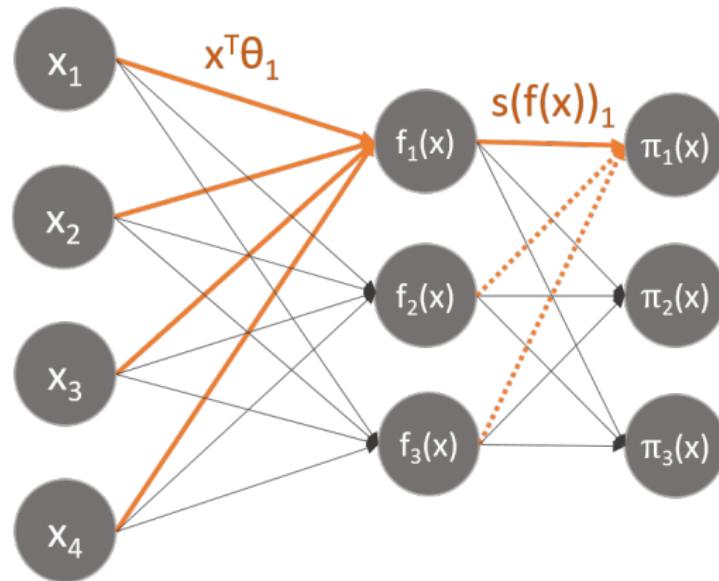
LOGISTIC VS. SOFTMAX REGRESSION

We can schematically depict softmax regression as follows:



LOGISTIC VS. SOFTMAX REGRESSION

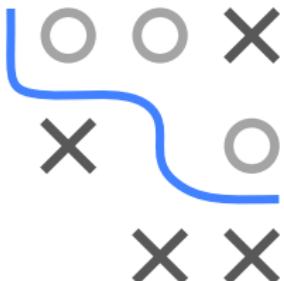
We can schematically depict softmax regression as follows:



LOGISTIC VS. SOFTMAX REGRESSION

Further comments:

- We can now, for instance, calculate gradients and optimize this with standard numerical optimization software.
- Softmax regression has an unusual property in that it has a “redundant” set of parameters. If we subtract a fixed vector from all θ_k , the predictions do not change at all. Hence, our model is “over-parameterized”. For any hypothesis we might fit, there are multiple parameter vectors that give rise to exactly the same hypothesis function. This also implies that the minimizer of $\mathcal{R}_{\text{emp}}(\theta)$ above is not unique! Hence, a numerical trick is to set $\theta_g = 0$ and only optimize the other θ_k . This does not restrict our hypothesis space, but the constrained problem is now convex, i.e., there exists exactly one parameter vector for every hypothesis.
- A similar approach is used in many ML models: multiclass LDA, naive Bayes, neural networks and boosting.



SOFTMAX: LINEAR DISCRIMINANT FUNCTIONS

Softmax regression gives us a **linear classifier**.

- The softmax function $s(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{j=1}^g \exp(z_j)}$ is
 - a rank-preserving function, i.e. the ranks among the elements of the vector \mathbf{z} are the same as among the elements of $s(\mathbf{z})$. This is because softmax transforms all scores by taking the $\exp(\cdot)$ (rank-preserving) and divides each element by **the same** normalizing constant.



Thus, the softmax function has a unique inverse function $s^{-1} : \mathbb{R}^g \rightarrow \mathbb{R}^g$ that is also monotonic and rank-preserving.

Applying s_k^{-1} to $\pi_k(\mathbf{x}) = \frac{\exp(\theta_k^\top \mathbf{x})}{\sum_{j=1}^g \exp(\theta_j^\top \mathbf{x})}$ gives us $f_k(\mathbf{x}) = \theta_k^\top \mathbf{x}$.

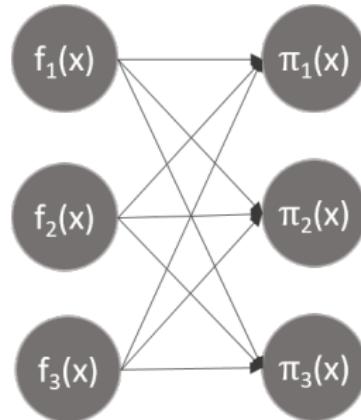
Thus, softmax regression is a linear classifier.

GENERALIZING SOFTMAX REGRESSION

Instead of simple linear discriminant functions we could use **any** model that outputs g scores

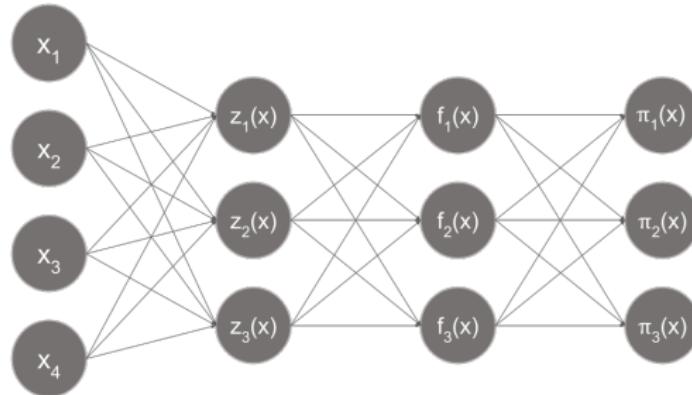
$$f_k(\mathbf{x}) \in \mathbb{R}, k = 1, 2, \dots, g$$

We can choose a multiclass loss and optimize the score functions $f_k, k \in \{1, \dots, g\}$ by multivariate minimization. The scores can be transformed to probabilities by the **softmax** function.



GENERALIZING SOFTMAX REGRESSION

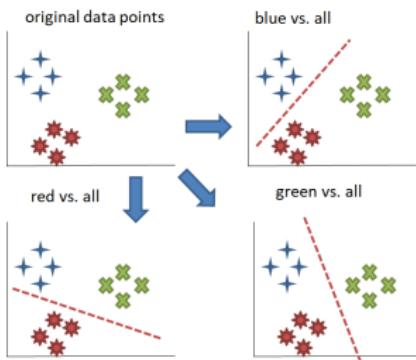
For example for a **neural network** (note that softmax regression is also a neural network with no hidden layers):



Remark: For more details about neural networks please refer to the lecture **Deep Learning**.

Introduction to Machine Learning

Multiclass Classification One-vs-Rest and One-vs-One



Learning goals

- Reduce a multiclass problem to multiple binary problems in a model-agnostic way
- Know one-vs-rest reduction
- Know one-vs-one reduction



MULTICLASS TO BINARY REDUCTION

- Assume we have a way to train binary classifiers, either outputting class labels $h(\mathbf{x})$, scores $f(\mathbf{x})$ or probabilities $\pi(\mathbf{x})$.
- We are now looking for a model-agnostic reduction principle to reduce a multiclass problem to the problem of solving **multiple binary problems**.
- Two common approaches are **one-vs-rest** and **one-vs-one** reductions.



CODEBOOKS

How binary problems are generated can be defined by a codebook.

Example:

Class	$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	$f_3(\mathbf{x})$
1	1	-1	-1
2	-1	1	1
3	0	1	-1



- The k -th column defines how classes of all observations are encoded in the binary subproblem / for binary classifier $f_k(\mathbf{x})$.
- Entry (m, i) takes values $\in \{-1, 0, +1\}$
 - if 0, observations of class $y^{(i)} = m$ are ignored.
 - if 1, observations of class $y^{(i)} = m$ are encoded as 1.
 - if -1, observations of class $y^{(i)} = m$ are encoded as -1.

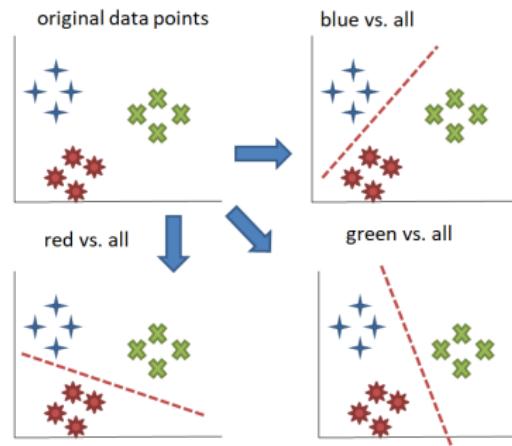
One-vs-Rest



ONE-VS-REST

Create g binary subproblems, where in each the k -th original class is encoded as $+1$, and all other classes (the **rest**) as -1 .

Class	$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	$f_3(\mathbf{x})$
1	1	-1	-1
2	-1	1	-1
3	-1	-1	1



ONE-VS-REST

- Making decisions means applying all classifiers to a sample $\mathbf{x} \in \mathcal{X}$ and predicting the label k for which the corresponding classifier reports the highest confidence:

$$\hat{y} = \arg \max_{k \in \{1, 2, \dots, g\}} \hat{f}_k(\mathbf{x}).$$

- Obtaining calibrated posterior probabilities is not completely trivial, we could fit a second-stage, multinomial logistic regression model on our output scores, so with inputs $(\hat{f}_1(\mathbf{x}^{(i)}), \dots, \hat{f}_g(\mathbf{x}^{(i)}))$ and outputs $y^{(i)}$ as training data.



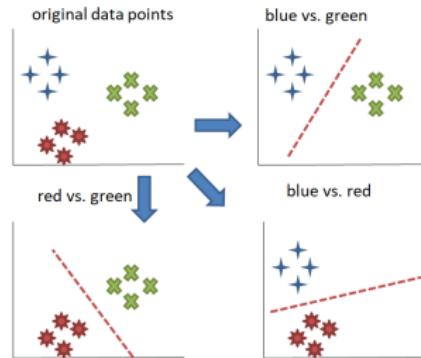
One-vs-One



ONE-VS-ONE

We create $\frac{g(g-1)}{2}$ binary sub-problems, where each $\mathcal{D}_{k,\tilde{k}} \subset \mathcal{D}$ only considers observations from a class-pair $y^{(i)} \in \{k, \tilde{k}\}$, other observations are omitted.

Class	$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	$f_3(\mathbf{x})$
1	1	-1	0
2	-1	0	1
3	0	1	-1



ONE-VS-ONE

- Label prediction is done via **majority voting**. We predict the label of a new \mathbf{x} with all classifiers and select the class that occurred most often.
- **Pairwise coupling** (see *Hastie, T. and Tibshirani, R. (1998). Classification by Pairwise Coupling*) is a heuristic to transform scores obtained by a one-vs-one reduction to probabilities.



COMPARISON ONE-VS-ONE AND ONE-VS-REST

- Note that each binary problem has now much less than n observations!
- For classifiers that scale (at least) quadratically with the number of observations, this means that one-vs-one usually does not create quadratic extra effort in g , but often only approximately linear extra effort in g .
- We experimentally investigate the train times of the one-vs-rest and one-vs-one approaches for an increasing number of classes g .
- We train a support vector machine classifier (SVMs will be covered later in the lecture) on an artificial dataset with $n = 1000$.



COMPARISON ONE-VS-ONE AND ONE-VS-REST

We see that the computational effort for one-vs-one is much higher than for one-vs-rest, but it does not scale proportionally to the (quadratic) number of trained classifiers.

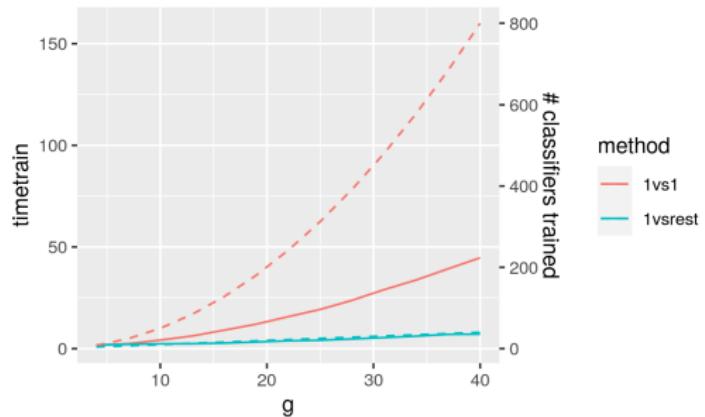
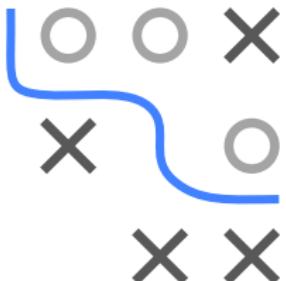


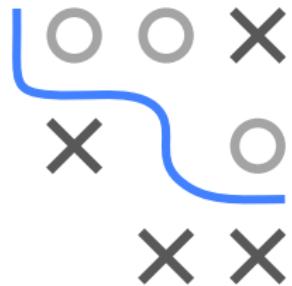
Figure: The number of classes vs. the training time (solid lines, left axis) and number of learners (dashed lines, right axis) for each of the two approaches.



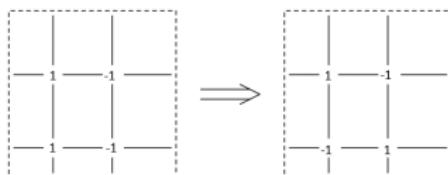
Introduction to Machine Learning

Multiclass Classification

Designing Codebooks and ECOC



Learning goals



- Know what a codebook is
- Understand that codebooks generalize one-vs-one and one-vs-rest
- Know how to define a good codebook and error-correcting output codes (ECOC)
- Know how randomized hill-climbing algorithm is used to find good

Designing Codebooks



CODEBOOKS

- We have already seen that we can write down principles like one-vs-rest and one-vs-one reduction compactly by so-called **codebooks**.
- During training, a scoring classifier is trained for each column.
- The k -th row is called **codeword** for class k .
- Knowing the principle of **codebooks**, we can define multiclass-to-binary reductions quite flexibly.
- We can now ask ourselves, how to create optimal codebooks.

Class	$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	$f_3(\mathbf{x})$
1	1	-1	-1
2	-1	1	-1
3	-1	-1	1

Class	$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	$f_3(\mathbf{x})$
1	1	-1	0
2	-1	0	1
3	0	1	-1

Left: one-vs-rest codebook. Right: one-vs-one codebook.



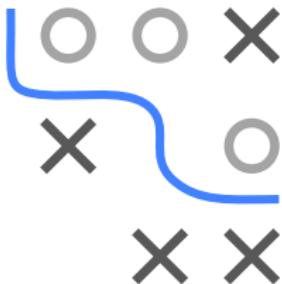
CODEBOOKS: DECIDING LABELS

For a general codebook, once we trained the classifiers, how to predict the class \hat{y} for a new input \mathbf{x} ?

- When a new sample \mathbf{x} is going to be classified, all classifiers f_k are applied to \mathbf{x} , scores are potentially transformed and turned into binary labels by $\text{sgn}(f_k(\mathbf{x}))$.

Class	$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	$f_3(\mathbf{x})$
1	1	1	0
2	-1	1	1
3	0	-1	-1
$\text{sgn}(\hat{f}(\mathbf{x}))$	-1	1	-1

- We obtain a code for the observation \mathbf{x} for which we can calculate the distance to the codewords of the other classes. This can be done by **Hamming distance** (counting the number of bits that differ) or by L_1 -distance.



CODEBOOKS: DECIDING LABELS

- For example, the L_1 -distance between $\text{sgn}(\hat{f}(\mathbf{x})) = (-1, 1, -1)$ and the class 1 codeword $(1, 1, 0)$ is 3.
- We can do so for all the classes to obtain respective distances:

Classes	Dist
1	3
2	2
3	3

The distance for class 2 is minimal, therefore we predict class 2 for the input \mathbf{x} .



DEFINING GOOD CODEBOOKS

Question: How to define a good codebook?

- Assume we are given a test observation (\mathbf{x}, y) with $y = 2$.
- Assume classifier $f_2(\mathbf{x})$ produces a false prediction:

Class	$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	$f_3(\mathbf{x})$	Dist
1	1	-1	0	3
2	-1	1	1	2
3	0	-1	-1	3
$ \hat{f}(\mathbf{x}) $	-1	-1	1	

- Even though $f_2(\mathbf{x})$ is wrong, the overall prediction will be correct in the above case, if we pick the best codeword w.r.t. distance from the predicted codeword.
- We effectively **corrected** for the error.
- This motivates a desirable characteristic of a codebook: we want to have codes that can correct for as many errors as possible.
- Which is called **error-correcting output codes** (ECOC).



Error-Correcting Codes (ECOC)



ERROR-CORRECTING CODES (ECOC)

The power of a code to correct errors is related to the **row separation**:

- Each codeword should be well-separated in Hamming distance from each of the other codewords.
- Otherwise, if the class codewords are very similar, a prediction error in a single binary classifier easily results in an “overall” error.
- If the minimum distance between any pair of codewords is d , the code can correct at least $\lfloor \frac{d-1}{2} \rfloor$ single bit errors.

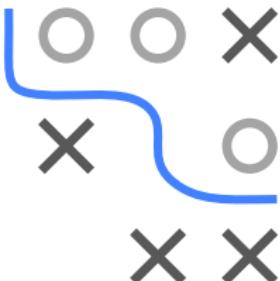


ERROR-CORRECTING CODES (ECOC)

Another desirable property is **column separation**:

- Columns should be uncorrelated.
- If two columns k and l are similar or identical, a learning algorithm will make similar (correlated) mistakes in learning f_k and f_l .
- Error-correcting codes only succeed if the errors made in the individual classifiers are relatively uncorrelated, so that the number of simultaneous errors in many classifiers is small.
- Errors in classifiers f_k and f_l will also be highly correlated if the bits in those columns are complementary.
- Try to ensure that columns are neither identical nor complementary.

→ **We want to maximize distances between rows, and want the distances between columns to not be too small (identical columns) or too high (complementary columns).**

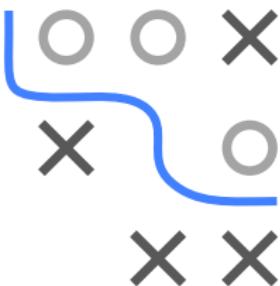


ERROR-CORRECTING CODES (ECOC)

Remark:

- In general, if there are k classes, there will be at most $2^{k-1} - 1$ usable binary columns.
- For example for $k = 3$, there are only $2^3 = 8$ possible columns. Of these, half are complements of the other half. The columns that only contain 1s or the one that only contains -1s are also not usable.

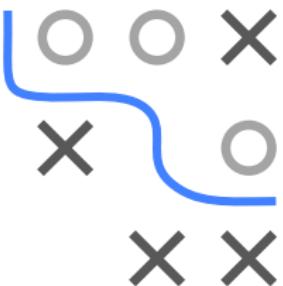
Class	$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	$f_3(\mathbf{x})$	$f_4(\mathbf{x})$	$f_5(\mathbf{x})$	$f_6(\mathbf{x})$	$f_7(\mathbf{x})$	$f_8(\mathbf{x})$
1	-1	-1	-1	-1	1	1	1	1
2	-1	-1	1	1	-1	-1	1	1
3	-1	1	-1	1	-1	1	-1	1



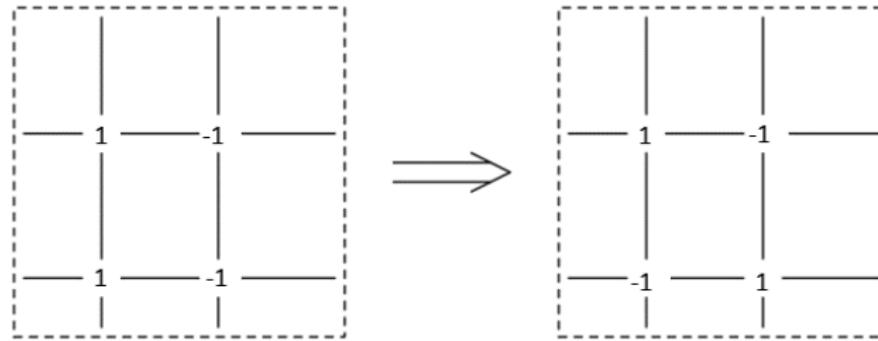
ERROR-CORRECTING CODES (ECOC)

Assume we have the budget to train L binary classifiers and now want to find an error-correcting code with maximal row and column separation.

- For only few classes $g \leq 11$, exhaustive search can be performed and a codebook that has good row and column separation is chosen.
- However, for many classes $g > 11$, it becomes more and more challenging to find the optimal codebook with codewords of length L .
- *Dietterich et al.* employed a randomized hill-climbing algorithm for this task.

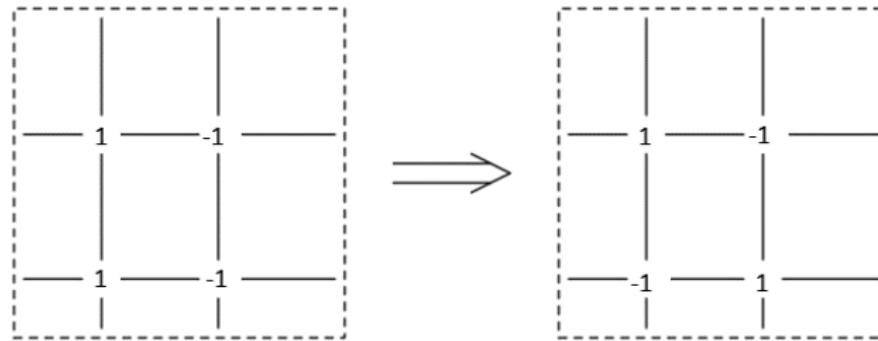


ECOC: RANDOMIZED HILL-CLIMBING ALGORITHM



- g codewords of length L are randomly drawn.
- Any pair of such random strings will be separated by a Hamming distance that is binomially distributed with mean $\frac{L}{2}$.
- The algorithm now iteratively improves the code: The algorithm repeatedly finds the pair of rows closest together (in Hamming distance or any other distance) and the pair of columns that have the “most extreme” distance (i.e. too close, or too far apart).

ECOC: RANDOMIZED HILL-CLIMBING ALGORITHM



- The algorithm then computes the four codeword bits where these rows and columns intersect and changes them to improve the row and column separations
- When the procedure reaches a local maximum, the algorithm randomly chooses pairs of rows and columns and tries to improve their separation.

INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

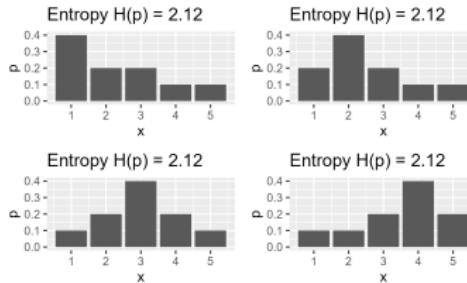
Boosting

Gaussian Processes



Introduction to Machine Learning

Entropy I



Learning goals

- Entropy measures expected information for discrete RVs
- Know entropy and its properties



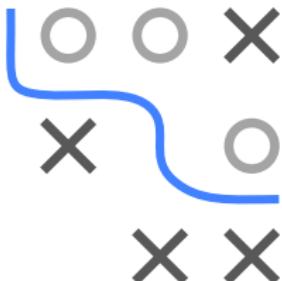
INFORMATION THEORY

- **Information Theory** is a field of study based on probability theory.
- Foundation was laid by Claude Shannon in 1948; since then been applied in: communication theory, computer science, optimization, cryptography, machine learning and statistical inference.
- Quantify the "amount" of information gained or uncertainty reduced when a random variable is observed.
- Also about storing and transmitting information.



INFORMATION THEORY / 2

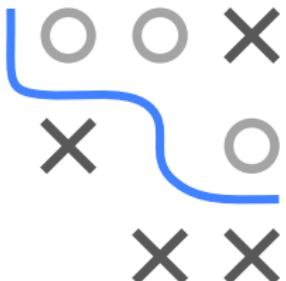
- We introduce the basic concepts from a probabilistic perspective, without referring too much to communication, channels or coding.
- We will show some proofs, but not for everything. We recommend *Elements of Information Theory* by Cover and Thomas as a reference for more.
- The application of information theory to the concepts of statistics and ML can sometimes be confusing, we will try to make the connection as clear as possible.
- In this unit we develop entropy as a measure of uncertainty in terms of expected information.



ENTROPY AS SURPRISAL AND UNCERTAINTY

For a discrete random variable X with domain $\mathcal{X} \ni x$ and pmf $p(x)$:

$$\begin{aligned} H(X) := H(p) &= -\mathbb{E}[\log_2(p(X))] = -\sum_{x \in \mathcal{X}} p(x) \log_2 p(x) \\ &= \mathbb{E} \left[\log_2 \left(\frac{1}{p(X)} \right) \right] = \sum_{x \in \mathcal{X}} p(x) \log_2 \frac{1}{p(x)} \end{aligned}$$



Some technicalities first:

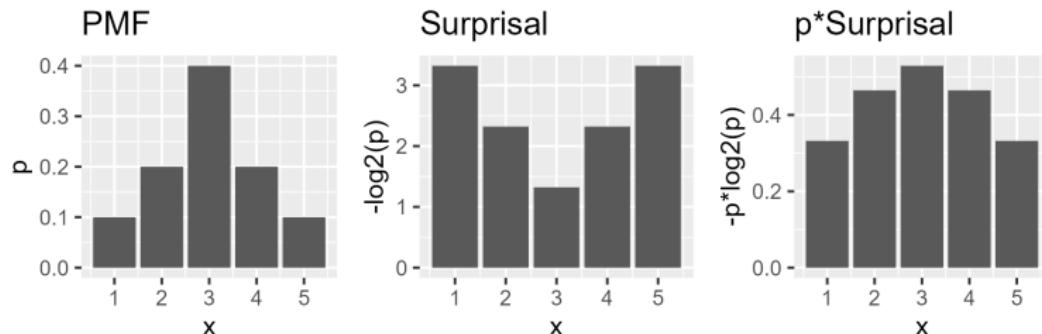
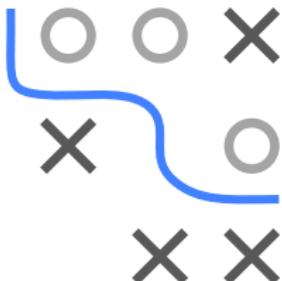
- H is actually Greek capital letter **Eta** (η) for **entropy**
- Base of the log simply specifies the unit we measure information in, usually bits (base 2) or 'nats' (base e)
- If $p(x) = 0$ for an x , then $p(x) \log_2 p(x)$ is taken to be zero, because $\lim_{p \rightarrow 0} p \log_2 p = 0$.

ENTROPY AS SURPRISAL AND UNCERTAINTY

$$H(X) = -\mathbb{E}[\log_2(p(X))] = - \sum_{x \in \mathcal{X}} p(x) \log_2 p(x)$$

Now: What's the point?

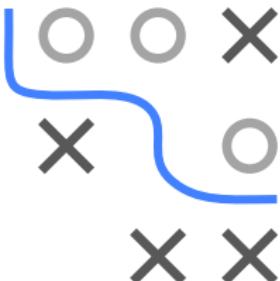
- The negative log probabilities $-\log_2 p(x)$ are called "surprisal"
- More surprising means less likely
- PMFs surprising, so with higher H, when events more equally likely
- Entropy is simply expected surprisal



- The final entropy is $H(X) = 2.12$ (bits).

ENTROPY BASIC PROPERTIES

$$H(X) := H(p) = -\mathbb{E}[\log_2(p(X))] = -\sum_{x \in \mathcal{X}} p(x) \log_2 p(x)$$

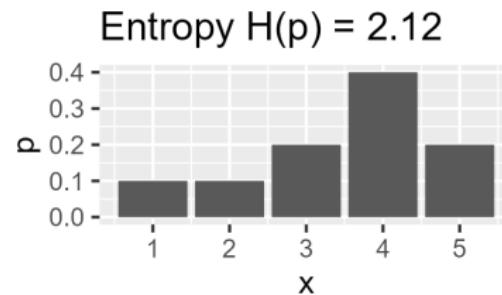
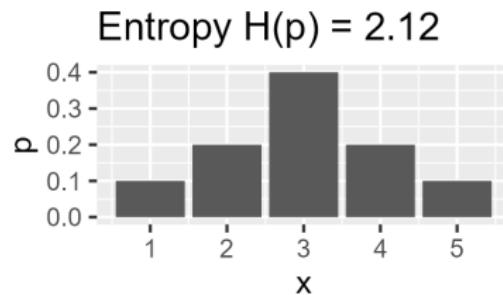
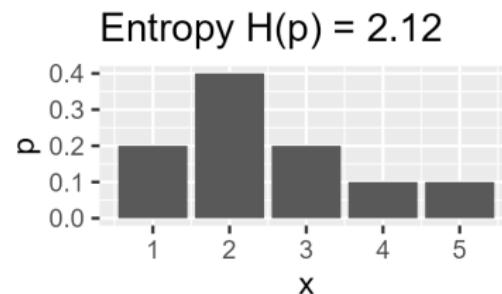
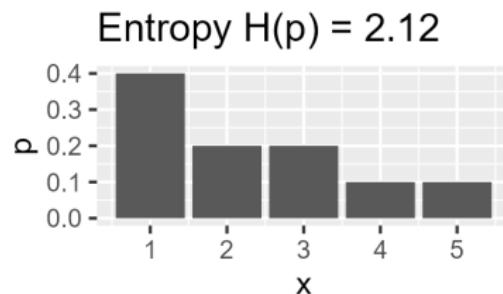


- ① Entropy is non-negative, so $H(X) \geq 0$
- ② If one event has probability $p(x) = 1$, then $H(X) = 0$
- ③ Adding or removing an event with $p(x) = 0$ doesn't change it
- ④ $H(X)$ is continuous in probabilities $p(x)$

All these properties follow directly from the definition.

ENTROPY RE-ORDERING

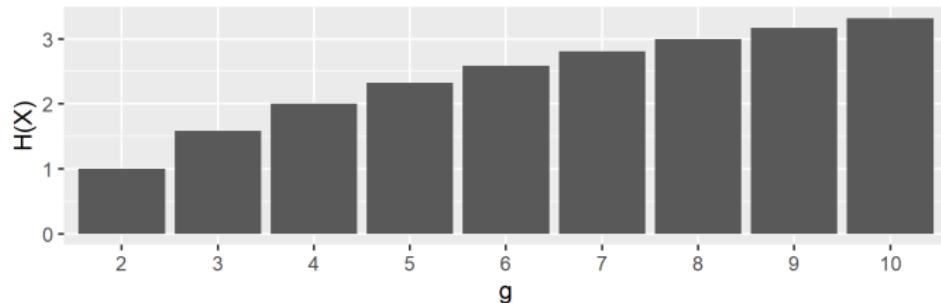
- ➅ Symmetry. If the values $p(x)$ in the pmf are re-ordered, entropy does not change (proof is trivial).



ENTROPY OF UNIFORM DISTRIBUTIONS

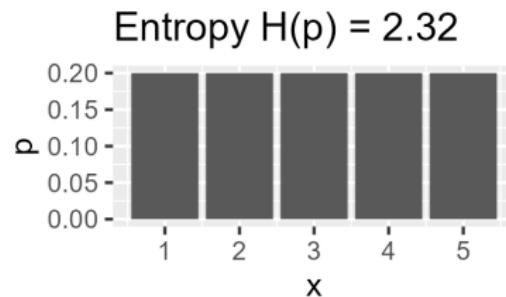
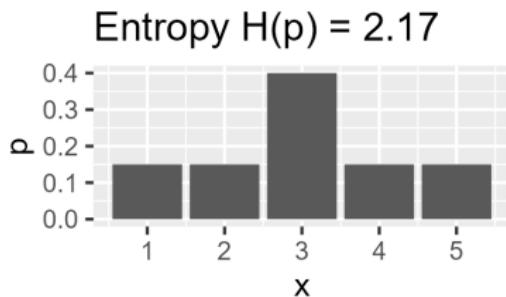
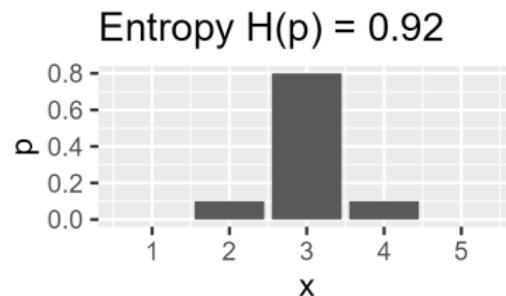
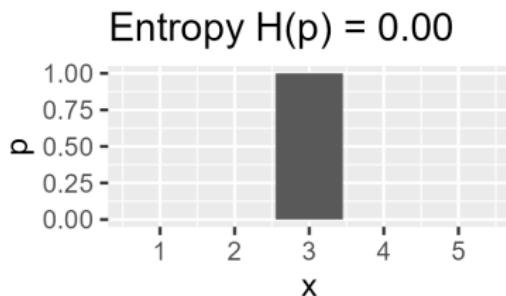
Let X be a uniform, discrete RV with g outcomes (g -sided fair die).

$$H(X) = - \sum_{i=1}^g \frac{1}{g} \log_2 \left(\frac{1}{g} \right) = \log_2 g$$



The more sides a die has, the harder it is to predict the outcome.
Unpredictability grows *monotonically* with the number of potential outcomes, but at a decreasing rate.

ENTROPY IS MAXIMAL FOR UNIFORM



- Naive observation:
Entropy min for 1-point and max for uniform distribution

ENTROPY IS MAXIMAL FOR UNIFORM

- Entropy is maximal for a uniform distribution,
for domain of size g : $H(X) \leq -g \frac{1}{g} \log_2(\frac{1}{g}) = \log_2(g)$.

Proof: So we want to maximize w.r.t. all p_i :

$$\underset{p_1, p_2, \dots, p_g}{\operatorname{argmax}} - \sum_{i=1}^g p_i \log_2 p_i$$

subject to

$$\sum_{i=1}^g p_i = 1$$



ENTROPY IS MAXIMAL FOR UNIFORM / 2

The Lagrangian $L(p_1, \dots, p_g, \lambda)$ is :

$$L(p_1, \dots, p_g, \lambda) = - \sum_{i=1}^g p_i \log_2(p_i) - \lambda \left(\sum_{i=1}^g p_i - 1 \right)$$

Solving when requiring $\nabla L = 0$,

$$\begin{aligned} \frac{\partial L(p_1, \dots, p_g, \lambda)}{\partial p_i} &= 0 = -\log_2(p_i) - \frac{1}{\log(2)} - \lambda \\ \implies p_i &= \frac{2^{-\lambda}}{e} \implies p_i = \frac{1}{g}, \end{aligned}$$

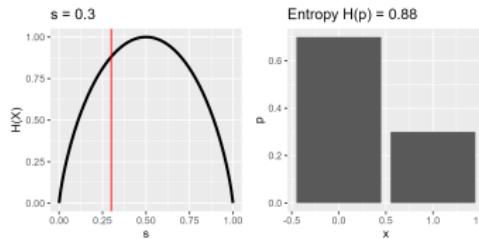
last step follows from that all p_i are equal and constraint

NB: We also could have solved the constraint for p_1 and substitute $p_1 = 1 - \sum_{i=2}^g p_i$ in the objective to avoid constrained optimization.



Introduction to Machine Learning

Information Theory Entropy II



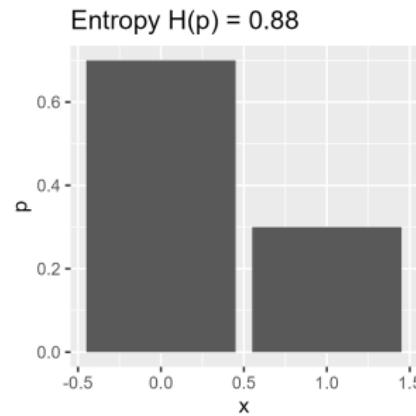
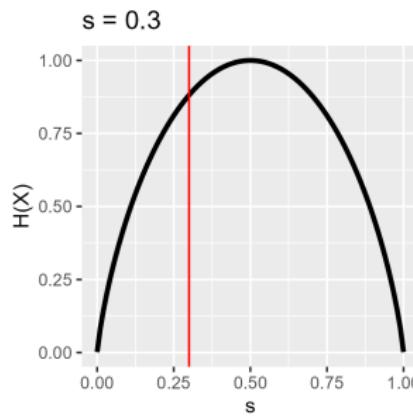
Learning goals

- Further properties of entropy and joint entropy
- Understand that uniqueness theorem justifies choice of entropy formula
- Maximum entropy principle

ENTROPY OF BERNOULLI DISTRIBUTION

Let X be Bernoulli / a coin with $\mathbb{P}(X = 1) = s$ and $\mathbb{P}(X = 0) = 1 - s$.

$$H(X) = -s \cdot \log_2(s) - (1 - s) \cdot \log_2(1 - s).$$



We note: If the coin is deterministic, so $s = 1$ or $s = 0$, then $H(s) = 0$; $H(s)$ is maximal for $s = 0.5$, a fair coin. $H(s)$ increases monotonically the closer we get to $s = 0.5$. This all seems plausible.

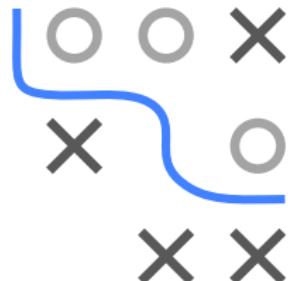
JOINT ENTROPY

- The **joint entropy** of two discrete random variables X and Y is:

$$H(X, Y) = H(p_{X,Y}) = - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log_2(p(x, y))$$

- Intuitively, the joint entropy is a measure of the total uncertainty in the two variables X and Y . In other words, it is simply the entropy of the joint distribution $p(x, y)$.
- There is nothing really new in this definition because $H(X, Y)$ can be considered to be a single vector-valued random variable.
- More generally:

$$H(X_1, X_2, \dots, X_n) = - \sum_{x_1 \in \mathcal{X}_1} \dots \sum_{x_n \in \mathcal{X}_n} p(x_1, x_2, \dots, x_n) \log_2(p(x_1, x_2, \dots, x_n))$$



ENTROPY IS ADDITIVE UNDER INDEPENDENCE

- ⑦ Entropy is additive for independent RVs.

Let X and Y be two independent RVs. Then:

$$\begin{aligned} H(X, Y) &= - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log_2(p(x, y)) \\ &= - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_X(x)p_Y(y) \log_2(p_X(x)p_Y(y)) \\ &= - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_X(x)p_Y(y) \log_2(p_X(x)) + p_X(x)p_Y(y) \log_2(p_Y(y)) \\ &= - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_X(x)p_Y(y) \log_2(p_X(x)) - \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} p_X(x)p_Y(y) \log_2(p_Y(y)) \\ &= - \sum_{x \in \mathcal{X}} p_X(x) \log_2(p_X(x)) - \sum_{y \in \mathcal{Y}} p_Y(y) \log_2(p_Y(y)) = H(X) + H(Y) \end{aligned}$$



THE UNIQUENESS THEOREM

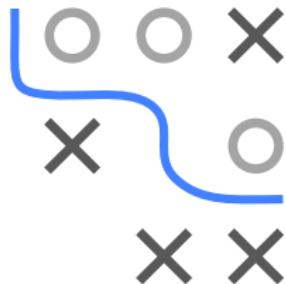
► Khinchin 1957 showed that the only family of functions satisfying

- $H(p)$ is continuous in probabilities $p(x)$
- adding or removing an event with $p(x) = 0$ does not change it
- is additive for independent RVs
- is maximal for a uniform distribution.

is of the following form:

$$H(p) = -\lambda \sum_{x \in \mathcal{X}} p(x) \log p(x)$$

where λ is a positive constant. Setting $\lambda = 1$ and using the binary logarithm gives us the Shannon entropy.



THE MAXIMUM ENTROPY PRINCIPLE

Assume we know M properties about a discrete distribution $p(x)$ on \mathcal{X} , stated as “moment conditions” for functions $g_m(\cdot)$ and scalars α_m :

$$\mathbb{E}[g_m(X)] = \sum_{x \in \mathcal{X}} g_m(x)p(x) = \alpha_m \text{ for } m = 0, \dots, M$$

Maximum entropy principle ▶ Jaynes 2003 : Among all feasible distributions satisfying the constraints, choose the one with maximum entropy!

- Motivation: ensure no unwarranted assumptions on $p(x)$ are made beyond what we know.
- MEP follows similar logic to Occam's razor and principle of insufficient reason



THE MAXIMUM ENTROPY PRINCIPLE

Can be solved via Lagrangian multipliers (here with base e)

$$L(p(x), (\lambda_m)_{m=0}^M) = - \sum_{x \in \mathcal{X}} p(x) \log(p(x)) + \lambda_0 \left(\sum_{x \in \mathcal{X}} p(x) - 1 \right) + \sum_{m=1}^M \lambda_m \left(\sum_{x \in \mathcal{X}} g_m(x)p(x) - \alpha_m \right)$$

Finding critical points $p^*(x)$:

$$\frac{\partial L}{\partial p(x)} = -\log(p(x)) - 1 + \lambda_0 + \sum_{m=1}^M \lambda_m g_m(x) \stackrel{!}{=} 0 \iff p^*(x) = \exp(\lambda_0 - 1) \exp \left(\sum_{m=1}^M \lambda_m g_m(x) \right)$$

This is a maximum as $-1/p(x) < 0$. Since probs must sum to 1 we get

$$1 \stackrel{!}{=} \sum_{x \in \mathcal{X}} p^*(x) = \frac{1}{\exp(1 - \lambda_0)} \sum_{x \in \mathcal{X}} \exp \left(\sum_{m=1}^M \lambda_m g_m(x) \right) \Rightarrow \exp(1 - \lambda_0) = \sum_{x \in \mathcal{X}} \exp \left(\sum_{m=1}^M \lambda_m g_m(x) \right)$$

Plugging $\exp(1 - \lambda_0)$ into $p^*(x)$ we obtain the constrained maxent distribution:

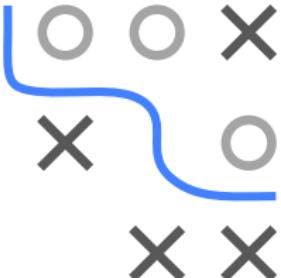
$$p^*(x) = \frac{\exp \sum_{m=1}^M \lambda_m g_m(x)}{\sum_{x \in \mathcal{X}} \exp \sum_{m=1}^M \lambda_m g_m(x)}$$



THE MAXIMUM ENTROPY PRINCIPLE

We now have: functional form of our distribution, up to M unknowns, the λ_m . But also: M equations, the moment conditions. So we can solve.

Example: Consider discrete RV representing a six-sided die roll and the moment condition $\mathbb{E}(X) = 4.8$. What is the maxent distribution?



- Condition means $g_1(x) = x$, $\alpha_1 = 4.8$. Then for some λ solution is

$$p^*(x) = \frac{\exp(\lambda g(x))}{\sum_{j=1}^6 \exp(\lambda g(x_j))} = \frac{\exp(\lambda x)}{\sum_{j=1}^6 \exp(\lambda x_j)}$$

- Inserting into moment condition and solving (numerically) for λ :

$$4.8 \stackrel{!}{=} \sum_{j=1}^6 x_j p^*(x_j) = \frac{e^\lambda + \dots + 6(e^\lambda)^6}{e^\lambda + \dots + (e^\lambda)^6} \Rightarrow \lambda \approx 0.5141$$

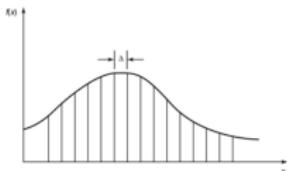
x	1	2	3	4	5	6
$p^*(x)$	3.22%	5.38%	9.01%	15.06%	25.19%	42.13%

Introduction to Machine Learning

Differential Entropy



Learning goals



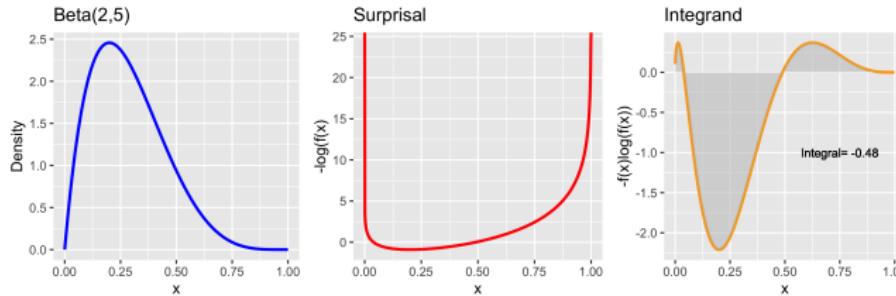
- Know that the entropy expresses expected information for continuous RVs
- Know the basic properties of the differential entropy

DIFFERENTIAL ENTROPY

- For a continuous random variable X with density function $f(x)$ and support \mathcal{X} , the analogue of entropy is **differential entropy**:

$$h(X) := h(f) := -\mathbb{E}[\log(f(x))] = - \int_{\mathcal{X}} f(x) \log(f(x)) dx$$

- The base of the log is again somewhat arbitrary, and we could either use 2 (and measure in bits) or e (to measure in nats).
- The integral above does not necessarily exist for all densities.
- Differential entropy lacks the non-negativeness of discrete entropy:
 $h(X) < 0$ is possible as $f(x) > 1$ is possible:



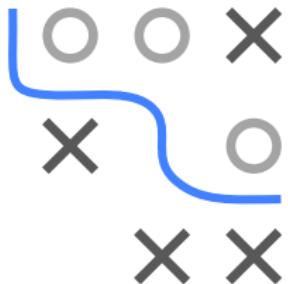
The diffent. is given by the integral:
 $h(X) = -0.48$.



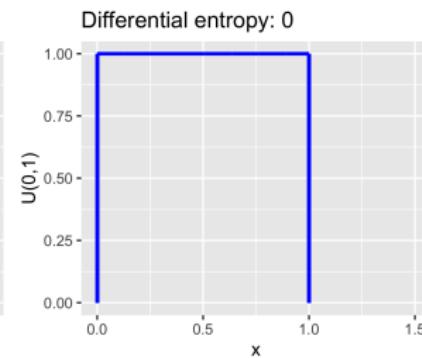
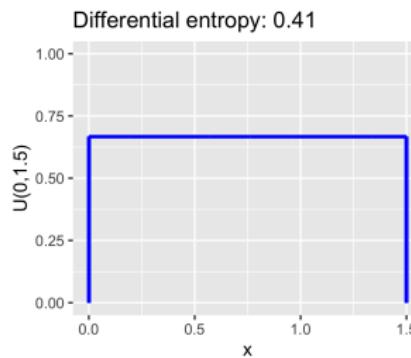
DIFF. ENTROPY OF UNIFORM DISTRIBUTION

Let X be a uniform random variable on $[0, a]$.

$$\begin{aligned} h(X) &= - \int_0^a f(x) \log(f(x)) dx \\ &= - \int_0^a \frac{1}{a} \log\left(\frac{1}{a}\right) dx = \log(a) \end{aligned}$$



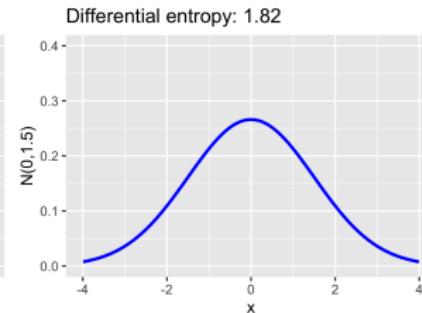
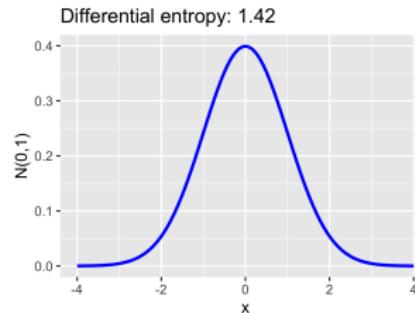
- For $a < 1$, $h(X) < 0$.



DIFF. ENTROPY OF GAUSSIAN

Let $X \sim \mathcal{N}(\mu, \sigma^2)$ and let us measure in nats:

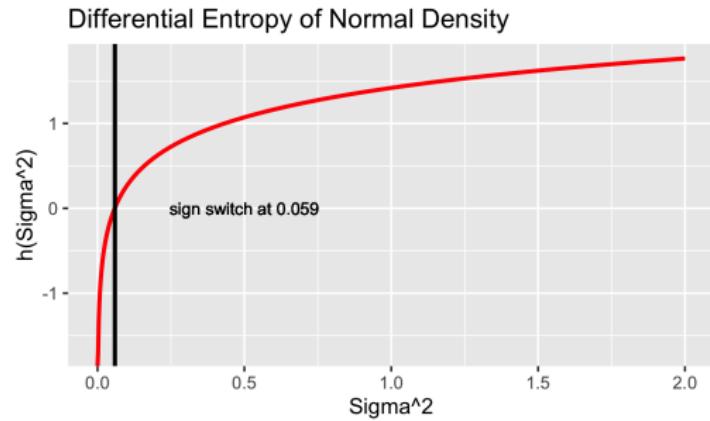
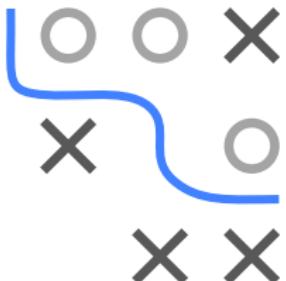
$$\begin{aligned} h(X) &= - \int_{\mathbb{R}} f(x) \log(f(x)) dx = - \int_{\mathbb{R}} f(x) \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \right) dx \\ &= - \int_{\mathbb{R}} f(x) \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) dx + \int_{\mathbb{R}} f(x) \frac{(x-\mu)^2}{2\sigma^2} dx \\ &= - \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) \underbrace{\int_{\mathbb{R}} f(x) dx}_{=1} + \frac{1}{2\sigma^2} \underbrace{\int_{\mathbb{R}} f(x)(x-\mu)^2 dx}_{=: \sigma^2} \\ &= \frac{1}{2} \log(2\pi\sigma^2) + \frac{1}{2} = \log(\sigma\sqrt{2\pi e}) \end{aligned}$$



DIFF. ENTROPY OF GAUSSIAN

$$h(X) = - \int_{\mathbb{R}} f(x) \log(f(x)) dx = \log(\sigma\sqrt{2\pi e})$$

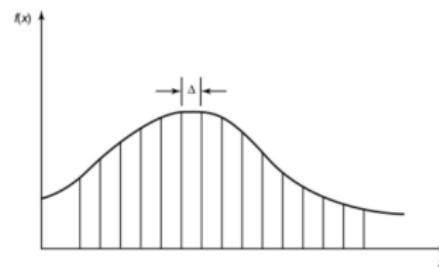
- $h(X)$ is not a function of μ (see translation invariance later).
- As σ^2 increases, the differential entropy also increases.
- For $\sigma^2 < \frac{1}{2\pi e} \approx 0.059$, it is negative.



DIFF. ENTROPY VS. DISCRETE

It is not so simple as to characterize $h(X)$ as a straightforward generalization of $H(X)$ of a limiting process. Consider the quantized random variable X^Δ , which is defined by

$$X^\Delta = x_i \quad \text{if} \quad i\Delta \leq X < (i+1)\Delta$$



If the density $f(x)$ of the random variable X is Riemann-integrable, then

$$H(X^\Delta) + \log(\Delta) \rightarrow h(X) \text{ as } \Delta \rightarrow 0.$$

Thus, the entropy of an n -bit quantization of a continuous random variable X is approximately $h(X) + n$.

JOINT DIFFERENTIAL ENTROPY

- For a continuous random vector X with density function $f(x)$ and support \mathcal{X} , differential entropy is also defined as:

$$h(X) = h(X_1, \dots, X_n) = h(f) = - \int_{\mathcal{X}} f(x) \log(f(x)) dx$$

- Hence this also defines the joint differential entropy for a set of continuous RVs.

Entropy of a multivariate normal distribution: If $X \sim N(\mu, \Sigma)$ is multivariate Gaussian, then

$$h(X) = \frac{1}{2} \log(2\pi e)^n |\Sigma| \quad (\text{nats})$$



PROPERTIES OF DIFFERENTIAL ENTROPY

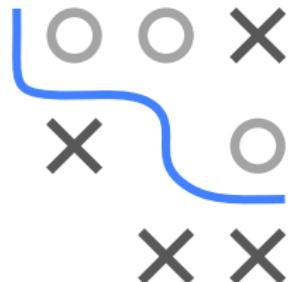
- ① $h(f)$ can be negative.
- ② $h(f)$ is additive for independent RVs.
- ③ $h(f)$ is maximized by the multivariate normal, if we restrict to all distributions with the same (co)variance, so
$$h(X) \leq \frac{1}{2} \log(2\pi e)^n |\Sigma|.$$
- ④ $h(f)$ is maximized by the continuous uniform distribution for a random variable with a fixed range.
- ⑤ Translation-invariant, $h(X + a) = h(X)$.
- ⑥ $h(aX) = h(X) + \log |a|$.
- ⑦ $h(AX) = h(X) + \log |A|$ for random vectors and matrix A.

3) and 4) are slightly involved to prove, while the other properties are relatively straightforward to show

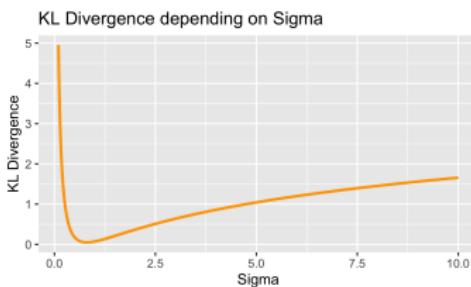


Introduction to Machine Learning

Kullback-Leibler Divergence



Learning goals



- Know the KL divergence as distance between distributions
- Understand KL as expected log-difference
- Understand how KL can be used as loss
- Understand that KL is equivalent to the expected likelihood ratio

KULLBACK-LEIBLER DIVERGENCE

We now want to establish a measure of distance between (discrete or continuous) distributions with the same support for $X \sim p(X)$:

$$D_{KL}(p\|q) = \mathbb{E}_{X \sim p} \left[\log \frac{p(X)}{q(X)} \right] = \sum_{x \in \mathcal{X}} p(x) \cdot \log \frac{p(x)}{q(x)},$$

or:

$$D_{KL}(p\|q) = \mathbb{E}_{X \sim p} \left[\log \frac{p(X)}{q(X)} \right] = \int_{x \in \mathcal{X}} p(x) \cdot \log \frac{p(x)}{q(x)} dx.$$

In the above definition, we use the conventions that $0 \log(0/0) = 0$, $0 \log(0/q) = 0$ and $p \log(p/0) = \infty$ (based on continuity arguments where $p \rightarrow 0$). Thus, if there is any realization $x \in \mathcal{X}$ such that $p(x) > 0$ and $q(x) = 0$, then $D_{KL}(p\|q) = \infty$.



KULLBACK-LEIBLER DIVERGENCE / 2

$$D_{KL}(p\|q) = \mathbb{E}_{X \sim p} \left[\log \frac{p(X)}{q(X)} \right]$$

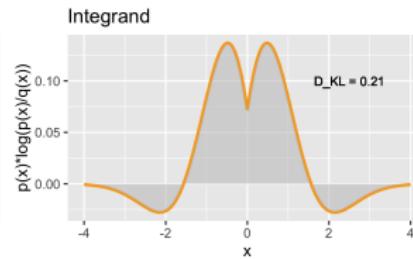
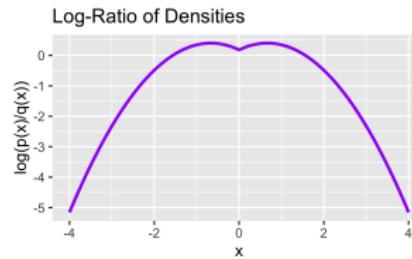
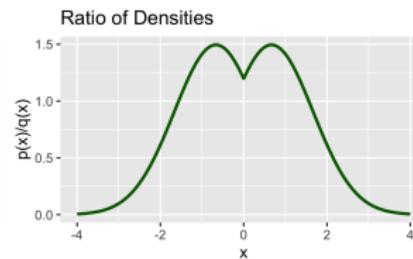
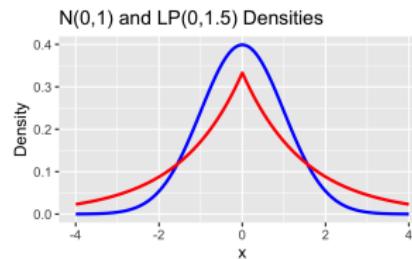
- What is the intuition behind this formula?
- We will soon see that KL has quite some value in measuring “differences” but is not a true distance.
- We already see that the formula is not symmetric and it often makes sense to think of p as the first or original form of the data, and q as something that we want to measure the quality of with reference to p .



KL-DIVERGENCE EXAMPLE

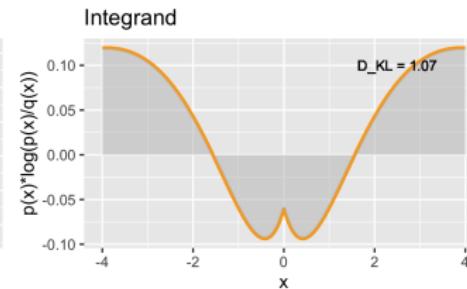
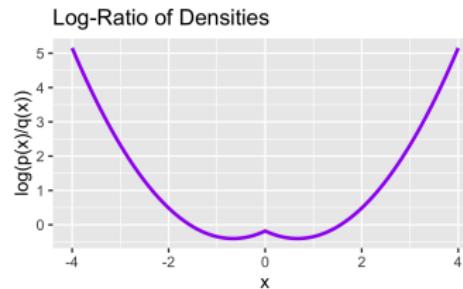
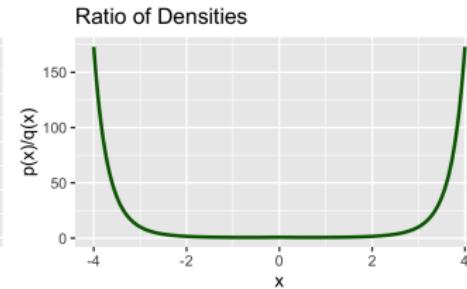
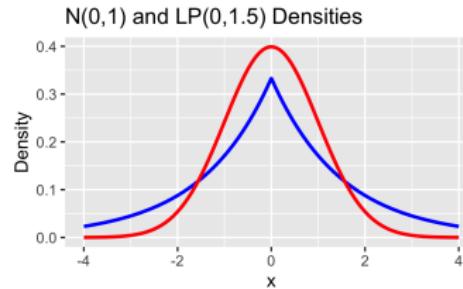
KL divergence between $p(x) = N(0, 1)$ and $q(x) = LP(0, 1.5)$ given by

$$D_{KL}(p\|q) = \int_{x \in \mathcal{X}} p(x) \cdot \log \frac{p(x)}{q(x)}.$$



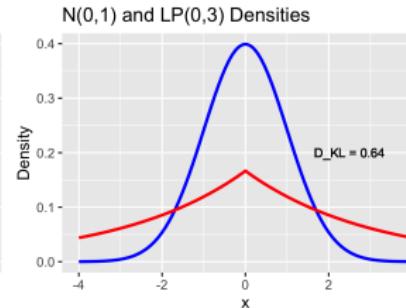
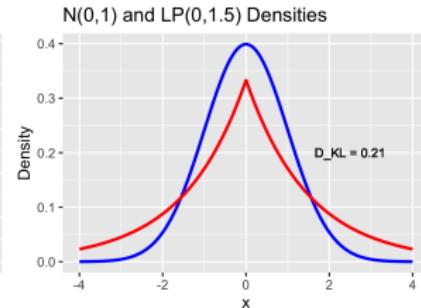
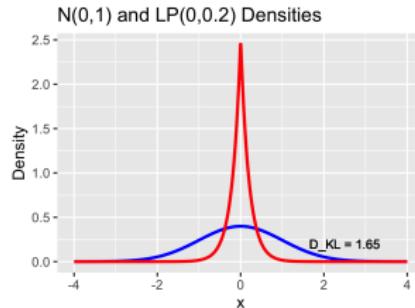
KL-DIVERGENCE EXAMPLE

KL divergence between $p(x) = LP(0, 1.5)$ and $q(x) = N(0, 1)$ is different since KL not symmetric



KL-DIVERGENCE EXAMPLE

KL divergence of $p(x) = N(0, 1)$ and $q(x) = LP(0, \sigma)$ for varying σ



INFORMATION INEQUALITY

$D_{KL}(p\|q) \geq 0$ holds always true for any pair of distributions, and holds with equality if and only if $p = q$.

We use Jensen's inequality. Let A be the support of p :

$$\begin{aligned} -D_{KL}(p\|q) &= - \sum_{x \in A} p(x) \log \frac{p(x)}{q(x)} \\ &= \sum_{x \in A} p(x) \log \frac{q(x)}{p(x)} \\ &\leq \log \sum_{x \in A} p(x) \frac{q(x)}{p(x)} \\ &\leq \log \sum_{x \in \mathcal{X}} q(x) = \log(1) = 0 \end{aligned}$$

As \log is strictly concave, Jensen also tells us that equality can only happen if $q(x)/p(x)$ is constant everywhere. That implies $p = q$.



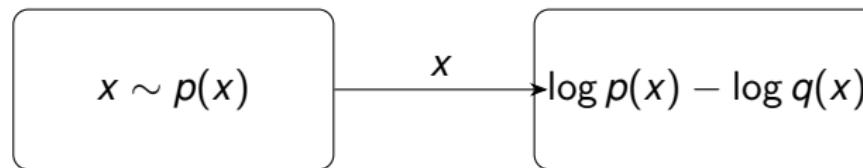
KL AS LOG-DIFFERENCE

Suppose that data is being generated from an unknown distribution $p(x)$ and we model $p(x)$ using an approximating distribution $q(x)$.

First, we could simply see KL as the expected log-difference between $p(x)$ and $q(x)$:

$$D_{KL}(p\|q) = \mathbb{E}_{X \sim p}[\log(p(X)) - \log(q(X))].$$

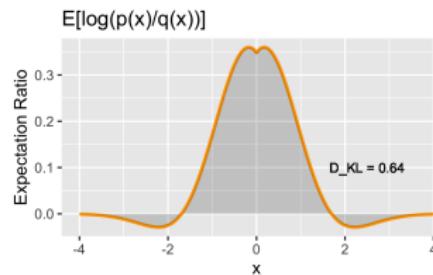
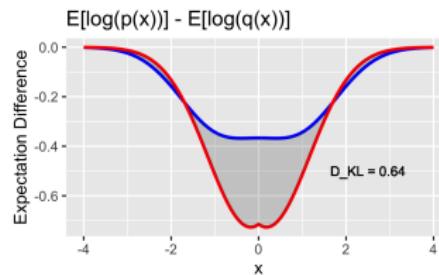
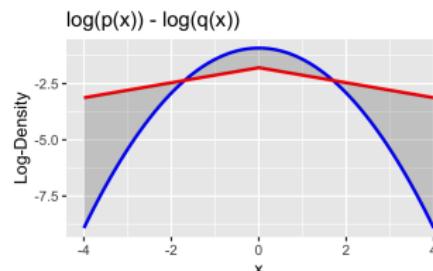
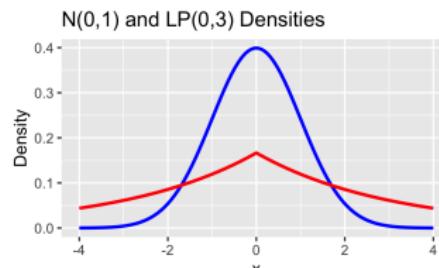
This is why we integrate out with respect to the data distribution p . A “good” approximation $q(x)$ should minimize the difference to $p(x)$.



KL AS LOG-DIFFERENCE / 2

Let $p(x) = N(0, 1)$ and $q(x) = LP(0, 3)$. Observe

$$\begin{aligned} D_{KL}(p\|q) &= \mathbb{E}_{X \sim p}[\log(p(X)) - \log(q(X))] \\ &= \mathbb{E}_{X \sim p}[\log(p(X))] - \mathbb{E}_{X \sim p}[\log(q(X))]. \end{aligned}$$

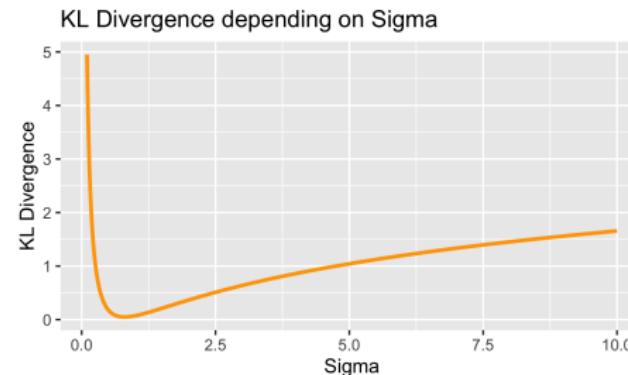
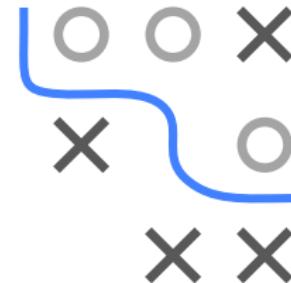


KL IN FITTING

In machine learning, KL divergence is commonly used to quantify how different one distribution is from another.

Because KL quantifies the difference between distributions, it can be used as a loss function between distributions.

In our example, we investigated the KL between $p = N(0, 1)$ and $q = LP(0, \sigma)$. Now, we identify an optimal σ which minimizes the KL.



KL AS LIKELIHOOD RATIO

- Let us assume we have some data and want to figure out whether $p(x)$ or $q(x)$ matches it better.
- How do we usually do that in stats? Likelihood ratio!



$$LR = \prod_i \frac{p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})} \quad LLR = \sum_i \log \frac{p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})}$$

If for $\mathbf{x}^{(i)}$ we have $p(\mathbf{x}^{(i)})/q(\mathbf{x}^{(i)}) > 1$, then p seems better, for $p(\mathbf{x}^{(i)})/q(\mathbf{x}^{(i)}) < 1$ q seems better.

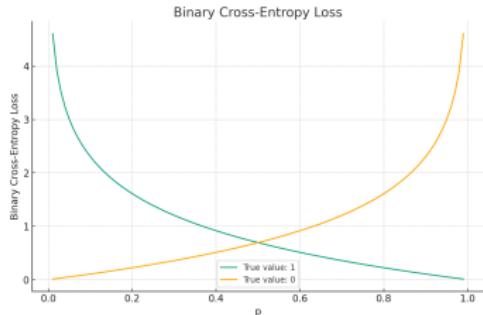
- Now assume that the data is generated by p . Can also ask:
- "How to quantify how much better does p fit than q , on average?"

$$\mathbb{E}_p \left[\log \frac{p(X)}{q(X)} \right]$$

That expected LLR is really KL!

Introduction to Machine Learning

Cross-Entropy and KL



Learning goals

- Know the cross-entropy
- Understand the connection between entropy, cross-entropy, and KL divergence

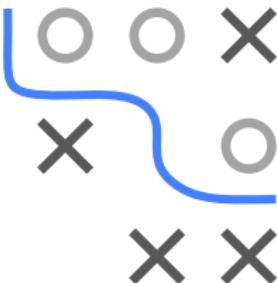


CROSS-ENTROPY - DISCRETE CASE

Cross-entropy measures the average amount of information required to represent an event from one distribution p using a predictive scheme based on another distribution q (assume they have the same domain \mathcal{X} as in KL).

$$H(p\|q) = \sum_{x \in \mathcal{X}} p(x) \log \left(\frac{1}{q(x)} \right) = - \sum_{x \in \mathcal{X}} p(x) \log (q(x)) = -\mathbb{E}_{X \sim p} [\log(q(X))]$$

For now, we accept the formula as-is. More on the underlying intuition follows in the content on inf. theory for ML and sourcecoding.



- Entropy = Avg. amount of information if we optimally encode p
- Cross-Entropy = Avg. amount of information if we suboptimally encode p with q
- $D_{KL}(p\|q)$: Difference between the two
- $H(p\|q)$ sometimes also denoted as $H_q(p)$ to set it apart from KL

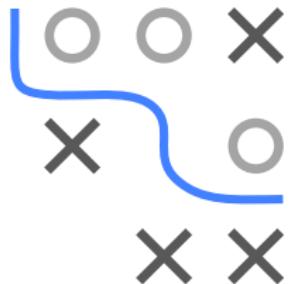
CROSS-ENTROPY - DISCRETE CASE / 2

We can summarize this also through this identity:

$$H(p\|q) = H(p) + D_{KL}(p\|q)$$

This is because:

$$\begin{aligned} H(p) + D_{KL}(p\|q) &= - \sum_{x \in \mathcal{X}} p(x) \log p(x) + \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \\ &= \sum_{x \in \mathcal{X}} p(x) (-\log p(x) + \log p(x) - \log q(x)) \\ &= - \sum_{x \in \mathcal{X}} p(x) \log q(x) = H(p\|q) \end{aligned}$$



CROSS-ENTROPY - CONTINUOUS CASE

For continuous density functions $p(x)$ and $q(x)$:

$$H(p\|q) = \int p(x) \log \left(\frac{1}{q(x)} \right) dx = - \int p(x) \log (q(x)) dx = -\mathbb{E}_{X \sim p} [\log(q(X))]$$

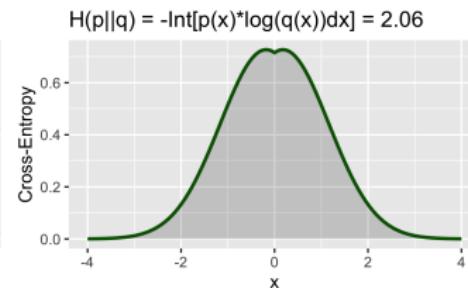
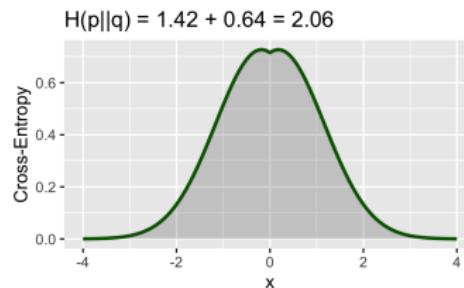
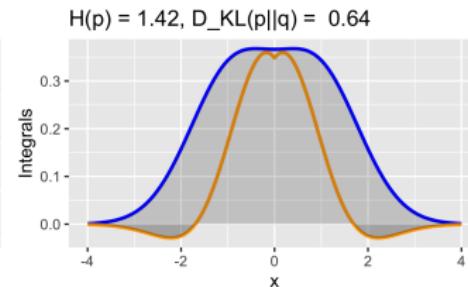
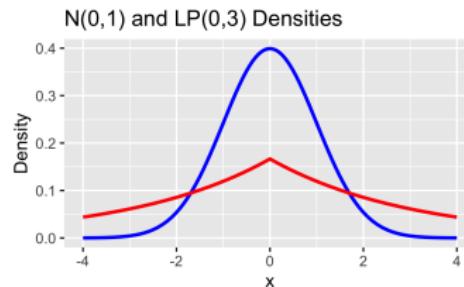
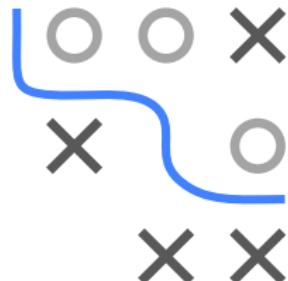


- It is not symmetric.
- As for the discrete case, $H(p\|q) = h(p) + D_{KL}(p\|q)$ holds.
- Can now become negative, as the $h(p)$ can be negative!

CROSS-ENTROPY EXAMPLE

Let $p(x) = N(0, 1)$ and $q(x) = LP(0, 3)$. We can visualize

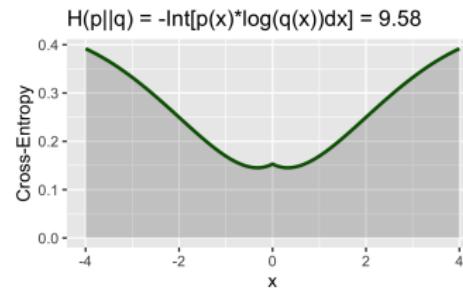
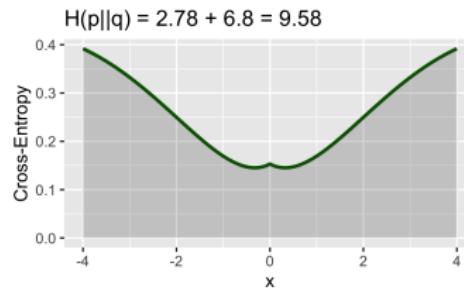
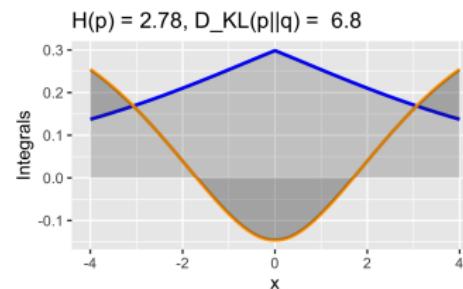
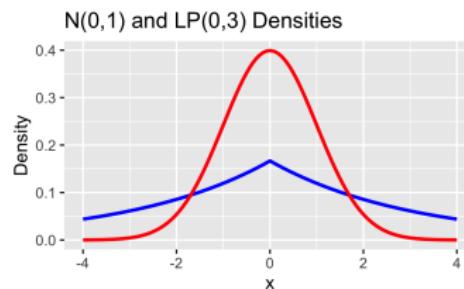
$$H(p\|q) = H(p) + D_{KL}(p\|q)$$



CROSS-ENTROPY EXAMPLE

Let $p(x) = LP(0, 3)$ and $q(x) = N(0, 1)$. We can visualize

$$H(p\|q) = H(p) + D_{KL}(p\|q)$$



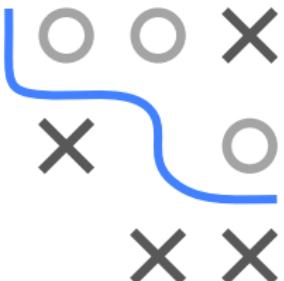
PROOF: MAXIMUM OF DIFFERENTIAL ENTROPY

Claim: For a given variance, the continuous distribution that maximizes differential entropy is the Gaussian.

Proof: Let $g(x)$ be a Gaussian with mean μ and variance σ^2 and $f(x)$ an arbitrary density function with the same variance. Since differential entropy is translation invariant, we can assume $f(x)$ and $g(x)$ have the same mean.

The KL divergence (which is non-negative) between $f(x)$ and $g(x)$ is:

$$\begin{aligned} 0 \leq D_{KL}(f\|g) &= -h(f) + H(p\|q) \\ &= -h(f) - \int_{-\infty}^{\infty} f(x) \log(g(x)) dx \end{aligned} \tag{1}$$



PROOF: MAXIMUM OF DIFFERENTIAL ENTROPY

/ 2

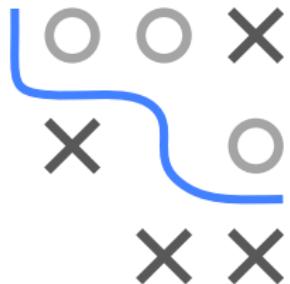
The second term in (1) is,

$$\begin{aligned} \int_{-\infty}^{\infty} f(x) \log(g(x)) dx &= \int_{-\infty}^{\infty} f(x) \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right) dx \\ &= \int_{-\infty}^{\infty} f(x) \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) dx + \log(e) \int_{-\infty}^{\infty} f(x) \left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx \\ &= -\frac{1}{2} \log(2\pi\sigma^2) - \log(e) \frac{\sigma^2}{2\sigma^2} = -\frac{1}{2} (\log(2\pi\sigma^2) + \log(e)) \\ &= -\frac{1}{2} \log(2\pi e \sigma^2) = -h(g), \end{aligned} \tag{2}$$

where the last equality follows from the normal distribution example of the entropy chapter. Combining (1) and (2) results in

$$h(g) - h(f) \geq 0$$

with equality when $f(x) = g(x)$ (following from the properties of Kullback-Leibler divergence).

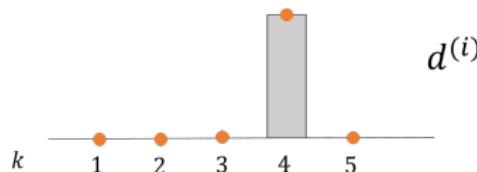


Introduction to Machine Learning

Information Theory for Machine Learning



Learning goals



- Minimizing KL = maximizing log-likelihood
- Minimizing KL = minimizing cross-entropy
- Minimizing CE between modeled and observed probabilities = log-loss minimization

KL VS MAXIMUM LIKELIHOOD

Minimizing KL between the true distribution $p(x)$ and approximating model $q(x|\theta)$ is equivalent to maximizing the log-likelihood.

$$\begin{aligned} D_{KL}(p\|q_\theta) &= \mathbb{E}_{X \sim p} \left[\log \frac{p(x)}{q(x|\theta)} \right] \\ &= \mathbb{E}_{X \sim p} \log p(x) - \mathbb{E}_{X \sim p} \log q(x|\theta) \end{aligned}$$

as first term above does not depend on θ . Therefore,

$$\begin{aligned} \arg \min_{\theta} D_{KL}(p\|q_\theta) &= \arg \min_{\theta} -\mathbb{E}_{X \sim p} \log q(x|\theta) \\ &= \arg \max_{\theta} \mathbb{E}_{X \sim p} \log q(x|\theta) \end{aligned}$$

For a finite dataset of n samples from p , this is approximated as

$$\arg \max_{\theta} \mathbb{E}_{X \sim p} \log q(x|\theta) \approx \arg \max_{\theta} \frac{1}{n} \sum_{i=1}^n \log q(\mathbf{x}^{(i)}|\theta).$$

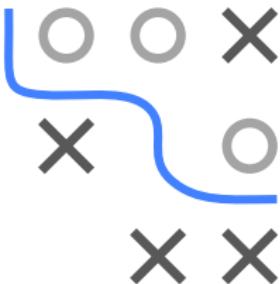
This also directly implies an equivalence to risk minimization!



KL VS CROSS-ENTROPY

From this here we can see much more:

$$\arg \min_{\theta} D_{KL}(p\|q_{\theta}) = \arg \min_{\theta} -\mathbb{E}_{X \sim p} \log q(x|\theta) = \arg \min_{\theta} H(p\|q_{\theta})$$

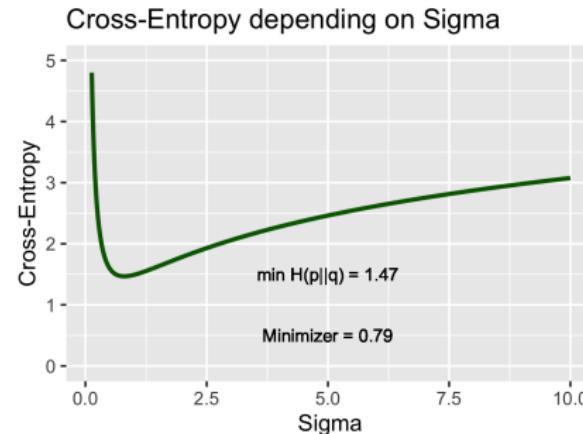
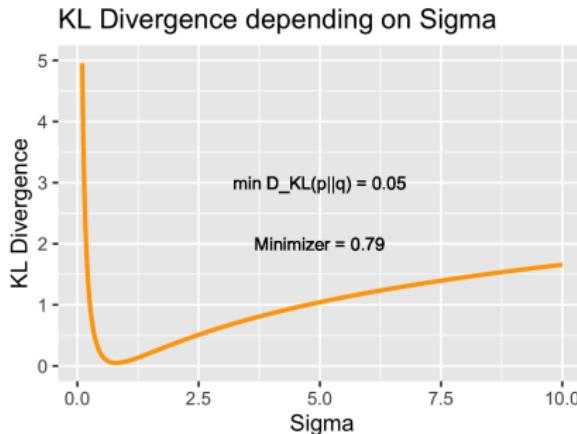


- So minimizing KL is the same as minimizing CE, is the same as maximum likelihood!
- We could now motivate CE as the "relevant" term that you have to minimize when you minimize KL - after you drop $\mathbb{E}_p \log p(x)$, which is simply the neg. entropy $H(p)$!
- Or we could say: CE between p and q is simply the expected negative log-likelihood of q , when our data comes from p !

KL VS CROSS-ENTROPY EXAMPLE

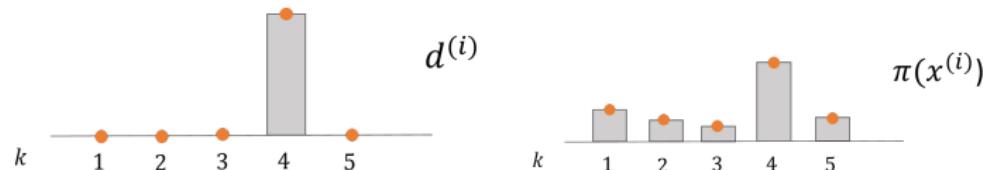
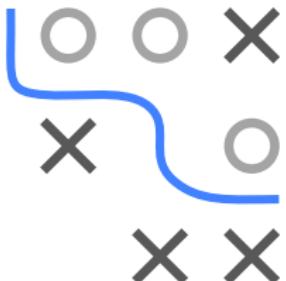
Let $p(x) = N(0, 1)$ and $q(x) = LP(0, \sigma)$ and consider again

$$\arg \min_{\theta} D_{KL}(p\|q_{\theta}) = \arg \min_{\theta} -\mathbb{E}_{X \sim p} \log q(x|\theta) = \arg \min_{\theta} H(p\|q_{\theta})$$



CROSS-ENTROPY VS. LOG-LOSS

- Consider a multi-class classification task with dataset $\mathcal{D} = ((\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}))$.
- For g classes, each $y^{(i)}$ can be one-hot-encoded as a vector $d^{(i)}$ of length g . $d^{(i)}$ can be interpreted as a categorical distribution which puts all its probability mass on the true label $y^{(i)}$ of $\mathbf{x}^{(i)}$.
- $\pi(\mathbf{x}^{(i)}|\theta)$ is the probability output vector of the model, and also a categorical distribution over the classes.

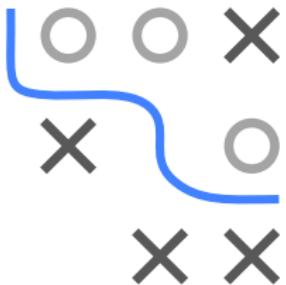


CROSS-ENTROPY VS. LOG-LOSS / 2

To train the model, we minimize KL between $d^{(i)}$ and $\pi(\mathbf{x}^{(i)}|\theta)$:

$$\arg \min_{\theta} \sum_{i=1}^n D_{KL}(d^{(i)} \| \pi(\mathbf{x}^{(i)}|\theta)) = \arg \min_{\theta} \sum_{i=1}^n H(d^{(i)} \| \pi(\mathbf{x}^{(i)}|\theta))$$

We see that this is equivalent to log-loss risk minimization!



$$\begin{aligned} R &= \sum_{i=1}^n H(d^{(i)} \| \pi_k(\mathbf{x}^{(i)}|\theta)) \\ &= \sum_{i=1}^n \left(- \sum_k d_k^{(i)} \log \pi_k(\mathbf{x}^{(i)}|\theta) \right) \\ &= \sum_{i=1}^n \underbrace{\left(- \sum_{k=1}^g [y^{(i)} = k] \log \pi_k(\mathbf{x}^{(i)}|\theta) \right)}_{\text{log loss}} \\ &= \sum_{i=1}^n (-\log \pi_{y^{(i)}}(\mathbf{x}^{(i)}|\theta)) \end{aligned}$$

CROSS-ENTROPY VS. BERNOULLI LOSS

For completeness sake:

Let us use the Bernoulli loss for binary classification:

$$L(y, \pi(\mathbf{x})) = -y \log(\pi(\mathbf{x})) - (1 - y) \log(1 - \pi(\mathbf{x}))$$

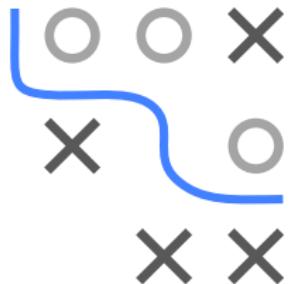


If p represents a $\text{Ber}(y)$ distribution (so deterministic, where the true label receives probability mass 1) and we also interpret $\pi(\mathbf{x})$ as a Bernoulli distribution $\text{Ber}(\pi(\mathbf{x}))$, the Bernoulli loss $L(y, \pi(\mathbf{x}))$ is the cross-entropy $H(p||\pi(\mathbf{x}))$.

ENTROPY AS PREDICTION LOSS

Assume log-loss for a situation where you only model with a constant probability vector π . We know the optimal model under that loss:

$$\pi_k = \frac{n_k}{n} = \frac{\sum_{i=1}^n [y^{(i)} = k]}{n}$$



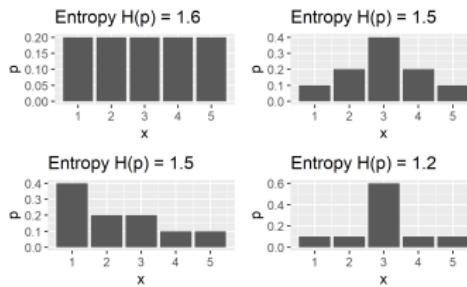
What is the (average) risk of that minimal constant model?

$$\begin{aligned}\mathcal{R} &= \frac{1}{n} \sum_{i=1}^n \left(- \sum_{k=1}^g [y^{(i)} = k] \log \pi_k \right) = - \frac{1}{n} \sum_{k=1}^g \sum_{i=1}^n [y^{(i)} = k] \log \pi_k \\ &= - \sum_{k=1}^g \frac{n_k}{n} \log \pi_k = - \sum_{k=1}^g \pi_k \log \pi_k = H(\pi)\end{aligned}$$

So entropy is the (average) risk of the optimal "observed class frequency" model under log-loss!

Introduction to Machine Learning

Joint Entropy and Mutual Information I



Learning goals

- Know the joint entropy
- Know conditional entropy as remaining uncertainty
- Know mutual information as the amount of information of an RV obtained by another



JOINT ENTROPY

- Recap: The **joint entropy** of two discrete RVs X and Y with joint pmf $p(x, y)$ is:

$$H(X, Y) = - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log(p(x, y)),$$

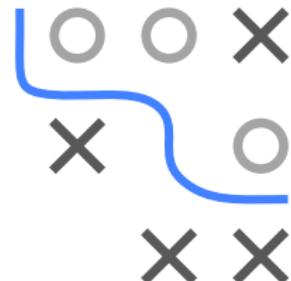
which can also be expressed as

$$H(X, Y) = -\mathbb{E} [\log(p(X, Y))].$$

- For continuous RVs X and Y with joint density $p(x, y)$, the differential joint entropy is:

$$h(X, Y) = - \int_{\mathcal{X} \times \mathcal{Y}} p(x, y) \log p(x, y) dx dy$$

For the rest of the section we will stick to the discrete case. Pretty much everything we show and discuss works in a completely analogous manner for the continuous case - if you change sums to integrals.



CONDITIONAL ENTROPY

- The **conditional entropy** $H(Y|X)$ quantifies the uncertainty of Y that remains if the outcome of X is given.
- $H(Y|X)$ is defined as the expected value of the entropies of the conditional distributions, averaged over the conditioning RV.
- If $(X, Y) \sim p(x, y)$, the conditional entropy $H(Y|X)$ is defined as

$$\begin{aligned} H(Y|X) &= \mathbb{E}_X[H(Y|X = x)] = \sum_{x \in \mathcal{X}} p(x)H(Y|X = x) \\ &= - \sum_{x \in \mathcal{X}} p(x) \sum_{y \in \mathcal{Y}} p(y|x) \log p(y|x) \\ &= - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(y|x) \\ &= -\mathbb{E} [\log p(Y|X)]. \end{aligned}$$

- For the continuous case with density f we have

$$h(Y|X) = - \int f(x, y) \log f(x|y) dx dy.$$



CHAIN RULE FOR ENTROPY

The **chain rule for entropy** is analogous to the chain rule for probability and derives directly from it.

$$H(X, Y) = H(X) + H(Y|X)$$

Proof:
$$\begin{aligned} H(X, Y) &= - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(x, y) \\ &= - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(x)p(y|x) \\ &= - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(x) - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(y|x) \\ &= - \sum_{x \in \mathcal{X}} p(x) \log p(x) - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(y|x) \\ &= H(X) + H(Y|X) \end{aligned}$$

n-variable version:

$$H(X_1, X_2, \dots, X_n) = \sum_{i=1}^n H(X_i | X_{i-1}, \dots, X_1).$$



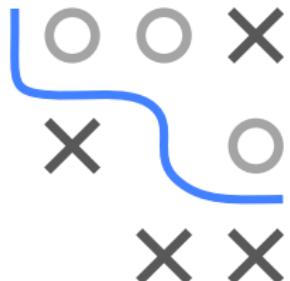
JOINT AND CONDITIONAL ENTROPY

The following relations hold:

$$H(X, X) = H(X)$$

$$H(X|X) = 0$$

$$H((X, Y)|Z) = H(X|Z) + H(Y|(X, Z))$$



Which can all be trivially derived from the previous considerations.

Furthermore, if $H(X|Y) = 0$, then X is a function of Y , so for all y with $p(y) > 0$, there is only one x with $p(x, y) > 0$. Proof is not hard, but also not completely trivial.

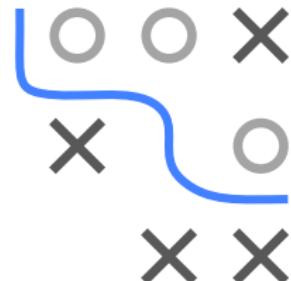
MUTUAL INFORMATION

- The MI describes the amount of info about one RV obtained through another RV or how different their joint distribution is from pure independence.
- Consider two RVs X and Y with a joint pmf $p(x, y)$ and marginal pmfs $p(x)$ and $p(y)$. The MI $I(X; Y)$ is the Kullback-Leibler Divergence between the joint distribution and the product distribution $p(x)p(y)$:

$$\begin{aligned} I(X; Y) &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\ &= D_{KL}(p(x, y) \| p(x)p(y)) \\ &= \mathbb{E}_{p(x, y)} \left[\log \frac{p(X, Y)}{p(X)p(Y)} \right]. \end{aligned}$$

- For two continuous random variables with joint density $f(x, y)$:

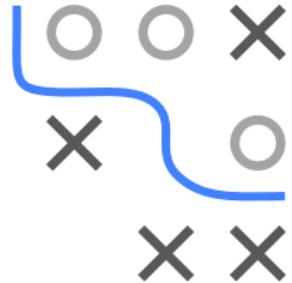
$$I(X; Y) = \int f(x, y) \log \frac{f(x, y)}{f(x)f(y)} dx dy.$$



MUTUAL INFORMATION

We can rewrite the definition of mutual information $I(X; Y)$ as

$$\begin{aligned} I(X; Y) &= \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\ &= \sum_{x,y} p(x, y) \log \frac{p(x|y)}{p(x)} \\ &= - \sum_{x,y} p(x, y) \log p(x) + \sum_{x,y} p(x, y) \log p(x|y) \\ &= - \sum_x p(x) \log p(x) - \left(- \sum_{x,y} p(x, y) \log p(x|y) \right) \\ &= H(X) - H(X|Y). \end{aligned}$$



So, $I(X; Y)$ is reduction in uncertainty of X due to knowledge of Y .

MUTUAL INFORMATION

The following relations hold:

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

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

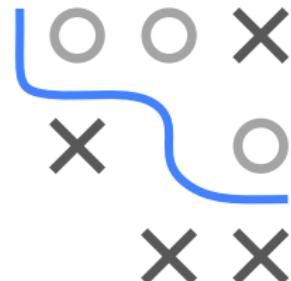
$I(X; Y) \leq \min\{H(X), H(Y)\}$ if X, Y are discrete RVs

$$I(X; Y) = H(X) + H(Y) - H(X, Y)$$

$$I(X; Y) = I(Y; X)$$

$$I(X; X) = H(X)$$

All of the above are trivial to prove.



MUTUAL INFORMATION - EXAMPLE

Let X, Y have the following joint distribution:

	X_1	X_2	X_3	X_4
Y_1	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$
Y_2	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{32}$	$\frac{1}{32}$
Y_3	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$
Y_4	$\frac{1}{4}$	0	0	0



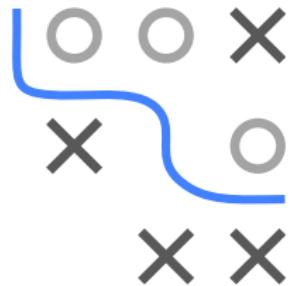
Marginal distribution of X is $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$ and marginal distribution of Y is $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$, and hence $H(X) = \frac{7}{4}$ bits and $H(Y) = 2$ bits.

MUTUAL INFORMATION - EXAMPLE / 2

The conditional entropy $H(X|Y)$ is given by:

$$\begin{aligned} H(X|Y) &= \sum_{i=1}^4 p(Y=i)H(X|Y=i) \\ &= \frac{1}{4}H\left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right) + \frac{1}{4}H\left(\frac{1}{4}, \frac{1}{2}, \frac{1}{8}, \frac{1}{8}\right) \\ &\quad + \frac{1}{4}H\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right) + \frac{1}{4}H(1, 0, 0, 0) \\ &= \frac{1}{4} \cdot \frac{7}{4} + \frac{1}{4} \cdot \frac{7}{4} + \frac{1}{4} \cdot 2 + \frac{1}{4} \cdot 0 \\ &= \frac{11}{8} \text{ bits.} \end{aligned}$$

Similarly, $H(Y|X) = \frac{13}{8}$ bits and $H(X, Y) = \frac{27}{8}$ bits.

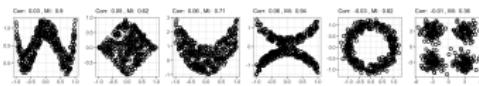


Introduction to Machine Learning

Joint Entropy and Mutual Information II



Learning goals

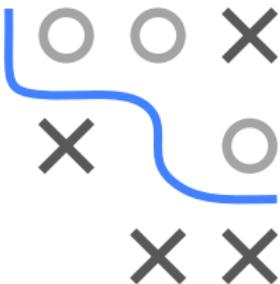


- Know mutual information as the amount of information of an RV obtained by another
- Know properties of MI

MUTUAL INFORMATION - COROLLARIES

Non-negativity of mutual information: For any two random variables, X, Y , $I(X; Y) \geq 0$, with equality if and only if X and Y are independent.

Proof: $I(X; Y) = D_{KL}(p(x, y) \| p(x)p(y)) \geq 0$, with equality if and only if $p(x, y) = p(x)p(y)$ (i.e., X and Y are independent).



Conditioning reduces entropy (information can't hurt):

$$H(X|Y) \leq H(X),$$

with equality if and only if X and Y are independent.

Proof: $0 \leq I(X; Y) = H(X) - H(X|Y)$

Intuitively, the theorem says that knowing another random variable Y can only reduce the uncertainty in X . Note that this is true only on average.

MUTUAL INFORMATION - COROLLARIES / 2

Independence bound on entropy:

$$H(X_1, X_2, \dots, X_n) \leq \sum_{i=1}^n H(X_i),$$

Holds with equality if and only if X_i are jointly independent.



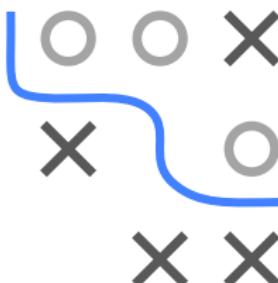
Proof: With chain rule and "conditioning reduces entropy"

$$H(X_1, X_2, \dots, X_n) = \sum_{i=1}^n H(X_i | X_{i-1}, \dots, X_1) \leq \sum_{i=1}^n H(X_i)$$

Equality holds iff X_i is independent of X_{i-1}, \dots, X_1 for all i , so iff all X_i are jointly independent.

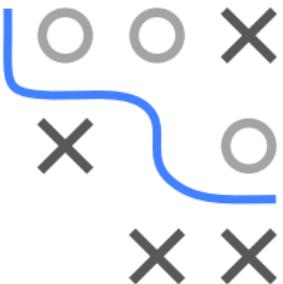
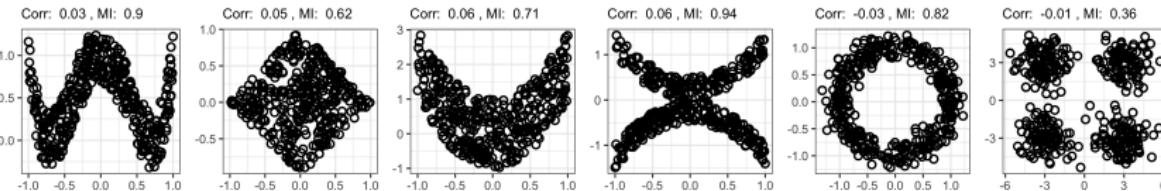
MUTUAL INFORMATION PROPERTIES

- MI is a measure of the amount of "dependence" between variables. It is zero if and only if the variables are independent.
- OTOH, if one RV is a deterministic function of the other, MI is maximal, i.e. entropy of the first RV.
- Unlike (Pearson) correlation, MI is not limited to real-valued RVs.
- Can use MI as a **feature filter**, sometimes called information gain.
- Can also be used in CART to select feature for split.
Splitting on MI/IG = risk reduction with log-loss.
- MI invariant under injective and continuously differentiable reparametrizations.



MUTUAL INFORMATION VS. CORRELATION

- If two RVs are independent, their correlation is 0.
- But: two dependent RVs can have correlation 0 because correlation only measures linear dependence.



- Above: Many examples with strong dependence, nearly 0 correlation and much larger MI.
- MI can be seen as more general measure of dependence than correlation.

MUTUAL INFORMATION - EXAMPLE

Let X, Y be two correlated Gaussian random variables.

$(X, Y) \sim \mathcal{N}(0, K)$ with correlation ρ and covariance matrix K :

$$K = \begin{pmatrix} \sigma^2 & \rho\sigma^2 \\ \rho\sigma^2 & \sigma^2 \end{pmatrix}$$

Then $h(X) = h(Y) = \frac{1}{2} \log((2\pi e)\sigma^2)$, and

$h(X, Y) = \frac{1}{2} \log((2\pi e)^2 |K|) = \frac{1}{2} \log((2\pi e)^2 \sigma^4 (1 - \rho^2))$, and thus

$$I(X; Y) = h(X) + h(Y) - h(X, Y) = -\frac{1}{2} \log(1 - \rho^2).$$

For $\rho = 0$, X and Y are independent and $I(X; Y) = 0$.

