



Supervised Learning

All slides

April 26, 2023

INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

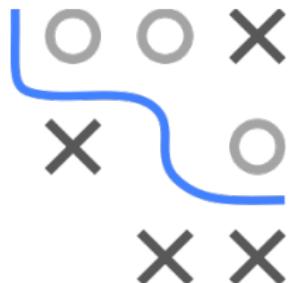
Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

Boosting

Feature Selection

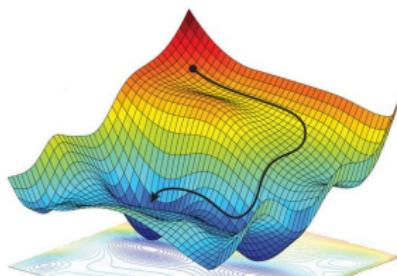




Introduction to Machine Learning

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
- Proper scoring rules

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 in other cases.

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

- The hypothesis space \mathcal{H} is unrestricted. We can choose a $f : \mathcal{X} \rightarrow \mathbb{R}^g$.
- 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

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



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

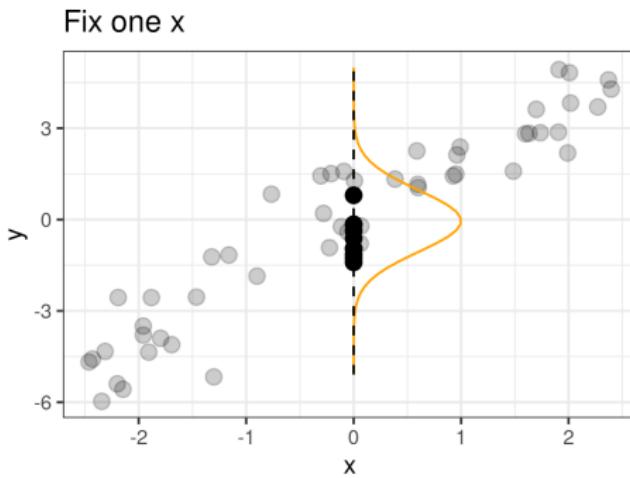
The resulting risk is called **Bayes risk**

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

OPTIMAL POINT-WISE PREDICTIONS

To derive the risk minimizer we usually make use of the following

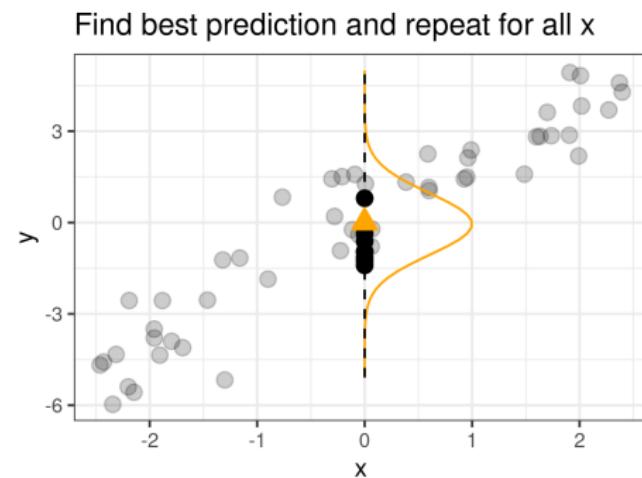
- We can choose $f(\mathbf{x})$ as we want (unrestricted hypothesis space no assumed functional form)
- Consequently, for a fixed value $\mathbf{x} \in \mathcal{X}$ we can select **any** y we want to predict
- So we construct the **point-wise optimizer** for every $\mathbf{x} \in \mathcal{X}$



OPTIMAL POINT-WISE PREDICTIONS

To derive the risk minimizer we usually make use of the following

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



THEORETICAL AND EMPIRICAL RISK

The risk minimizer is mainly a theoretical tool:

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

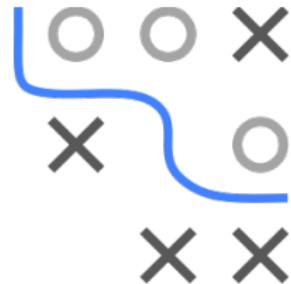
Goal of learning: Train a model \hat{f} for which the true risk $\mathcal{R}_L(\hat{f})$ is close to the Bayes risk \mathcal{R}_L^* . In other words, we want the **Bayes**