For $\rho = \pm 1$, X and Y are perfectly correlated and $I(X; Y) \rightarrow \infty$.



Introduction to Machine Learning

Entropy and Optimal Code Length



Learning goals

0 0 0 1 0 0 1 1	encoded string
00 01 00 11	codewords
dog cat dog bird	source symbols

- Know that source coding is about encoding messages efficiently
- Know how to compute the average length of a code
- Know that the entropy of the source distribution is the lower bound for the average code length

SOURCE CODING

- There is an interesting connection between entropy and a subfield of information theory known as **source coding**.
- Abstractly, a source is any system or process that generates messages or information.
- A code is simply a way to represent the message so that it can be stored or transmitted over a communication channel (such as radio or fiber-optic cables).
- For example, one could use binary strings (0's and 1's) to encode messages.
- Because it may be expensive to transmit or store information, an important problem addressed by source coding is efficient coding schemes of minimal average length.



SOURCE CODING / 2

- Formally, given a discrete alphabet/dictionary X of message symbols, a **binary code** is a mapping from symbols in X to a set of codewords of binary strings.
- For example, if our dictionary only consists of the words "dog", "cat", "fish" and "bird", each word can be encoded as a binary string of length 2 : "dog" → **00**, "cat" → **01**, "fish" → **10** and "bird" → **11**.
- For this code, a binary string can be decoded by replacing each successive pair of digits with the associated word.

0 0 0 1 0 0 1 1 encoded string

00 01 00 11 codewords

dog cat dog bird source symbols

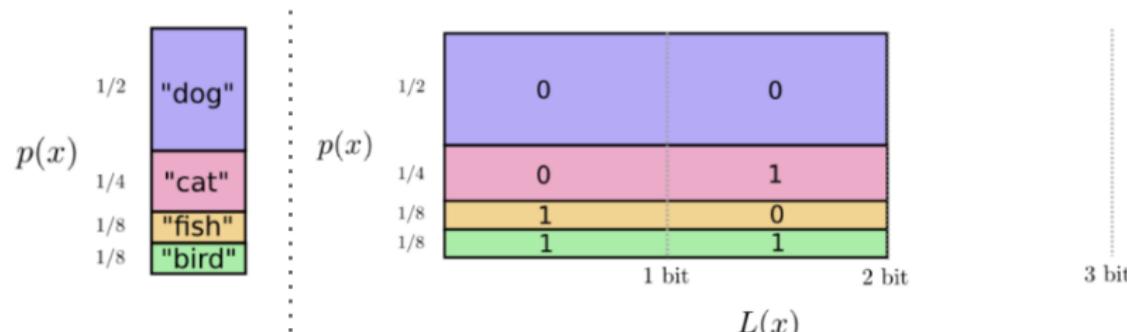
Credit: Chris Olah

Chris Olah (2015): Visual Information Theory. <http://colah.github.io/posts/2015-09-Visual-Information/>



SOURCE CODING / 3

- Encoded messages are emitted by a source which can be modeled as a probability distribution over the message symbols in the dictionary.
- Let X be a random variable that represents a symbol from our data source and let $p(x) = \mathbb{P}(X = x)$, for symbol x in our dictionary.

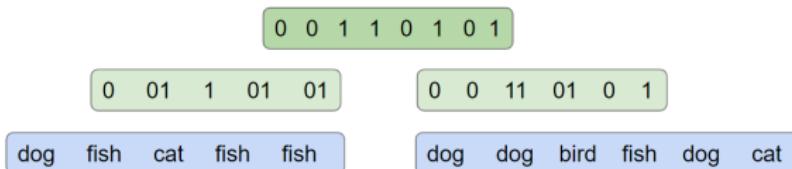


Credit: Chris Olah

- Length $L(x)$ is simply number of bits in corresponding codeword.
Here all codewords have length 2.
- Area of rectangles on the right reflect contributions to $\mathbb{E}[L(X)]$

SOURCE CODING / 4

- Maybe we can create better average-length coding schemes with **variable-length** codes by assigning shorter codes to more likely messages and longer one to less likely messages.
- However, this can be problematic because we want the receiver to be able to unambiguously decode the encoded string.
- Let us say the words in our dictionary are encoded in this way: "dog" → **0**, "cat" → **1**, "fish" → **01** and "bird" → **11**.
- In this case, the string 00110101 can be decoded in multiple ways.



- One way to make variable-length messages unambiguous is by ensuring that no codeword is a prefix (initial segment) of any other codeword. Such a code is known as a **prefix code**.



SOURCE CODING / 5

- In general, the number of possible codewords grows exponentially in length L .
- For binary codes, there are two possible words of length one, four possible words of length two and 2^L possible words of length L .

		0	0	
		1	1	
0		0	0	
		1	1	
1		0	0	
		1	1	
		0	0	
		1	1	

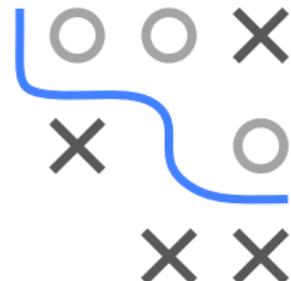


- In total, there are $(2^{L+1} - 2)$ codewords of length $\leq L$.

SOURCE CODING / 6

0	0	0	$\frac{1}{2^L} = \frac{1}{4}$
	1	1	
1	0	0	$\frac{1}{2^L} = \frac{1}{4}$
	1	1	

bit 1 bit 2 bit 3

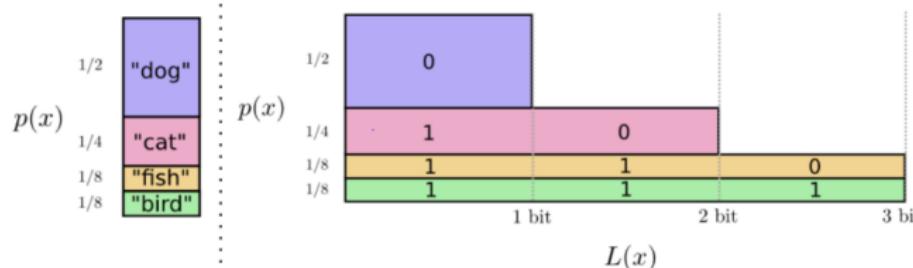


- Here, if the codeword **01** is assigned to a symbol, then **010** and **011** cannot be assigned to any other symbol because that would break the prefix property.
- If a codeword of length L is assigned to a symbol, then $\frac{1}{2^L}$ of the possible codewords of length $> L$ must be discarded.
- If some symbols are assigned short codewords, due to the prefix property, many marginally longer codewords cannot be assigned to other symbols.

SOURCE CODING / 7

- An example of prefix code:

"dog" → **0**, "cat" → **10**, "fish" → **110** and "bird" → **111**.



- Here, the expected code length is :

$$\begin{aligned}\mathbb{E}[L(X)] &= \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 \\ &= -\frac{1}{2} \cdot \log_2 \left(\frac{1}{2} \right) - \frac{1}{4} \cdot \log_2 \left(\frac{1}{4} \right) - \frac{1}{8} \cdot \log_2 \left(\frac{1}{8} \right) - \frac{1}{8} \cdot \log_2 \left(\frac{1}{8} \right) \\ &= H(X) = \textcolor{red}{1.75} \text{ bits. } (< 2 \text{ bits})\end{aligned}$$

SOURCE CODING / 8

- Actually, this coding scheme is the most efficient way to store and transmit these messages. It is simply not possible to do better!
- In fact, Shannon's **source coding theorem** (or **noiseless coding theorem**) tells us that the optimal trade-off is made when the code length of a symbol with probability p is $\log(1/p)$.
- In other words, the entropy of the source distribution is the theoretical lower bound on the average code length.
- If it is any lower, some information will be distorted or lost.
- In practice, algorithms such as Huffman Coding can be used to find variable-length codes that are close (in terms of expected length) to the theoretical limit.



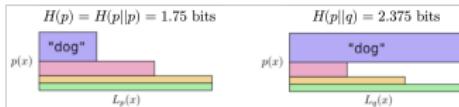
Introduction to Machine Learning

Source Coding and Cross-Entropy



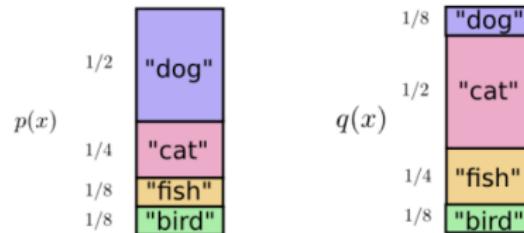
Learning goals

- Know connection between source coding and (cross-)entropy
- Know that the entropy of the source distribution is the lower bound for the average code length

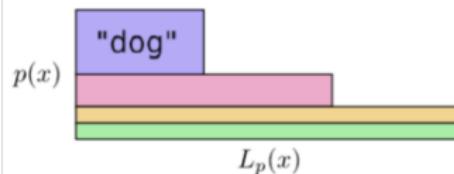


SOURCE CODING AND CROSS-ENTROPY

- For a random source / distribution p , the minimal number of bits to optimally encode messages from is the entropy $H(p)$.
- If the optimal code for a different distribution $q(x)$ is instead used to encode messages from $p(x)$, expected code length will grow.



$$H(p) = H(p||p) = 1.75 \text{ bits}$$



$$H(p||q) = 2.375 \text{ bits}$$

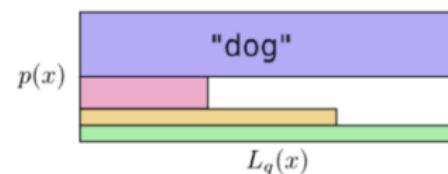


Figure: $L_p(x)$, $L_q(x)$ are the optimal code lengths for $p(x)$ and $q(x)$

SOURCE CODING AND CROSS-ENTROPY / 2

Cross-entropy is the average length of communicating an event from one distribution with the optimal code for another distribution (assumed they have the same domain \mathcal{X} as in KL).

$$H(p\|q) = \sum_{x \in \mathcal{X}} p(x) \log \left(\frac{1}{q(x)} \right) = - \sum_{x \in \mathcal{X}} p(x) \log (q(x))$$

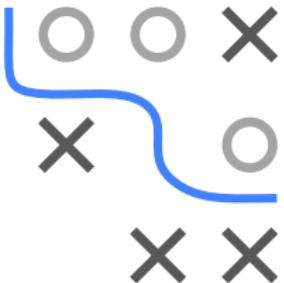
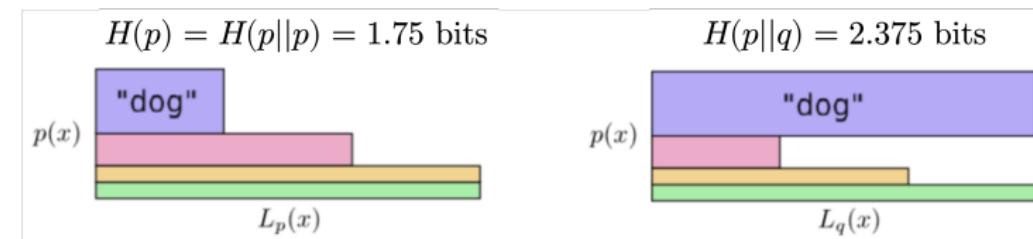
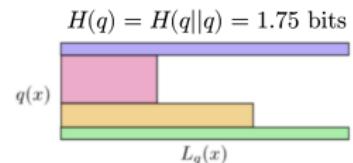
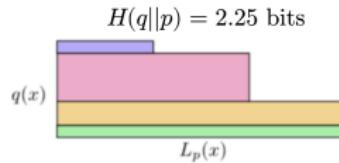
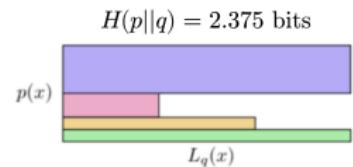
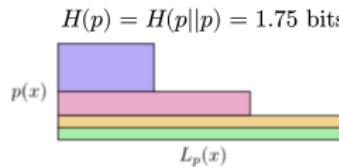


Figure: $L_p(x)$, $L_q(x)$ are the optimal code lengths for $p(x)$ and $q(x)$

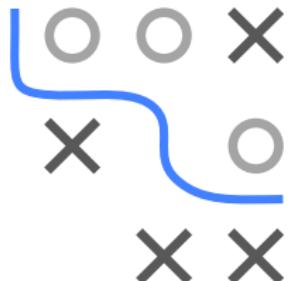
We directly see: cross-entropy of p with itself is entropy:

$$H(p||p) = H(p).$$

SOURCE CODING AND CROSS-ENTROPY / 3



Credit: Chris Olah



- In top, $H(p||q)$ is greater than $H(p)$ primarily because the blue event that is very likely under p has a very long codeword in q .
- Same, in bottom, for pink when we go from q to p .
- Note that $H(p||q) \neq H(q||p)$.

SOURCE CODING AND CROSS-ENTROPY / 4

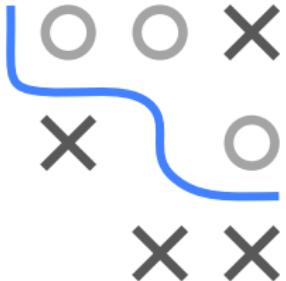
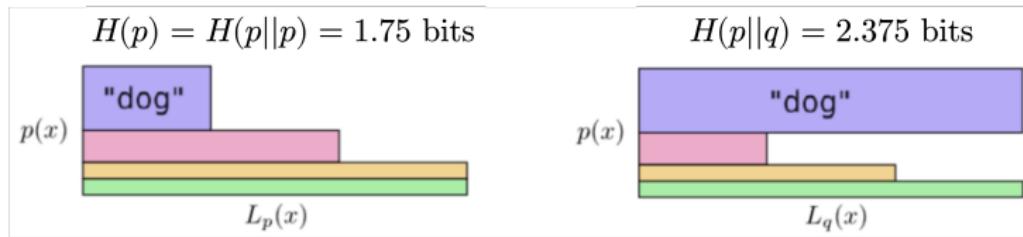


Figure: $L_p(x)$, $L_q(x)$ are the optimal code lengths for $p(x)$ and $q(x)$

- Let x' denote the symbol "dog". The difference in code lengths is:

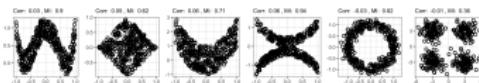
$$\log\left(\frac{1}{q(x')}\right) - \log\left(\frac{1}{p(x')}\right) = \log \frac{p(x')}{q(x')}$$

- If $p(x') > q(x')$, this is positive, if $p(x') < q(x')$, it is negative.
- The expected difference is KL, if we encode symbols from p :

$$D_{KL}(p\|q) = \sum_{x \in \mathcal{X}} p(x) \cdot \log \frac{p(x)}{q(x)}$$

Introduction to Machine Learning

Mutual Information under Reparametrization (Deep-Dive)



Learning goals

- Understand why MI is invariant under certain reparametrizations



MUTUAL INFORMATION PROPERTIES

- MI is invariant w.r.t. injective reparametrizations that are in \mathcal{C}^1 :

Let $f, g : \mathbb{R}^d \rightarrow \mathbb{R}^d \in \mathcal{C}^1$ be injective transformations and X, Y be continuous random variables in \mathbb{R}^d then by the change of variables the joint and marginal densities of $\tilde{X} = f(X), \tilde{Y} = g(Y)$

$$\tilde{p}(\tilde{x}, \tilde{y}) = p(f^{-1}(\tilde{x}), g^{-1}(\tilde{y})) \cdot |J_{f^{-1}}(\tilde{x})| \cdot |J_{g^{-1}}(\tilde{y})|,$$

$$\tilde{p}(\tilde{x}) = p(f^{-1}(\tilde{x})) \cdot |J_{f^{-1}}(\tilde{x})|, \quad \tilde{p}(\tilde{y}) = p(g^{-1}(\tilde{y})) \cdot |J_{g^{-1}}(\tilde{y})|,$$

where $p(x, y)$ is the joint density of X and Y and $p(x), p(y)$ are the respective marginal densities. (J denotes the Jacobian)

With this, it follows that

$$I(\tilde{X}; \tilde{Y}) = \int \tilde{p}(\tilde{x}, \tilde{y}) \log \left(\frac{\tilde{p}(\tilde{x}, \tilde{y})}{\tilde{p}(\tilde{x})\tilde{p}(\tilde{y})} \right) d\tilde{x}d\tilde{y} = *$$



MUTUAL INFORMATION PROPERTIES

$$\begin{aligned} * &= \int p(f^{-1}(\tilde{x}), g^{-1}(\tilde{y})) \cdot |J_{f^{-1}}(\tilde{x})| \cdot |J_{g^{-1}}(\tilde{y})| \\ &\quad \cdot \log \left(\frac{p(f^{-1}(\tilde{x}), g^{-1}(\tilde{y})) \cdot |J_{f^{-1}}(\tilde{x})| \cdot |J_{g^{-1}}(\tilde{y})|}{p(f^{-1}(\tilde{x}))|J_{f^{-1}}(\tilde{x})| \cdot p(g^{-1}(\tilde{y}))|J_{g^{-1}}(\tilde{y})|} \right) d\tilde{x}d\tilde{y} \\ &= \int p(f^{-1}(f(x)), g^{-1}(g(y))) \cdot |J_{f^{-1}}(f(x))| \cdot |J_{g^{-1}}(g(y))| \\ &\quad \cdot \log \left(\frac{p(f^{-1}(f(x)), g^{-1}(g(y)))}{p(f^{-1}(f(x)))p(g^{-1}(g(y)))} \right) |J_f(x)| \cdot |J_g(y)| dx dy \\ &= \int p(x, y) \cdot |J_{f^{-1}}(f(x))J_f(x)| \cdot |J_{g^{-1}}(g(y))J_g(y)| \log \left(\frac{p(x, y)}{p(x)p(y)} \right) dx dy \\ &= \int p(x, y) \cdot \log \left(\frac{p(x, y)}{p(x)p(y)} \right) dx dy = I(X; Y). \end{aligned}$$

(The fourth equality holds by the inverse function theorem)



INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

Boosting

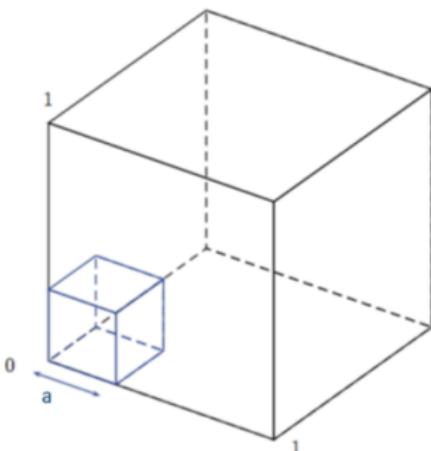
Gaussian Processes



Introduction to Machine Learning

Curse of Dimensionality

Curse of Dimensionality



Learning goals

- Understand that our intuition about geometry fails in high-dimensional spaces
- Understand the effects of the curse of dimensionality

CURSE OF DIMENSIONALITY

- The phenomenon of data becoming sparse in high-dimensional spaces is one effect of the **curse of dimensionality**.
- The **curse of dimensionality** refers to various phenomena that arise when analyzing data in high-dimensional spaces that do not occur in low-dimensional spaces.
- Our intuition about the geometry of a space is formed in two and three dimensions.
- We will see: This intuition is often misleading in high-dimensional spaces.



CURSE OF DIMENSIONALITY: EXAMPLE

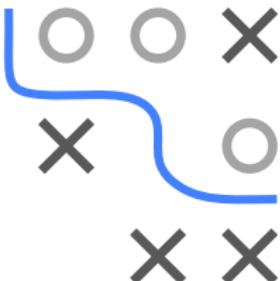
To illustrate one of the problematic phenomena of data in high dimensional data, we look at an introductory example:

We are given 20 emails, 10 of them are spam and 10 are not.
Our goal is to predict if a new incoming mail is spam or not.

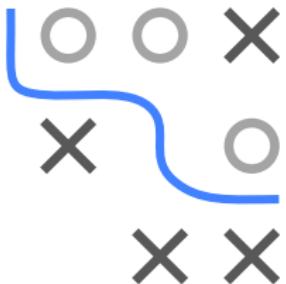
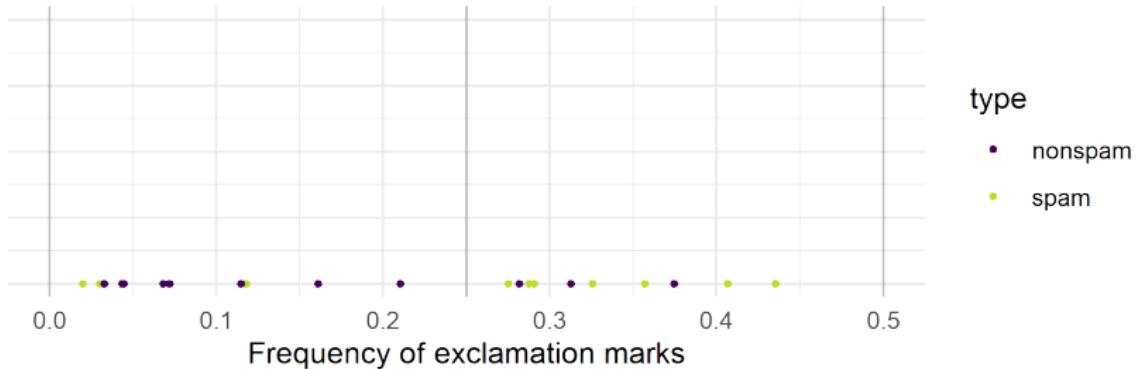
For each email, we extract the following features:

- frequency of exclamation marks (in %)
- the length of the longest sequence of capital letters
- the frequency of certain words, e.g., “free” (in %)
- ...

... and we could extract many more features!



CURSE OF DIMENSIONALITY: EXAMPLE

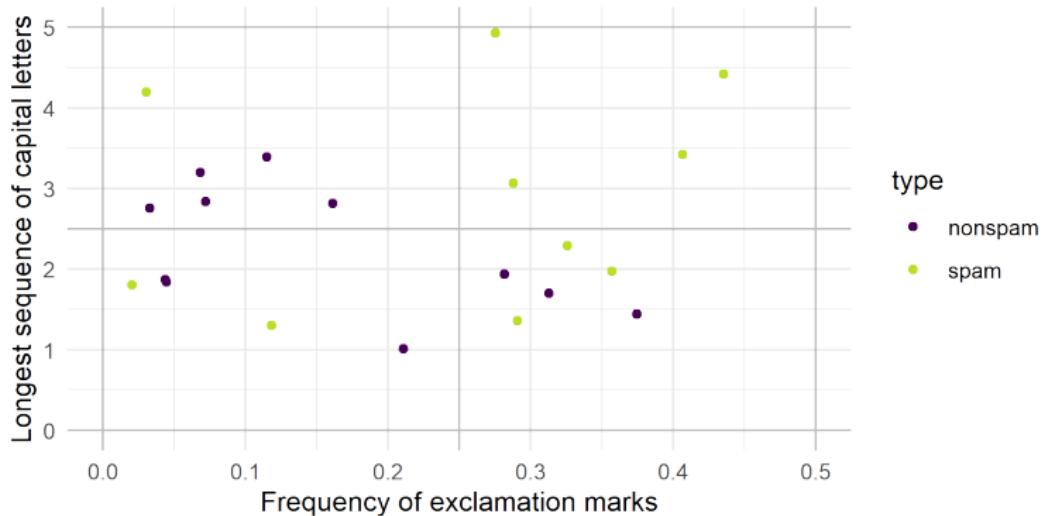


Based on the frequency of exclamation marks, we train a very simple classifier (a decision stump with split point $x = 0.25$):

- We divide the input space into 2 equally sized regions.
- In the second region $[0.25, 0.5]$, 7 out of 10 are spam.
- Given that at least 0.25% of all letters are exclamation marks, an email is spam with a probability of $\frac{7}{10} = 0.7$.

CURSE OF DIMENSIONALITY: EXAMPLE

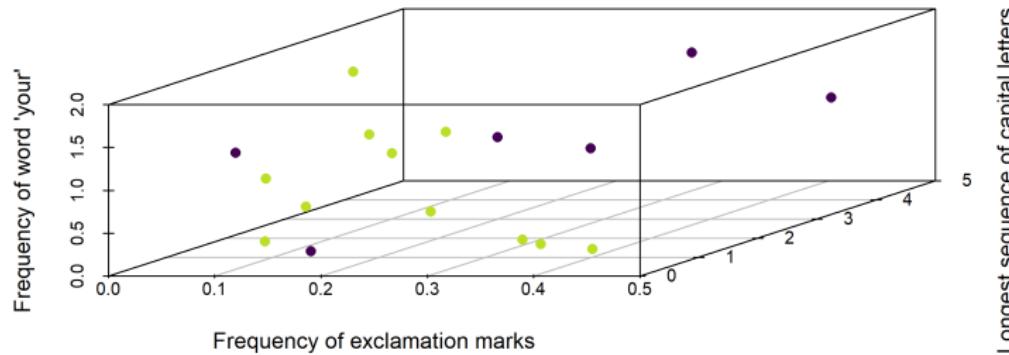
Let us feed more information into our classifier. We include a feature that contains the length of the longest sequence of capital letters.



- In the 1D case we had 20 observations across 2 regions.
- The same number is now spread across 4 regions.

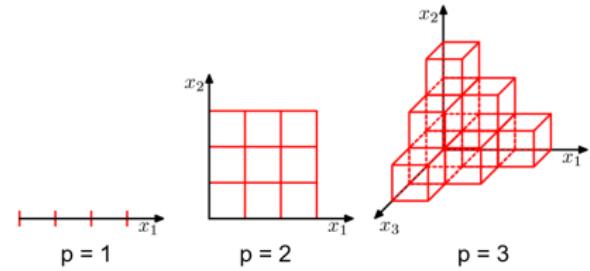
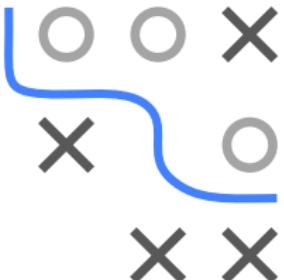
CURSE OF DIMENSIONALITY: EXAMPLE

Let us further increase the dimensionality to 3 by using the frequency of the word “your” in an email.



CURSE OF DIMENSIONALITY: EXAMPLE

- When adding a third dimension, the same number of observations is spread across 8 regions.
- In 4 dimensions the data points are spread across 16 cells, in 5 dimensions across 32 cells and so on ...
- As dimensionality increases, the data become **sparse**; some of the cells become empty.
- There might be too few data in each of the blocks to understand the distribution of the data and to model it.



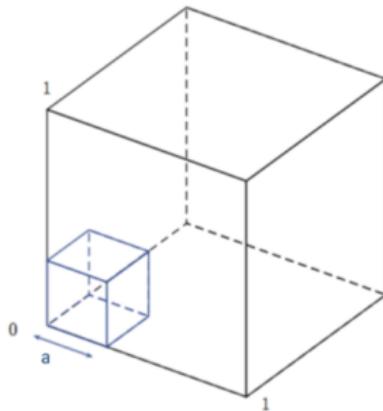
Bishop, Pattern Recognition and Machine Learning, 2006

Geometry of High-Dimensional Spaces

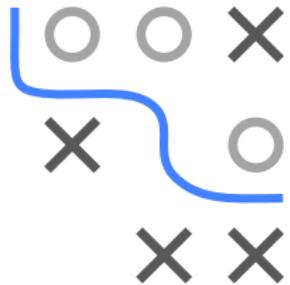
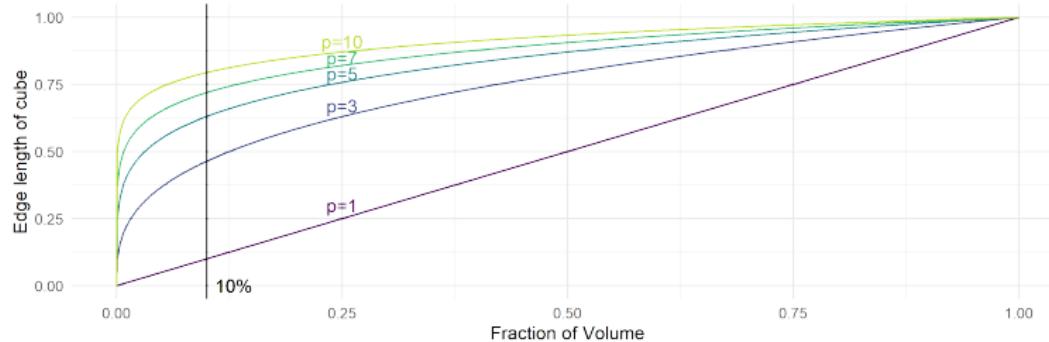


THE HIGH-DIMENSIONAL CUBE

- We embed a small cube with edge length a inside a unit cube.
- How long does the edge length a of this small hypercube have to be so that the hypercube covers 10%, 20%, ... of the volume of the unit cube (volume 1)?



THE HIGH-DIMENSIONAL CUBE



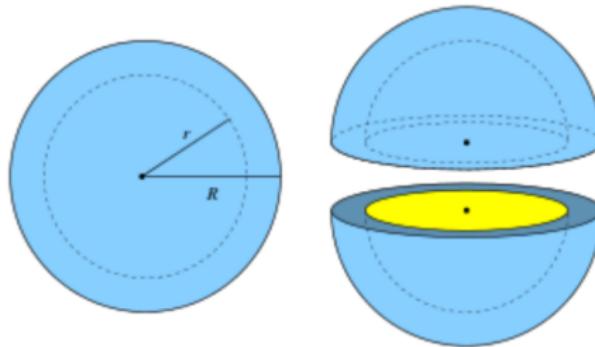
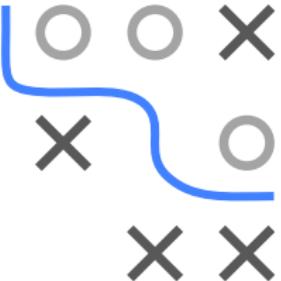
$$a^p = \frac{1}{10} \Leftrightarrow a = \frac{1}{\sqrt[p]{10}}$$

- So: covering 10% of total volume in a cell requires cells with almost 50% of the entire range in 3 dimensions, 80% in 10 dimensions.

THE HIGH-DIMENSIONAL SPHERE

Another manifestation of the **curse of dimensionality** is that the majority of data points are close to the outer edges of the sample. Consider a hypersphere of radius 1. The fraction of volume that lies in the ϵ -“edge”, $\epsilon := R - r$, of this hypersphere can be calculated by the formula

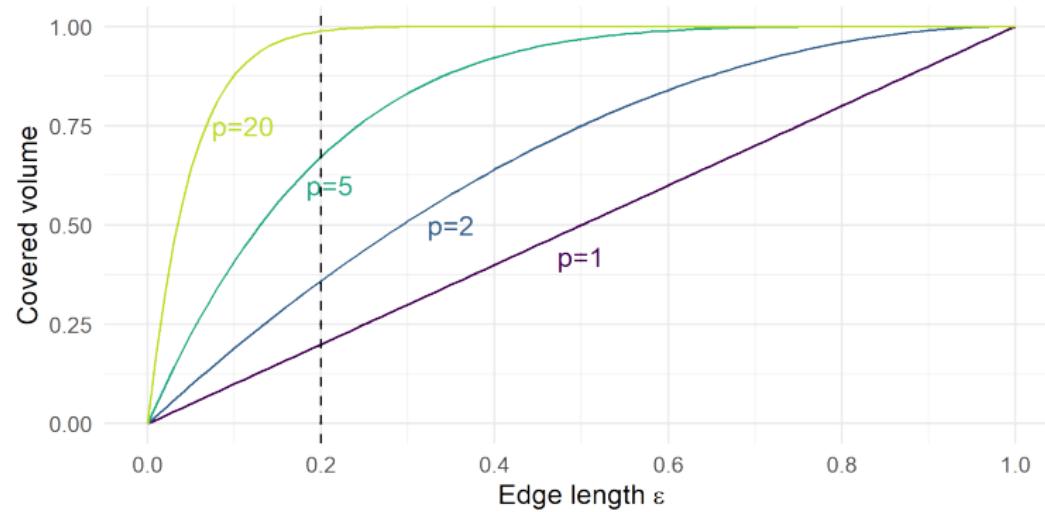
$$1 - \left(1 - \frac{\epsilon}{R}\right)^p.$$



If we peel a high-dimensional orange, there is almost nothing left.

THE HIGH-DIMENSIONAL SPHERE

Consider a 20-dimensional sphere. Nearly all of the volume lies in its outer shell of thickness 0.2:

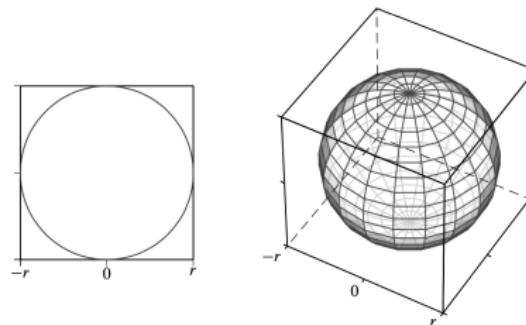
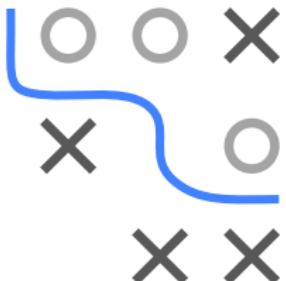


HYPERSPHERE WITHIN HYPERCUBE

Consider a p-dimensional hypersphere of radius r and volume $S_p(r)$ inscribed in a p-dimensional hypercube with sides of length $2r$ and volume $C_p(r)$. Then it holds that

$$\lim_{p \rightarrow \infty} \frac{S_p(r)}{C_p(r)} = \lim_{p \rightarrow \infty} \frac{\left(\frac{\pi^{\frac{p}{2}}}{\Gamma(\frac{p}{2}+1)} \right) r^p}{(2r)^p} = \lim_{p \rightarrow \infty} \frac{\pi^{\frac{p}{2}}}{2^p \Gamma(\frac{p}{2} + 1)} = 0,$$

i.e., as the dimensionality increases, most of the volume of the hypercube can be found in its corners.

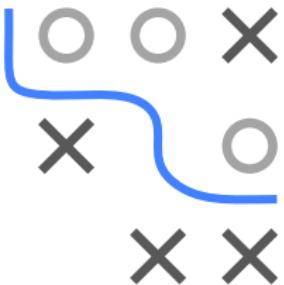
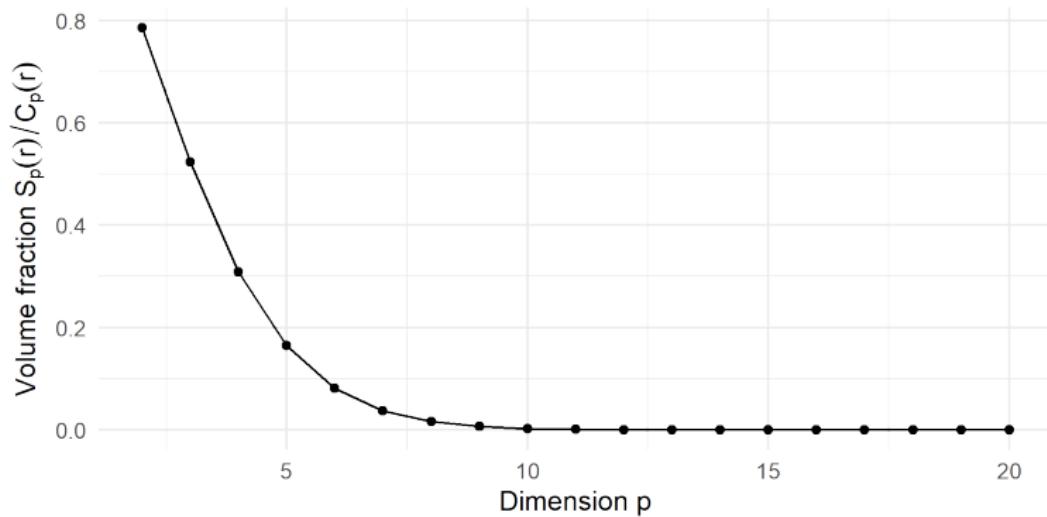


Mohammed J. Zaki, Wagner Meira, Jr., Data Mining and Analysis: Fundamental Concepts and Algorithms, 2014

HYPERSPHERE WITHIN HYPERCUBE

Consider a 10-dimensional sphere inscribed in a 10-dimensional cube.

Nearly all of the volume lies in the corners of the cube:



Note: For $r > 0$, the volume fraction $\frac{S_p(r)}{C_p(r)}$ is independent of r .

UNIFORMLY DISTRIBUTED DATA

The consequences of the previous results for uniformly distributed data in the high-dimensional hypercube are:

- Most of the data points will lie on the boundary of the space.
- The points will be mainly scattered on the large number of corners of the hypercube, which themselves will become very long spikes.
- Hence the higher the dimensionality, the more similar the minimum and maximum distances between points will become.
- This degrades the effectiveness of most distance functions.
- Neighborhoods of points will not be local anymore.



GAUSSIANS IN HIGH DIMENSIONS

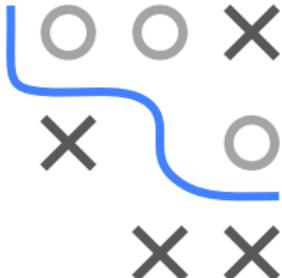
A further manifestation of the **curse of dimensionality** appears if we consider a standard Gaussian $N_p(\mathbf{0}, \mathbf{I}_p)$ in p dimensions.

- After transforming from Cartesian to polar coordinates and integrating out the directional variables, we obtain an expression for the density $p(r)$ as a function of the radius r (i.e., the point's distance from the origin), s.t.

$$p(r) = \frac{S_p r^{p-1}}{(2\pi\sigma^2)^{p/2}} \exp\left(-\frac{r^2}{2\sigma^2}\right),$$

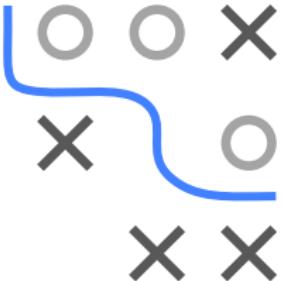
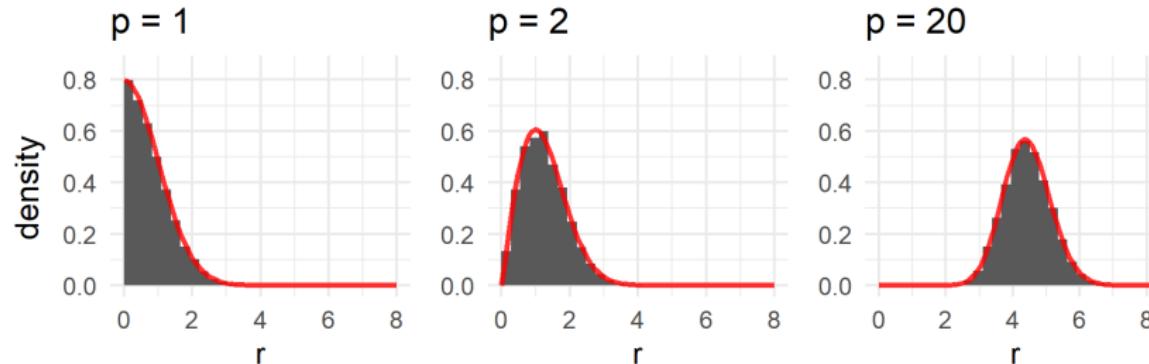
where S_p is the surface area of the p -dimensional unit hypersphere.

- Thus $p(r)\delta r$ is the approximate probability mass inside a thin shell of thickness δr located at radius r .



GAUSSIANS IN HIGH DIMENSIONS

- To verify this functional relationship empirically, we draw 10^4 points from the p -dimensional standard normal distribution and plot $p(r)$ over the histogram of the points' distances to the origin:



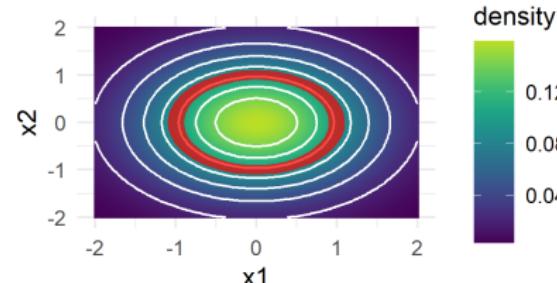
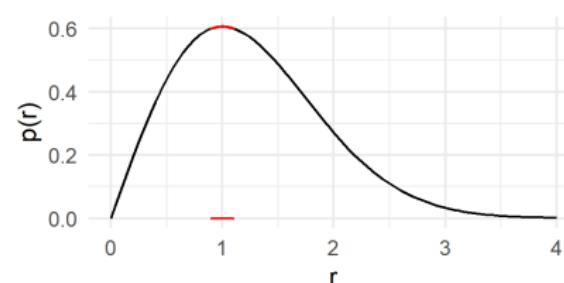
- We can see that for large p the probability mass of the Gaussian is concentrated in a fairly thin “shell” rather far away from the origin. This may seem counterintuitive, but:

GAUSSIANS IN HIGH DIMENSIONS

- For the probability mass of a hyperspherical shell it follows that

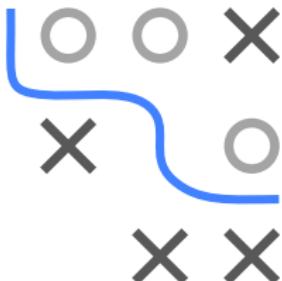
$$\int_{r-\frac{\delta r}{2}}^{r+\frac{\delta r}{2}} p(\tilde{r}) d\tilde{r} = \int_{r-\frac{\delta r}{2} \leq \|\mathbf{x}\|_2 \leq r+\frac{\delta r}{2}} f_p(\tilde{\mathbf{x}}) d\tilde{\mathbf{x}},$$

where $f_p(\mathbf{x})$ is the density of the p -dimensional standard normal distribution and $p(r)$ the associated radial density.



Example: 2D normal distribution

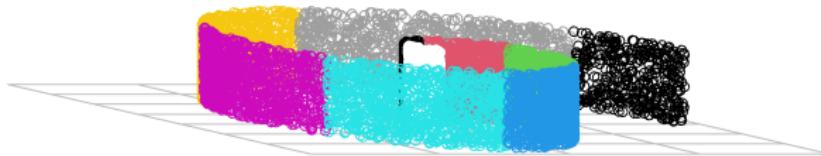
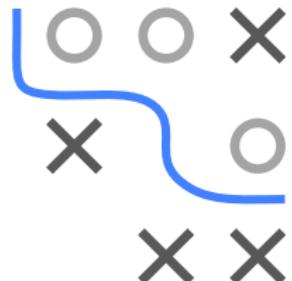
- While f_p becomes smaller with increasing r , the region of the integral -the hyperspherical shell- becomes bigger.



INTERMEDIATE REMARKS

However, we can find effective techniques applicable to high-dimensional spaces if we exploit these properties of real data:

- Often the data is restricted to a manifold of a lower dimension.
(Or at least the directions in the feature space over which significant changes in the target variables occur may be confined.)
- At least locally small changes in the input variables usually will result in small changes in the target variables.

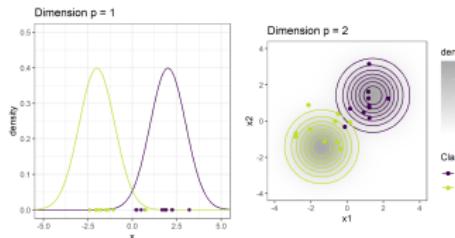


Introduction to Machine Learning

Curse of Dimensionality

Curse of Dimensionality - Examples

Learning Algorithms



Learning goals

- See how the performance of k-NN and the linear model deteriorates in high-dimensional spaces

EXAMPLE: K-NN

Let us look at the performance of algorithms for increasing dimensionality. First, we consider the k-NN algorithm:

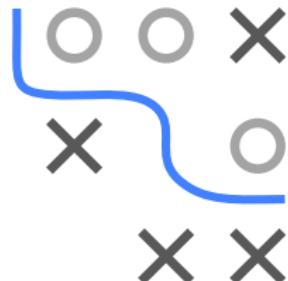
- In a high dimensional space, data points are spread across a huge space.
- The distance to the **next neighbor** $d_{NN1}(\mathbf{x})$ becomes extremely large.
- The distance might even get so large that all points are **equally far** away - we cannot really determine the nearest neighbor anymore.



EXAMPLE: K-NN

Minimal, mean and maximal (NN)-distances of 10^4 points uniformly distributed in the hypercube $[0, 1]^p$:

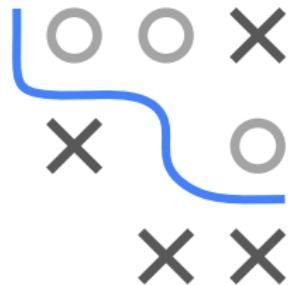
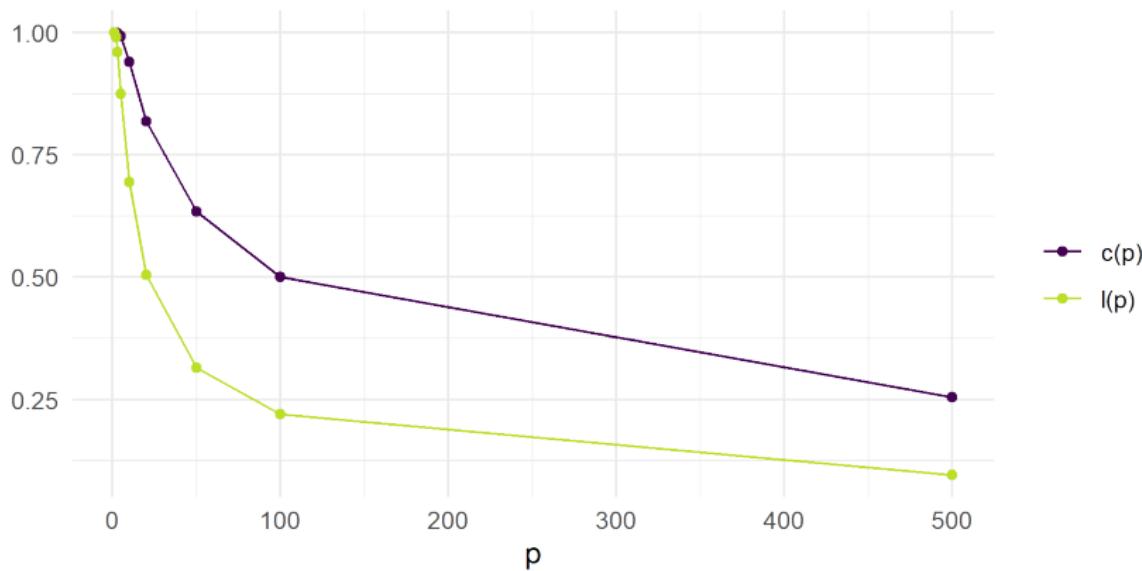
p	$\min d(\mathbf{x}, \tilde{\mathbf{x}})$	$\overline{d(\mathbf{x}, \tilde{\mathbf{x}})}$	$\max d(\mathbf{x}, \tilde{\mathbf{x}})$	$\overline{d_{NN1}(\mathbf{x})}$	$\max d_{NN1}(\mathbf{x})$
1	1.2e-08	0.33	1	5e-05	0.00042
2	0.00011	0.52	1.4	0.0051	0.02
3	0.0021	0.66	1.7	0.026	0.073
5	0.016	0.88	2	0.11	0.23
10	0.15	1.3	2.5	0.39	0.63
20	0.55	1.8	3	0.9	1.2
50	1.5	2.9	4.1	2	2.4
100	2.7	4.1	5.4	3.2	3.5
500	7.8	9.1	10	8.2	8.6



EXAMPLE: K-NN

We see a decrease of relative contrast¹ $c := \frac{\max(d(\mathbf{x}, \tilde{\mathbf{x}})) - \min(d(\mathbf{x}, \tilde{\mathbf{x}}))}{\max(d(\mathbf{x}, \tilde{\mathbf{x}}))}$ and

“locality”² $l := \frac{d(\mathbf{x}, \tilde{\mathbf{x}}) - d_{NN1}(\mathbf{x})}{d(\mathbf{x}, \tilde{\mathbf{x}})}$ with increasing number of dimensions p :



¹[Aggarwal et al., 2001]

²our non-standard definition

EXAMPLE: K-NN

The consequences for the k-nearest neighbors approach can be summarized as follows:

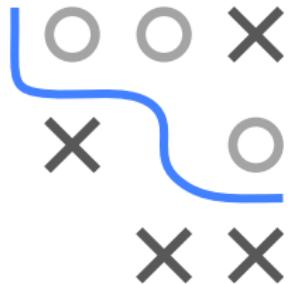
- At constant sample size n and growing p , the distance between the observations increases
 - the coverage of the p -dimensional space decreases,
 - every point becomes isolated / far way from all other points.
- The size of the neighborhood $N_k(x)$ also “increases”
(at constant k)
 - it is no longer a “local” method.
- Reducing k dramatically does not help much either, since the fewer observations we average, the higher the variance of our fit.
 - k-NN estimates get more inaccurate with increasing dimensionality of the data.



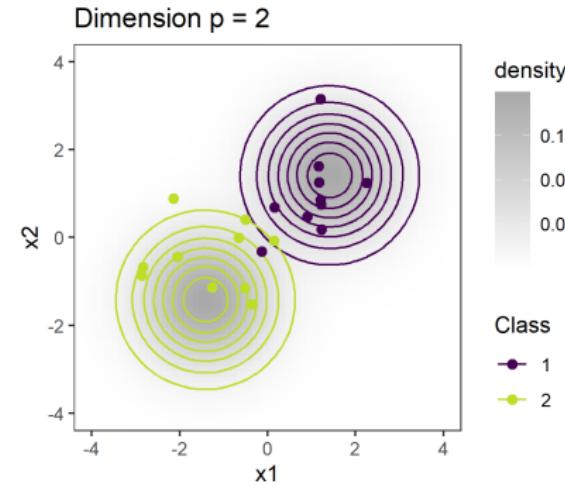
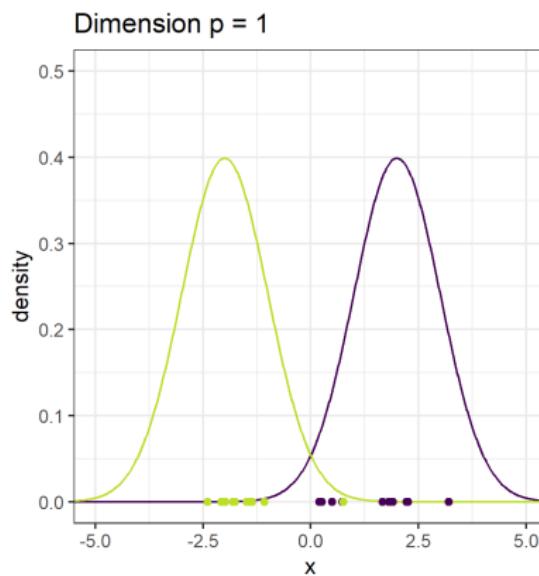
EXAMPLE: K-NN

To demonstrate this, we generate an artificial data set of dimension p as follows: We define $a = \frac{2}{\sqrt{p}}$ and

- with probability $\frac{1}{2}$ we generate a sample from class 1 by sampling from a Gaussian with mean $\mu = (a, a, \dots, a)$ and unit covariance matrix
- with probability $\frac{1}{2}$ we generate a sample from class 2 by sampling from a Gaussian with mean $-\mu = (-a, -a, \dots, -a)$ and unit covariance matrix



EXAMPLE: K-NN



EXAMPLE: K-NN

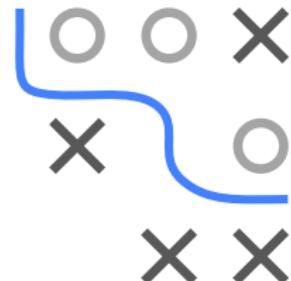
This example is constructed such that the Bayes error is always constant and does not depend on the dimension p .

The Bayes optimal classifiers predicts $\hat{y} = 1$ iff

$$\begin{aligned}\mathbb{P}(y = 1 \mid \mathbf{x}) &= \frac{p(\mathbf{x} \mid y = 1)\mathbb{P}(y = 1)}{p(\mathbf{x})} = \frac{1}{2} \cdot \frac{p(\mathbf{x} \mid y = 1)}{p(\mathbf{x})} \\ &\geq \frac{1}{2} \cdot \frac{p(\mathbf{x} \mid y = 2)}{p(\mathbf{x})} \\ &= \frac{p(\mathbf{x} \mid y = 2)\mathbb{P}(y = 2)}{p(\mathbf{x})} = \mathbb{P}(y = 2 \mid \mathbf{x}).\end{aligned}$$

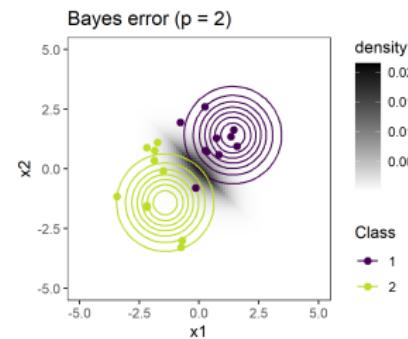
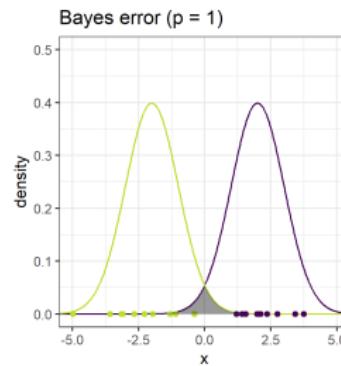
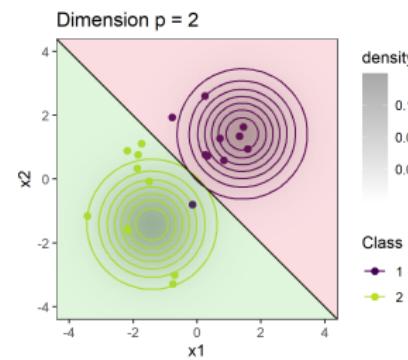
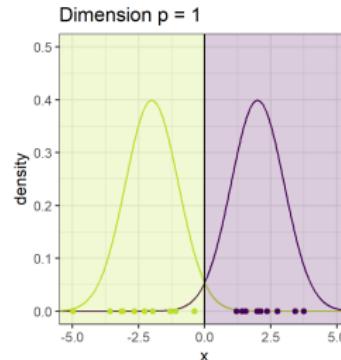
This is equivalent to

$$\begin{aligned}\hat{y} = 1 &\Leftrightarrow \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top(\mathbf{x} - \boldsymbol{\mu})\right) \geq \exp\left(-\frac{1}{2}(\mathbf{x} + \boldsymbol{\mu})^\top(\mathbf{x} + \boldsymbol{\mu})\right) \\ &\Leftrightarrow \mathbf{x}^\top \boldsymbol{\mu} \geq 0.\end{aligned}$$



EXAMPLE: K-NN

Optimal Bayes classifier and Bayes error (shaded area):



EXAMPLE: K-NN

We can calculate the corresponding expected misclassification error
(Bayes error)

$$\begin{aligned} & p(\hat{y} = 1 \mid y = 2) \mathbb{P}(y = 2) + p(\hat{y} = 2 \mid y = 1) \mathbb{P}(y = 1) \\ = & \frac{1}{2} \cdot p(\mathbf{x}^\top \boldsymbol{\mu} \geq 0 \mid y = 2) + \frac{1}{2} \cdot p(\mathbf{x}^\top \boldsymbol{\mu} \leq 0 \mid y = 1) \\ \stackrel{\text{symm.}}{=} & p(\mathbf{x}^\top \boldsymbol{\mu} \leq 0 \mid y = 1) = p\left(\sum_{i=1}^p a \mathbf{x}_i \leq 0 \mid y = 1\right) \\ = & p\left(\sum_{i=1}^p \mathbf{x}_i \leq 0 \mid y = 1\right). \end{aligned}$$

$\sum_{i=1}^p \mathbf{x}_i \mid y = 1 \sim \mathcal{N}(p \cdot a, p)$, because it is the sum of independent normal random variables $\mathbf{x}_i \mid y = 1 \sim \mathcal{N}(a, 1)$ (the vector $\mathbf{x} \mid y = 1$ follows a $\mathcal{N}(\boldsymbol{\mu}, \mathbf{I})$ distribution with $\boldsymbol{\mu} = (a, \dots, a)$).



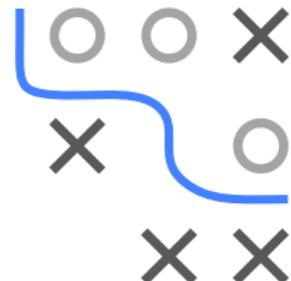
EXAMPLE: K-NN

We get for the Bayes error:

$$\begin{aligned} &= p \left(\frac{\sum_{i=1}^p \mathbf{x}_i - p \cdot a}{\sqrt{p}} \leq \frac{-p \cdot a}{\sqrt{p}} \mid y = 1 \right) \\ &= \Phi(-\sqrt{p}a) \stackrel{a=\frac{2}{\sqrt{p}}}{=} \Phi(-2) \approx 0.0228, \end{aligned}$$

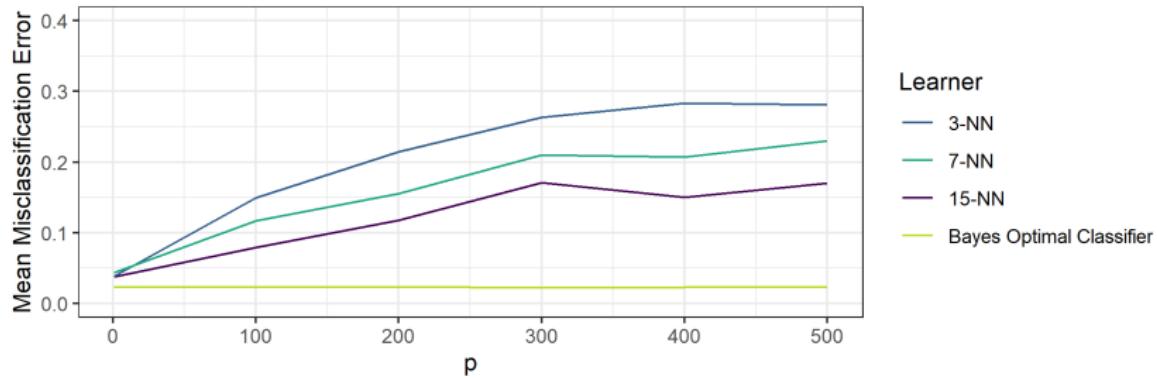
where Φ is the distribution function of a standard normal random variable.

We see that the Bayes error is independent of p .



EXAMPLE: K-NN

We also train a k-NN classifier for $k = 3, 7, 15$ for increasing dimensions and monitor its performance (evaluated by 10 times repeated 10-fold CV).

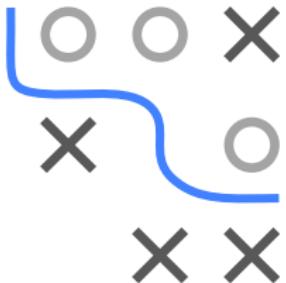


→ k-NN deteriorates quickly with increasing dimension

EXAMPLE: LINEAR MODEL

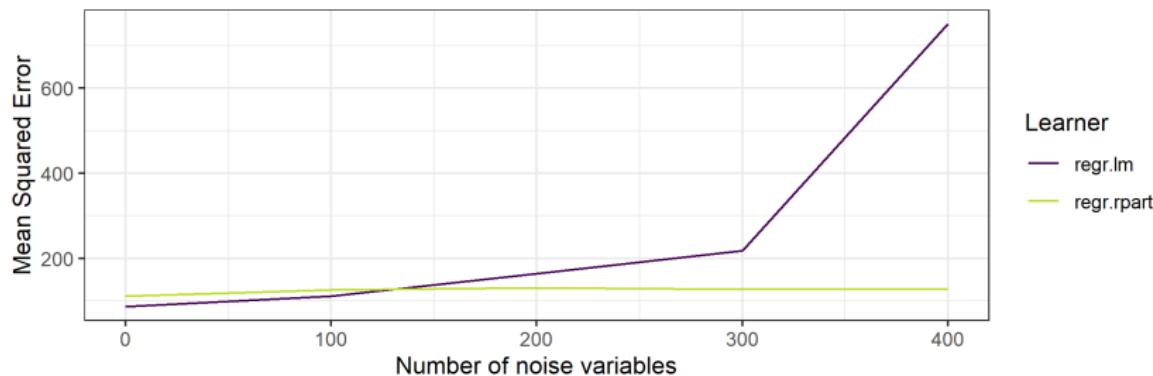
We also investigate how the linear model behaves in high dimensional spaces.

- We take the Boston Housing data set, where the value of houses in the area around Boston is predicted based on 13 features describing the region (e.g., crime rate, status of the population, etc.).
- We train a linear model on the data consisting of 506 observations.
- We artificially create a high-dimensional dataset by adding 100, 200, 300, ... noise variables (containing no information at all) and look at the performance of a linear model trained on this modified data (10 times repeated 10-fold CV).



EXAMPLE: LINEAR MODEL

We compare the performance of an LM to that of a regression tree.



→ The unregularized LM struggles with the added noise features, while our tree seems to nicely filter them out.

Note: Trees automatically perform feature selection as only one feature at a time is considered for splitting (the smaller the depth of the tree, the less features are selected). Thus, they often perform well in high-dimensional settings.

EXAMPLE: LINEAR MODEL

- The regression coefficients of the noise features can not be estimated precisely as zero in the unregularized LM due to small random correlations.
- With an increasing number of these noise features, the prediction error rises.
- To see this, we can quantify the influence of the noise features on the prediction of each observation.

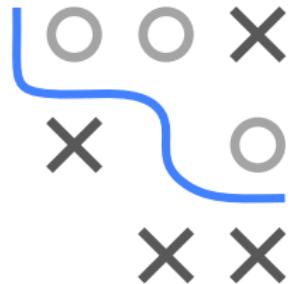
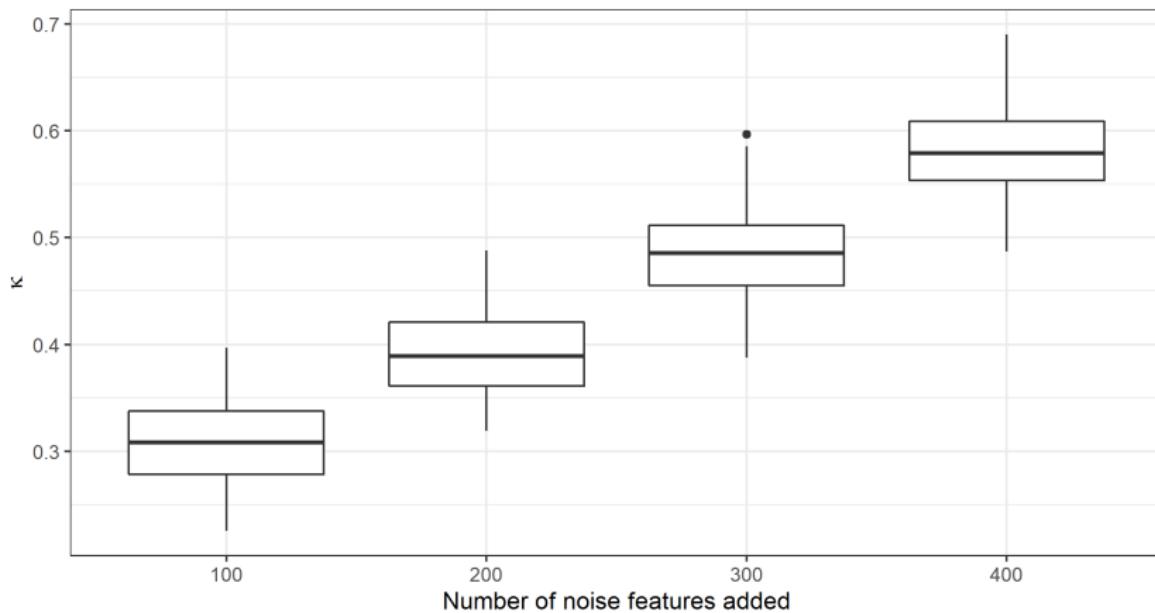


Therefore we decompose the response $\hat{y}^{(i)}$ of each iterations' test set into $\hat{y}_{\text{true}}^{(i)}$ (predicted with noise features set to 0) and $\hat{y}_{\text{noise}}^{(i)}$ (predicted with true features set to 0), s.t.

$$\hat{y}^{(i)} = \hat{y}_{\text{true}}^{(i)} + \hat{y}_{\text{noise}}^{(i)} + \text{intercept}.$$

With this, we can define the “average proportional influence of the noise features” $\kappa := \overline{\left(\frac{|\hat{y}_{\text{noise}}^{(i)}|}{|\hat{y}_{\text{true}}^{(i)}| + |\hat{y}_{\text{noise}}^{(i)}|} \right)}$.

EXAMPLE: LINEAR MODEL



When we add 400 noise features to the model, most of the time, on average, over 50% of the flexible part of the prediction ($\hat{y}^{(i)} - \text{intercept}$) is determined by the noise features.

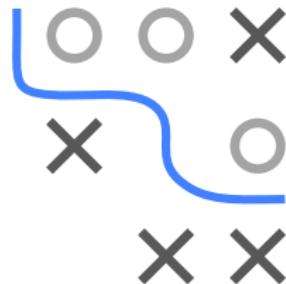
COD: WAYS OUT

Many methods besides k-NN struggle with the curse of dimensionality.

A large part of ML is concerned with dealing with this problem and finding ways around it.

Possible approaches are:

- Increasing the space coverage by gathering more observations
(not always viable in practice!)
- Reducing the number of dimensions before training (e.g. by using domain knowledge, PCA or feature selection)
- Regularization



INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

Boosting

Gaussian Processes



Introduction to Machine Learning

Regularization Introduction



Learning goals

- Overfitting
- Motivation of regularization
- First overview of techniques
- Pattern of regularized ERM formula

WHAT IS REGULARIZATION?

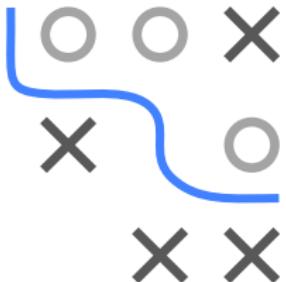
Methods that add **inductive bias** to model, usually some “low complexity” priors (shrinkage and sparsity) to reduce overfitting and get better bias-variance tradeoff



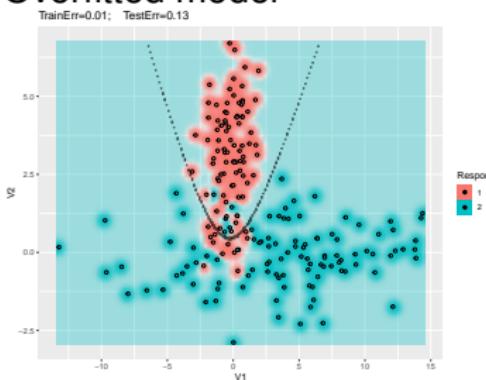
- **Explicit regularization:** penalize explicit measure of model complexity in ERM (e.g., $L1/L2$)
- **Implicit regularization:** early stopping, data augmentation, parameter sharing, dropout or ensembling
- **Structured regularization:** structural prior knowledge over groups of parameters or subnetworks (e.g., group lasso [► Yuan and Lin 2006](#))

RECAP: OVERTFITTING

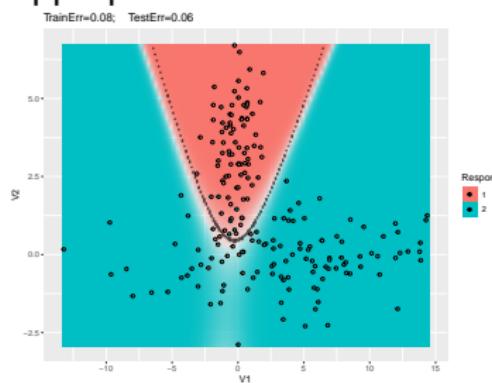
- Occurs when model reflects noise or artifacts in training data
- Model often then does not generalize well (small train error, high test error) – or at least works better on train than on test data



Overfitted model

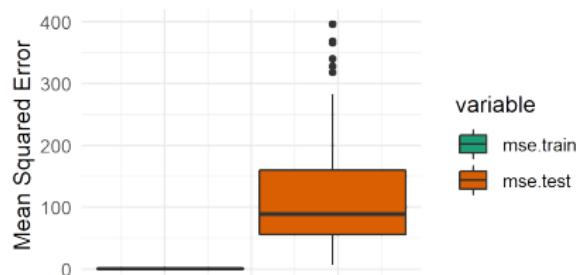


Appropriate model



EXAMPLE I: OVERFITTING

- Data set: daily maximum **ozone level** in LA; $n = 50$
- 12 features: time (weekday, month); weather (temperature at stations, humidity, wind speed); pressure gradient
- Orig. data was subsetted, so it feels “high-dim.” now (low n in relation to p)
- LM with all features (L2 loss)
- MSE evaluation under 10×10 REP-CV



Model fits train data well, but generalizes poorly.

EXAMPLE II: OVERRFITTING

- We train an MLP and a CART on the mtcars data
- Both models are not regularized
- And configured to make overfitting more likely

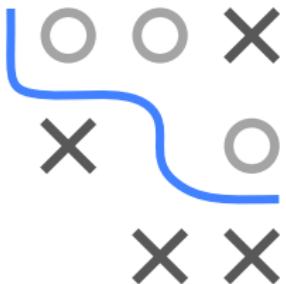


	Train MSE	Test MSE
Neural Network	3.68	19.98
CART	0.00	10.21

(And we now switch back to the Ozone example...)

AVOIDING OVERFITTING – COLLECT MORE DATA

We explore our results for increased dataset size.



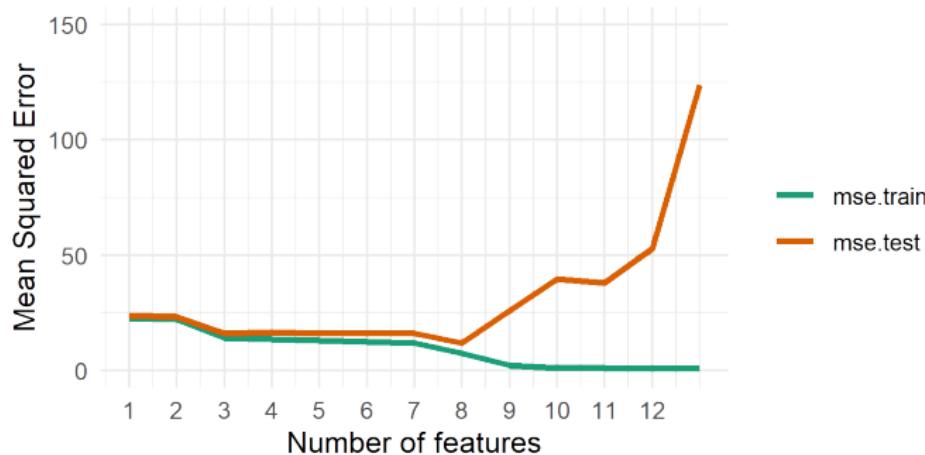
Fit slightly worsens, but test error decreases.

But: Often not feasible in practice.

AVOIDING OVERFITTING – REDUCE COMPLEXITY

We try the simplest model: a constant. So for $L2$ loss the mean of $y^{(i)}$.

We then increase complexity by adding one feature at a time.



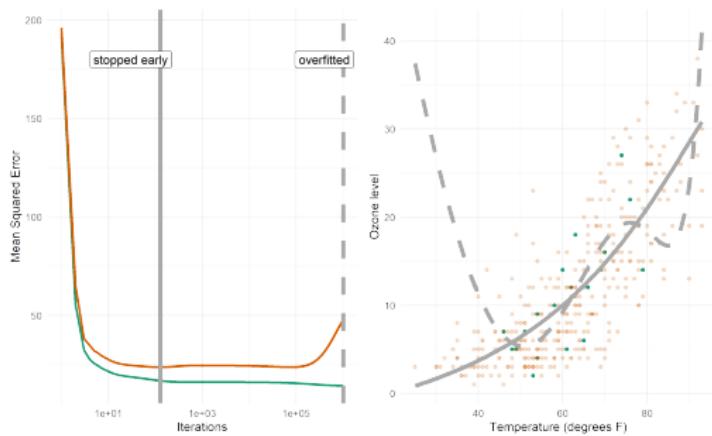
NB: We added features in a specific (clever) order, so we cheated a bit.

AVOIDING OVERFITTING – OPTIMIZE LESS

Now: polynomial regression with temperature as single feature

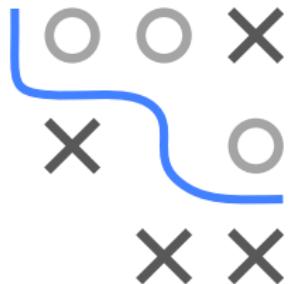
$$f(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{k=0}^d \theta_k \cdot (x_T)^k$$

We set $d = 15$ to overfit to small data. To investigate early stopping, we don't analytically solve the OLS problem, but run GD stepwise.



We see: Early stopping GD can improve results.

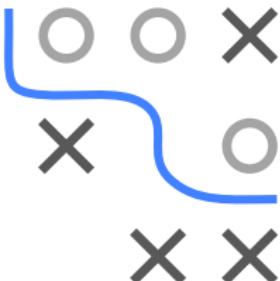
NB: GD for poly-regr usually needs many iters before it starts to overfit, so we used a very small training set.



REGULARIZED EMPIRICAL RISK MINIMIZATION

We have contradictory goals:

- **maximizing fit** (minimizing the train loss)
- **minimizing complexity** of the model

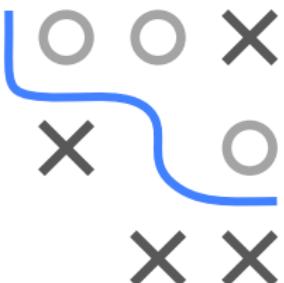


We saw how we can include features in a binary fashion.
But we would rather control complexity **on a continuum**.

REGULARIZED EMPIRICAL RISK MINIMIZATION

Common pattern:

$$\mathcal{R}_{\text{reg}}(f) = \mathcal{R}_{\text{emp}}(f) + \lambda \cdot J(f) = \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) + \lambda \cdot J(f)$$

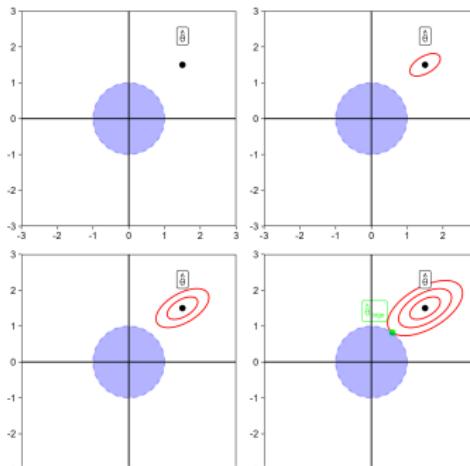


- $J(f)$: **complexity penalty, roughness penalty or regularizer**
- $\lambda \geq 0$: **complexity control** parameter
- The higher λ , the more we penalize complexity
- $\lambda = 0$: We just do simple ERM; $\lambda \rightarrow \infty$: we don't care about loss, models become as "simple" as possible
- λ is hard to set manually and is usually selected via CV
- As for \mathcal{R}_{emp} , \mathcal{R}_{reg} and J are often defined in terms of θ :

$$\mathcal{R}_{\text{reg}}(\theta) = \mathcal{R}_{\text{emp}}(\theta) + \lambda \cdot J(\theta)$$

Introduction to Machine Learning

Regularization Ridge Regression



Learning goals

- Regularized linear model
- Ridge regression / L_2 penalty
- Understand parameter shrinkage
- Understand correspondence to constrained optimization

REGULARIZATION IN LM

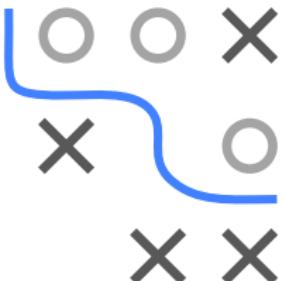
- Can also overfit if p large and n small(er)
- OLS estimator requires full-rank design matrix
- For highly correlated features, OLS becomes sensitive to random errors in response, results in large variance in fit
- We now add a complexity penalty to the loss:

$$\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) = \sum_{i=1}^n \left(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)} \right)^2 + \lambda \cdot J(\boldsymbol{\theta}).$$



RIDGE REGRESSION / L2 PENALTY

Intuitive measure of model complexity is deviation from 0-origin; coeffs then have no or a weak effect. So we measure $J(\theta)$ through a vector norm, shrinking coeffs closer to 0.



$$\begin{aligned}\hat{\theta}_{\text{ridge}} &= \arg \min_{\theta} \sum_{i=1}^n \left(y^{(i)} - \theta^T \mathbf{x}^{(i)} \right)^2 + \lambda \sum_{j=1}^p \theta_j^2 \\ &= \arg \min_{\theta} \|\mathbf{y} - \mathbf{X}\theta\|_2^2 + \lambda \|\theta\|_2^2\end{aligned}$$

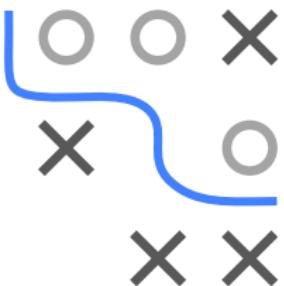
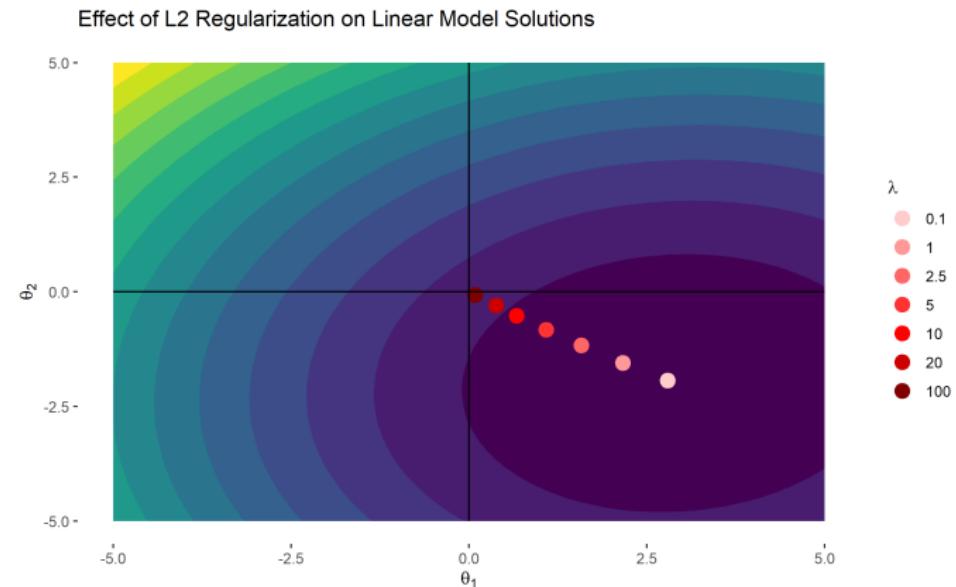
Can still analytically solve this:

$$\hat{\theta}_{\text{ridge}} = (\mathbf{X}^T \mathbf{X} + \lambda I)^{-1} \mathbf{X}^T \mathbf{y}$$

Name: We add pos. entries along the diagonal "ridge" of $\mathbf{X}^T \mathbf{X}$

RIDGE REGRESSION / L2 PENALTY

Let $y = 3x_1 - 2x_2 + \epsilon$, $\epsilon \sim N(0, 1)$. The true minimizer is $\theta^* = (3, -2)^T$, with $\hat{\theta}_{\text{ridge}} = \arg \min_{\theta} \|\mathbf{y} - \mathbf{X}\theta\|^2 + \lambda \|\theta\|^2$.

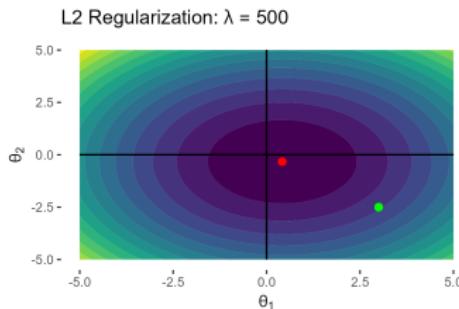
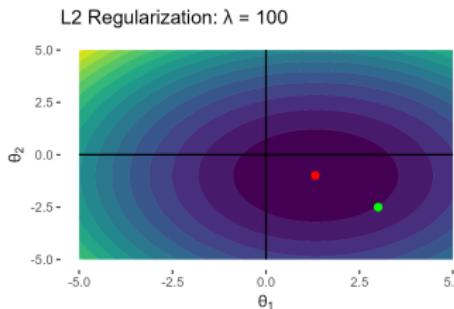
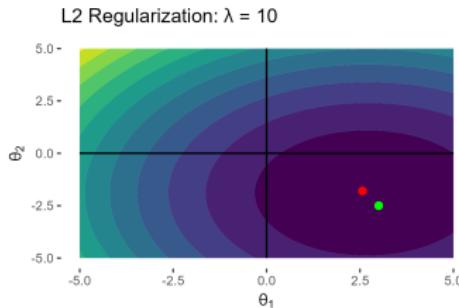
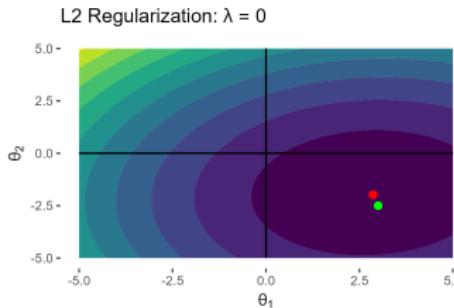


With increasing regularization, $\hat{\theta}_{\text{ridge}}$ is pulled back to the origin
(contour lines show unregularized objective).

RIDGE REGRESSION / L2 PENALTY

Contours of regularized objective for different λ values.

$$\hat{\theta}_{\text{ridge}} = \arg \min_{\theta} \|\mathbf{y} - \mathbf{X}\theta\|^2 + \lambda \|\theta\|^2.$$

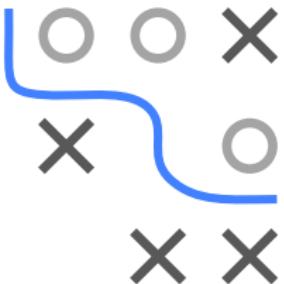
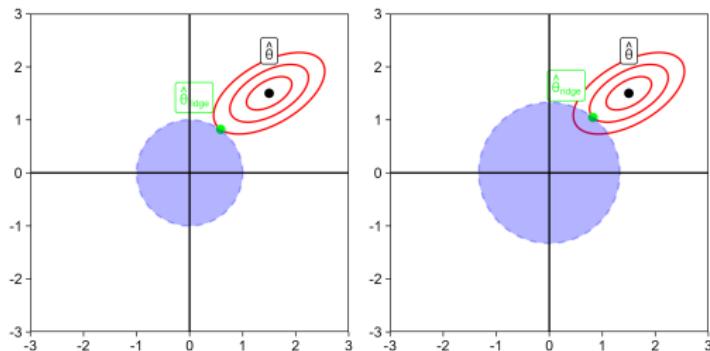


Green = true coeffs of the DGP and red = ridge solution.

RIDGE REGRESSION / L2 PENALTY

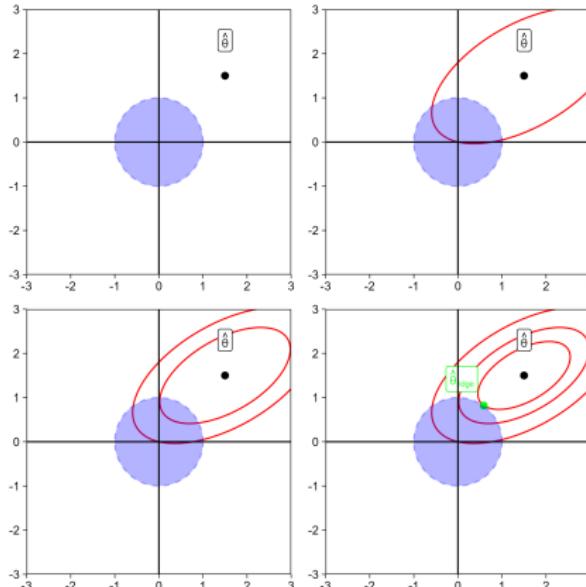
We understand the geometry of these 2 mixed components in our regularized risk objective much better, if we formulate the optimization as a constrained problem (see this as Lagrange multipliers in reverse).

$$\begin{aligned} \min_{\theta} \quad & \sum_{i=1}^n \left(y^{(i)} - f(\mathbf{x}^{(i)} | \theta) \right)^2 \\ \text{s.t.} \quad & \|\theta\|_2^2 \leq t \end{aligned}$$



NB: There is a bijective relationship between λ and t : $\lambda \uparrow \Rightarrow t \downarrow$ and vice versa.

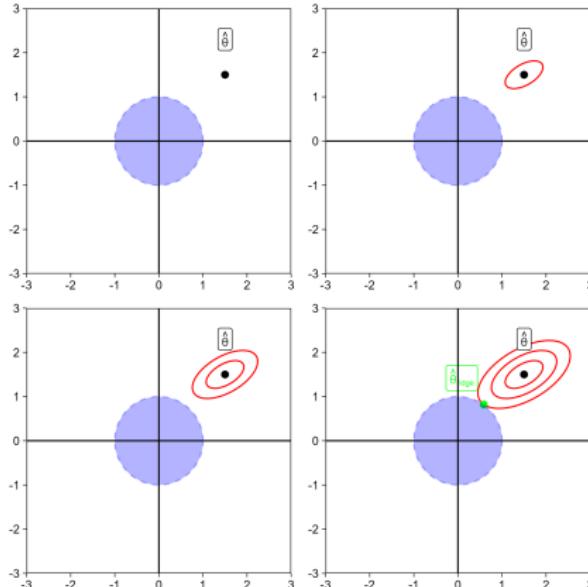
RIDGE REGRESSION / L2 PENALTY



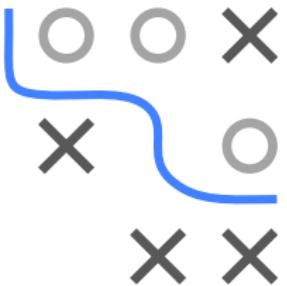
- Inside constraints perspective: From origin, jump from contour line to contour line (better) until you become infeasible, stop before.
- We still optimize the $\mathcal{R}_{\text{emp}}(\theta)$, but cannot leave a ball around the origin.
- $\mathcal{R}_{\text{emp}}(\theta)$ grows monotonically if we move away from $\hat{\theta}$ (elliptic contours).
- Solution path moves from origin to border of feasible region with minimal L_2 distance.



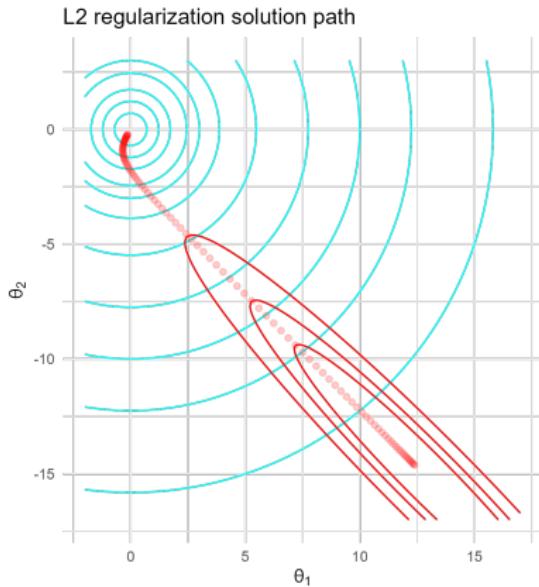
RIDGE REGRESSION / L2 PENALTY



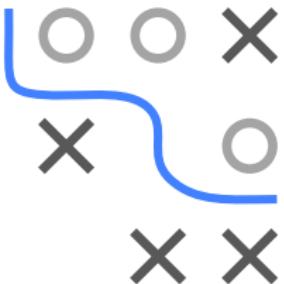
- Outside constraints perspective:
From $\hat{\theta}$, jump from contour line to contour line (worse) until you become feasible, stop then.
- So our new optimum will lie on the boundary of that ball.
- Solution path moves from unregularized estimate to feasible region of regularized objective with minimal L_2 distance.



RIDGE REGRESSION / L2 PENALTY



- Here we can see entire solution path for ridge regression
- Cyan contours indicate feasible regions induced by different λ s
- Red contour lines indicate different levels of the unreg. objective
- Ridge solution (red points) gets pulled toward origin for increasing λ



EXAMPLE: POLYNOMIAL RIDGE REGRESSION

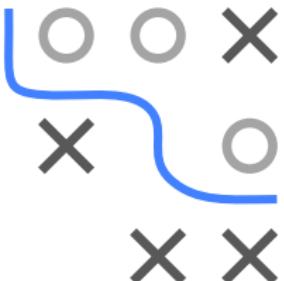
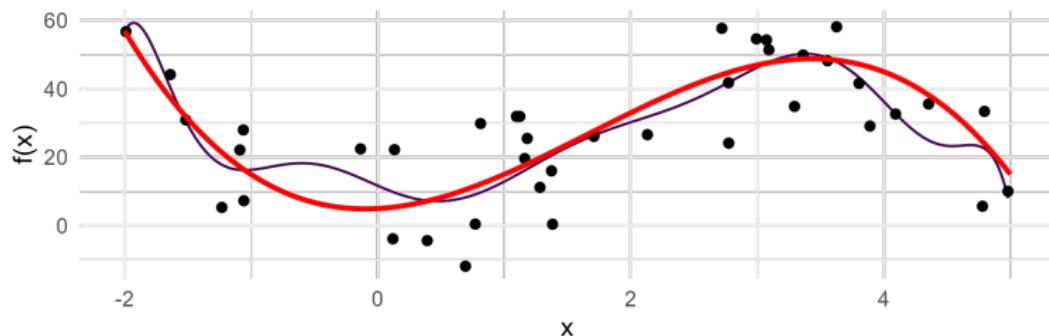
Consider $y = f(x) + \epsilon$ where the true (unknown) function is

$$f(x) = 5 + 2x + 10x^2 - 2x^3 \text{ (in red).}$$

Let's use a d th-order polynomial

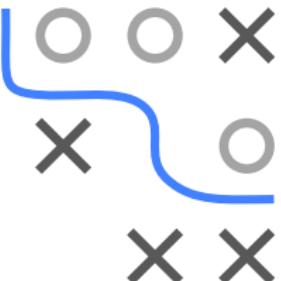
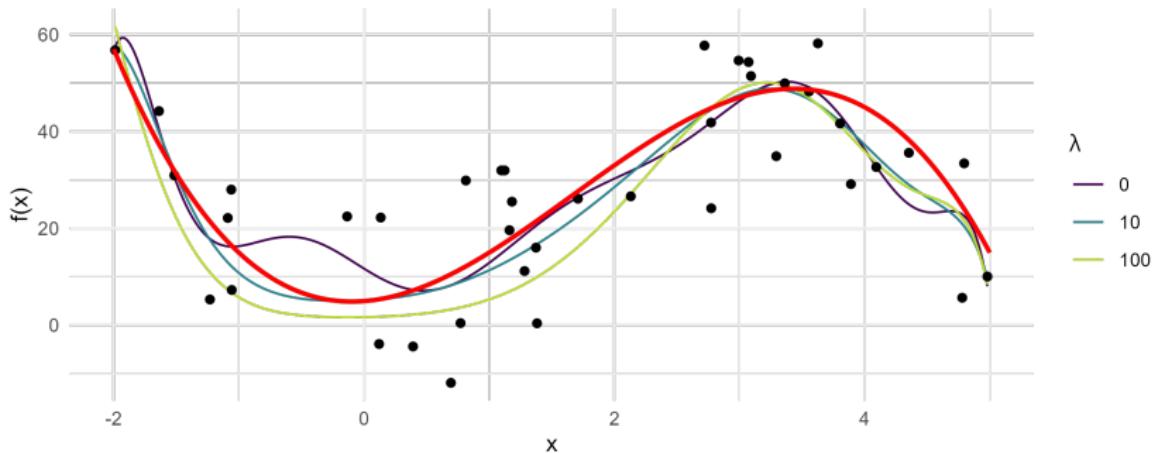
$$f(x) = \theta_0 + \theta_1 x + \cdots + \theta_d x^d = \sum_{j=0}^d \theta_j x^j.$$

Using model complexity $d = 10$ overfits:



EXAMPLE: POLYNOMIAL RIDGE REGRESSION

With an $L2$ penalty we can now select d "too large" but regularize our model by shrinking its coefficients. Otherwise we have to optimize over the discrete d .

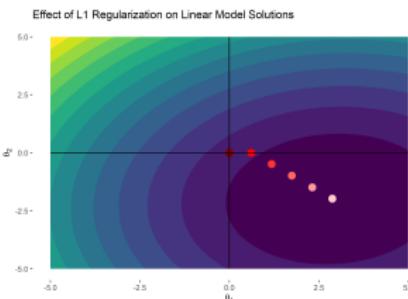


λ	θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}
0.00	12.00	-16.00	4.80	23.00	-5.40	-9.30	4.20	0.53	-0.63	0.13	-0.01
10.00	5.20	1.30	3.70	0.69	1.90	-2.00	0.47	0.20	-0.14	0.03	-0.00
100.00	1.70	0.46	1.80	0.25	1.80	-0.94	0.34	-0.01	-0.06	0.02	-0.00

Introduction to Machine Learning

Regularization

Lasso Regression



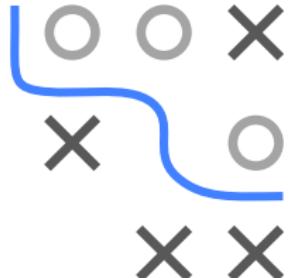
Learning goals

- Lasso regression / $L1$ penalty
- Know that lasso selects features
- Support recovery

LASSO REGRESSION

Another shrinkage method is the so-called **lasso regression** (least absolute shrinkage and selection operator), which uses an $L1$ penalty on θ :

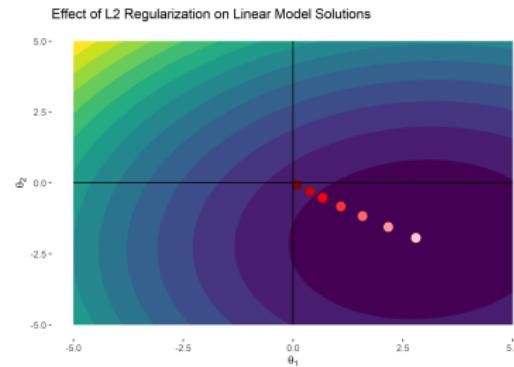
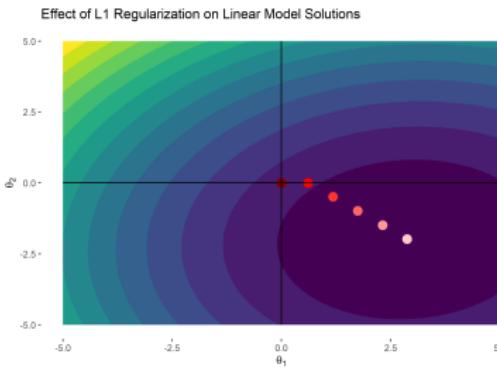
$$\begin{aligned}\hat{\theta}_{\text{lasso}} &= \arg \min_{\theta} \sum_{i=1}^n \left(y^{(i)} - \theta^T \mathbf{x}^{(i)} \right)^2 + \lambda \sum_{j=1}^p |\theta_j| \\ &= \arg \min_{\theta} (\mathbf{y} - \mathbf{X}\theta)^\top (\mathbf{y} - \mathbf{X}\theta) + \lambda \|\theta\|_1\end{aligned}$$



Optimization is much harder now. $\mathcal{R}_{\text{reg}}(\theta)$ is still convex, but in general there is no analytical solution and it is non-differentiable.

LASSO REGRESSION

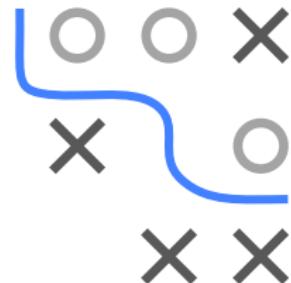
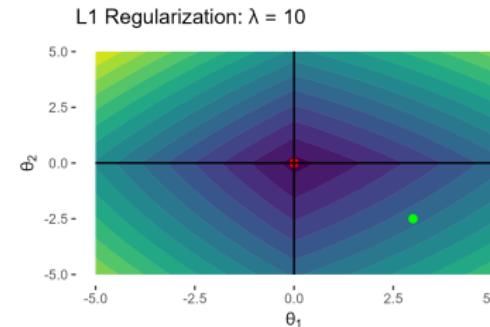
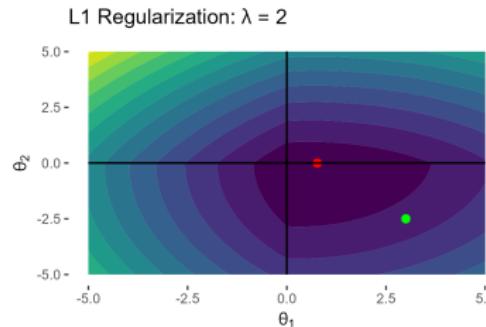
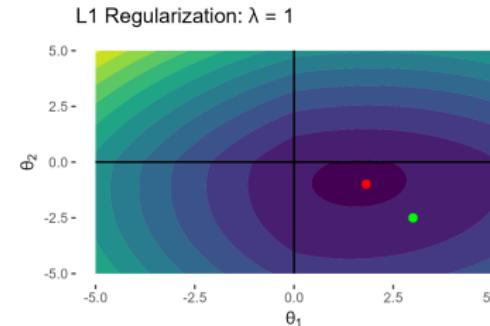
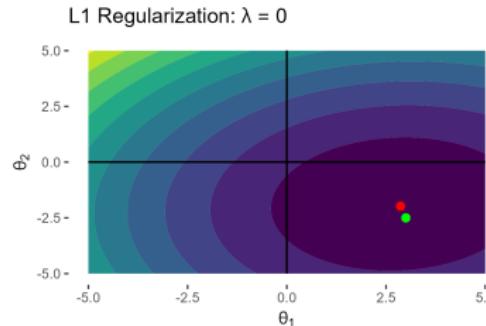
Let $y = 3x_1 - 2x_2 + \epsilon$, $\epsilon \sim N(0, 1)$. The true minimizer is $\theta^* = (3, -2)^T$. LHS = L1 regularization; RHS = L2



With increasing regularization, $\hat{\theta}_{lasso}$ is pulled back to the origin, but takes a different “route”. θ_2 eventually becomes 0!

LASSO REGRESSION

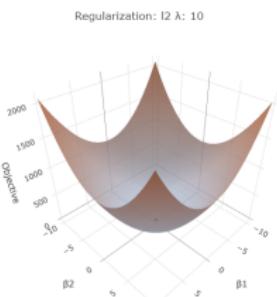
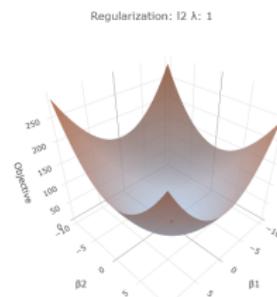
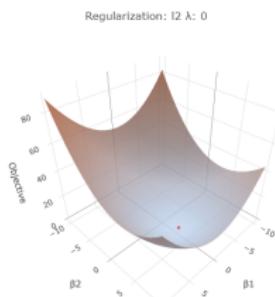
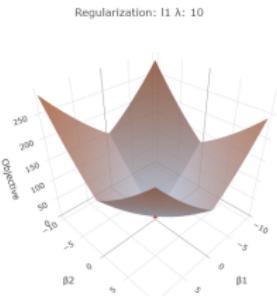
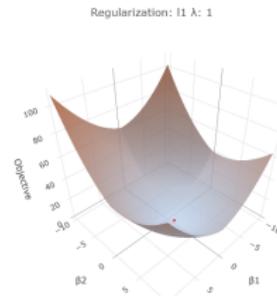
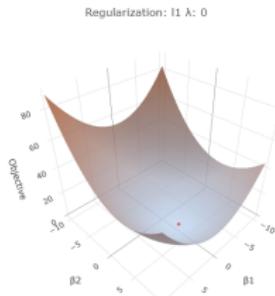
Contours of regularized objective for different λ values.



Green = true minimizer of the unreg.objective and red = lasso solution.

LASSO REGRESSION

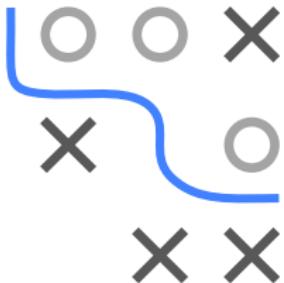
Regularized empirical risk $\mathcal{R}_{\text{reg}}(\theta_1, \theta_2)$ using squared loss for $\lambda \uparrow$. $L1$ penalty makes non-smooth kinks at coordinate axes more pronounced, while $L2$ penalty warps \mathcal{R}_{reg} toward a “basin” (elliptic paraboloid).



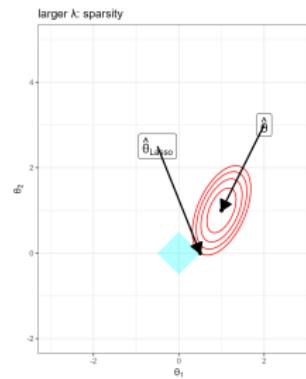
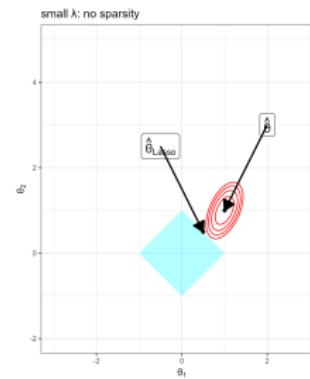
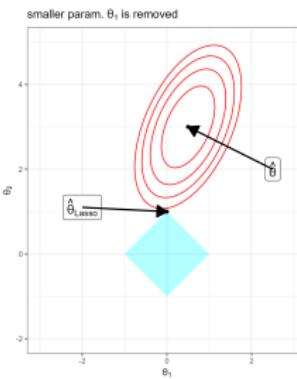
LASSO REGRESSION

We can also rewrite this as a constrained optimization problem. The penalty results in the constrained region to look like a diamond shape.

$$\min_{\theta} \sum_{i=1}^n \left(y^{(i)} - f(\mathbf{x}^{(i)} | \theta) \right)^2 \text{ subject to: } \|\theta\|_1 \leq t$$



The kinks in $L1$ enforce sparse solutions because “the loss contours first hit the sharp corners of the constraint” at coordinate axes where (some) entries are zero.



L1 AND L2 REG. WITH ORTHONORMAL DESIGN

For special case of orthonormal design $\mathbf{X}^\top \mathbf{X} = \mathbf{I}$ we can derive a closed-form solution in terms of $\hat{\theta}_{\text{OLS}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} = \mathbf{X}^\top \mathbf{y}$:

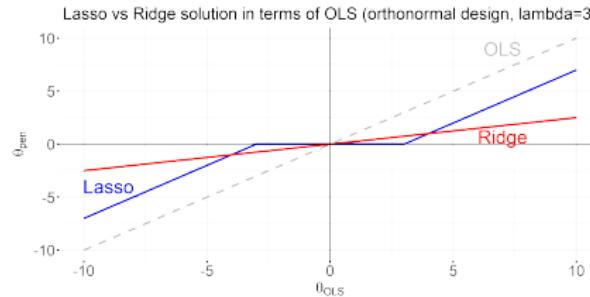
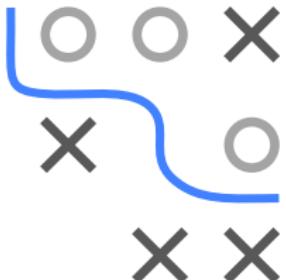
$$\hat{\theta}_{\text{lasso}} = \text{sign}(\hat{\theta}_{\text{OLS}})(|\hat{\theta}_{\text{OLS}}| - \lambda)_+ \quad (\text{sparsity})$$

Function $S(\theta, \lambda) := \text{sign}(\theta)(|\theta| - \lambda)_+$ is called **soft thresholding** operator:

For $|\theta| \leq \lambda$ it returns 0, whereas params $|\theta| > \lambda$ are shrunk toward 0 by λ .

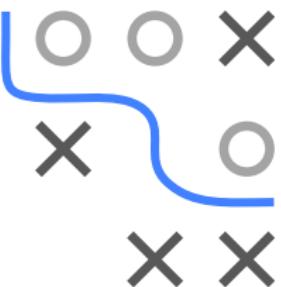
Comparing this to $\hat{\theta}_{\text{Ridge}}$ under orthonormal design:

$$\hat{\theta}_{\text{Ridge}} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y} = ((1 + \lambda) \mathbf{I})^{-1} \hat{\theta}_{\text{OLS}} = \frac{\hat{\theta}_{\text{OLS}}}{1 + \lambda} \quad (\text{no sparsity})$$

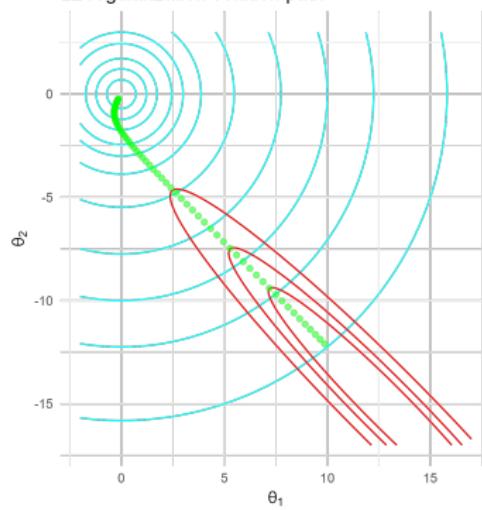


COMPARING SOLUTION PATHS FOR L1/L2

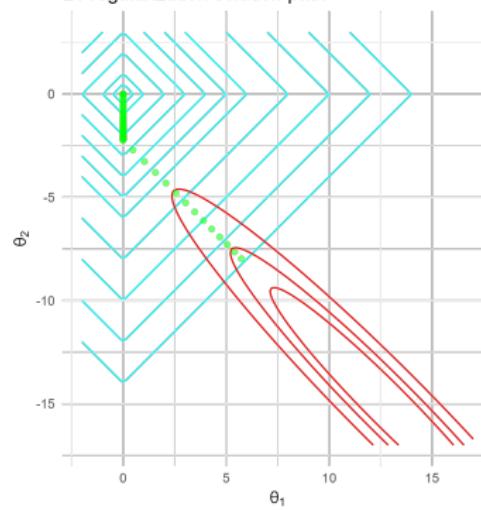
- Ridge results in smooth solution path with non-sparse params
- Lasso induces sparsity, but only for large enough λ



L2 regularization solution path



L1 regularization solution path



SUPPORT RECOVERY OF LASSO

► Zhao and Yu 2006

When can lasso select true support of θ , i.e., only the non-zero parameters?

Can be formalized as sign-consistency:

$$\mathbb{P}(\text{sign}(\hat{\theta}) = \text{sign}(\theta)) \rightarrow 1 \text{ as } n \rightarrow \infty \quad (\text{where } \text{sign}(0) := 0)$$

Suppose the true DGP given a partition into subvectors $\theta = (\theta_1, \theta_2)$ is

$$\mathbf{Y} = \mathbf{X}\theta + \epsilon = \mathbf{X}_1\theta_1 + \mathbf{X}_2\theta_2 + \epsilon \text{ with } \epsilon \sim (0, \sigma^2 \mathbf{I})$$

and only θ_1 is non-zero. Let \mathbf{X}_1 denote the $n \times q$ matrix with the relevant features and \mathbf{X}_2 the matrix of noise features. It can be shown that $\hat{\theta}_{\text{lasso}}$ is sign consistent under an **irrepresentable condition**:

$$|(\mathbf{X}_2^\top \mathbf{X}_1)(\mathbf{X}_1^\top \mathbf{X}_1)^{-1} \text{sign}(\theta_1)| < 1 \text{ (element-wise)}$$

In fact, lasso can only be sign-consistent if this condition holds.

Intuitively, the irrelevant variables in \mathbf{X}_2 must not be too correlated with (or *representable* by) the informative features

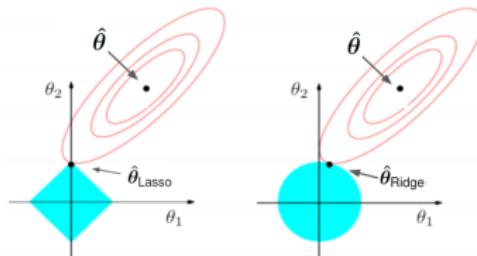
► Meinshausen and Yu 2009



Introduction to Machine Learning

Regularization

Lasso vs. Ridge

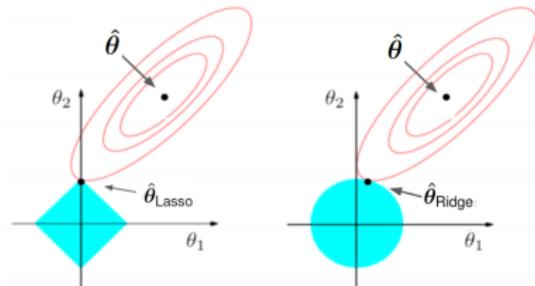


Learning goals

- Properties of ridge vs. lasso
- Coefficient paths
- What happens with corr. features
- Why we need feature scaling

LASSO VS. RIDGE GEOMETRY

$$\min_{\theta} \sum_{i=1}^n \left(y^{(i)} - f(\mathbf{x}^{(i)} | \theta) \right)^2 \quad \text{s.t. } \|\theta\|_p^p \leq t$$

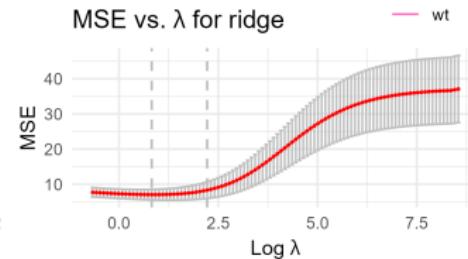
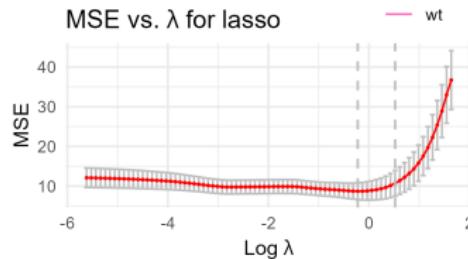
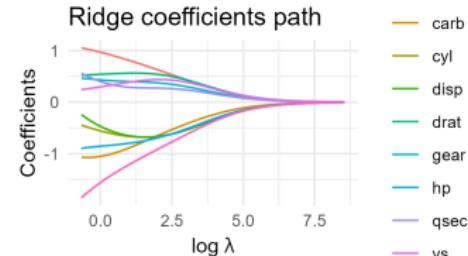
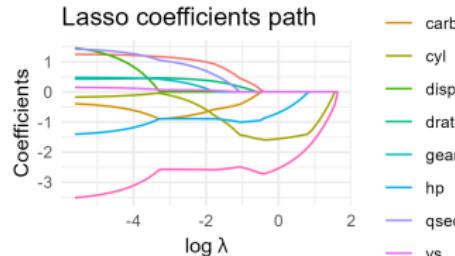
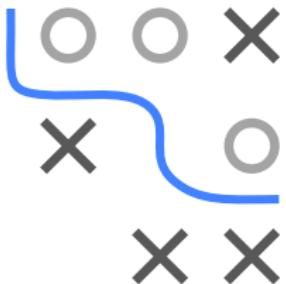


- In both cases (and for sufficiently large λ), the solution which minimizes $\mathcal{R}_{\text{reg}}(\theta)$ is always a point on the boundary of the feasible region.
- As expected, $\hat{\theta}_{\text{Lasso}}$ and $\hat{\theta}_{\text{ridge}}$ have smaller parameter norms than $\hat{\theta}$.
- For lasso, solution likely touches a vertex of constraint region.
Induces sparsity and is a form of variable selection.
- For $p > n$: lasso selects at most n features ► Zou and Hastie 2005.

COEFFICIENT PATHS AND 0-SHRINKAGE

Example 1: Motor Trend Car Roads Test (mtcars)

We see how only lasso shrinks to exactly 0.

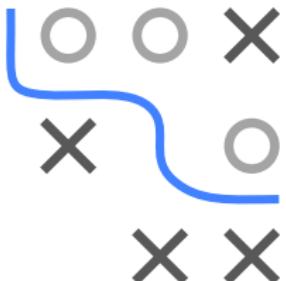


NB: No real overfitting here, as data is so low-dim.

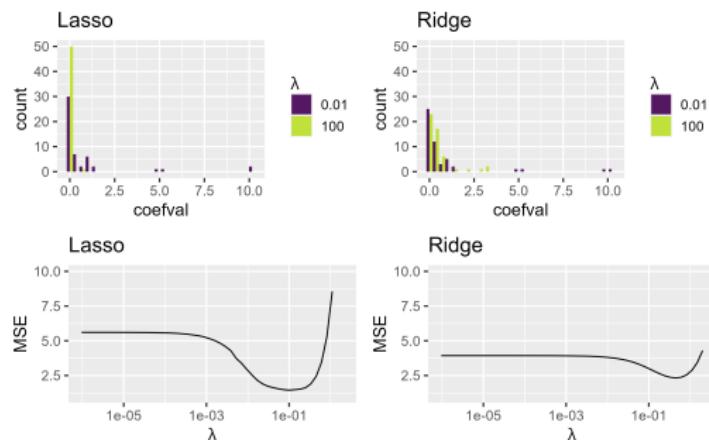
COEFFICIENT PATHS AND 0-SHRINKAGE

Example 2: High-dim., corr. simulated data: $p = 50$; $n = 100$

$$y = 10 \cdot (x_1 + x_2) + 5 \cdot (x_3 + x_4) + 1 \cdot \sum_{j=5}^{14} x_j + \epsilon$$

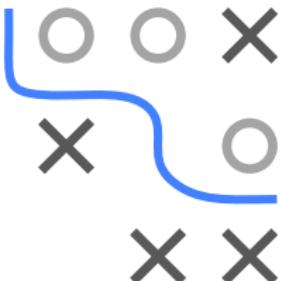


36/50 vars are noise; $\epsilon \sim \mathcal{N}(0, 1)$; $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \Sigma)$; $\Sigma_{k,l} = 0.7^{|k-l|}$



REGULARIZATION AND FEATURE SCALING

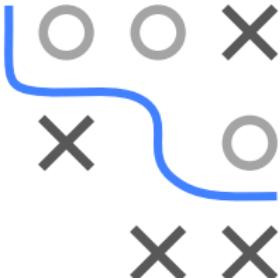
- Typically we omit θ_0 in penalty $J(\theta)$ so that the “infinitely” regularized model is the constant model (but can be implementation-dependent).
- Unregularized LM has **rescaling equivariance**, if you scale some features, can simply “anti-scale” coefs and risk does not change.
- Not true for Reg-LM: if you down-scale features, coeffs become larger to counteract. They are then penalized stronger in $J(\theta)$, making them less attractive without any relevant reason.
- **So: usually standardize features in regularized models, whether linear or non-linear!**



REGULARIZATION AND FEATURE SCALING

- Let the DGP be $y = \sum_{j=1}^5 \theta_j x_j + \varepsilon$ for $\theta = (1, 2, 3, 4, 5)^\top$, $\varepsilon \sim \mathcal{N}(0, 1)$
- Suppose x_5 was measured in m but we change the unit to cm ($\tilde{x}_5 = 100 \cdot x_5$):

Method	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$	$\hat{\theta}_4$	$\hat{\theta}_5$	MSE
OLS	0.984	2.147	3.006	3.918	5.205	0.812
OLS Rescaled	0.984	2.147	3.006	3.918	0.052	0.812



- Estimate $\hat{\theta}_5$ gets scaled by $1/100$ while other estimates and MSE are invariant
- Running ridge regression with $\lambda = 10$ on same data shows that rescaling of x_5 does not result in inverse rescaling of $\hat{\theta}_5$ (everything changes!)
- This is because $\hat{\theta}_5$ now lives on small scale while $L2$ constraint stays the same. Hence remaining estimates can “afford” larger magnitudes.

Method	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$	$\hat{\theta}_4$	$\hat{\theta}_5$	MSE
Ridge	0.709	1.874	2.661	3.558	4.636	1.366
Ridge Rescaled	0.802	1.943	2.675	3.569	0.051	1.08

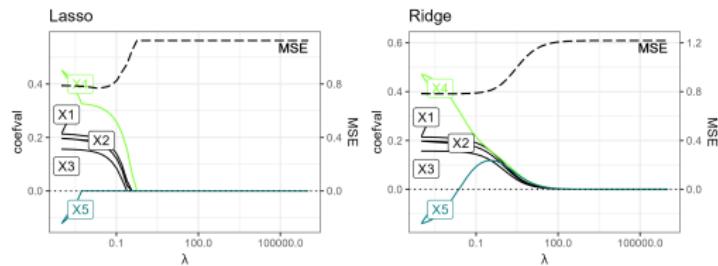
- For lasso, especially for very correlated features, we could arbitrarily force a feature out of the model through a unit change.

CORRELATED FEATURES: L1 VS L2

Simulation with $n = 100$:

$$y = 0.2x_1 + 0.2x_2 + 0.2x_3 + 0.2x_4 + 0.2x_5 + \epsilon$$

x_1-x_4 are independent, but x_4 and x_5 are strongly correlated.



- L1 removes x_5 early, L2 has similar coeffs for x_4, x_5 for larger λ
- Also called “grouping property”: for ridge highly corr. features tend to have equal effects; lasso however “decides” what to select
- L1 selection is somewhat “arbitrary”

CORRELATED FEATURES: L1 VS L2

More detailed answer: The “random” decision is in fact a complex deterministic interaction of data geometry (e.g., corr. structures), the optimization method, and its hyperparameters (e.g., initialization). The theoretical reason for this behavior relates to the convexity of the penalties

► Zou and Hastie 2005

Considering perfectly collinear features $x_4 = x_5$ in the last example, we can obtain some more formal intuition for this phenomenon:

- Because L2 penalty is *strictly* convex:

$$x_4 = x_5 \implies \hat{\theta}_{4,\text{ridge}} = \hat{\theta}_{5,\text{ridge}} \text{ (grouping prop.)}$$

- L1 penalty is not *strictly* convex. Hence, no unique solution exists if $x_4 = x_5$, and sum of coefficients can be arbitrarily allocated to

both features while remaining minimizers (no grouping property!):
For any solution $\hat{\theta}_{4,\text{lasso}}, \hat{\theta}_{5,\text{lasso}}$, equivalent minimizers are given by

$$\tilde{\theta}_{4,\text{lasso}} = s \cdot (\hat{\theta}_{4,\text{lasso}} + \hat{\theta}_{5,\text{lasso}}) \text{ and } \tilde{\theta}_{5,\text{lasso}} = (1 - s) \cdot (\hat{\theta}_{4,\text{lasso}} + \hat{\theta}_{5,\text{lasso}}) \quad \forall s \in [0, 1]$$



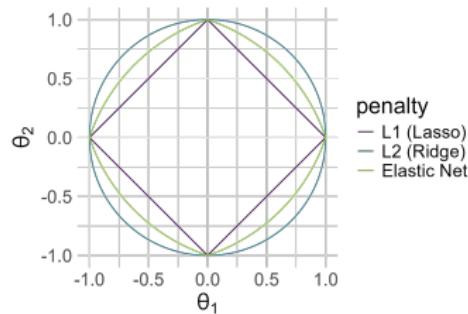
- Neither ridge nor lasso can be classified as better overall
- Lasso can shrink some coeffs to zero, so selects features; ridge usually leads to dense solutions, with smaller coeffs
- Lasso likely better if true underlying structure is sparse
ridge works well if there are many (weakly) influential features
- Lasso has difficulties handling correlated predictors;
for high correlation, ridge dominates lasso in performance
- Lasso: for (highly) correlated predictors, usually an “arbitrary” one
is selected, with large coeff, while the others are (nearly) zeroed
- Ridge: coeffs of correlated features are similar



Introduction to Machine Learning

Regularization

Elastic Net and regularized GLMs



Learning goals

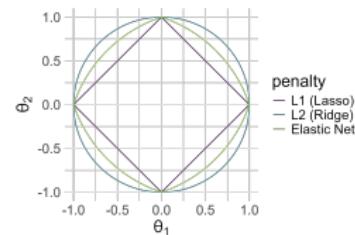
- Compromise between L1 and L2
- Regularized logistic regression

ELASTIC NET AS L1/L2 COMBO

Zou and Hastie 2005

$$\mathcal{R}_{\text{elnet}}(\boldsymbol{\theta}) = \sum_{i=1}^n (y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)})^2 + \lambda_1 \|\boldsymbol{\theta}\|_1 + \lambda_2 \|\boldsymbol{\theta}\|_2^2$$

$$= \sum_{i=1}^n (y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)})^2 + \lambda ((1-\alpha)\|\boldsymbol{\theta}\|_1 + \alpha \|\boldsymbol{\theta}\|_2^2), \quad \alpha = \frac{\lambda_2}{\lambda_1 + \lambda_2}, \quad \lambda = \lambda_1$$



- 2nd formula is simply more convenient to interpret hyperpars;
 λ controls how much we penalize, α sets the “L2-portions”
- Correlated features tend to be either selected or zeroed out together
- Selection of more than n features possible for $p > n$

SIMULATED EXAMPLE

5-fold CV with $n_{train} = 100$ and 20 repetitions with $n_{test} = 10000$ for setups:

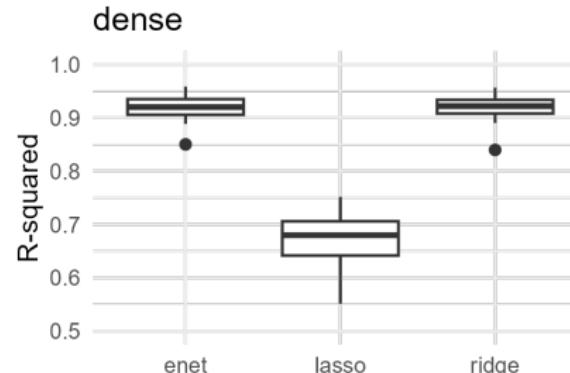
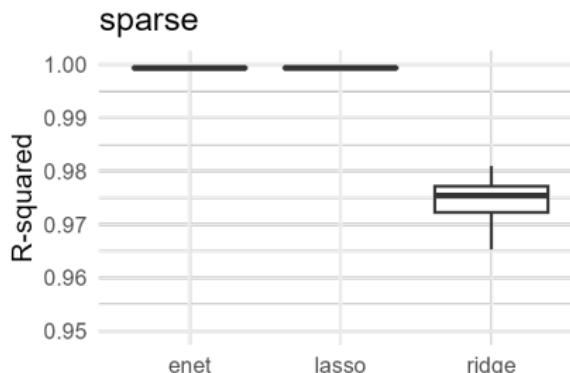
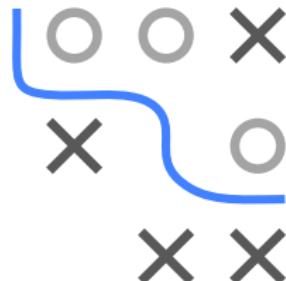
$$y = \mathbf{x}^T \boldsymbol{\theta} + \epsilon; \quad \epsilon \sim N(0, 0.1^2); \quad \mathbf{x} \sim N(0, \Sigma); \quad \Sigma_{k,l} = 0.8^{|k-l|};$$

Lasso better for sparse features:

$$\boldsymbol{\theta} = (\underbrace{1, \dots, 1}_{5}, \underbrace{0, \dots, 0}_{495})$$

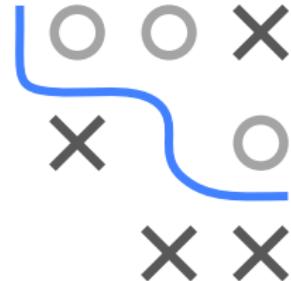
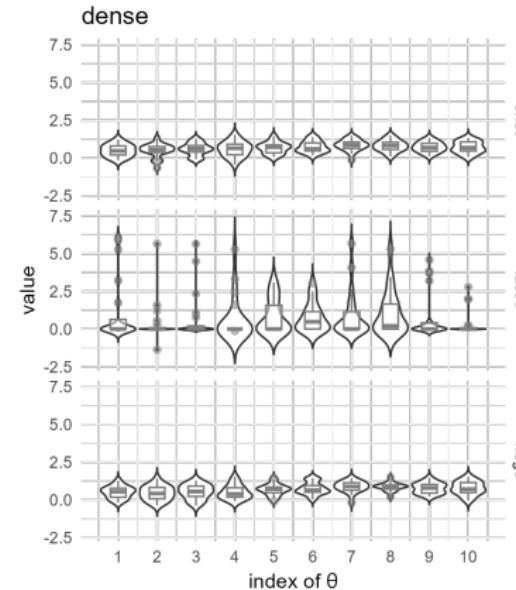
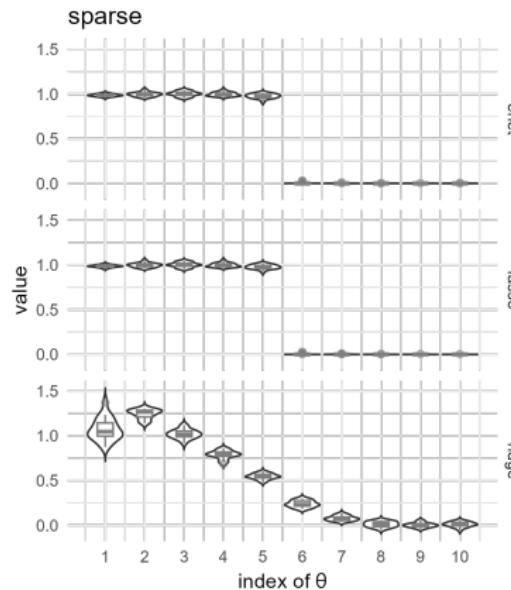
Ridge better for dense features:

$$\boldsymbol{\theta} = (\underbrace{1, \dots, 1, 1}_{500}, \dots, 1)$$



⇒ elastic net handles both cases well

SIMULATED EXAMPLE



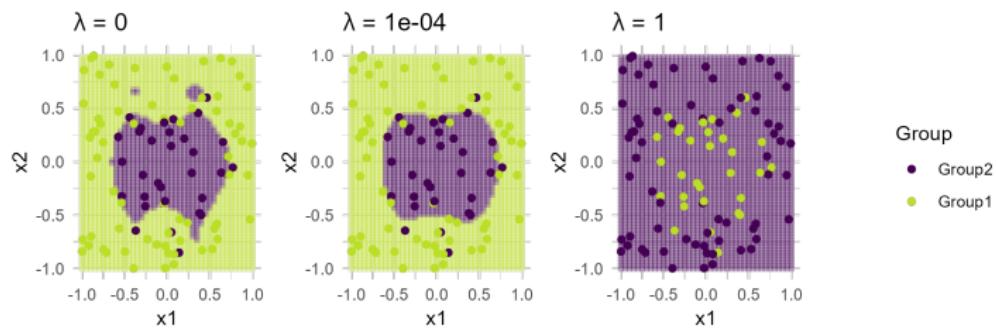
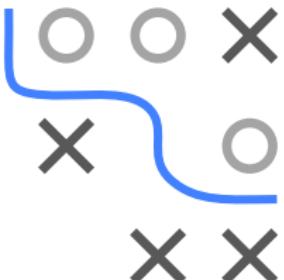
LHS: ridge estimates of noise features hover around 0 while lasso/e-net produce 0s.

RHS: ridge cannot perform variable selection compared to lasso/e-net.

Lasso more frequently ignores relevant features than e-net (longer tails in violin plot).

REGULARIZED LOGISTIC REGRESSION

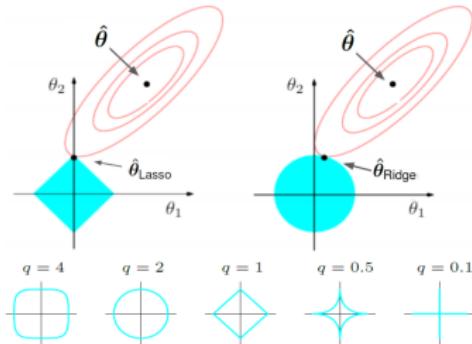
- Penalties can be added very flexibly to any model based on ERM
- E.g.: L_1 - or L_2 -penalized logistic regression for high-dim. spaces and feature selection
- Now: LR with polynomial features for x_1, x_2 up to degree 7 and L2 penalty on 2D “circle data” below



- $\lambda = 0$: LR without penalty seems to overfit
- $\lambda = 0.0001$: We get better
- $\lambda = 1$: Fit looks pretty good

Introduction to Machine Learning

Regularization Other Regularizers



Learning goals

- L1/L2 regularization induces bias
- L q (quasi-)norm regularization
- L0 regularization
- SCAD and MCP



RIDGE AND LASSO ARE BIASED ESTIMATORS

Although ridge and lasso have many nice properties, they are biased estimators and the bias does not (necessarily) vanish as $n \rightarrow \infty$.

For example, in the orthonormal case ($\mathbf{X}^\top \mathbf{X} = \mathbf{I}$) the bias of the lasso is

$$\begin{cases} \mathbb{E} |\hat{\theta}_j - \theta_j| = 0 & \text{if } \theta_j = 0 \\ \mathbb{E} |\hat{\theta}_j - \theta_j| \approx \theta_j & \text{if } |\theta_j| \in [0, \lambda] \\ \mathbb{E} |\hat{\theta}_j - \theta_j| \approx \lambda & \text{if } |\theta_j| > \lambda \end{cases}$$



To reduce the bias/shrinkage of regularized estimators various penalties were proposed, a few of which we briefly introduce now.

Besides $L1/L2$ we could use any Lq (quasi-)norm penalty $\lambda \|\theta\|_q^q$

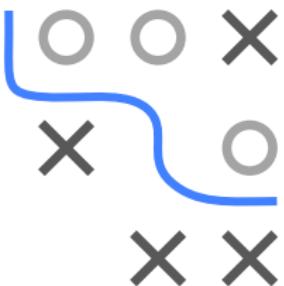
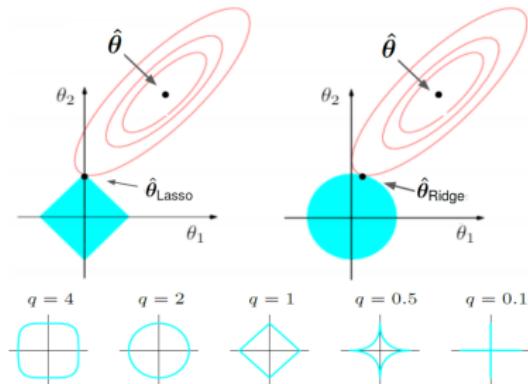
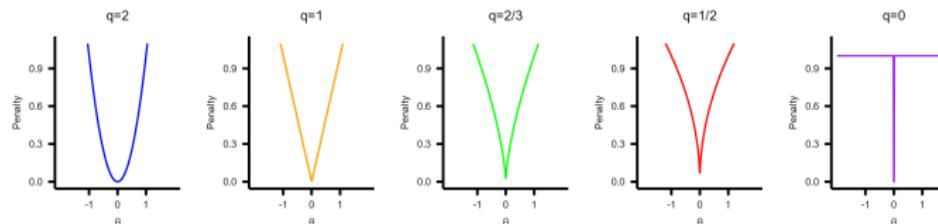


Figure: Top: loss contours and $L1/L2$ constraints. Bottom: Constraints for Lq norms $\sum_j |\theta_j|^q$.

- For $q < 1$ penalty becomes non-convex but for $q > 1$ no sparsity is achieved
- Non-convex Lq has nice properties like **oracle property** ► Zou and Hastie 2005 :
consistent (+ asy. unbiased) param estimation and var selection
- Downside: non-convexity makes optimization even harder than $L1$
(no unique global minimum but multiple local minima)

L0 REGULARIZATION

$$\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) = \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_0 := \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + \lambda \sum_j |\theta_j|^0.$$

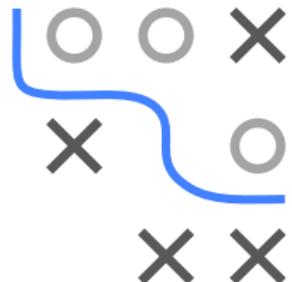


- L0 "norm" simply counts the nr of non-zero params
- Induces sparsity more aggressively than $L1$, but does not shrink
- AIC and BIC are special cases of $L0$
- $L0$ -regularized risk is not continuous or convex
- NP-hard to optimize; for smaller n and p somewhat tractable, efficient approximations are still current research

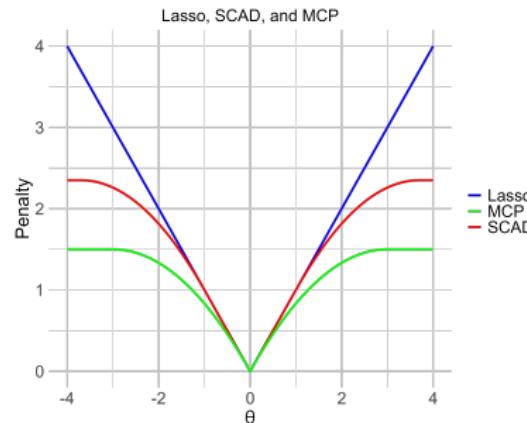
Smoothly Clipped Absolute Deviations:

non-convex, $\gamma > 2$ controls how fast penalty “tapers off”

$$\text{SCAD}(\theta | \lambda, \gamma) = \begin{cases} \lambda|\theta| & \text{if } |\theta| \leq \lambda \\ \frac{2\gamma\lambda|\theta| - \theta^2 - \lambda^2}{2(\gamma-1)} & \text{if } \lambda < |\theta| < \gamma\lambda \\ \frac{\lambda^2(\gamma+1)}{2} & \text{if } |\theta| \geq \gamma\lambda \end{cases}$$



- Lasso, quadratic, then const
- Smooth
- Contrary to lasso/ridge, SCAD continuously relaxes penalization rate as $|\theta|$ increases above λ



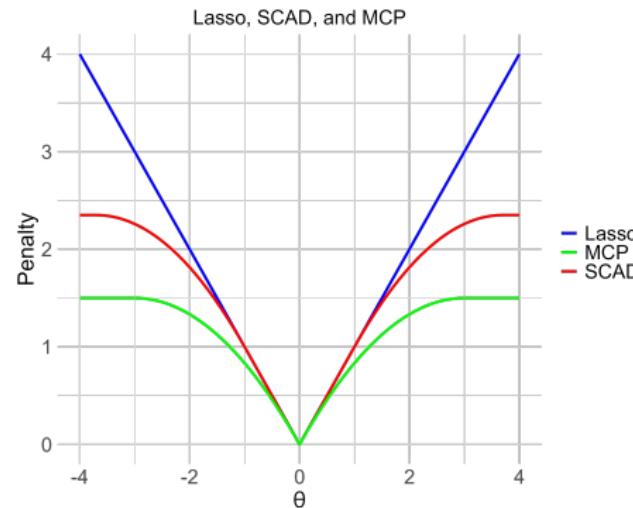
Minimax Concave Penalty:

also non-convex; similar idea as SCAD with $\gamma > 1$

$$MCP(\theta|\lambda, \gamma) = \begin{cases} \lambda|\theta| - \frac{\theta^2}{2\gamma}, & \text{if } |\theta| \leq \gamma\lambda \\ \frac{1}{2}\gamma\lambda^2, & \text{if } |\theta| > \gamma\lambda \end{cases}$$



- As with SCAD, MCP starts by applying same penalization rate as lasso, then smoothly reduces rate to zero as $|\theta| \uparrow$
- Different from SCAD, MCP immediately starts relaxing the penalization rate, while for SCAD rate remains flat until $|\theta| > \lambda$
- Both SCAD and MCP possess oracle property: they can consistently select true model as $n \rightarrow \infty$ while lasso may fail

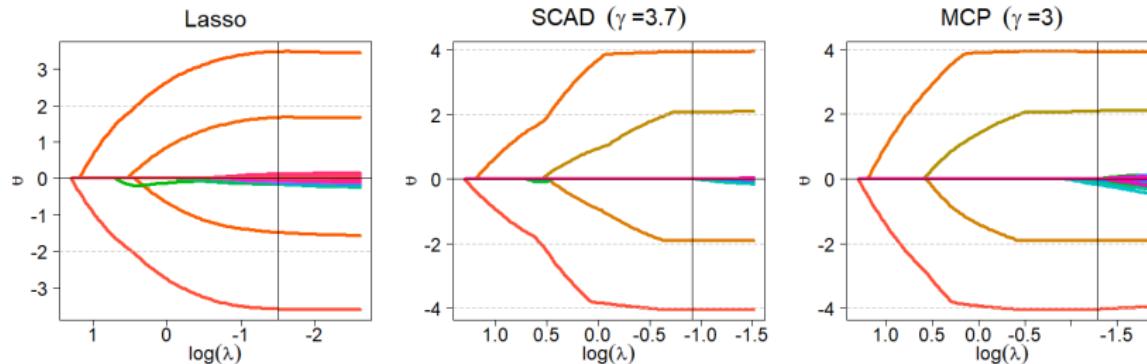


EXAMPLE: COMPARING REGULARIZERS

Let's compare coeff paths for lasso, SCAD, and MCP.

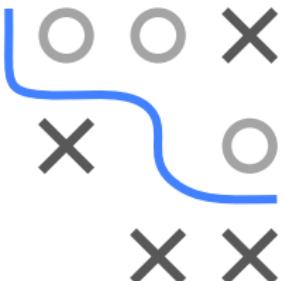
We simulate $n = 100$ samples from the following DGP:

$$y = \mathbf{x}^\top \boldsymbol{\theta} + \varepsilon, \quad \boldsymbol{\theta} = (4, -4, -2, 2, 0, \dots, 0)^\top \in \mathbb{R}^{1500}, \quad x_j, \varepsilon \sim \mathcal{N}(0, 1)$$



Vertical lines mark optimal λ from 10CV.

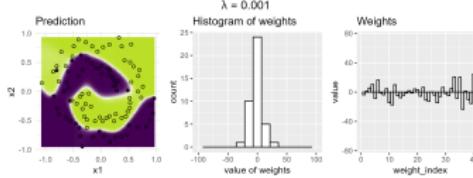
Conclusion: Lasso underestimates true coeffs while SCAD/MCP achieve unbiased estimation and better variable selection



Introduction to Machine Learning

Regularization

Non-Linear Models and Structural Risk Minimization



Learning goals

- Regularization even more important in non-linear models
- Norm penalties applied similarly
- Structural risk minimization

SUMMARY: REGULARIZED RISK MINIMIZATION

If we define (supervised) ML in one line, this might be it:

$$\min_{\theta} \mathcal{R}_{\text{reg}}(\theta) = \min_{\theta} \left(\sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \theta\right)\right) + \lambda \cdot J(\theta) \right)$$

Can choose for task at hand:

- **hypothesis space** of f , controls how features influence prediction
- **loss** function L , measures how errors are treated
- **regularizer** $J(\theta)$, encodes inductive bias



By varying these choices one can construct a huge number of different ML models. Many ML models follow this construction principle or can be interpreted through the lens of RRM.

REGULARIZATION IN NONLINEAR MODELS

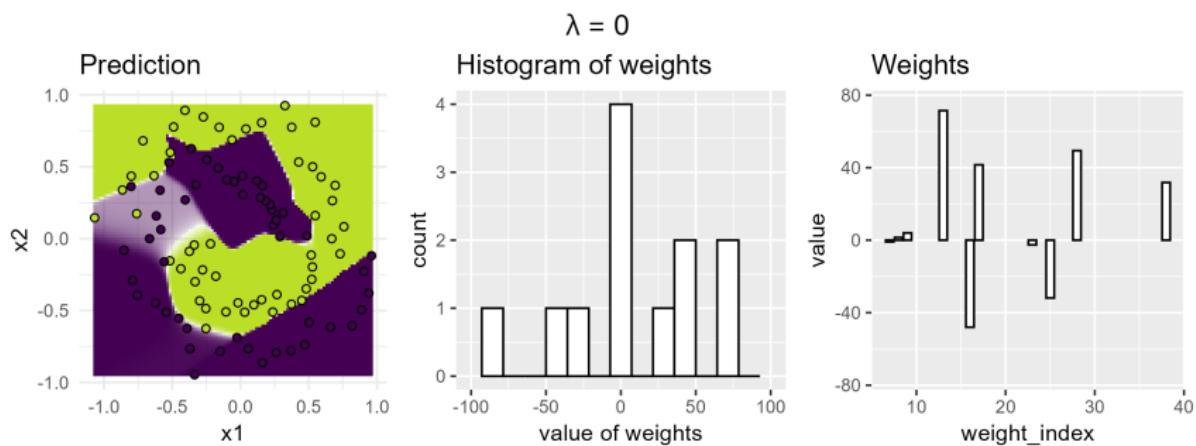
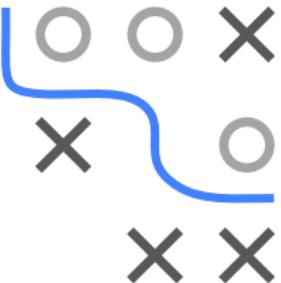
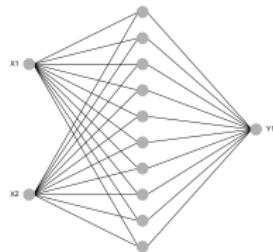
- So far we have mainly considered regularization in LMs
- Can in general also be applied to non-linear models; vector-norm penalties require numeric params
- Here, we typically use L_2 regularization, which still results in parameter shrinkage and weight decay
- For non-linear models, regularization is even more important / basically required to prevent overfitting
- Commonplace in methods such as NNs, SVMs, or boosting
- Prediction surfaces / decision boundaries become smoother



REGULARIZATION IN NONLINEAR MODELS

Classification for spirals data.

NN with single hidden layer, size 10, L_2 penalty:

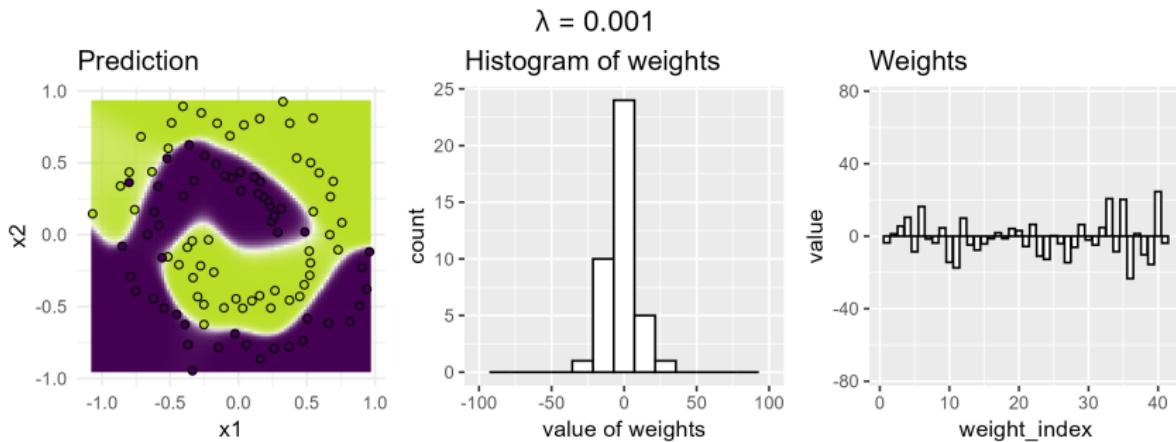
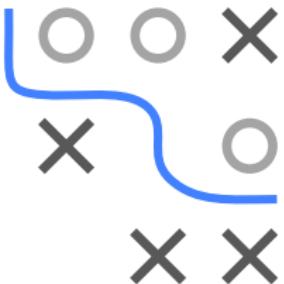
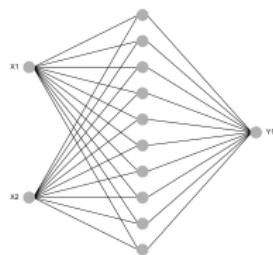


λ affects smoothness of decision boundary and magnitude of weights

REGULARIZATION IN NONLINEAR MODELS

Classification for spirals data.

NN with single hidden layer, size 10, L_2 penalty:

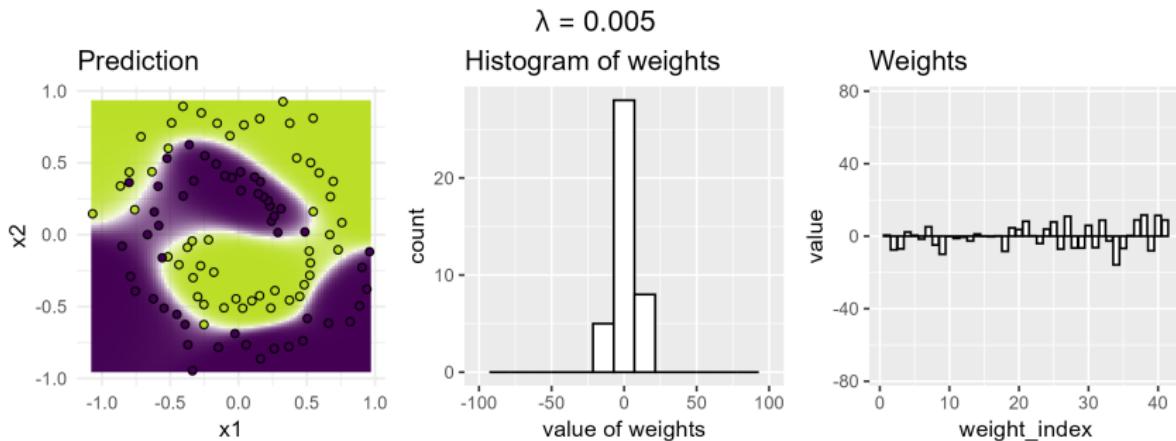
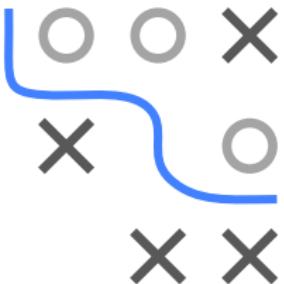
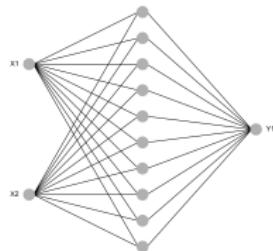


λ affects smoothness of decision boundary and magnitude of weights

REGULARIZATION IN NONLINEAR MODELS

Classification for spirals data.

NN with single hidden layer, size 10, L_2 penalty:

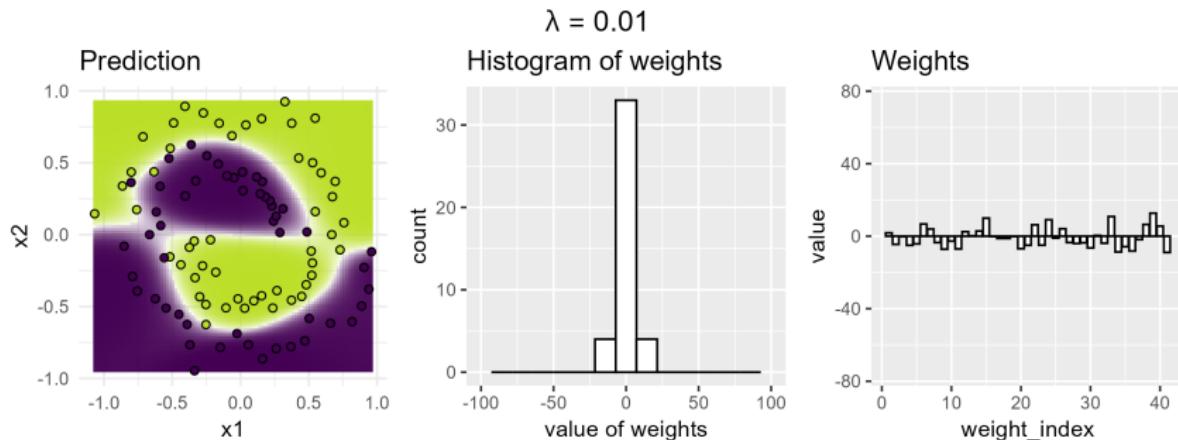
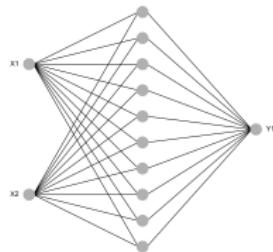


λ affects smoothness of decision boundary and magnitude of weights

REGULARIZATION IN NONLINEAR MODELS

Classification for spirals data.

NN with single hidden layer, size 10, L_2 penalty:

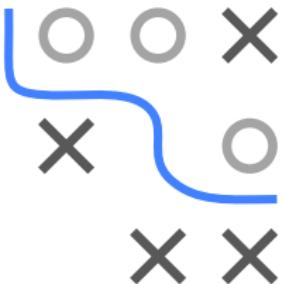
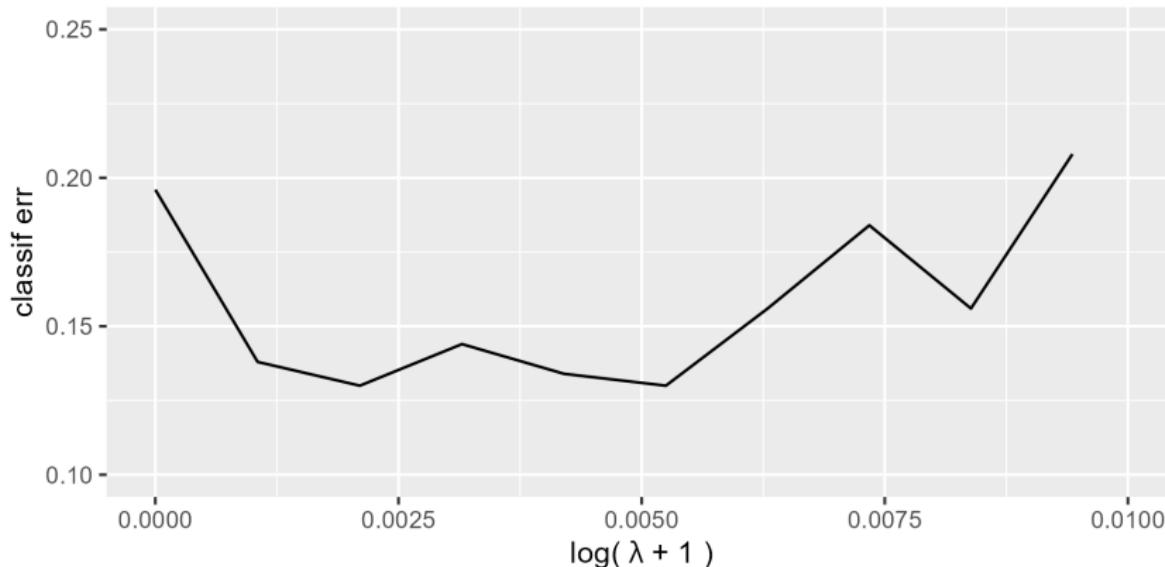


λ affects smoothness of decision boundary and magnitude of weights

REGULARIZATION IN NONLINEAR MODELS

Prevention of overfitting can also be seen in CV.

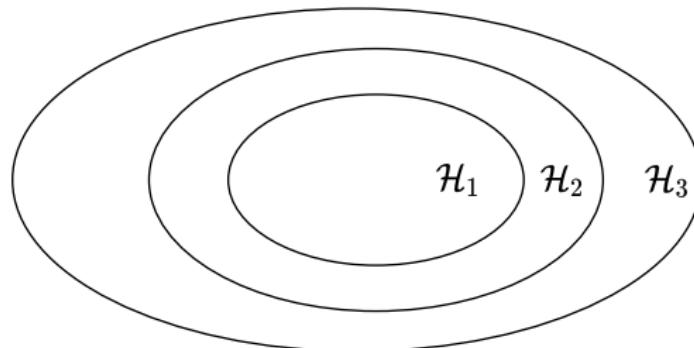
Same settings as before, but each λ is evaluated with 5x10 REP-CV



Typical U-shape with sweet spot between overfitting and underfitting

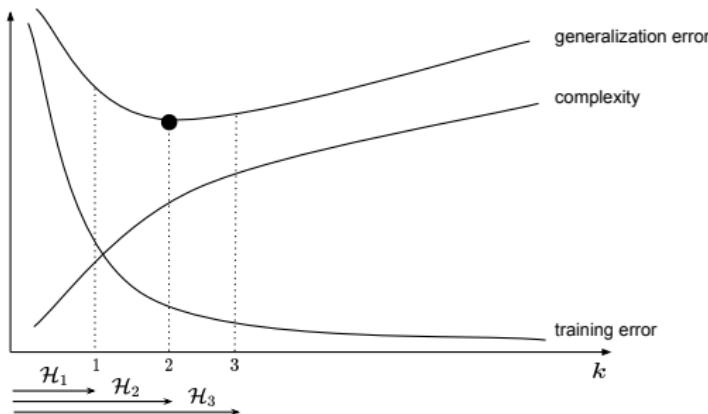
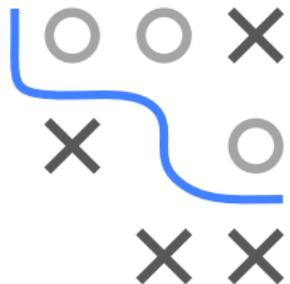
STRUCTURAL RISK MINIMIZATION

- Can also see this as an iterative process;
more a “discrete” view on things
- SRM assumes that \mathcal{H} can be decomposed into increasingly complex hypotheses: $\mathcal{H} = \cup_{k \geq 1} \mathcal{H}_k$
- Complexity parameters can be, e.g. the degree of polynomials in linear models or the size of hidden layers in neural networks



STRUCTURAL RISK MINIMIZATION

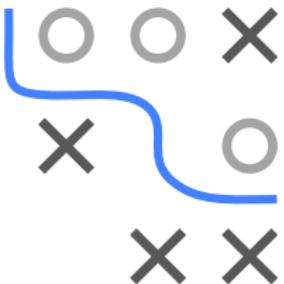
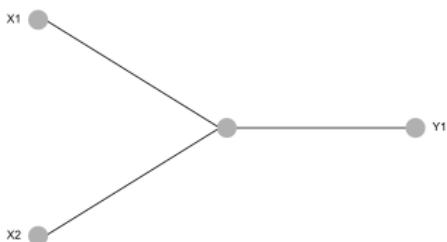
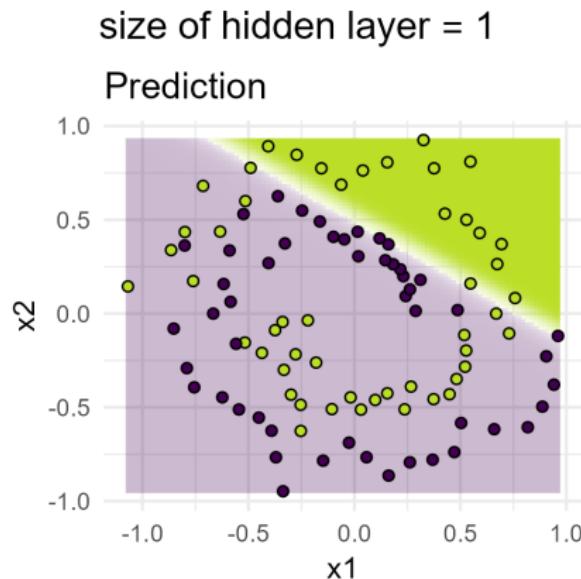
- SRM chooses the smallest k such that the optimal model from \mathcal{H}_k found by ERM or RRM cannot significantly be outperformed by a model from a \mathcal{H}_m with $m > k$
- Principle of Occam's razor
- One challenge might be choosing an adequate complexity measure, as for some models, multiple exist



STRUCTURAL RISK MINIMIZATION

Again spirals.

NN with 1 hidden layer, and fixed (small) L2 penalty.

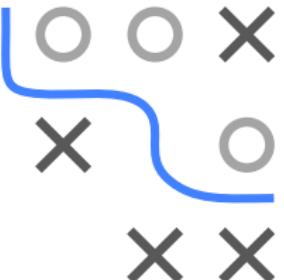


Size affects complexity and smoothness of decision boundary

STRUCTURAL RISK MINIMIZATION

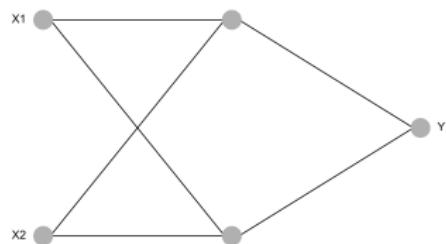
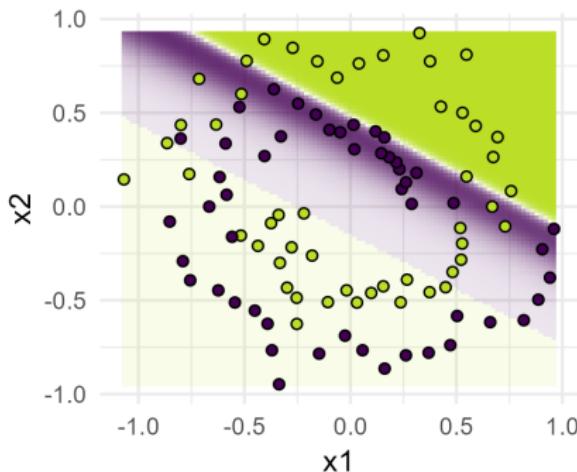
Again spirals.

NN with 1 hidden layer, and fixed (small) L2 penalty.



size of hidden layer = 2

Prediction



Size affects complexity and smoothness of decision boundary

STRUCTURAL RISK MINIMIZATION

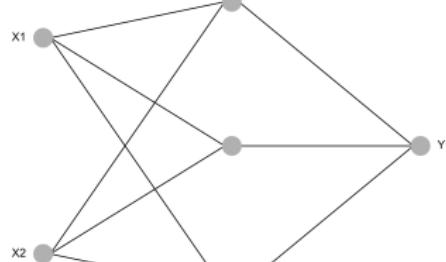
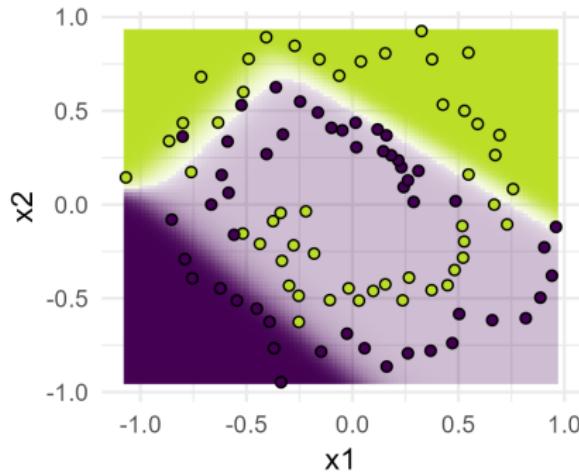
Again spirals.

NN with 1 hidden layer, and fixed (small) L2 penalty.



size of hidden layer = 3

Prediction

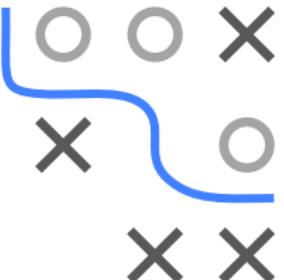


Size affects complexity and smoothness of decision boundary

STRUCTURAL RISK MINIMIZATION

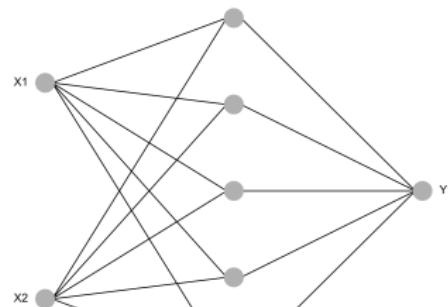
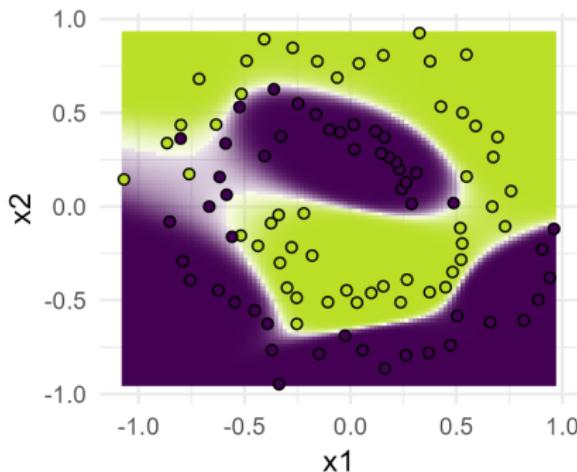
Again spirals.

NN with 1 hidden layer, and fixed (small) L2 penalty.



size of hidden layer = 5

Prediction

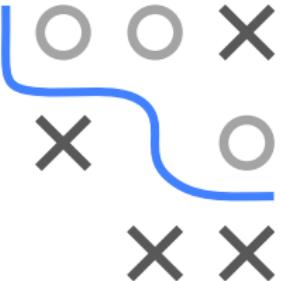


Size affects complexity and smoothness of decision boundary

STRUCTURAL RISK MINIMIZATION

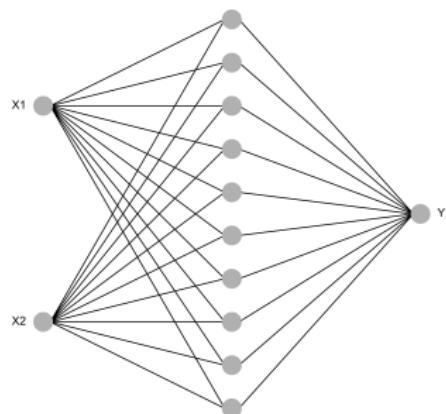
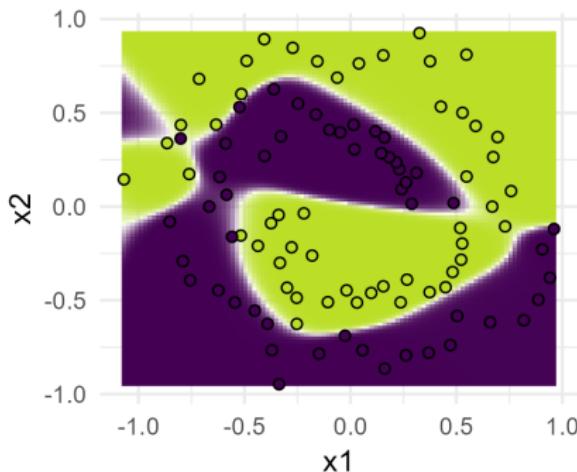
Again spirals.

NN with 1 hidden layer, and fixed (small) L2 penalty.



size of hidden layer = 10

Prediction

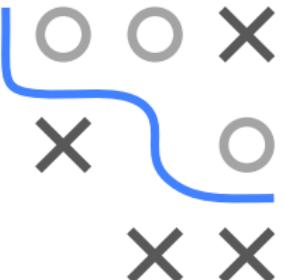


Size affects complexity and smoothness of decision boundary

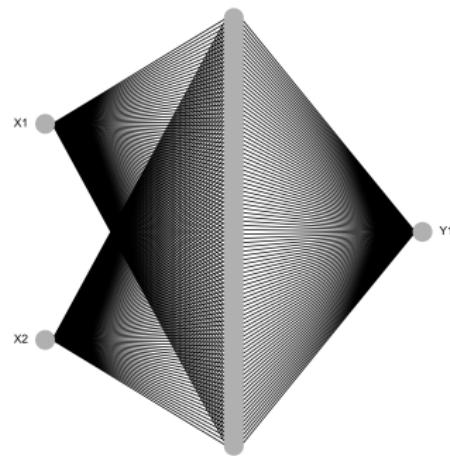
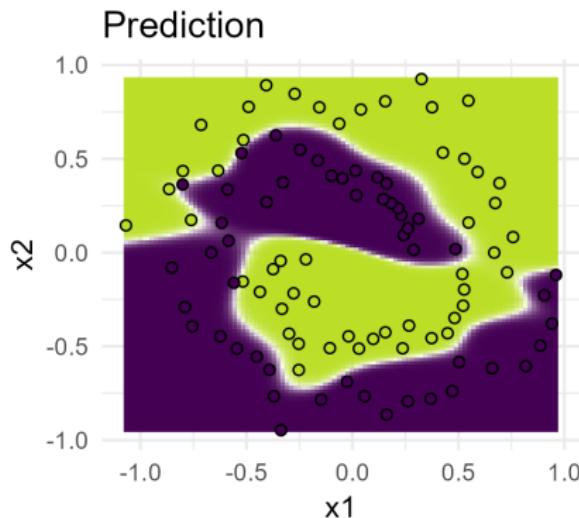
STRUCTURAL RISK MINIMIZATION

Again spirals.

NN with 1 hidden layer, and fixed (small) L2 penalty.



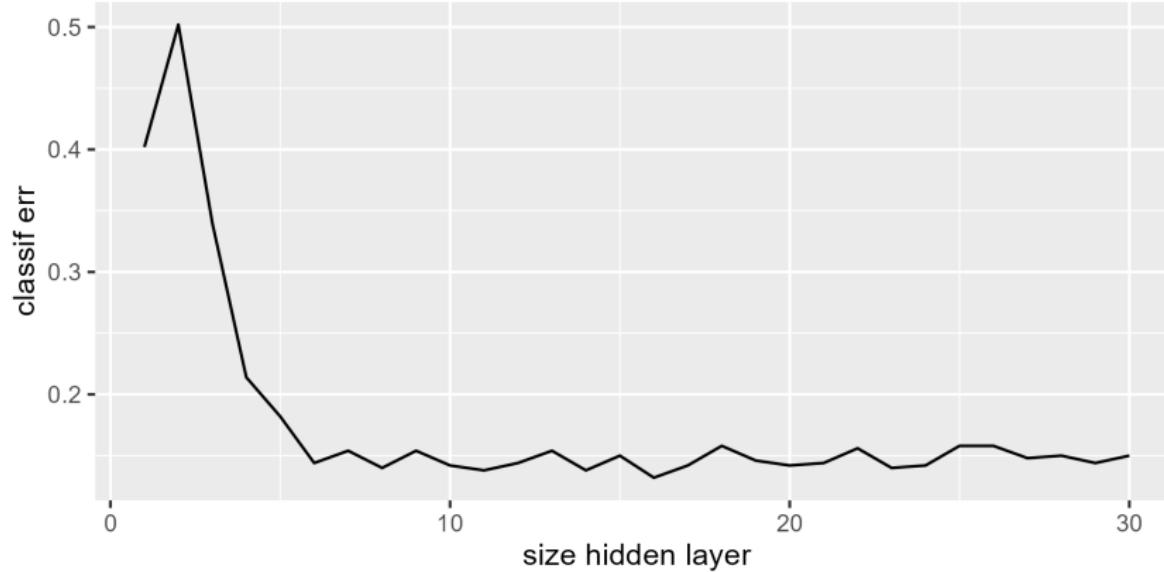
size of hidden layer = 100



Size affects complexity and smoothness of decision boundary

STRUCTURAL RISK MINIMIZATION

Again, complexity vs CV score.

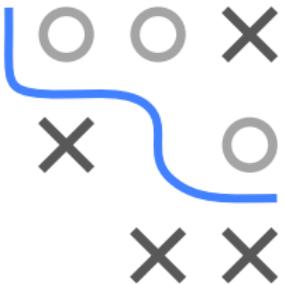
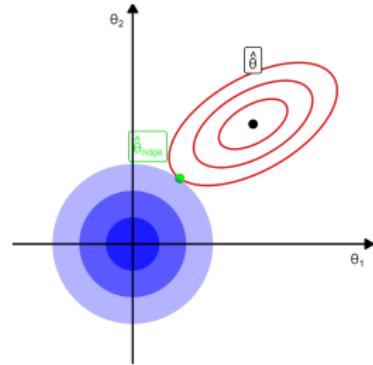


Minimal model with good generalization seems to size=10

STRUCTURAL RISK MINIMIZATION AND RRM

RRM can also be interpreted through SRM,
if we rewrite it in constrained form:

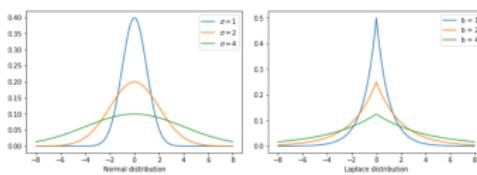
$$\begin{aligned} \min_{\theta} \quad & \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)} | \theta)) \\ \text{s.t.} \quad & \|\theta\|_2^2 \leq t \end{aligned}$$



Can interpret going through λ from large to small as through t from small to large. Constructs series of ERM problems with hypothesis spaces \mathcal{H}_λ , where we constrain norm of θ to unit balls of growing sizes.

Introduction to Machine Learning

Regularization Bayesian Priors



Learning goals

- RRM is same as MAP in Bayes
- Gaussian/Laplace prior corresponds to L_2/L_1 penalty

RRM VS. BAYES

We already created a link between max. likelihood estimation and ERM.

Now we will generalize this for RRM.

Assume we have a parameterized distribution $p(y|\theta, \mathbf{x})$ for our data and a prior $q(\theta)$ over our param space, all in Bayesian framework.

From Bayes theorem:

$$p(\theta|\mathbf{x}, y) = \frac{p(y|\theta, \mathbf{x})q(\theta)}{p(y|\mathbf{x})} \propto p(y|\theta, \mathbf{x})q(\theta)$$

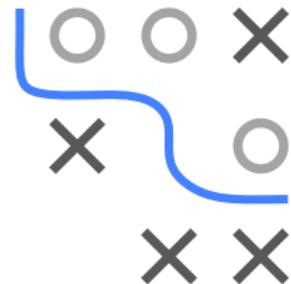


RRM VS. BAYES

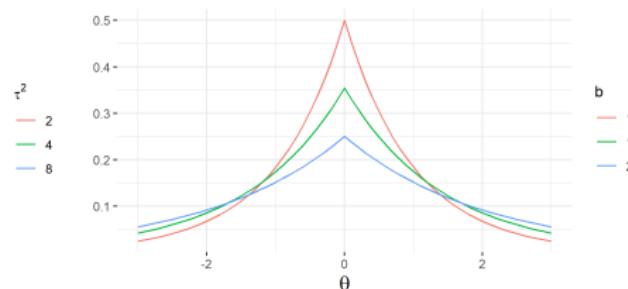
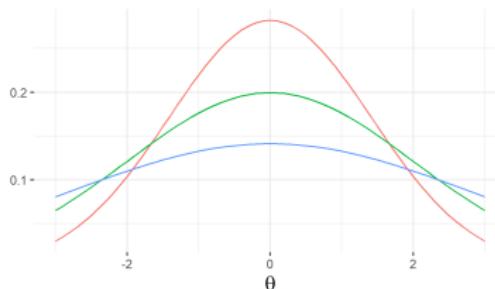
The maximum a posteriori (MAP) estimator of θ is now the minimizer of

$$-\log p(y | \theta, \mathbf{x}) - \log q(\theta).$$

- Again, we identify the loss $L(y, f(\mathbf{x} | \theta))$ with $-\log(p(y|\theta, \mathbf{x}))$.
- If $q(\theta)$ is constant (i.e., we used a uniform, non-informative prior), the second term is irrelevant and we arrive at ERM.
- If not, we can identify $J(\theta) \propto -\log(q(\theta))$, i.e., the log-prior corresponds to the regularizer, and the additional λ , which controls the strength of our penalty, usually influences the peakedness / inverse variance / strength of our prior.



RRM VS. BAYES



- L_2 regularization corresponds to a zero-mean Gaussian prior with constant variance on our parameters: $\theta_j \sim \mathcal{N}(0, \tau^2)$
- L_1 corresponds to a zero-mean Laplace prior: $\theta_j \sim \text{Laplace}(0, b)$. $\text{Laplace}(\mu, b)$ has density $\frac{1}{2b} \exp(-\frac{|\mu-x|}{b})$, with scale parameter b , mean μ and variance $2b^2$.
- In both cases, regularization strength increases as variance of prior decreases: more prior mass concentrated around 0 encourages shrinkage.
- Elastic-net regularization corresponds to a compromise between Gaussian and Laplacian priors ▶ Zou and Hastie 2005 ▶ Hans 2011

EXAMPLE: BAYESIAN L2 REGULARIZATION

We can easily see the equivalence of $L2$ regularization and a Gaussian prior:

- Gaussian prior $\mathcal{N}_d(\mathbf{0}, \text{diag}(\tau^2))$ with uncorrelated components for θ :

$$q(\theta) = \prod_{j=1}^d \phi_{0, \tau^2}(\theta_j) = (2\pi\tau^2)^{-\frac{d}{2}} \exp\left(-\frac{1}{2\tau^2} \sum_{j=1}^d \theta_j^2\right)$$

- MAP:

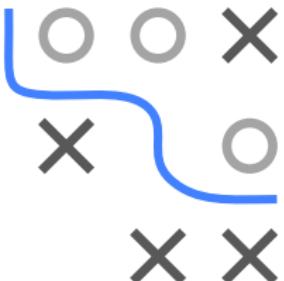
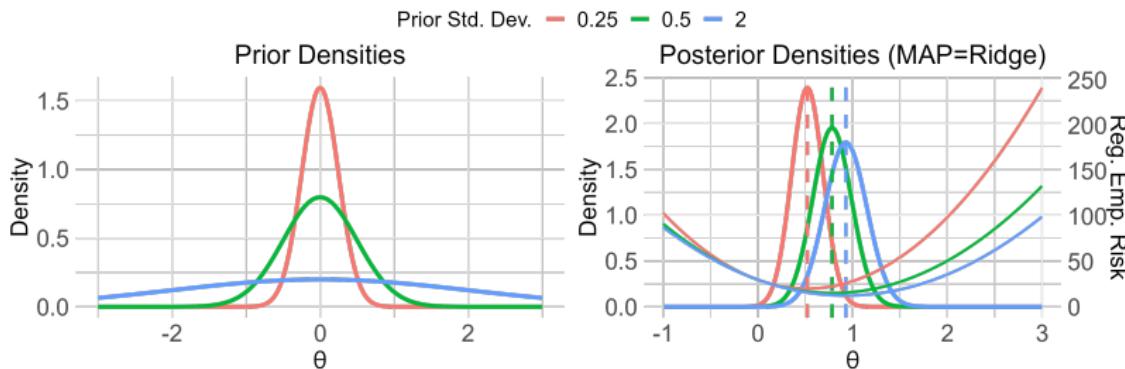
$$\begin{aligned}\hat{\theta}^{\text{MAP}} &= \arg \min_{\theta} (-\log p(y | \theta, \mathbf{x}) - \log q(\theta)) \\ &= \arg \min_{\theta} \left(-\log p(y | \theta, \mathbf{x}) + \frac{d}{2} \log(2\pi\tau^2) + \frac{1}{2\tau^2} \sum_{j=1}^d \theta_j^2 \right) \\ &= \arg \min_{\theta} \left(-\log p(y | \theta, \mathbf{x}) + \frac{1}{2\tau^2} \|\theta\|_2^2 \right)\end{aligned}$$

- We see how the inverse variance (precision) $1/\tau^2$ controls shrinkage



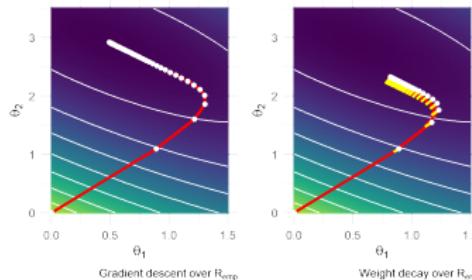
EXAMPLE: BAYESIAN L2 REGULARIZATION

- DGP $y = \theta + \varepsilon$ where $\varepsilon \sim \mathcal{N}(0, 1)$ and $\theta = 1$;
with Gaussian prior on θ , so $\mathcal{N}(0, \tau^2)$ for $\tau \in \{0.25, 0.5, 2\}$
- For $n = 20$, posterior of θ and MAP can be calculated analytically
- Plotting the L_2 regularized empirical risk $\mathcal{R}_{\text{reg}}(\theta) = \sum_{i=1}^n (y_i - \theta)^2 + \lambda\theta^2$
with $\lambda = 1/\tau^2$ shows that ridge solution is identical with MAP
- In our simulation, the empirical mean is $\bar{y} = 0.94$, with shrinkage toward 0 induced in the MAP



Introduction to Machine Learning

Regularization Weight Decay and L2



Learning goals

- $L2$ regularization with GD is equivalent to weight decay
- Understand how weight decay changes the optimization trajectory



WEIGHT DECAY VS. L2 REGULARIZATION

Let's optimize $L2$ -regularized risk of a model $f(\mathbf{x} \mid \theta)$

$$\min_{\theta} \mathcal{R}_{\text{reg}}(\theta) = \min_{\theta} \mathcal{R}_{\text{emp}}(\theta) + \frac{\lambda}{2} \|\theta\|_2^2$$

by GD. The gradient is

$$\nabla_{\theta} \mathcal{R}_{\text{reg}}(\theta) = \nabla_{\theta} \mathcal{R}_{\text{emp}}(\theta) + \lambda \theta$$

We iteratively update θ by step size α times the negative gradient

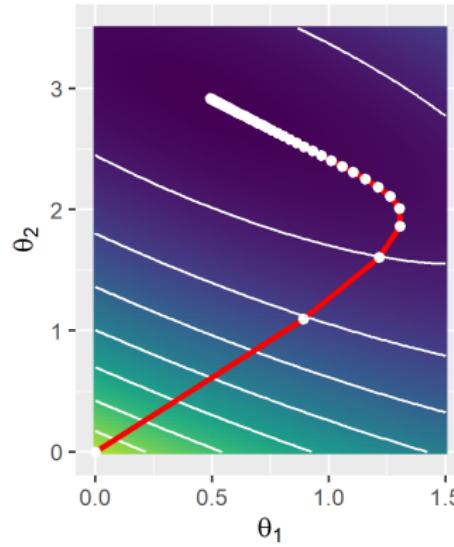
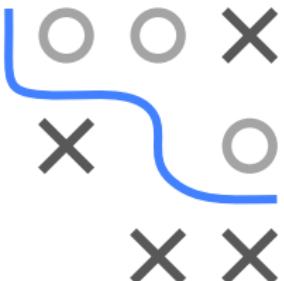
$$\begin{aligned}\theta^{[\text{new}]} &= \theta^{[\text{old}]} - \alpha \left(\nabla_{\theta} \mathcal{R}_{\text{emp}}(\theta^{[\text{old}]}) + \lambda \theta^{[\text{old}]} \right) \\ &= \theta^{[\text{old}]} (1 - \alpha \lambda) - \alpha \nabla_{\theta} \mathcal{R}_{\text{emp}}(\theta^{[\text{old}]})\end{aligned}$$

We see how $\theta^{[\text{old}]}$ decays in magnitude – for small α and λ – before we do the gradient step. Performing the decay directly, under this name, is a very well-known technique in DL - and simply $L2$ regularization in disguise (for GD).

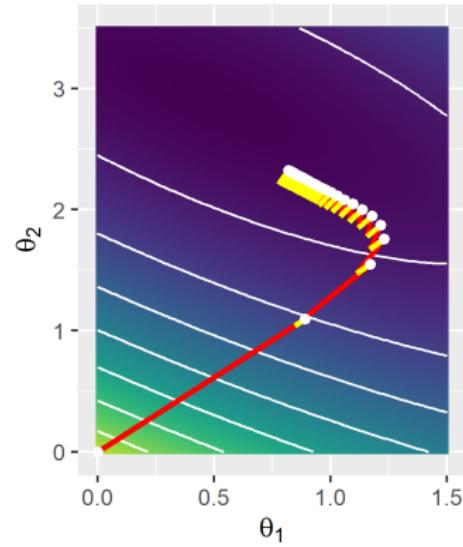


WEIGHT DECAY VS. L2 REGULARIZATION / 2

In GD With WD, we slide down neg. gradients of \mathcal{R}_{emp} ,
but in every step, we are pulled back to origin.



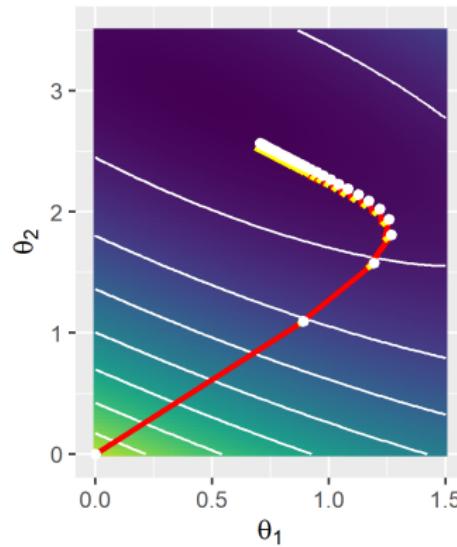
Without WD



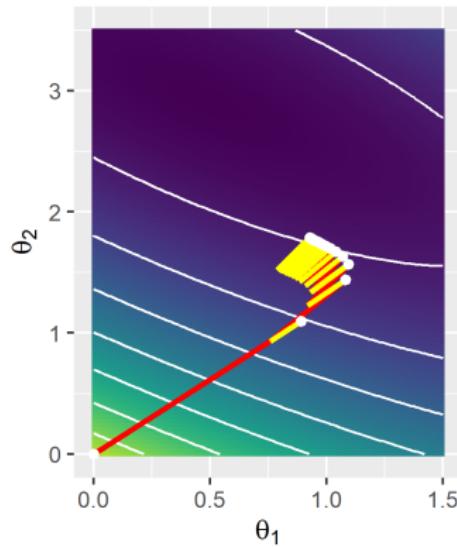
With GD

WEIGHT DECAY VS. L2 REGULARIZATION / 3

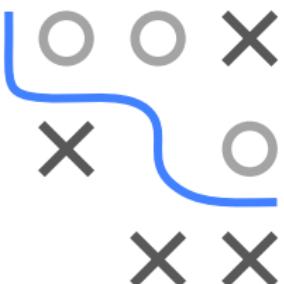
How strongly we are pulled back (for fixed α) depends on λ :



Small λ



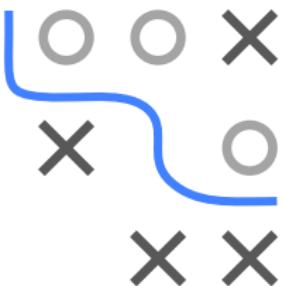
Large λ



CAVEAT AND OTHER OPTIMIZERS

Caveat: Equivalence of weight decay and $L2$ only holds for (S)GD!

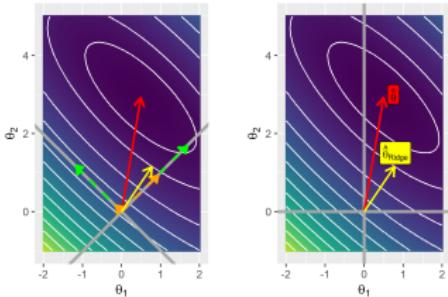
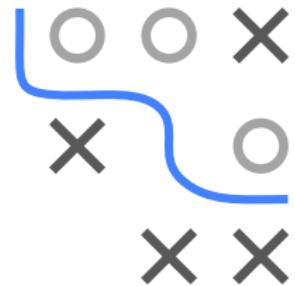
- ▶ Hanson and Pratt 1988 originally define WD “decoupled” from gradient-updates $\alpha \nabla_{\theta} \mathcal{R}_{\text{emp}}(\theta^{[\text{old}]})$ as
$$\theta^{[\text{new}]} = \theta^{[\text{old}]}(1 - \lambda') - \alpha \nabla_{\theta} \mathcal{R}_{\text{emp}}(\theta^{[\text{old}]})$$
- This is equivalent to modern WD/ $L2$ (last slide) using reparameterization $\lambda' = \alpha \lambda$
- Consequence: if there is optimal λ' , then optimal $L2$ penalty is tightly coupled to α as $\lambda = \lambda'/\alpha$ (and vice versa)
- ▶ Loshchilov and Hutter 2019 show no equivalence of $L2$ and WD possible for adaptive methods like Adam (Prop. 2)
- In many cases where SGD+ $L2$ works well, Adam+ $L2$ underperforms due to non-equivalence with WD
- They propose a variant of Adam decoupling WD from gradient updates (AdamW), increasing performance over Adam+ $L2$



Introduction to Machine Learning

Regularization

Geometry of L2 Regularization



Learning goals

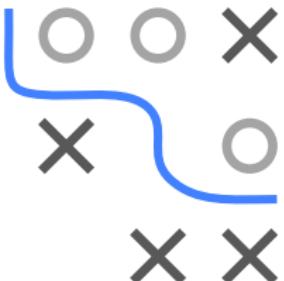
- Approximate transformation of unregularized minimizer to regularized
- Principal components of Hessian influence where parameters are decayed

GEOMETRIC ANALYSIS OF L_2 REGULARIZATION

Quadratic Taylor approx of the unregularized objective $\mathcal{R}_{\text{emp}}(\theta)$ around its minimizer $\hat{\theta}$:

$$\tilde{\mathcal{R}}_{\text{emp}}(\theta) = \mathcal{R}_{\text{emp}}(\hat{\theta}) + \nabla_{\theta} \mathcal{R}_{\text{emp}}(\hat{\theta}) \cdot (\theta - \hat{\theta}) + \frac{1}{2}(\theta - \hat{\theta})^T \mathbf{H}(\theta - \hat{\theta})$$

where \mathbf{H} is the Hessian of $\mathcal{R}_{\text{emp}}(\theta)$ at $\hat{\theta}$



We notice:

- First-order term is 0, because gradient must be 0 at minimizer
- \mathbf{H} is positive semidefinite, because we are at the minimizer

$$\tilde{\mathcal{R}}_{\text{emp}}(\theta) = \mathcal{R}_{\text{emp}}(\hat{\theta}) + \frac{1}{2}(\theta - \hat{\theta})^T \mathbf{H}(\theta - \hat{\theta})$$

GEOMETRIC ANALYSIS OF L2 REGULARIZATION

/ 2

The minimum of $\tilde{\mathcal{R}}_{\text{emp}}(\theta)$ occurs where $\nabla_{\theta}\tilde{\mathcal{R}}_{\text{emp}}(\theta) = \mathbf{H}(\theta - \hat{\theta})$ is 0.
Now we L2-regularize $\tilde{\mathcal{R}}_{\text{emp}}(\theta)$, such that

$$\tilde{\mathcal{R}}_{\text{reg}}(\theta) = \tilde{\mathcal{R}}_{\text{emp}}(\theta) + \frac{\lambda}{2} \|\theta\|_2^2$$

and solve this approximation of \mathcal{R}_{reg} for the minimizer $\hat{\theta}_{\text{ridge}}$:

$$\nabla_{\theta}\tilde{\mathcal{R}}_{\text{reg}}(\theta) = 0$$

$$\lambda\theta + \mathbf{H}(\theta - \hat{\theta}) = 0$$

$$(\mathbf{H} + \lambda\mathbf{I})\theta = \mathbf{H}\hat{\theta}$$

$$\hat{\theta}_{\text{ridge}} = (\mathbf{H} + \lambda\mathbf{I})^{-1} \mathbf{H}\hat{\theta}$$

We see: minimizer of L2-regularized version is (approximately!) transformation of minimizer of the unpenalized version.

Doesn't matter whether the model is an LM – or something else!



GEOMETRIC ANALYSIS OF L2 REGULARIZATION

/ 3

- As λ approaches 0, the regularized solution $\hat{\theta}_{\text{ridge}}$ approaches $\hat{\theta}$. What happens as λ grows?
- Because H is a real symmetric matrix, it can be decomposed as $H = Q\Sigma Q^T$, where Σ is a diagonal matrix of eigenvalues and Q is an orthonormal basis of eigenvectors.
- Rewriting the transformation formula with this:

$$\begin{aligned}\hat{\theta}_{\text{ridge}} &= (Q\Sigma Q^T + \lambda I)^{-1} Q\Sigma Q^T \hat{\theta} \\ &= [Q(\Sigma + \lambda I)Q^T]^{-1} Q\Sigma Q^T \hat{\theta} \\ &= Q(\Sigma + \lambda I)^{-1} \Sigma Q^T \hat{\theta}\end{aligned}$$

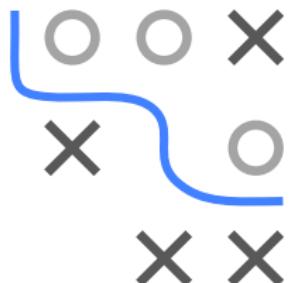
- So: We rescale $\hat{\theta}$ along axes defined by eigenvectors of H . The component of $\hat{\theta}$ that is associated with the j -th eigenvector of H is rescaled by factor of $\frac{\sigma_j}{\sigma_j + \lambda}$, where σ_j is eigenvalue.



GEOMETRIC ANALYSIS OF L_2 REGULARIZATION

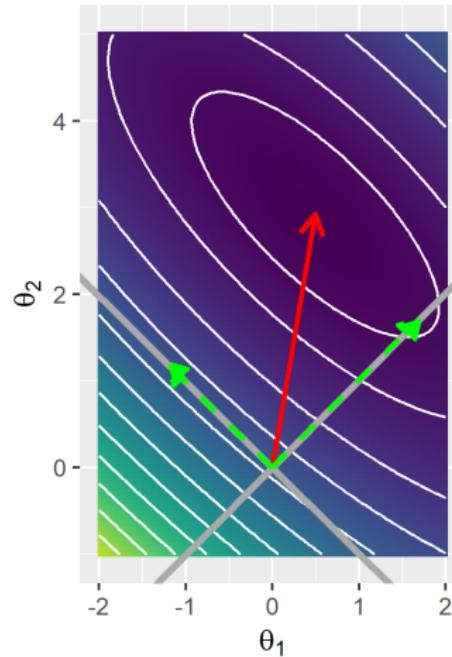
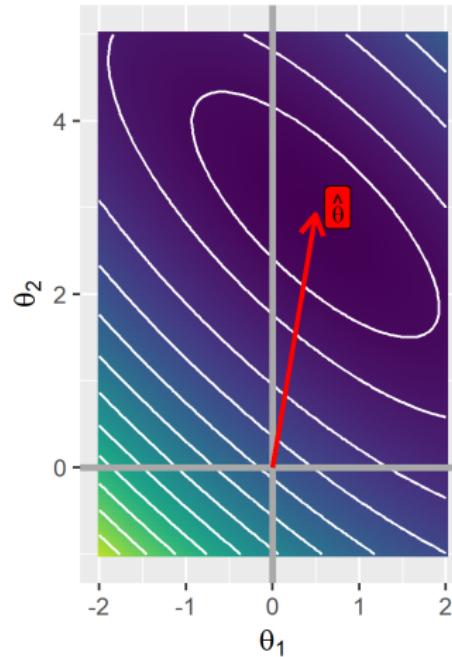
/ 4

First, $\hat{\theta}$ is rotated by \mathbf{Q}^\top , which we can interpret as projection of $\hat{\theta}$ on rotated coord system defined by principal directions of \mathbf{H} :



GEOMETRIC ANALYSIS OF L^2 REGULARIZATION

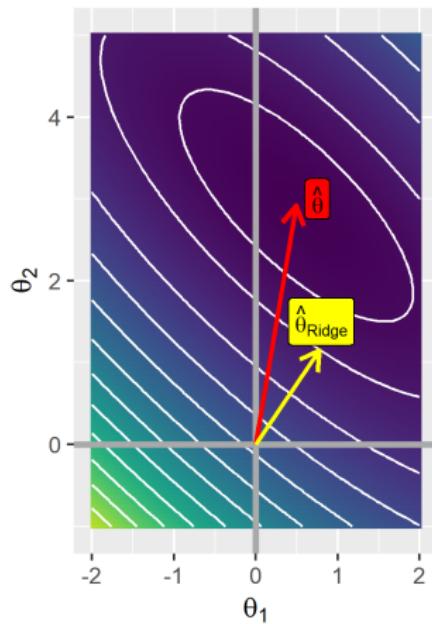
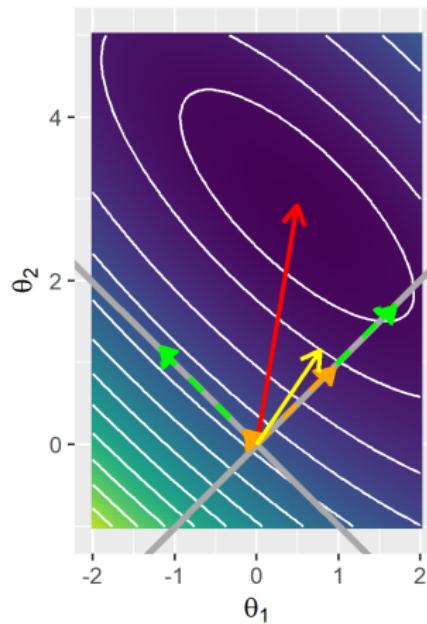
/ 5



GEOMETRIC ANALYSIS OF L2 REGULARIZATION

/ 6

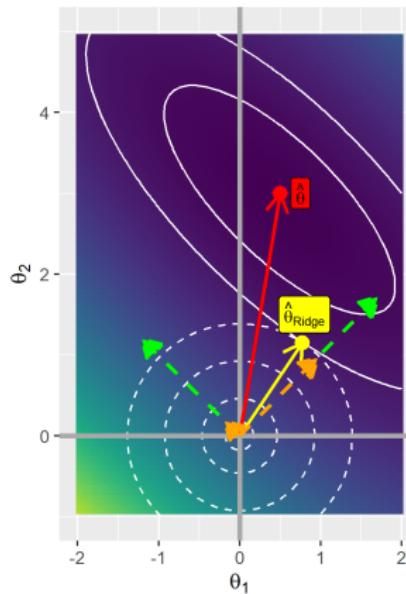
j -th (new) axis is rescaled by $\frac{\sigma_j}{\sigma_j + \lambda}$ before we rotate back.



GEOMETRIC ANALYSIS OF L2 REGULARIZATION

/ 7

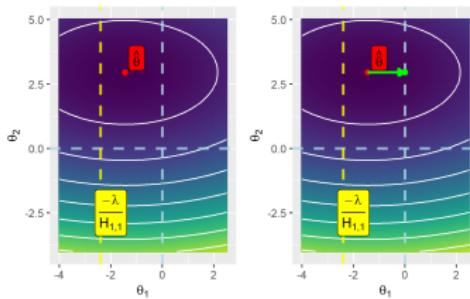
- Decay: $\frac{\sigma_j}{\sigma_j + \lambda}$
- Along directions where eigenvalues of H are relatively large, e.g., $\sigma_j \gg \lambda$, effect of regularization is small.
- Components / directions with $\sigma_j \ll \lambda$ are strongly shrunken.
- So: Directions along which parameters contribute strongly to objective are preserved relatively intact.
- In other directions, small eigenvalue of Hessian means that moving in this direction will not decrease objective much. For such unimportant directions, corresponding components of θ are decayed away.