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

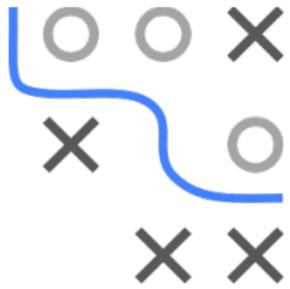
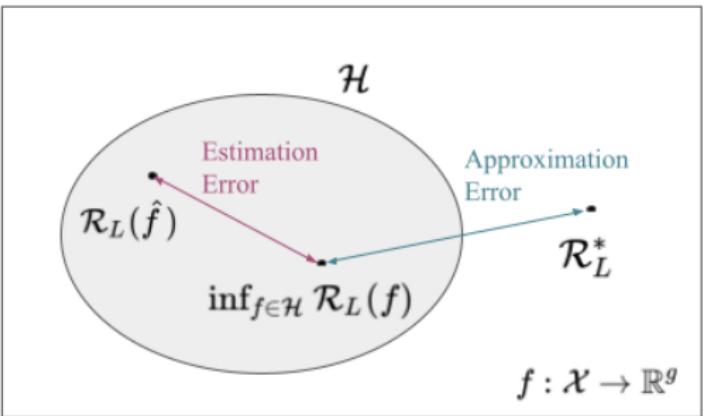
to be as low as possible.

The Bayes regret can be decomposed as follows:

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



ESTIMATION AND APPROXIMATION ERROR



- $\mathcal{R}_L(\hat{f}) - \inf_{f \in \mathcal{H}} \mathcal{R}(f)$ 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, ensures the algorithm returns **the correct model** when given **unlimited data**.



Let $\mathcal{I} : \mathbb{D} \times \Lambda \rightarrow \mathcal{H}$ be a learning algorithm^(*) that takes a train $\mathcal{D}_{\text{train}} \sim \mathbb{P}_{xy}$ 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 ∞ :

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

(*) $\lambda \in \Lambda$ denotes hyperparameters of the learning algorithm.

(UNIVERSALLY) CONSISTENT LEARNERS

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 in choosing 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 \mathbb{P}_{xy} .

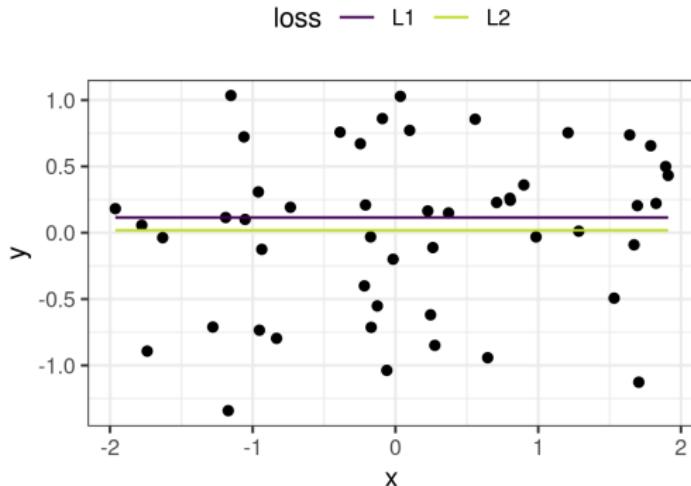
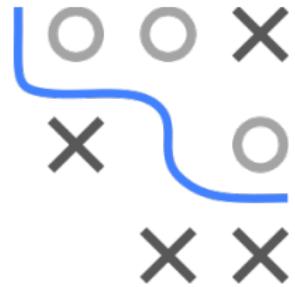
In Stone's famous consistency theorem from 1977, the universal consistency of a weighted average estimator as KNN was proved. Many other ML models have since then been proven to be universally consistent (SVMs, ANNs, etc.).

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 computable empirical lower baseline solution.

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



RISK MINIMIZER AND OPTIMAL CONSTANT

Later, we will derive risk minimizers for various losses.