Introduction to Machine Learning

Regularization

Geometry of L1 Regularization



Learning goals

- Approximate transformation of unregularized minimizer to regularized
- Soft-Thresholding

L1-REGULARIZATION

- The L1-regularized risk of a model $f(\mathbf{x} \mid \boldsymbol{\theta})$ is

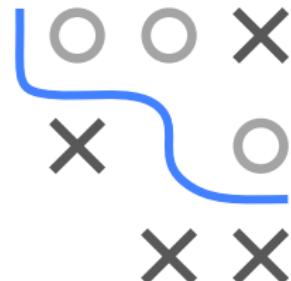
$$\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) = \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + \sum_j \lambda |\theta_j|$$

and the (sub-)gradient is:

$$\nabla_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + \lambda \text{sign}(\boldsymbol{\theta})$$

- Unlike in L2, contribution to grad. doesn't scale with θ_j elements.
- Again: quadratic Taylor approximation of $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$ around its minimizer $\hat{\boldsymbol{\theta}}$, then regularize:

$$\tilde{\mathcal{R}}_{\text{reg}}(\boldsymbol{\theta}) = \mathcal{R}_{\text{emp}}(\hat{\boldsymbol{\theta}}) + \frac{1}{2} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{H} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) + \sum_j \lambda |\theta_j|$$



L1-REGULARIZATION / 2

- To cheat and simplify, we assume the H is diagonal, with $H_{j,j} \geq 0$
- Now $\tilde{\mathcal{R}}_{\text{reg}}(\theta)$ decomposes into sum over params θ_j (separable!):

$$\tilde{\mathcal{R}}_{\text{reg}}(\theta) = \mathcal{R}_{\text{emp}}(\hat{\theta}) + \sum_j \left[\frac{1}{2} H_{j,j} (\theta_j - \hat{\theta}_j)^2 \right] + \sum_j \lambda |\theta_j|$$

- We can minimize analytically:

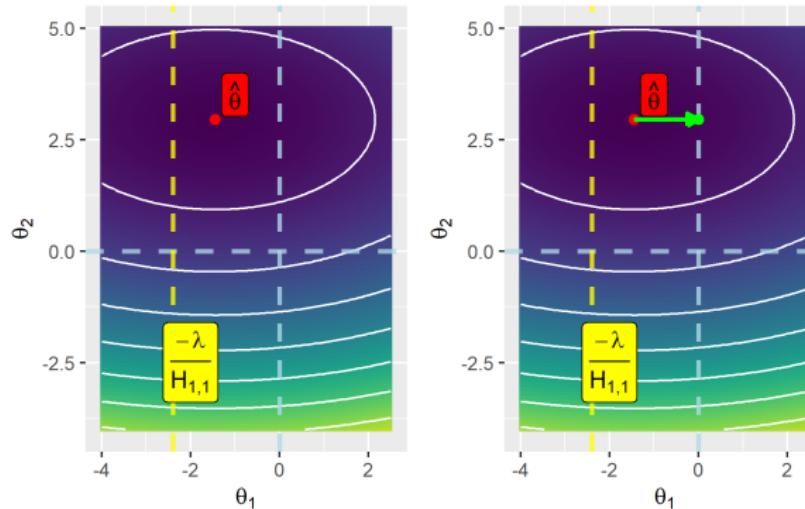
$$\begin{aligned}\hat{\theta}_{\text{lasso},j} &= \text{sign}(\hat{\theta}_j) \max \left\{ |\hat{\theta}_j| - \frac{\lambda}{H_{j,j}}, 0 \right\} \\ &= \begin{cases} \hat{\theta}_j + \frac{\lambda}{H_{j,j}} & , \text{if } \hat{\theta}_j < -\frac{\lambda}{H_{j,j}} \\ 0 & , \text{if } \hat{\theta}_j \in [-\frac{\lambda}{H_{j,j}}, \frac{\lambda}{H_{j,j}}] \\ \hat{\theta}_j - \frac{\lambda}{H_{j,j}} & , \text{if } \hat{\theta}_j > \frac{\lambda}{H_{j,j}} \end{cases}\end{aligned}$$

- Shows how lasso (approx) transforms the normal minimizer
- If $H_{j,j} = 0$ exactly, $\hat{\theta}_{\text{lasso},j} = 0$



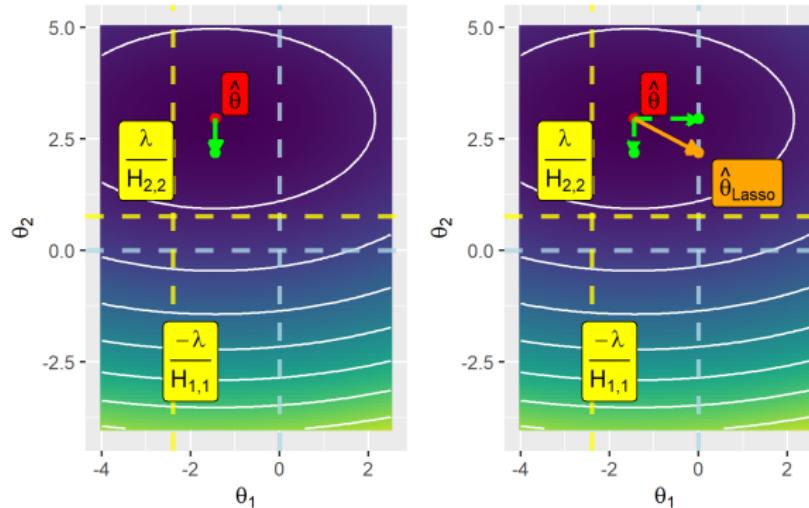
L1-REGULARIZATION / 3

- If $0 < \hat{\theta}_j \leq \frac{\lambda}{H_{j,j}}$ or $0 > \hat{\theta}_j \geq -\frac{\lambda}{H_{j,j}}$, the optimal value of θ_j (for the regularized risk) is 0 because the contribution of $\mathcal{R}_{\text{emp}}(\theta)$ to $\mathcal{R}_{\text{reg}}(\theta)$ is overwhelmed by the L1 penalty, which forces it to be 0.



L1-REGULARIZATION / 4

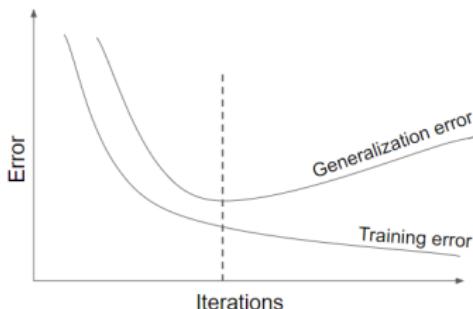
- If $0 < \frac{\lambda}{H_{j,j}} < \hat{\theta}_j$ or $0 > -\frac{\lambda}{H_{j,j}} > \hat{\theta}_j$, the L1 penalty shifts the optimal value of θ_j toward 0 by the amount $\frac{\lambda}{H_{j,j}}$.



- Yellow dotted lines are limits from soft-thresholding
- Therefore, the L1 penalty induces sparsity in the parameter vector.

Introduction to Machine Learning

Regularization Early Stopping



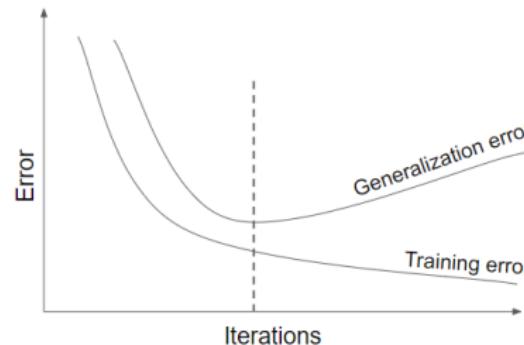
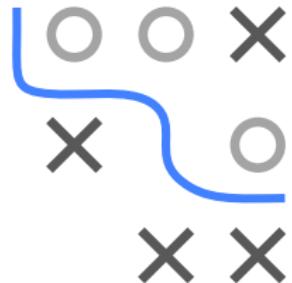
Learning goals

- Know how early stopping works
- Understand how early stopping acts as a regularizer



EARLY STOPPING

- Especially for complex nonlinear models we can easily overfit
- In optimization: Often, after a certain number of iterations, generalization error begins to increase even though training error continues to decrease



EARLY STOPPING / 2

For iterative optimizers like SGD,
we can monitor this step-by-step over small iterations:

- ➊ Split train data $\mathcal{D}_{\text{train}}$ into $\mathcal{D}_{\text{subtrain}}$ and \mathcal{D}_{val} (e.g. with ratio of 2:1)
- ➋ Train on $\mathcal{D}_{\text{subtrain}}$ and eval model on \mathcal{D}_{val}
- ➌ Stop when validation error stops decreasing
(after a range of “patience” steps)
- ➍ Use parameters of the previous step for the actual model



More sophisticated forms also apply cross-validation.

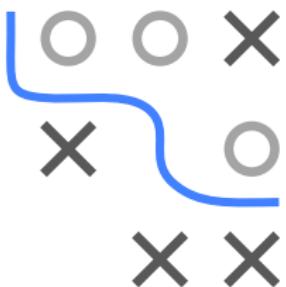
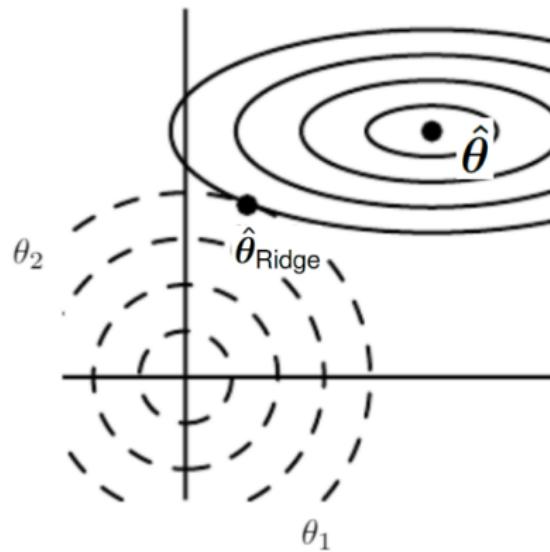
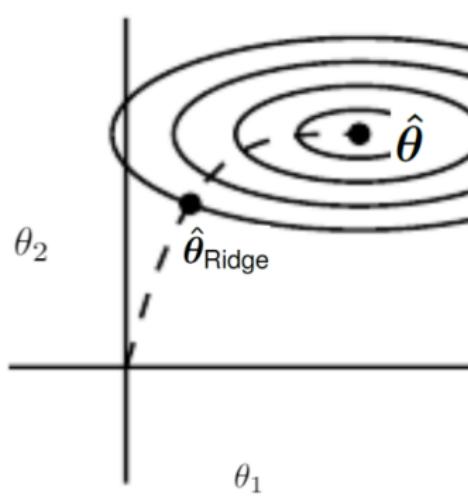
Strengths	Weaknesses
Effective and simple	Periodical evaluation of validation error
Applicable to almost any model without adjustment	Temporary copy of θ (we have to save the whole model each time validation error improves)
Combinable with other regularization methods	Less data for training \rightarrow include \mathcal{D}_{val} afterwards



- For simple case of LM with squared loss and GD optim initialized at $\theta = 0$: Early stopping has exact correspondence with L_2 regularization/WD: optimal early-stopping iter T_{stop} inversely proportional to λ scaled by step-size α

$$T_{\text{stop}} \approx \frac{1}{\alpha \lambda} \Leftrightarrow \lambda \approx \frac{1}{T_{\text{stop}} \alpha}$$

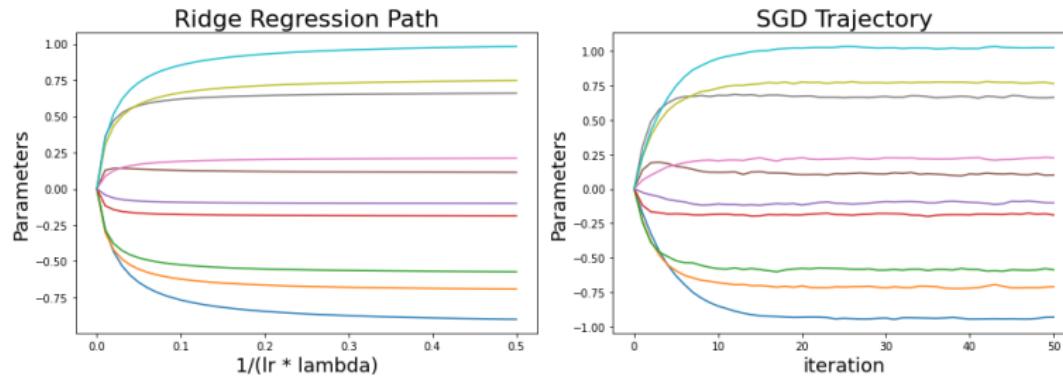
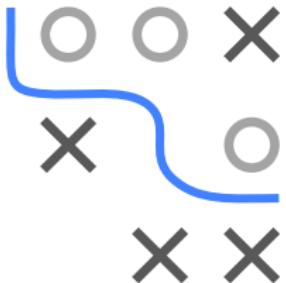
- Small λ (regu. \downarrow) \Rightarrow large T_{stop} (complexity \uparrow) and vice versa



Goodfellow et al. (2016)

- Solid lines are $\mathcal{R}_{\text{emp}}(\theta)$
- LHS: Trajectory of GD early stopped, initialized at origin
- RHS: Constrained form of ridge regularization

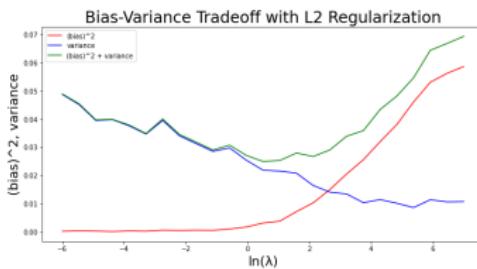
Solution paths for $L2$ regularized linear model closely matches SGD trajectory of unregularized LM initialized at $\theta = 0$



Caveat: Initialization at the origin is crucial for this equivalence to hold, which is almost never exactly used in practice in ML/DL applications

Introduction to Machine Learning

Regularization Perspectives on Ridge Regression (Deep-Dive)



Learning goals

- Interpretation of L_2 regularization as row-augmentation
- Interpretation of L_2 regularization as minimizing risk under feature noise

PERSPECTIVES ON L2 REGULARIZATION

We already saw two interpretations of L2 regularization.

- We know that it is equivalent to a constrained optimization problem:

$$\hat{\theta}_{\text{ridge}} = \arg \min_{\theta} \sum_{i=1}^n \left(y^{(i)} - \theta^T \mathbf{x}^{(i)} \right)^2 + \lambda \|\theta\|_2^2 = (\mathbf{x}^T \mathbf{x} + \lambda I)^{-1} \mathbf{x}^T \mathbf{y}$$



For some t depending on λ this is equivalent to:

$$\hat{\theta}_{\text{ridge}} = \arg \min_{\theta} \sum_{i=1}^n \left(y^{(i)} - \theta^T \mathbf{x}^{(i)} \right)^2 \text{ s.t. } \|\theta\|_2^2 \leq t$$

- Bayesian interpretation of ridge regression: For additive Gaussian errors $\mathcal{N}(0, \sigma^2)$ and i.i.d. normal priors $\theta_j \sim \mathcal{N}(0, \tau^2)$, the resulting MAP estimate is $\hat{\theta}_{\text{ridge}}$ with $\lambda = \frac{\sigma^2}{\tau^2}$:

$$\hat{\theta}_{\text{MAP}} = \arg \max_{\theta} \log[p(\mathbf{y} | \mathbf{X}, \theta) p(\theta)] = \arg \min_{\theta} \sum_{i=1}^n \left(y^{(i)} - \theta^T \mathbf{x}^{(i)} \right)^2 + \frac{\sigma^2}{\tau^2} \|\theta\|_2^2$$

L2 AND ROW-AUGMENTATION

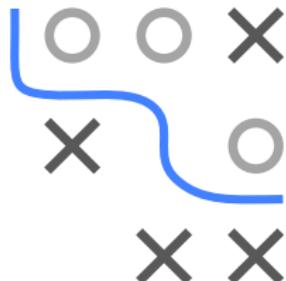
We can also recover the ridge estimator by performing least-squares on a **row-augmented** data set: Let $\tilde{\mathbf{X}} := \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda} \mathbf{I}_p \end{pmatrix}$ and $\tilde{\mathbf{y}} := \begin{pmatrix} \mathbf{y} \\ \mathbf{0}_p \end{pmatrix}$.

With the augmented data, the unreg. least-squares solution $\tilde{\boldsymbol{\theta}}$ is:

$$\begin{aligned}\tilde{\boldsymbol{\theta}} &= \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^{n+p} \left(\tilde{y}^{(i)} - \boldsymbol{\theta}^T \tilde{\mathbf{x}}^{(i)} \right)^2 \\ &= \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^n \left(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)} \right)^2 + \sum_{j=1}^p \left(0 - \sqrt{\lambda} \theta_j \right)^2 \\ &= \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^n \left(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)} \right)^2 + \lambda \|\boldsymbol{\theta}\|_2^2\end{aligned}$$

$\implies \hat{\boldsymbol{\theta}}_{\text{ridge}}$ is the least-squares solution $\tilde{\boldsymbol{\theta}}$ but using $\tilde{\mathbf{X}}, \tilde{\mathbf{y}}$ instead of \mathbf{X}, \mathbf{y} !

This is a sometimes useful “recasting” or “rewriting” for ridge.



L2 AND NOISY FEATURES

Now consider perturbed features $\tilde{\mathbf{x}}^{(i)} := \mathbf{x}^{(i)} + \boldsymbol{\delta}^{(i)}$ where $\boldsymbol{\delta}^{(i)} \stackrel{iid}{\sim} (\mathbf{0}, \lambda \mathbf{I}_p)$.

We assume no specific distribution. Now minimize risk with L2 loss, we define it slightly different than usual, as here our data $\mathbf{x}^{(i)}, y^{(i)}$ are fixed, but we integrate over the random permutations $\boldsymbol{\delta}$:



$$\mathcal{R}(\boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{\delta}} \left[\sum_{i=1}^n (y^{(i)} - \boldsymbol{\theta}^\top \tilde{\mathbf{x}}^{(i)})^2 \right] = \mathbb{E}_{\boldsymbol{\delta}} \left[\sum_{i=1}^n (y^{(i)} - \boldsymbol{\theta}^\top (\mathbf{x}^{(i)} + \boldsymbol{\delta}^{(i)}))^2 \right] \mid \text{expand}$$

$$\mathcal{R}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\delta}} \left[\sum_{i=1}^n ((y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)})^2 - 2\boldsymbol{\theta}^\top \boldsymbol{\delta}^{(i)} (y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)}) + \boldsymbol{\theta}^\top \boldsymbol{\delta}^{(i)} \boldsymbol{\delta}^{(i)\top} \boldsymbol{\theta}) \right]$$

By linearity of expectation, $\mathbb{E}_{\boldsymbol{\delta}}[\boldsymbol{\delta}^{(i)}] = \mathbf{0}_p$ and $\mathbb{E}_{\boldsymbol{\delta}}[\boldsymbol{\delta}^{(i)} \boldsymbol{\delta}^{(i)\top}] = \lambda \mathbf{I}_p$, this is

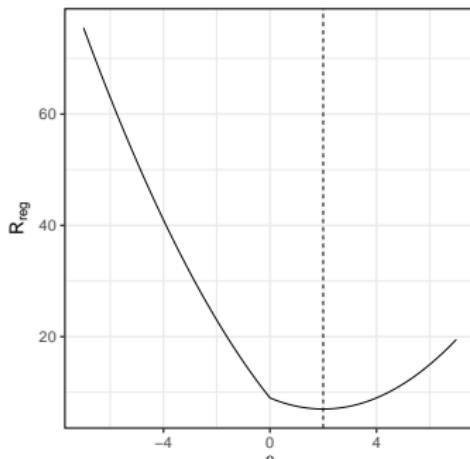
$$\begin{aligned} \mathcal{R}(\boldsymbol{\theta}) &= \sum_{i=1}^n ((y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)})^2 - 2\boldsymbol{\theta}^\top \mathbb{E}_{\boldsymbol{\delta}}[\boldsymbol{\delta}^{(i)}] (y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)}) + \boldsymbol{\theta}^\top \mathbb{E}_{\boldsymbol{\delta}}[\boldsymbol{\delta}^{(i)} \boldsymbol{\delta}^{(i)\top}] \boldsymbol{\theta}) \\ &= \sum_{i=1}^n (y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)})^2 + \lambda \|\boldsymbol{\theta}\|_2^2 \end{aligned}$$

⇒ Ridge regression on unperturbed features $\mathbf{x}^{(i)}$ turns out to be the same as minimizing squared loss averaged over feature noise distribution!

Introduction to Machine Learning

Regularization

Soft-thresholding and lasso (Deep-Dive)



Learning goals

- Understand the relationship between soft-thresholding and L1 regularization



SOFT-THRESHOLDING AND L1 REGULARIZATION

In the lecture, we wanted to solve

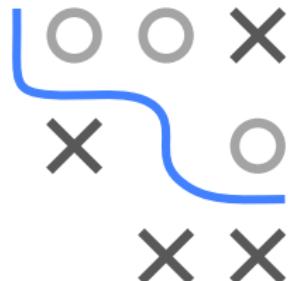
$$\min_{\theta} \tilde{\mathcal{R}}_{\text{reg}}(\theta) = \min_{\theta} \mathcal{R}_{\text{emp}}(\hat{\theta}) + \sum_j \left[\frac{1}{2} H_{j,j} (\theta_j - \hat{\theta}_j)^2 \right] + \sum_j \lambda |\theta_j|$$

with $H_{j,j} \geq 0$, $\lambda > 0$. Note that we can separate the dimensions, i.e.,

$$\tilde{\mathcal{R}}_{\text{reg}}(\theta) = \sum_j z_j(\theta_j) \text{ with } z_j(\theta_j) = \frac{1}{2} H_{j,j} (\theta_j - \hat{\theta}_j)^2 + \lambda |\theta_j|.$$

Hence, we can minimize each z_j separately to find the global minimum.

If $H_{j,j} = 0$, then z_j is clearly minimized by $\hat{\theta}_{\text{lasso},j} = 0$. Otherwise, z_j is strictly convex since $\frac{1}{2} H_{j,j} (\theta_j - \hat{\theta}_j)^2$ is strictly convex and the sum of a strictly convex function and a convex function is strictly convex.



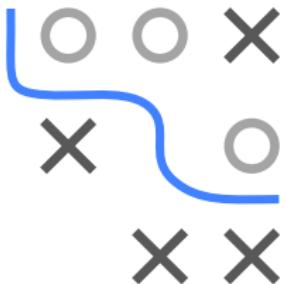
SOFT-THRESHOLDING AND L1 REGULARIZATION

For strictly convex functions, there exists only one unique minimum and for convex functions a stationary point (if it exists) is a minimum.

We now separately investigate z_j for $\theta_j > 0$ and $\theta_j < 0$.

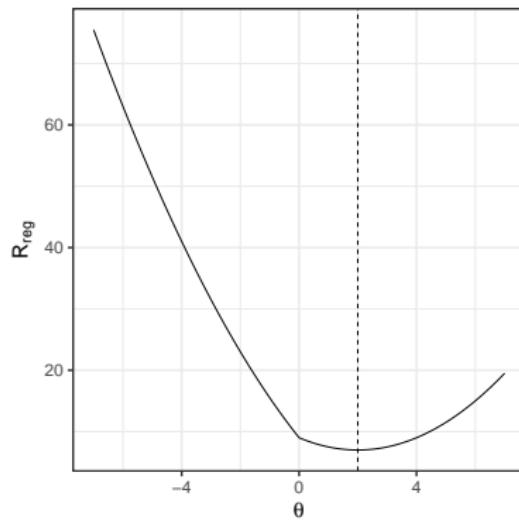
NB: on these halflines z_j is differentiable (with possible stationary point) since

- for $\theta_j > 0$: $\frac{d}{d\theta_j} |\theta_j| = \frac{d}{d\theta_j} \theta_j = 1$,
- for $\theta_j < 0$: $\frac{d}{d\theta_j} |\theta_j| = \frac{d}{d\theta_j} (-\theta_j) = -1$.



SOFT-THRESHOLDING AND L1 REGULARIZATION

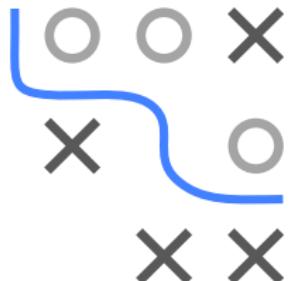
1) $\theta_j > 0$:



$$\begin{aligned}\frac{d}{d\theta_j} z_j(\theta_j) &= H_{j,j}\theta_j - H_{j,j}\hat{\theta}_j + \lambda \stackrel{!}{=} 0 \\ \Rightarrow \hat{\theta}_{\text{lasso},j} &= \hat{\theta}_j - \frac{\lambda}{H_{j,j}} > 0\end{aligned}$$

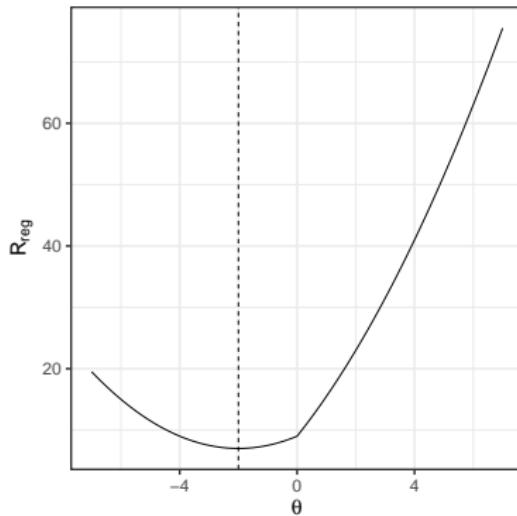
This minimum is only valid if $\hat{\theta}_{\text{lasso},j} > 0$ and so it must hold that

$$\hat{\theta}_j > \frac{\lambda}{H_{j,j}}.$$



SOFT-THRESHOLDING AND L1 REGULARIZATION

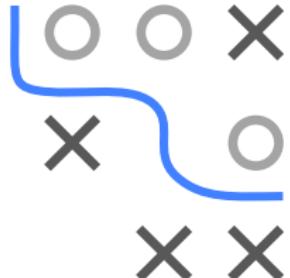
2) $\hat{\theta}_{\text{lasso},j} < 0$:



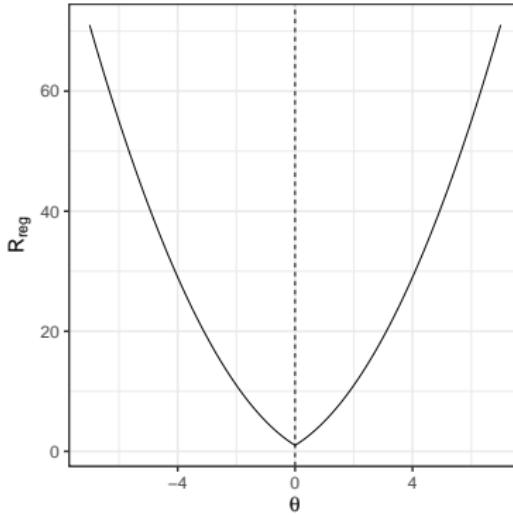
$$\begin{aligned}\frac{d}{d\theta_j} z_j(\theta_j) &= H_{j,j}\theta_j - H_{j,j}\hat{\theta}_j - \lambda \stackrel{!}{=} 0 \\ \Rightarrow \hat{\theta}_{\text{lasso},j} &= \hat{\theta}_j + \frac{\lambda}{H_{j,j}} < 0\end{aligned}$$

This minimum is only valid if $\hat{\theta}_{\text{lasso},j} < 0$ and so it must hold that

$$\hat{\theta}_j < -\frac{\lambda}{H_{j,j}}.$$



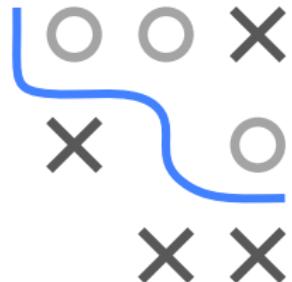
SOFT-THRESHOLDING AND L1 REGULARIZATION



\Rightarrow If $\hat{\theta}_j \in [-\frac{\lambda}{H_{j,j}}, \frac{\lambda}{H_{j,j}}]$ then z_j has no stationary point with

$$\hat{\theta}_{\text{lasso},j} < 0 \text{ or } \hat{\theta}_{\text{lasso},j} > 0.$$

However, a unique minimum must exist since z_j is strictly convex for $H_{j,j} > 0$. This means the only possible minimizer of z_j is $\hat{\theta}_{\text{lasso},j} = 0$.



$$\Rightarrow \hat{\theta}_{\text{lasso},j} = \begin{cases} \hat{\theta}_j + \frac{\lambda}{H_{j,j}} & , \text{if } \hat{\theta}_j < -\frac{\lambda}{H_{j,j}} \text{ and } H_{j,j} > 0 \\ 0 & , \text{if } \hat{\theta}_j \in [-\frac{\lambda}{H_{j,j}}, \frac{\lambda}{H_{j,j}}] \text{ or } H_{j,j} = 0 \\ \hat{\theta}_j - \frac{\lambda}{H_{j,j}} & , \text{if } \hat{\theta}_j > \frac{\lambda}{H_{j,j}} \text{ and } H_{j,j} > 0 \end{cases}$$

INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

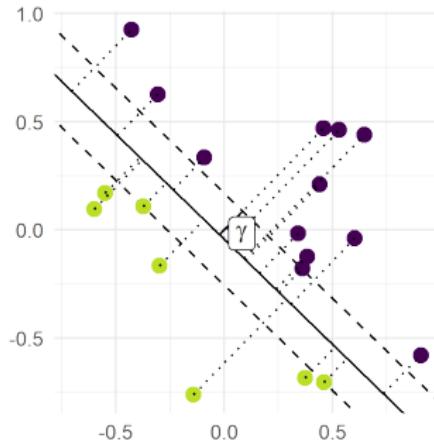
Boosting

Gaussian Processes



Introduction to Machine Learning

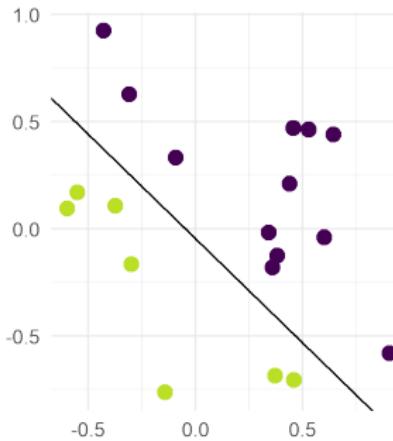
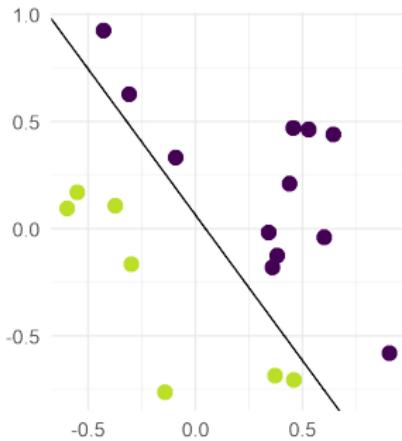
Linear Hard Margin SVM



Learning goals

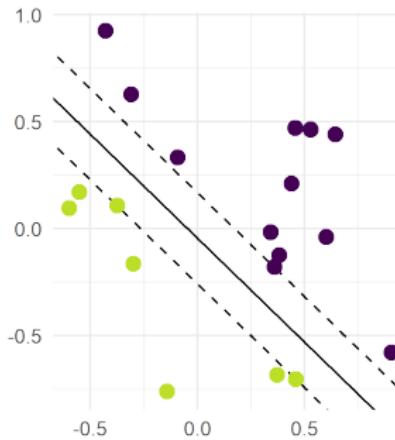
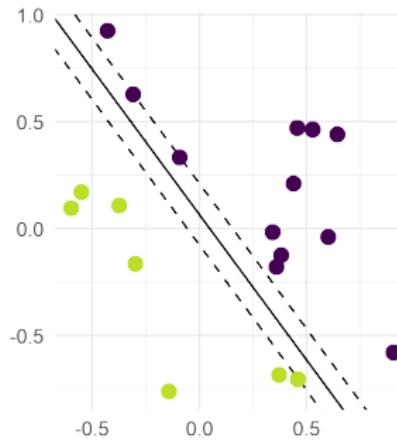
- Know that the hard-margin SVM maximizes the margin between data points and hyperplane
- Know that this is a quadratic program
- Know that support vectors are the data points closest to the separating hyperplane

LINEAR CLASSIFIERS



- We want study how to build a binary, linear classifier from solid geometrical principles.
- Which of these two classifiers is “better”?

LINEAR CLASSIFIERS



- We want study how to build a binary, linear classifier from solid geometrical principles.
- Which of these two classifiers is “better”?
 - The decision boundary on the right has a larger **safety margin**.

SUPPORT VECTOR MACHINES: GEOMETRY

For labeled data $\mathcal{D} = ((\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}))$, with $y^{(i)} \in \{-1, +1\}$:

- Assume linear separation by $f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} + \theta_0$, such that all $+1$ -observations are in the positive halfspace

$$\{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) > 0\}$$



and all -1 -observations are in the negative halfspace

$$\{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) < 0\}.$$

- For a linear separating hyperplane, we have

$$y^{(i)} \underbrace{\left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \theta_0 \right)}_{=f(\mathbf{x}^{(i)})} > 0 \quad \forall i \in \{1, 2, \dots, n\}.$$

SUPPORT VECTOR MACHINES: GEOMETRY

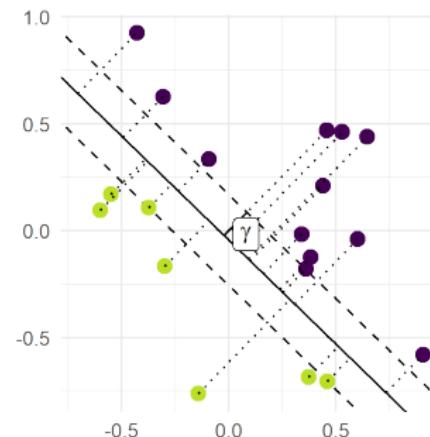
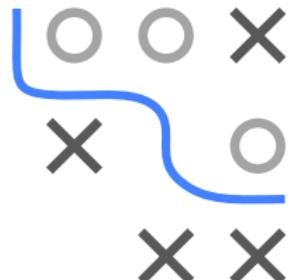
-

$$d(f, \mathbf{x}^{(i)}) = \frac{y^{(i)} f(\mathbf{x}^{(i)})}{\|\theta\|} = y^{(i)} \frac{\theta^T \mathbf{x}^{(i)} + \theta_0}{\|\theta\|}$$

computes the (signed) distance to the separating hyperplane

$f(\mathbf{x}) = 0$, positive for correct classifications, negative for incorrect.

- This expression becomes negative for misclassified points.

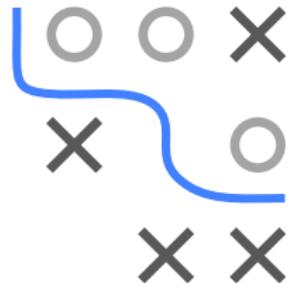
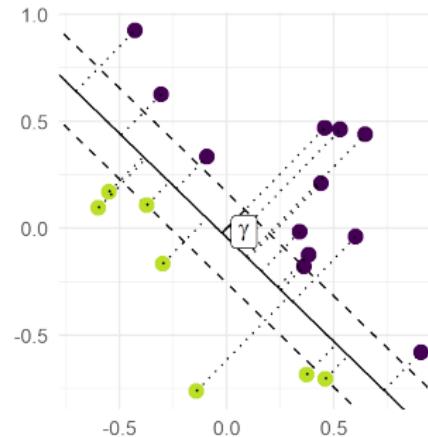


SUPPORT VECTOR MACHINES: GEOMETRY

- The distance of f to the whole dataset \mathcal{D} is the smallest distance

$$\gamma = \min_i \left\{ d(f, \mathbf{x}^{(i)}) \right\}.$$

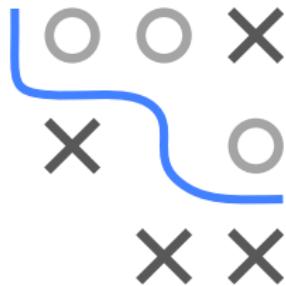
- This represents the “safety margin”, it is positive if f separates and we want to maximize it.



MAXIMUM MARGIN SEPARATION

We formulate the desired property of a large “safety margin” as an optimization problem:

$$\begin{aligned} \max_{\theta, \theta_0} \quad & \gamma \\ \text{s.t.} \quad & d(f, \mathbf{x}^{(i)}) \geq \gamma \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$



- The constraints mean: We require that any instance i should have a “safety” distance of at least γ from the decision boundary defined by $f(= \theta^T \mathbf{x} + \theta_0) = 0$.
- Our objective is to maximize the “safety” distance.

MAXIMUM MARGIN SEPARATION

We reformulate the problem:

$$\begin{aligned} \max_{\theta, \theta_0} \quad & \gamma \\ \text{s.t.} \quad & \frac{y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0)}{\|\theta\|} \geq \gamma \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$



- The inequality is rearranged by multiplying both sides with $\|\theta\|$:

$$\begin{aligned} \max_{\theta, \theta_0} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq \|\theta\| \gamma \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$

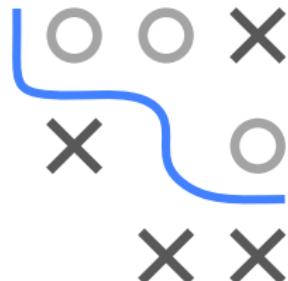
MAXIMUM MARGIN SEPARATION

- Note that the same hyperplane does not have a unique representation:

$$\{\mathbf{x} \in \mathcal{X} \mid \boldsymbol{\theta}^\top \mathbf{x} = 0\} = \{\mathbf{x} \in \mathcal{X} \mid c \cdot \boldsymbol{\theta}^\top \mathbf{x} = 0\}$$

for arbitrary $c \neq 0$.

- To ensure uniqueness of the solution, we make a reference choice
 - we only consider hyperplanes with $\|\boldsymbol{\theta}\| = 1/\gamma$:



$$\max_{\boldsymbol{\theta}, \theta_0} \gamma$$

$$\text{s.t. } y^{(i)} (\langle \boldsymbol{\theta}, \mathbf{x}^{(i)} \rangle + \theta_0) \geq 1 \quad \forall i \in \{1, \dots, n\}.$$

MAXIMUM MARGIN SEPARATION

- Substituting $\gamma = 1/\|\theta\|$ in the objective yields:

$$\max_{\theta, \theta_0} \frac{1}{\|\theta\|}$$

$$\text{s.t. } y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq 1 \quad \forall i \in \{1, \dots, n\}.$$



- Maximizing $1/\|\theta\|$ is the same as minimizing $\|\theta\|$, which is the same as minimizing $\frac{1}{2}\|\theta\|^2$:

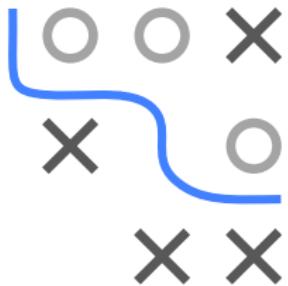
$$\min_{\theta, \theta_0} \frac{1}{2}\|\theta\|^2$$

$$\text{s.t. } y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq 1 \quad \forall i \in \{1, \dots, n\}.$$

QUADRATIC PROGRAM

We derived the following optimization problem:

$$\begin{aligned} \min_{\theta, \theta_0} \quad & \frac{1}{2} \|\theta\|^2 \\ \text{s.t.} \quad & y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq 1 \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$



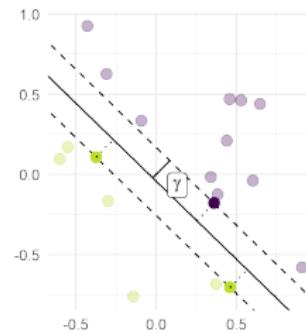
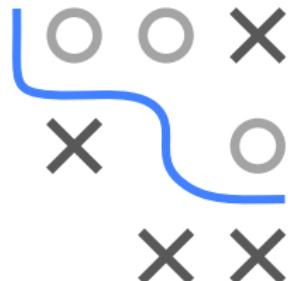
This turns out to be a **convex optimization problem** – particularly, a **quadratic program**: The objective function is quadratic, and the constraints are linear inequalities.

This is called the **primal** problem. We will later show that we can also derive a dual problem from it.

We will call this the **linear hard-margin SVM**.

SUPPORT VECTORS

- There exist instances $(\mathbf{x}^{(i)}, y^{(i)})$ with minimal margin $y^{(i)} f(\mathbf{x}^{(i)}) = 1$, fulfilling the inequality constraints with equality.
- They are called **support vectors (SVs)**. They are located exactly at a distance of $\gamma = 1/\|\theta\|$ from the separating hyperplane.
- It is already geometrically obvious that the solution does not depend on the non-SVs! We could delete them from the data and would arrive at the same solution.



Introduction to Machine Learning

Hard-Margin SVM Dual



Learning goals

- Know how to derive the SVM dual problem

HARD MARGIN SVM DUAL

We have derived the primal quadratic program for the hard margin SVM. We could directly solve this, but traditionally the SVM is solved in the dual and this has some advantages. In any case, many algorithms and derivations are based on it, so we need to know it.

$$\begin{aligned} \min_{\theta, \theta_0} \quad & \frac{1}{2} \|\theta\|^2 \\ \text{s.t.} \quad & y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq 1 \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$

The Lagrange function of the SVM optimization problem is

$$\begin{aligned} L(\theta, \theta_0, \alpha) = \quad & \frac{1}{2} \|\theta\|^2 - \sum_{i=1}^n \alpha_i [y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) - 1] \\ \text{s.t.} \quad & \alpha_i \geq 0 \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$

The **dual** form of this problem is

$$\max_{\alpha} \min_{\theta, \theta_0} L(\theta, \theta_0, \alpha).$$



HARD MARGIN SVM DUAL

Notice how the $(p+1)$ decision variables (θ, θ_0) have become n decisions variables α , as constraints turned into variables and vice versa. Now every data point has an associated non-negative weight.

$$L(\theta, \theta_0, \alpha) = \frac{1}{2} \|\theta\|^2 - \sum_{i=1}^n \alpha_i \left[y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) - 1 \right]$$

s.t. $\alpha_i \geq 0 \quad \forall i \in \{1, \dots, n\}.$

We find the stationary point of $L(\theta, \theta_0, \alpha)$ w.r.t. θ, θ_0 and obtain

$$\begin{aligned}\theta &= \sum_{i=1}^n \alpha_i y^{(i)} \mathbf{x}^{(i)}, \\ 0 &= \sum_{i=1}^n \alpha_i y^{(i)} \quad \forall i \in \{1, \dots, n\}.\end{aligned}$$



HARD MARGIN SVM DUAL

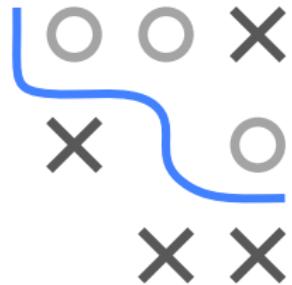
By inserting these expressions & simplifying we obtain the dual problem

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle \\ \text{s.t.} \quad & \sum_{i=1}^n \alpha_i y^{(i)} = 0, \\ & \alpha_i \geq 0 \quad \forall i \in \{1, \dots, n\}, \end{aligned}$$

or, equivalently, in matrix notation:

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & \mathbf{1}^T \alpha - \frac{1}{2} \alpha^T \text{diag}(\mathbf{y}) \mathbf{K} \text{diag}(\mathbf{y}) \alpha \\ \text{s.t.} \quad & \alpha^T \mathbf{y} = 0, \\ & \alpha \geq 0, \end{aligned}$$

with $\mathbf{K} := \mathbf{X} \mathbf{X}^T$.



HARD MARGIN SVM DUAL

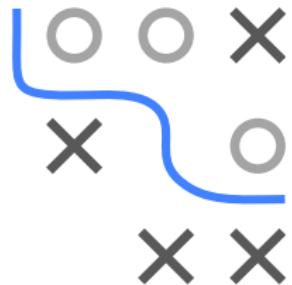
If $(\theta, \theta_0, \alpha)$ fulfills the KKT conditions (stationarity, primal/dual feasibility, complementary slackness), it solves both the primal and dual problem (strong duality).

Under these conditions, and if we solve the dual problem and obtain $\hat{\alpha}$, we know that θ is a linear combination of our data points:

$$\hat{\theta} = \sum_{i=1}^n \hat{\alpha}_i y^{(i)} \mathbf{x}^{(i)}$$

Complementary slackness means:

$$\hat{\alpha}_i \left[y^{(i)} \left(\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0 \right) - 1 \right] = 0 \quad \forall i \in \{1, \dots, n\}.$$



HARD MARGIN SVM DUAL



$$\hat{\theta} = \sum_{i=1}^n \hat{\alpha}_i y^{(i)} \mathbf{x}^{(i)}$$

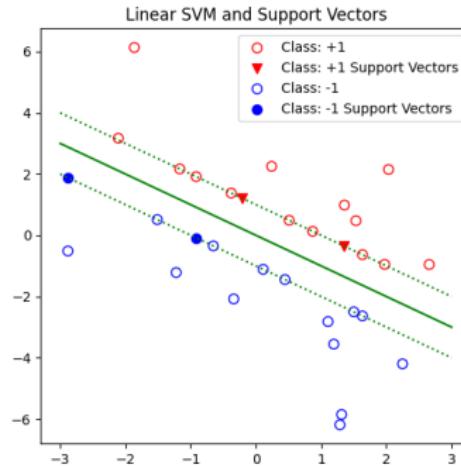
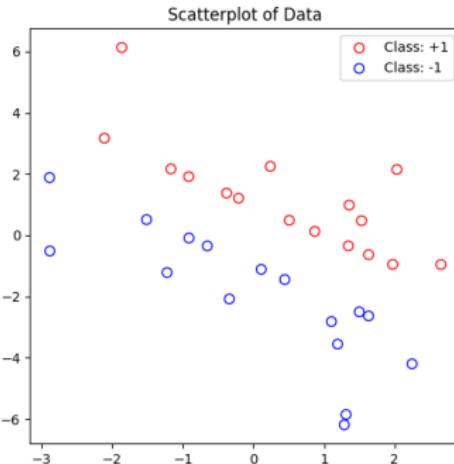
$$\hat{\alpha}_i \left[y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) - 1 \right] = 0 \quad \forall i \in \{1, \dots, n\}.$$

- So either $\hat{\alpha}_i = 0$, and is not active in the linear combination, or $\hat{\alpha}_i > 0$, then $y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) = 1$, and $(\mathbf{x}^{(i)}, y^{(i)})$ has minimal margin and is a support vector!
- We see that we can directly extract the support vectors from the dual variables and the θ solution only depends on them.
- We can reconstruct the bias term θ_0 from any support vector:

$$\theta_0 = y^{(i)} - \langle \theta, \mathbf{x}^{(i)} \rangle.$$

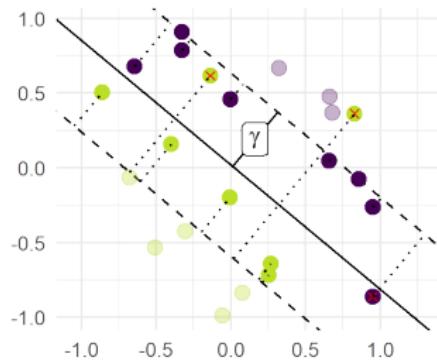
DUAL VARIABLE AND SUPPORT VECTORS

- SVs are defined to be points with $\hat{\alpha}_i > 0$. In the case of hard margin linear SVM, the SVs are on the edge of margin.
- However, not all points on edge of margin are necessarily SVs.
- In other words, it is possible that both $\hat{\alpha}_i = 0$ and $y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle) - 1 = 0$ hold.



Introduction to Machine Learning

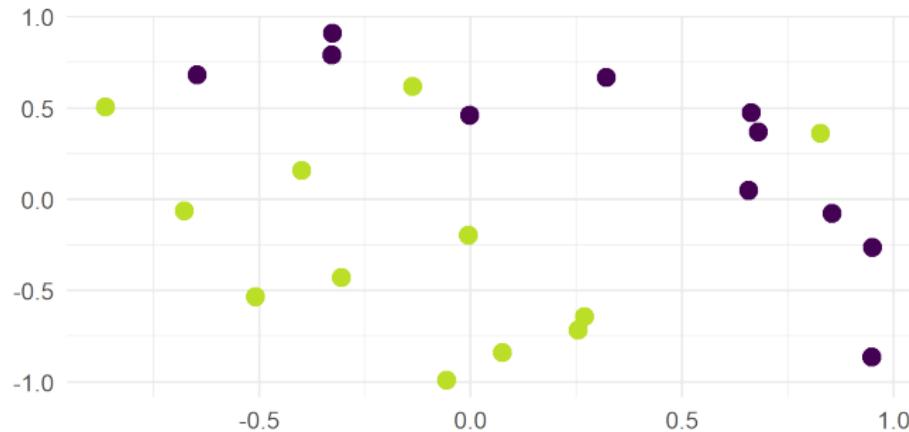
Soft-Margin SVM



Learning goals

- Understand that the hard-margin SVM problem is only solvable for linearly separable data
- Know that the soft-margin SVM problem therefore allows margin violations
- The degree to which margin violations are tolerated is controlled by a hyperparameter

NON-SEPARABLE DATA



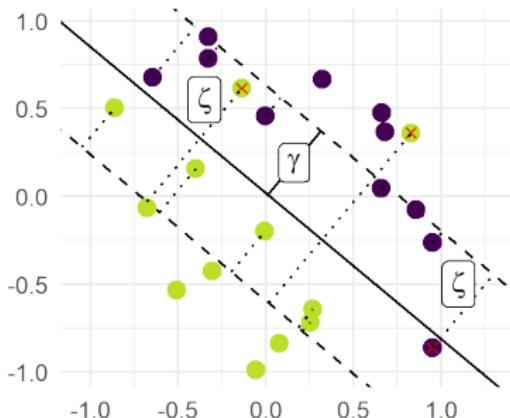
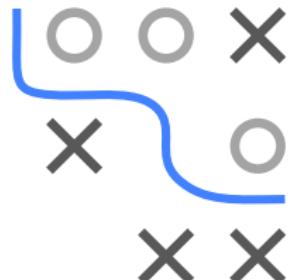
- Assume that dataset \mathcal{D} is not linearly separable.
- Margin maximization becomes meaningless because the hard-margin SVM optimization problem has contradictory constraints and thus an empty **feasible region**.

MARGIN VIOLATIONS

- We still want a large margin for most of the examples.
- We allow violations of the margin constraints via slack vars $\zeta^{(i)} \geq 0$

$$y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq 1 - \zeta^{(i)}$$

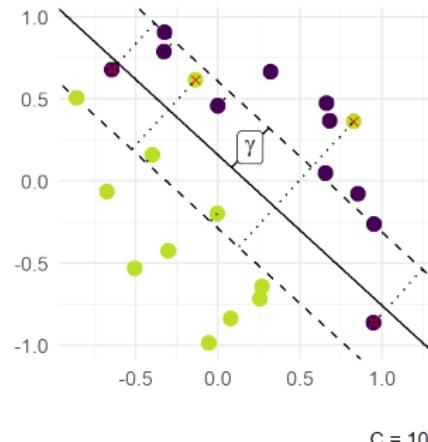
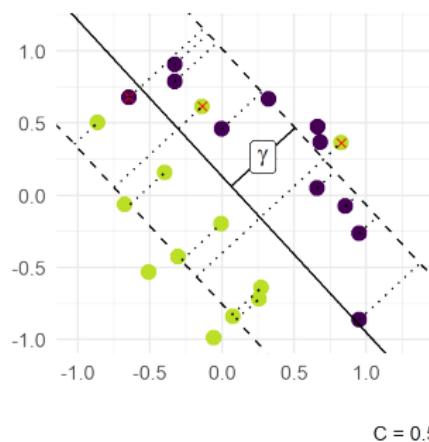
- Even for separable data, a decision boundary with a few violations and a large average margin may be preferable to one without any violations and a small average margin.



We assume $\gamma = 1$ to not further complicate presentation.

MARGIN VIOLATIONS

- Now we have two distinct and contradictory goals:
 - Maximize the margin.
 - Minimize margin violations.
- Let's minimize a weighted sum of them: $\frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n \zeta^{(i)}$
- Constant $C > 0$ controls the relative importance of the two parts.



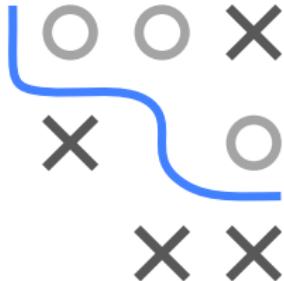
SOFT-MARGIN SVM

The linear **soft-margin** SVM is the convex quadratic program:

$$\min_{\theta, \theta_0, \zeta^{(i)}} \frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n \zeta^{(i)}$$

$$\text{s.t. } y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq 1 - \zeta^{(i)} \quad \forall i \in \{1, \dots, n\},$$

$$\text{and } \zeta^{(i)} \geq 0 \quad \forall i \in \{1, \dots, n\}.$$

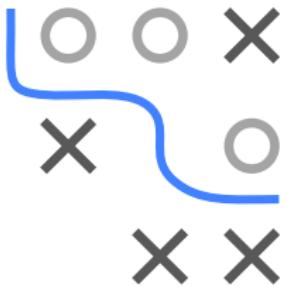


This is called “soft-margin” SVM because the “hard” margin constraint is replaced with a “softened” constraint that can be violated by an amount $\zeta^{(i)}$.

LAGRANGE FUNCTION AND KKT

The Lagrange function of the soft-margin SVM is given by:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}, \theta_0, \zeta, \alpha, \mu) = & \frac{1}{2} \|\boldsymbol{\theta}\|_2^2 + C \sum_{i=1}^n \zeta^{(i)} - \sum_{i=1}^n \alpha_i \left(y^{(i)} (\langle \boldsymbol{\theta}, \mathbf{x}^{(i)} \rangle + \theta_0) - 1 + \zeta^{(i)} \right) \\ & - \sum_{i=1}^n \mu_i \zeta^{(i)} \quad \text{with Lagrange multipliers } \alpha \text{ and } \mu.\end{aligned}$$



The KKT conditions for $i = 1, \dots, n$ are:

$$\begin{aligned}\alpha_i \geq 0, & \quad \mu_i \geq 0, \\ y^{(i)} (\langle \boldsymbol{\theta}, \mathbf{x}^{(i)} \rangle + \theta_0) - 1 + \zeta^{(i)} \geq 0, & \quad \zeta^{(i)} \geq 0, \\ \alpha_i (y^{(i)} (\langle \boldsymbol{\theta}, \mathbf{x}^{(i)} \rangle + \theta_0) - 1 + \zeta^{(i)}) = 0, & \quad \zeta^{(i)} \mu_i = 0.\end{aligned}$$

With these, we derive (see our optimization course) that

$$\boldsymbol{\theta} = \sum_{i=1}^n \alpha_i y^{(i)} \mathbf{x}^{(i)}, \quad 0 = \sum_{i=1}^n \alpha_i y^{(i)}, \quad \alpha_i = C - \mu_i \quad \forall i = 1, \dots, n.$$

SOFT-MARGIN SVM DUAL FORM

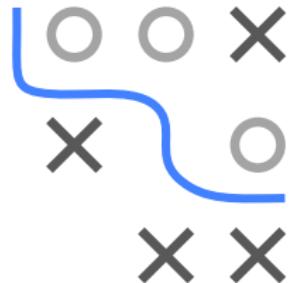
Can be derived exactly as for the hard margin case.

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \\ & \sum_{i=1}^n \alpha_i y^{(i)} = 0, \end{aligned}$$

or, in matrix notation:

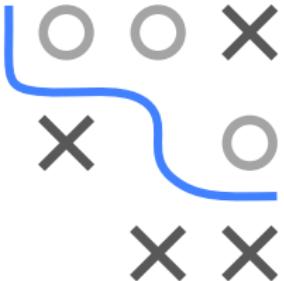
$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & \mathbf{1}^T \alpha - \frac{1}{2} \alpha^T \text{diag}(\mathbf{y}) \mathbf{K} \text{diag}(\mathbf{y}) \alpha \\ \text{s.t.} \quad & \alpha^T \mathbf{y} = 0, \\ & 0 \leq \alpha \leq C, \end{aligned}$$

with $\mathbf{K} := \mathbf{X} \mathbf{X}^T$.



COST PARAMETER C

- The parameter C controls the trade-off between the two conflicting objectives of maximizing the size of the margin and minimizing the frequency and size of margin violations.
- It is known under different names, such as “trade-off parameter”, “regularization parameter”, and “complexity control parameter”.
- For sufficiently large C margin violations become extremely costly, and the optimal solution does not violate any margins if the data is separable. The hard-margin SVM is obtained as a special case.

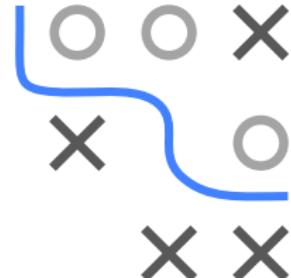
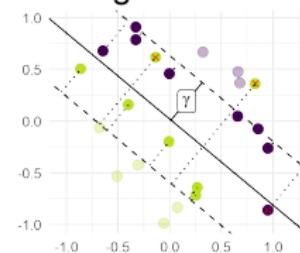


SUPPORT VECTORS

There are three types of training examples:

- Non-SVs have $\alpha_i = 0 \Rightarrow \mu_i = C \Rightarrow \zeta^{(i)} = 0$ and can be removed from the problem without changing the solution. Their margin $yf(\mathbf{x}) \geq 1$. They are always classified correctly and are never inside of the margin.
- SVs with $0 < \alpha_i < C \Rightarrow \mu_i > 0 \Rightarrow \zeta^{(i)} = 0$ are located exactly on the margin and have $yf(\mathbf{x}) = 1$.
- SVs with $\alpha_i = C$ have an associated slack $\zeta^{(i)} \geq 0$. They can be on the margin or can be margin violators with $yf(\mathbf{x}) < 1$ (they can even be misclassified if $\zeta^{(i)} \geq 1$).

As for hard-margin case: on the margin we can have SVs and non-SVs.

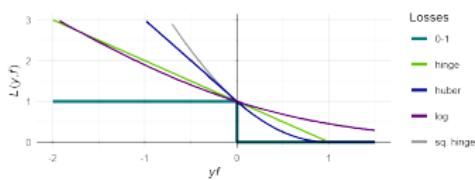


Introduction to Machine Learning

SVMs and Empirical Risk Minimization



Learning goals



- Know why the SVM problem can be understood as (regularized) empirical risk minimization problem
- Know that the corresponding loss is the hinge loss

REGULARIZED EMPIRICAL RISK MINIMIZATION

- We motivated SVMs from a geometrical point of view: The margin is a distance to be maximized.
- This is not really true anymore under margin violations: The slack variables are not really distances. Instead, $\gamma \cdot \zeta^{(i)}$ is the distance by which an observation violates the margin.
- This already indicates that transferring the geometric intuition from hard-margin SVMs to the soft-margin case has its limits.
- There is an alternative approach to understanding soft-margin SVMs: They are **regularized empirical risk minimizers**.



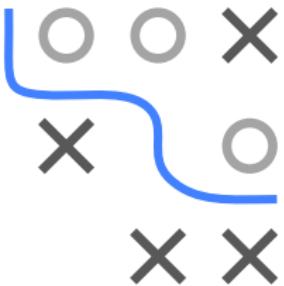
SOFT-MARGIN SVM WITH ERM AND HINGE LOSS

We derived this QP for the soft-margin SVM:

$$\min_{\theta, \theta_0, \zeta^{(i)}} \frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n \zeta^{(i)}$$

$$\text{s.t. } y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq 1 - \zeta^{(i)} \quad \forall i \in \{1, \dots, n\},$$

$$\text{and } \zeta^{(i)} \geq 0 \quad \forall i \in \{1, \dots, n\}.$$



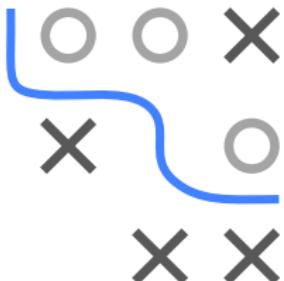
In the optimum, the inequalities will hold with equality (as we minimize the slacks), so $\zeta^{(i)} = 1 - y^{(i)} f(\mathbf{x}^{(i)})$, but the lowest value $\zeta^{(i)}$ can take is 0 (we do not get a bonus for points beyond the margin on the correct side). So we can rewrite the above:

$$\frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)})); \quad L(y, f) = \begin{cases} 1 - yf & \text{if } yf \leq 1 \\ 0 & \text{if } yf > 1 \end{cases}$$

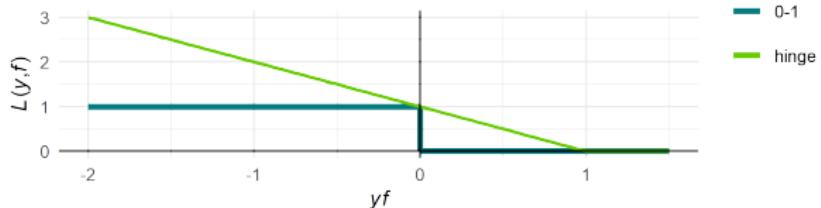
We can also write $L(y, f) = \max(1 - yf, 0)$.

SOFT-MARGIN SVM WITH ERM AND HINGE LOSS

$$\mathcal{R}_{\text{emp}}(\theta) = \frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)})) ; \quad L(y, f) = \max(1 - yf, 0)$$



- This now obviously L2-regularized empirical risk minimization.
- Actually, a lot of ERM theory was established when Vapnik (co-)invented the SVM in the beginning of the 90s.
- L is called hinge loss – as it looks like a door hinge.
- It is a continuous, convex, upper bound on the zero-one loss. In a certain sense it is the best upper convex relaxation of the 0-1.



SOFT-MARGIN SVM WITH ERM AND HINGE LOSS

$$\frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right); L(y, f) = \max(1 - yf, 0)$$

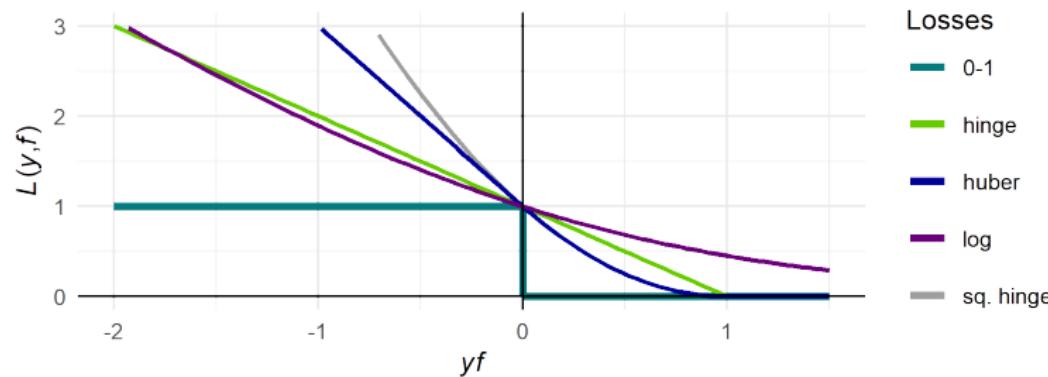
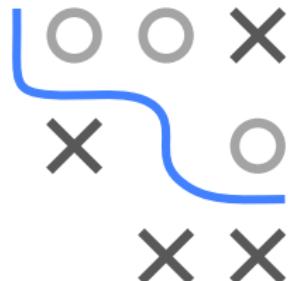


- The ERM interpretation does not require any of the terms – the loss or the regularizer – to be geometrically meaningful.
- The above form is a very compact form to define the convex optimization problem of the SVM.
- It is "well-behaved" due to convexity, every minimum is global.
- The above is convex, without constraints! We might see this as "easier to optimize" than the QP from before. But note it is non-differentiable due to the hinge. So specialized techniques (e.g. sub-gradient) would have to be used.
- Some literature claims this primal cannot be easily kernelized - which is not really true.

OTHER LOSSES

SVMs can easily be generalized by changing the loss function.

- Squared hinge loss / Least Squares SVM:
$$L(y, f) = \max(0, (1 - yf)^2)$$
- Huber loss (smoothed hinge loss)
- Bernoulli/Log loss. This is L2-regularized logistic regression!
- NB: These other losses usually do not generate sparse solutions in terms of data weights and hence have no "support vectors".

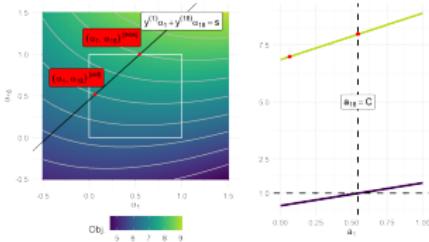


Introduction to Machine Learning

Support Vector Machine Training



Learning goals



- Know that the SVM problem is not differentiable
- Know how to optimize the SVM problem in the primal via subgradient descent
- Know how to optimize SVM in the dual formulation via pairwise coordinate ascent

SUPPORT VECTOR MACHINE TRAINING

- Until now, we have ignored the issue of solving the various convex optimization problems.
- The first question is whether we should solve the **primal** or the **dual problem**.
- In the literature SVMs are usually trained in the dual.
- However, SVMs can be trained both in the primal and the dual – each approach has its advantages and disadvantages.
- It is not easy to create an efficient SVM solver, and often specialized approaches have been developed, we only cover basic ideas here.



TRAINING SVM IN THE PRIMAL

Unconstrained formulation of soft-margin SVM:

$$\min_{\theta, \theta_0} \frac{\lambda}{2} \|\theta\|^2 + \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)} | \theta))$$

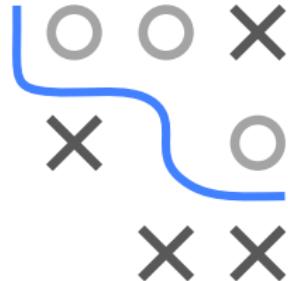
where $L(y, f) = \max(0, 1 - yf)$ and $f(\mathbf{x} | \theta) = \theta^T \mathbf{x} + \theta_0$.

(We inconsequentially changed the regularization constant.)

We cannot directly use GD, as the above is not differentiable.

Solutions:

- ① Use smoothed loss (squared hinge, huber), then do GD.
NB: Will not create a sparse SVM if we do not add extra tricks.
- ② Use **subgradient** methods.
- ③ Do stochastic subgradient descent.
Pegasos: Primal Estimated sub-GrAdient SOlver for SVM.



PEGASOS: SSGD IN THE PRIMAL

Approximate the risk by a stochastic 1-sample version:

$$\frac{\lambda}{2} \|\theta\|^2 + L(y^{(i)}, f(\mathbf{x}^{(i)} | \theta))$$

With: $f(\mathbf{x} | \theta) = \theta^T \mathbf{x} + \theta_0$ and $L(y, f) = \max(0, 1 - yf)$

The subgradient for θ is $\lambda\theta - y^{(i)}\mathbf{x}^{(i)}\mathbb{I}_{yf < 1}$



Stochastic subgradient descent (without intercept θ_0)

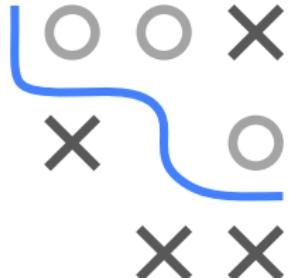
- 1: **for** $t = 1, 2, \dots$ **do**
 - 2: Pick step size α
 - 3: Randomly pick an index i
 - 4: If $y^{(i)}f(\mathbf{x}^{(i)}) < 1$ set $\theta^{[t+1]} = (1 - \lambda\alpha)\theta^{[t]} + \alpha y^{(i)}\mathbf{x}^{(i)}$
 - 5: If $y^{(i)}f(\mathbf{x}^{(i)}) \geq 1$ set $\theta^{[t+1]} = (1 - \lambda\alpha)\theta^{[t]}$
 - 6: **end for**
-

Note the weight decay due to the L2-regularization.

TRAINING SVM IN THE DUAL

The dual problem of the soft-margin SVM is

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \quad \sum_{i=1}^n \alpha_i y^{(i)} = 0 \end{aligned}$$



We could solve this problem using coordinate ascent. That means we optimize w.r.t. α_1 , for example, while holding $\alpha_2, \dots, \alpha_n$ fixed.

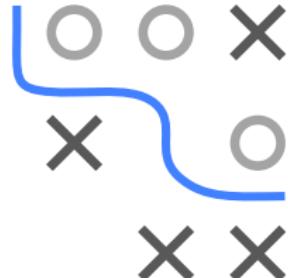
But: We cannot make any progress since α_1 is determined by
 $\sum_{i=1}^n \alpha_i y^{(i)} = 0$!

TRAINING SVM IN THE DUAL

Solution: Update two variables simultaneously

$$\max_{\alpha} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \quad \sum_{i=1}^n \alpha_i y^{(i)} = 0$$



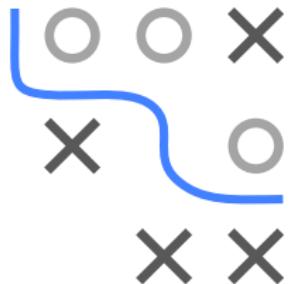
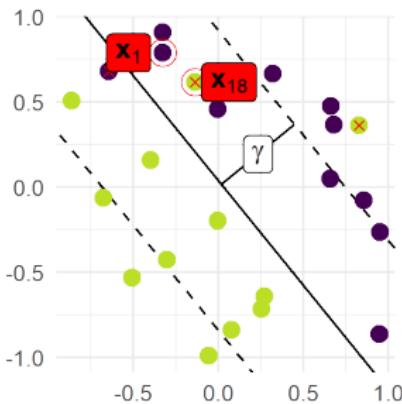
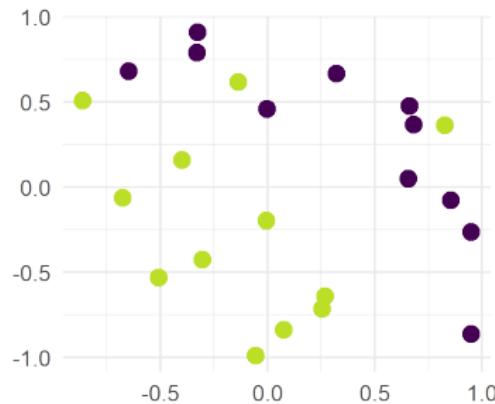
Pairwise coordinate ascent in the dual

- 1: Initialize $\alpha = 0$ (or more cleverly)
 - 2: **for** $t = 1, 2, \dots$ **do**
 - 3: Select some pair α_i, α_j to update next
 - 4: Optimize dual w.r.t. α_i, α_j , while holding α_k ($k \neq i, j$) fixed
 - 5: **end for**
-

The objective is quadratic in the pair, and $s := y^{(i)}\alpha_i + y^{(j)}\alpha_j$ must stay constant. So both α are changed by same (absolute) amount, the signs of the change depend on the labels.

TRAINING SVM IN THE DUAL

Assume we are in a valid state, $0 \leq \alpha_i \leq C$. Then we chose¹ two observations (encircled in red) for the next iteration. Note they have opposite labels so the sign of their change is equal.



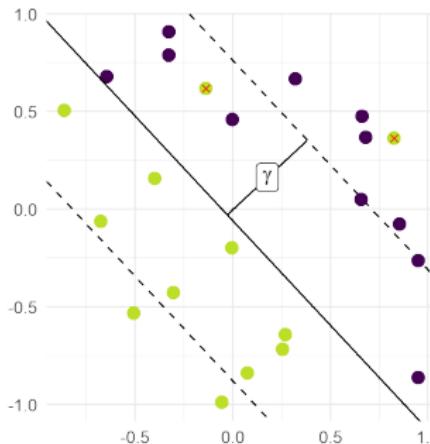
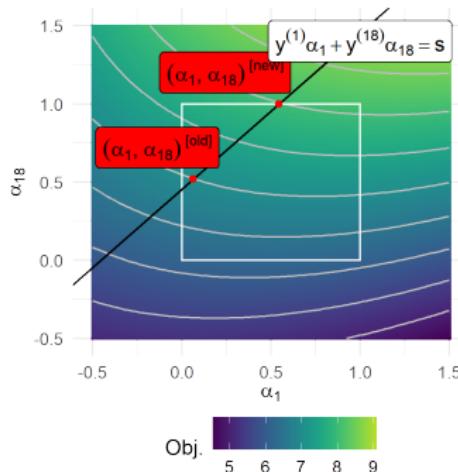
TRAINING SVM IN THE DUAL

$$\max_{\alpha} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \quad \sum_{i=1}^n \alpha_i y^{(i)} = 0$$

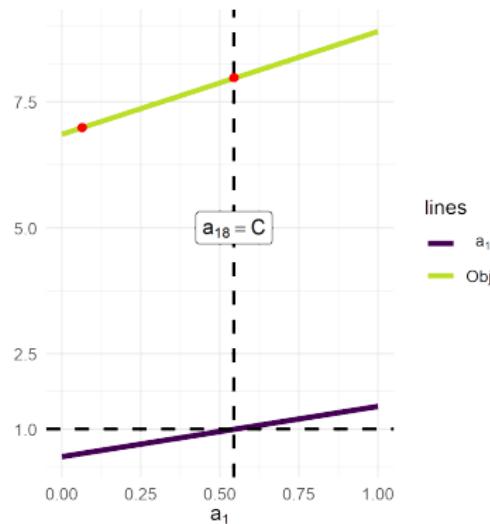
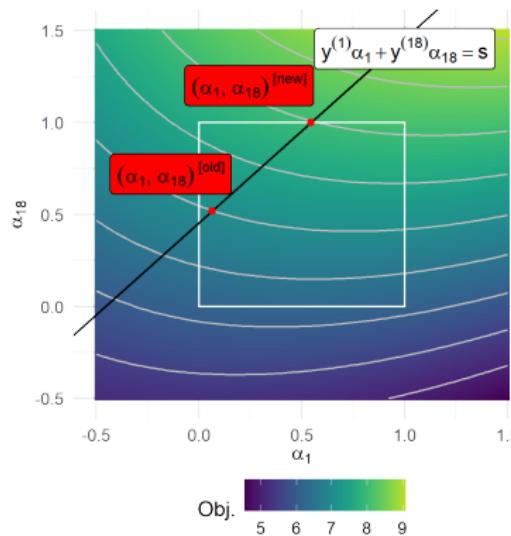
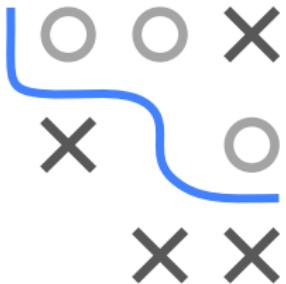


We move on the linear constraint until the pair-optimum or the bounday (here: $C = 1$).



TRAINING SVM IN THE DUAL

Sequential Minimal Optimization (SMO) exploits the fact that effectively we only need to solve a one-dimensional quadratic problem, over in interval, for which an analytical solution exists.



INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

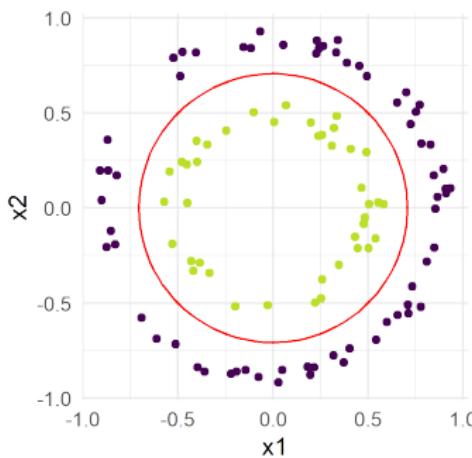
Boosting

Gaussian Processes



Introduction to Machine Learning

Feature Generation for Nonlinear Separation

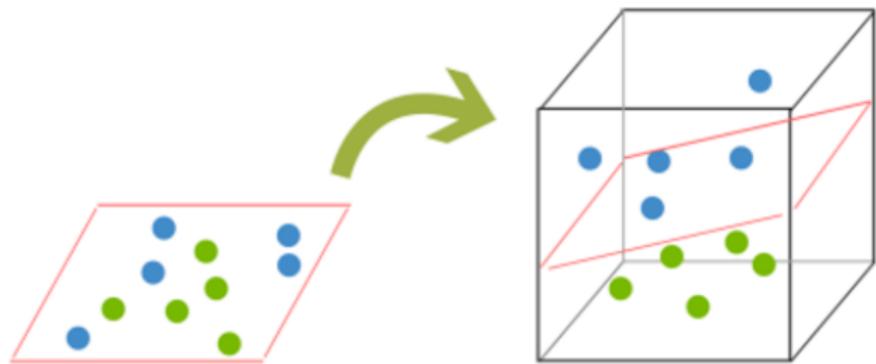


Learning goals

- Understand how nonlinearity can be introduced via feature maps in SVMs
- Know the limitation of feature maps

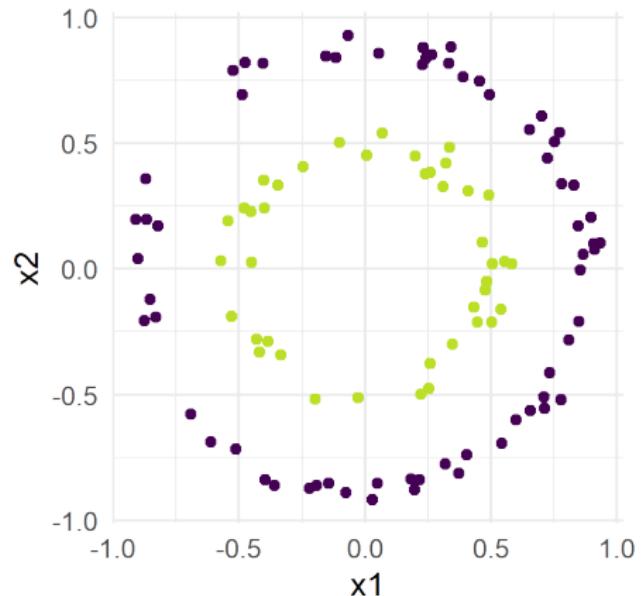
NONLINEARITY VIA FEATURE MAPS

- How to extend a linear classifier, e.g. the SVM, to nonlinear separation between classes?
- We could project the data from 2D into a richer 3D feature space!



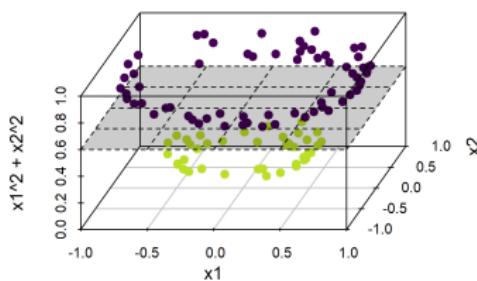
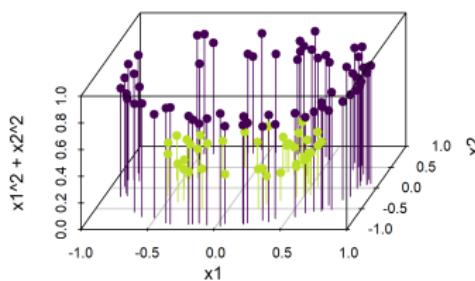
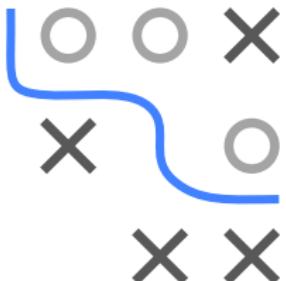
NONLINEARITY VIA FEATURE MAPS

In order to “lift” the data points into a higher dimension, we have to find a suitable **feature map** $\phi : \mathcal{X} \rightarrow \Phi$. Let us consider another example where the classes lie on two concentric circles:



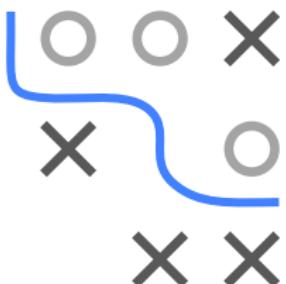
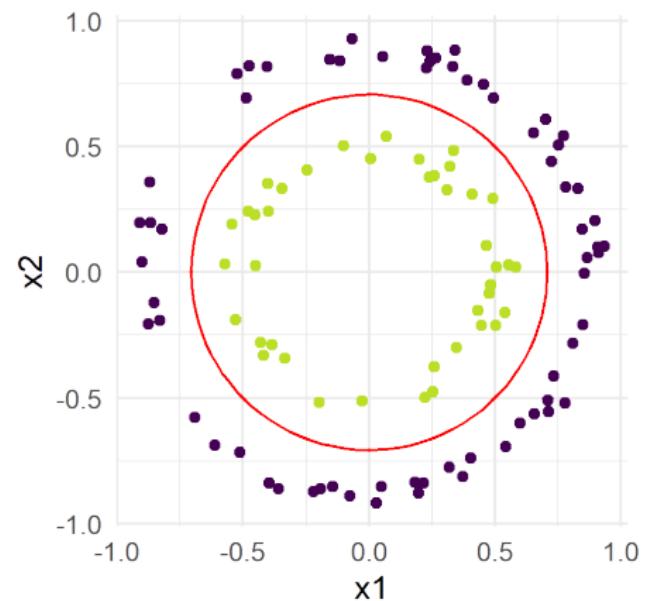
NONLINEARITY VIA FEATURE MAPS

We apply the feature map $\phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$ to map our points into a 3D space. Now our data can be separated by a hyperplane.



NONLINEARITY VIA FEATURE MAPS

The hyperplane learned in $\Phi \subset \mathbb{R}^3$ yields a nonlinear decision boundary when projected back to $\mathcal{X} = \mathbb{R}^2$.



FEATURE MAPS: COMPUTATIONAL LIMITATIONS

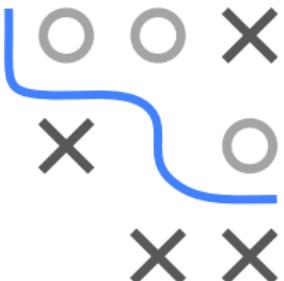
Let us have a look at a similar nonlinear feature map $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^5$, where we collect all monomial feature extractors up to degree 2 (pairwise interactions and quadratic effects):

$$\phi(x_1, x_2) = (x_1^2, x_2^2, x_1 x_2, x_1, x_2).$$

For p features vectors, there are k_1 different monomials where the degree is exactly d , and k_2 different monomials up to degree d .

$$k_1 = \binom{d + p - 1}{d} \quad k_2 = \binom{d + p}{d} - 1$$

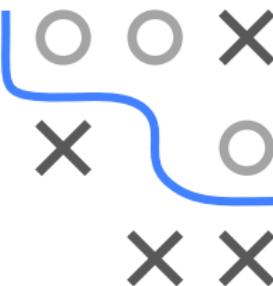
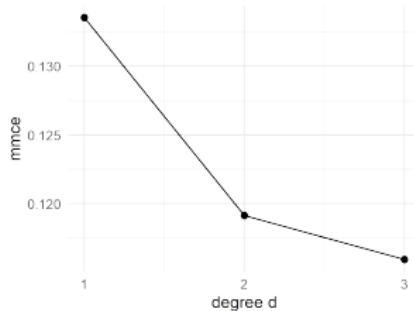
Which is quite a lot, if p is large.



FEATURE MAPS: COMPUTATIONAL LIMITATIONS

Let us see how well we can classify the 28×28 -pixel images of the handwritten digits of the MNIST dataset (70K observations across 10 classes). We use SVM with a nonlinear feature map which projects the images to a space of all monomials up to the degree d and $C = 1$:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
```

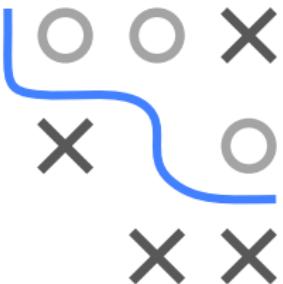
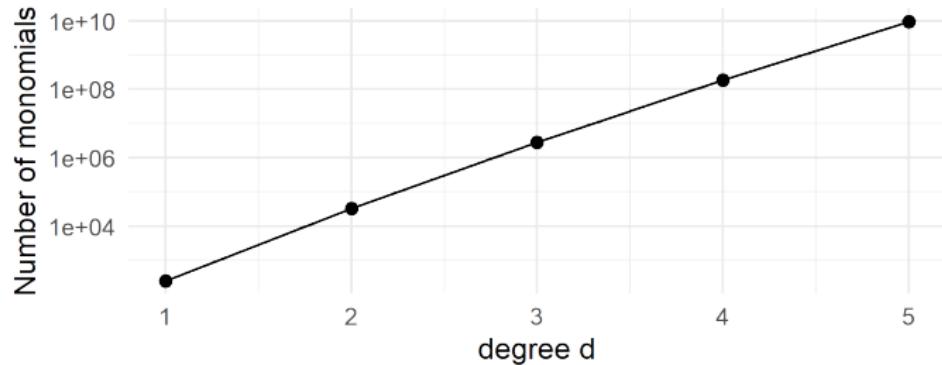


For this scenario, with increasing degree d the test mmce decreases.

NB: We handle the multiclass task with the "one-against-one" approach. We are somewhat lazy and only use 700 observations to train (rest for testing). We do not do any tuning - as we always should for the SVM!

FEATURE MAPS: COMPUTATIONAL LIMITATIONS

However, even a 16×16 -pixel input image results in infeasible dimensions for our extracted features (monomials up to degree d).



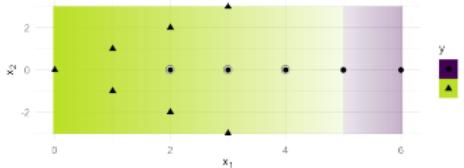
In this case, training classifiers like a linear SVM via dataset transformations will incur serious **computational and memory problems**.

Are we at a “dead end”?

Answer: No, this is why kernels exist!

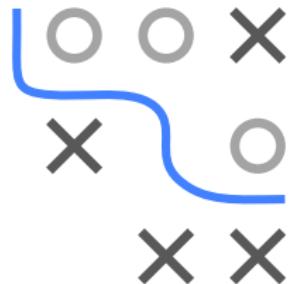
Introduction to Machine Learning

The Kernel Trick



Learning goals

- Know how to efficiently introduce non-linearity via the kernel trick
- Know common kernel functions (linear, polynomial, radial)
- Know how to compute predictions of the kernel SVM



DUAL SVM PROBLEM WITH FEATURE MAP

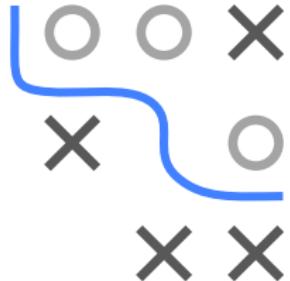
The dual (soft-margin) SVM is:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \\ & \sum_{i=1}^n \alpha_i y^{(i)} = 0, \end{aligned}$$

Here we replaced all features $\mathbf{x}^{(i)}$ with feature-generated, transformed versions $\phi(\mathbf{x}^{(i)})$.

We see: The optimization problem only depends on **pair-wise inner products** of the inputs.

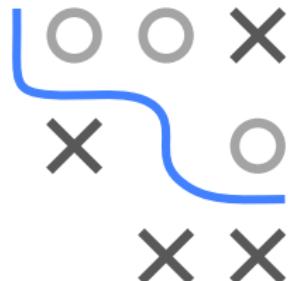
This now allows a trick to enable efficient solving.



KERNEL = FEATURE MAP + INNER PRODUCT

Instead of first mapping the features to the higher-dimensional space and calculating the inner products afterwards,

$$\begin{array}{ccc} \mathbf{x}^{(i)} & \xrightarrow{\quad} & \phi(\mathbf{x}^{(i)}) \\ & & \searrow \\ & & \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle \\ \mathbf{x}^{(j)} & \xrightarrow{\quad} & \phi(\mathbf{x}^{(j)}) \end{array}$$



it would be nice to have an efficient “shortcut” computation:

$$\begin{array}{ccc} \mathbf{x}^{(i)} & & \xrightarrow{\quad} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \\ & \swarrow & \\ \mathbf{x}^{(j)} & & \xrightarrow{\quad} \end{array}$$

We will see: **Kernels** give us such a “shortcut”.

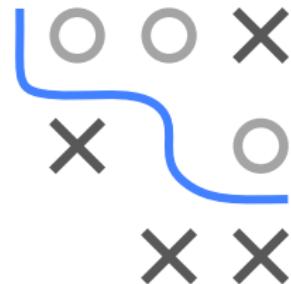
MERCER KERNEL

Definition: A (Mercer) kernel on a space \mathcal{X} is a continuous function

$$k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$$

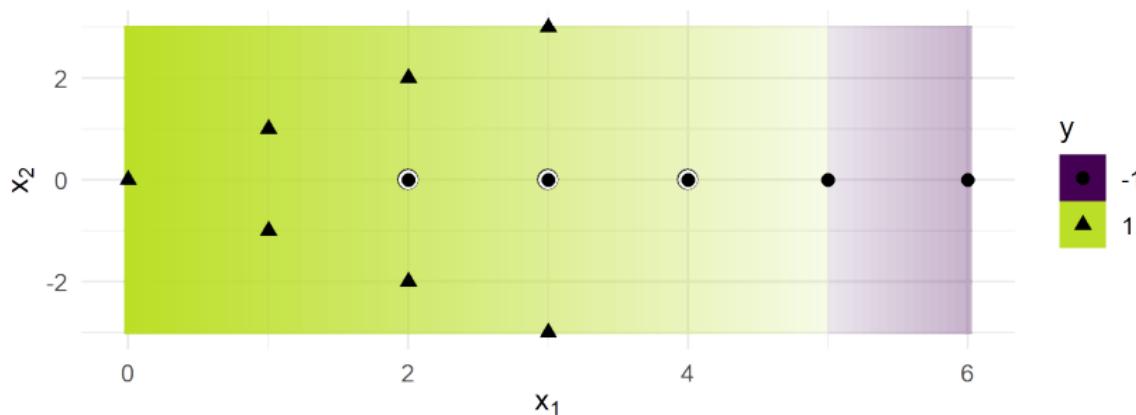
of two arguments with the properties

- Symmetry: $k(\mathbf{x}, \tilde{\mathbf{x}}) = k(\tilde{\mathbf{x}}, \mathbf{x})$ for all $\mathbf{x}, \tilde{\mathbf{x}} \in \mathcal{X}$.
- Positive definiteness: For each finite subset $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$ the **kernel Gram matrix** $K \in \mathbb{R}^{n \times n}$ with entries $K_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ is positive semi-definite.



CONSTANT AND LINEAR KERNEL

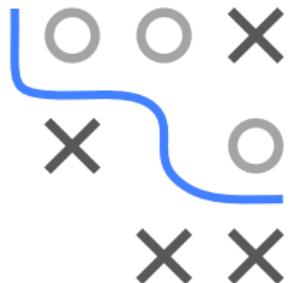
- Every constant function taking a non-negative value is a (very boring) kernel.
- An inner product is a kernel. We call the standard inner product $k(\mathbf{x}, \tilde{\mathbf{x}}) = \mathbf{x}^\top \tilde{\mathbf{x}}$ the **linear kernel**. This is simply our usual linear SVM as discussed.



SUM AND PRODUCT KERNELS

A kernel can be constructed from other kernels k_1 and k_2 :

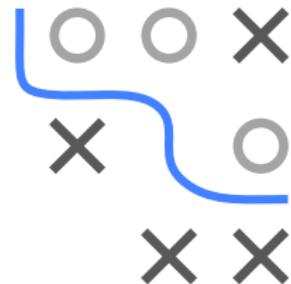
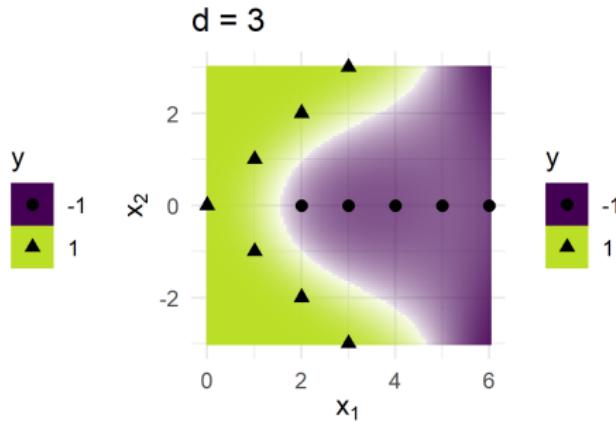
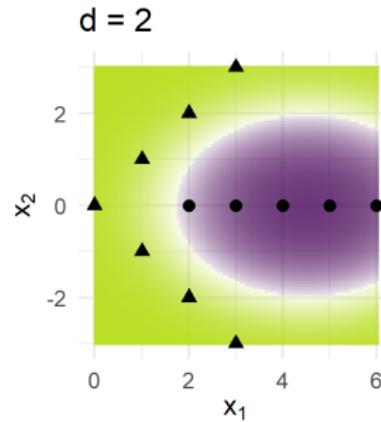
- For $\lambda \geq 0$, $\lambda \cdot k_1$ is a kernel.
- $k_1 + k_2$ is a kernel.
- $k_1 \cdot k_2$ is a kernel (thus also k_1^n).



The proofs remain as (simple) exercises.

POLYNOMIAL KERNEL

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = (\mathbf{x}^\top \tilde{\mathbf{x}} + b)^d, \text{ for } b \geq 0, d \in \mathbb{N}$$



From the sum-product rules it directly follows that this is a kernel.

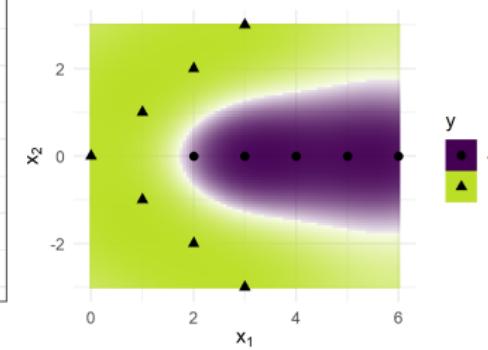
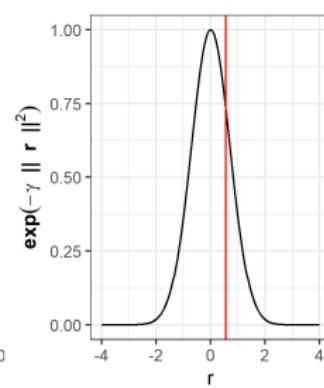
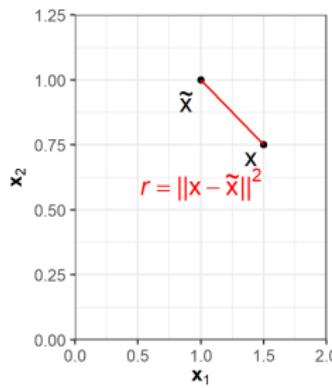
RBF KERNEL

The “radial” **Gaussian kernel** is defined as

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = \exp\left(-\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|^2}{2\sigma^2}\right)$$

or

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = \exp(-\gamma \|\mathbf{x} - \tilde{\mathbf{x}}\|^2), \gamma > 0$$



KERNEL SVM

We kernelize the dual (soft-margin) SVM problem by replacing all inner products $\langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle$ by kernels $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle$$

$$\text{s.t. } 0 \leq \alpha_i \leq C,$$

$$\sum_{i=1}^n \alpha_i y^{(i)} = 0.$$



This problem is still convex because \mathbf{K} is psd!

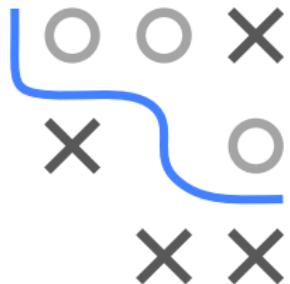
KERNEL SVM

We kernelize the dual (soft-margin) SVM problem by replacing all inner products $\langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle$ by kernels $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$

$$\max_{\alpha} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C,$$

$$\sum_{i=1}^n \alpha_i y^{(i)} = 0.$$



This problem is still convex because \mathbf{K} is psd!

KERNEL SVM

We kernelize the dual (soft-margin) SVM problem by replacing all inner products $\langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle$ by kernels $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$



$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

$$\text{s.t. } 0 \leq \alpha_i \leq C,$$

$$\sum_{i=1}^n \alpha_i y^{(i)} = 0.$$

In more compact matrix notation with \mathbf{K} denoting the kernel matrix:

$$\max_{\alpha \in \mathbb{R}^n} \mathbf{1}^\top \alpha - \frac{1}{2} \alpha^\top \text{diag}(\mathbf{y}) \mathbf{K} \text{diag}(\mathbf{y}) \alpha$$

$$\text{s.t. } \alpha^\top \mathbf{y} = 0,$$

$$0 \leq \alpha \leq C.$$

This problem is still convex because \mathbf{K} is psd!

KERNEL SVM: PREDICTIONS

For the linear soft-margin SVM we had:

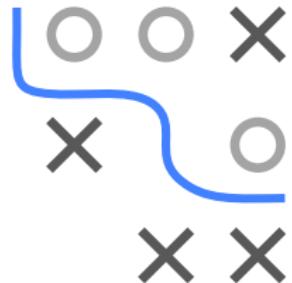
$$f(\mathbf{x}) = \hat{\theta}^T \mathbf{x} + \theta_0 \quad \text{and} \quad \hat{\theta} = \sum_{i=1}^n \alpha_i y^{(i)} \mathbf{x}^{(i)}$$

After the feature map this becomes:

$$f(\mathbf{x}) = \langle \hat{\theta}, \phi(\mathbf{x}) \rangle + \theta_0 \quad \text{and} \quad \hat{\theta} = \sum_{i=1}^n \alpha_i y^{(i)} \phi(\mathbf{x}^{(i)})$$

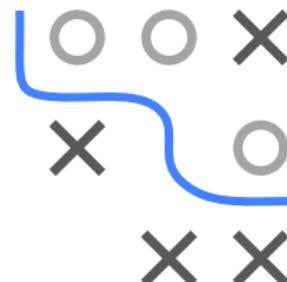
Assuming that the dot-product still follows its bi-linear rules in the mapped space and using the kernel trick again:

$$\begin{aligned} \langle \hat{\theta}, \phi(\mathbf{x}) \rangle &= \left\langle \sum_{i=1}^n \alpha_i y^{(i)} \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}) \right\rangle = \sum_{i=1}^n \alpha_i y^{(i)} \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}) \rangle = \\ &= \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}), \quad \text{so:} \quad f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0 \end{aligned}$$



MNIST EXAMPLE

- Through this kernelization we can now conveniently perform feature generation even for higher-dimensional data. Actually, this is how we computed all previous examples, too.
- We again consider MNIST with 28×28 bitmaps of gray values.
- A polynomial kernel extracts $\binom{d+p}{d} - 1$ features and for the RBF kernel the dimensionality would be infinite.
- We train SVMs again on 700 observations of the MNIST data set and use the rest of the data for testing; and use $C=1$.



0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

	Error
linear	0.134
poly ($d = 2$)	0.119
RBF ($\gamma = 0.001$)	0.12
RBF ($\gamma = 1$)	0.184

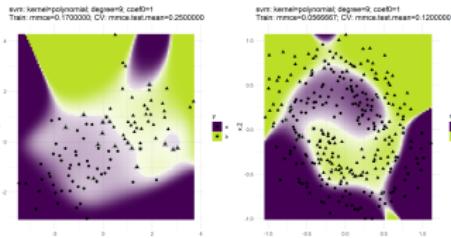
FINAL COMMENTS

- The kernel trick allows us to make linear machines non-linear in a very efficient manner.
- Linear separation in high-dimensional spaces is **very flexible**.
- Learning takes place in the feature space, while predictions are computed in the input space.
- Both the polynomial and Gaussian kernels can be computed in linear time. Computing inner products of features is **much faster** than computing the features themselves.
- What if a good feature map ϕ is already available? Then this feature map canonically induces a kernel by defining $k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle$. There is no problem with an explicit feature representation as long as it is efficiently computable.



Introduction to Machine Learning

The Polynomial Kernel



Learning goals

- Know the homogeneous and non-homogeneous polynomial kernel
- Understand the influence of the choice of the degree on the decision boundary

HOMOGENEOUS POLYNOMIAL KERNEL

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = (\mathbf{x}^T \tilde{\mathbf{x}})^d, \text{ for } d \in \mathbb{N}$$

The feature map contains all monomials of exactly order d .

$$\phi(\mathbf{x}) = \left(\sqrt{\binom{d}{k_1, \dots, k_p}} x_1^{k_1} \dots x_p^{k_p} \right)_{k_i \geq 0, \sum_i k_i = d}$$

That $\langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle = k(\mathbf{x}, \tilde{\mathbf{x}})$ holds can easily be checked by simple calculation and using the multinomial formula

$$(x_1 + \dots + x_p)^d = \sum_{k_i \geq 0, \sum_i k_i = d} \binom{d}{k_1, \dots, k_p} x_1^{k_1} \dots x_p^{k_p}$$

The map $\phi(\mathbf{x})$ has $\binom{p+d-1}{d}$ dimensions. We see that $\phi(\mathbf{x})$ contains no terms of "lesser" order, so, e.g., linear effects. As an example for $p = d = 2$: $\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2)$.



NONHOMOGENEOUS POLYNOMIAL KERNEL

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = (\mathbf{x}^T \tilde{\mathbf{x}} + b)^d, \text{ for } b \geq 0, d \in \mathbb{N}$$

The maths is very similar as before, we kind of add a further constant term in the original space, with

$$(\mathbf{x}^T \tilde{\mathbf{x}} + b)^d = (x_1 \tilde{x}_1 + \dots + x_p \tilde{x}_p + b)^d$$

The feature map contains all monomials up to order d .

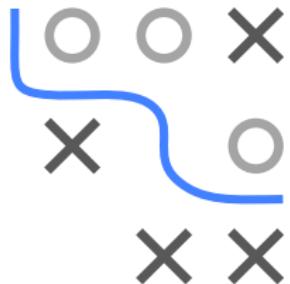
$$\phi(\mathbf{x}) = \left(\sqrt{\binom{d}{k_1, \dots, k_{p+1}}} x_1^{k_1} \dots x_p^{k_p} b^{k_{p+1}/2} \right)_{k_i \geq 0, \sum_i k_i = d}$$

The map $\phi(\mathbf{x})$ has $\binom{p+d}{d}$ dimensions. For $p = d = 2$:

$$(x_1 \tilde{x}_1 + x_2 \tilde{x}_2 + b)^2 = x_1^2 \tilde{x}_1^2 + x_2^2 \tilde{x}_2^2 + 2x_1 x_2 \tilde{x}_1 \tilde{x}_2 + 2bx_1 \tilde{x}_1 + 2bx_2 \tilde{x}_2 + b^2$$

Therefore,

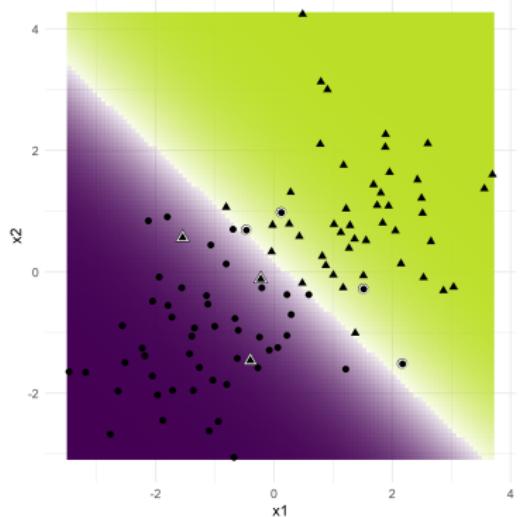
$$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2, \sqrt{2b}x_1, \sqrt{2b}x_2, b)$$



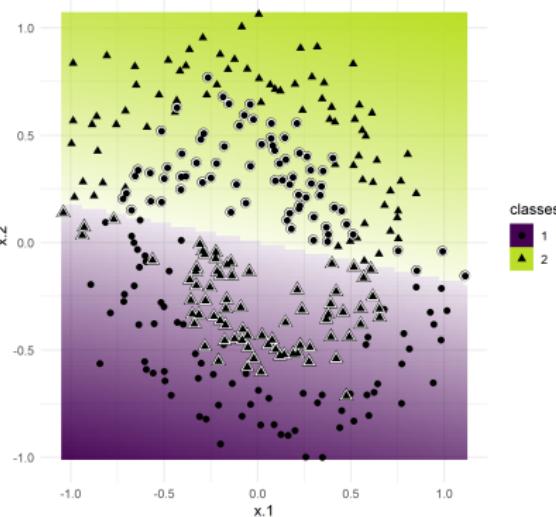
POLYNOMIAL KERNEL

Degree $d = 1$ yields a linear decision boundary.

```
svm: kernel=polynomial; degree=1; coef0=1  
Train: mmce=0.0700000; CV: mmce.test.mean=0.1100000
```



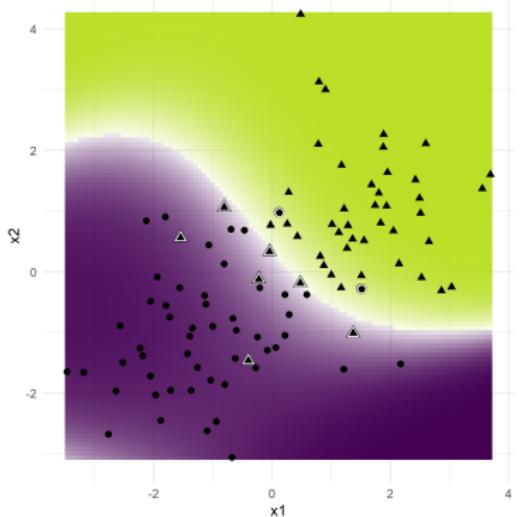
```
svm: kernel=polynomial; degree=1; coef0=1  
Train: mmce=0.5000000; CV: mmce.test.mean=0.5066667
```



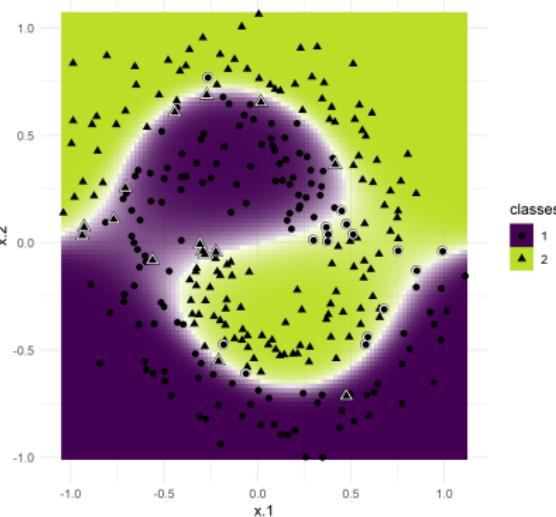
POLYNOMIAL KERNEL

The higher the degree, the more nonlinearity in the decision boundary.

```
svm: kernel=polynomial; degree=3; coef0=1  
Train: mmce=0.0900000; CV: mmce.test.mean=0.1200000
```



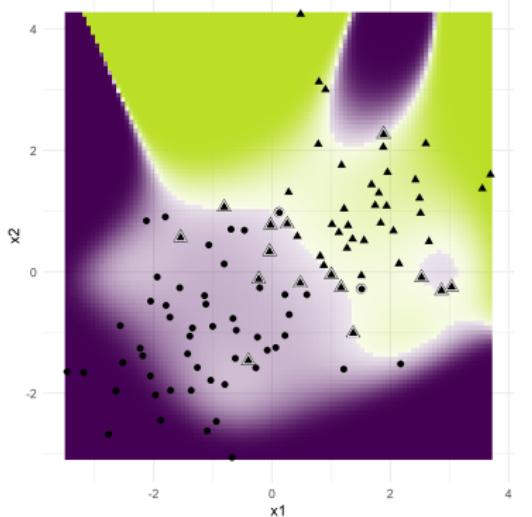
```
svm: kernel=polynomial; degree=3; coef0=1  
Train: mmce=0.1033333; CV: mmce.test.mean=0.1233333
```



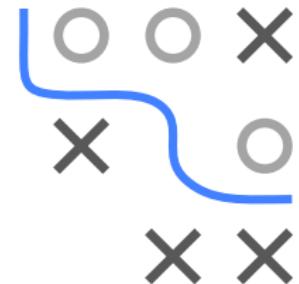
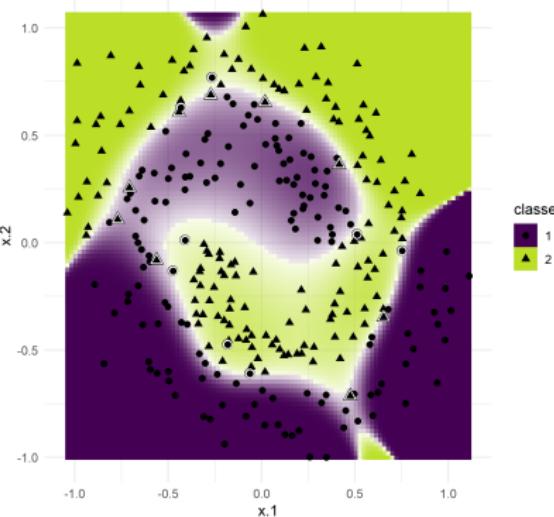
POLYNOMIAL KERNEL

The higher the degree, the more nonlinearity in the decision boundary.

```
svm: kernel=polynomial; degree=9; coef0=1  
Train: mmce=0.1700000; CV: mmce.test.mean=0.2500000
```



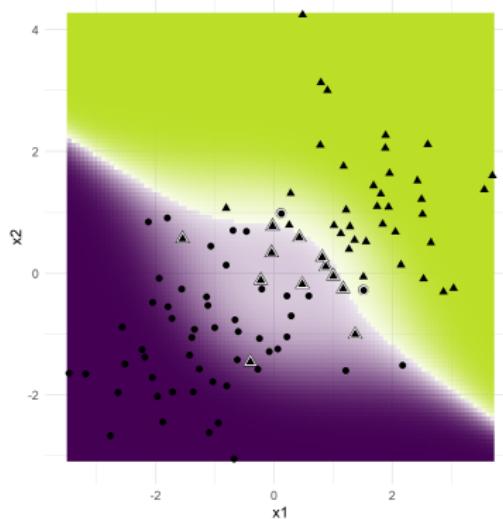
```
svm: kernel=polynomial; degree=9; coef0=1  
Train: mmce=0.0566667; CV: mmce.test.mean=0.1200000
```



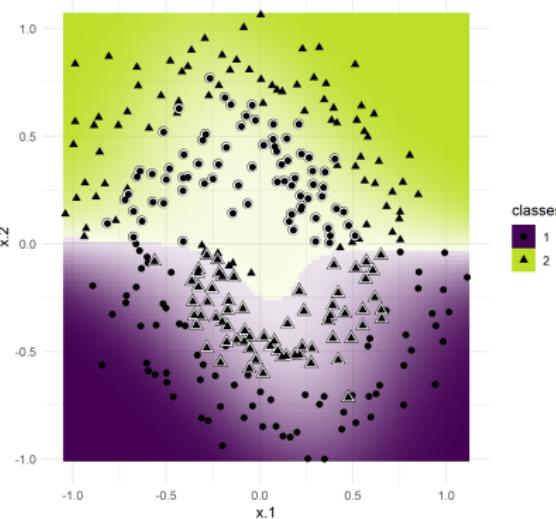
POLYNOMIAL KERNEL

For $k(\mathbf{x}, \tilde{\mathbf{x}}) = (\mathbf{x}^\top \tilde{\mathbf{x}} + 0)^d$ we get no lower order effects.

```
svm: kernel=polynomial; degree=3; coef0=0  
Train: mmce=0.1400000; CV: mmce.test.mean=0.1800000
```

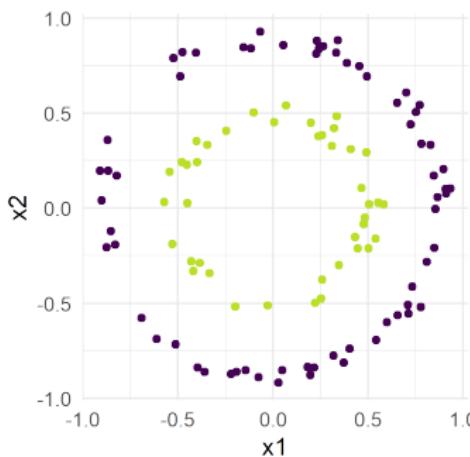
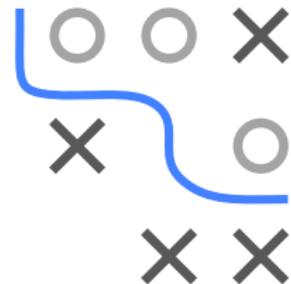


```
svm: kernel=polynomial; degree=3; coef0=0  
Train: mmce=0.4733333; CV: mmce.test.mean=0.4833333
```



Introduction to Machine Learning

Reproducing Kernel Hilbert Space and Representer Theorem



Learning goals

- Know that for every kernel there is an associated feature map and space (Mercer's Theorem)
- Know that this feature map is not unique, and the reproducing kernel Hilbert space (RKHS) is a reference space
- Know the representation of the solution of a SVM is given by the

.

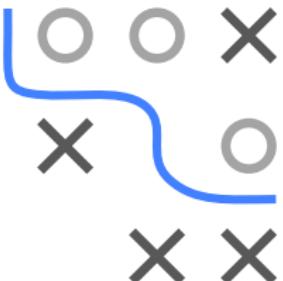
.

.

KERNELS: MERCER'S THEOREM

- Kernels are symmetric, positive definite functions $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.
- A kernel can be thought of as a shortcut computation for a two-step procedure: the feature map and the inner product.

Mercer's theorem says that for every kernel there exists an associated (well-behaved) feature space where the kernel acts as a dot-product.



- There exists a Hilbert space Φ of continuous functions $\mathcal{X} \rightarrow \mathbb{R}$ (think of it as a vector space with inner product where all operations are meaningful, including taking limits of sequences; this is non-trivial in the infinite-dimensional case)
- and a continuous “feature map” $\phi : \mathcal{X} \rightarrow \Phi$,
- so that the kernel computes the inner product of the features:

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle .$$

REPRODUCING KERNEL HILBERT SPACE

- There are many possible Hilbert spaces and feature maps for the same kernel, but they are all “equivalent” (isomorphic).
- It is often helpful to have a reference space for a kernel $k(\cdot, \cdot)$, called the **reproducing kernel Hilbert space (RKHS)**.
- The feature map of this space is

$$\phi : \mathcal{X} \rightarrow \mathcal{C}(\mathcal{X}); \quad \mathbf{x} \mapsto k(\mathbf{x}, \cdot) ,$$

where $\mathcal{C}(\mathcal{X})$ is the space of continuous functions $\mathcal{X} \rightarrow \mathbb{R}$. The “features” of the RKHS are the kernel functions evaluated at an \mathbf{x} .

- The Hilbert space is the completion of the span of the features:

$$\Phi = \overline{\text{span}\{\phi(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\}} \subset \mathcal{C}(\mathcal{X}) .$$

- The so-called **reproducing property** states:

$$\langle k(\mathbf{x}, \cdot), k(\tilde{\mathbf{x}}, \cdot) \rangle = \langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle = k(\mathbf{x}, \tilde{\mathbf{x}}).$$

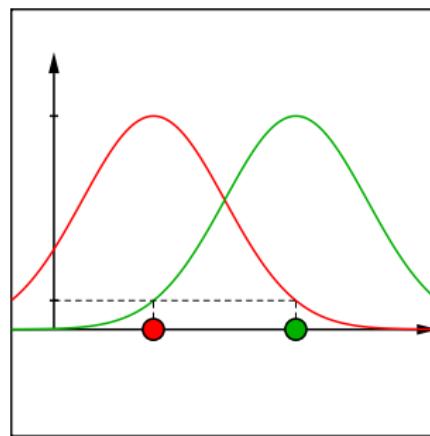


REPRODUCING KERNEL HILBERT SPACE

- The RKHS provides us with a useful interpretation:
an input $\mathbf{x} \in \mathcal{X}$ mapped to the **basis function** $\phi(\mathbf{x}) = k(\mathbf{x}, \cdot)$.
- The kernel maps 2 points and computes the inner product:

$$\langle k(\mathbf{x}, \cdot), k(\tilde{\mathbf{x}}, \cdot) \rangle = k(\mathbf{x}, \tilde{\mathbf{x}}) .$$

- This is best illustrated with the Gaussian kernel.



REPRODUCING KERNEL HILBERT SPACE

- Caveat: Not all elements of the Hilbert space are of the form $k(\mathbf{x}, \cdot)$ for some $\mathbf{x} \in \mathcal{X}$!
- A general element in the span takes the form

$$\sum_{i=1}^n \alpha_i k(\mathbf{x}^{(i)}, \cdot) \in \Phi .$$



- A general element in the closure of the span takes the form

$$\sum_{i=1}^{\infty} \alpha_i k(\mathbf{x}^{(i)}, \cdot) \in \Phi .$$

with $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$.

REPRODUCING KERNEL HILBERT SPACE

What is $\langle f, g \rangle$ for two elements

$$f = \sum_{i=1}^n \alpha_i k(\mathbf{x}^{(i)}, \cdot), \quad g = \sum_{j=1}^m \beta_j k(\mathbf{x}^{(j)}, \cdot) ?$$



We use the bilinearity of the inner product:

$$\begin{aligned} \left\langle \sum_{i=1}^n \alpha_i k(\mathbf{x}^{(i)}, \cdot), \sum_{j=1}^m \beta_j k(\mathbf{x}^{(j)}, \cdot) \right\rangle &= \sum_{i=1}^n \alpha_i \left\langle k(\mathbf{x}^{(i)}, \cdot), \sum_{j=1}^m \beta_j k(\mathbf{x}^{(j)}, \cdot) \right\rangle \\ &= \sum_{i=1}^n \sum_{j=1}^m \alpha_i \beta_j \left\langle k(\mathbf{x}^{(i)}, \cdot), k(\mathbf{x}^{(j)}, \cdot) \right\rangle \\ &= \sum_{i=1}^n \sum_{j=1}^m \alpha_i \beta_j k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \end{aligned}$$

The kernel defines the inner products of all elements in the span of the basis functions.

REPRESENTER THEOREM

The **representer theorem** tells us that the solution of a support vector machine problem

$$\min_{\theta, \theta_0, \zeta^{(i)}} \frac{1}{2} \theta^\top \theta + C \sum_{i=1}^n \zeta^{(i)}$$

$$\text{s.t. } y^{(i)} \left(\langle \theta, \phi(\mathbf{x}^{(i)}) \rangle + \theta_0 \right) \geq 1 - \zeta^{(i)} \quad \forall i \in \{1, \dots, n\},$$

$$\text{and } \zeta^{(i)} \geq 0 \quad \forall i \in \{1, \dots, n\}$$

can be written as

$$\theta = \sum_{j=1}^n \beta_j \phi(\mathbf{x}^{(j)})$$

for $\beta_j \in \mathbb{R}$.



REPRESENTER THEOREM

Theorem (Representer Theorem):

The solution θ, θ_0 of the support vector machine optimization problem fulfills $\theta \in V = \text{span} \{ \phi(\mathbf{x}^{(1)}), \dots, \phi(\mathbf{x}^{(n)}) \}$.

Proof: Let V^\perp denote the space orthogonal to V , so that $\Phi = V \oplus V^\perp$. The vector θ has a unique decomposition into components $v \in V$ and $v^\perp \in V^\perp$, so that $v + v^\perp = \theta$.

The regularizer becomes $\|\theta\|^2 = \|v\|^2 + \|v^\perp\|^2$. The constraints

$y^{(i)} (\langle \theta, \phi(\mathbf{x}^{(i)}) \rangle + \theta_0) \geq 1 - \zeta^{(i)}$ do not depend on v^\perp at all:

$$\langle \theta, \phi(\mathbf{x}^{(i)}) \rangle = \underbrace{\langle v, \phi(\mathbf{x}^{(i)}) \rangle}_{=0} + \underbrace{\langle v^\perp, \phi(\mathbf{x}^{(i)}) \rangle}_{=0} \quad \forall i \in \{1, 2, \dots, n\}.$$

Thus, we have two independent optimization problems, namely the standard SVM problem for v and the unconstrained minimization problem of $\|v^\perp\|^2$ for v^\perp , with obvious solution $v^\perp = 0$. Thus, $\theta = v \in V$.



REPRESENTER THEOREM

- Hence, we can restrict the SVM optimization problem to the **finite-dimensional** subspace span $\{\phi(\mathbf{x}^{(1)}), \dots, \phi(\mathbf{x}^{(n)})\}$. Its dimension grows with the size of the training set.
- More explicitly, we can assume the form

$$\boldsymbol{\theta} = \sum_{j=1}^n \beta_j \cdot \phi(\mathbf{x}^{(j)})$$

for the weight vector $\boldsymbol{\theta} \in \Phi$.

- The SVM prediction on $\mathbf{x} \in \mathcal{X}$ can be computed as

$$f(\mathbf{x}) = \sum_{j=1}^n \beta_j \langle \phi(\mathbf{x}^{(j)}), \phi(\mathbf{x}) \rangle + \theta_0 .$$

It can be shown that the sum is **sparse**: $\beta_j = 0$ for non-support vectors.



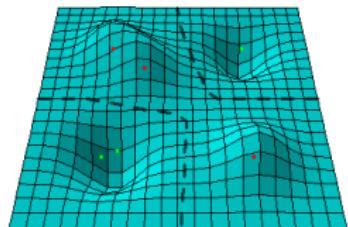
Introduction to Machine Learning

The Gaussian RBF Kernel



Learning goals

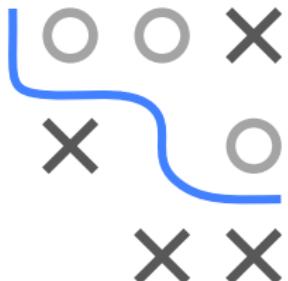
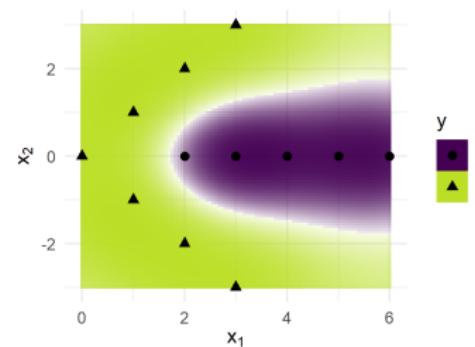
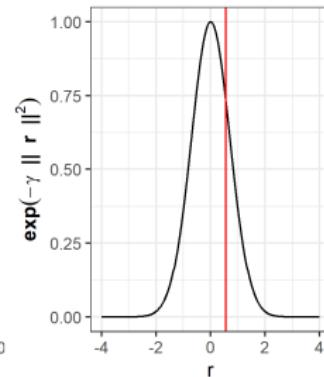
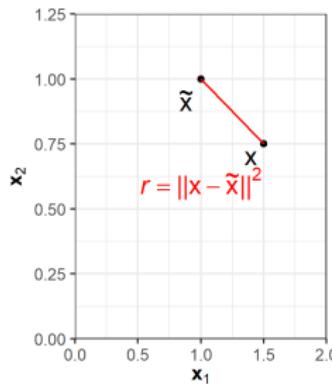
- Know the Gaussian (RBF) kernel
- Understand that all data sets are separable with this kernel
- Understand the effect of the kernel hyperparameter σ



RBF KERNEL

The “radial” **Gaussian kernel** is defined as

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = \exp\left(-\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|^2}{2\sigma^2}\right) \quad \text{or} \quad k(\mathbf{x}, \tilde{\mathbf{x}}) = \exp(-\gamma\|\mathbf{x} - \tilde{\mathbf{x}}\|^2)$$



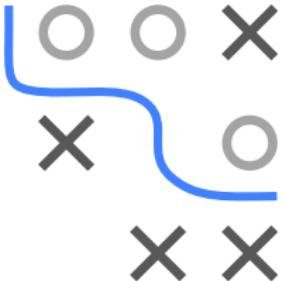
A straightforward extension is

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = \exp\left(-(\mathbf{x} - \tilde{\mathbf{x}})^T C (\mathbf{x} - \tilde{\mathbf{x}})\right)$$

for a symmetric, positive definite matrix C .

RBF KERNEL

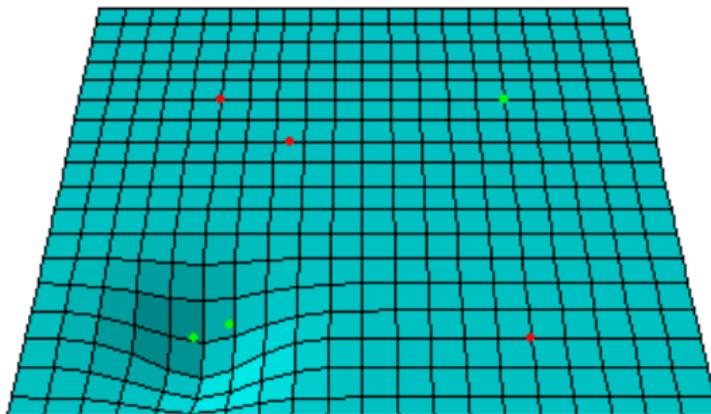
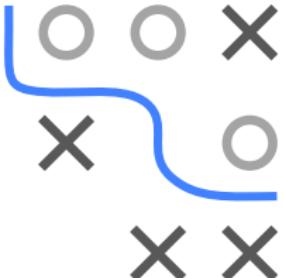
- With a Gaussian kernel, all RKHS basis functions $\phi(\mathbf{x}) = k(\mathbf{x}, \cdot)$ are linearly independent - which we will not prove here.
- This means that all (finite) data sets are linearly separable!
- Do we then need soft-margin machines? The answer is “yes”. The roles of the nonlinear feature map and the soft-margin constraints are very different:
 - The purpose of the kernel (and its feature map) is to make learning “easy”.
 - Even in an infinite-dimensional feature space we may want some margin violators because we should not trust noisy data. A hard-margin SVM with Gaussian kernels may be able to separate any dataset but will usually overfit.



WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the effect of the RBF kernel much better as a local model.

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$

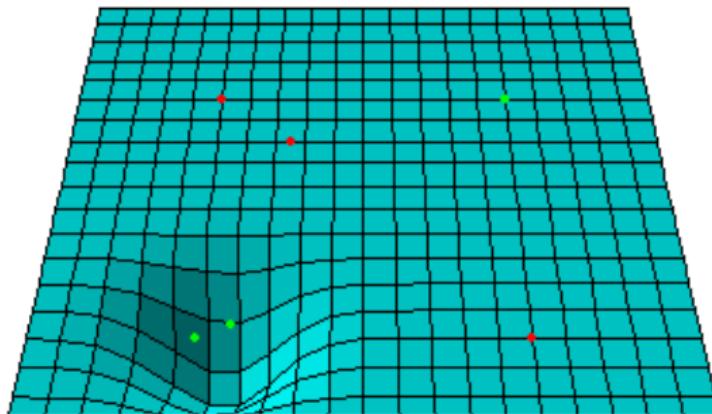
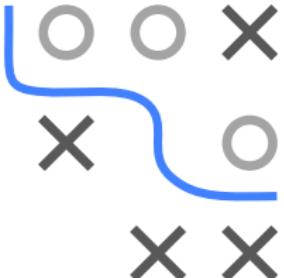


All support vectors are assigned RBF "bumps", these are weighted with the dual variables / Lagrange multipliers α_i and labels $y^{(i)}$. We then "mix" these bumps together to form the decision score function. Which becomes a bumpy surface.

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the effect of the RBF kernel much better as a local model.

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$

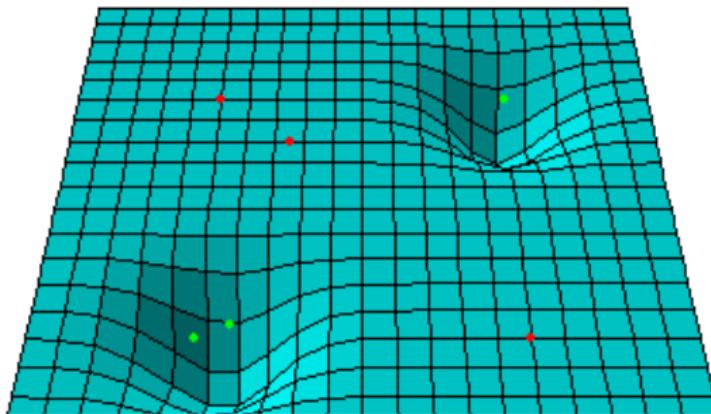
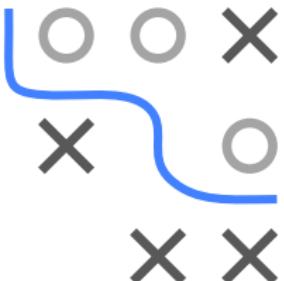


All support vectors are assigned RBF "bumps", these are weighted with the dual variables / Lagrange multipliers α_i and labels $y^{(i)}$. We then "mix" these bumps together to form the decision score function. Which becomes a bumpy surface.

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the effect of the RBF kernel much better as a local model.

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$

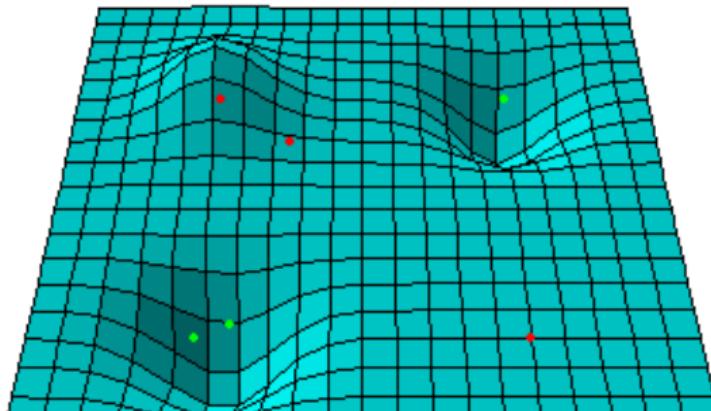


All support vectors are assigned RBF "bumps", these are weighted with the dual variables / Lagrange multipliers α_i and labels $y^{(i)}$. We then "mix" these bumps together to form the decision score function. Which becomes a bumpy surface.

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the effect of the RBF kernel much better as a local model.

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$

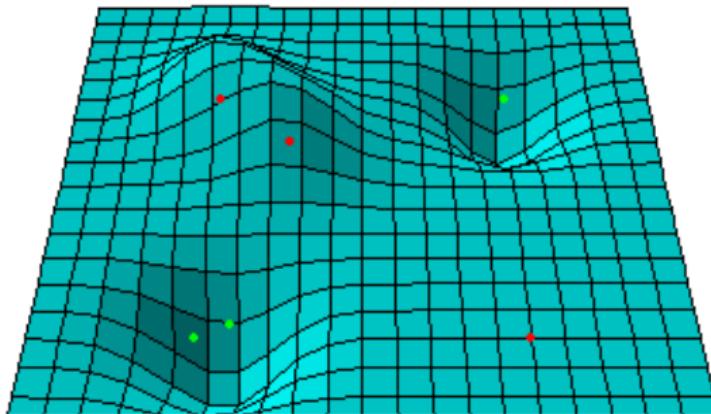


All support vectors are assigned RBF "bumps", these are weighted with the dual variables / Lagrange multipliers α_i and labels $y^{(i)}$. We then "mix" these bumps together to form the decision score function. Which becomes a bumpy surface.

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the effect of the RBF kernel much better as a local model.

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$

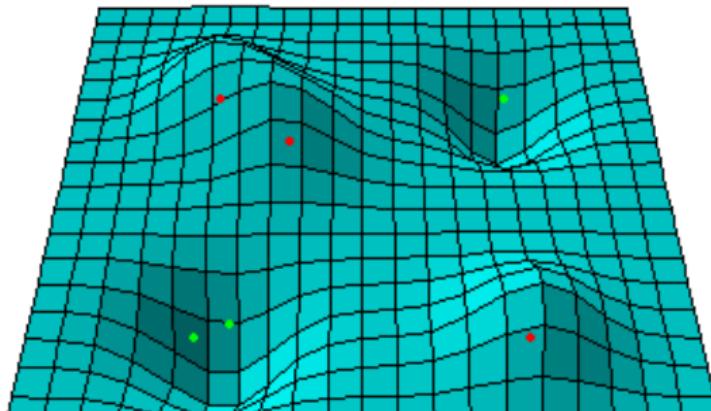


All support vectors are assigned RBF "bumps", these are weighted with the dual variables / Lagrange multipliers α_i and labels $y^{(i)}$. We then "mix" these bumps together to form the decision score function. Which becomes a bumpy surface.

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the effect of the RBF kernel much better as a local model.

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$

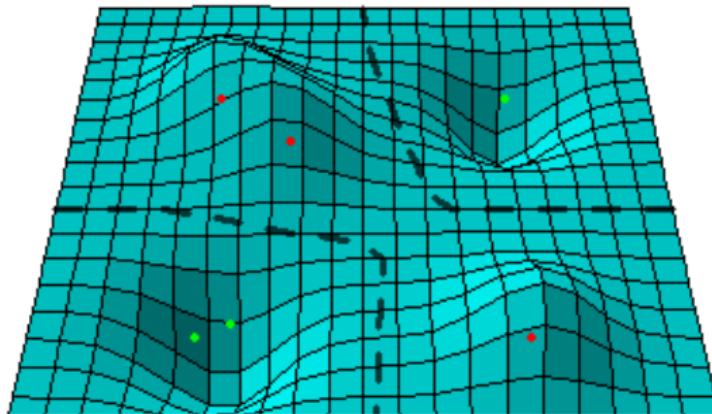
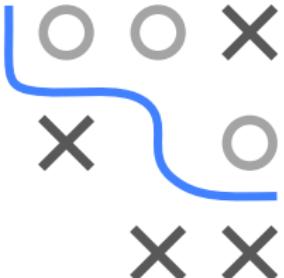


All support vectors are assigned RBF "bumps", these are weighted with the dual variables / Lagrange multipliers α_i and labels $y^{(i)}$. We then "mix" these bumps together to form the decision score function. Which becomes a bumpy surface.

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the effect of the RBF kernel much better as a local model.

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$

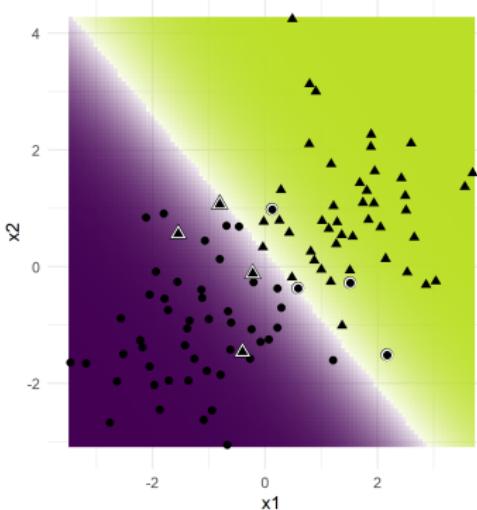


All support vectors are assigned RBF "bumps", these are weighted with the dual variables / Lagrange multipliers α_i and labels $y^{(i)}$. We then "mix" these bumps together to form the decision score function. Which becomes a bumpy surface.

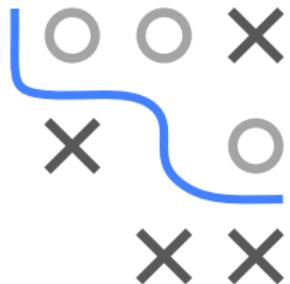
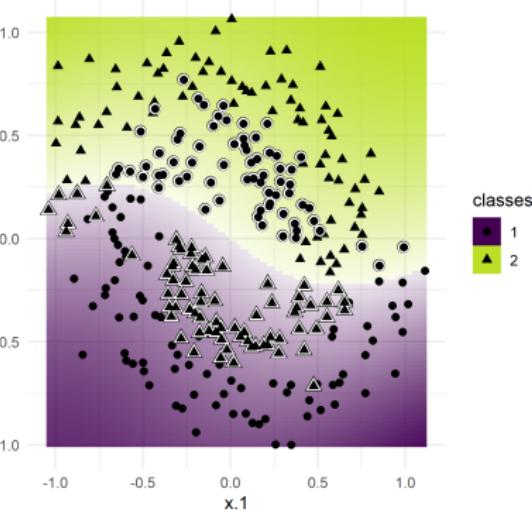
RBF KERNEL WIDTH

A large σ (or a small γ) will make the decision boundary very smooth and in the limit almost linear.

svm: kernel=radial; cost=1; gamma=0.01
Train: mmce=0.0800000; CV: mmce.test.mean=0.1100000

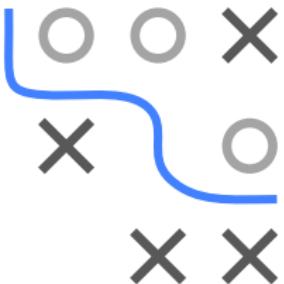
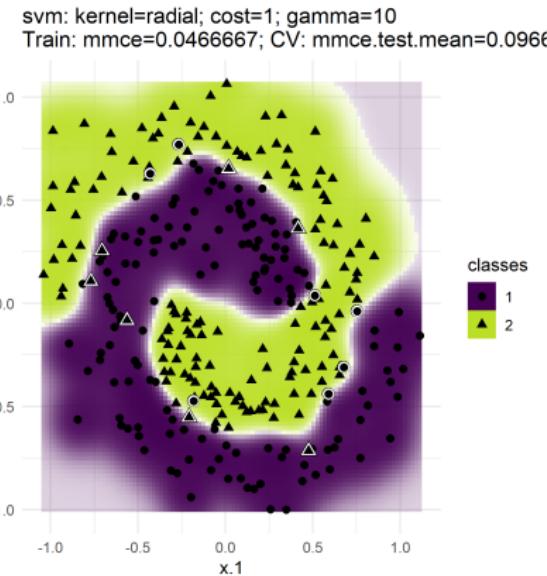
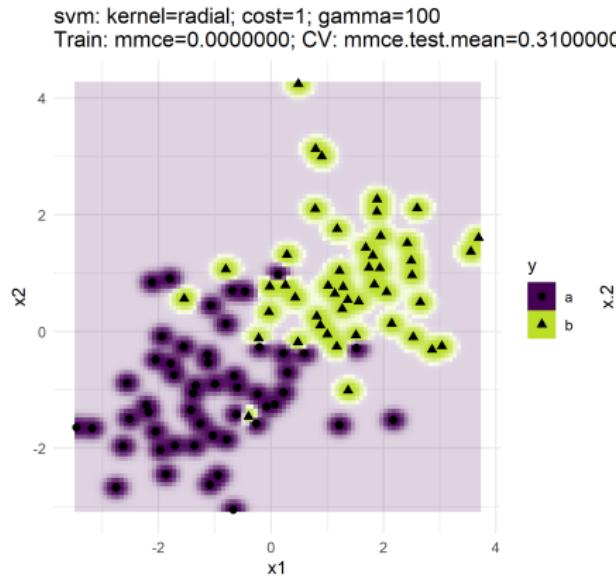


svm: kernel=radial; cost=1; gamma=0.08
Train: mmce=0.4766667; CV: mmce.test.mean=0.4900000



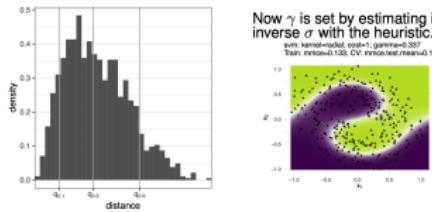
RBF KERNEL WIDTH

A small σ parameter makes the function more “wiggly”, in the limit we totally over fit the data by basically modelling each training data point - and maximal uncertainty at all other test points.



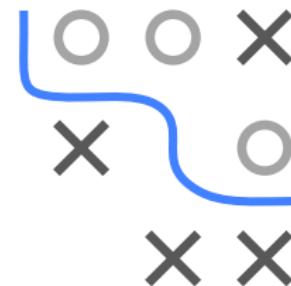
Introduction to Machine Learning

SVM Model Selection



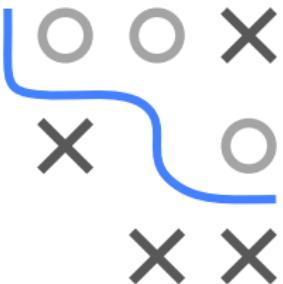
Learning goals

- Know that the SVM is sensitive to hyperparameter choices
- Understand the effect of different (kernel) hyperparameters



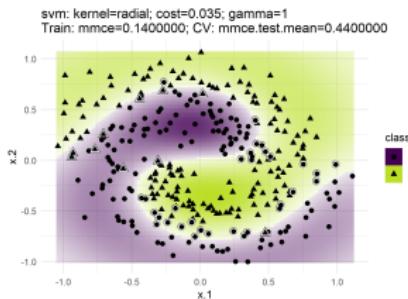
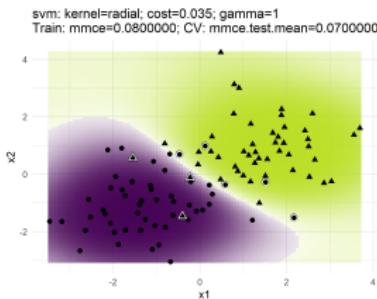
MODEL SELECTION FOR KERNEL SVMS

- “Kernelizing” a linear algorithm effectively turns this algorithm into a family of algorithms — one for each kernel. There are infinitely many kernels, and many efficiently computable kernels.
- However, the choice of C , the choice of the kernel, the kernel parameters are all up to the user.
- On the one hand this allows very flexible modelling, and also to incorporate prior knowledge into the learning process.
- On the other hand this puts a huge burden on the user. The machine has no mechanism for identifying a good kernel by itself.
- SVMs are somewhat sensitive to its hyperparameters and should always be tuned.
- Gaussian processes are very related kernel methods, with the big advantage that kernel parameters are directly estimated during training.

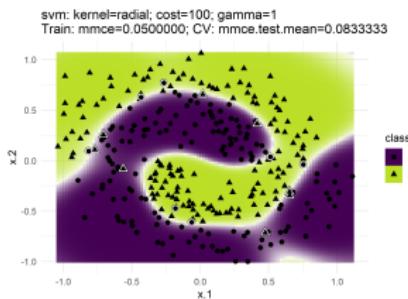
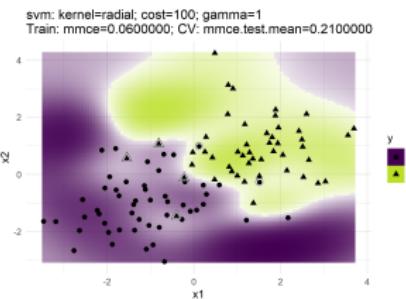


SVM HYPERPARAMETERS

Small C “allows” for margin-violating points in favor of a large margin.



Large C penalizes margin violators, decision boundary is more “wiggly”.

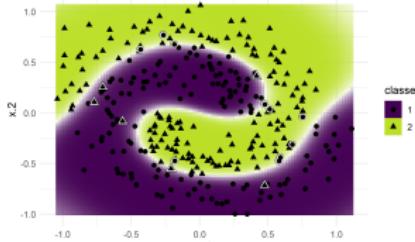


SVM HYPERPARAMETERS

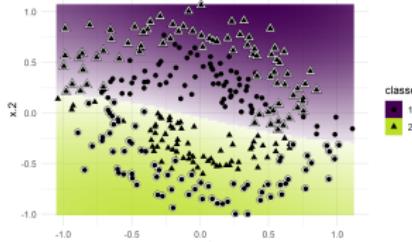
Hyperparameters strongly influence the model: RBF kernel.



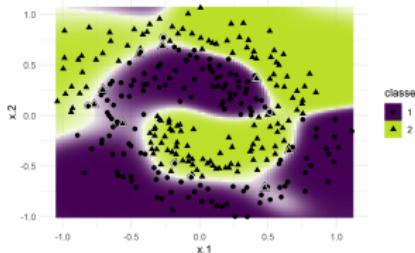
svm: kernel=radial; cost=1; gamma=1
Train: mmce=0.0533333; CV: mmce.test.mean=0.0733333



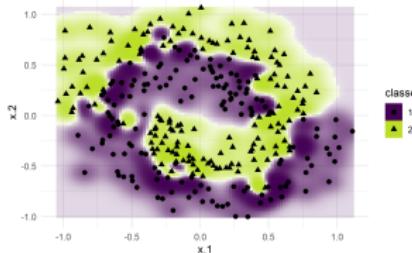
svm: kernel=radial; cost=0.1; gamma=0.1
Train: mmce=0.5066667; CV: mmce.test.mean=0.5266667



svm: kernel=radial; cost=1e+03; gamma=1
Train: mmce=0.0466667; CV: mmce.test.mean=0.0966667



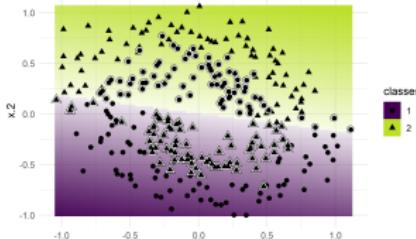
svm: kernel=radial; cost=100; gamma=20
Train: mmce=0.0000000; CV: mmce.test.mean=0.1466667



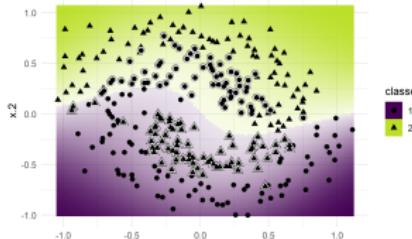
SVM HYPERPARAMETERS

Hyperparameters strongly influence the model: Polynomial kernel.

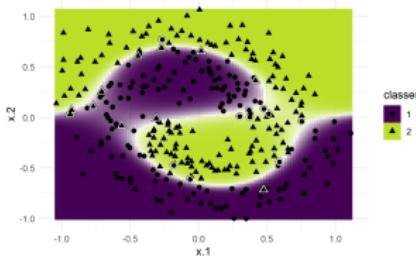
svm: kernel=polynomial; cost=1; coef0=1; degree=1
Train: mmce=0.500000; CV: mmce.test.mean=0.5066667



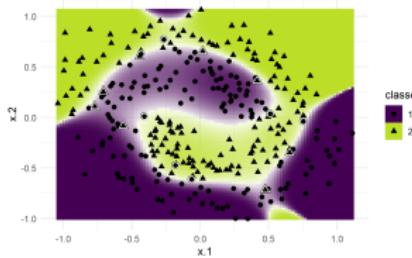
svm: kernel=polynomial; cost=0.01; coef0=1; degree=3
Train: mmce=0.4566667; CV: mmce.test.mean=0.4700000



svm: kernel=polynomial; cost=10; coef0=1; degree=3
Train: mmce=0.0733333; CV: mmce.test.mean=0.1033333

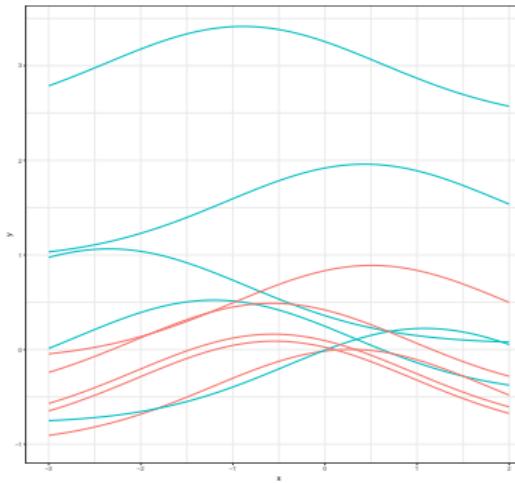
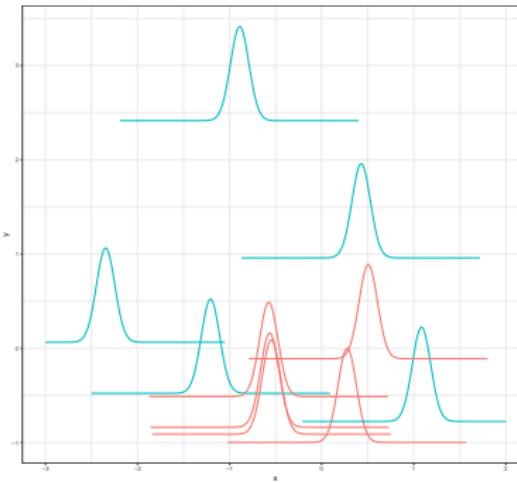
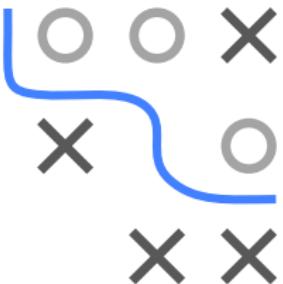


svm: kernel=polynomial; cost=10; coef0=1; degree=7
Train: mmce=0.0500000; CV: mmce.test.mean=0.1066667



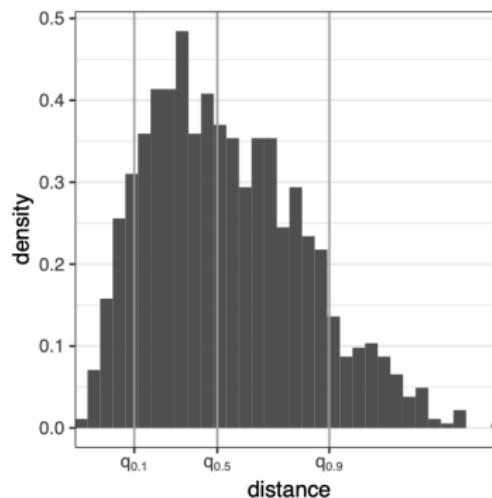
RBF SIGMA HEURISTIC

For the RBF kernel $k(\mathbf{x}, \tilde{\mathbf{x}}) = \exp\left(-\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|^2}{2\sigma^2}\right)$ a simple heuristic exists for the width hyperparameter σ^2 .



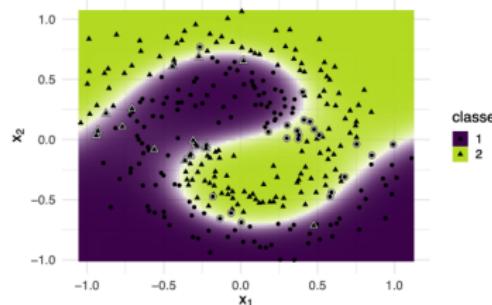
RBF SIGMA HEURISTIC

- Draw a random subset of the data.
- Compute pairwise distances $\|\mathbf{x} - \tilde{\mathbf{x}}\|$.
- Take a "central quantile" from their distribution, e.g., the median.
- This relates the kernel width to the "average distance" between points, which does make intuitive sense.



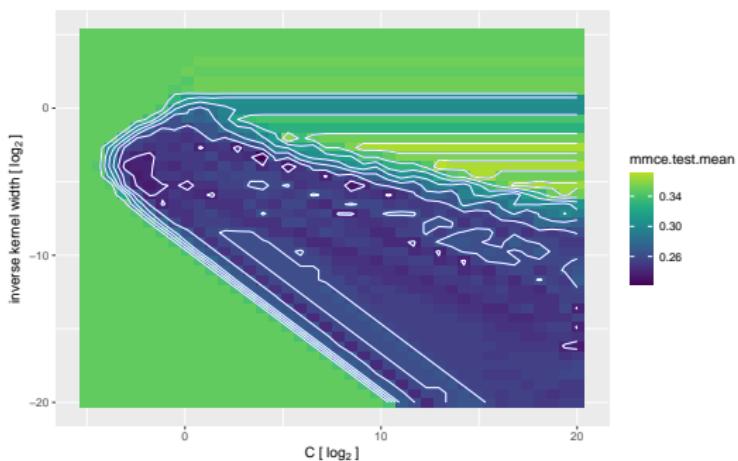
Now γ is set by estimating is inverse σ with the heuristic.

svm: kernel=radial; cost=1; gamma=0.337
Train: mmce=0.133; CV: mmce.test.mean=0.143



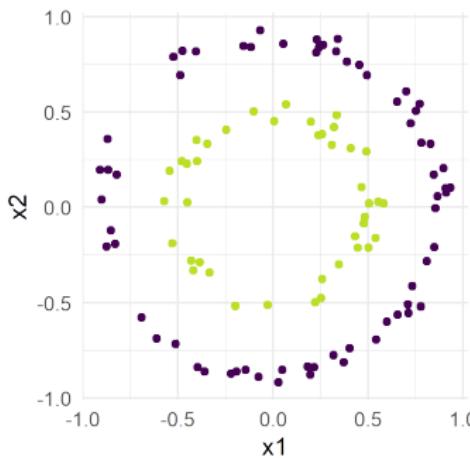
SVM HYPERPARAMETERS

- RBF-SVM parameters are often optimized on log-scale, as we want to explore large values and values close to 0.
- E.g.: $C \in [2^{-15}, 2^{15}]$, $\gamma \in [2^{-15}, 2^{15}]$
- The cross-validated performance landscape often forms a characteristic "ridge" with a larger area of equally good values.



Introduction to Machine Learning

Details on Support Vector Machines



Learning goals

- Know that SVMs are non-parameteric models
- Understand the concept of universal consistency
- Know that SVMs with an universal kernel (e.g. Gaussian kernel) are universally consistent

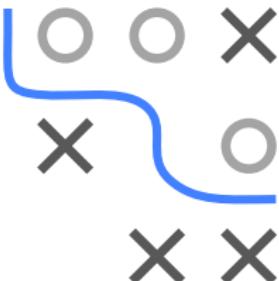


SVMs as Non-Parametric Models



SVMS AS NON-PARAMETRIC MODELS

- In contrast to linear models, for an SVM we do not have to decide the number of coefficients of the decision function before training.
- The number of coefficients depends on the size of the dataset, or on the number of support vectors.
- Such models are called **non-parametric**.
- The big advantage of non-parametric models is that their modeling capacity is not *a priori* restricted to a finite-dimensional subspace of a function space.
- It turns out that SVMs do even better: There exist kernels so that an SVM can model all continuous functions arbitrarily well. It is also known that the SVM learning algorithm can approximate the Bayes optimal decision function arbitrarily well in the limit of infinite data.
- This property is known as **universal consistency**.



SVMS AS NON-PARAMETRIC MODELS

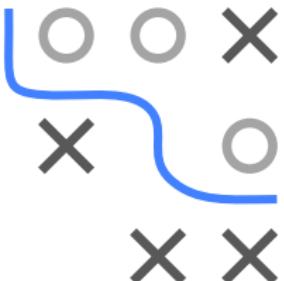
Definition [Steinwart, 2002]: Let $\mathcal{X} \subset \mathbb{R}^p$ be compact. A continuous kernel $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is called universal if the set of all induced functions $\sum_i \beta_i k(\mathbf{x}^{(i)}, \cdot)$ is dense in $\mathcal{C}(\mathcal{X})$; i.e., for all $g \in \mathcal{C}(\mathcal{X})$ and all $\varepsilon > 0$ there exists a function f induced by k with $\|f - g\|_\infty \leq \varepsilon$.

Example: Gaussian kernels are universal.

Theorem [simplified from Steinwart, 2002]: For compact $\mathcal{X} \subset \mathbb{R}^p$ define $C(n) = C_0 \cdot n^{q-1}$ for some $C_0 > 0$ and $0 < q < 1/p$. Fix any distribution \mathbb{P} on $\mathcal{X} \times \{\pm 1\}$ from which i.i.d. datasets \mathcal{D}_n of size n are drawn. Let h_n denote the soft-margin SVM model, trained with a universal kernel and regularization constant $C(n)$ on the data \mathcal{D}_n . Then it holds

$$\lim_{n \rightarrow \infty} \mathbb{E}[\mathcal{R}(h_n)] = \mathcal{R}^* ,$$

where \mathcal{R}^* denotes the Bayes risk.



ASYMPTOTIC PERFORMANCE

- Convergence of the risk to the Bayes risk for all distributions is called **universal consistency**.
- A universally consistent learning machine can solve all problems optimally, provided enough data.
- Parametric models are too inflexible for this property. They can model only a finite-dimensional subspace (manifold) of decision functions.
- Thus, in the limit of infinite data, they will systematically underfit.
- Universal consistency requires more than infinite-dimensional modeling power: We also need a learning rule that uses the flexibility wisely and avoids overfitting.
- The existence of universally consistent learners is one of the most exciting facts from non-parametric statistics.



ASYMPTOTIC PERFORMANCE

- Note the arbitrary positive constant C_0 in the definition of $C(n) = C_0 \cdot n^{q-1}$.
- This means that for a single fixed n , $C(n)$ can have any positive value.
- This is not a problem for the theorem since all it requires is that C changes at the right rate with n :
 - $n \cdot C(n)$ tends to infinity, which means that the relative impact of the regularizer compared to the empirical risk decays to zero, so, the risk term takes over for large n ;
 - The convergence of $n \cdot C(n)$ to infinity is slow enough to avoid overfitting (this is far from obvious, but it is in the details of the proof of the theorem).
- Importantly, since C can be arbitrary for fixed n , this theorem does not tell us which C to use for a given problem size.

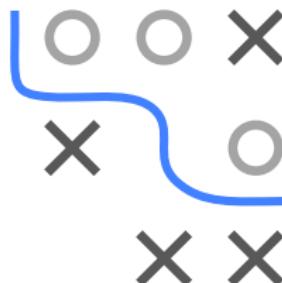


Kernels on Infinite-Dimensional Vector Spaces



KERNELS ON INFINITE-DIMENSIONAL VECTOR SPACES

- Note that the input space \mathcal{X} does not need to be a finite-dimensional vector space.
- \mathcal{X} could be the set of all character strings (of unlimited length) or of graphs, or of trees.
- Such data structures are natural representations for, e.g., HTML documents.
- There are many examples of data that do not naturally come in vector form.
- Most often meaningful and cheap-to-compute kernels can be defined directly on the input data structures – they simply define a similarity measure over these data.
- SVMs (and other kernel methods) allow to learn and predict directly on these spaces.



INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

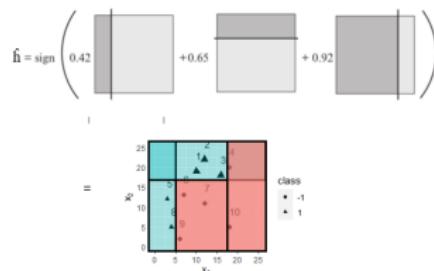
Boosting

Gaussian Processes



Introduction to Machine Learning

Gradient Boosting: Introduction and AdaBoost

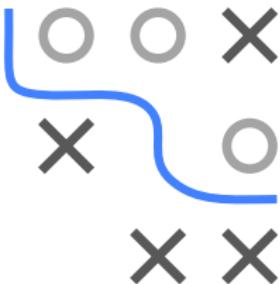


Learning goals

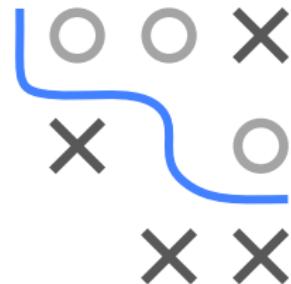
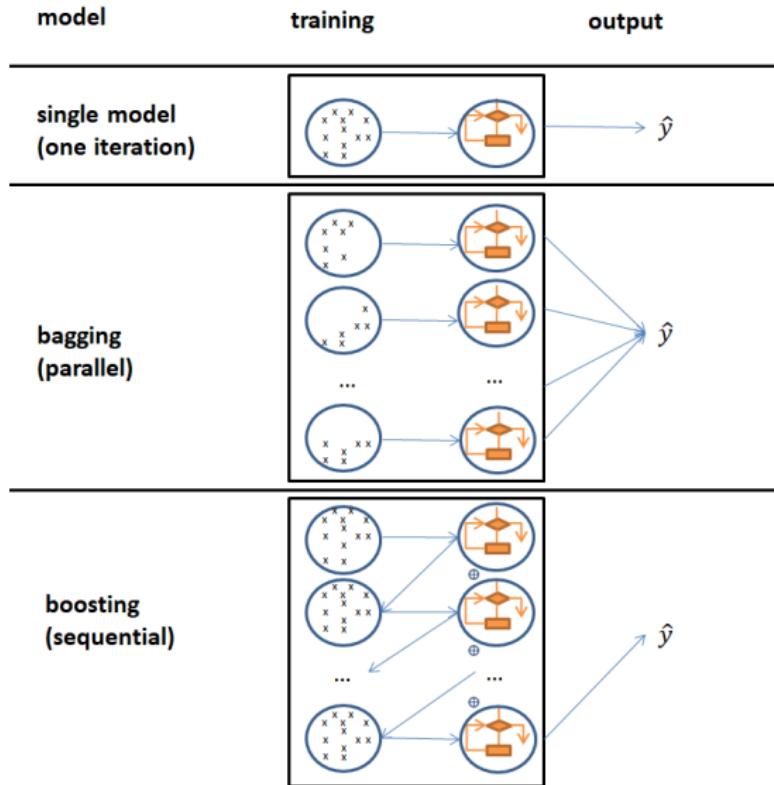
- Understand general idea of boosting
- Learn AdaBoost algorithm
- Understand difference between bagging and boosting

INTRODUCTION TO BOOSTING

- Boosting is considered to be one of the most powerful learning ideas within the last twenty years.
- Originally designed for classification, (especially gradient) boosting handles regression (and many other supervised tasks) naturally nowadays.
- Homogeneous ensemble method (like bagging), but fundamentally different approach.
- **Idea:** Take a weak classifier and sequentially apply it to modified versions of the training data.
- We will begin by describing an older, simpler boosting algorithm designed for binary classification, the popular “AdaBoost”.



BOOSTING VS. BAGGING



THE BOOSTING QUESTION

The first boosting algorithm ever was in fact no algorithm for practical purposes, but the solution for a theoretical problem:

“Does the existence of a weak learner for a certain problem imply the existence of a strong learner?” ▶ Kearns, 1988



- **Weak learners** are defined as a prediction rule with a correct classification rate that is at least slightly better than random guessing (> 50% accuracy on a balanced binary problem).
- We call a learner a **strong learner** “if there exists a polynomial-time algorithm that achieves low error with high confidence for all concepts in the class” ▶ Schapire, 1990 .

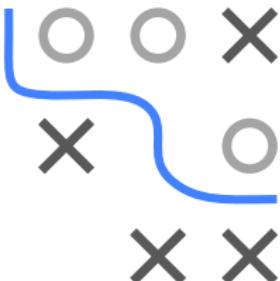
In practice it is typically easy to construct weak learners, but difficult to build a strong one.

THE BOOSTING ANSWER - ADABOOST

Any weak (base) learner can be iteratively boosted to become a strong learner. The proof of this ground-breaking idea generated the first boosting algorithm.

- The **AdaBoost** (Adaptive Boosting) algorithm is a **boosting** method for binary classification by ▶ Freund and Schapire (1996).
- The base learner is sequentially applied to weighted training observations.
- After each base learner fit, currently misclassified observations receive a higher weight for the next iteration, so we focus more on instances that are harder to classify.

Leo Breiman (referring to the success of AdaBoost):
“Boosting is the best off-the-shelf classifier in the world.”

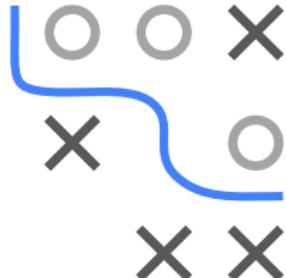


THE BOOSTING ANSWER - ADABOOST

- Assume a target variable y encoded as $\{-1, +1\}$, and weak base learners (e.g., tree stumps) from a hypothesis space \mathcal{B} .
- Base learner models $b^{[m]}$ are binary classifiers that map to $\mathcal{Y} = \{-1, +1\}$. We might sometimes write $b(\mathbf{x}, \theta^{[m]})$ instead.
- Predictions from all base models $b^{[m]}$ over M iterations are combined in an additive manner by the formula:

$$f(\mathbf{x}) = \sum_{m=1}^M \beta^{[m]} b^{[m]}(\mathbf{x}).$$

- Weights $\beta^{[m]}$ are computed by the boosting algorithm. Their purpose is to give higher weights to base learners with higher predictive accuracy.
- The number of iterations M is the main tuning parameter.
- The discrete prediction function is $h(\mathbf{x}) = \text{sign}(f(\mathbf{x})) \in \{-1, +1\}$.



THE BOOSTING ANSWER - ADABOOST

Algorithm AdaBoost

- 1: Initialize observation weights: $w^{[1](i)} = \frac{1}{n} \quad \forall i \in \{1, \dots, n\}$
- 2: **for** $m = 1 \rightarrow M$ **do**
- 3: Fit classifier to training data with weights $w^{[m]}$ and get hard label classifier $\hat{b}^{[m]}$
- 4: Calculate weighted in-sample misclassification rate

$$\text{err}^{[m]} = \sum_{i=1}^n w^{[m](i)} \cdot \mathbb{1}_{\{y^{(i)} \neq \hat{b}^{[m]}(\mathbf{x}^{(i)})\}}$$

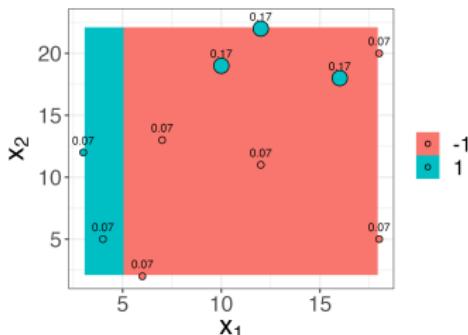
- 5: Compute: $\hat{\beta}^{[m]} = \frac{1}{2} \log \left(\frac{1-\text{err}^{[m]}}{\text{err}^{[m]}} \right)$
- 6: Set: $w^{[m+1](i)} = w^{[m](i)} \cdot \exp \left(-\hat{\beta}^{[m]} \cdot y^{(i)} \cdot \hat{b}^{[m]}(\mathbf{x}^{(i)}) \right)$
- 7: Normalize $w^{[m+1](i)}$ such that $\sum_{i=1}^n w^{[m+1](i)} = 1$
- 8: **end for**
- 9: Output: $\hat{f}(\mathbf{x}) = \sum_{m=1}^M \hat{\beta}^{[m]} \hat{b}^{[m]}(\mathbf{x})$



ADABOOST ILLUSTRATION

Example description

- $n = 10$ observations and two features x_1 and x_2
- Tree stumps as base learners $b^{[m]}(\mathbf{x})$
- Balanced classification task with y encoded as $\{-1, +1\}$
- $M = 3$ iterations \Rightarrow initial weights $w^{[1](i)} = \frac{1}{10} \quad \forall i \in 1, \dots, 10.$



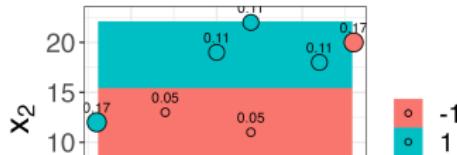
New observation weights:

- Prediction correct:
 $w^{[2](i)} = w^{[1](i)} \cdot \exp(-\hat{\beta}^{[1]} \cdot 1)$
 $\approx 0.065.$
- For 3 misclassified observations:
 $w^{[2](i)} = w^{[1](i)} \cdot \exp(-\hat{\beta}^{[1]} \cdot (-1))$
 $\approx 0.15.$
- After normalization:
 - correctly classified: $w^{[2](i)} \approx 0.07$
 - misclassified: $w^{[2](i)} \approx 0.17$

Iteration $m = 1$:

- $\text{err}^{[1]} = 0.3$
- $\hat{\beta}^{[1]} = \frac{1}{2} \log\left(\frac{1-0.3}{0.3}\right) \approx 0.42$

ADABOOST ILLUSTRATION

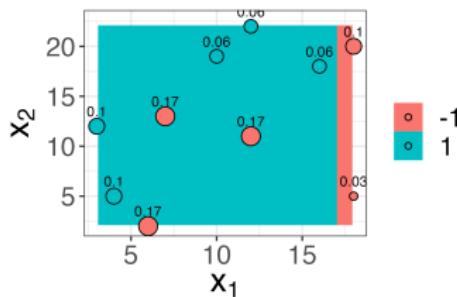


Iteration $m = 2$:

- $\text{err}^{[2]} \approx 3 \cdot 0.07 = 0.21$
- $\hat{\beta}^{[2]} \approx 0.65$

New observation weights (before normalization):

- For misclassified observations: $w^{[3](i)} = w^{[2](i)} \cdot \exp(-\hat{\beta}^{[2]} \cdot (-1)) \approx w^{[2](i)} \cdot 1.92$
- For correctly classified observations: $w^{[3](i)} = w^{[2](i)} \cdot \exp(-\hat{\beta}^{[2]} \cdot 1) \approx w^{[2](i)} \cdot 0.52$



Iteration $m = 3$:

- $\text{err}^{[3]} \approx 3 \cdot 0.05 = 0.15$
- $\hat{\beta}^{[3]} \approx 0.92$

Note: the smaller the error rate of a base learner, the larger the weight, e.g.,

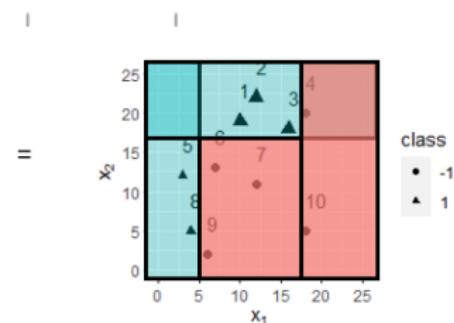
$\text{err}^{[3]} \approx 0.15 < \text{err}^{[1]} \approx 0.3$ and $\hat{\beta}^{[3]} \approx 0.92 > \hat{\beta}^{[1]} \approx 0.42$.



ADABOOST ILLUSTRATION

With $\hat{f}(\mathbf{x}) = \sum_{m=1}^M \hat{\beta}^{[m]} \hat{b}^{[m]}(\mathbf{x})$ and $h(\mathbf{x}) = \text{sign}(f(\mathbf{x})) \in \{-1, +1\}$, we get:

$$\hat{h} = \text{sign} \left(0.42 \begin{array}{|c|c|} \hline & | \\ \hline \end{array} + 0.65 \begin{array}{|c|c|} \hline & | \\ \hline \end{array} + 0.92 \begin{array}{|c|c|} \hline & | \\ \hline \end{array} \right)$$



Hence, when all three base classifiers are combined, all samples are classified correctly.

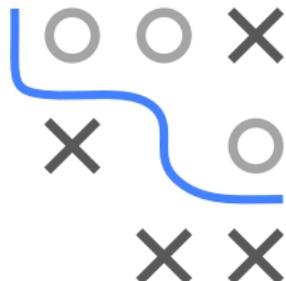
BAGGING VS BOOSTING

Random forest

- Base learners are typically deeper decision trees (not only stumps!)
- Equal weights for base learners
- Base learners independent of each other
- Aim: variance reduction
- Tends **not** to overfit

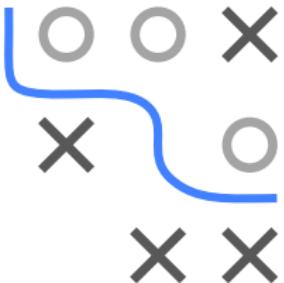
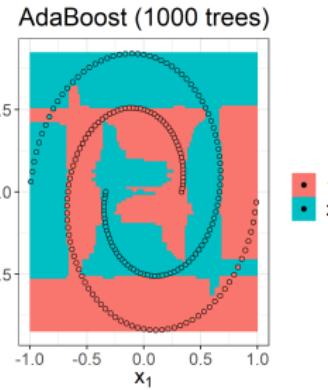
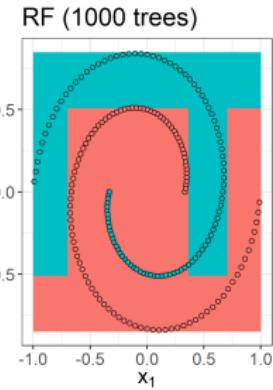
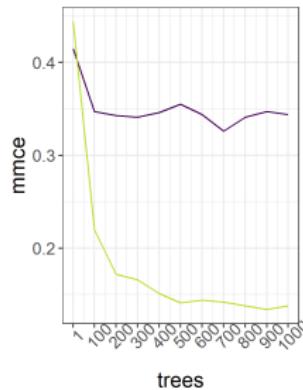
AdaBoost

- Base learners are weak learners, e.g., only stumps
- Base learners have different weights depending on their predictive accuracy
- Sequential algorithm, hence order matters
- Aim: bias and variance reduction
- Tends to overfit



BAGGING VS BOOSTING STUMPS

Random forest versus AdaBoost (both with stumps) on Spirals data from `mlbench` ($n = 200$, $sd = 0$), with 5×5 repeated CV.

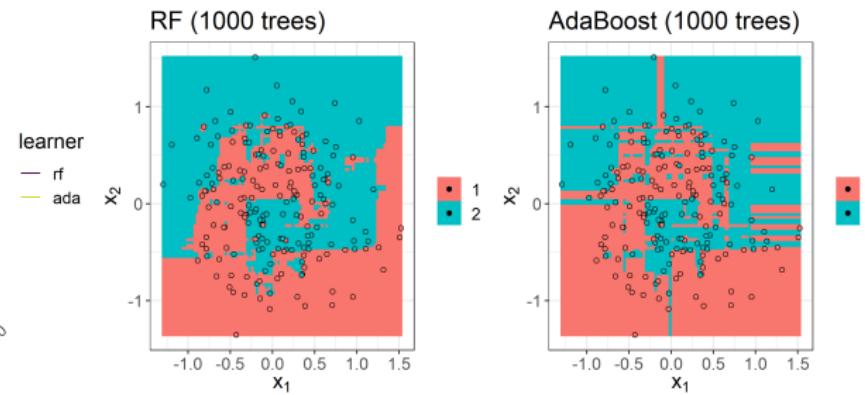
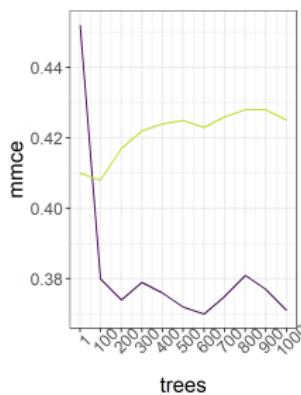
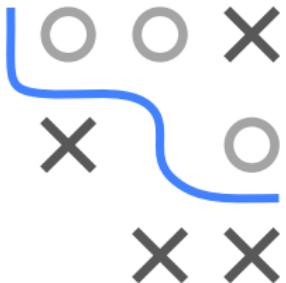


Weak learners do not work well with bagging as only variance, but no bias reduction happens.

OVERFITTING BEHAVIOR

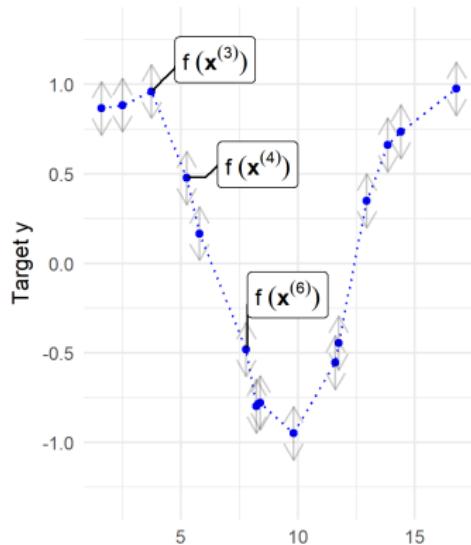
Historically, the overfitting behavior of AdaBoost was often discussed.

Increasing standard deviation to $sd = 0.3$ and allowing for more flexibility in the base learners, AdaBoost overfits with increasing number of trees while the RF only saturates. The overfitting of AdaBoost here is quite typical as data is very noisy.



Introduction to Machine Learning

Gradient Boosting: Concept



Learning goals

- Understand idea of forward stagewise modelling
- Understand fitting process of gradient boosting for regression problems

FORWARD STAGEWISE ADDITIVE MODELING

Assume a regression problem for now (as this is simpler to explain);
and assume a space of base learners \mathcal{B} .

We want to learn an additive model:

$$f(\mathbf{x}) = \sum_{m=1}^M \alpha^{[m]} b(\mathbf{x}, \boldsymbol{\theta}^{[m]}).$$

Hence, we minimize the empirical risk:

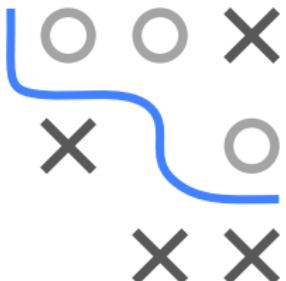
$$\mathcal{R}_{\text{emp}}(f) = \sum_{i=1}^n L \left(y^{(i)}, f \left(\mathbf{x}^{(i)} \right) \right) = \sum_{i=1}^n L \left(y^{(i)}, \sum_{m=1}^M \alpha^{[m]} b(\mathbf{x}^{(i)}, \boldsymbol{\theta}^{[m]}) \right)$$



FORWARD STAGEWISE ADDITIVE MODELING

Why is gradient boosting a good choice for this problem?

- Because of the additive structure it is difficult to jointly minimize $\mathcal{R}_{\text{emp}}(f)$ w.r.t. $((\alpha^{[1]}, \theta^{[1]}), \dots, (\alpha^{[M]}, \theta^{[M]}))$, which is a very high-dimensional parameter space (though this is less of a problem nowadays, especially in the case of numeric parameter spaces).
- Considering trees as base learners is worse as we would have to grow M trees in parallel so they work optimally together as an ensemble.
- Stagewise additive modeling has nice properties, which we want to make use of, e.g. for regularization, early stopping, ...



FORWARD STAGewise ADDITIVE MODELING

Hence, we add additive components in a greedy fashion by sequentially minimizing the risk only w.r.t. the next additive component:

$$\min_{\alpha, \theta} \sum_{i=1}^n L \left(y^{(i)}, \hat{f}^{[m-1]} \left(\mathbf{x}^{(i)} \right) + \alpha b \left(\mathbf{x}^{(i)}, \theta \right) \right)$$



Doing this iteratively is called **forward stagewise additive modeling**.

Algorithm Forward Stagewise Additive Modeling.

- 1: Initialize $\hat{f}^{[0]}(\mathbf{x})$ with loss optimal constant model
 - 2: **for** $m = 1 \rightarrow M$ **do**
 - 3: $(\alpha^{[m]}, \hat{\theta}^{[m]}) = \arg \min_{\alpha, \theta} \sum_{i=1}^n L \left(y^{(i)}, \hat{f}^{[m-1]} \left(\mathbf{x}^{(i)} \right) + \alpha b \left(\mathbf{x}^{(i)}, \theta \right) \right)$
 - 4: Update $\hat{f}^{[m]}(\mathbf{x}) \leftarrow \hat{f}^{[m-1]}(\mathbf{x}) + \alpha^{[m]} b \left(\mathbf{x}, \hat{\theta}^{[m]} \right)$
 - 5: **end for**
-

GRADIENT BOOSTING

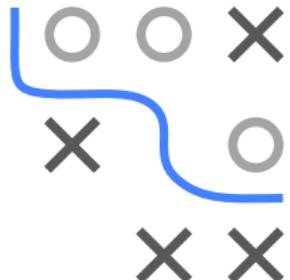
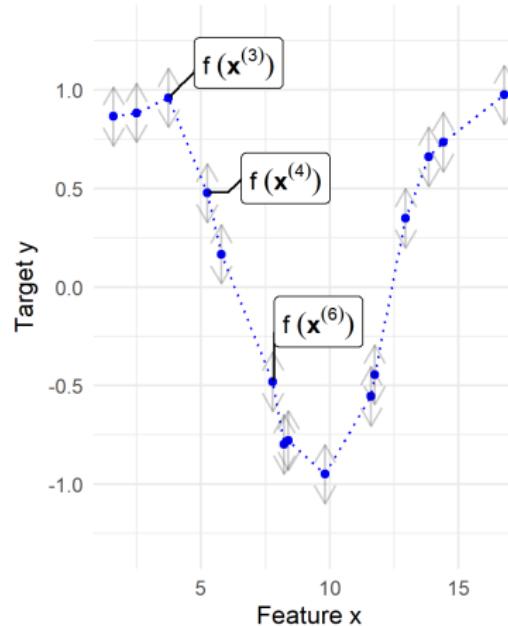
The algorithm we just introduced is not really an algorithm, but rather an abstract principle. We need to find the new additive component $b(\mathbf{x}, \theta^{[m]})$ and its weight coefficient $\alpha^{[m]}$ in each iteration m . This can be done by gradient descent, but in function space.

Thought experiment: Consider a completely non-parametric model f whose predictions we can arbitrarily define on every point of the training data $\mathbf{x}^{(i)}$. So we basically specify f as a discrete, finite vector.

$$\left(f\left(\mathbf{x}^{(1)}\right), \dots, f\left(\mathbf{x}^{(n)}\right) \right)^T$$

This implies n parameters $f\left(\mathbf{x}^{(i)}\right)$ (and the model would provide no generalization...).

Furthermore, we assume our loss function $L(\cdot)$ to be differentiable.

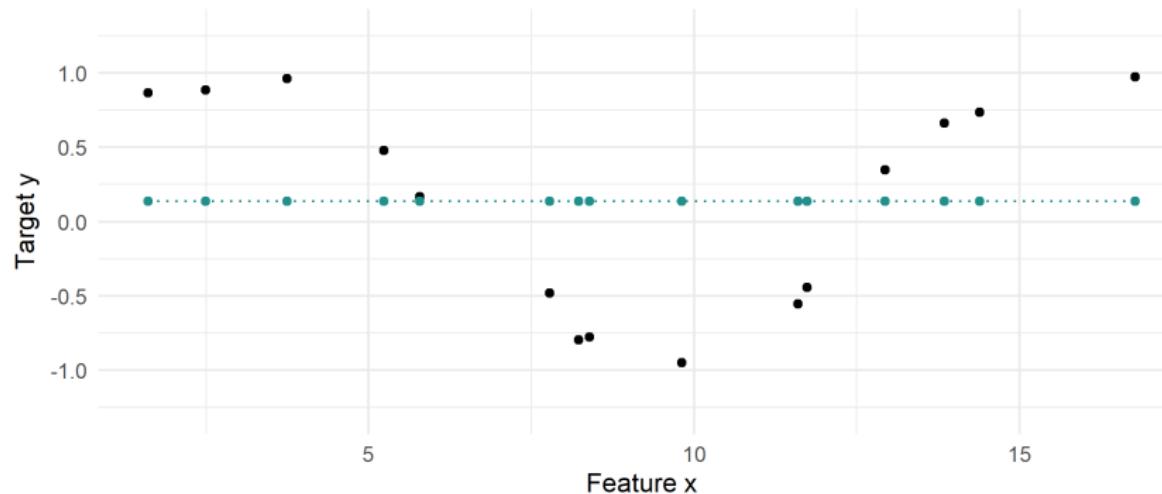
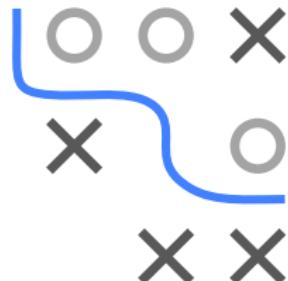


GRADIENT BOOSTING

Aim: Define a movement in function space so we can push our current function towards the data points.

Given: Regression problem with one feature x and target variable y .

Initialization: Set all parameters to the optimal constant value (e.g., the mean of y for $L2$).



PSEUDO RESIDUALS

How do we have to distort this function to move it towards the observations and drive loss down?

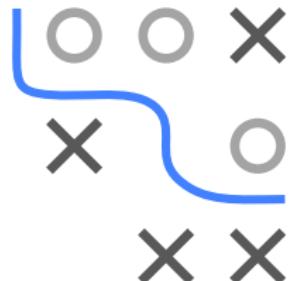
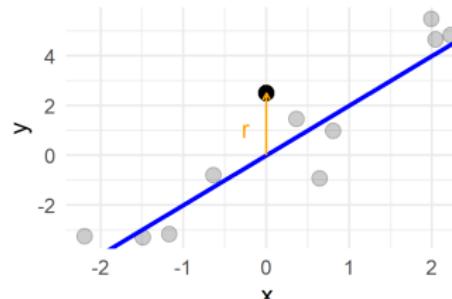
We minimize the risk of such a model with gradient descent (yes, this makes no sense, suspend all doubts for a few seconds).

So, we calculate the gradient at a point of the parameter space, that is, the derivative w.r.t. each component of the parameter vector (which is 0 for all terms with $i \neq j$):

$$\tilde{r}^{(i)} = -\frac{\partial \mathcal{R}_{\text{emp}}}{\partial f(\mathbf{x}^{(i)})} = -\frac{\partial \sum_j L(y^{(j)}, f(\mathbf{x}^{(j)}))}{\partial f(\mathbf{x}^{(i)})} = -\frac{\partial L(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f(\mathbf{x}^{(i)})}.$$

Reminder: The pseudo-residuals $\tilde{r}(f)$ match the usual residuals for the squared loss:

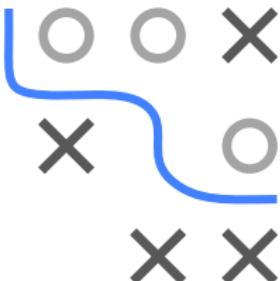
$$-\frac{\partial L(y, f(\mathbf{x}))}{\partial f(\mathbf{x})} = -\frac{\partial 0.5(y - f(\mathbf{x}))^2}{\partial f(\mathbf{x})} = y - f(\mathbf{x})$$



BOOSTING AS GRADIENT DESCENT

Combining this with the iterative additive procedure of “forward stagewise modeling”, we are at the spot $f^{[m-1]}$ during minimization. At this point, we now calculate the direction of the negative gradient or also called pseudo-residuals $\tilde{r}^{[m](i)}$:

$$\tilde{r}^{[m](i)} = - \left[\frac{\partial L(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f(\mathbf{x}^{(i)})} \right]_{f=f^{[m-1]}}$$



The gradient descent update for each vector component of f is:

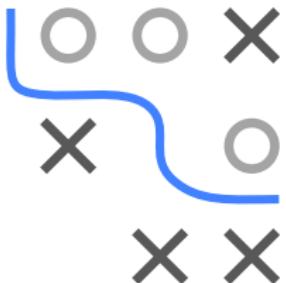
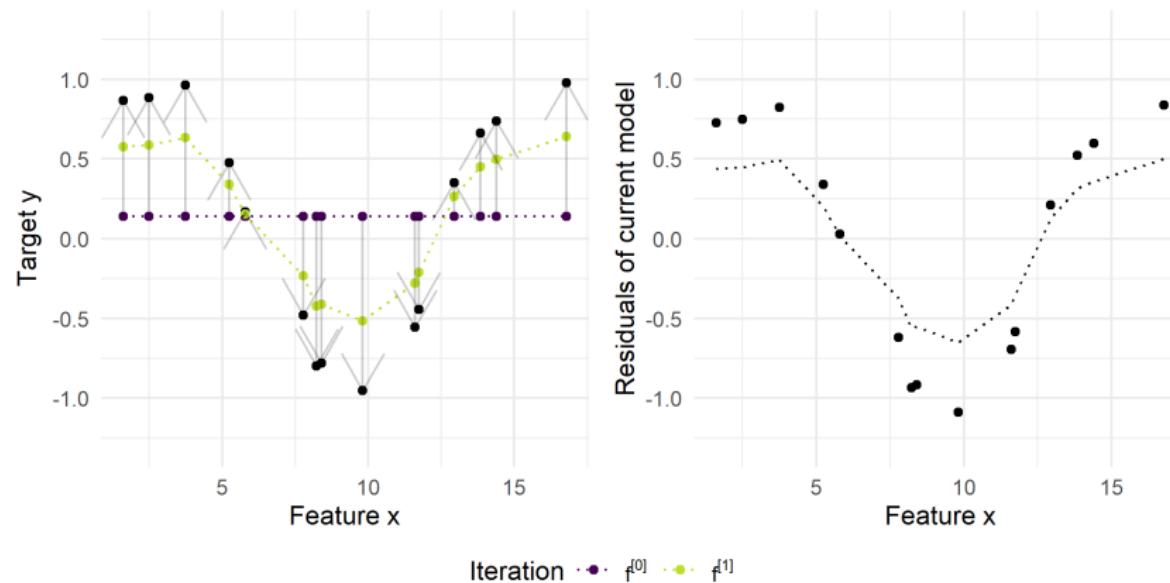
$$f^{[m]}(\mathbf{x}^{(i)}) = f^{[m-1]}(\mathbf{x}^{(i)}) - \alpha \frac{\partial L(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f^{[m-1]}(\mathbf{x}^{(i)})}.$$

This tells us how we could “nudge” our whole function f in the direction of the data to reduce its empirical risk.

GRADIENT BOOSTING

Iteration 1:

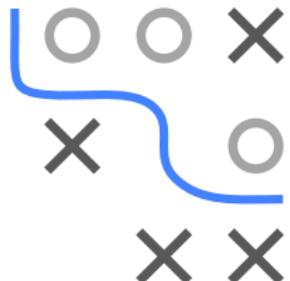
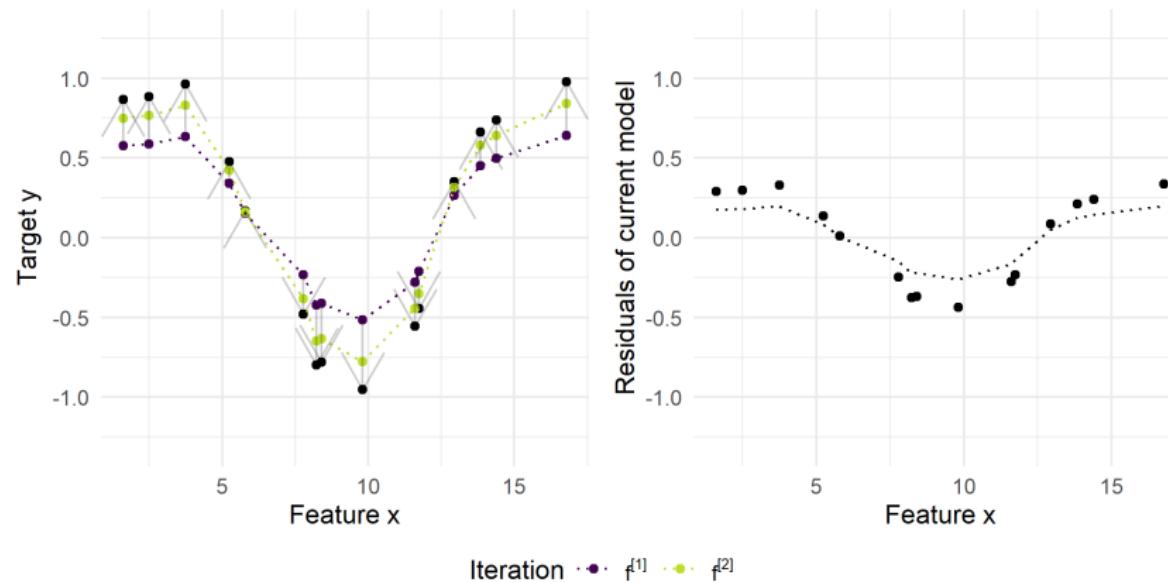
Let's move our function $f(\mathbf{x}^{(i)})$ a fraction towards the pseudo-residuals with a learning rate of $\alpha = 0.6$.



GRADIENT BOOSTING

Iteration 2:

Let's move our function $f(\mathbf{x}^{(i)})$ a fraction towards the pseudo-residuals with a learning rate of $\alpha = 0.6$.



GRADIENT BOOSTING

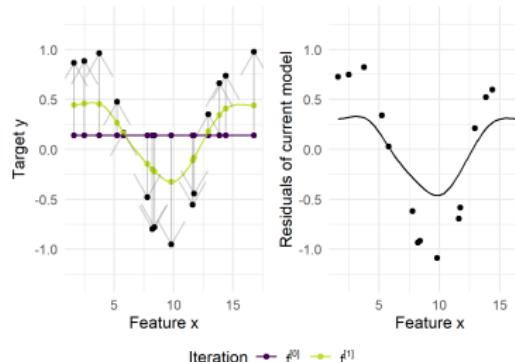
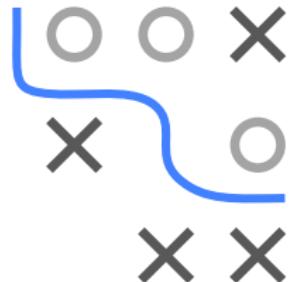
To parameterize a model in this way is pointless, as it just memorizes the instances of the training data.

So, we restrict our additive components to $b(\mathbf{x}, \theta^{[m]}) \in \mathcal{B}$.

The pseudo-residuals are calculated exactly as stated above, then we fit a simple model $b(\mathbf{x}, \theta^{[m]})$ to them:

$$\hat{\theta}^{[m]} = \arg \min_{\theta} \sum_{i=1}^n \left(\tilde{r}^{[m](i)} - b(\mathbf{x}^{(i)}, \theta) \right)^2.$$

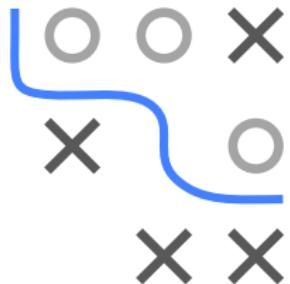
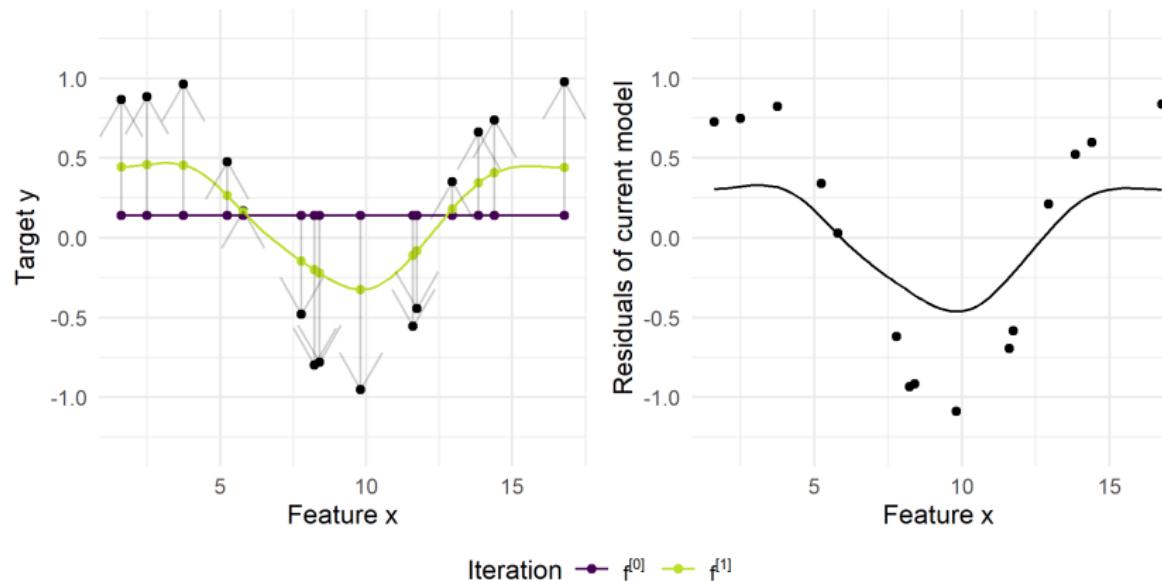
So, evaluated on the training data, our $b(\mathbf{x}, \theta^{[m]})$ corresponds as closely as possible to the negative loss function gradient and generalizes over the whole space.



GRADIENT BOOSTING

In a nutshell: One boosting iteration is exactly one approximated gradient descent step in function space, which minimizes the empirical risk as much as possible.

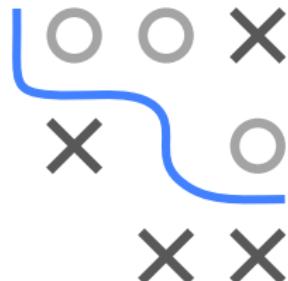
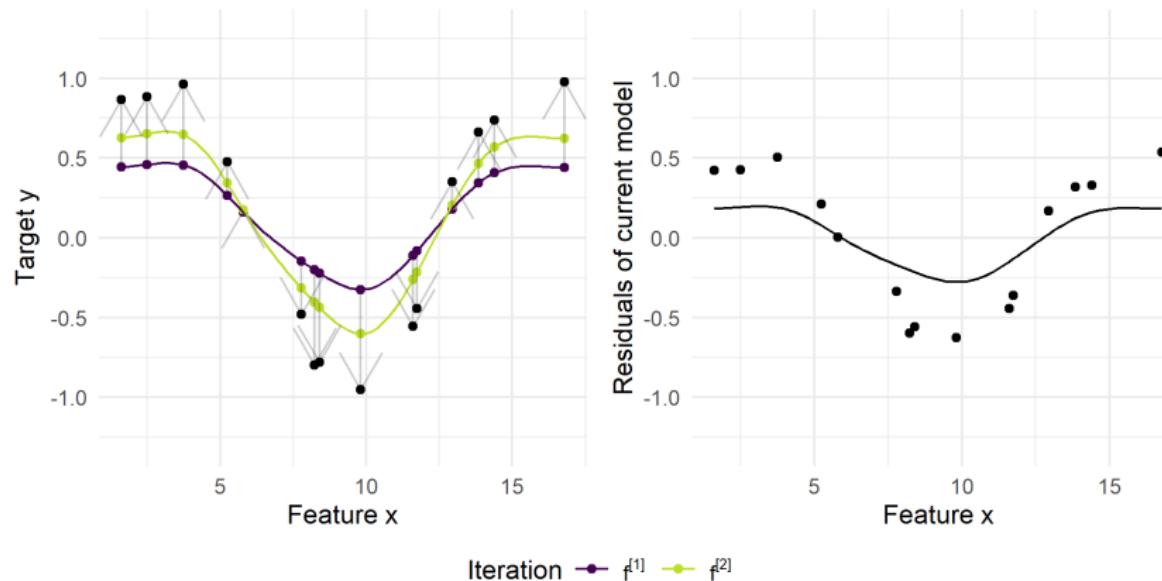
Iteration 1:



GRADIENT BOOSTING

Instead of moving the function values for each observation by a fraction closer to the observed data, we fit a regression base learner to the pseudo-residuals (right plot).

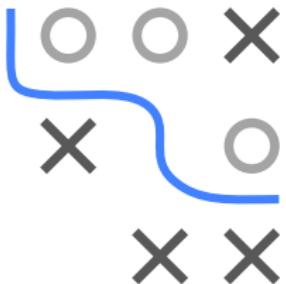
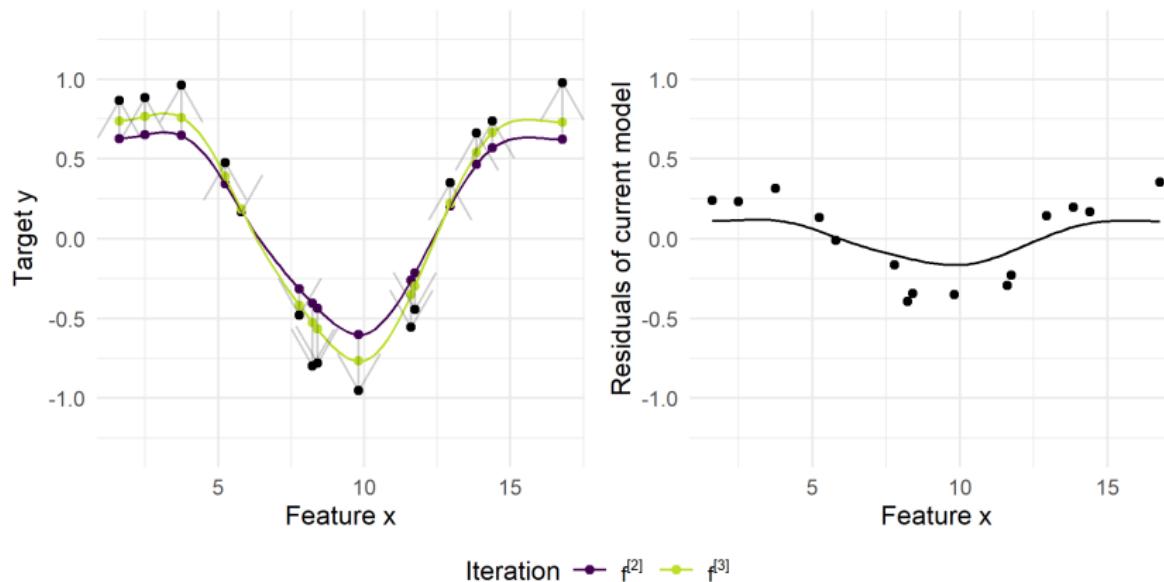
Iteration 2:



GRADIENT BOOSTING

This base learner is then added to the current state of the ensemble weighted by the learning rate (here: $\alpha = 0.4$) and for the next iteration again the pseudo-residuals of the adapted ensemble are calculated and a base learner is fitted to them.

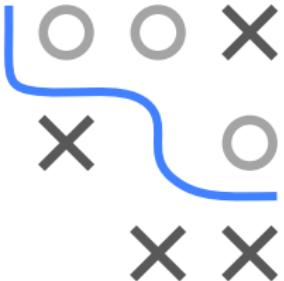
Iteration 3:



GRADIENT BOOSTING ALGORITHM

Algorithm Gradient Boosting Algorithm.

- 1: Initialize $\hat{f}^{[0]}(\mathbf{x}) = \arg \min_{\theta_0 \in \mathbb{R}} \sum_{i=1}^n L(y^{(i)}, \theta_0)$
 - 2: **for** $m = 1 \rightarrow M$ **do**
 - 3: For all i : $\tilde{r}^{[m](i)} = - \left[\frac{\partial L(y, f)}{\partial f} \right]_{f=\hat{f}^{[m-1]}(\mathbf{x}^{(i)}), y=y^{(i)}}$
 - 4: Fit a regression base learner to the vector of pseudo-residuals $\tilde{r}^{[m]}$:
 - 5: $\hat{\theta}^{[m]} = \arg \min_{\theta} \sum_{i=1}^n (\tilde{r}^{[m](i)} - b(\mathbf{x}^{(i)}, \theta))^2$
 - 6: Set $\alpha^{[m]}$ to α being a small constant value or via line search
 - 7: Update $\hat{f}^{[m]}(\mathbf{x}) = \hat{f}^{[m-1]}(\mathbf{x}) + \alpha^{[m]} b(\mathbf{x}, \hat{\theta}^{[m]})$
 - 8: **end for**
 - 9: Output $\hat{f}(\mathbf{x}) = \hat{f}^{[M]}(\mathbf{x})$
-



Note that we also initialize the model in a loss-optimal manner.

LINE SEARCH

The learning rate in gradient boosting influences how fast the algorithm converges. Although a small constant learning rate is commonly used in practice, it can also be replaced by a line search.

Line search is an iterative approach to find a local minimum. In the case of setting the learning rate, the following one-dimensional optimization problem has to be solved:

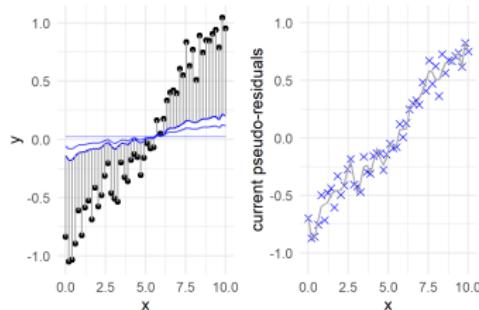
$$\hat{\alpha}^{[m]} = \arg \min_{\alpha} \sum_{i=1}^n L(y^{(i)}, f^{[m-1]}(\mathbf{x}) + \alpha b(\mathbf{x}, \hat{\theta}^{[m]}))$$

Optionally, an (inexact) backtracking line search can be used to find the $\alpha^{[m]}$ that minimizes the above equation.



Introduction to Machine Learning

Gradient Boosting: Illustration



Learning goals

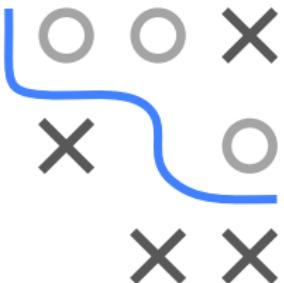
- See simple visualizations of boosting in regression
- Understand impact of different losses and base learners



GRADIENT BOOSTING ILLUSTRATION - GAM

GAM / Splines as BL and compare L_2 vs. L_1 loss.

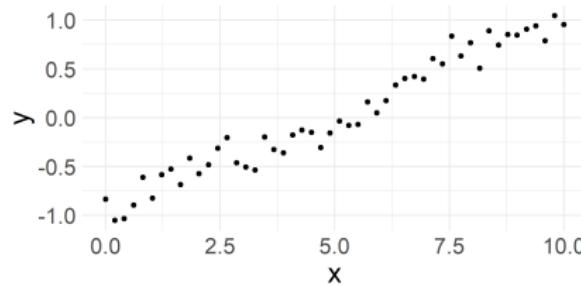
- L2: Init = optimal constant = $\text{mean}(y)$; for L1 it's $\text{median}(y)$
- BLs are cubic B -splines with 40 knots.
- PRs L_2 : $\tilde{r}(f) = r(f) = y - f(\mathbf{x})$
- PRs L_1 : $\tilde{r}(f) = \text{sign}(y - f(\mathbf{x}))$
- Constant learning rate 0.2



Univariate toy data:

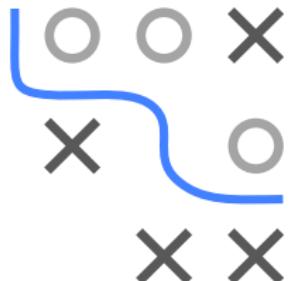
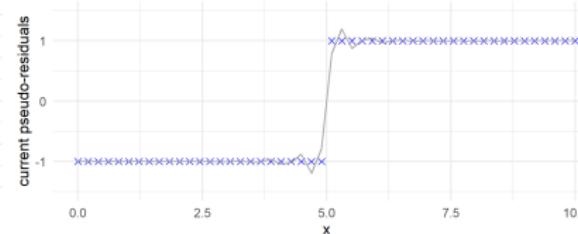
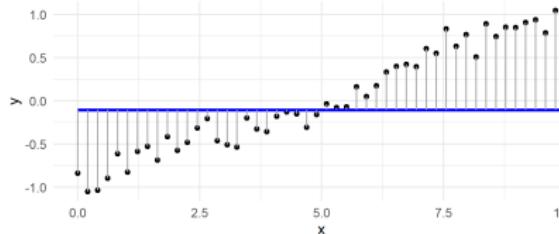
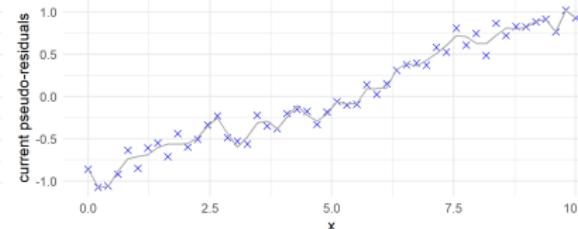
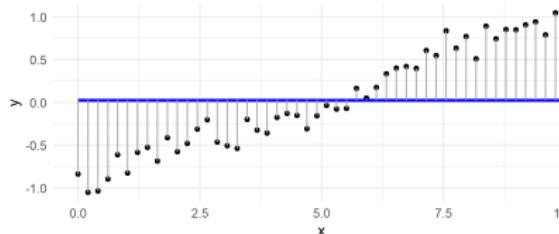
$$y^{(i)} = -1 + 0.2 \cdot x^{(i)} + 0.1 \cdot \sin(x^{(i)}) + \epsilon^{(i)}$$

$$n = 50 ; \epsilon^{(i)} \sim \mathcal{N}(0, 0.1)$$



GAM WITH L_2 VS L_1 LOSS

Top: L_2 loss, bottom: L_1 loss

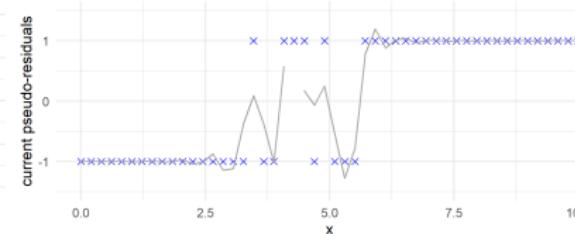
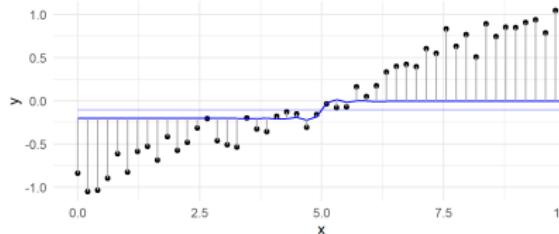
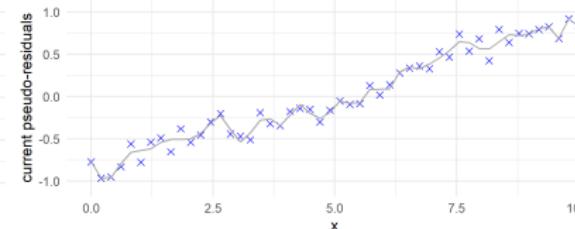
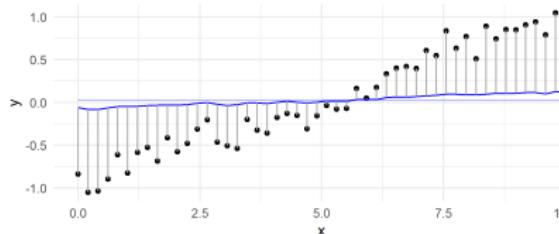


Iteration 1

Shape of PRs affects gradual model fit: L_1 only sees resids' sign, BLs are not affected size of values as in L_2 and hence lead to more moderate changes.

GAM WITH L_2 VS L_1 LOSS

Top: L_2 loss, bottom: L_1 loss

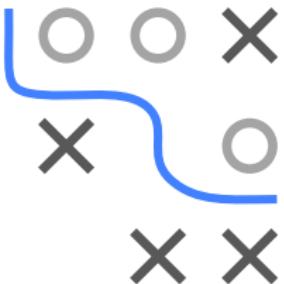
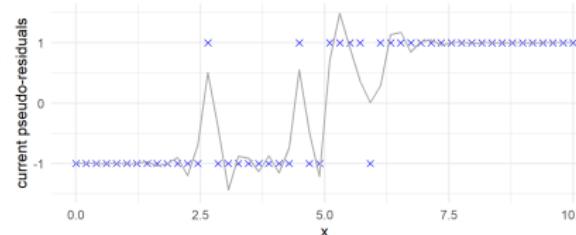
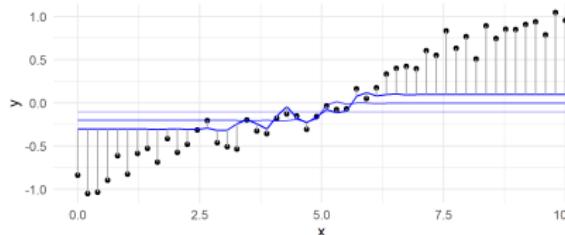
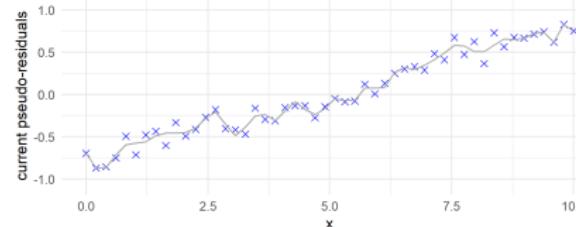
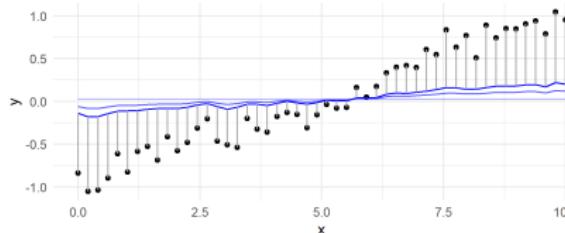


Iteration 2

Shape of PRs affects gradual model fit: L_1 only sees resid's sign, BLs are not affected size of values as in L_2 and hence lead to more moderate changes.

GAM WITH L_2 VS L_1 LOSS

Top: L_2 loss, bottom: L_1 loss

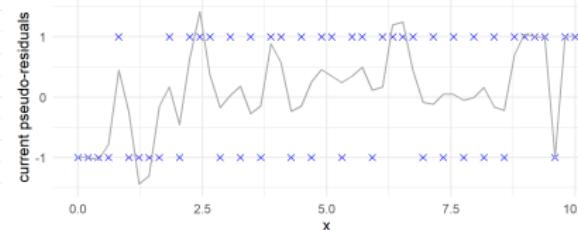
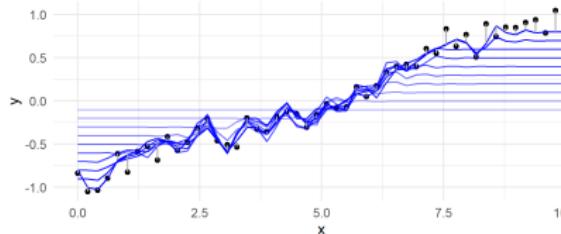
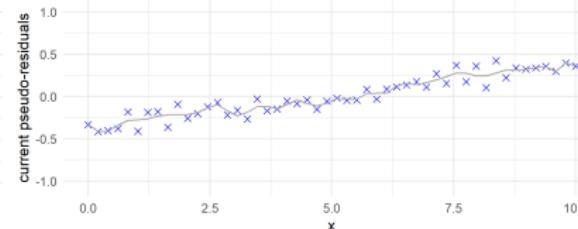
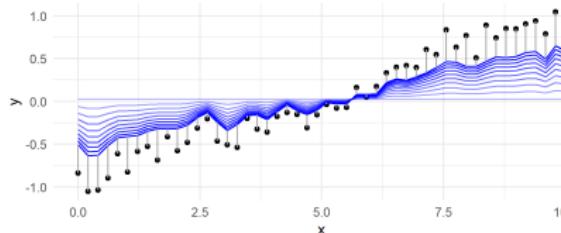


Iteration 3

Shape of PRs affects gradual model fit: L_1 only sees resids' sign, BLs are not affected size of values as in L_2 and hence lead to more moderate changes.

GAM WITH L_2 VS L_1 LOSS

Top: L_2 loss, bottom: L_1 loss

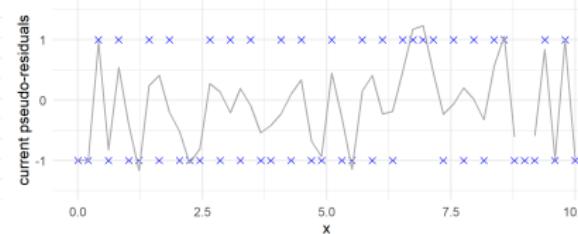
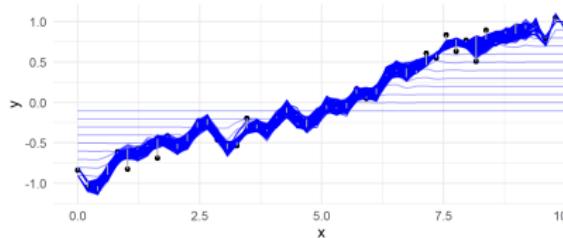
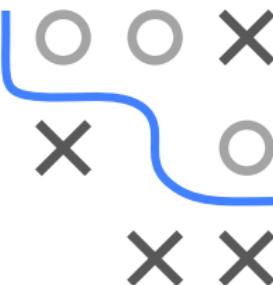
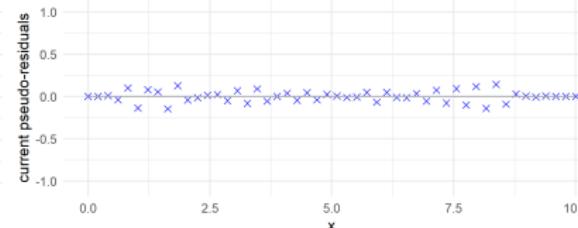
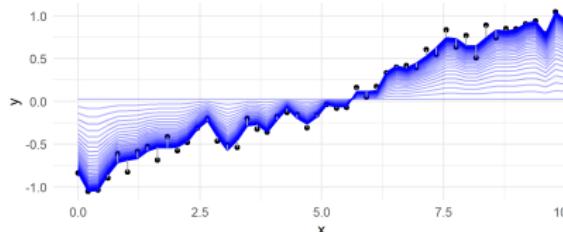


Iteration 10

Shape of PRs affects gradual model fit: L_1 only sees resids' sign, BLs are not affected size of values as in L_2 and hence lead to more moderate changes.

GAM WITH L_2 VS L_1 LOSS

Top: L_2 loss, bottom: L_1 loss

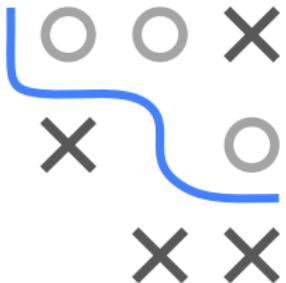
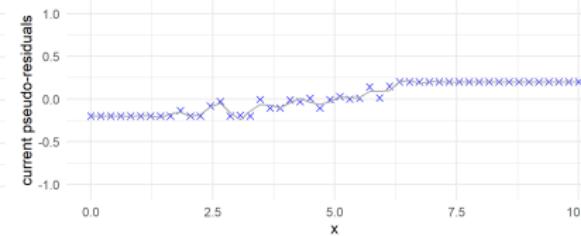
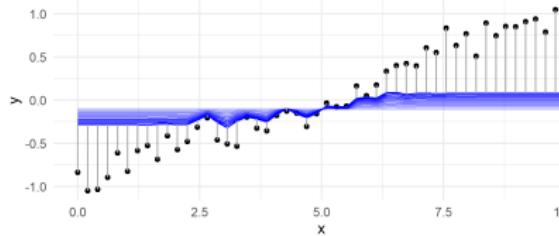
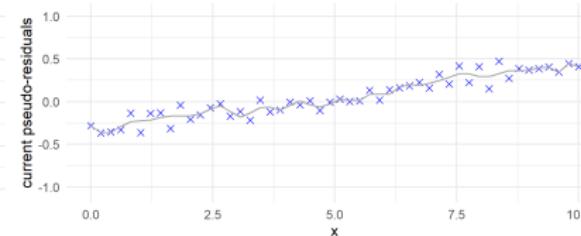
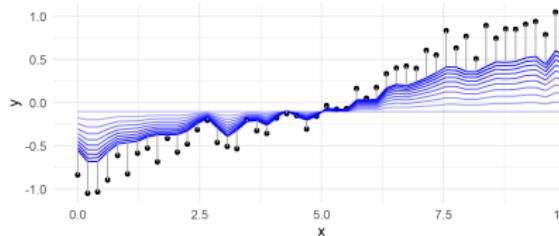


Iteration 100

Shape of PRs affects gradual model fit: L_1 only sees resids' sign, BLs are not affected
size of values as in L_2 and hence lead to more moderate changes.

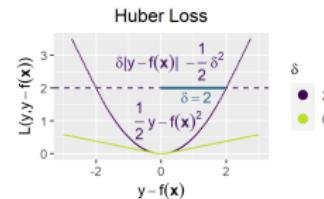
GAM WITH HUBER LOSS

Top: $\delta = 2$, bottom: $\delta = 0.2$.



Iteration 10

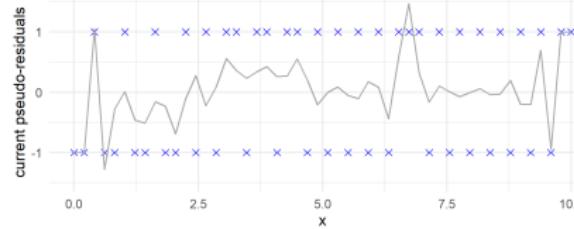
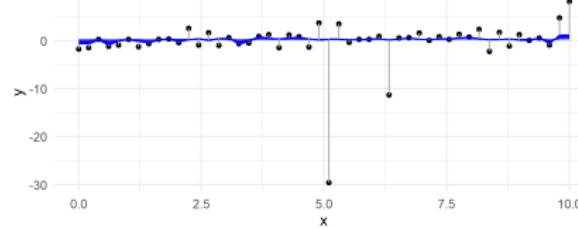
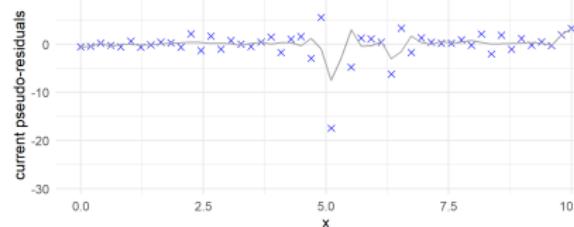
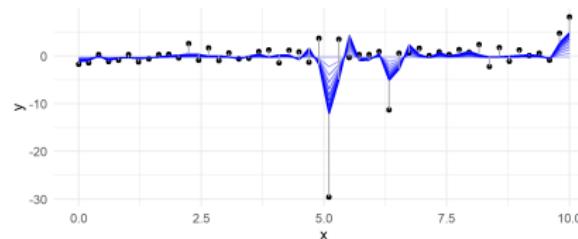
For small δ , PRs are often bounded, resulting in $L1$ -like behavior, while the upper plot more closely resembles $L2$ loss.



GAM WITH OUTLIERS

Instead of Gaussian noise, let's use t -distrib, that leads to outliers in y .

Top: L_2 , bottom: L_1 .



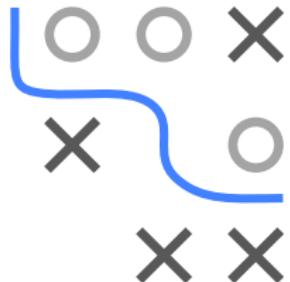
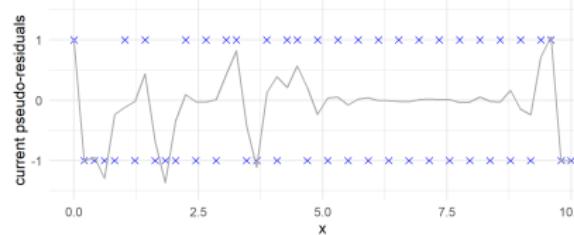
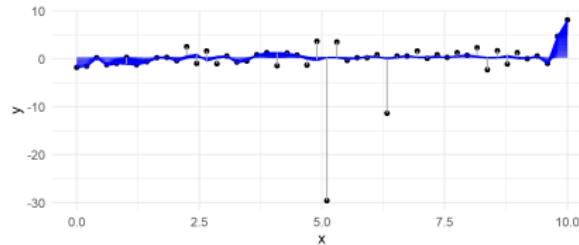
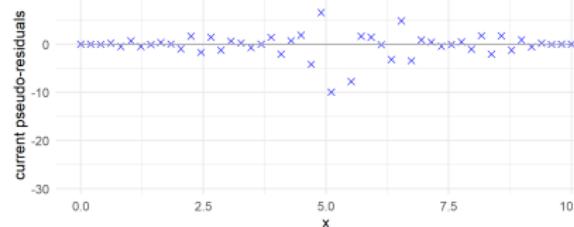
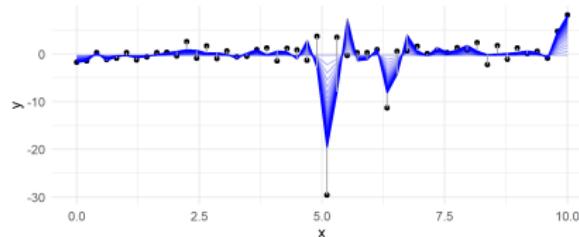
Iteration 10

L_2 loss is affected by outliers rather strongly, whereas L_1 solely considers residuals' sign and not their magnitude, resulting in a more robust model.

GAM WITH OUTLIERS

Instead of Gaussian noise, let's use t -distrib, that leads to outliers in y .

Top: L_2 , bottom: L_1 .

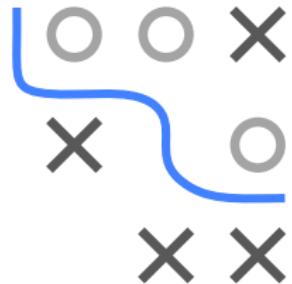
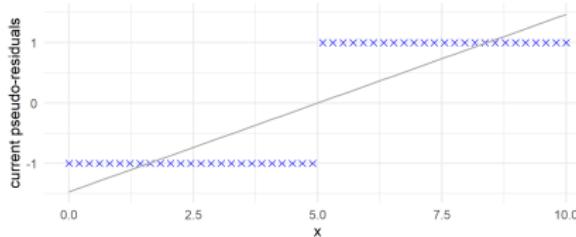
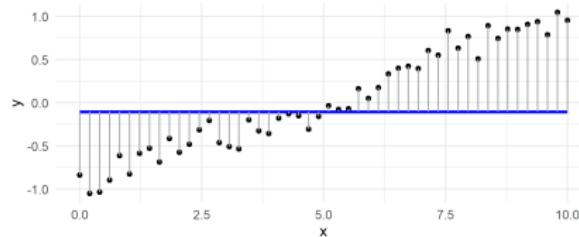
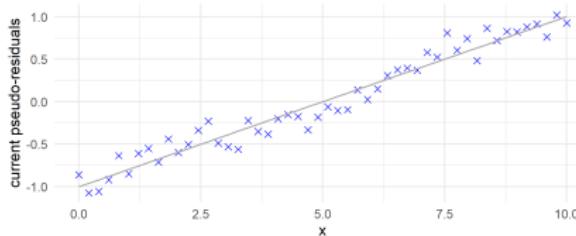
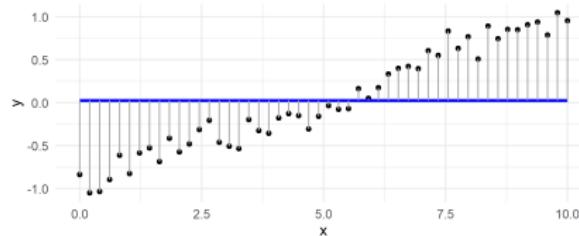


Iteration 100

L_2 loss is affected by outliers rather strongly, whereas L_1 solely considers residuals' sign and not their magnitude, resulting in a more robust model.

LM WITH L_2 VS L_1 LOSS

Top: L_2 , bottom: L_1 .

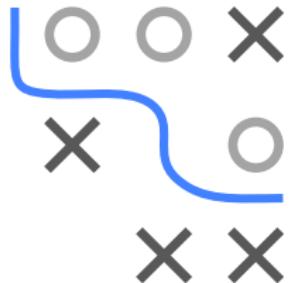
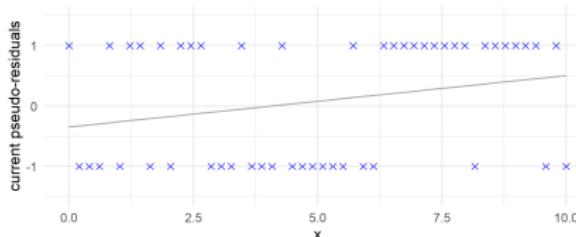
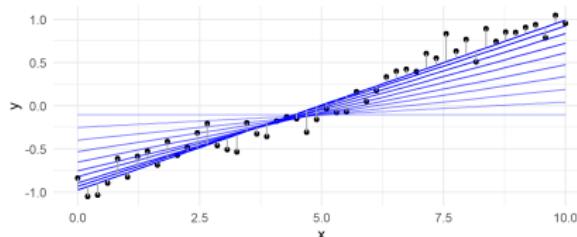
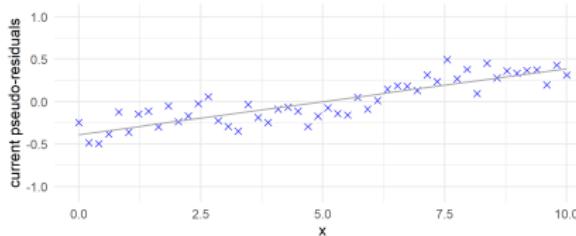
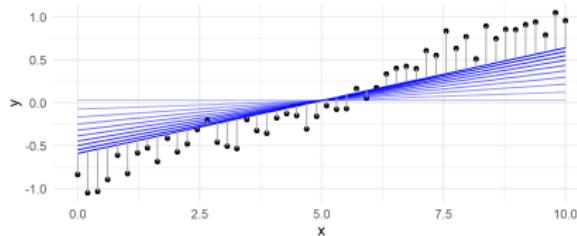


Iteration 1

L_2 : as $\tilde{r}(f) = r(f)$, BL of 1st iter already optimal; but learn rate slows us down.

LM WITH L_2 VS L_1 LOSS

Top: L_2 , bottom: L_1 .

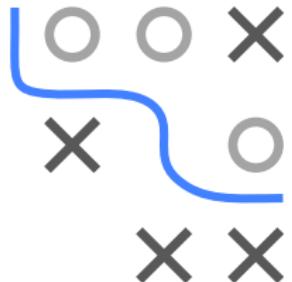
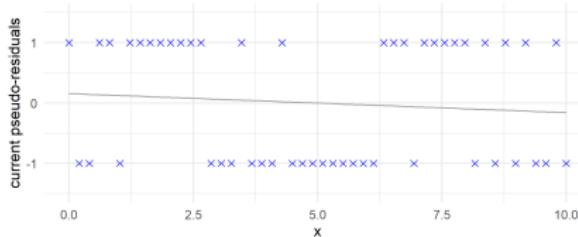
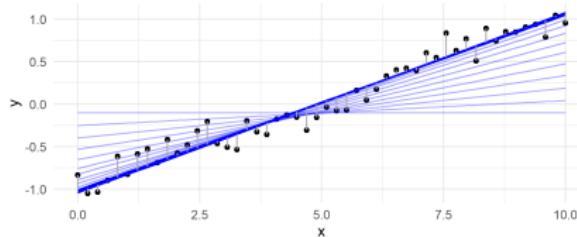
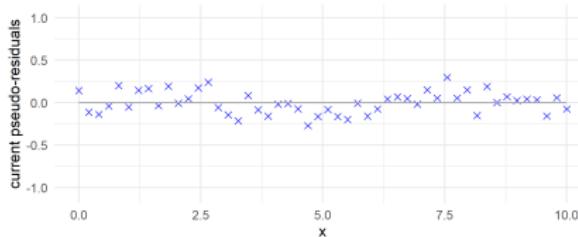
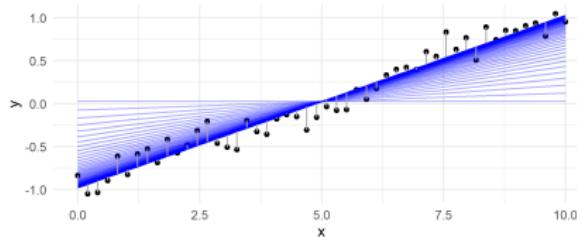


Iteration 10

L_2 : as $\tilde{r}(f) = r(f)$, BL of 1st iter already optimal; but learn rate slows us down.

LM WITH $L2$ VS $L1$ LOSS

Top: $L2$, bottom: $L1$.

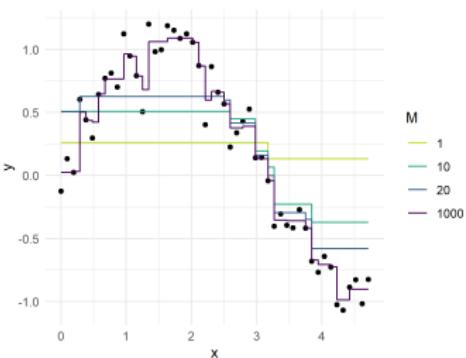


Iteration 100

$L2$: as $\tilde{r}(f) = r(f)$, BL of 1st iter already optimal; but learn rate slows us down.

Introduction to Machine Learning

Gradient Boosting: Regularization



Learning goals

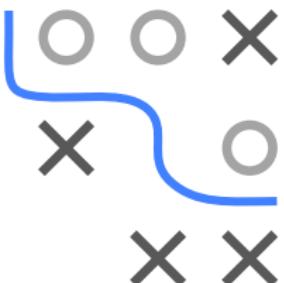
- Learn about three main regularization options: number of iterations, tree depth and shrinkage
- Understand how regularization influences model fit

ITERS, TREE DEPTH, LEARN RATE

GB can overfit easily, due to its aggressive loss minimization.

Options for regularization:

- Limit nr of iters M , i.e., additive components (“early stopping”),
- Limit depth of trees. Can also be interpreted as choosing the order of interaction (see later).
- Use a small learn rate α for only mild model updates.
 α a.k.a. shrinkage.

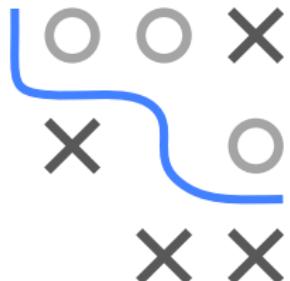


Practical hints:

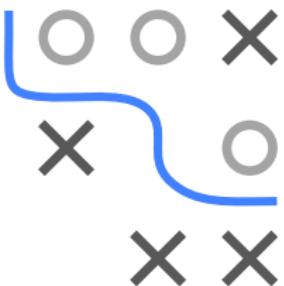
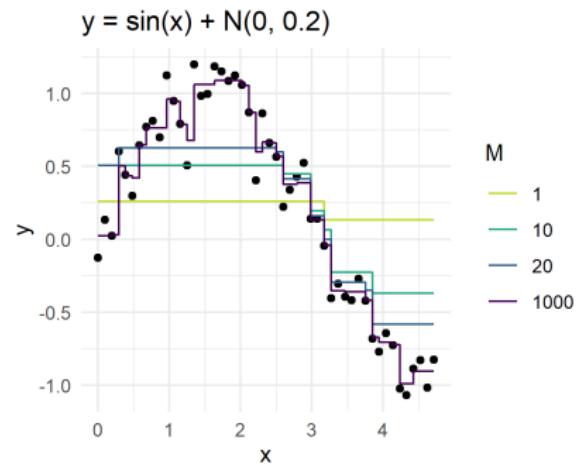
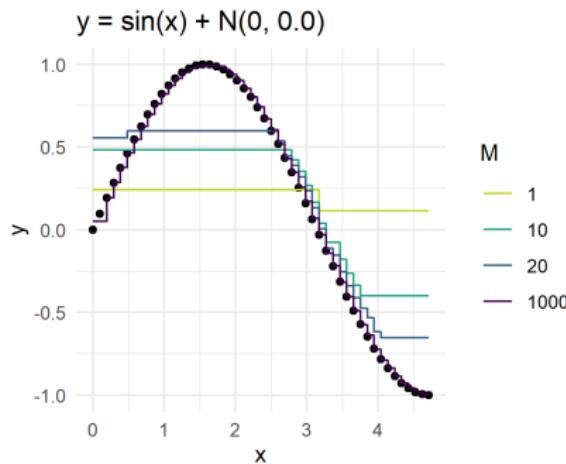
- Optimal values for M and α strongly depend on each other: by increasing M one can use a smaller value for α and vice versa.
- Fast option = Make α small and choose M by CV.
- Probably best to tune all 3 hyperpars jointly via, e.g., CV.

STOCHASTIC GRADIENT BOOSTING

- Minor modification to incorporate the advantages of bagging
- In each iter, we only fit on a random subsample of the train data
- Especially for small train sets, this often leads helps
- Size of random sets = new hyperpar



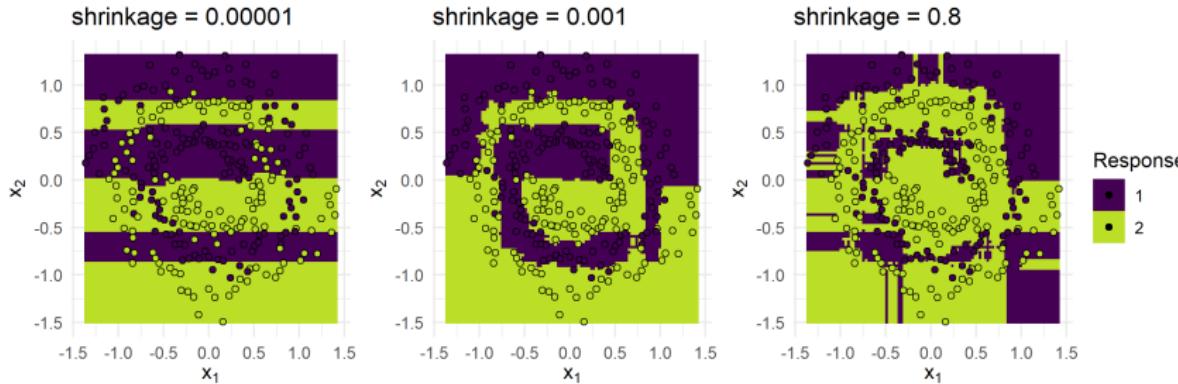
EXAMPLE: SINUSOIDAL WITH TREE STUMPS



Works quite nicely without noise, but overfits on the RHS.

EXAMPLE: SPIRALS DATA

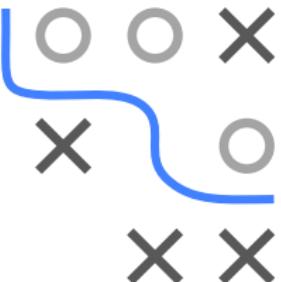
We examine effect of learn rate, with fixed nr of trees and fixed depth.



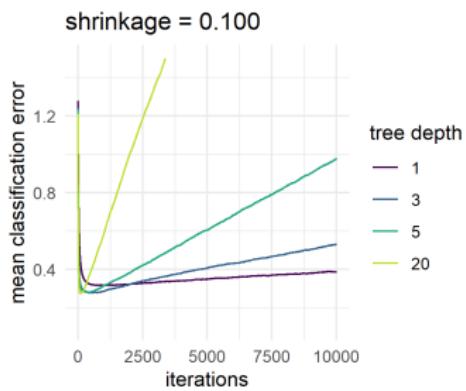
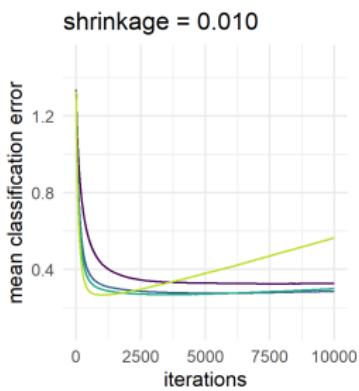
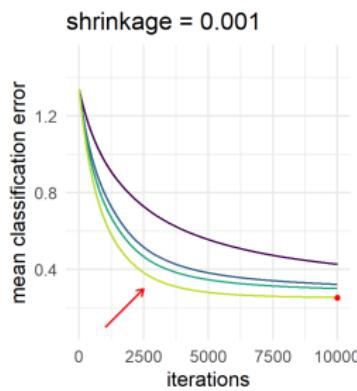
We observe an oversmoothing effect in the left scenario with strong regularization (i.e., very small learning rate) and overfitting when regularization is too weak (right). $\alpha = 0.001$ yields a pretty good fit.

EXAMPLE: SPAM DETECTION WITH TREES

Hyperpar	Range
Loss	Bernoulli (for classification)
Number of trees M	$\{0, 1, \dots, 10000\}$
Shrinkage α	$\{0.001, 0.01, 0.1\}$
Max. tree depth	$\{1, 3, 5, 20\}$

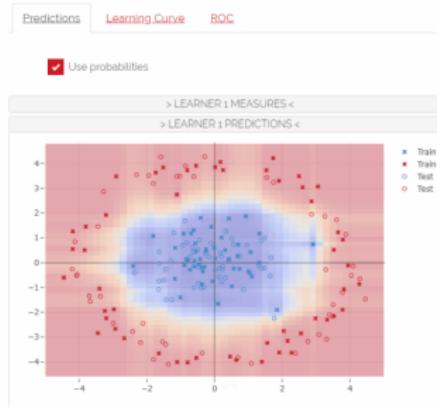


Use 3-CV in grid search; optimal config in red:



Introduction to Machine Learning

Gradient Boosting: Classification



Learning goals

- GB for binary classification simply uses Bernoulli or exponential loss
- For multiclass we fit g discriminant functions in parallel

BINARY CLASSIFICATION

For $\mathcal{Y} = \{0, 1\}$, we simply have to select an appropriate loss function, so let us use Bernoulli loss as in logistic regression:

$$L(y, f(\mathbf{x})) = -y \cdot f(\mathbf{x}) + \log(1 + \exp(f(\mathbf{x}))).$$

Then,

$$\begin{aligned}\tilde{r}(f) &= -\frac{\partial L(y, f(\mathbf{x}))}{\partial f(\mathbf{x})} \\ &= y - \frac{\exp(f(\mathbf{x}))}{1 + \exp(f(\mathbf{x}))} \\ &= y - \frac{1}{1 + \exp(-f(\mathbf{x}))} = y - s(f(\mathbf{x})).\end{aligned}$$

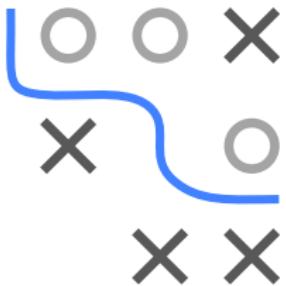
Here, $s(f(\mathbf{x}))$ is the logistic function, applied to a scoring model. Hence, effectively, the pseudo-residuals are $y - \pi(\mathbf{x})$.

Through $\pi(\mathbf{x}) = s(f(\mathbf{x}))$ we can also estimate posterior probabilities.



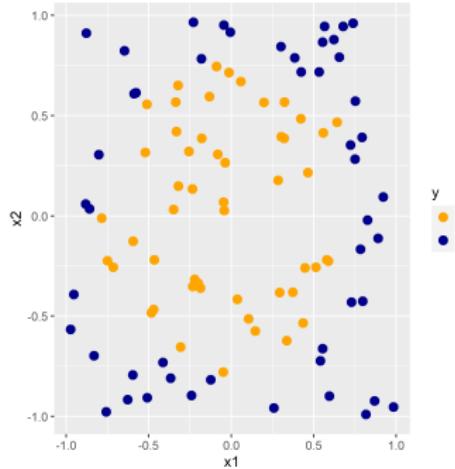
BINARY CLASSIFICATION

- Rest works as in regression.
- NB: We fit regression BLs against the PRs with L_2 loss.
- Exponential loss works too. In practice there is no big difference, although Bernoulli loss makes a bit more sense from a theoretical (maximum likelihood) perspective.
- It can be shown GB with exp loss is basically equivalent to and generalizes AdaBoost.



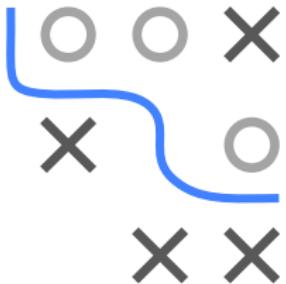
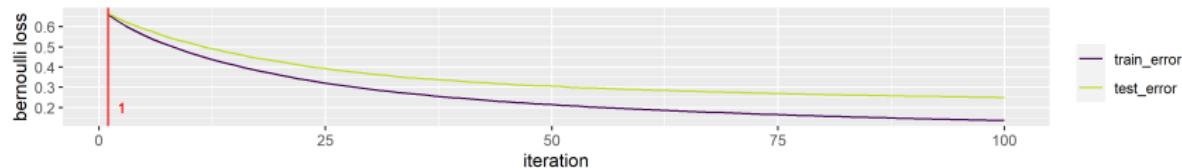
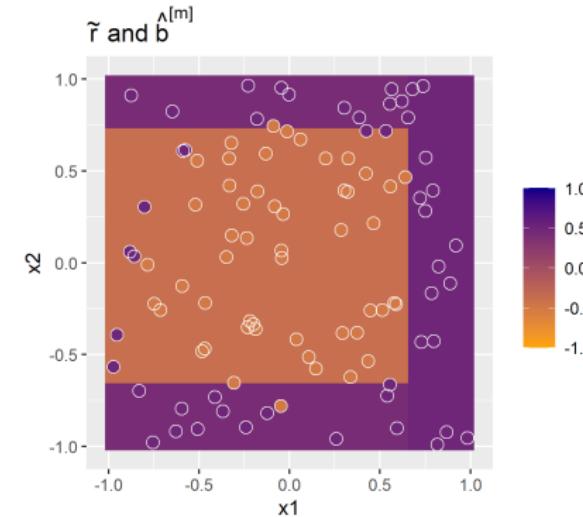
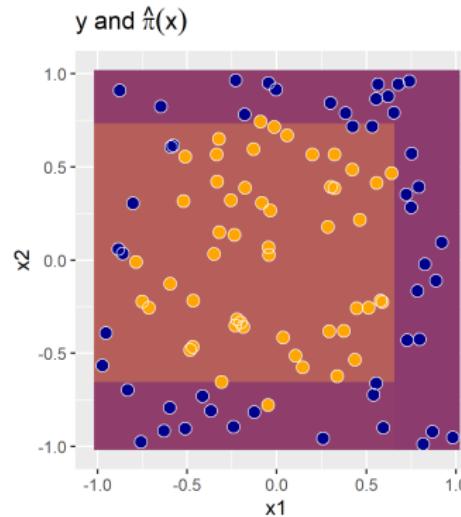
EXAMPLE: 2D CIRCLE DATA

- mlbench circle data with $n = 100$
- Bernoulli loss
- BL = shallow tree with max. depth of 3
- We initialized with $f^{[0]} = 0$.



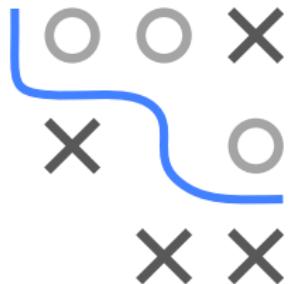
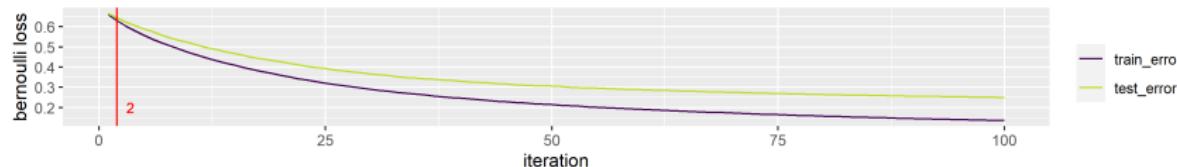
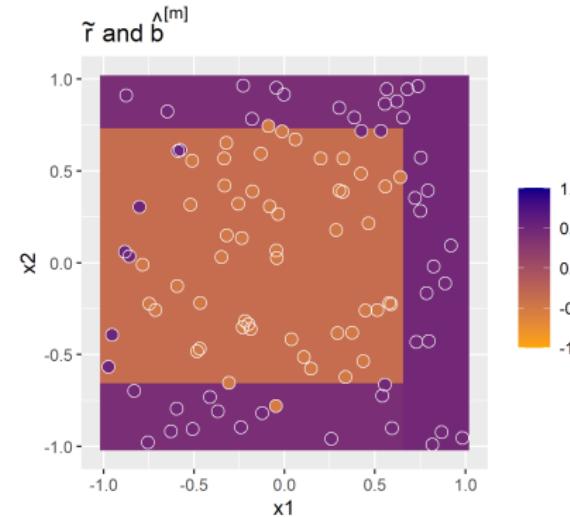
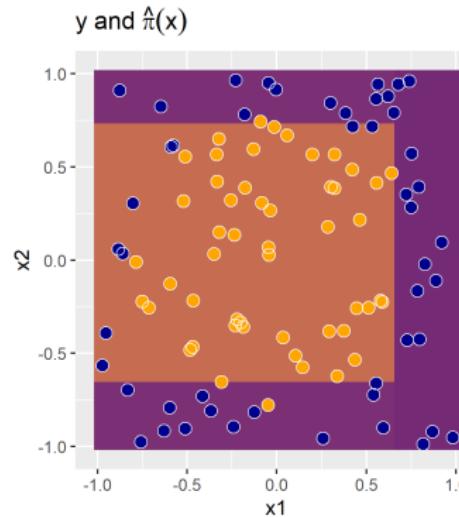
EXAMPLE: 2D CIRCLE DATA

BG color is predicted probs on LHS on RHS we show and preds of BL.



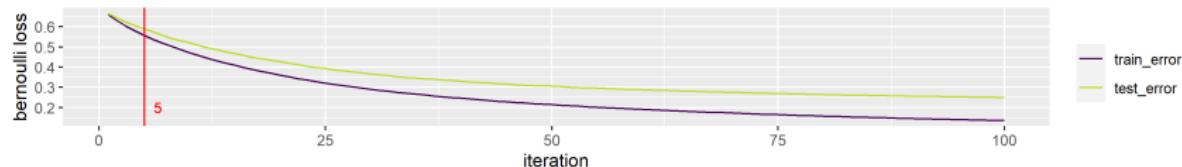
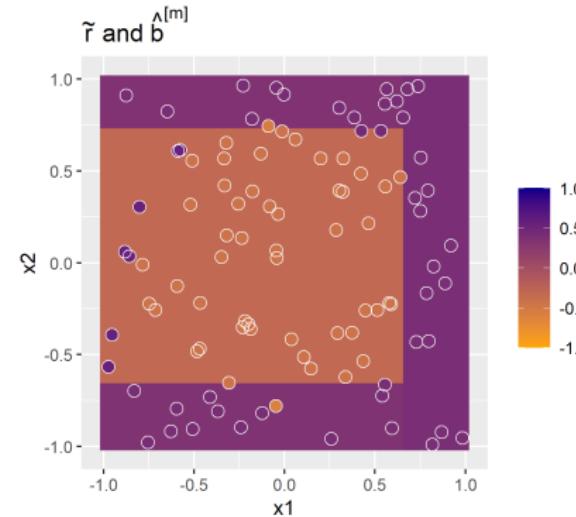
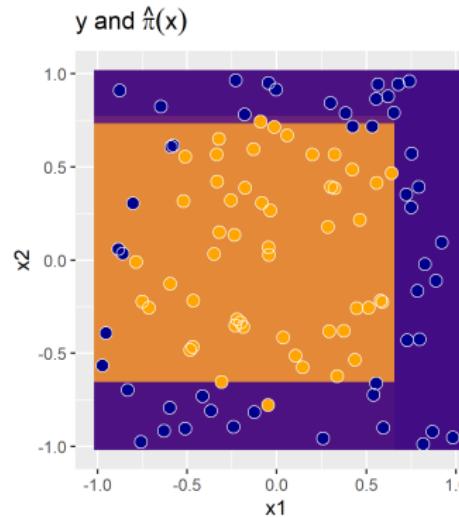
EXAMPLE: 2D CIRCLE DATA

BG color is predicted probs on LHS on RHS we show and preds of BL.



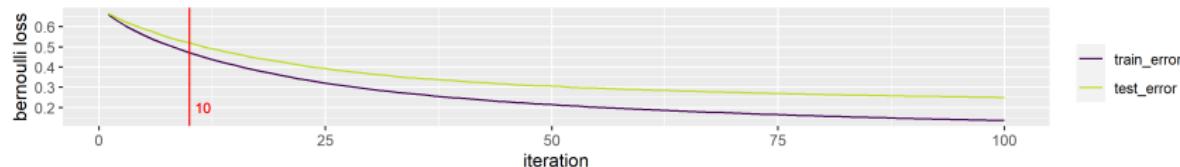
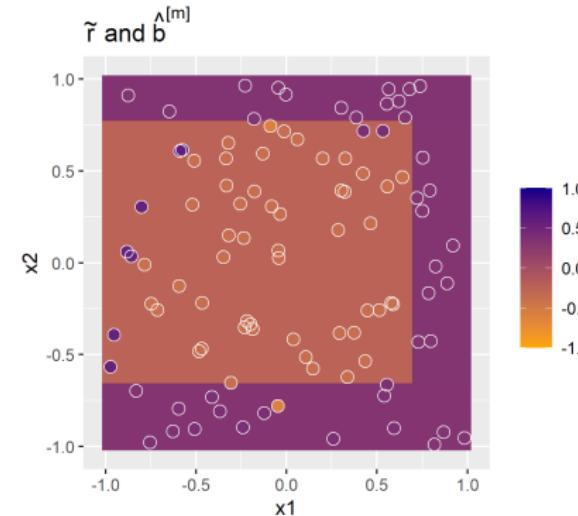
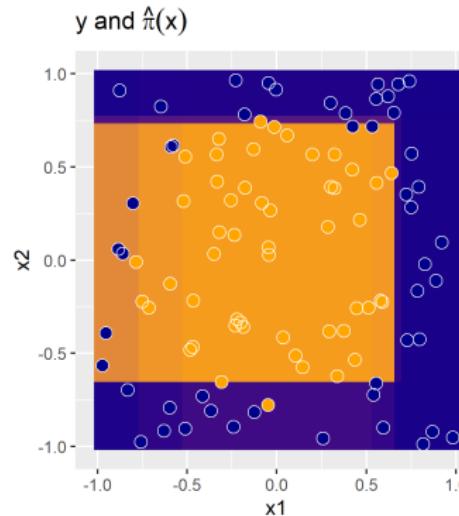
EXAMPLE: 2D CIRCLE DATA

BG color is predicted probs on LHS on RHS we show and preds of BL.



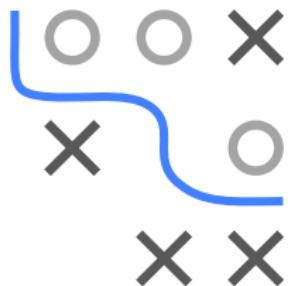
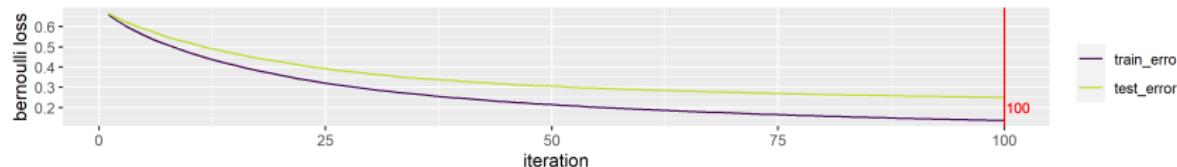
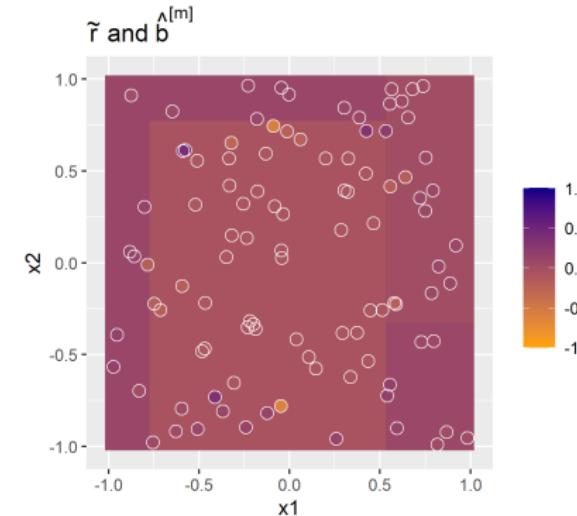
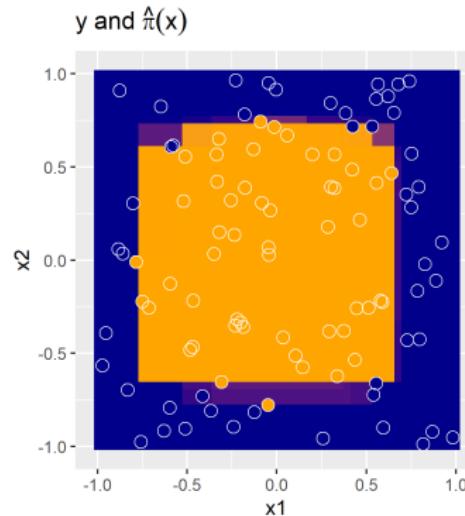
EXAMPLE: 2D CIRCLE DATA

BG color is predicted probs on LHS on RHS we show and preds of BL.



EXAMPLE: 2D CIRCLE DATA

BG color is predicted probs on LHS on RHS we show and preds of BL.



MULTICLASS PROBLEMS

We proceed as in softmax regression and model a categorical distribution with multinomial / log loss. For $\mathcal{Y} = \{1, \dots, g\}$, we create g discriminant functions $f_k(\mathbf{x})$, one for each class and each one being an **additive** model of base learners.

We define the $\pi_k(\mathbf{x})$ through the softmax function:

$$\pi_k(\mathbf{x}) = s_k(f_1(\mathbf{x}), \dots, f_g(\mathbf{x})) = \exp(f_k(\mathbf{x})) / \sum_{j=1}^g \exp(f_j(\mathbf{x})).$$

Multinomial loss L :

$$L(y, f_1(\mathbf{x}), \dots, f_g(\mathbf{x})) = - \sum_{k=1}^g \mathbb{1}_{\{y=k\}} \ln \pi_k(\mathbf{x}).$$

Pseudo-residuals:

$$-\frac{\partial L(y, f_1(\mathbf{x}), \dots, f_g(\mathbf{x}))}{\partial f_k(\mathbf{x})} = \mathbb{1}_{\{y=k\}} - \pi_k(\mathbf{x}).$$



MULTICLASS PROBLEMS

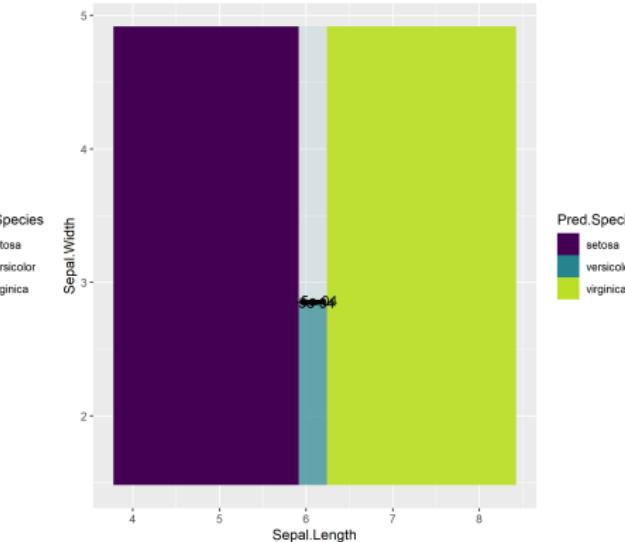
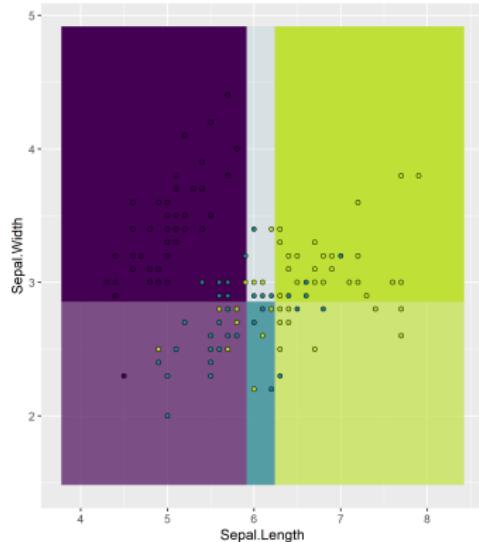
Algorithm GB for Multiclass

```
1: Initialize  $f_k^{[0]}(\mathbf{x}) = 0$ ,  $k = 1, \dots, g$ 
2: for  $m = 1 \rightarrow M$  do
3:   Set  $\pi_k^{[m]}(\mathbf{x}) = \frac{\exp(f_k^{[m]}(\mathbf{x}))}{\sum_j \exp(f_j^{[m]}(\mathbf{x}))}$ ,  $k = 1, \dots, g$ 
4:   for  $k = 1 \rightarrow g$  do
5:     For all  $i$ : Compute  $\tilde{r}_k^{[m](i)} = \mathbb{1}_{\{y^{(i)}=k\}} - \pi_k^{[m]}(\mathbf{x}^{(i)})$ 
6:     Fit a regression base learner  $\hat{b}_k^{[m]}$  to the pseudo-residuals  $\tilde{r}_k^{[m](i)}$ .
7:     Update  $\hat{f}_k^{[m]} = \hat{f}_k^{[m-1]} + \alpha \hat{b}_k^{[m]}$ 
8:   end for
9: end for
10: Output  $\hat{f}_1^{[M]}, \dots, \hat{f}_g^{[M]}$ 
```



EXAMPLE: 2D IRIS

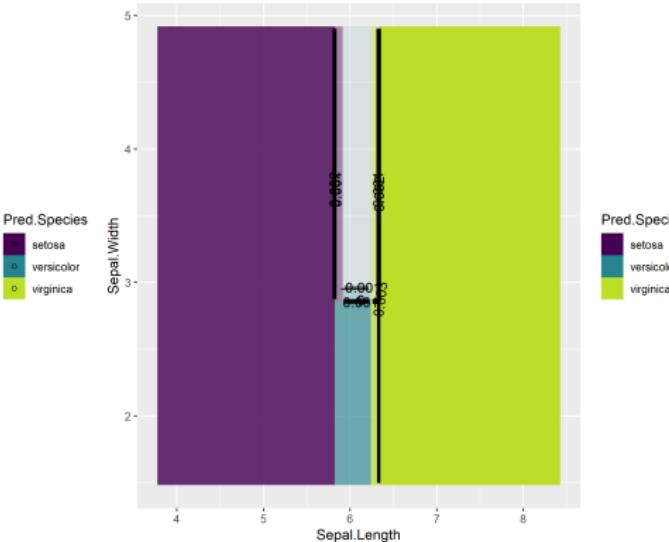
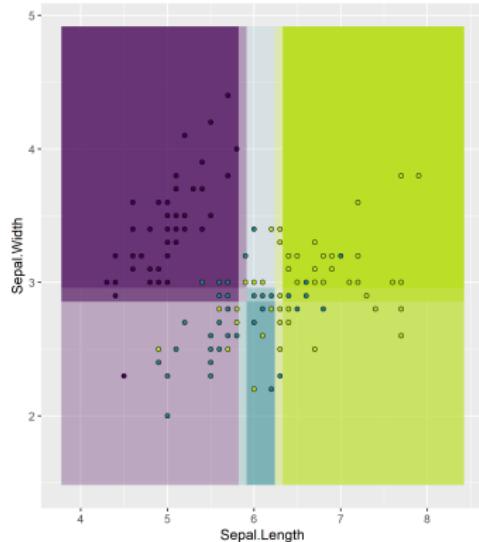
LHS: BG color is predicted probs and point col is true label; RHS:
Contour lines of discriminant functions.



Iteration=1

EXAMPLE: 2D IRIS

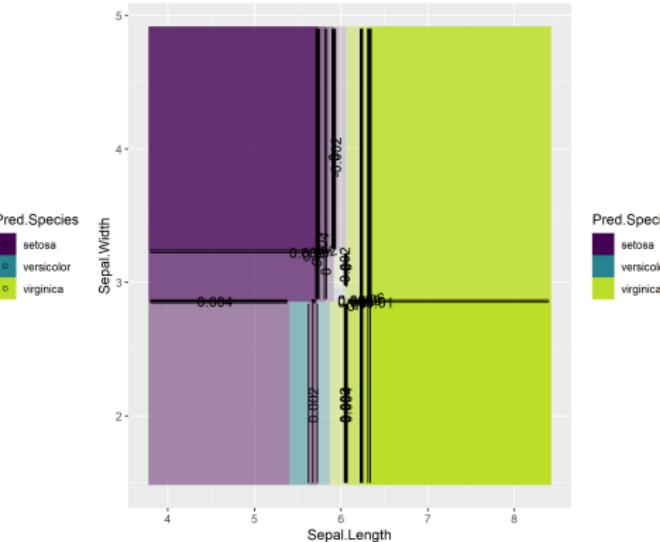
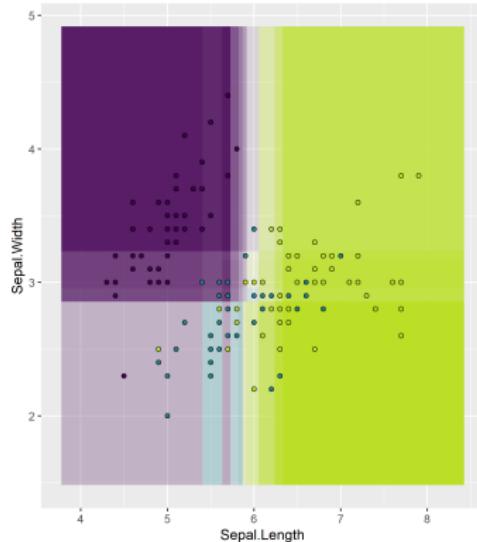
LHS: BG color is predicted probs and point col is true label; RHS:
Contour lines of discriminant functions.



Iteration=2

EXAMPLE: 2D IRIS

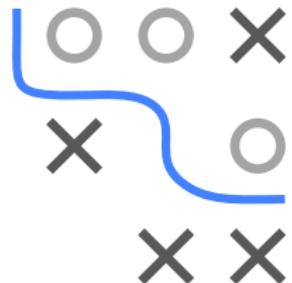
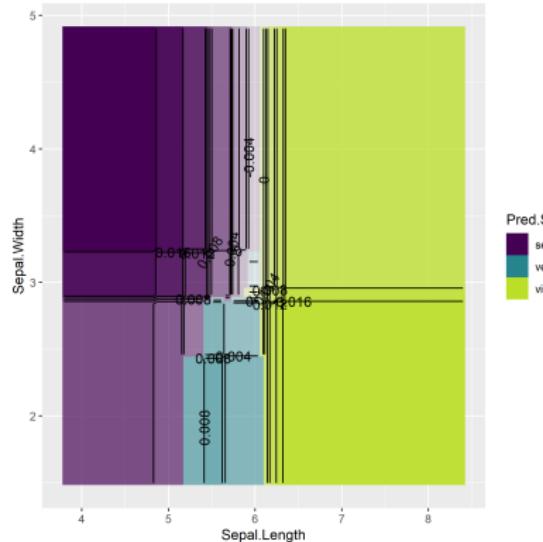
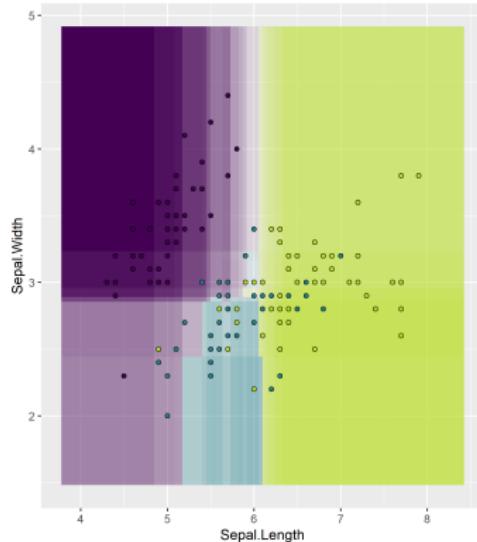
LHS: BG color is predicted probs and point col is true label; RHS:
Contour lines of discriminant functions.



Iteration=5

EXAMPLE: 2D IRIS

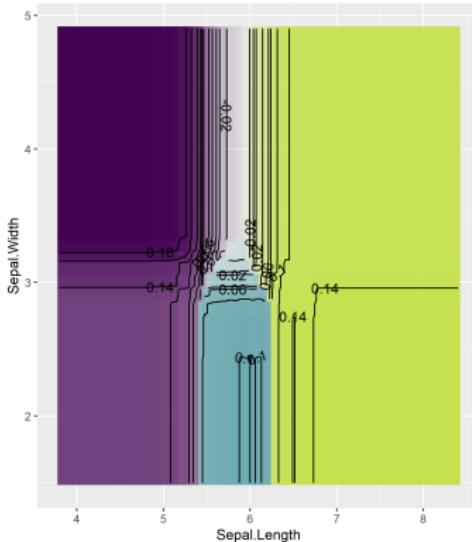
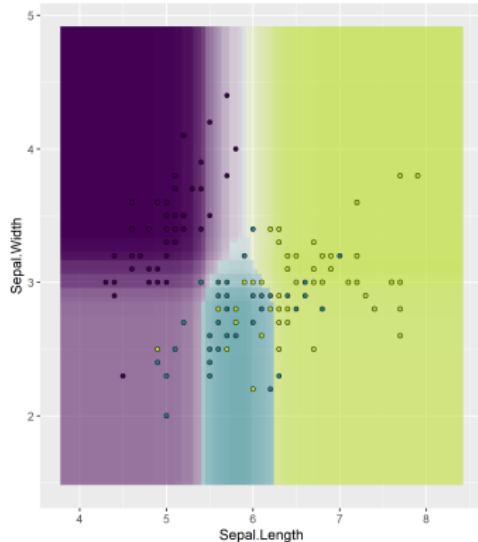
LHS: BG color is predicted probs and point col is true label; RHS:
Contour lines of discriminant functions.



Iteration=10

EXAMPLE: 2D IRIS

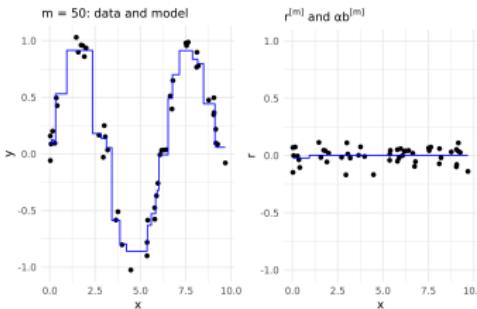
LHS: BG color is predicted probs and point col is true label; RHS:
Contour lines of discriminant functions.



Iteration=100

Introduction to Machine Learning

Gradient Boosting with Trees 1



Learning goals

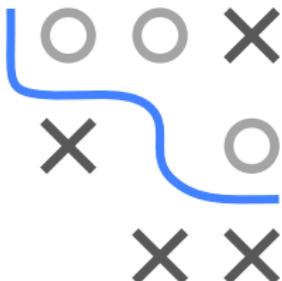
- Examples for GB with trees
- Understand relationship between model structure and interaction depth

GRADIENT BOOSTING WITH TREES

Trees are most popular BLs in GB.

Reminder: advantages of trees

- No problems with categorical features.
- No problems with outliers in feature values.
- No problems with missing values.
- No problems with monotone transformations of features.
- Trees (and stumps!) can be fitted quickly, even for large n .
- Trees have a simple, built-in type of variable selection.

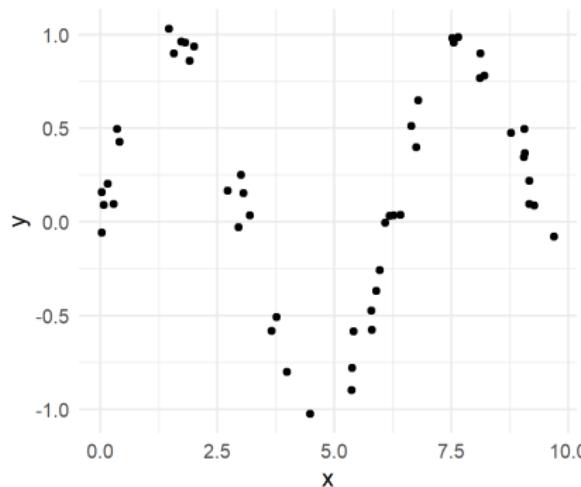
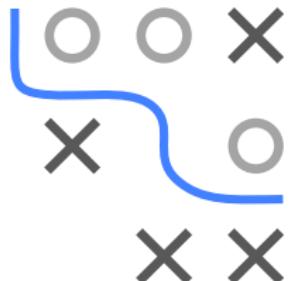


GB with trees inherits these, and strongly improves predictive power.

EXAMPLE 1

Simulation setting:

- Given: one feature x and one numeric target variable y of 50 observations.
- x is uniformly distributed between 0 and 10.
- y depends on x as follows: $y^{(i)} = \sin(x^{(i)}) + \epsilon^{(i)}$ with $\epsilon^{(i)} \sim \mathcal{N}(0, 0.01)$,
 $\forall i \in \{1, \dots, 50\}$.



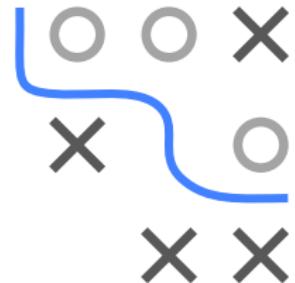
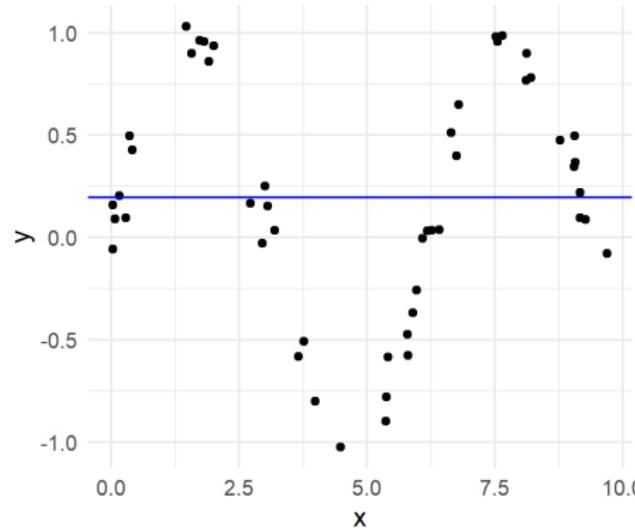
Aim: we want to fit a gradient boosting model to the data by using stumps as base learners.

Since we are facing a regression problem, we use $L2$ loss.

EXAMPLE 1

Iteration 0: initialization by optimal constant (mean) prediction $\hat{f}^{[0]}(x) = \bar{y} \approx 0.2$.

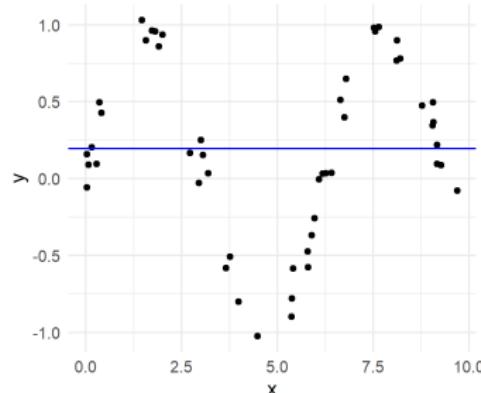
i	$x^{(i)}$	$y^{(i)}$	$\hat{f}^{[0]}$
1	0.03	0.16	0.20
2	0.03	-0.06	0.20
3	0.07	0.09	0.20
\vdots	\vdots	\vdots	\vdots
50	9.69	-0.08	0.20



EXAMPLE 1

Iteration 1: (1) Calculate pseudo-residuals $\tilde{r}^{[m](i)}$ and (2) fit a regression stump $b^{[m]}$.

i	$x^{(i)}$	$y^{(i)}$	$\hat{f}^{[0]}$	$\tilde{r}^{[1](i)}$	$\hat{b}^{[1](i)}$
1	0.03	0.16	0.20	-0.04	-0.17
2	0.03	-0.06	0.20	-0.25	-0.17
3	0.07	0.09	0.20	-0.11	-0.17
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
50	9.69	-0.08	0.20	-0.27	0.33

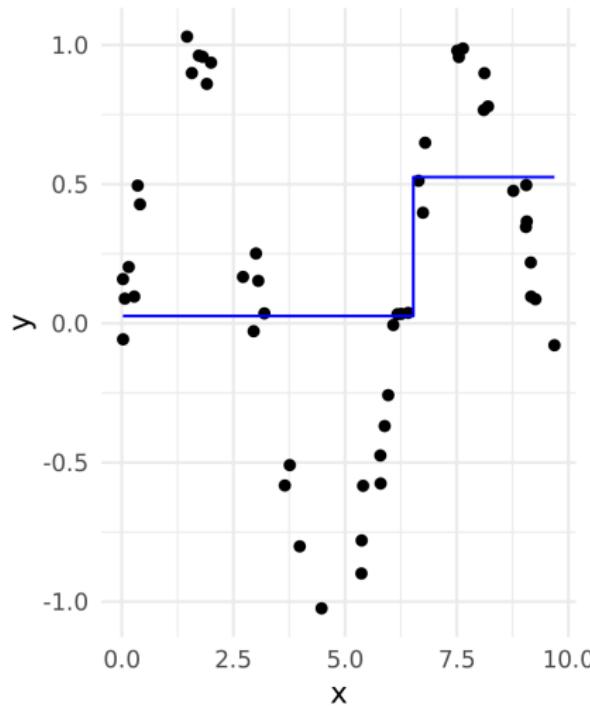


(3) Update model by $\hat{f}^{[1]}(x) = \hat{f}^{[0]}(x) + \hat{b}^{[1]}$.

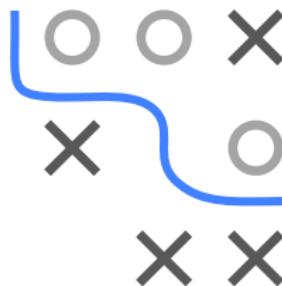
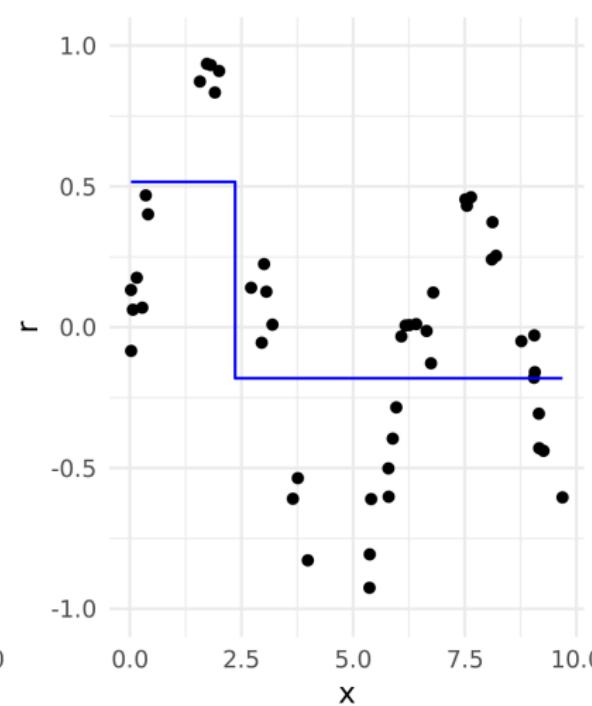
EXAMPLE 1

Repeat step (1) to (3):

$m = 1$: data and model

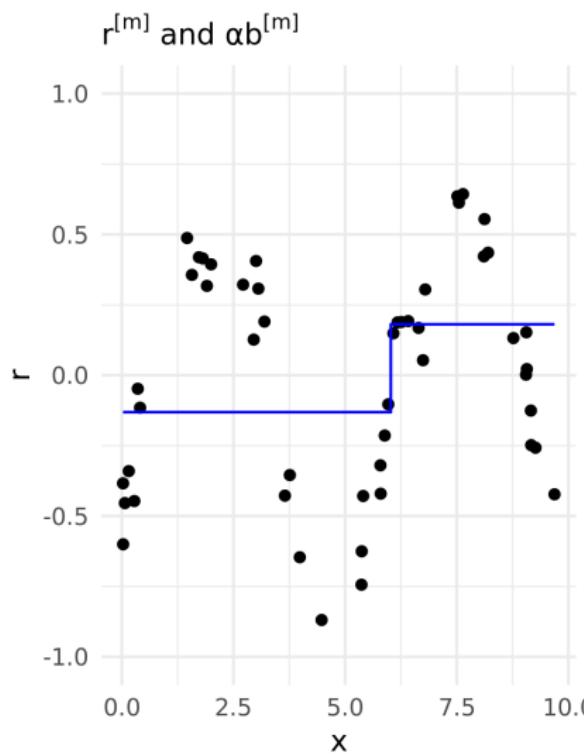
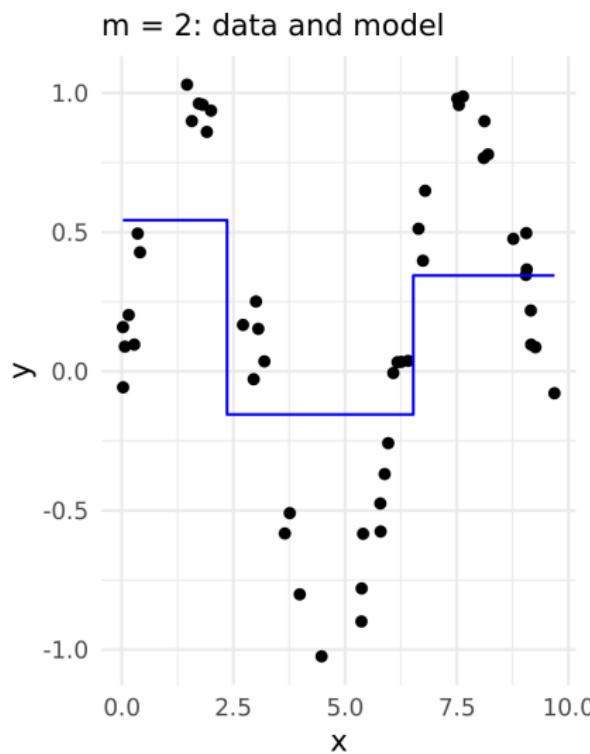


$r^{[m]}$ and $\alpha b^{[m]}$



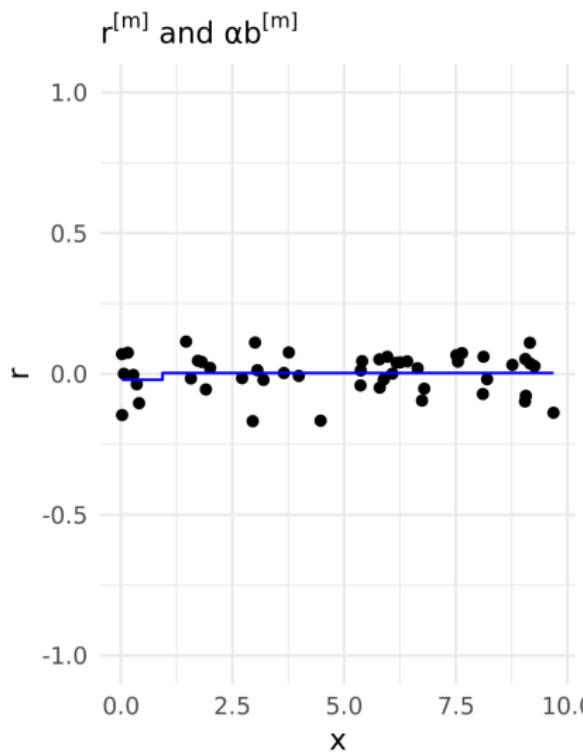
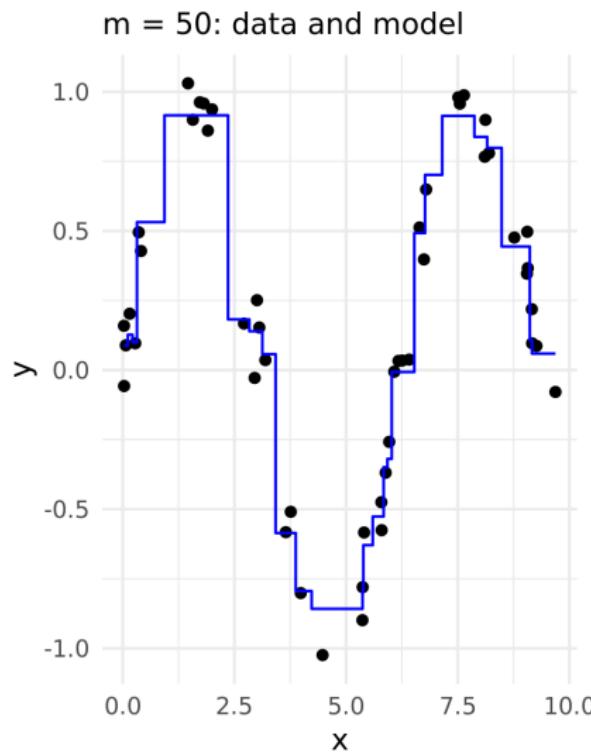
EXAMPLE 1

Repeat step (1) to (3):



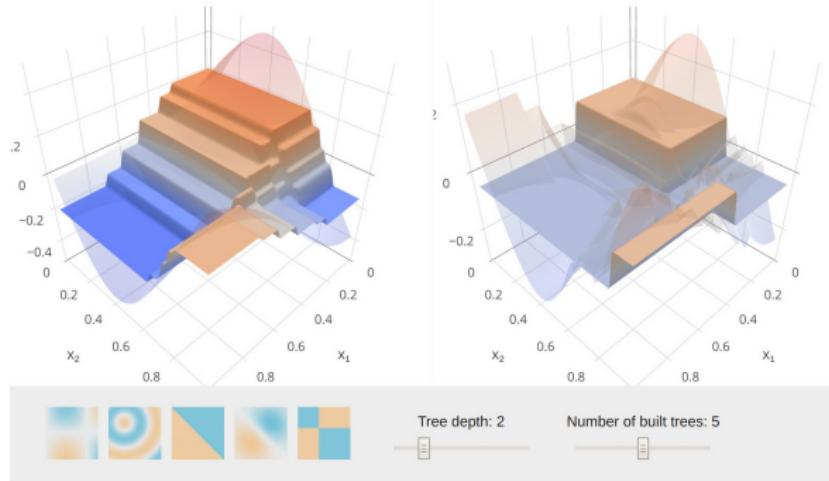
EXAMPLE 1

Repeat step (1) to (3):



EXAMPLE 2

This [website](#) shows on various 3D examples how tree depth and number of iterations influence the model fit of a GBM with trees.

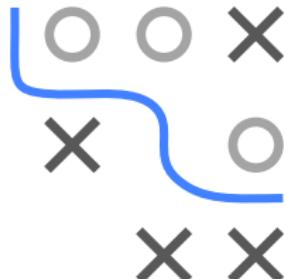


MODEL STRUCTURE AND INTERACTION DEPTH

Model structure directly influenced by depth of $b^{[m]}(\mathbf{x})$.

$$f(\mathbf{x}) = \sum_{m=1}^M \alpha^{[m]} b^{[m]}(\mathbf{x})$$

Remember how we can write trees as additive model over paths to leafs.



With stumps (depth = 1), $f(\mathbf{x})$ is additive model

(GAM) without interactions:

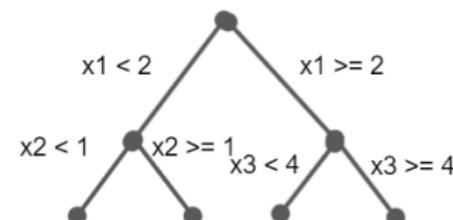
$$f(\mathbf{x}) = f_0 + \sum_{j=1}^p f_j(x_j)$$



With trees of depth 2, we have two-way interactions:

$$f(\mathbf{x}) = f_0 + \sum_{j=1}^p f_j(x_j) + \sum_{j \neq k} f_{j,k}(x_j, x_k)$$

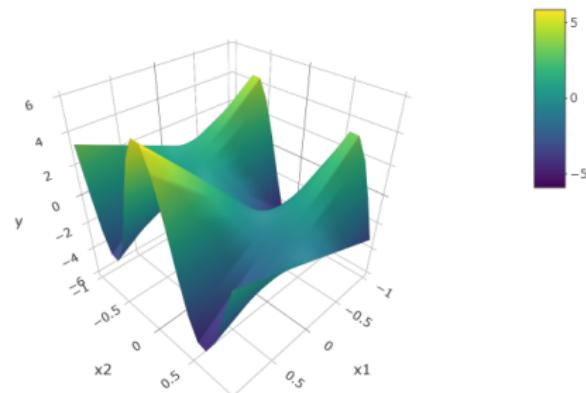
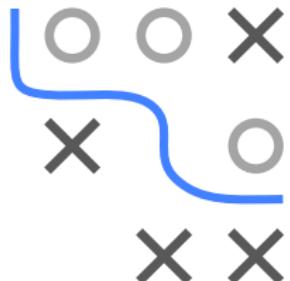
with f_0 being a constant intercept.



MODEL STRUCTURE AND INTERACTION DEPTH

Simulation setting:

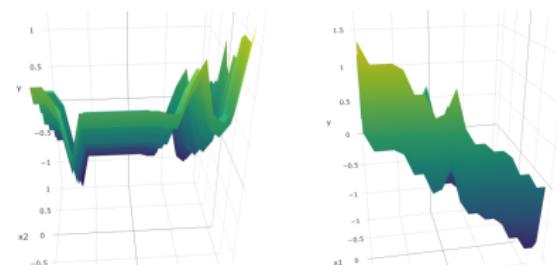
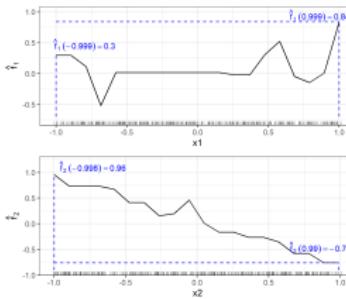
- Features x_1 and x_2 and numeric y ; with $n = 500$
- x_1 and x_2 are uniformly distributed between -1 and 1
- $y^{(i)} = x_1^{(i)} - x_2^{(i)} + 5 \cos(5x_2^{(i)}) \cdot x_1^{(i)} + \epsilon^{(i)}$ with $\epsilon^{(i)} \sim \mathcal{N}(0, 1)$
- We fit 2 GB models, with tree depth 1 and 2, respectively.



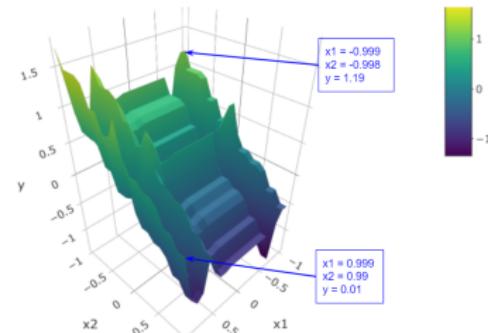
MODEL STRUCTURE AND INTERACTION DEPTH

GBM with interaction depth of 1 (GAM)

No interactions are modelled: Marginal effects of x_1 and x_2 add up to joint effect (plus the constant intercept $\hat{f}_0 = -0.07$).



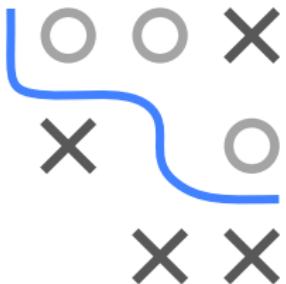
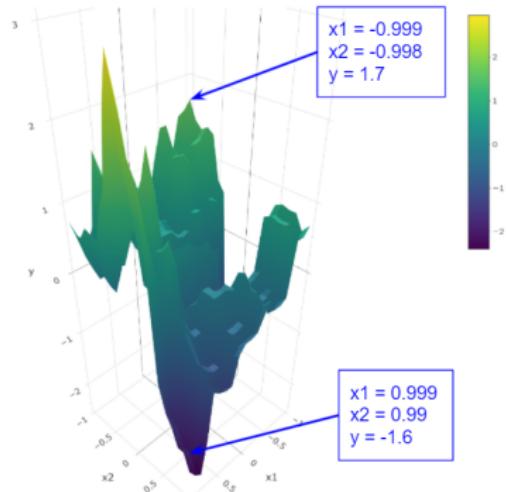
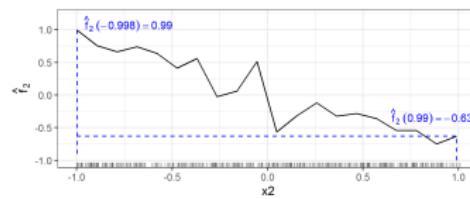
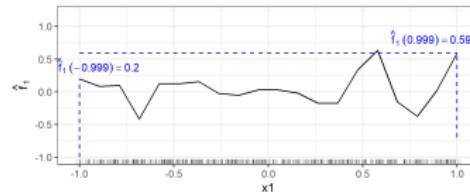
$$\begin{aligned}\hat{f}(-0.999, -0.998) \\&= \hat{f}_0 + \hat{f}_1(-0.999) + \hat{f}_2(-0.998) \\&= -0.07 + 0.3 + 0.96 = 1.19\end{aligned}$$



MODEL STRUCTURE AND INTERACTION DEPTH

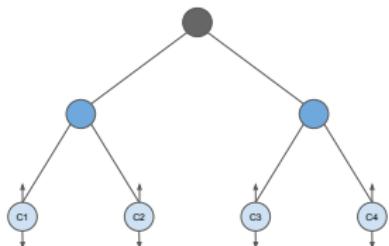
GBM with interaction depth of 2

Interactions between x_1 and x_2 are modelled: Marginal effects of x_1 and x_2 do NOT add up to joint effect due to interaction effects.



Introduction to Machine Learning

Gradient Boosting with Trees 2



Learning goals

- Loss optimal terminal coefficients
- GB with trees for multiclass problems

ADAPTING TERMINAL COEFFICIENTS

- Tree as additive model: $b(\mathbf{x}) = \sum_{t=1}^T c_t \mathbb{1}_{\{\mathbf{x} \in R_t\}}$,
- R_t are the terminal regions; c_t are terminal constants

The GB model is still additive in the regions:



$$\begin{aligned}f^{[m]}(\mathbf{x}) &= f^{[m-1]}(\mathbf{x}) + \alpha^{[m]} b^{[m]}(\mathbf{x}) \\&= f^{[m-1]}(\mathbf{x}) + \alpha^{[m]} \sum_{t=1}^{T^{[m]}} c_t^{[m]} \mathbb{1}_{\{\mathbf{x} \in R_t^{[m]}\}} \\&= f^{[m-1]}(\mathbf{x}) + \sum_{t=1}^{T^{[m]}} \tilde{c}_t^{[m]} \mathbb{1}_{\{\mathbf{x} \in R_t^{[m]}\}}.\end{aligned}$$

With $\tilde{c}_t^{[m]} = \alpha^{[m]} \cdot c_t^{[m]}$ in the case that $\alpha^{[m]}$ is a constant learning rate

ADAPTING TERMINAL COEFFICIENTS

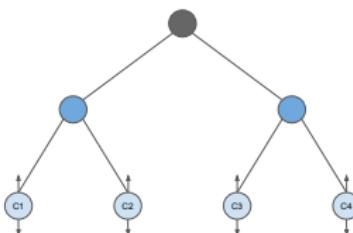
After the tree has been fitted against the PRs, we can adapt terminal constants in a second step to become more loss optimal.



$$f^{[m]}(\mathbf{x}) = f^{[m-1]}(\mathbf{x}) + \sum_{t=1}^{T^{[m]}} \tilde{c}_t^{[m]} \mathbb{1}_{\{\mathbf{x} \in R_t^{[m]}\}}.$$

We can determine/change all $\tilde{c}_t^{[m]}$ individually and directly L -optimally:

$$\tilde{c}_t^{[m]} = \arg \min_c \sum_{\mathbf{x}^{(i)} \in R_t^{[m]}} L(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + c).$$

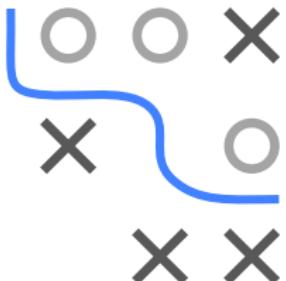


ADAPTING TERMINAL COEFFICIENTS

An alternative approach is to directly fit a loss-optimal tree. Risk for data in a node:

$$\mathcal{R}(\mathcal{N}') = \sum_{i \in \mathcal{N}'} L(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + c)$$

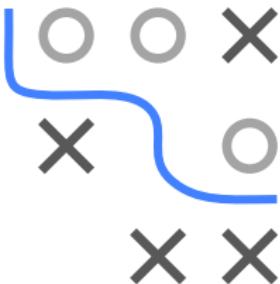
with \mathcal{N}' being the index set of a specific (left or right) node after splitting and c the constant of the node.



c can be found by line search or analytically for some losses.

GB MULTICLASS WITH TREES

- From Friedman, J. H. - Greedy Function Approximation: A Gradient Boosting Machine (1999)
- We again model one discriminant function per class.
- Determining the tree structure works just like before.
- In the estimation of the c values, i.e., the heights of the terminal regions, however, all models depend on each other because of the definition of L . Optimizing this is more difficult, so we will skip some details and present the main idea and results.



GB MULTICLASS WITH TREES

- There is no closed-form solution for finding the optimal $\hat{c}_{tk}^{[m]}$ values. Additionally, the regions corresponding to the different class trees overlap, so that the solution does not reduce to a separate calculation within each region of each tree.
- Hence, we approximate the solution with a single Newton-Raphson step, using a diagonal approximation to the Hessian (we leave out the details here).
- This decomposes the problem into a separate calculation for each terminal node of each tree.
- The result is

$$\hat{c}_{tk}^{[m]} = \frac{g-1}{g} \frac{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} \tilde{r}_k^{[m](i)}}{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} \left| \tilde{r}_k^{[m](i)} \right| \left(1 - \left| \tilde{r}_k^{[m](i)} \right| \right)}.$$



GB MULTICLASS WITH TREES

Algorithm Gradient Boosting for g -class Classification.

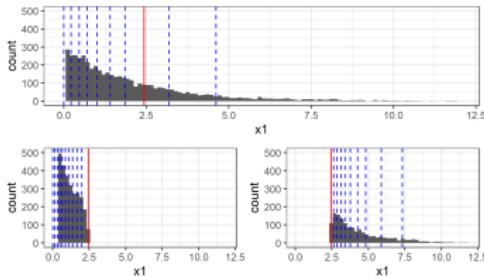
```
1: Initialize  $f_k^{[0]}(\mathbf{x}) = 0$ ,  $k = 1, \dots, g$ 
2: for  $m = 1 \rightarrow M$  do
3:   Set  $\pi_k(\mathbf{x}) = \frac{\exp(f_k^{[m]}(\mathbf{x}))}{\sum_j \exp(f_j^{[m]}(\mathbf{x}))}$ ,  $k = 1, \dots, g$ 
4:   for  $k = 1 \rightarrow g$  do
5:     For all  $i$ : Compute  $\tilde{r}_k^{[m](i)} = \mathbb{1}_{\{y^{(i)}=k\}} - \pi_k(\mathbf{x}^{(i)})$ 
6:     Fit regr. tree to the  $\tilde{r}_k^{[m](i)}$  giving terminal regions  $R_{tk}^{[m]}$ 
7:     Compute
        
$$\hat{c}_{tk}^{[m]} = \frac{g-1}{g} \frac{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} \tilde{r}_k^{[m](i)}}{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} |\tilde{r}_k^{[m](i)}| (1 - |\tilde{r}_k^{[m](i)}|)}$$

9:     Update  $\hat{f}_k^{[m]}(\mathbf{x}) = \hat{f}_k^{[m-1]}(\mathbf{x}) + \sum_t \hat{c}_{tk}^{[m]} \mathbb{1}_{\{\mathbf{x} \in R_{tk}^{[m]}\}}$ 
10:    end for
11:  end for
12: Output  $\hat{f}_1^{[M]}, \dots, \hat{f}_g^{[M]}$ 
```



Introduction to Machine Learning

Gradient Boosting: XGBoost



Learning goals

- Overview over XGB
- Regularization in XGB
- Approximate split finding

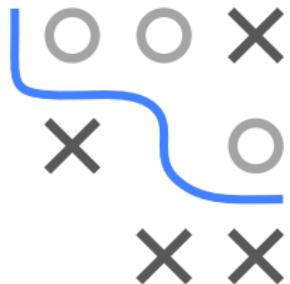
XBG - EXTREME GRADIENT BOOSTING

- Open-source and scalable tree boosting system
- Efficient implementation in *C++* with interfaces to many other programming languages
- Parallel approximate split finding
- Additional regularization techniques
- Feature and data subsampling
- Cluster and GPU support
- Highly optimized and often achieves top performance in benchmarks – if properly tuned



3 EXTRA REGULARIZATION TERMS

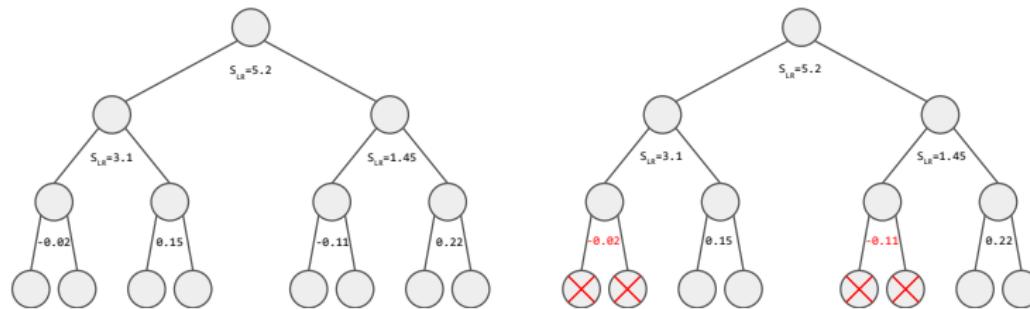
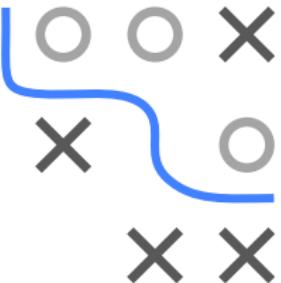
$$\begin{aligned}\mathcal{R}_{\text{reg}}^{[m]} = & \sum_{i=1}^n L \left(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + b^{[m]}(\mathbf{x}^{(i)}) \right) \\ & + \lambda_1 J_1(b^{[m]}) + \lambda_2 J_2(b^{[m]}) + \lambda_3 J_3(b^{[m]}),\end{aligned}$$



- $J_1(b^{[m]}) = T^{[m]}$: Nr of leaves to penalize tree depth
- $J_2(b^{[m]}) = \|\mathbf{c}^{[m]}\|_2^2$: L2 penalty over leaf values
- $J_3(b^{[m]}) = \|\mathbf{c}^{[m]}\|_1$: L1 penalty over leaf values

TREE GROWING

- Grown to max depth
- Fully expanded and leaves split even if no improvement
- At the end, each split that did not improve risk is pruned



SUBSAMPLING

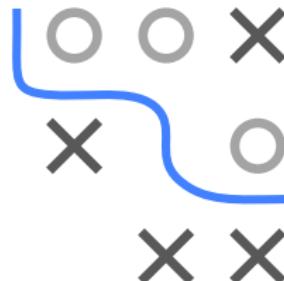
Data Subsampling: XGB uses stochastic GB.

Feature Subsampling: Similar to `mtry` in a random forest only a random subset of features is used for split finding.

The fraction of features for a split can be randomly sampled for each

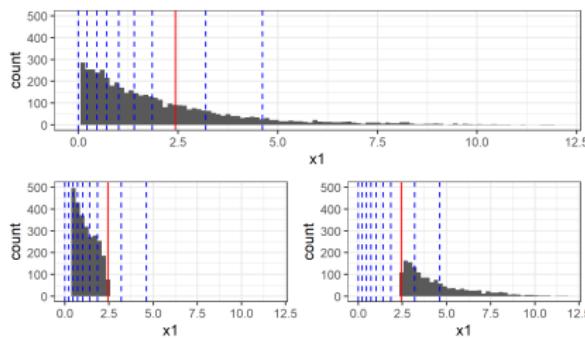
- ① tree
- ② level of a tree
- ③ split

Feature subsampling speeds up training even further and can create a more diverse ensemble that often performs better.

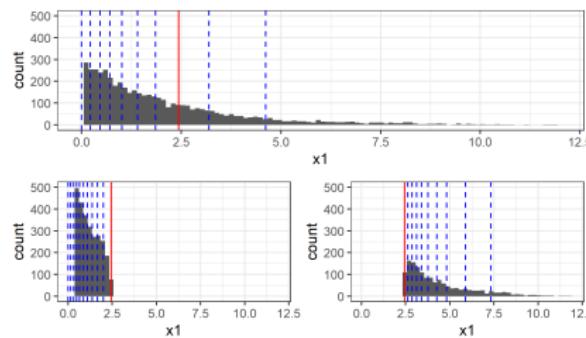


APPROXIMATE SPLIT-FINDING ALGORITHMS

- Speeds up tree building for large data
- Considers not all, but only \lceil / \rceil splits per feature
- Usually percentiles of the empirical distribution of each feature
- Computed once (global) or recomputed after each split (local)
- Called **Histogram-based Gradient Boosting**



Global

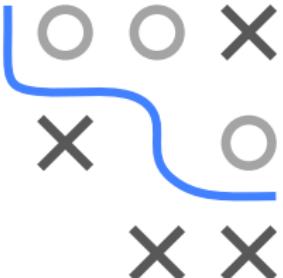


Local

Blue lines are percentiles and red = selected split

DROPOUT ADDITIVE-REGRESSION TREES

DART introduces idea of *dropout* regularization used in DL to boosting



- In iteration m we construct $\hat{b}^{[m]}$
 - To compute PRs we need $\hat{f}^{[m-1]}$
 - We compute this differently, by using random subset $D \subset \hat{b}^{[1]}, \dots, \hat{b}^{[m-1]}$ of size $(m - 1) \cdot p_{\text{drop}}$ is ignored
 - To avoid *overshot predictions* in ensemble, we scale the BLs at the end of the iteration, by $\frac{1}{|D|+1} \hat{b}^{[m]}$ and $\frac{|D|}{|D|+1} \hat{b} \quad \forall \hat{b} \in D$.
-
- $p_{\text{drop}} = 0$: Ordinary GB
 - $p_{\text{drop}} = 1$: All BLs are trained independently, and equally weighted.
Model is very similar to random forest.
- ⇒ p_{drop} is smooth transition from GB to RF

PARALLELISM AND GPU COMPUTATION

- GB is inherently sequential, not easy to parallelize
- **But:** Building of BLs can be parallelized
- Data sort and split eval in different branches of tree BLs can be computed in parallel by using efficient block data structures
- Can also gain huge speed-up by moving from CPU to GPU



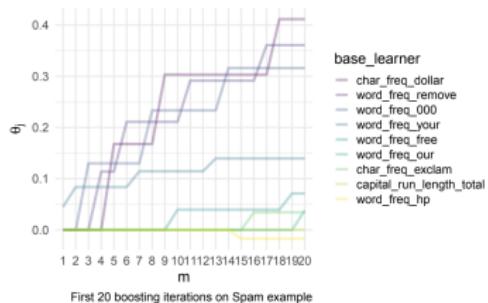
OVERVIEW OF IMPORTANT HYPERPARAMETERS

HP (as named in software)	Type	Typical Range	Trafo	Default	Description
eta	R	$[-4, 0]$	10^x	0.3	learning rate (also called ν) shrinks contribution of each boosting update
nrounds	I	$\{1, \dots, 5000\}$	-	-	number of boosting iterations. Can also be optimized with early stopping.
gamma	R	$[-7, 6]$	2^x	0	minimum loss reduction required to make a further partition on a leaf node of the tree
max_depth	I	$\{1, \dots, 20\}$	-	6	maximum depth of a tree
colsample_bytree	R	$[0.1, 1]$	-	1	subsample ratio of columns for each tree
colsample_bylevel	R	$[0.1, 1]$	-	1	subsample ratio of columns for each depth level
lambda	R	$[-10, 10]$	2^x	1	L_2 regularization term on weights
alpha	R	$[-10, 10]$	2^x	0	L_1 regularization term on weights
subsample	R	$[0.1, 1]$	-	1	subsample ratio of the training instances



Introduction to Machine Learning

Componentwise Gradient Boosting

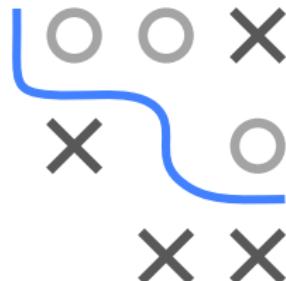


Learning goals

- Learn the concept of componentwise boosting and its relation to GLM
- Understand the built-in feature selection process
- Understand the problem of fair base learner selection

COMPONENTWISE GRADIENT BOOSTING

Gradient boosting, especially when using trees, has strong predictive performance but is difficult to interpret unless the base learners are stumps.



The aim of componentwise gradient boosting is to find a model

- has strong predictive performance,
- has components that are still interpretable,
- does automatic selection of components,
- is sparser than a model fitted with maximum-likelihood esti

This is achieved by using “nice” base learners which yield familiar statistical models in the end.

Because of this, componentwise gradient boosting is also often referred to as **model-based boosting**.

BASE LEARNERS

In classical gradient boosting only one kind of base learner is used, e.g., regression trees.



For componentwise gradient boosting we generalize this to multiple base learners

$$\{b_j^{[m]}(\mathbf{x}, \boldsymbol{\theta}^{[m]}) : j = 1, 2, \dots, J\},$$

where j indexes the type of base learner.

Again, in each iteration, only the best base learner $b_{j^*}^{[m]}(\mathbf{x}, \boldsymbol{\theta}^{[m]})$ is selected and updated.

BASE LEARNERS

We restrict these base learners to additive models, i.e.,

$$b_j^{[m]}(\mathbf{x}, \theta^{[m]}) + b_j^{[m']}(\mathbf{x}, \theta^{[m']}) = b_j(\mathbf{x}, \theta^{[m]} + \theta^{[m']}).$$



Often base learners are not defined on the entire feature vector on a single feature x_j :

$$b_j^{[m]}(x_j, \theta^{[m]}) \quad \text{for } j = 1, 2, \dots, p.$$

This directly incorporates a variable selection mechanism into the process, since in each iteration only the best base learner is selected in combination with the associated feature, and each base learner is (substantially) more complex than a stump (e.g., univariate linear effects or splines).

COMPONENTWISE BOOSTING ALGORITHM

Algorithm Componentwise Gradient Boosting.

- 1: Initialize $f^{[0]}(\mathbf{x}) = \arg \min_{\theta} \sum_{i=1}^n L(y^{(i)}, \theta)$
- 2: **for** $m = 1 \rightarrow M$ **do**
- 3: For all i : $\tilde{r}^{[m](i)} = - \left[\frac{\partial L(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f(\mathbf{x}^{(i)})} \right]_{f=f^{[m-1]}}$
- 4: **for** $j = 1 \rightarrow J$ **do**
- 5: Fit regression base learner b_j to the pseudo-residuals $\tilde{r}^{[m](i)}$:
- 6: $\hat{\theta}_j^{[m]} = \arg \min_{\theta_j} \sum_{i=1}^n (\tilde{r}^{[m](i)} - b_j(\mathbf{x}^{(i)}, \theta_j))^2$
- 7: **end for**
- 8: $j^* = \arg \min_j \sum_{i=1}^n (\tilde{r}^{[m](i)} - b_j(\mathbf{x}^{(i)}, \hat{\theta}_j^{[m]}))^2$
- 9: Update $f^{[m]}(\mathbf{x}) = f^{[m-1]}(\mathbf{x}) + \nu b_{j^*}(\mathbf{x}, \hat{\theta}_{j^*}^{[m]})$
- 10: **end for**
- 11: Output $\hat{f}(\mathbf{x}) = f^{[M]}(\mathbf{x})$

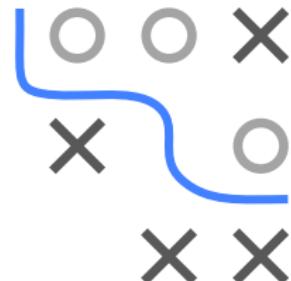


RELATION TO GLM

In the simplest case we use linear models (without intercept) or features as base learners:

$$b(x_j, \theta^{[m]}) = x_j \theta^{[m]} \quad \text{for } j = 1, 2, \dots, p.$$

This definition will result in an ordinary **linear regression** model that linear base learners without intercept only make sense for covariates that have been centered before).



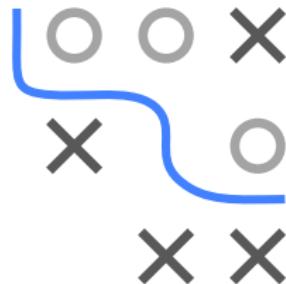
- If we let the boosting algorithm converge, i.e., let it run for a long time, the parameters estimated by the boosting mode converge to the **same solution** as the ML estimate.
- This means that, by specifying a loss function according to negative likelihood of a distribution from an exponential family defining a link function accordingly, this kind of boosting is equivalent to a (regularized) **generalized linear model (GLM)**.

RELATION TO GLM

But: We do not *need* an exponential family and thus are able to model to all kinds of other distributions and losses, as long as calculate (or approximate) a derivative of the loss.

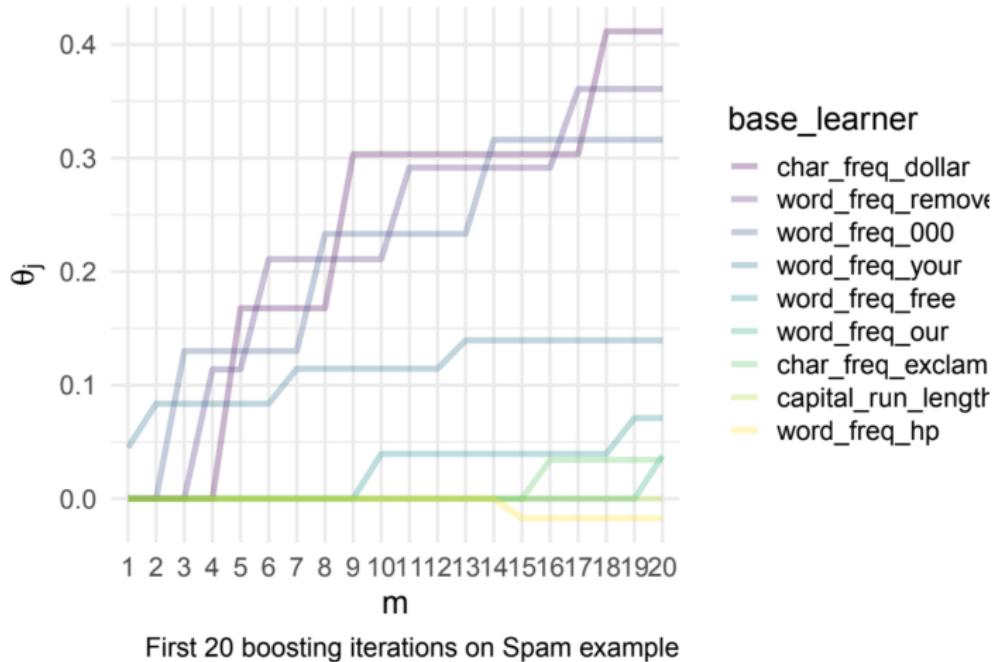
Usually we do not let the boosting model converge fully, but **stop** for the sake of regularization and feature selection.

Even though the resulting model looks like a GLM, we do not have standard errors for our coefficients, so cannot provide confidence prediction intervals or perform tests etc. → post-selection inference



FEATURE SELECTION

We can visualize the updates of the θ_j together with the number of iterations to see which base learner was selected when.



FEATURE SELECTION

- We can further exploit the additive structure of the boosted ensemble to compute measures of **variable importance**.
- To this end, we simply sum for each feature x_j the improved empirical risk achieved over all iterations until $1 < m_{\text{stop}} \leq$

$$VI_j = \sum_{m=1}^{m_{\text{stop}}} \left(\mathcal{R}_{\text{emp}} \left(f^{[m-1]}(\mathbf{x}) \right) - \mathcal{R}_{\text{emp}} \left(f^{[m]}(\mathbf{x}) \right) \right) \cdot \mathbb{I}_{j \in s}$$

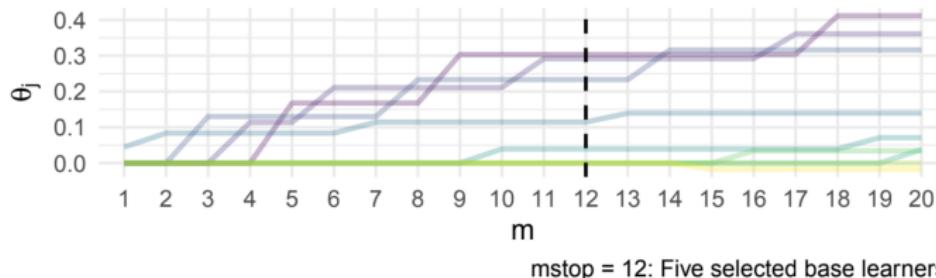
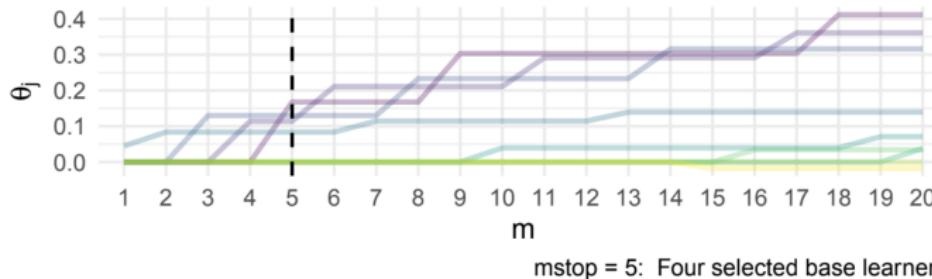
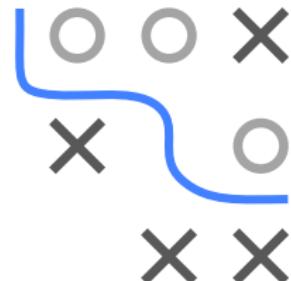
where $s(m)$ denotes the index set of features selected in m -th iteration.



FEATURE SELECTION

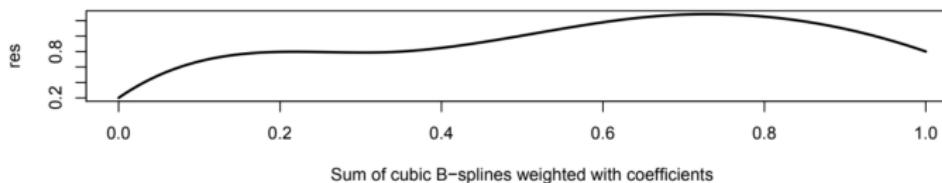
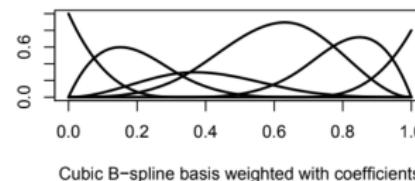
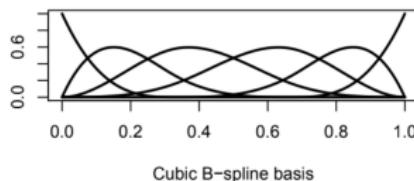
The number of features effectively included in the final model depends on the value of M .

→ A sparse logistic regression is fitted.



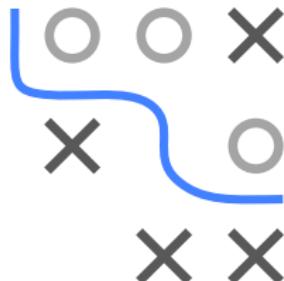
NONLINEAR BASE LEARNERS

Nonlinear base learners like P - or B -splines make the model equivalent to a **generalized additive model (GAM)**, as long as the base learners keep their additive structure (which is the case for splines).



NONLINEAR BASE LEARNERS

Even when allowing for more complexity we typically want to keep solutions as simple as possible.



Kneib et al. (2009) proposed a decomposition of each base learner into a constant, a linear and a nonlinear part. The boosting algorithm automatically decide which feature to include – linear, nonlinear or none at all:

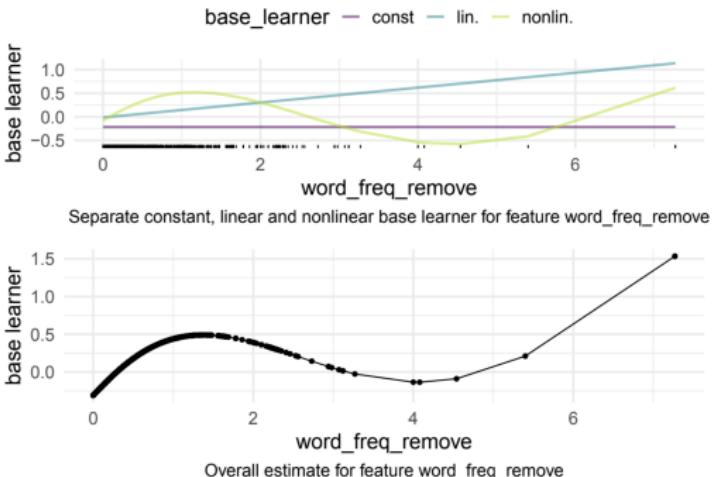
$$\begin{aligned} b_j(x_j, \theta^{[m]}) &= b_{j,\text{const}}(x_j, \theta^{[m]}) + b_{j,\text{lin}}(x_j, \theta^{[m]}) + b_{j,\text{nonlin}}(x_j, \theta^{[m]}) \\ &= \theta_{\text{const}}^{[m]} + x_j \cdot \theta_{j,\text{lin}}^{[m]} + s_j(x_j, \theta_{j,\text{nonlin}}^{[m]}), \end{aligned}$$

where

- θ_{const} is a constant term over all base learners,
- $x_j \cdot \theta_{j,\text{lin}}^{[m]}$ is a feature-specific linear base learner, and
- $s_j(x_j, \theta_{j,\text{nonlin}}^{[m]})$ is a (centered) nonlinear base learner capturing deviations from the linear effect.

SPAM EXAMPLE: CONTINUED

- This time we fit a componentwise gradient boosting model spam dataset using the R package mboost with $M = 100$.
- We specify a linear and nonlinear (P -spline) base learner for each feature and let the boosting model decide which to select.
- The model selects 17 different base learners (out of 114):



FAIR BASE LEARNER SELECTION

- Using splines and linear base learners in componentwise I will favor the more complex spline base learner over the linear base learner.
- This makes it harder to achieve the desired behavior of the learner decomposition as explained previously.
- To conduct a fair base learner selection we set the degrees of freedom of all base learners equal.
- The idea is to set the regularization/penalty term of a single learner in a manner that their complexity is treated equally.

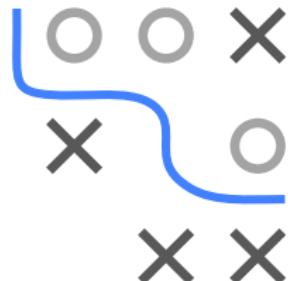


FAIR BASE LEARNER SELECTION

Especially for linear models and GAMs it is possible to transform degrees of freedom into a corresponding penalty term.

- Parameters of the base learners are estimated via:

$$\boldsymbol{\theta}_j^{[m]} = (\mathbf{Z}_j^T \mathbf{Z}_j + \lambda_j \mathbf{K}_j)^{-1} \mathbf{Z}_j^T \tilde{r}^{[m]},$$



with \mathbf{Z}_j the design matrix of the j -th base learner, λ_j the penalty term, and \mathbf{K}_j the penalty matrix.

- Having that kind of model, we use the hat matrix

$$\mathbf{S}_j = \mathbf{Z}_j \left(\mathbf{Z}_j^T \mathbf{Z}_j + \lambda_j \mathbf{K}_j \right)^{-1} \mathbf{Z}_j^T$$
 to define the degrees of freedom

$$df(\lambda_j) = \text{tr} (2\mathbf{S}_j - \mathbf{S}_j^T \mathbf{S}_j).$$

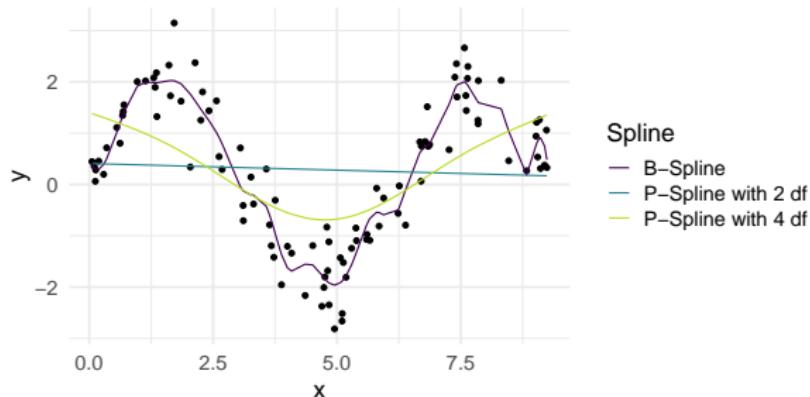
Note: With $\lambda_j = 0$, \mathbf{S}_j is the projection matrix into the target space with $\text{tr}(\mathbf{S}_j) = \text{rank}(\mathbf{X})$, which corresponds to the number of parameters in the model.

FAIR BASE LEARNER SELECTION

It is possible to calculate λ_j by applying the Demmler-Reinsch orthogonalization (see Hofer et al. (2011)).

Consider the following example of a GAM using splines with 24 parameters:

- Setting $df = 24$ gives B -splines with $\lambda_j = 0$.
- Setting $df = 4$ gives P -splines with $\lambda_j = 418$.
- Setting $df = 2$ gives P -splines with $\lambda_j = 42751174892$.



AVAILABLE BASE LEARNERS

There is a large amount of possible base learners, e.g.:

- Linear effects and interactions (with or without intercept)
- Uni- or multivariate splines and tensor product splines
- Trees
- Random effects and Markov random fields
- Effects of functional covariates
- ...



In combination with the flexible choice of loss functions, this allows boosting to fit a huge number of different statistical models with the same algorithm. Recent extensions include GAMLSS-models, where multiple additive predictors are boosted to model different distributional parameters (e.g., conditional mean and variance for a Gaussian).

TAKE-HOME MESSAGE

- Componentwise gradient boosting is the statistical re-interpretation of gradient boosting.
- We can fit a large number of statistical models, even in high dimensions ($p \gg n$).
- A drawback compared to statistical models is that we do not have valid inference for coefficients → post-selection inference.
- In most cases, gradient boosting with trees will dominate componentwise boosting in terms of performance due to its inherent ability to include higher-order interaction terms.



INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

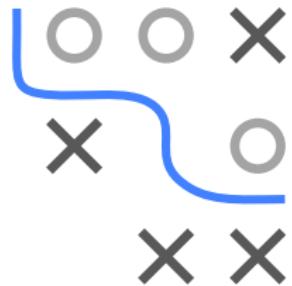
Boosting

Gaussian Processes

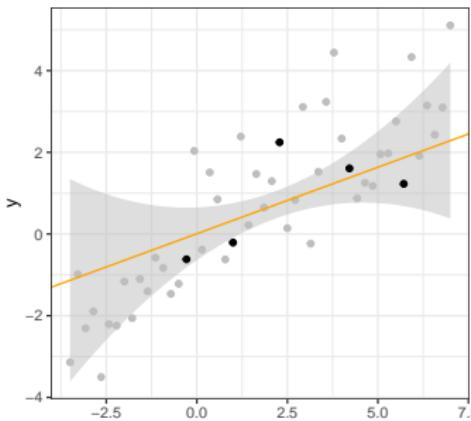


Introduction to Machine Learning

Gaussian Processes Bayesian Linear Model



MAP after observing 5 data points

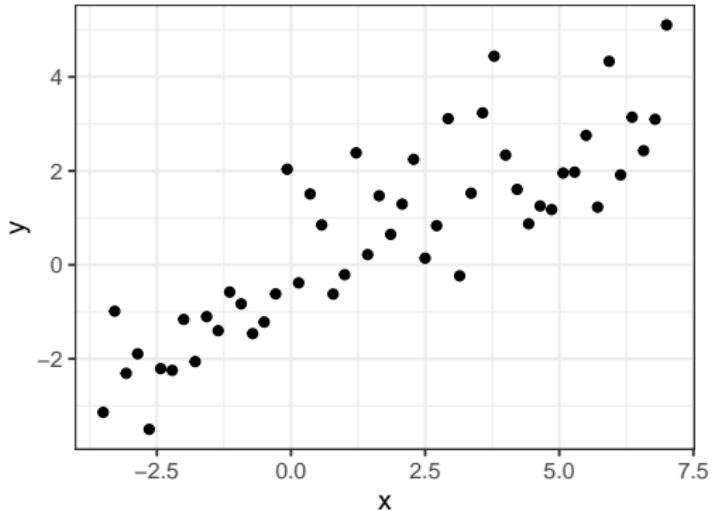


Learning goals

- Know the Bayesian linear model
- The Bayesian LM returns a (posterior) distribution instead of a point estimate
- Know how to derive the posterior distribution for a Bayesian LM

REVIEW: THE BAYESIAN LINEAR MODEL

Let $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})\}$ be a training set of i.i.d. observations from some unknown distribution.



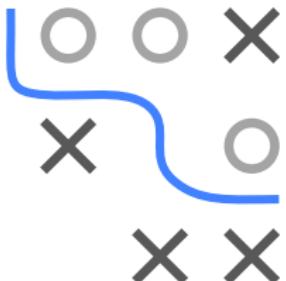
Let $\mathbf{y} = (y^{(1)}, \dots, y^{(n)})^\top$ and $\mathbf{X} \in \mathbb{R}^{n \times p}$ be the design matrix where the i -th row contains vector $\mathbf{x}^{(i)}$.

REVIEW: THE BAYESIAN LINEAR MODEL

The linear regression model is defined as

$$y = f(\mathbf{x}) + \epsilon = \boldsymbol{\theta}^T \mathbf{x} + \epsilon$$

or on the data:



$$y^{(i)} = f(\mathbf{x}^{(i)}) + \epsilon^{(i)} = \boldsymbol{\theta}^T \mathbf{x}^{(i)} + \epsilon^{(i)}, \quad \text{for } i \in \{1, \dots, n\}$$

We now assume (from a Bayesian perspective) that also our parameter vector $\boldsymbol{\theta}$ is stochastic and follows a distribution. The observed values $y^{(i)}$ differ from the function values $f(\mathbf{x}^{(i)})$ by some additive noise, which is assumed to be i.i.d. Gaussian

$$\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$$

and independent of \mathbf{x} and $\boldsymbol{\theta}$.

REVIEW: THE BAYESIAN LINEAR MODEL

Let us assume we have **prior beliefs** about the parameter θ that are represented in a prior distribution $\theta \sim \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I}_p)$.

Whenever data points are observed, we update the parameters' prior distribution according to Bayes' rule

$$\underbrace{p(\theta | \mathbf{X}, \mathbf{y})}_{\text{posterior}} = \frac{\overbrace{p(\mathbf{y} | \mathbf{X}, \theta)}^{\text{likelihood}} \overbrace{q(\theta)}^{\text{prior}}}{\underbrace{p(\mathbf{y} | \mathbf{X})}_{\text{marginal}}}.$$



REVIEW: THE BAYESIAN LINEAR MODEL

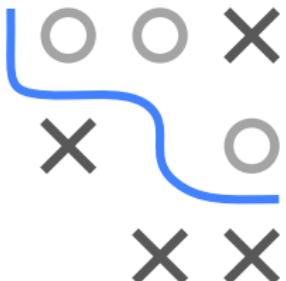
The posterior distribution of the parameter θ is again normal distributed (the Gaussian family is self-conjugate):

$$\theta | \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\sigma^{-2} \mathbf{A}^{-1} \mathbf{X}^\top \mathbf{y}, \mathbf{A}^{-1})$$

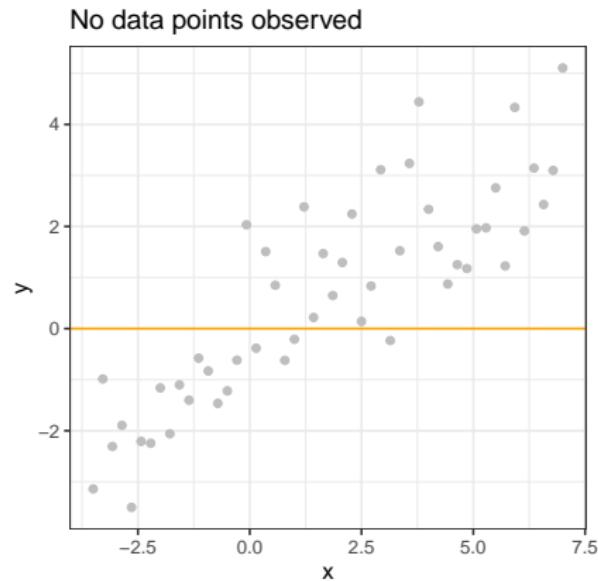
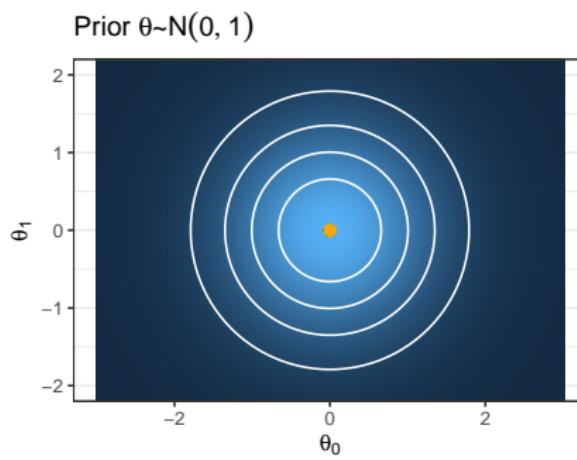
with $\mathbf{A} := \sigma^{-2} \mathbf{X}^\top \mathbf{X} + \frac{1}{\tau^2} \mathbf{I}_p$.

Note: If the posterior distribution $p(\theta | \mathbf{X}, \mathbf{y})$ are in the same probability distribution family as the prior $q(\theta)$ w.r.t. a specific likelihood function $p(\mathbf{y} | \mathbf{X}, \theta)$, they are called **conjugate distributions**. The prior is then called a **conjugate prior** for the likelihood.

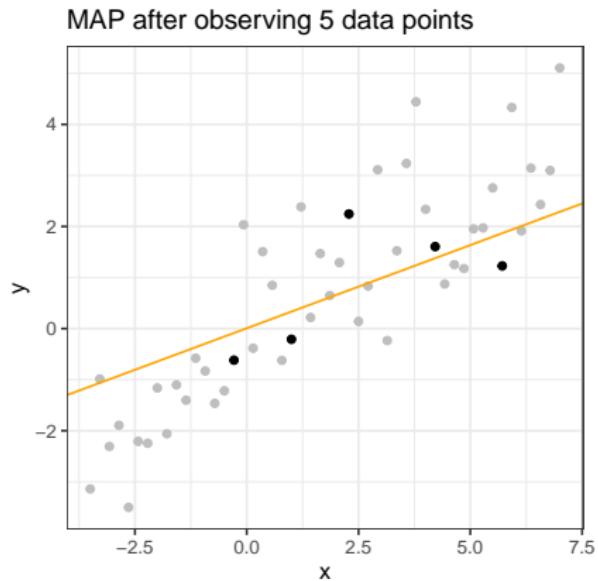
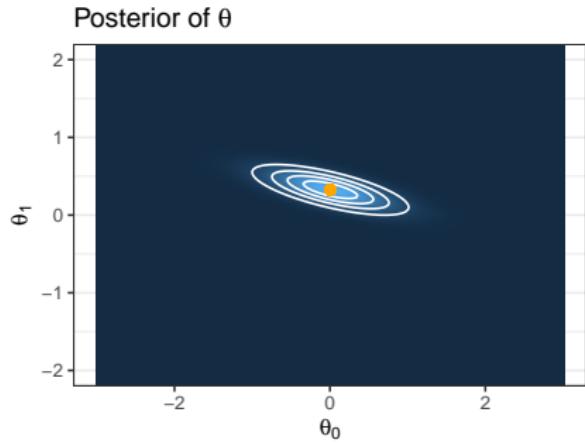
The Gaussian family is self-conjugate: Choosing a Gaussian prior for a Gaussian Likelihood ensures that the posterior is Gaussian.



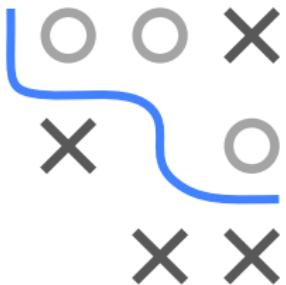
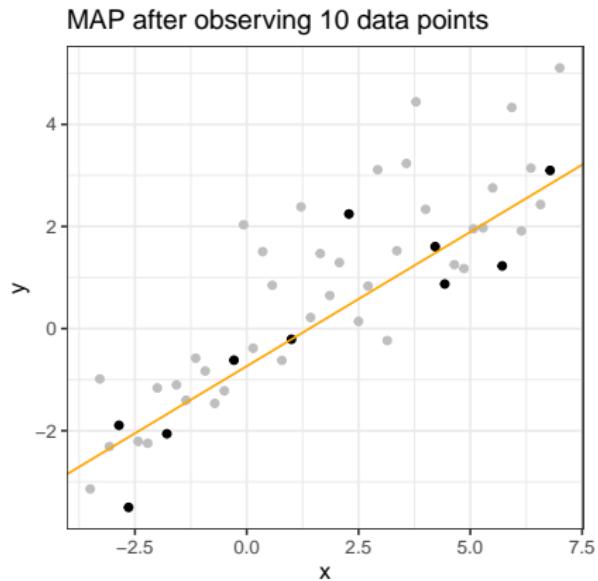
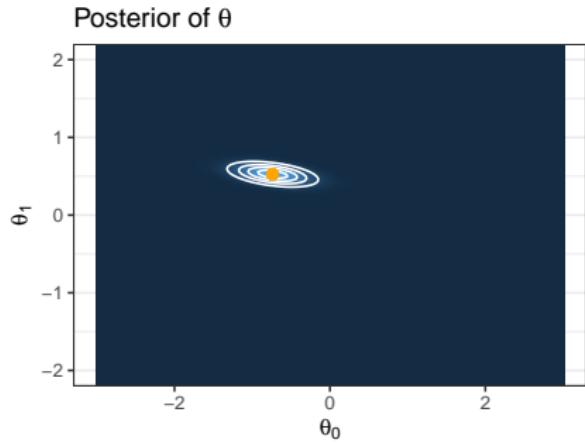
REVIEW: THE BAYESIAN LINEAR MODEL



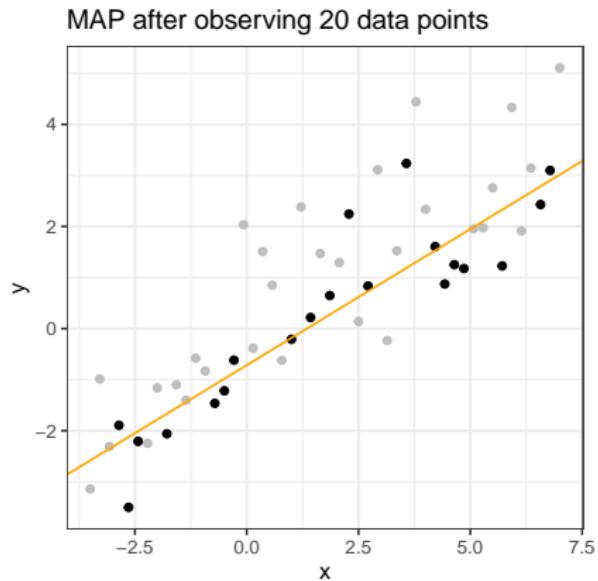
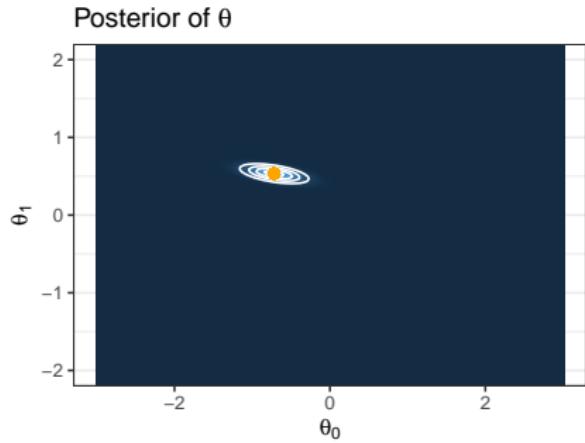
REVIEW: THE BAYESIAN LINEAR MODEL



REVIEW: THE BAYESIAN LINEAR MODEL



REVIEW: THE BAYESIAN LINEAR MODEL



REVIEW: THE BAYESIAN LINEAR MODEL

Proof:

We want to show that

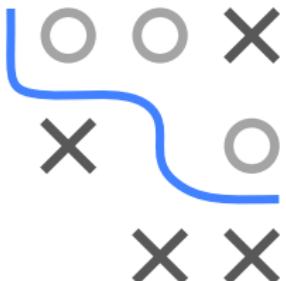
- for a Gaussian prior on $\theta \sim \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I}_p)$
- for a Gaussian Likelihood $y | \mathbf{X}, \theta \sim \mathcal{N}(\mathbf{X}^\top \theta, \sigma^2 \mathbf{I}_n)$

the resulting posterior is Gaussian $\mathcal{N}(\sigma^{-2} \mathbf{A}^{-1} \mathbf{X}^\top \mathbf{y}, \mathbf{A}^{-1})$ with $\mathbf{A} := \sigma^{-2} \mathbf{X}^\top \mathbf{X} + \frac{1}{\tau^2} \mathbf{I}_p$.

Plugging in Bayes' rule and multiplying out yields

$$\begin{aligned} p(\theta | \mathbf{X}, \mathbf{y}) &\propto p(\mathbf{y} | \mathbf{X}, \theta) q(\theta) \propto \exp \left[-\frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{X}\theta)^\top (\mathbf{y} - \mathbf{X}\theta) - \frac{1}{2\tau^2} \theta^\top \theta \right] \\ &= \exp \left[-\frac{1}{2} \left(\underbrace{\sigma^{-2} \mathbf{y}^\top \mathbf{y}}_{\text{doesn't depend on } \theta} - 2\sigma^{-2} \mathbf{y}^\top \mathbf{X}\theta + \sigma^{-2} \theta^\top \mathbf{X}^\top \mathbf{X}\theta + \tau^{-2} \theta^\top \theta \right) \right] \\ &\propto \exp \left[-\frac{1}{2} \left(\sigma^{-2} \theta^\top \mathbf{X}^\top \mathbf{X}\theta + \tau^{-2} \theta^\top \theta - 2\sigma^{-2} \mathbf{y}^\top \mathbf{X}\theta \right) \right] \\ &= \exp \left[-\frac{1}{2} \theta^\top \underbrace{\left(\sigma^{-2} \mathbf{X}^\top \mathbf{X} + \tau^{-2} \mathbf{I}_p \right)}_{:= \mathbf{A}} \theta + \cancel{\sigma^{-2} \mathbf{y}^\top \mathbf{X}\theta} \right] \end{aligned}$$

This expression resembles a normal density - except for the term in red!



REVIEW: THE BAYESIAN LINEAR MODEL

Note: We need not worry about the normalizing constant since its mere role is to convert probability functions to density functions with a total probability of one. We subtract a (not yet defined) constant c while compensating for this change by adding the respective terms (“adding 0”), emphasized in green:

$$\begin{aligned} p(\theta | \mathbf{X}, \mathbf{y}) &\propto \exp \left[-\frac{1}{2} (\theta - c)^T \mathbf{A} (\theta - c) - c^T \mathbf{A} \theta + \underbrace{\frac{1}{2} c^T \mathbf{A} c}_{\text{doesn't depend on } \theta} + \sigma^{-2} \mathbf{y}^T \mathbf{X} \theta \right] \\ &\propto \exp \left[-\frac{1}{2} (\theta - c)^T \mathbf{A} (\theta - c) - c^T \mathbf{A} \theta + \sigma^{-2} \mathbf{y}^T \mathbf{X} \theta \right] \end{aligned}$$

If we choose c such that $-c^T \mathbf{A} \theta + \sigma^{-2} \mathbf{y}^T \mathbf{X} \theta = 0$, the posterior is normal with mean c and covariance matrix \mathbf{A}^{-1} . Taking into account that \mathbf{A} is symmetric, this is if we choose

$$\begin{aligned} \sigma^{-2} \mathbf{y}^T \mathbf{X} &= c^T \mathbf{A} \\ \Leftrightarrow \sigma^{-2} \mathbf{y}^T \mathbf{X} \mathbf{A}^{-1} &= c^T \\ \Leftrightarrow c &= \sigma^{-2} \mathbf{A}^{-1} \mathbf{X}^T \mathbf{y} \end{aligned}$$

as claimed.



REVIEW: THE BAYESIAN LINEAR MODEL

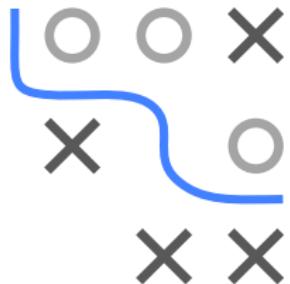
Based on the posterior distribution

$$\theta | \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\sigma^{-2} \mathbf{A}^{-1} \mathbf{X}^\top \mathbf{y}, \mathbf{A}^{-1})$$

we can derive the predictive distribution for a new observations \mathbf{x}_* . The predictive distribution for the Bayesian linear model, i.e. the distribution of $\theta^\top \mathbf{x}_*$, is

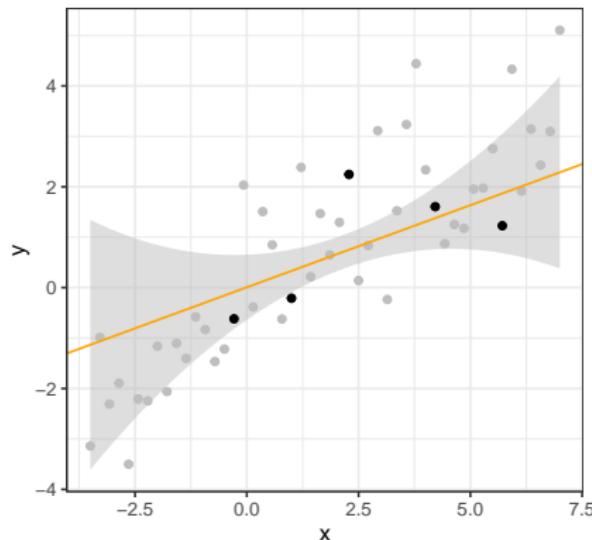
$$y_* | \mathbf{X}, \mathbf{y}, \mathbf{x}_* \sim \mathcal{N}(\sigma^{-2} \mathbf{y}^\top \mathbf{X} \mathbf{A}^{-1} \mathbf{x}_*, \mathbf{x}_*^\top \mathbf{A}^{-1} \mathbf{x}_*)$$

(applying the rules for linear transformations of Gaussians).



REVIEW: THE BAYESIAN LINEAR MODEL

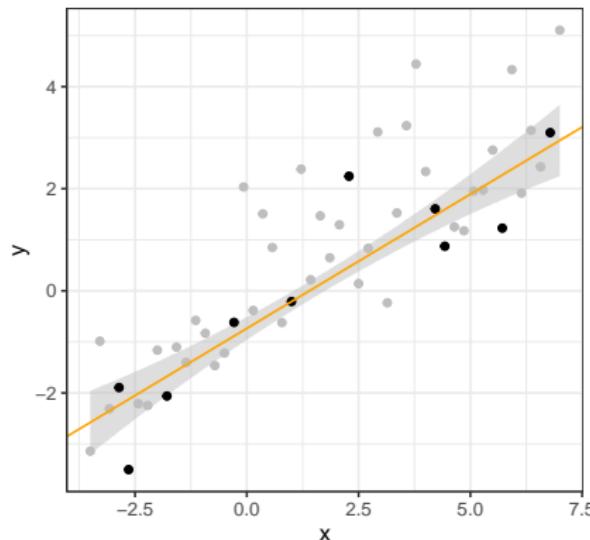
MAP after observing 5 data points



For every test input \mathbf{x}_* , we get a distribution over the prediction y_* . In particular, we get a posterior mean (orange) and a posterior variance (grey region equals $+/-$ two times standard deviation).

REVIEW: THE BAYESIAN LINEAR MODEL

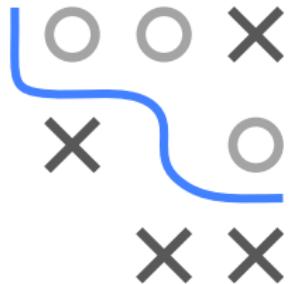
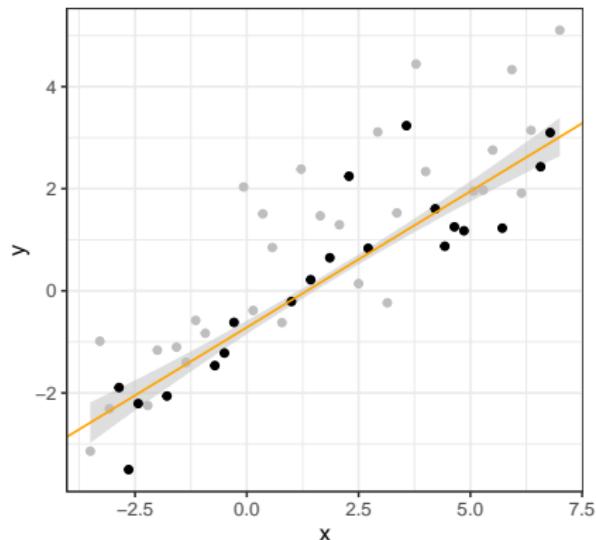
MAP after observing 10 data points



For every test input \mathbf{x}_* , we get a distribution over the prediction y_* . In particular, we get a posterior mean (orange) and a posterior variance (grey region equals $+/-$ two times standard deviation).

REVIEW: THE BAYESIAN LINEAR MODEL

MAP after observing 20 data points



For every test input \mathbf{x}_* , we get a distribution over the prediction y_* . In particular, we get a posterior mean (orange) and a posterior variance (grey region equals $+/-$ two times standard deviation).

SUMMARY: THE BAYESIAN LINEAR MODEL

- By switching to a Bayesian perspective, we do not only have point estimates for the parameter θ , but whole **distributions**
- From the posterior distribution of θ , we can derive a predictive distribution for $y_* = \theta^\top \mathbf{x}_*$.
- We can perform online updates: Whenever datapoints are observed, we can update the **posterior distribution** of θ

Next, we want to develop a theory for general shape functions, and not only for linear function.



Introduction to Machine Learning

Gaussian Processes Basics



$f(x)$

⋮

$\sim \mathcal{N}(\mu, \Sigma)$

Learning goals

- GPs model distributions over functions
- The marginalization property makes this distribution easily tractable
- GPs are fully specified by mean and covariance function
- GPs are indexed families

WEIGHT-SPACE VIEW

- Until now we considered a hypothesis space \mathcal{H} of parameterized functions $f(\mathbf{x} | \theta)$ (in particular, the space of linear functions).
- Using Bayesian inference, we derived distributions for θ after having observed data \mathcal{D} .
- Prior beliefs about the parameter are expressed via a prior distribution $q(\theta)$, which is updated according to Bayes' rule



$$\underbrace{p(\theta | \mathbf{X}, \mathbf{y})}_{\text{posterior}} = \frac{\overbrace{p(\mathbf{y} | \mathbf{X}, \theta)}^{\text{likelihood}} \overbrace{q(\theta)}^{\text{prior}}}{\underbrace{p(\mathbf{y} | \mathbf{X})}_{\text{marginal}}}.$$

FUNCTION-SPACE VIEW

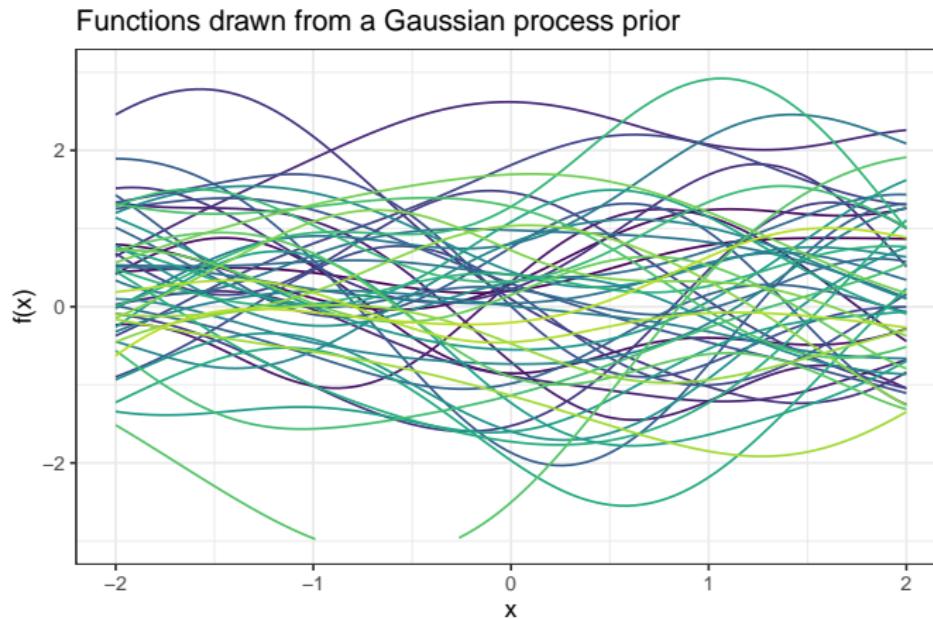
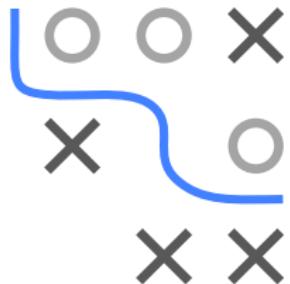
Let us change our point of view:

- Instead of “searching” for a parameter θ in the parameter space, we directly search in a space of “allowed” functions \mathcal{H} .
- We still use Bayesian inference, but instead specifying a prior distribution over a parameter, we specify a prior distribution **over functions** and update it according to the data points we have observed.



FUNCTION-SPACE VIEW

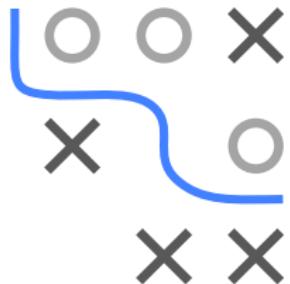
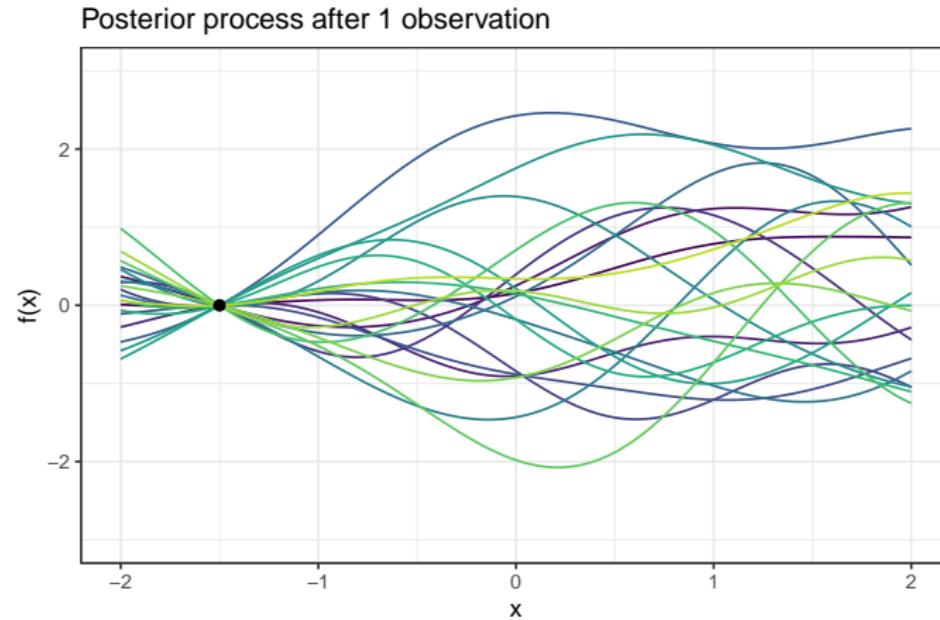
Intuitively, imagine we could draw a huge number of functions from some prior distribution over functions (*).



(*) We will see in a minute how distributions over functions can be specified.

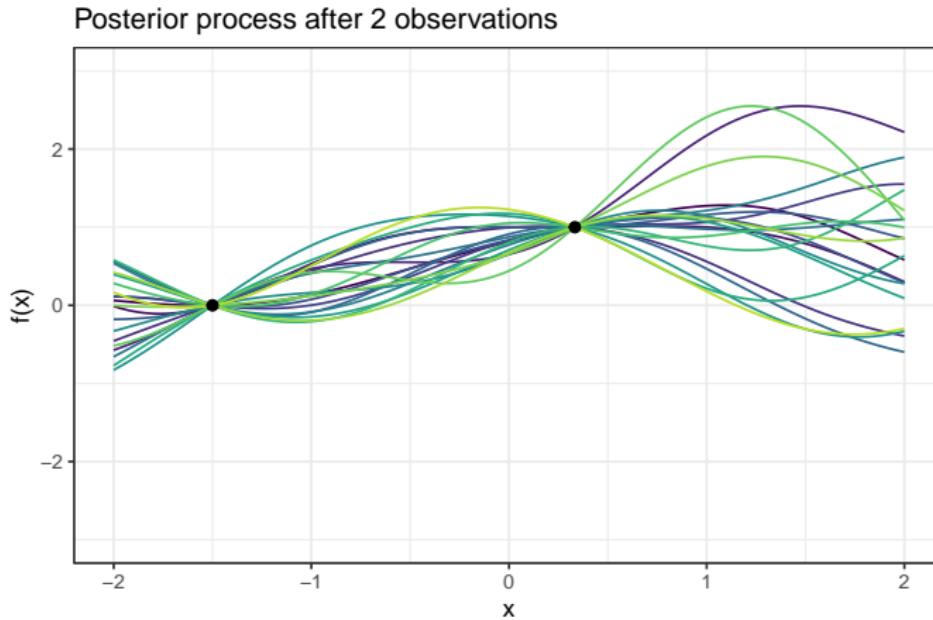
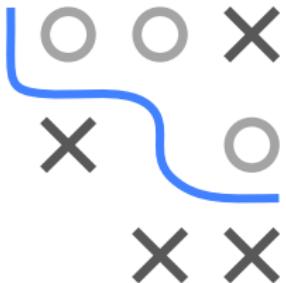
FUNCTION-SPACE VIEW

After observing some data points, we are only allowed to sample those functions, that are consistent with the data.



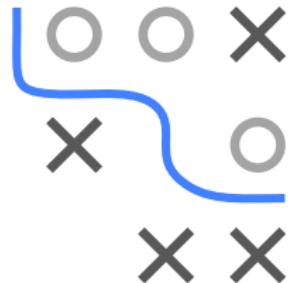
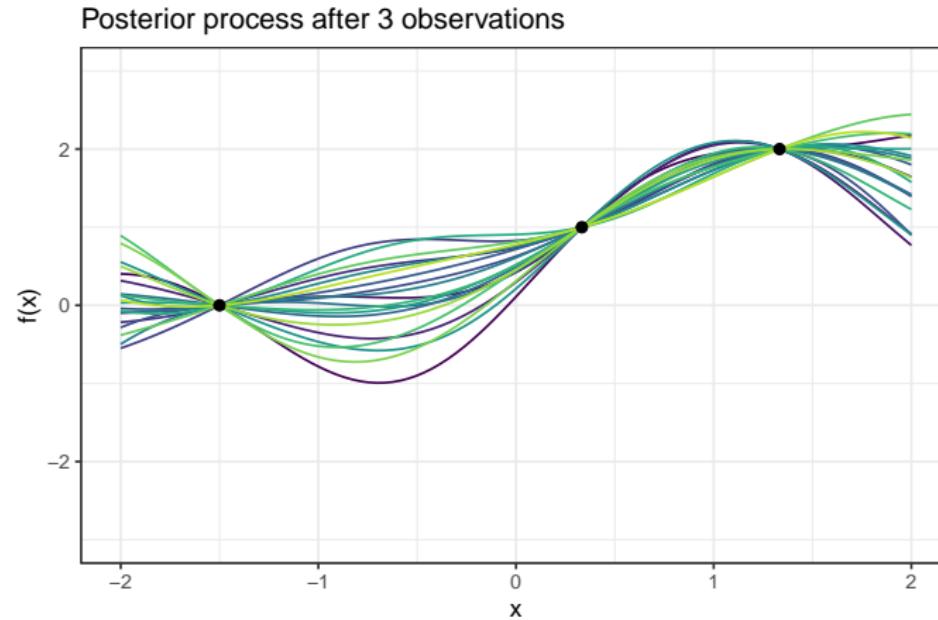
FUNCTION-SPACE VIEW

After observing some data points, we are only allowed to sample those functions, that are consistent with the data.



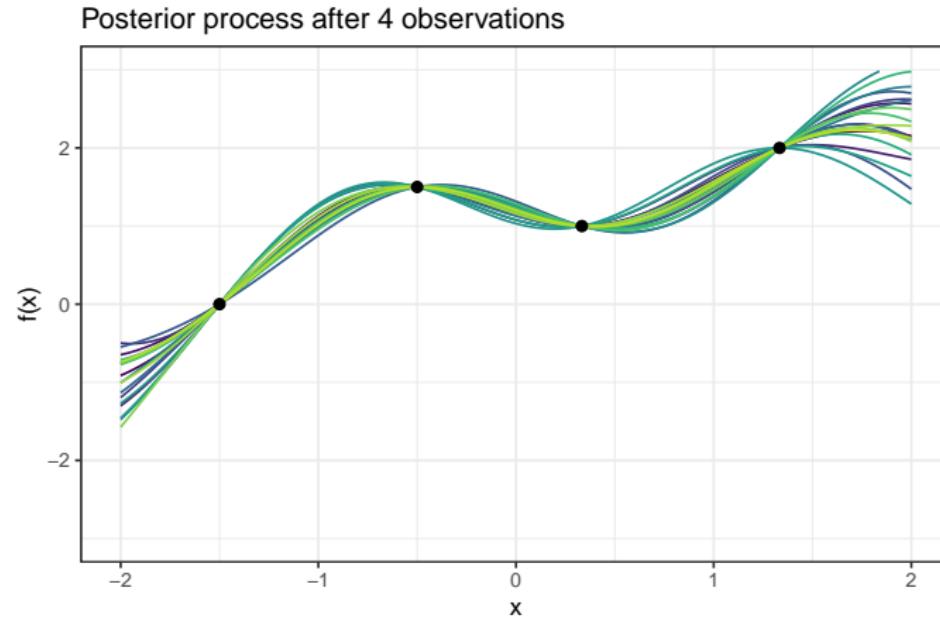
FUNCTION-SPACE VIEW

After observing some data points, we are only allowed to sample those functions, that are consistent with the data.



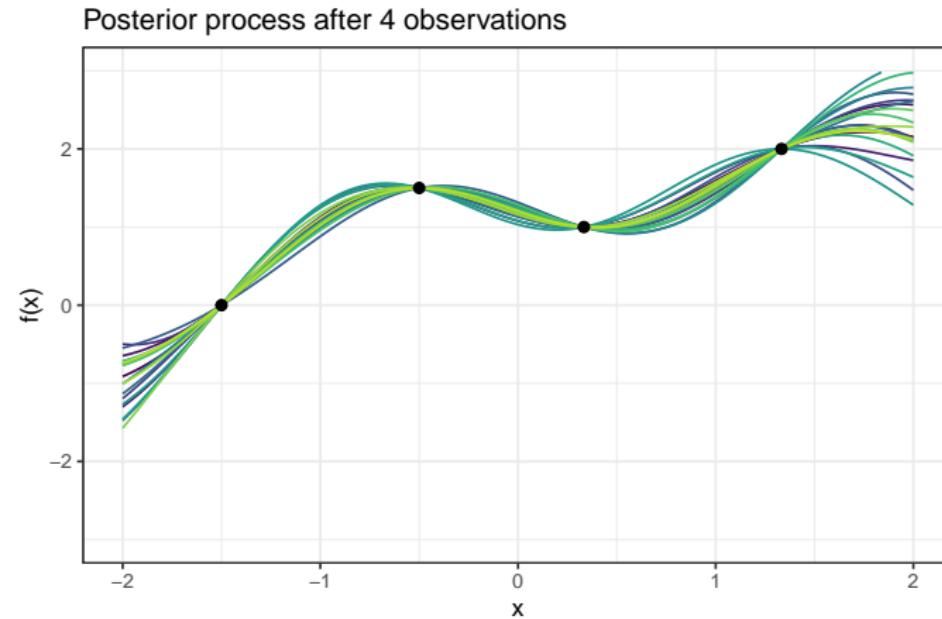
FUNCTION-SPACE VIEW

As we observe more and more data points, the variety of functions consistent with the data shrinks.



FUNCTION-SPACE VIEW

Intuitively, there is something like “mean” and a “variance” of a distribution over functions.



WEIGHT-SPACE VS. FUNCTION-SPACE VIEW

Weight-Space View

Parameterize functions

Example: $f(\mathbf{x} | \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}$

Define distributions on $\boldsymbol{\theta}$

Inference in parameter space Θ Inference in function space \mathcal{H}

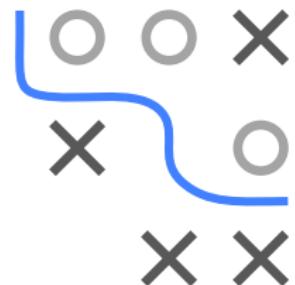
Function-Space View

Define distributions on f



Next, we will see how we can define distributions over functions mathematically.

Distributions on Functions



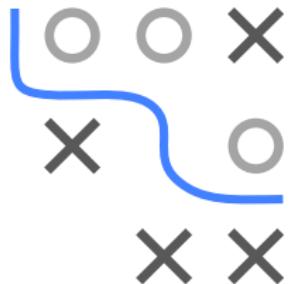
DISCRETE FUNCTIONS

For simplicity, let us consider functions with finite domains first.

Let $\mathcal{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$ be a finite set of elements and \mathcal{H} the set of all functions from $\mathcal{X} \rightarrow \mathbb{R}$.

Since the domain of any $h(\cdot) \in \mathcal{H}$ has only n elements, we can represent the function $h(\cdot)$ compactly as a n -dimensional vector

$$\mathbf{h} = [h(\mathbf{x}^{(1)}), \dots, h(\mathbf{x}^{(n)})].$$



DISCRETE FUNCTIONS

Example 1: Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **two** points $\mathcal{X} = \{0, 1\}$.

Examples for functions that live in this space:



DISCRETE FUNCTIONS

Example 1: Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **two** points $\mathcal{X} = \{0, 1\}$.

Examples for functions that live in this space:



DISCRETE FUNCTIONS

Example 1: Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **two** points $\mathcal{X} = \{0, 1\}$.

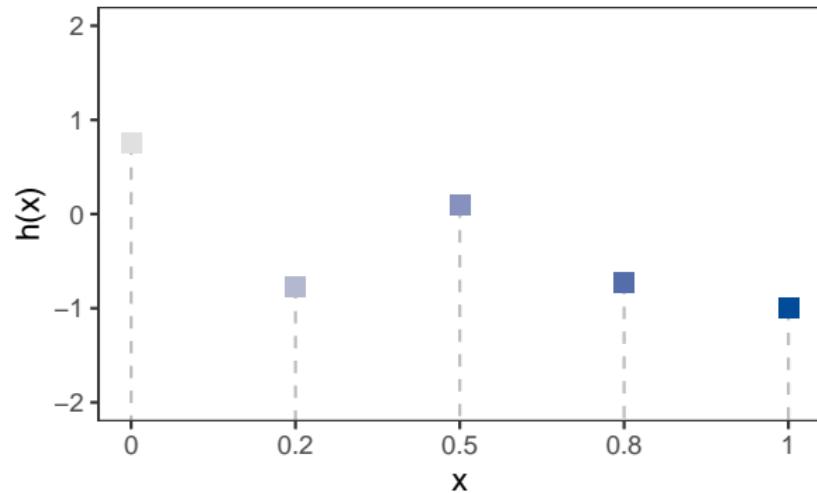
Examples for functions that live in this space:



DISCRETE FUNCTIONS

Example 2: Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points $\mathcal{X} = \{0, 0.25, 0.5, 0.75, 1\}$.

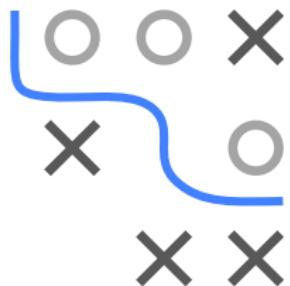
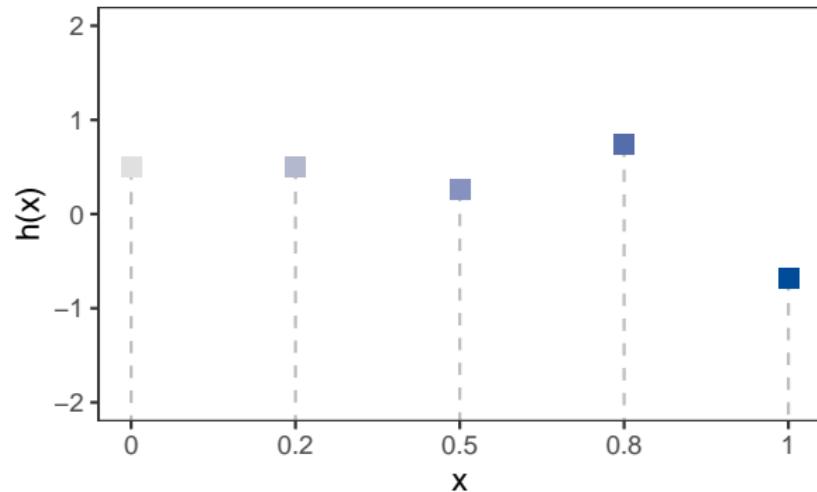
Examples for functions that live in this space:



DISCRETE FUNCTIONS

Example 2: Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points $\mathcal{X} = \{0, 0.25, 0.5, 0.75, 1\}$.

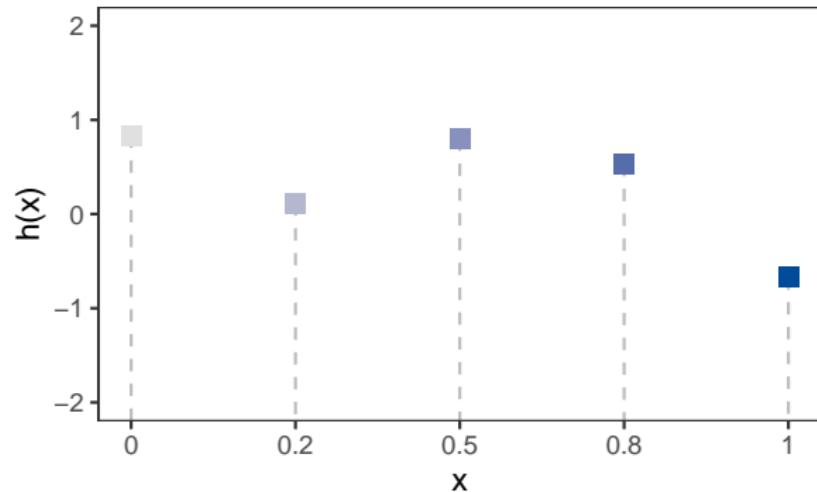
Examples for functions that live in this space:



DISCRETE FUNCTIONS

Example 2: Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points $\mathcal{X} = \{0, 0.25, 0.5, 0.75, 1\}$.

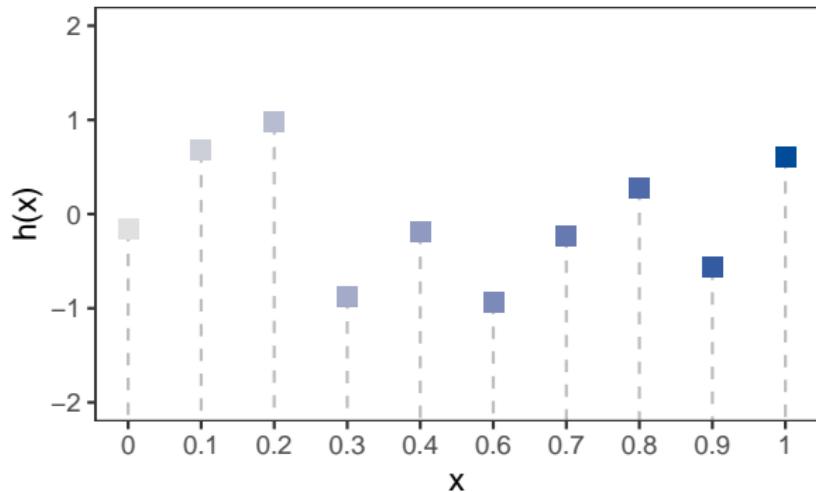
Examples for functions that live in this space:



DISCRETE FUNCTIONS

Example 3: Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points.

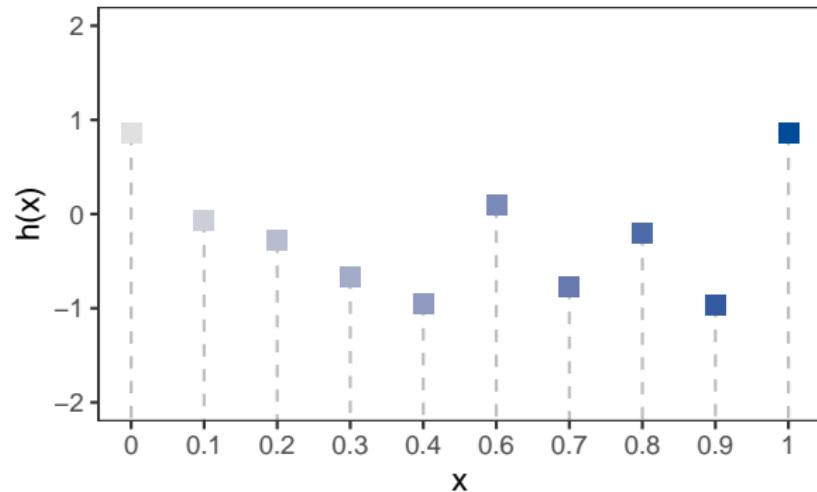
Examples for functions that live in this space:



DISCRETE FUNCTIONS

Example 3: Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points.

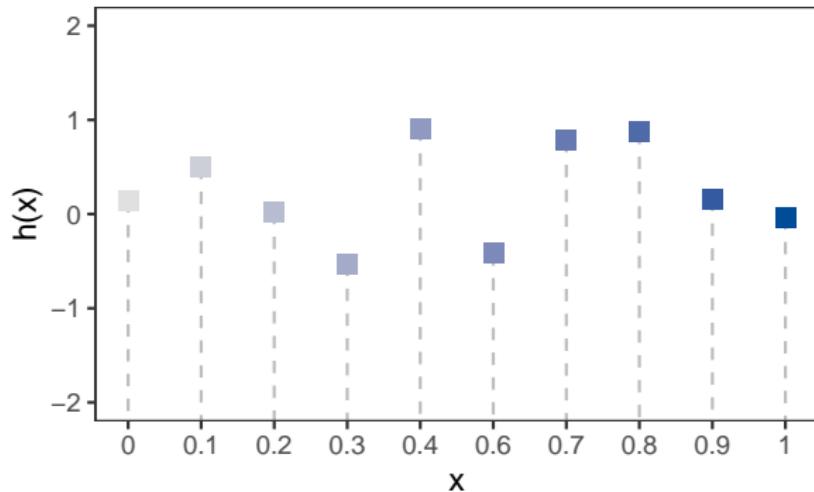
Examples for functions that live in this space:



DISCRETE FUNCTIONS

Example 3: Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points.

Examples for functions that live in this space:



DISTRIBUTIONS ON DISCRETE FUNCTIONS

One natural way to specify a probability function on discrete function $h \in \mathcal{H}$ is to use the vector representation

$$\mathbf{h} = [h(\mathbf{x}^{(1)}), h(\mathbf{x}^{(2)}), \dots, h(\mathbf{x}^{(n)})]$$

of the function.

Let us see \mathbf{h} as a n -dimensional random variable. We will further assume the following normal distribution:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$

Note: For now, we set $\mathbf{m} = \mathbf{0}$ and take the covariance matrix \mathbf{K} as given. We will see later how they are chosen / estimated.



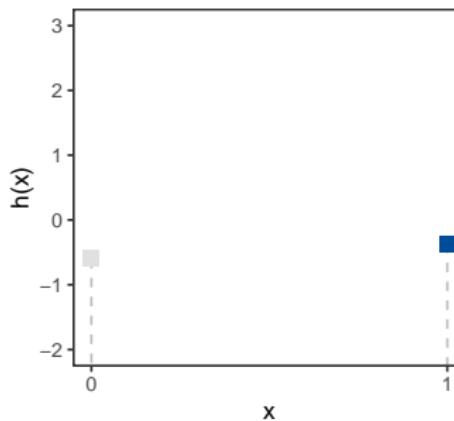
DISCRETE FUNCTIONS

Example 1 (continued): Let $h : \mathcal{X} \rightarrow \mathcal{Y}$ be a function that is defined on **two** points \mathcal{X} . We sample functions by sampling from a two-dimensional normal variable

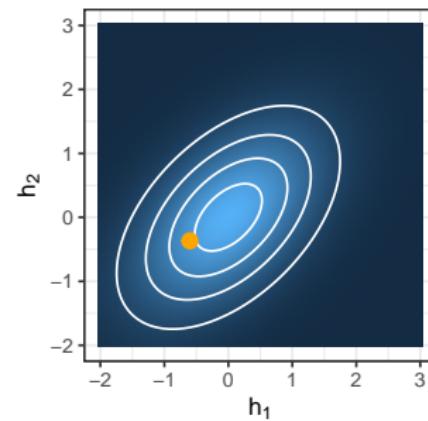
$$\mathbf{h} = [h(1), h(2)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$



Sample Function 1, n = 2



Density of a 2-D Gaussian

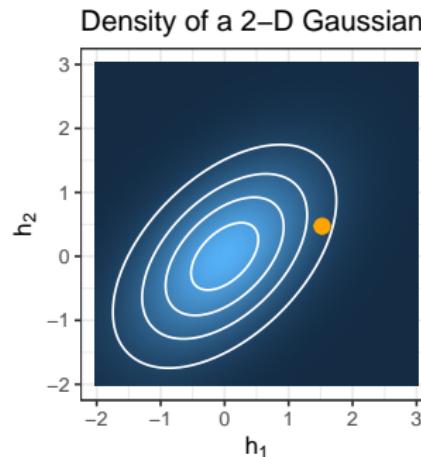
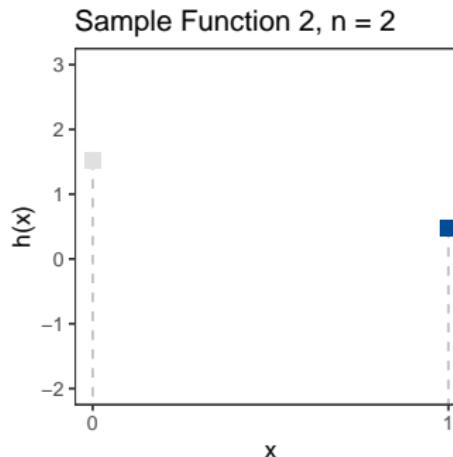


In this example, $m = (0, 0)$ and $K = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$.

DISCRETE FUNCTIONS

Example 1 (continued): Let $h : \mathcal{X} \rightarrow \mathcal{Y}$ be a function that is defined on **two** points \mathcal{X} . We sample functions by sampling from a two-dimensional normal variable

$$\mathbf{h} = [h(1), h(2)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$

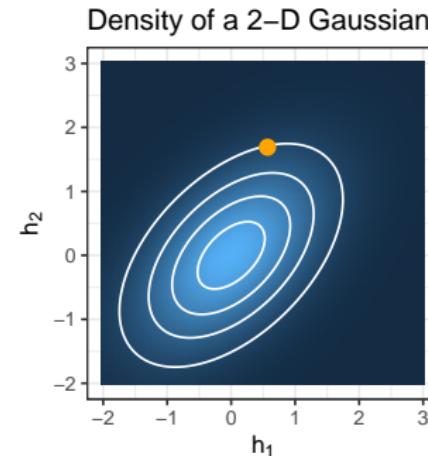
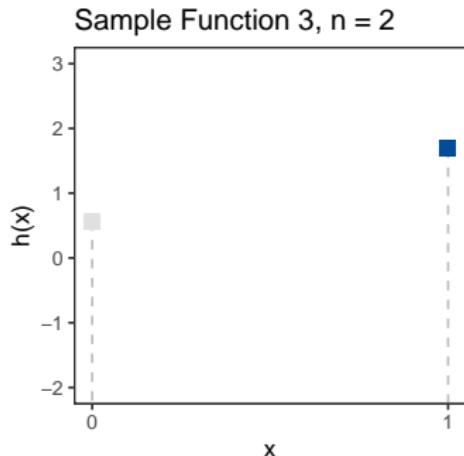


In this example, $m = (0, 0)$ and $K = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$.

DISCRETE FUNCTIONS

Example 1 (continued): Let $h : \mathcal{X} \rightarrow \mathcal{Y}$ be a function that is defined on **two** points \mathcal{X} . We sample functions by sampling from a two-dimensional normal variable

$$\mathbf{h} = [h(1), h(2)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$

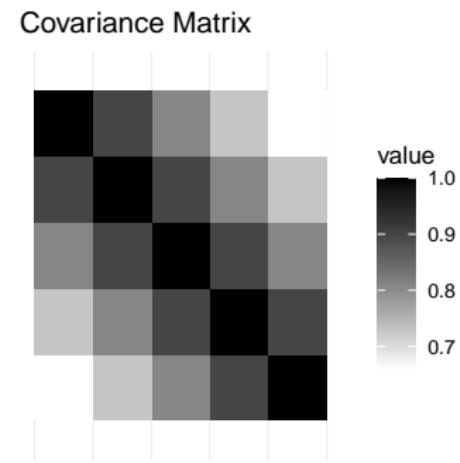
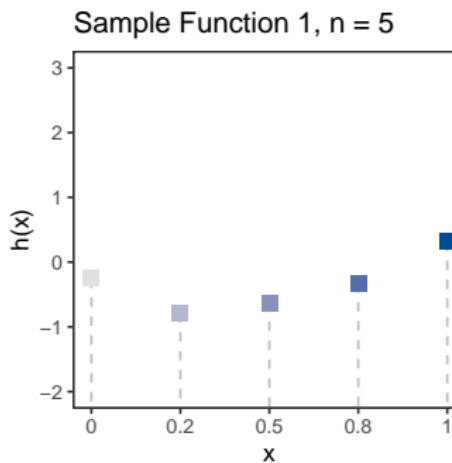


In this example, $m = (0, 0)$ and $K = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$.

DISCRETE FUNCTIONS

Example 2 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points. We sample functions by sampling from a five-dimensional normal variable

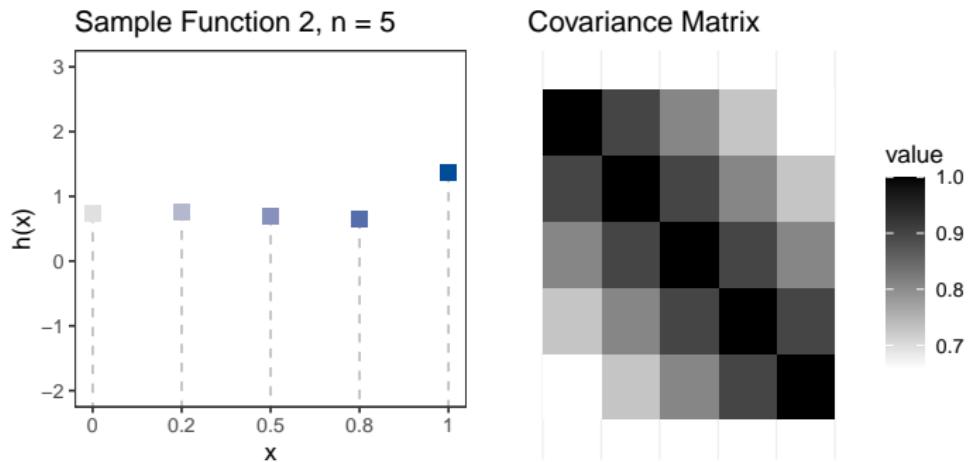
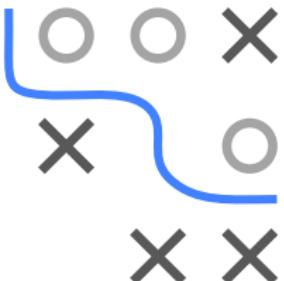
$$\mathbf{h} = [h(1), h(2), h(3), h(4), h(5)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$



DISCRETE FUNCTIONS

Example 2 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points. We sample functions by sampling from a five-dimensional normal variable

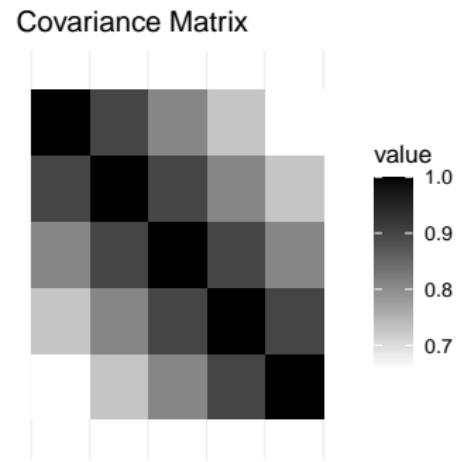
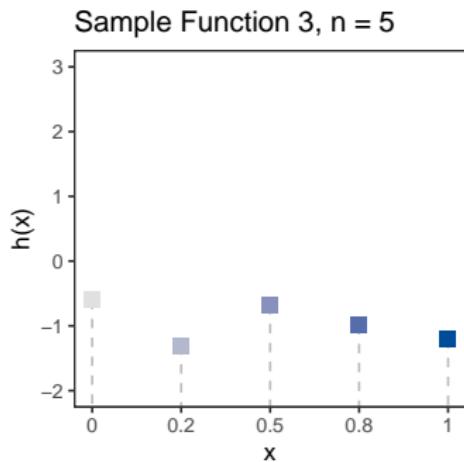
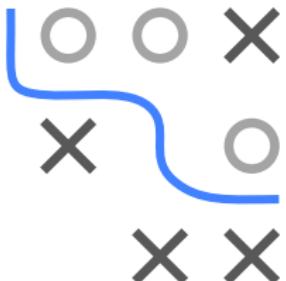
$$\mathbf{h} = [h(1), h(2), h(3), h(4), h(5)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$



DISCRETE FUNCTIONS

Example 2 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points. We sample functions by sampling from a five-dimensional normal variable

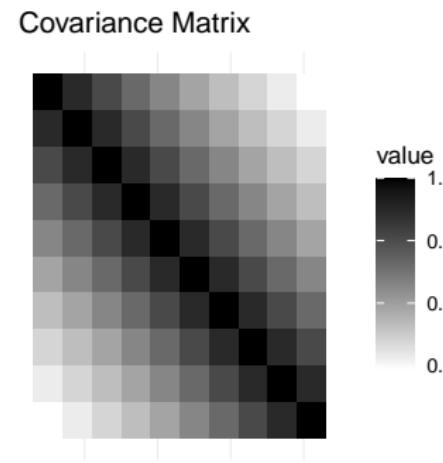
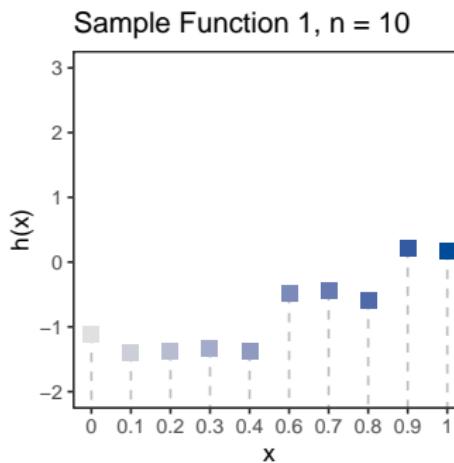
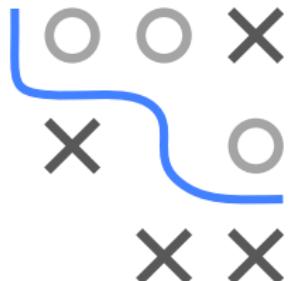
$$\mathbf{h} = [h(1), h(2), h(3), h(4), h(5)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$



DISCRETE FUNCTIONS

Example 3 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points. We sample functions by sampling from ten-dimensional normal variable

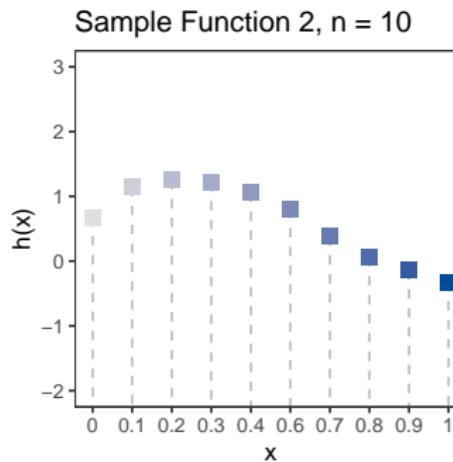
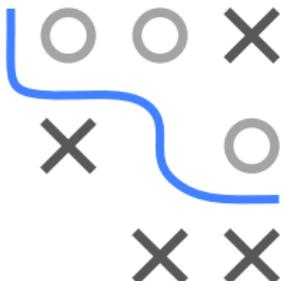
$$\mathbf{h} = [h(1), h(2), \dots, h(10)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$



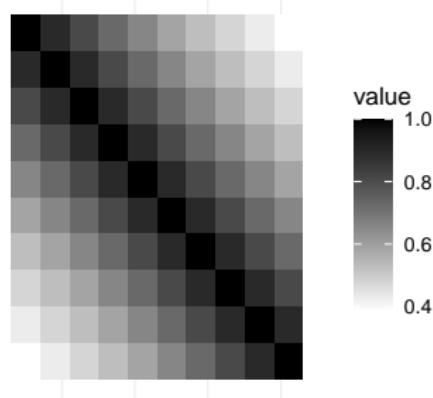
DISCRETE FUNCTIONS

Example 3 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points. We sample functions by sampling from ten-dimensional normal variable

$$\mathbf{h} = [h(1), h(2), \dots, h(10)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$



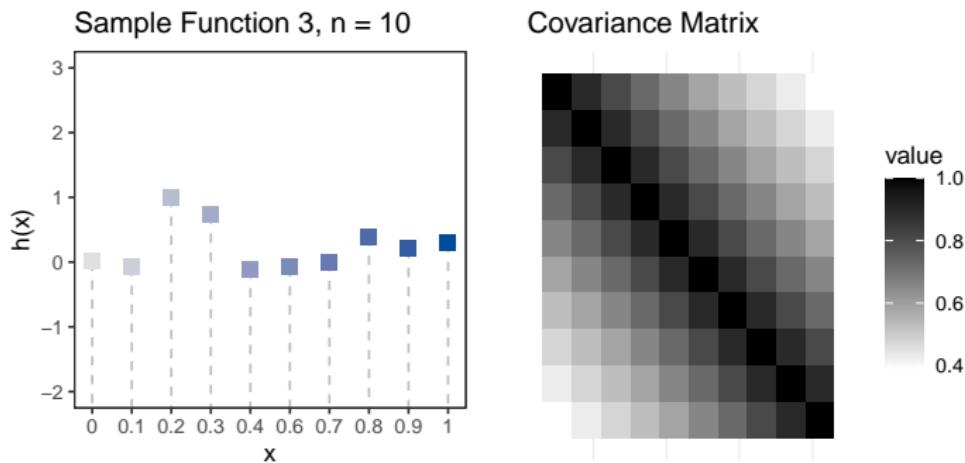
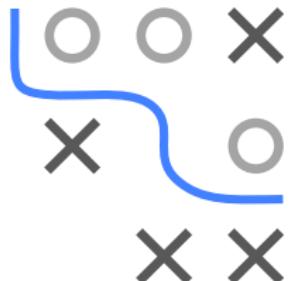
Covariance Matrix



DISCRETE FUNCTIONS

Example 3 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points. We sample functions by sampling from ten-dimensional normal variable

$$\mathbf{h} = [h(1), h(2), \dots, h(10)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$



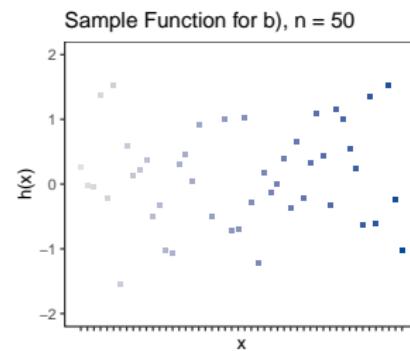
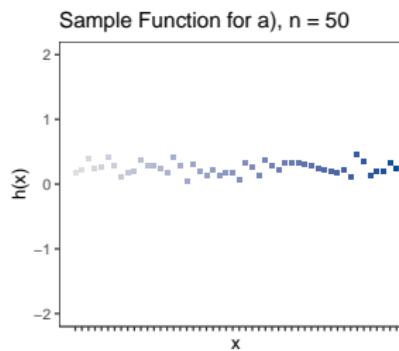
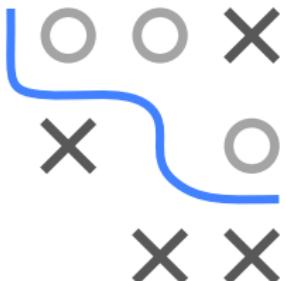
ROLE OF THE COVARIANCE FUNCTION

Note that the covariance controls the “shape” of the drawn function.

Consider two extreme cases where function values are

a) strongly correlated: $K = \begin{pmatrix} 1 & 0.99 & \dots & 0.99 \\ 0.99 & 1 & \dots & 0.99 \\ 0.99 & 0.99 & \ddots & 0.99 \\ 0.99 & \dots & 0.99 & 1 \end{pmatrix}$

b) uncorrelated: $K = I$



ROLE OF THE COVARIANCE FUNCTION

- “Meaningful” functions (on a numeric space \mathcal{X}) may be characterized by a spatial property:

If two points $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$ are close in \mathcal{X} -space, their function values $f(\mathbf{x}^{(i)}), f(\mathbf{x}^{(j)})$ should be close in \mathcal{Y} -space.

In other words: If they are close in \mathcal{X} -space, their function values should be **correlated**!

- We can enforce that by choosing a covariance function with

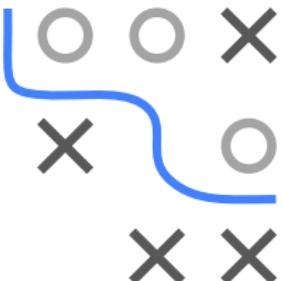
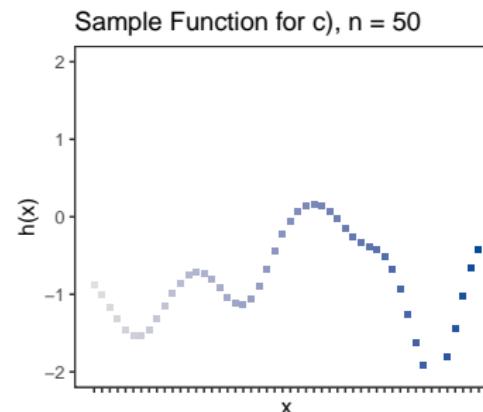
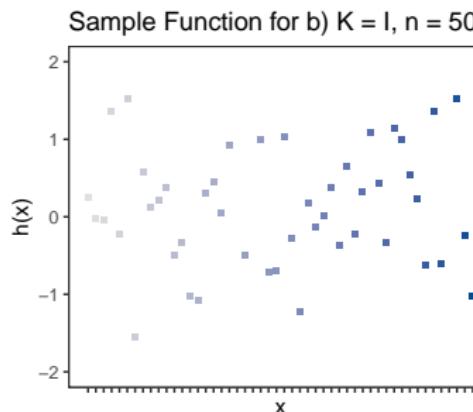
K_{ij} high, if $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$ close.



ROLE OF THE COVARIANCE FUNCTION

- We can compute the entries of the covariance matrix by a function that is based on the distance between $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$, for example:

c) Spatial correlation: $K_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\left(-\frac{1}{2}\left|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right|^2\right)$



Note: $k(\cdot, \cdot)$ is known as the **covariance function** or **kernel**. It will be studied in more detail later on.

Gaussian Processes

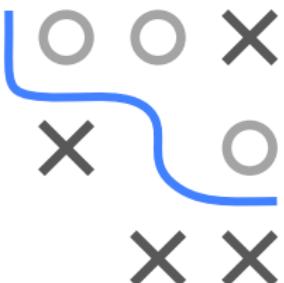
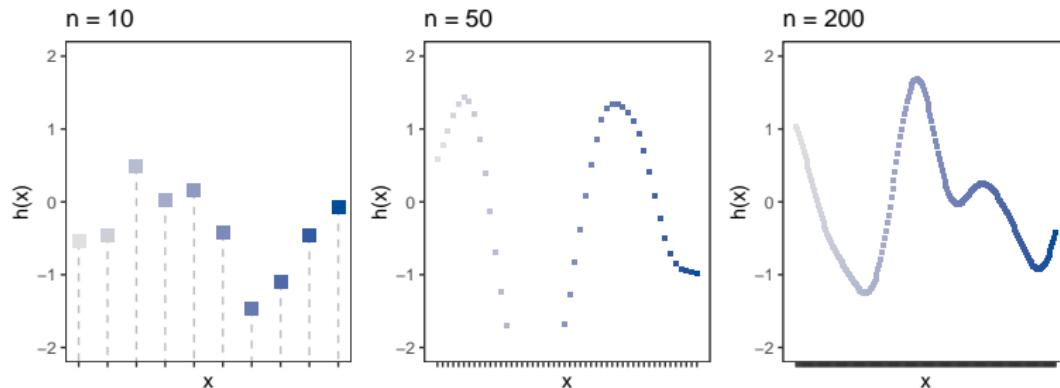


FROM DISCRETE TO CONTINUOUS FUNCTIONS

- We defined distributions on functions with discrete domain by defining a Gaussian on the vector of the respective function values

$$\mathbf{h} = [h(\mathbf{x}^{(1)}), h(\mathbf{x}^{(2)}), \dots, h(\mathbf{x}^{(n)})] \sim \mathcal{N}(\mathbf{m}, \mathbf{K})$$

- We can do this for $n \rightarrow \infty$ (as “granular” as we want)



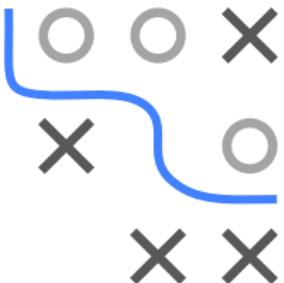
FROM DISCRETE TO CONTINUOUS FUNCTIONS

- No matter how large n is, we are still considering a function over a discrete domain.
- How can we extend our definition to functions with **continuous domain** $\mathcal{X} \subset \mathbb{R}$?



GAUSSIAN PROCESSES: INTUITION

- Intuitively, a function f drawn from **Gaussian process** can be understood as an “infinite” long Gaussian random vector.
- It is unclear how to handle an “infinite” long Gaussian random vector!



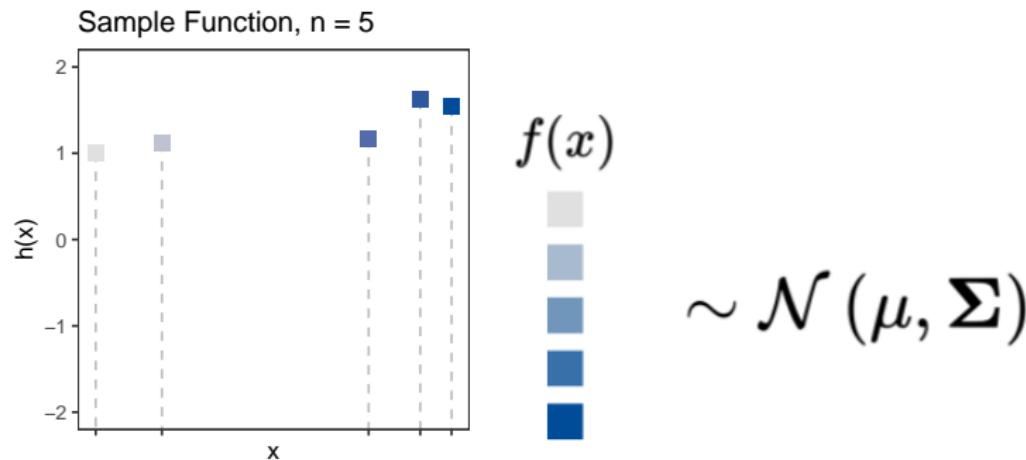
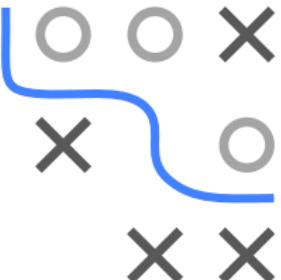
GAUSSIAN PROCESSES: INTUITION

- Thus, it is required that for **any finite set** of inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\} \subset \mathcal{X}$, the vector \mathbf{f} has a Gaussian distribution

$$\mathbf{f} = [f(\mathbf{x}^{(1)}), \dots, f(\mathbf{x}^{(n)})] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}),$$

with \mathbf{m} and \mathbf{K} being calculated by a mean function $m(\cdot)$ / covariance function $k(\cdot, \cdot)$.

- This property is called **Marginalization Property**.



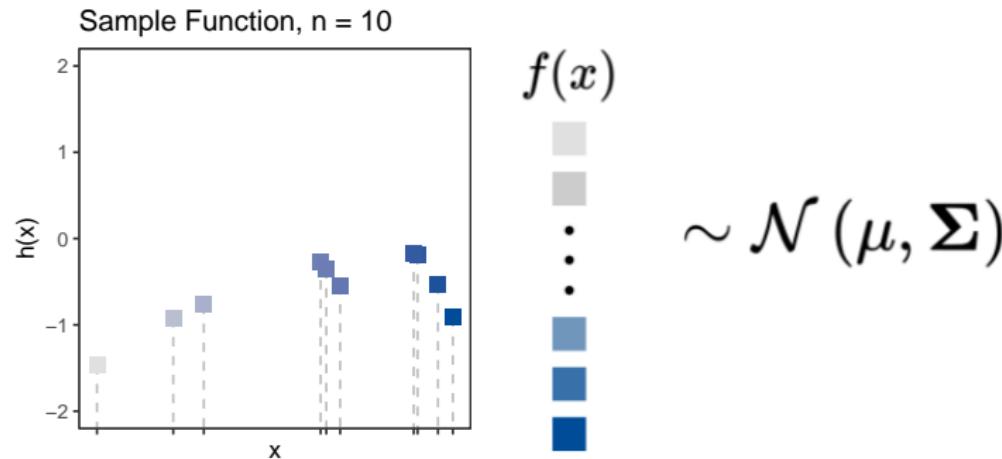
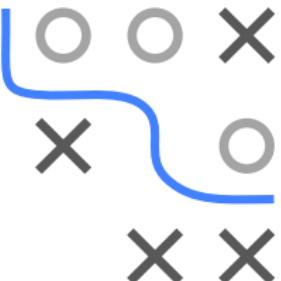
GAUSSIAN PROCESSES: INTUITION

- Thus, it is required that for **any finite set** of inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\} \subset \mathcal{X}$, the vector \mathbf{f} has a Gaussian distribution

$$\mathbf{f} = [f(\mathbf{x}^{(1)}), \dots, f(\mathbf{x}^{(n)})] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}),$$

with \mathbf{m} and \mathbf{K} being calculated by a mean function $m(\cdot)$ / covariance function $k(\cdot, \cdot)$.

- This property is called **Marginalization Property**.



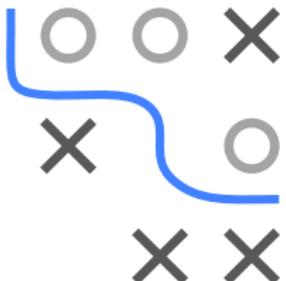
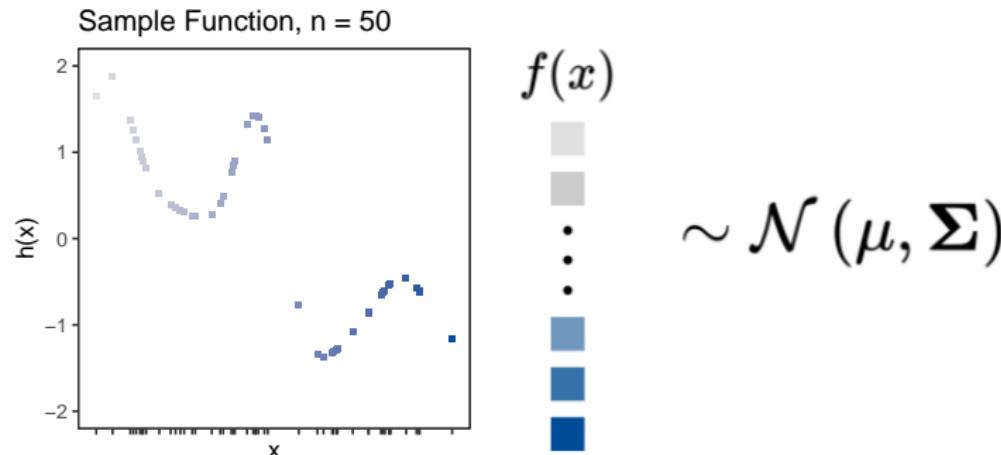
GAUSSIAN PROCESSES: INTUITION

- Thus, it is required that for **any finite set** of inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\} \subset \mathcal{X}$, the vector \mathbf{f} has a Gaussian distribution

$$\mathbf{f} = [f(\mathbf{x}^{(1)}), \dots, f(\mathbf{x}^{(n)})] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}),$$

with \mathbf{m} and \mathbf{K} being calculated by a mean function $m(\cdot)$ / covariance function $k(\cdot, \cdot)$.

- This property is called **Marginalization Property**.



GAUSSIAN PROCESSES

This intuitive explanation is formally defined as follows:

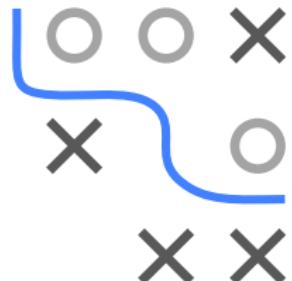
A function $f(\mathbf{x})$ is generated by a GP $\mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$ if for **any finite** set of inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$, the associated vector of function values $\mathbf{f} = (f(\mathbf{x}^{(1)}), \dots, f(\mathbf{x}^{(n)}))$ has a Gaussian distribution

$$\mathbf{f} = \left[f\left(\mathbf{x}^{(1)}\right), \dots, f\left(\mathbf{x}^{(n)}\right) \right] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}),$$

with

$$\mathbf{m} := \left(m\left(\mathbf{x}^{(i)}\right) \right)_i, \quad \mathbf{K} := \left(k\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) \right)_{i,j},$$

where $m(\mathbf{x})$ is called mean function and $k(\mathbf{x}, \mathbf{x}')$ is called covariance function.



GAUSSIAN PROCESSES

A GP is thus **completely specified** by its mean and covariance function

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}\left[(f(\mathbf{x}) - \mathbb{E}[f(\mathbf{x})]) (f(\mathbf{x}') - \mathbb{E}[f(\mathbf{x}')]) \right]$$



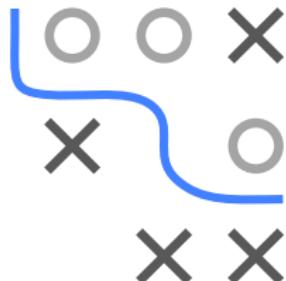
Note: For now, we assume $m(\mathbf{x}) \equiv 0$. This is not necessarily a drastic limitation - thus it is common to consider GPs with a zero mean function.

SAMPLING FROM A GAUSSIAN PROCESS PRIOR

We can draw functions from a Gaussian process prior. Let us consider $f(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$ with the squared exponential covariance function (*)

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\ell^2}\|\mathbf{x} - \mathbf{x}'\|^2\right), \quad \ell = 1.$$

This specifies the Gaussian process completely.



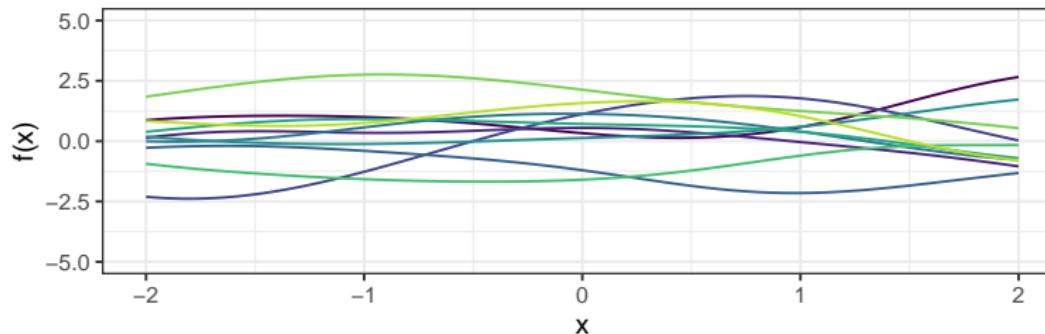
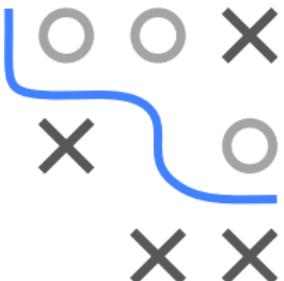
(*) We will talk later about different choices of covariance functions.

SAMPLING FROM A GAUSSIAN PROCESS PRIOR

To visualize a sample function, we

- choose a high number n (equidistant) points $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$
- compute the corresponding covariance matrix $\mathbf{K} = (k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}))_{i,j}$ by plugging in all pairs $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$
- sample from a Gaussian $\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$.

We draw 10 times from the Gaussian, to get 10 different samples.



Since we specified the mean function to be zero $m(\mathbf{x}) \equiv 0$, the drawn functions have zero mean

Gaussian Processes as Indexed Family



GAUSSIAN PROCESSES AS AN INDEXED FAMILY

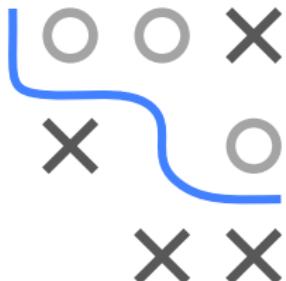
A Gaussian process is a special case of a **stochastic process** which is defined as a collection of random variables indexed by some index set (also called an **indexed family**). What does it mean?

An **indexed family** is a mathematical function (or “rule”) to map indices $t \in T$ to objects in \mathcal{S} .

Definition

A **family of elements in \mathcal{S} indexed by T** (indexed family) is a surjective function

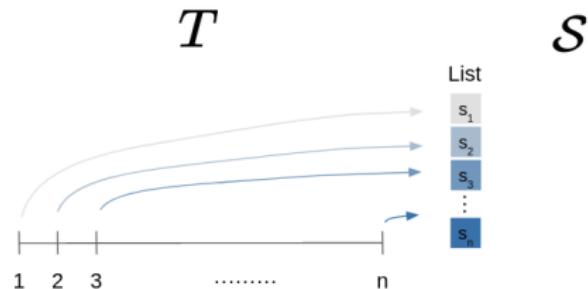
$$\begin{aligned} s : T &\rightarrow \mathcal{S} \\ t &\mapsto s_t = s(t) \end{aligned}$$



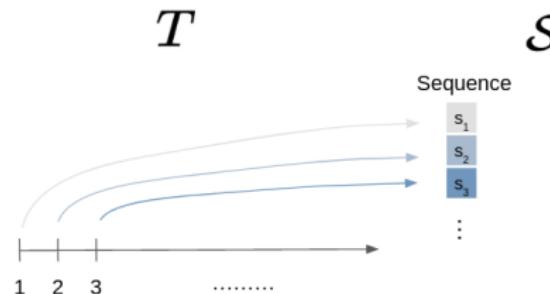
INDEXED FAMILY

Some simple examples for indexed families are:

- finite sequences (lists):
 $T = \{1, 2, \dots, n\}$ and
 $(s_t)_{t \in T} \in \mathbb{R}$



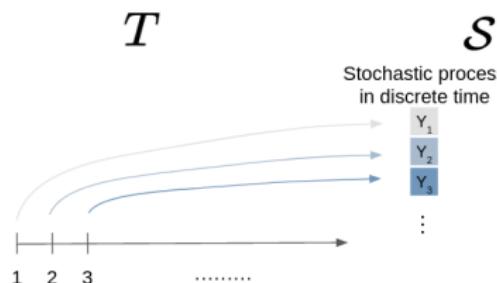
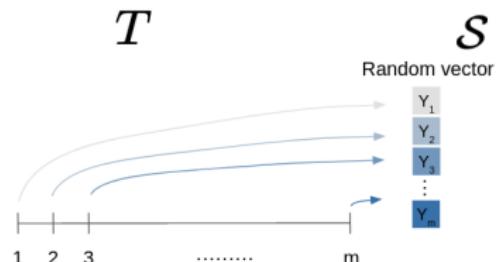
- infinite sequences:
 $T = \mathbb{N}$ and $(s_t)_{t \in T} \in \mathbb{R}$



INDEXED FAMILY

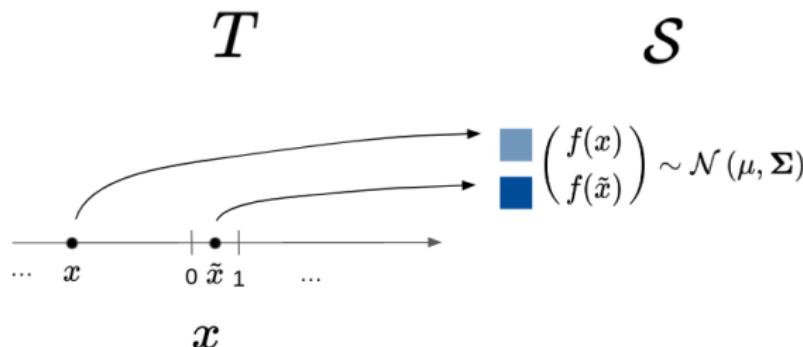
But the indexed set S can be something more complicated, for example functions or **random variables** (RV):

- $T = \{1, \dots, m\}$, Y_t 's are RVs: Indexed family is a random vector.
 - $T = \{1, \dots, m\}$, Y_t 's are RVs: Indexed family is a stochastic process in discrete time
 - $T = \mathbb{Z}^2$, Y_t 's are RVs: Indexed family is a 2D-random walk.



INDEXED FAMILY

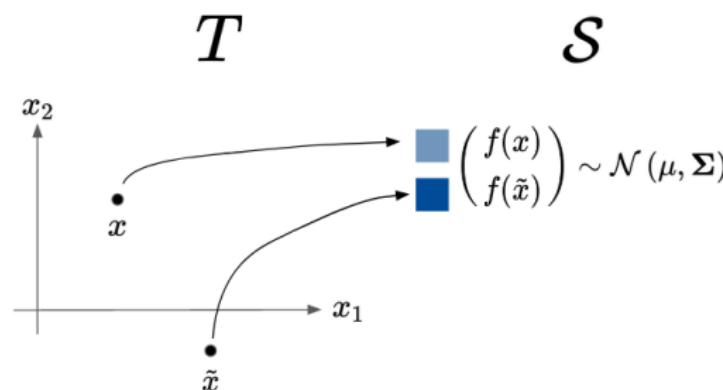
- A Gaussian process is also an indexed family, where the random variables $f(\mathbf{x})$ are indexed by the input values $\mathbf{x} \in \mathcal{X}$.
- Their special feature: Any indexed (finite) random vector has a multivariate Gaussian distribution (which comes with all the nice properties of Gaussianity!).



Visualization for a one-dimensional \mathcal{X} .

INDEXED FAMILY

- A Gaussian process is also an indexed family, where the random variables $f(\mathbf{x})$ are indexed by the input values $\mathbf{x} \in \mathcal{X}$.
- Their special feature: Any indexed (finite) random vector has a multivariate Gaussian distribution (which comes with all the nice properties of Gaussianity!).

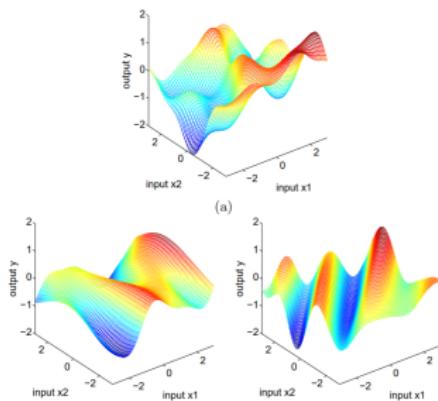
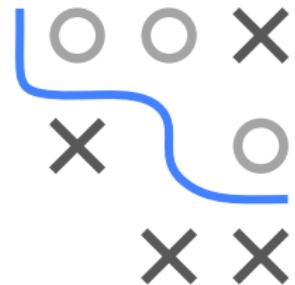


Visualization for a two-dimensional \mathcal{X} .

Introduction to Machine Learning

Gaussian Processes

Covariance functions for GPs



Learning goals

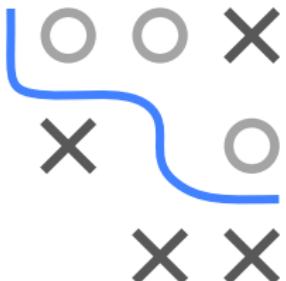
- Covariance functions encode key assumptions about the GP
- Know common covariance functions like squared exponential and Matérn

COVARIANCE FUNCTION OF A GP

The marginalization property of the Gaussian process implies that for any finite set of input values, the corresponding vector of function values is Gaussian:

$$\mathbf{f} = \left[f(\mathbf{x}^{(1)}), \dots, f(\mathbf{x}^{(n)}) \right] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}),$$

- The covariance matrix \mathbf{K} is constructed based on the chosen inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$.
- Entry K_{ij} is computed by $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$.
- Technically, for **every** choice of inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$, \mathbf{K} needs to be positive semi-definite in order to be a valid covariance matrix.
- A function $k(\cdot, \cdot)$ satisfying this property is called **positive definite**.



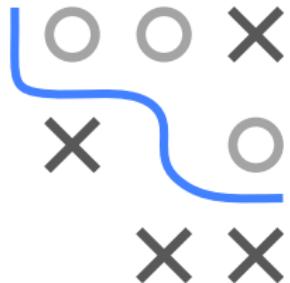
COVARIANCE FUNCTION OF A GP

- Recall, the purpose of the covariance function is to control to which degree the following is fulfilled:

If two points $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$ are close in \mathcal{X} -space, their function values $f(\mathbf{x}^{(i)}), f(\mathbf{x}^{(j)})$ should be close (**correlated!**) in \mathcal{Y} -space.

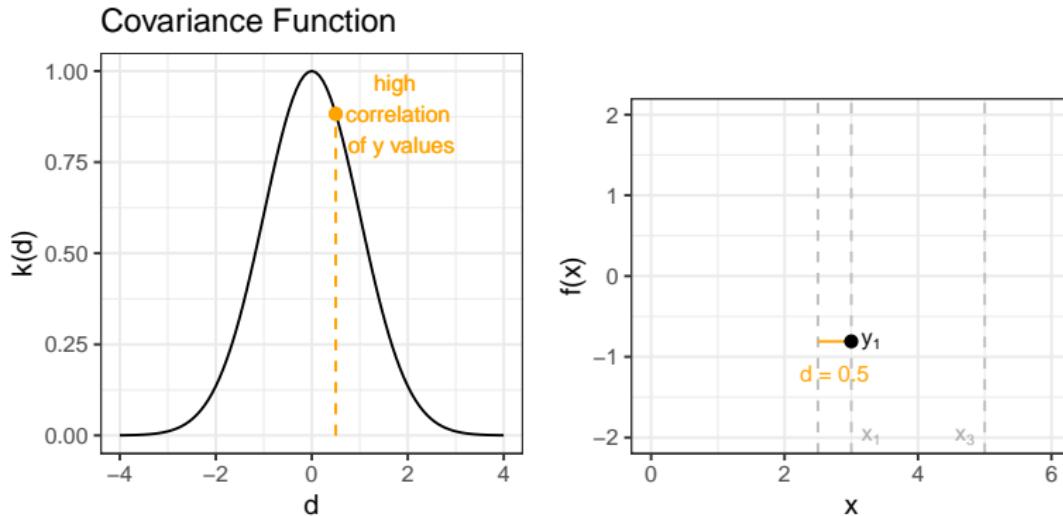
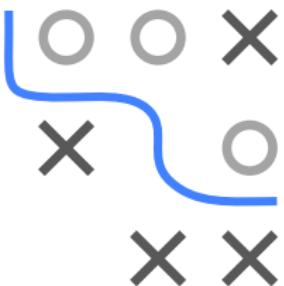
- Closeness of two points $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}$ in input space \mathcal{X} is measured in terms of $\mathbf{d} = \mathbf{x}^{(i)} - \mathbf{x}^{(j)}$:

$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = k(\mathbf{d})$$



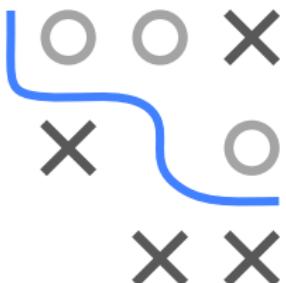
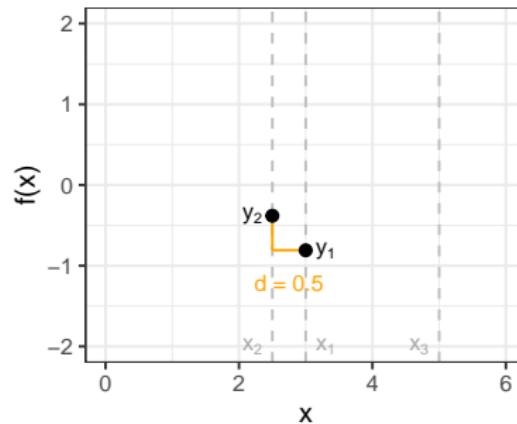
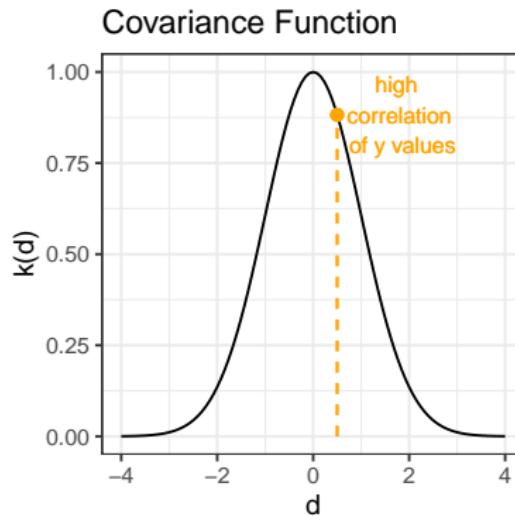
COVARIANCE FUNCTION OF A GP: EXAMPLE

- Let $f(\mathbf{x})$ be a GP with $k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2}\|\mathbf{d}\|^2)$ with $\mathbf{d} = \mathbf{x} - \mathbf{x}'$.
- Consider two points $\mathbf{x}^{(1)} = 3$ and $\mathbf{x}^{(2)} = 2.5$.
- If you want to know how correlated their function values are, compute their correlation!



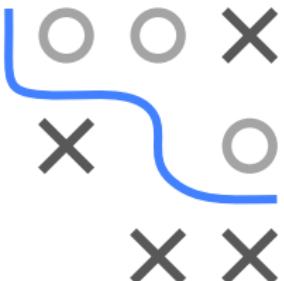
COVARIANCE FUNCTION OF A GP: EXAMPLE

- Assume we observed a value $y^{(1)} = -0.8$, the value of $y^{(2)}$ should be close under the assumption of the above Gaussian process.

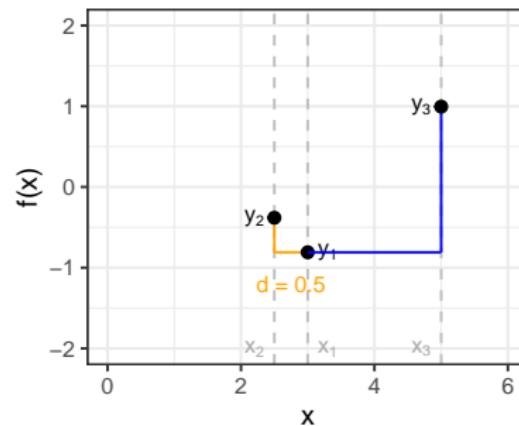
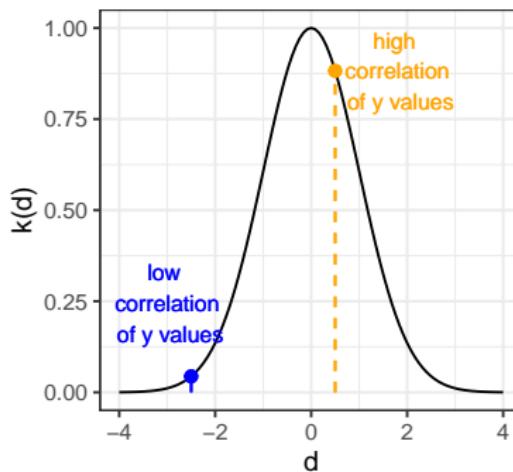


COVARIANCE FUNCTION OF A GP: EXAMPLE

- Let us compare another point $\mathbf{x}^{(3)}$ to the point $\mathbf{x}^{(1)}$
- We again compute their correlation
- Their function values are not very much correlated; $y^{(1)}$ and $y^{(3)}$ might be far away from each other



Covariance Function



COVARIANCE FUNCTIONS

There are three types of commonly used covariance functions:

- $k(., .)$ is called stationary if it is as a function of $\mathbf{d} = \mathbf{x} - \mathbf{x}'$, we write $k(\mathbf{d})$.

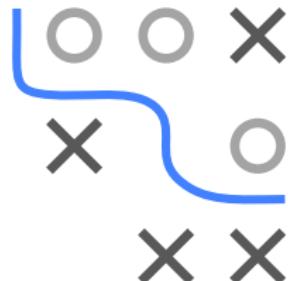
Stationarity is invariance to translations in the input space:

$$k(\mathbf{x}, \mathbf{x} + \mathbf{d}) = k(\mathbf{0}, \mathbf{d})$$

- $k(., .)$ is called isotropic if it is a function of $r = \|\mathbf{x} - \mathbf{x}'\|$, we write $k(r)$.

Isotropy is invariance to rotations of the input space and implies stationarity.

- $k(., .)$ is a dot product covariance function if k is a function of $\mathbf{x}^T \mathbf{x}'$



COMMONLY USED COVARIANCE FUNCTIONS

Name	$k(\mathbf{x}, \mathbf{x}')$
constant	σ_0^2
linear	$\sigma_0^2 + \mathbf{x}^T \mathbf{x}'$
polynomial	$(\sigma_0^2 + \mathbf{x}^T \mathbf{x}')^p$
squared exponential	$\exp\left(-\frac{\ \mathbf{x}-\mathbf{x}'\ ^2}{2\ell^2}\right)$
Matérn	$\frac{1}{2^\nu \Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{\ell} \ \mathbf{x} - \mathbf{x}'\ \right)^\nu K_\nu \left(\frac{\sqrt{2\nu}}{\ell} \ \mathbf{x} - \mathbf{x}'\ \right)$
exponential	$\exp\left(-\frac{\ \mathbf{x}-\mathbf{x}'\ }{\ell}\right)$

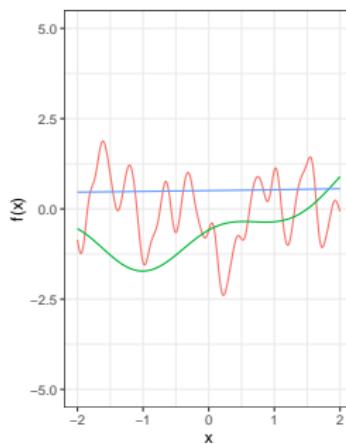


$K_\nu(\cdot)$ is the modified Bessel function of the second kind.

COMMONLY USED COVARIANCE FUNCTIONS

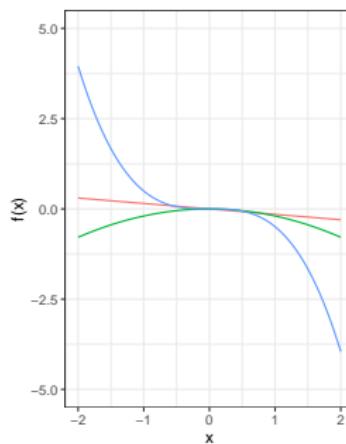
Squared Exponential Covariance F

Length Scale — 0.1 — 1 — 10



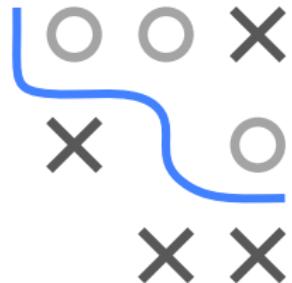
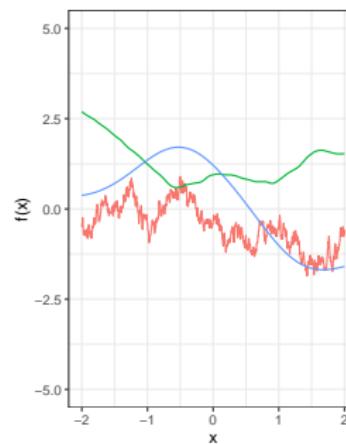
Polynomial Covariance Function

Degree — 1 — 2 — 3



Matérn Covariance Functions

ν — 0.5 — 2 — 10



- Random functions drawn from Gaussian processes with a Squared Exponential Kernel (left), Polynomial Kernel (middle), and a Matérn Kernel (right, $\ell = 1$).
- The length-scale hyperparameter determines the “wiggleness” of the function.
- For Matérn, the ν parameter determines how differentiable the process is.

SQUARED EXPONENTIAL COVARIANCE FUNCTION

The squared exponential function is one of the most commonly used covariance functions.

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right).$$



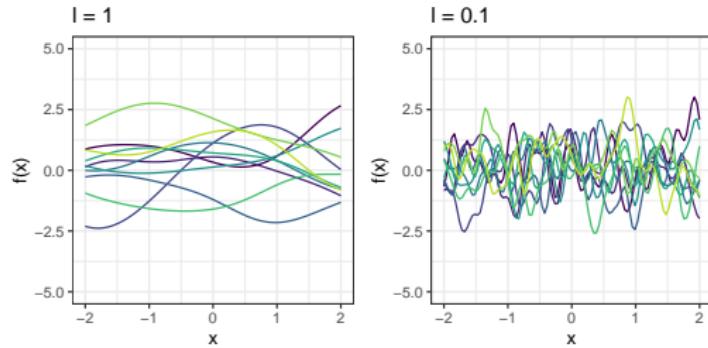
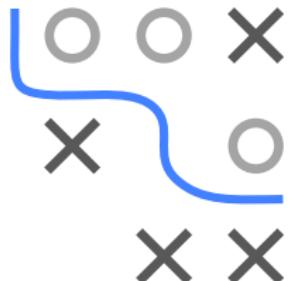
Properties:

- It depends merely on the distance $r = \|\mathbf{x} - \mathbf{x}'\| \rightarrow$ isotropic and stationary.
- Infinitely differentiable \rightarrow sometimes deemed unrealistic for modeling most of the physical processes.

CHARACTERISTIC LENGTH-SCALE

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\ell^2} \|\mathbf{x} - \mathbf{x}'\|^2\right)$$

ℓ is called **characteristic length-scale**. Loosely speaking, the characteristic length-scale describes how far you need to move in input space for the function values to become uncorrelated. Higher ℓ induces smoother functions, lower ℓ induces more wiggly functions.



CHARACTERISTIC LENGTH-SCALE

For $p \geq 2$ dimensions, the squared exponential can be parameterized:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2} (\mathbf{x} - \mathbf{x}')^\top \mathbf{M} (\mathbf{x} - \mathbf{x}')\right)$$

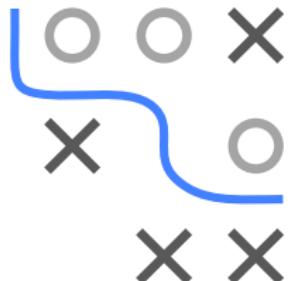
Possible choices for the matrix \mathbf{M} include

$$\mathbf{M}_1 = \ell^{-2} \mathbf{I} \quad \mathbf{M}_2 = \text{diag}(\ell)^{-2} \quad \mathbf{M}_3 = \Gamma \Gamma^\top + \text{diag}(\ell)^{-2}$$

where ℓ is a p -vector of positive values and Γ is a $p \times k$ matrix.

The 2nd (and most important) case can also be written as

$$k(\mathbf{d}) = \exp\left(-\frac{1}{2} \sum_{j=1}^p \frac{d_j^2}{l_j^2}\right)$$



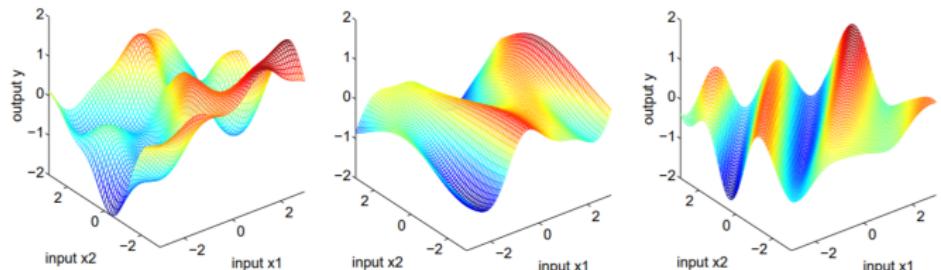
CHARACTERISTIC LENGTH-SCALE

What is the benefit of having an individual hyperparameter ℓ_i for each dimension?

- The ℓ_1, \dots, ℓ_p hyperparameters play the role of **characteristic length-scales**.
- Loosely speaking, ℓ_i describes how far you need to move along axis i in input space for the function values to be uncorrelated.
- Such a covariance function implements **automatic relevance determination** (ARD), since the inverse of the length-scale ℓ_i determines the relevancy of input feature i to the regression.
- If ℓ_i is very large, the covariance will become almost independent of that input, effectively removing it from inference.
- If the features are on different scales, the data can be automatically **rescaled** by estimating ℓ_1, \dots, ℓ_p



CHARACTERISTIC LENGTH-SCALE

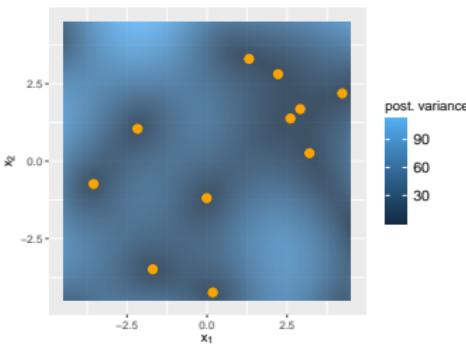


For the first plot, we have chosen $\mathbf{M} = \mathbf{I}$: the function varies the same in all directions. The second plot is for $\mathbf{M} = \text{diag}(\ell)^{-2}$ and $\ell = (1, 3)$: The function varies less rapidly as a function of x_2 than x_1 as the length-scale for x_1 is less. In the third plot $\mathbf{M} = \Gamma\Gamma^T + \text{diag}(\ell)^{-2}$ for $\Gamma = (1, -1)^\top$ and $\ell = (6, 6)^\top$. Here Γ gives the direction of the most rapid variation. (Image from Rasmussen & Williams, 2006)

Introduction to Machine Learning

Gaussian Processes

Gaussian Posterior Process and Prediction

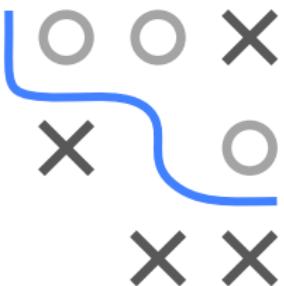


Learning goals

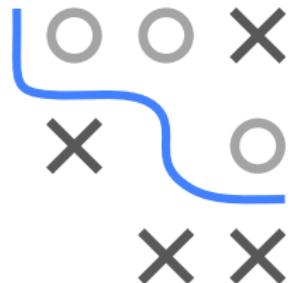
- Know how to derive the posterior process
- GPs are interpolating and spatial models
- Model noise via a nugget term

GAUSSIAN POSTERIOR PROCESS AND PREDICTION

- So far, we have learned how to **sample** from a GP prior.
- However, most of the time, we are not interested in drawing random functions from the prior. Instead, we usually like to use the knowledge provided by the training data to predict values of f at a new test point \mathbf{x}_* .
- In what follows, we will investigate how to update the Gaussian process prior (\rightarrow posterior process) and how to make predictions.



Gaussian Posterior Process and Prediction



POSTERIOR PROCESS

- Let us now distinguish between observed training inputs, also denote by a design matrix \mathbf{X} , and the corresponding observed values

$$\mathbf{f} = \left[f(\mathbf{x}^{(1)}), \dots, f(\mathbf{x}^{(n)}) \right]$$



and one single **unobserved test point** \mathbf{x}_* with $f_* = f(\mathbf{x}_*)$.

- We now want to infer the distribution of $f_* | \mathbf{x}_*, \mathbf{X}, \mathbf{f}$.

$$f_* = f(\mathbf{x}_*)$$

- Assuming a zero-mean GP prior $\mathcal{GP}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}'))$ we know

$$\begin{bmatrix} \mathbf{f} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^T & \mathbf{k}_{**} \end{bmatrix}\right).$$

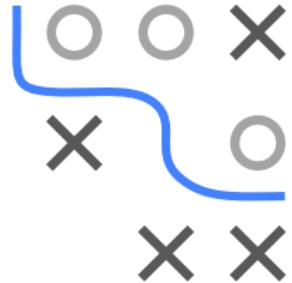
Here, $\mathbf{K} = (k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}))_{i,j}$, $\mathbf{k}_* = [k(\mathbf{x}_*, \mathbf{x}^{(1)}), \dots, k(\mathbf{x}_*, \mathbf{x}^{(n)})]$ and $\mathbf{k}_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$.

POSTERIOR PROCESS

- Given that \mathbf{f} is observed, we can apply the general rule for condition (*) of Gaussian random variables and obtain the following formula:

$$f_* \mid \mathbf{x}_*, \mathbf{X}, \mathbf{f} \sim \mathcal{N}(\mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{f}, \mathbf{k}_{**} - \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{k}_*).$$

- As the posterior is a Gaussian, the maximum a-posteriori estimate, i.e. the mode of the posterior distribution, is $\mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{f}$.



POSTERIOR PROCESS

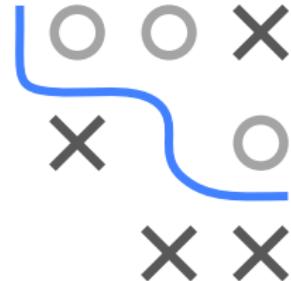
(*) General rule for condition of Gaussian random variables:

If the m -dimensional Gaussian vector $\mathbf{z} \sim \mathcal{N}(\mu, \Sigma)$ can be partitioned with $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2)$ where \mathbf{z}_1 is m_1 -dimensional and \mathbf{z}_2 is m_2 -dimensional, and:

$$(\mu_1, \mu_2), \quad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix},$$

then the conditioned distribution of $\mathbf{z}_2 \mid \mathbf{z}_1 = \mathbf{a}$ is a multivariate normal

$$\mathcal{N}(\mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{a} - \mu_1), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12})$$



GP PREDICTION: TWO POINTS

Let us visualize this by a simple example:

- Assume we observed a single training point $\mathbf{x} = -0.5$, and want to make a prediction at a test point $\mathbf{x}_* = 0.5$.
- Under a zero-mean GP with $k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2)$, we compute the cov-matrix:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} 1 & 0.61 \\ 0.61 & 1 \end{bmatrix}\right).$$



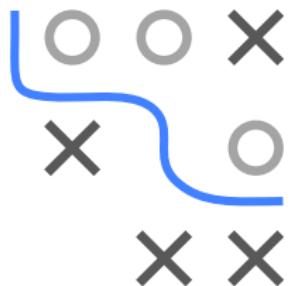
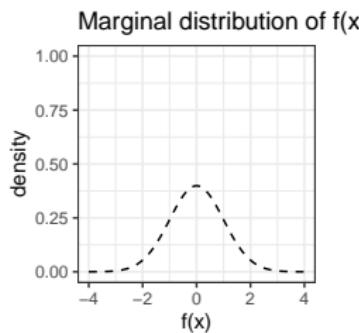
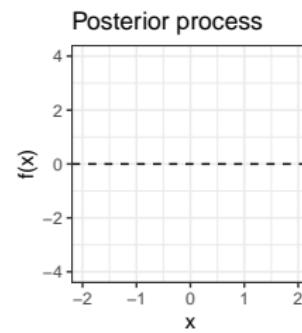
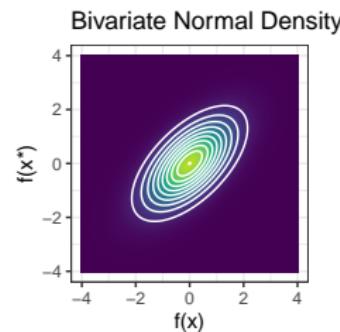
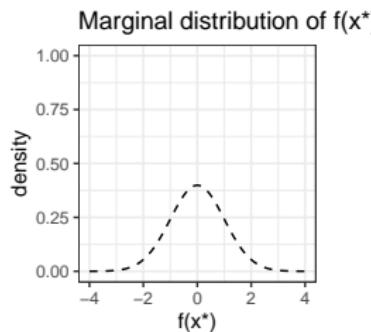
- Assume that we observe the point $f(\mathbf{x}) = 1$.
- We compute the posterior distribution:

$$\begin{aligned} f_* | \mathbf{x}_*, \mathbf{x}, f &\sim \mathcal{N}(\mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{f}, k_{**} - \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{k}_*) \\ &\sim \mathcal{N}(0.61 \cdot 1 \cdot 1, 1 - 0.61 \cdot 1 \cdot 0.61) \\ &\sim \mathcal{N}(0.61, 0.6279) \end{aligned}$$

- The MAP-estimate for \mathbf{x}_* is $f(\mathbf{x}_*) = 0.61$, and the uncertainty estimate is 0.6279.

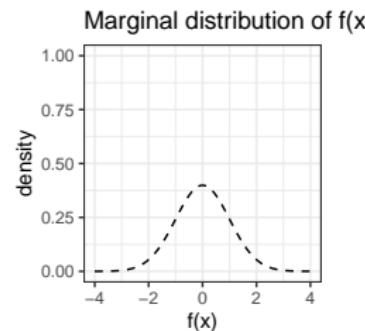
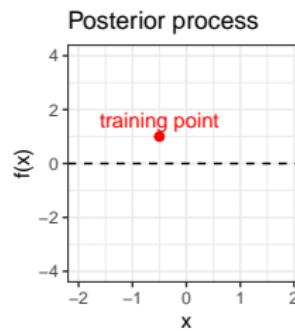
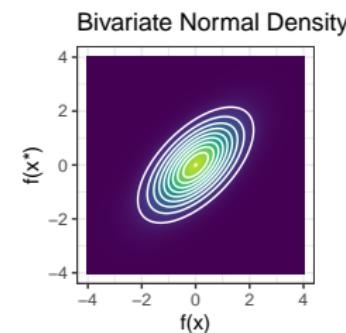
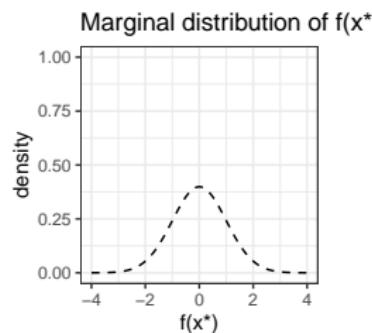
GP PREDICTION: TWO POINTS

Shown is the bivariate normal density, and the respective marginals.



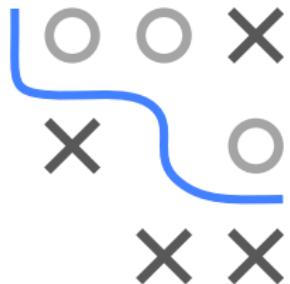
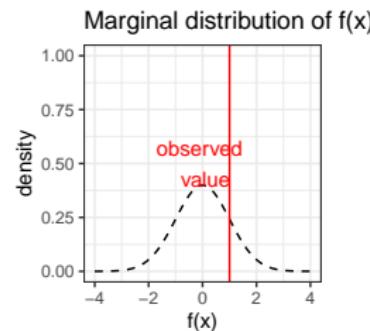
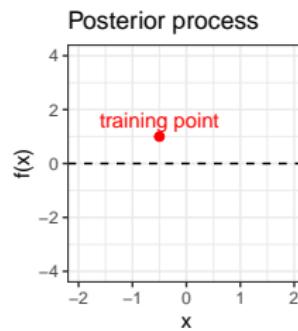
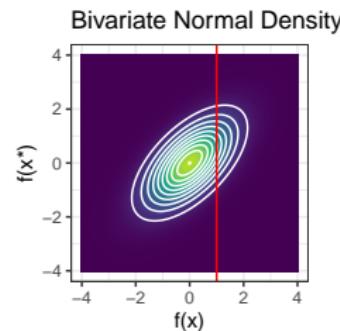
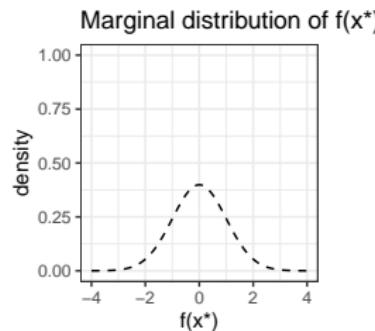
GP PREDICTION: TWO POINTS

Assume we observed $f(\mathbf{x}) = 1$ for the training point $\mathbf{x} = -0.5$.



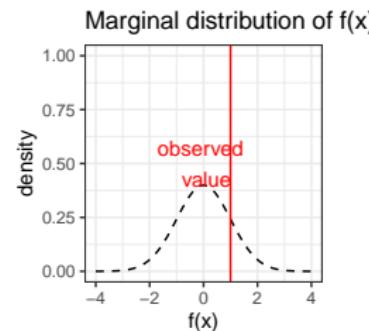
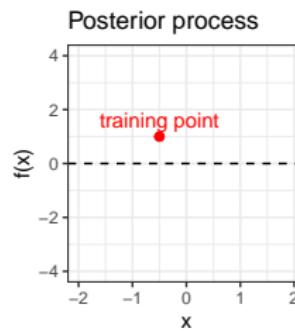
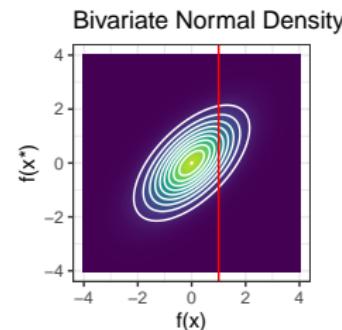
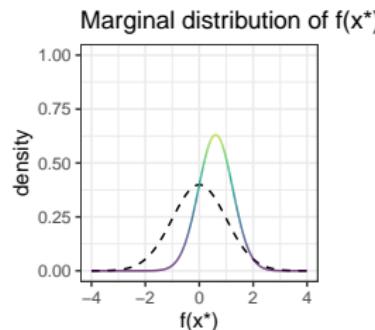
GP PREDICTION: TWO POINTS

We condition the Gaussian on $f(\mathbf{x}) = 1$.



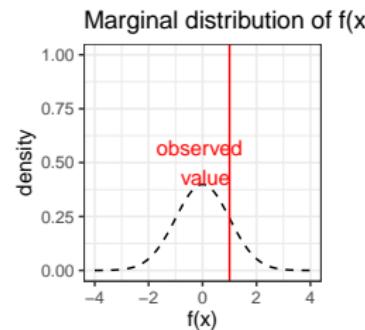
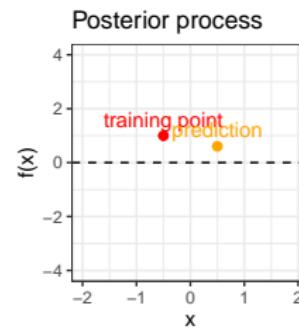
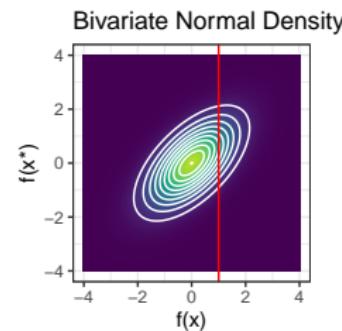
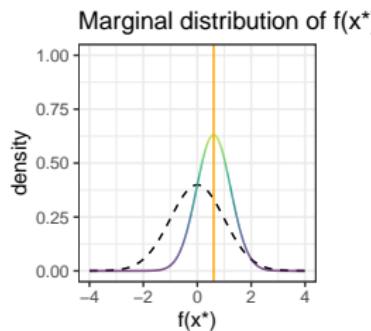
GP PREDICTION: TWO POINTS

We compute the posterior distribution of $f(x_*)$ given that $f(x) = 1$.



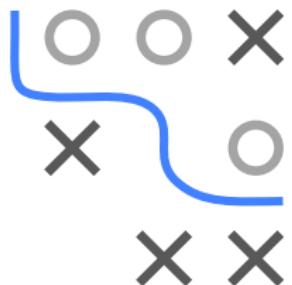
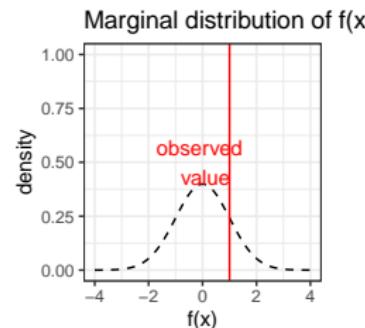
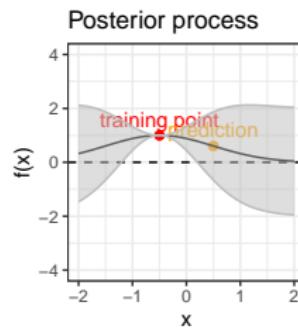
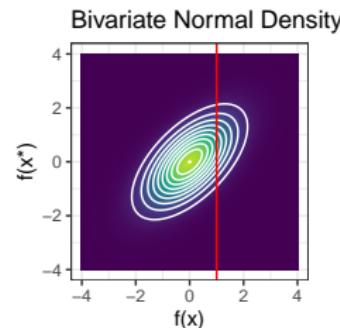
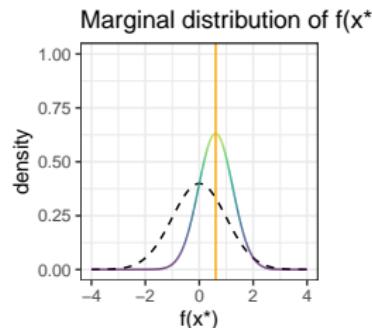
GP PREDICTION: TWO POINTS

A possible predictor for f at \mathbf{x}_* is the MAP of the posterior distribution.



GP PREDICTION: TWO POINTS

We can do this for different values \mathbf{x}_* , and show the respective mean (grey line) and standard deviations (grey area is mean $\pm 2 \cdot$ posterior standard deviation).



POSTERIOR PROCESS

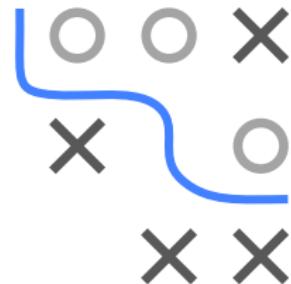
- We can generalize the formula for the posterior process for multiple unobserved test points:

$$\mathbf{f}_* = \left[f\left(\mathbf{x}_*^{(1)}\right), \dots, f\left(\mathbf{x}_*^{(m)}\right) \right].$$

- Under a zero-mean Gaussian process, we have

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix}\right),$$

with $\mathbf{K}_* = \left(k\left(\mathbf{x}^{(i)}, \mathbf{x}_*^{(j)}\right)\right)_{i,j}$, $\mathbf{K}_{**} = \left(k\left(\mathbf{x}_*^{(i)}, \mathbf{x}_*^{(j)}\right)\right)_{i,j}$.

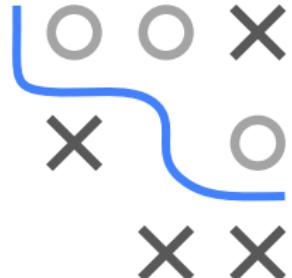


POSTERIOR PROCESS

- Similar to the single test point situation, to get the posterior distribution, we exploit the general rule of conditioning for Gaussians:

$$\mathbf{f}_* \mid \mathbf{X}_*, \mathbf{X}, \mathbf{f} \sim \mathcal{N}(\mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{f}, \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_*).$$

- This formula enables us to talk about correlations among different test points and sample functions from the posterior process.



Properties of a Gaussian Process

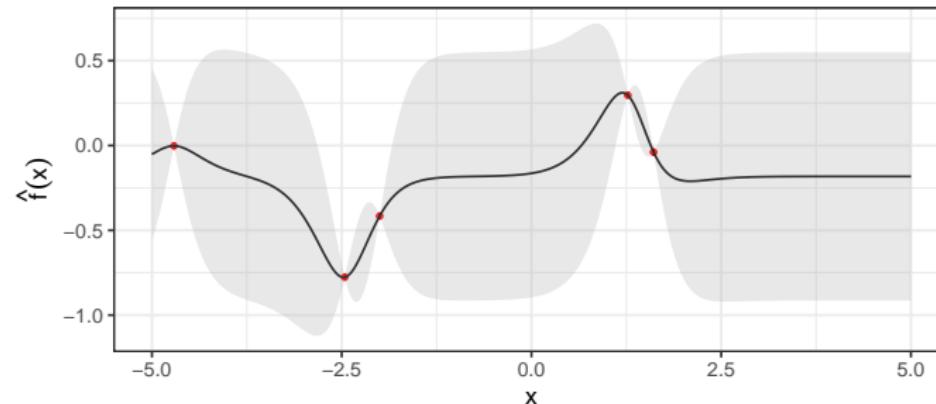
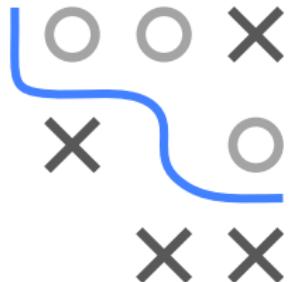


GP AS INTERPOLATOR

The “prediction” for a training point $\mathbf{x}^{(i)}$ is the exact function value $f(\mathbf{x}^{(i)})$

$$\mathbf{f} | \mathbf{X}, \mathbf{f} \sim \mathcal{N}(\mathbf{K}\mathbf{K}^{-1}\mathbf{f}, \mathbf{K} - \mathbf{K}^T\mathbf{K}^{-1}\mathbf{K}) = \mathcal{N}(\mathbf{f}, \mathbf{0}).$$

Thus, a Gaussian process is a function **interpolator**.



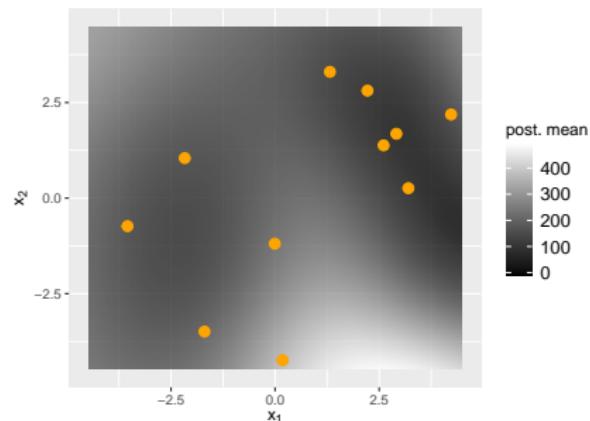
After observing the training points (red), the posterior process (black) interpolates the training points.
($k(x, x')$ is Matérn with $\nu = 2.5$, the default for DiceKriging::km)

GP AS A SPATIAL MODEL

- The correlation among two outputs depends on distance of the corresponding input points \mathbf{x} and \mathbf{x}' (e.g. Gaussian covariance kernel)

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-\|\mathbf{x} - \mathbf{x}'\|^2}{2l^2}\right)$$

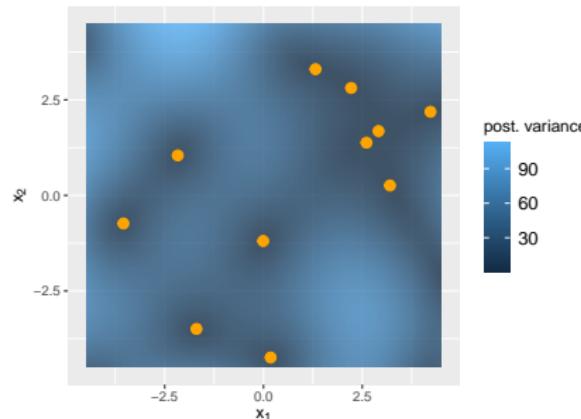
- Hence, close data points with high spatial similarity $k(\mathbf{x}, \mathbf{x}')$ enter into more strongly correlated predictions: $\mathbf{k}_*^\top \mathbf{K}^{-1} \mathbf{f}$ ($\mathbf{k}_* := (k(\mathbf{x}, \mathbf{x}^{(1)}), \dots, k(\mathbf{x}, \mathbf{x}^{(n)}))$).



Example: Posterior mean of a GP that was fitted with the Gaussian covariance kernel with $l = 1$.

GP AS A SPATIAL MODEL

- Posterior uncertainty increases if the new data points are far from the design points.
- The uncertainty is minimal at the design points, since the posterior variance is zero at these points.



Example (continued): Posterior variance.

Noisy Gaussian Process

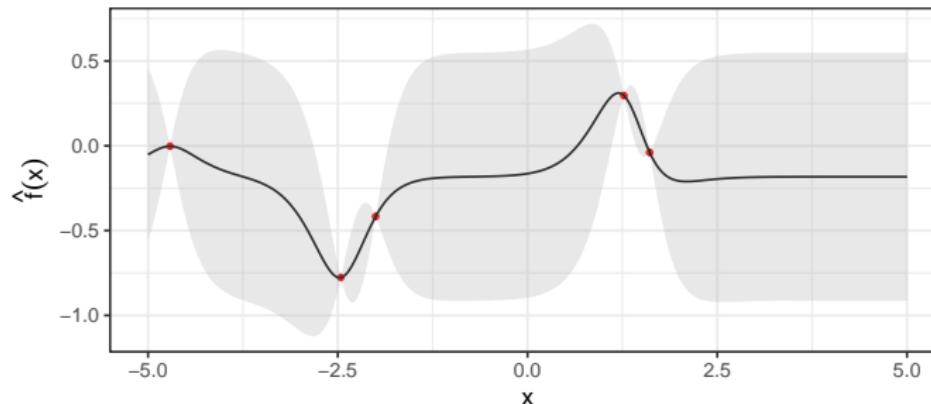


NOISY GAUSSIAN PROCESS

- So far, we implicitly assumed that we had access to the true function value $f(\mathbf{x})$.
- For the squared exponential kernel, for example, we have

$$\text{Cov} \left(f(\mathbf{x}^{(i)}), f(\mathbf{x}^{(i)}) \right) = 1.$$

- As a result, the posterior Gaussian process is an interpolator:



After observing the training points (red), the posterior process (black) interpolates the training points.
 $(k(x,x'))$ is Matérn with $\nu = 2.5$, the default for DiceKriging::km)



NOISY GAUSSIAN PROCESS

- In reality, however, this is often not the case.
- We often only have access to a noisy version of the true function value

$$y = f(\mathbf{x}) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2).$$

- Let us still assume that $f(\mathbf{x})$ is a Gaussian process.
- Then,

$$\begin{aligned}\text{Cov}(y^{(i)}, y^{(j)}) &= \text{Cov} \left(f \left(\mathbf{x}^{(i)} \right) + \epsilon^{(i)}, f \left(\mathbf{x}^{(j)} \right) + \epsilon^{(j)} \right) \\ &= \text{Cov} \left(f \left(\mathbf{x}^{(i)} \right), f \left(\mathbf{x}^{(j)} \right) \right) + 2 \cdot \text{Cov} \left(f \left(\mathbf{x}^{(i)} \right), \epsilon^{(j)} \right) + \text{Cov} \left(\epsilon^{(i)}, \epsilon^{(j)} \right) \\ &= k \left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)} \right) + \sigma^2 \delta_{ij}.\end{aligned}$$

- σ^2 is called **nugget**.



NOISY GAUSSIAN PROCESS

- Let us now derive the predictive distribution for the case of noisy observations.
- The prior distribution of y , assuming that f is modeled by a Gaussian process is then

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{pmatrix} \sim \mathcal{N}(\mathbf{m}, \mathbf{K} + \sigma^2 \mathbf{I}_n),$$



with

$$\mathbf{m} := \left(m(\mathbf{x}^{(i)}) \right)_i, \quad \mathbf{K} := \left(k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \right)_{i,j}.$$

NOISY GAUSSIAN PROCESS

- We distinguish again between
 - observed training points \mathbf{X}, \mathbf{y} , and
 - unobserved test inputs \mathbf{X}_* with unobserved values \mathbf{f}_*

and get

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + \sigma^2 \mathbf{I}_n & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix}\right).$$



NOISY GAUSSIAN PROCESS

- Similarly to the noise-free case, we condition according to the rule of conditioning for Gaussians to get the posterior distribution for the test outputs \mathbf{f}_* at \mathbf{X}_* :

$$\mathbf{f}_* \mid \mathbf{X}_*, \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\mathbf{m}_{\text{post}}, \mathbf{K}_{\text{post}}).$$

with

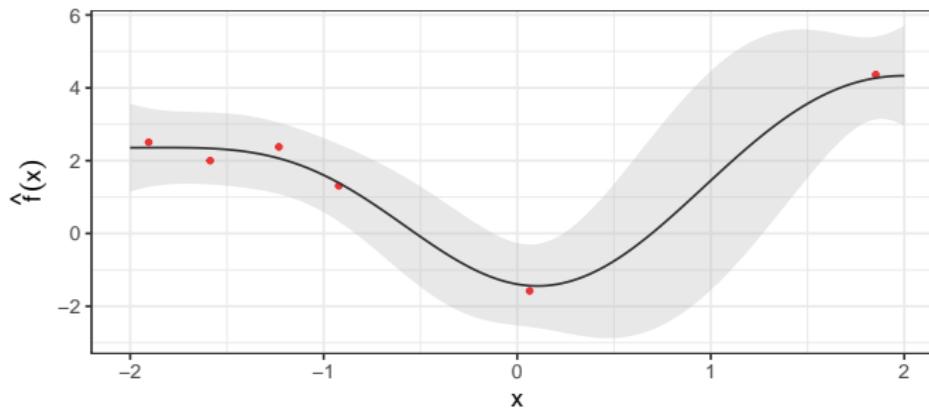
$$\begin{aligned}\mathbf{m}_{\text{post}} &= \mathbf{K}_*^T (\mathbf{K} + \sigma^2 \cdot \mathbf{I})^{-1} \mathbf{y} \\ \mathbf{K}_{\text{post}} &= \mathbf{K}_{**} - \mathbf{K}_*^T (\mathbf{K} + \sigma^2 \cdot \mathbf{I})^{-1} \mathbf{K}_*,\end{aligned}$$

- This converts back to the noise-free formula if $\sigma^2 = 0$.



NOISY GAUSSIAN PROCESS

- The noisy Gaussian process is not an interpolator any more.
- A larger nugget term leads to a wider “band” around the observed training points.
- The nugget term is estimated during training.



After observing the training points (red), we have a nugget–band around the oberved points.
 $(k(x,x'))$ is the squared exponential)



Decision Theory for Gaussian Processes

RISK MINIMIZATION FOR GAUSSIAN PROCESSES

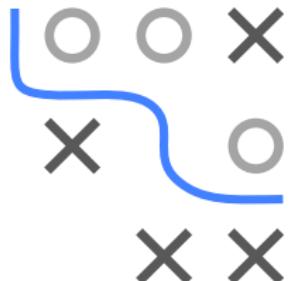
In machine learning, we learned about risk minimization. We usually choose a loss function and minimize the empirical risk

$$\mathcal{R}_{\text{emp}}(f) := \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)}))$$

as an approximation to the theoretical risk

$$\mathcal{R}(f) := \mathbb{E}_{xy}[L(y, f(\mathbf{x}))] = \int L(y, f(\mathbf{x})) dP_{xy}.$$

- How does the theory of Gaussian processes fit into this theory?
- What if we want to make a prediction which is optimal w.r.t. a certain loss function?



RISK MINIMIZATION FOR GAUSSIAN PROCESSES

- The theory of Gaussian process gives us a posterior distribution

$$p(y | \mathcal{D})$$

- If we now want to make a prediction at a test point \mathbf{x}_* , we approximate the theoretical risk in a different way, by using the posterior distribution:

$$\mathcal{R}(y_* | \mathbf{x}_*) \approx \int L(\tilde{y}_*, y_*) p(\tilde{y}_* | \mathbf{x}_*, \mathcal{D}) d\tilde{y}_*.$$

- The optimal prediction w.r.t the loss function is then:

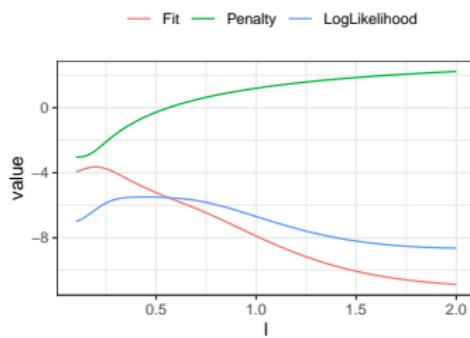
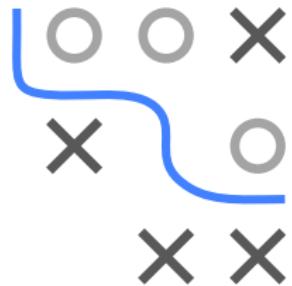
$$\hat{y}_* | \mathbf{x}_* = \arg \min_{y_*} \mathcal{R}(y_* | \mathbf{x}_*).$$



Introduction to Machine Learning

Gaussian Processes

Training of a Gaussian Process



Learning goals

- Training of GPs via Maximum Likelihood estimation of its hyperparameters
- Computational complexity is governed by matrix inversion of the covariance matrix

TRAINING OF A GAUSSIAN PROCESS

- To make predictions for a regression task by a Gaussian process, one simply needs to perform matrix computations.
- But for this to work out, we assume that the covariance functions is fully given, including all of its hyperparameters.
- A very nice property of GPs is that we can learn the numerical hyperparameters of a selected covariance function directly during GP training.



TRAINING A GP VIA MAXIMUM LIKELIHOOD

Let us assume

$$y = f(\mathbf{x}) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2),$$

where $f(\mathbf{x}) \sim \mathcal{GP}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}'|\theta))$.

Observing $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K} + \sigma^2 \mathbf{I})$, the marginal log-likelihood (or evidence) is

$$\begin{aligned}\log p(\mathbf{y} | \mathbf{X}, \theta) &= \log \left[(2\pi)^{-n/2} |\mathbf{K}_y|^{-1/2} \exp \left(-\frac{1}{2} \mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y} \right) \right] \\ &= -\frac{1}{2} \mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_y| - \frac{n}{2} \log 2\pi.\end{aligned}$$

with $\mathbf{K}_y := \mathbf{K} + \sigma^2 \mathbf{I}$ and θ denoting the hyperparameters (the parameters of the covariance function).



TRAINING A GP VIA MAXIMUM LIKELIHOOD

The three terms of the marginal likelihood have interpretable roles, considering that the model becomes less flexible as the length-scale increases:

- the data fit $-\frac{1}{2}\mathbf{y}^T \mathbf{K}_y^{-1} \mathbf{y}$, which tends to decrease if the length scale increases
- the complexity penalty $-\frac{1}{2} \log |\mathbf{K}_y|$, which depends on the covariance function only and which increases with the length-scale, because the model gets less complex with growing length-scale
- a normalization constant $-\frac{n}{2} \log 2\pi$



TRAINING A GP: EXAMPLE

To visualize this, we consider a zero-mean Gaussian process with squared exponential kernel

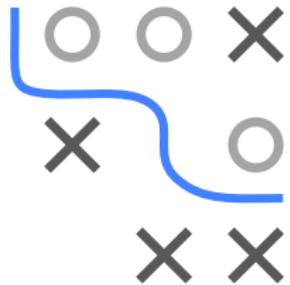
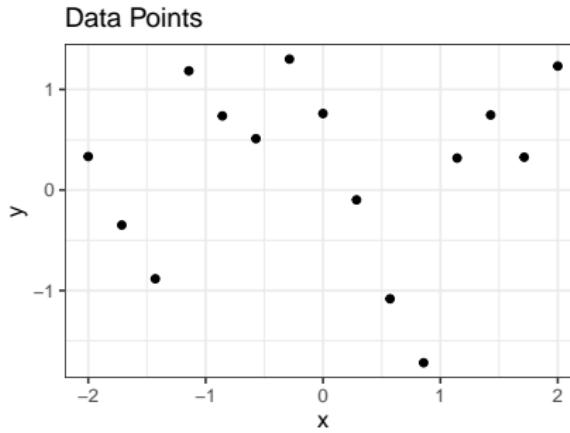
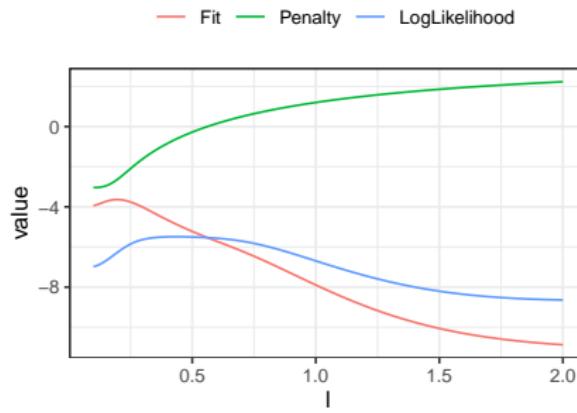
$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\ell^2}\|\mathbf{x} - \mathbf{x}'\|^2\right),$$



- Recall, the model is smoother and less complex for higher length-scale ℓ .
- We show how the
 - data fit $-\frac{1}{2}\mathbf{y}^T \mathbf{K}_y^{-1} \mathbf{y}$,
 - the complexity penalty $-\frac{1}{2} \log |\mathbf{K}_y|$, and
 - the overall value of the marginal likelihood $\log p(\mathbf{y} | \mathbf{X}, \theta)$

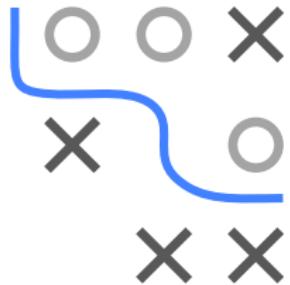
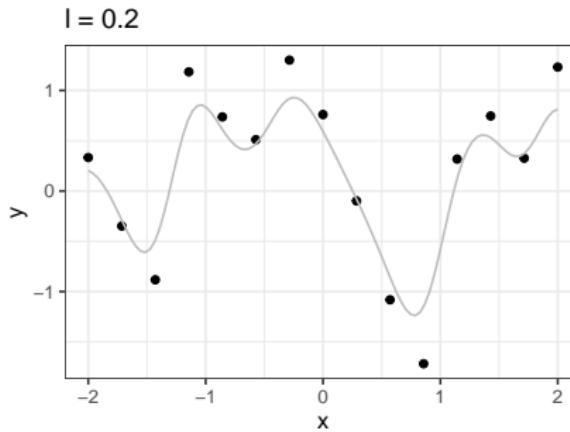
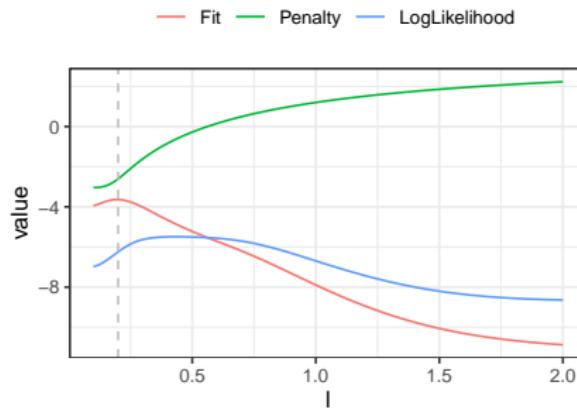
behave for increasing value of ℓ .

TRAINING A GP: EXAMPLE



The left plot shows how values of the data fit $-\frac{1}{2}\mathbf{y}^T \mathbf{K}_y^{-1} \mathbf{y}$, the complexity penalty $-\frac{1}{2} \log |\mathbf{K}_y|$ (high value means less penalization) and the overall marginal likelihood $\log p(\mathbf{y} | \mathbf{X}, \theta)$ behave for increasing values of ℓ .

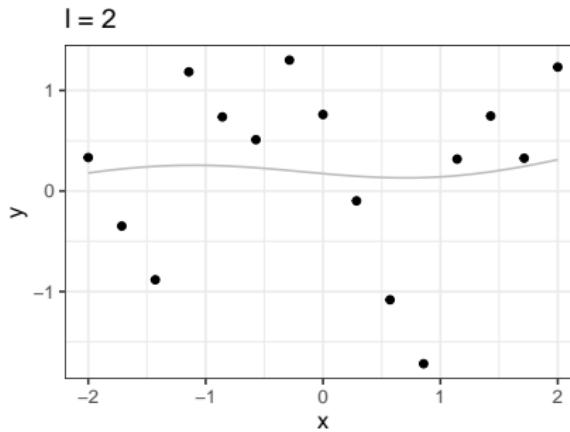
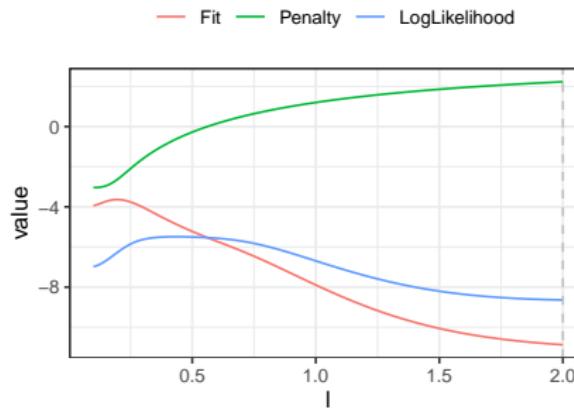
TRAINING A GP: EXAMPLE



The left plot shows how values of the data fit $-\frac{1}{2}\mathbf{y}^T \mathbf{K}_y^{-1} \mathbf{y}$, the complexity penalty $-\frac{1}{2} \log |\mathbf{K}_y|$ (high value means less penalization) and the overall marginal likelihood $\log p(\mathbf{y} | \mathbf{X}, \theta)$ behave for increasing values of ℓ .

A small ℓ results in a good fit, but a high complexity penalty (low $-\frac{1}{2} \log |\mathbf{K}_y|$).

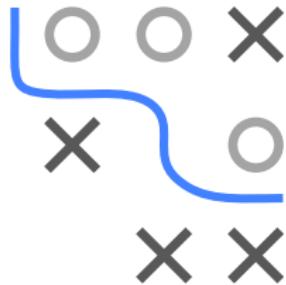
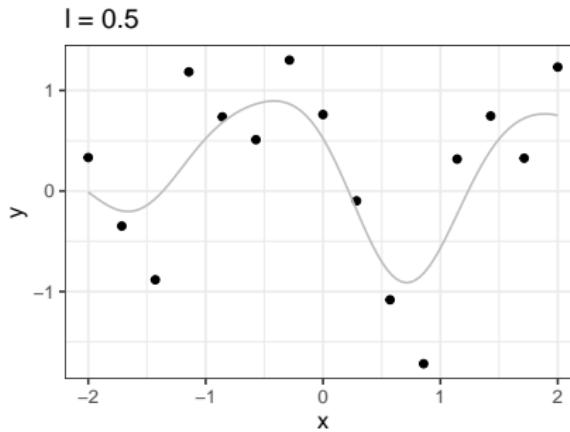
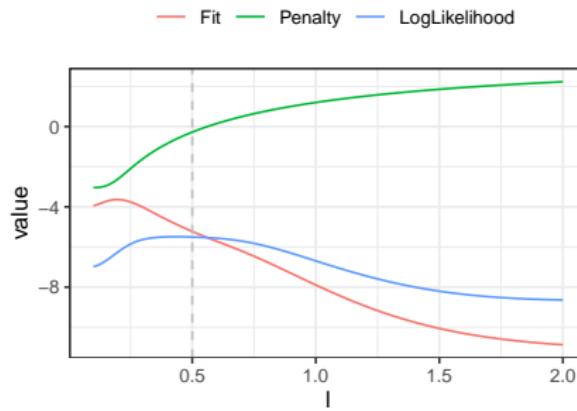
TRAINING A GP: EXAMPLE



The left plot shows how values of the data fit $-\frac{1}{2}\mathbf{y}^T \mathbf{K}_y^{-1} \mathbf{y}$, the complexity penalty $-\frac{1}{2} \log |\mathbf{K}_y|$ (high value means less penalization) and the overall marginal likelihood $\log p(\mathbf{y} | \mathbf{X}, \theta)$ behave for increasing values of ℓ .

A large ℓ results in a poor fit.

TRAINING A GP: EXAMPLE



The left plot shows how values of the data fit $-\frac{1}{2}\mathbf{y}^T \mathbf{K}_y^{-1} \mathbf{y}$, the complexity penalty $-\frac{1}{2} \log |\mathbf{K}_y|$ (high value means less penalization) and the overall marginal likelihood $\log p(\mathbf{y} | \mathbf{X}, \theta)$ behave for increasing values of ℓ .

The maximizer of the log-likelihood, $\ell = 0.5$, balances complexity and fit.

TRAINING A GP VIA MAXIMUM LIKELIHOOD

To set the hyperparameters by maximizing the marginal likelihood, we seek the partial derivatives w.r.t. the hyperparameters

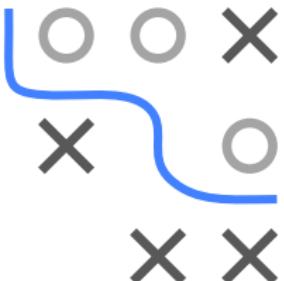
$$\begin{aligned}\frac{\partial}{\partial \theta_j} \log p(\mathbf{y} | \mathbf{X}, \boldsymbol{\theta}) &= \frac{\partial}{\partial \theta_j} \left(-\frac{1}{2} \mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_y| - \frac{n}{2} \log 2\pi \right) \\ &= \frac{1}{2} \mathbf{y}^\top \mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta_j} \mathbf{K}^{-1} \mathbf{y} - \frac{1}{2} \text{tr} \left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta} \right) \\ &= \frac{1}{2} \text{tr} \left((\mathbf{K}^{-1} \mathbf{y} \mathbf{y}^\top \mathbf{K}^{-1} - \mathbf{K}^{-1}) \frac{\partial \mathbf{K}}{\partial \theta_j} \right)\end{aligned}$$

using $\frac{\partial}{\partial \theta_j} \mathbf{K}^{-1} = -\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta_j} \mathbf{K}^{-1}$ and $\frac{\partial}{\partial \theta} \log |\mathbf{K}| = \text{tr} \left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta} \right)$.



TRAINING A GP VIA MAXIMUM LIKELIHOOD

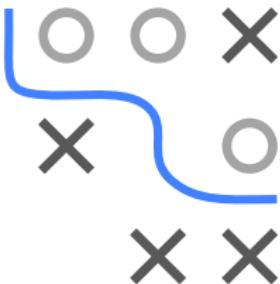
- The complexity and the runtime of training a Gaussian process is dominated by the computational task of inverting \mathbf{K} - or let's rather say for decomposing it.
- Standard methods require $\mathcal{O}(n^3)$ time (!) for this.
- Once \mathbf{K}^{-1} - or rather the decomposition -is known, the computation of the partial derivatives requires only $\mathcal{O}(n^2)$ time per hyperparameter.
- Thus, the computational overhead of computing derivatives is small, so using a gradient based optimizer is advantageous.



TRAINING A GP VIA MAXIMUM LIKELIHOOD

Workarounds to make GP estimation feasible for big data include:

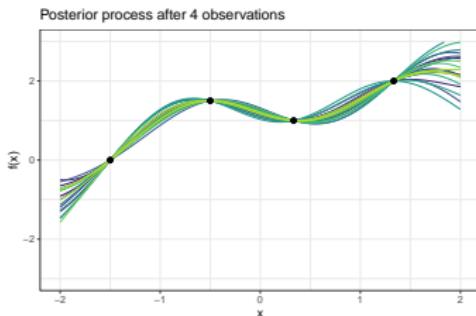
- using kernels that yield sparse \mathbf{K} : cheaper to invert.
 - subsampling the data to estimate θ : $\mathcal{O}(m^3)$ for subset of size m .
 - combining estimates on different subsets of size m :
Bayesian committee, $\mathcal{O}(nm^2)$.
 - using low-rank approximations of \mathbf{K} by using only a representative subset (“inducing points”) of m training data \mathbf{X}_m :
Nyström approximation $\mathbf{K} \approx \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{mn}$,
 $\mathcal{O}(nmk + m^3)$ for a rank-k-approximate inverse of \mathbf{K}_{mm} .
 - exploiting structure in \mathbf{K} induced by the kernel: exact solutions but complicated maths, not applicable for all kernels.
- ... this is still an active area of research.



Introduction to Machine Learning

Gaussian Processes

Mean functions for GPs



Learning goals

- Trends can be modeled via specification of the mean function

THE ROLE OF MEAN FUNCTIONS

- It is common but by no means necessary to consider GPs with a zero-mean function

$$m(\mathbf{x}) \equiv 0$$

- Note that this is not necessarily a drastic limitation, since the mean of the posterior process is not confined to be zero

$$\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{f} \sim \mathcal{N}(\mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{f}, \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_*).$$

- Yet there are several reasons why one might wish to explicitly model a mean function, including interpretability, convenience of expressing prior informations, ...
- When assuming a non-zero mean GP prior $\mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$ with mean $m(\mathbf{x})$, the predictive mean becomes

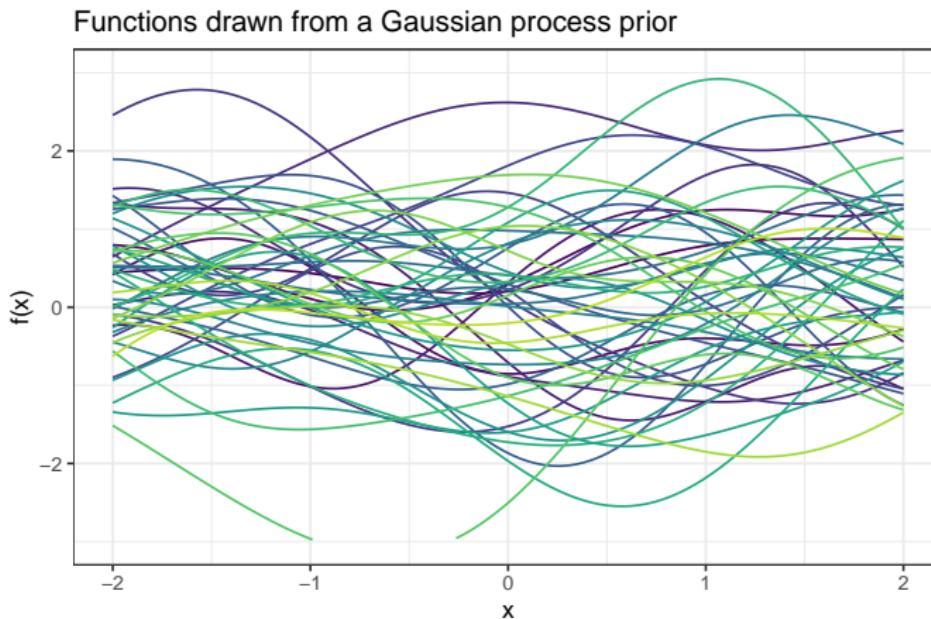
$$m(\mathbf{X}_*) + \mathbf{K}_* \mathbf{K}_y^{-1} (\mathbf{y} - m(\mathbf{X}))$$

while the predictive variance remains unchanged.



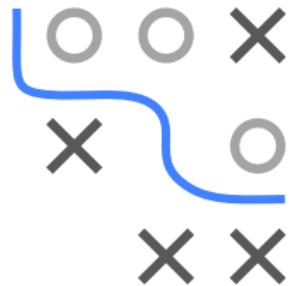
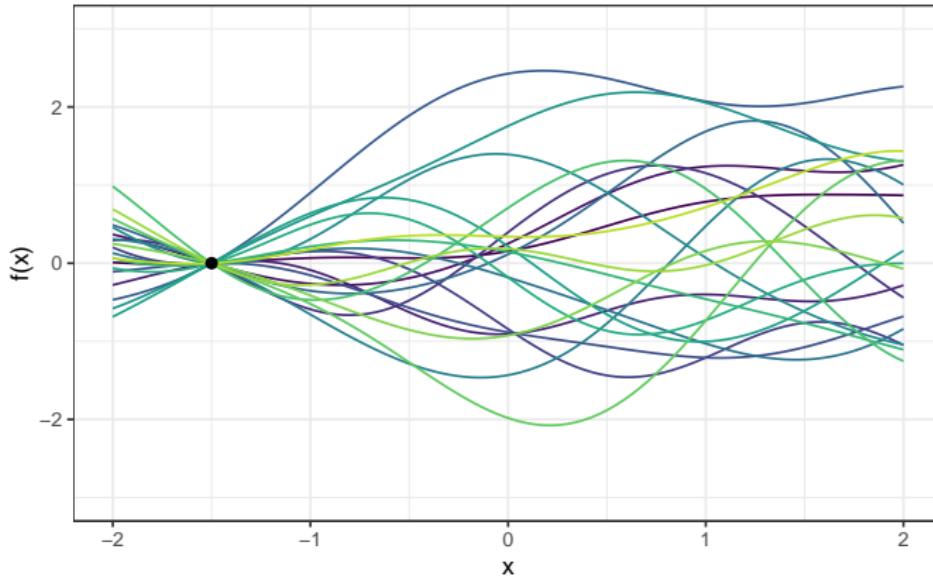
THE ROLE OF MEAN FUNCTIONS

- Gaussian processes with non-zero mean Gaussian process priors are also called Gaussian processes with trend.



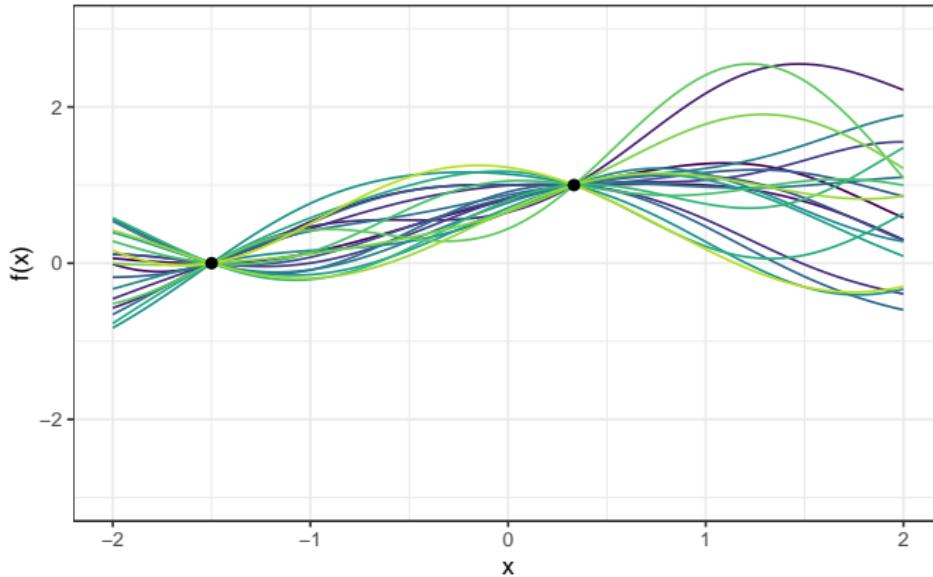
THE ROLE OF MEAN FUNCTIONS

Posterior process after 1 observation



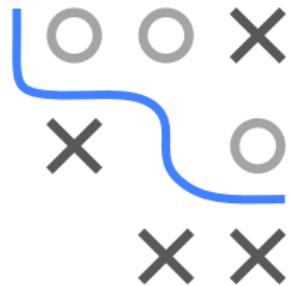
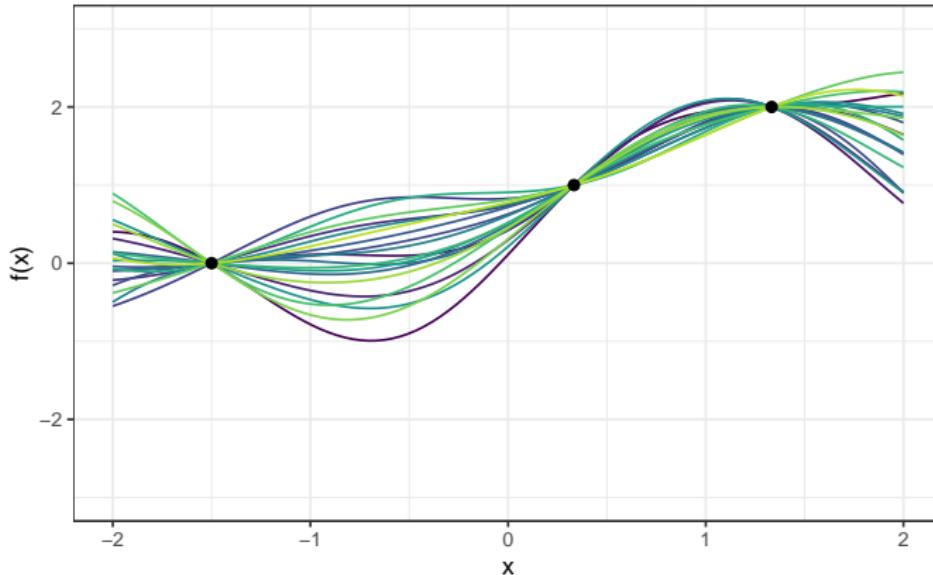
THE ROLE OF MEAN FUNCTIONS

Posterior process after 2 observations



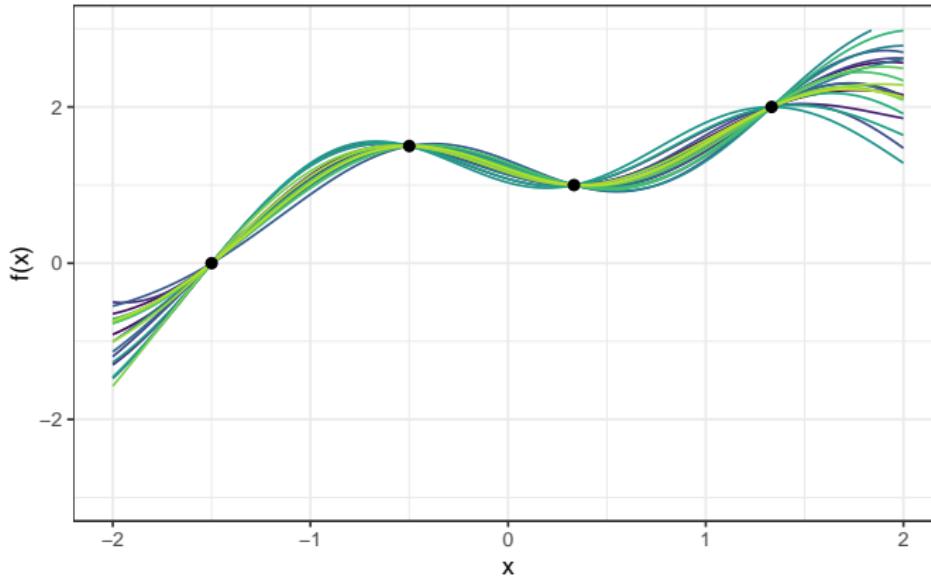
THE ROLE OF MEAN FUNCTIONS

Posterior process after 3 observations



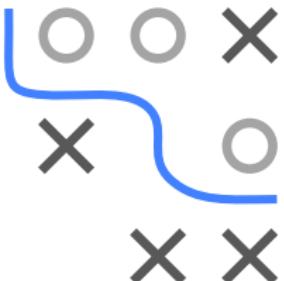
THE ROLE OF MEAN FUNCTIONS

Posterior process after 4 observations



THE ROLE OF MEAN FUNCTIONS

- In practice it can often be difficult to specify a fixed mean function
- In many cases it may be more convenient to specify a few fixed basis functions, whose coefficients, β , are to be inferred from the data
- Consider



$$g(\mathbf{x}) = \mathbf{b}(\mathbf{x})^\top \boldsymbol{\beta} + f(\mathbf{x}), \text{ where } f(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \tilde{\mathbf{x}}))$$

- This formulation expresses that the data is close to a global linear model with the residuals being modelled by a GP.
- For the estimation of $g(\mathbf{x})$ please refer to *Rasmussen, Gaussian Processes for Machine Learning, 2006*