Name	Risk Minimizer	Optimal Cons
L2	$f^*(\mathbf{x}) = \mathbb{E}_{y \mathbf{x}} [y \mathbf{x}]$	$\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n y_i$
L1	$f^*(\mathbf{x}) = \text{med}_{y \mathbf{x}} [y \mathbf{x}]$	$\hat{f}(\mathbf{x}) = \text{med}(\hat{y}_1, \dots, \hat{y}_n)$
0-1	$h^*(\mathbf{x}) = \arg \max_{l \in \mathcal{Y}} \mathbb{P}(y = l \mathbf{x})$	$\hat{h}(\mathbf{x}) = \text{mode}(\hat{y}_1, \dots, \hat{y}_n)$
Brier	$\pi^*(\mathbf{x}) = \mathbb{P}(y = 1 \mathbf{x})$	$\hat{\pi}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \hat{y}_i$
Bernoulli (on probs)	$\pi^*(\mathbf{x}) = \mathbb{P}(y = 1 \mathbf{x})$	$\hat{\pi}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \hat{y}_i$
Bernoulli (on scores)	$f^*(\mathbf{x}) = \log \left(\frac{\mathbb{P}(y=1 \mathbf{x})}{1 - \mathbb{P}(y=1 \mathbf{x})} \right)$	$\hat{f}(\mathbf{x}) = \log \frac{\hat{y}_1}{1 - \hat{y}_1}$

We see: For regression, the RMs model the conditional expectation; for classification, they model the median of the underlying distribution. This makes intuitive sense depending on your concept of how to best estimate central location, and how robust this location should be.

RISK MINIMIZER AND OPTIMAL CONSTANT

Later, we will derive risk minimizers for various losses.

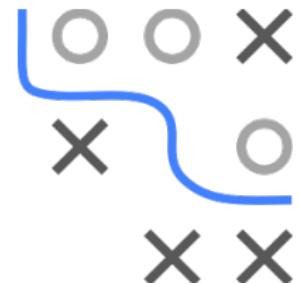


Name	Risk Minimizer	Optimal Cons
L2	$f^*(\mathbf{x}) = \mathbb{E}_{y \mathbf{x}} [y \mathbf{x}]$	$\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n y_i$
L1	$f^*(\mathbf{x}) = \text{med}_{y \mathbf{x}} [y \mathbf{x}]$	$\hat{f}(\mathbf{x}) = \text{med}(\hat{y}_1, \dots, \hat{y}_n)$
0-1	$h^*(\mathbf{x}) = \arg \max_{l \in \mathcal{Y}} \mathbb{P}(y = l \mathbf{x})$	$\hat{h}(\mathbf{x}) = \text{mode}(\hat{y}_1, \dots, \hat{y}_n)$
Brier	$\pi^*(\mathbf{x}) = \mathbb{P}(y = 1 \mathbf{x})$	$\hat{\pi}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \hat{y}_i$
Bernoulli (on probs)	$\pi^*(\mathbf{x}) = \mathbb{P}(y = 1 \mathbf{x})$	$\hat{\pi}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \hat{y}_i$
Bernoulli (on scores)	$f^*(\mathbf{x}) = \log \left(\frac{\mathbb{P}(y=1 \mathbf{x})}{1 - \mathbb{P}(y=1 \mathbf{x})} \right)$	$\hat{f}(\mathbf{x}) = \log \frac{\hat{y}_1}{1 - \hat{y}_1}$

For the 0-1 loss, the risk minimizer constructs the **optimal Bayes decision rule**: We predict the class with maximal posterior probability.

RISK MINIMIZER AND OPTIMAL CONSTANT

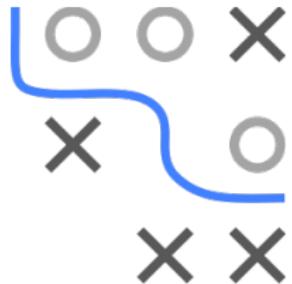
Later, we will derive risk minimizers for various losses.



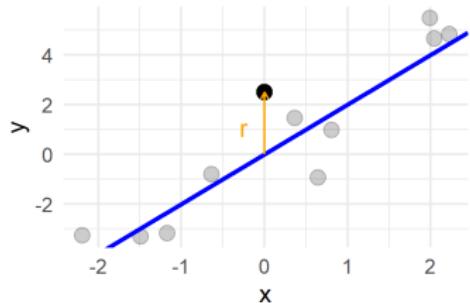
Name	Risk Minimizer	Optimal Const
L2	$f^*(\mathbf{x}) = \mathbb{E}_{y \mathbf{x}} [y \mathbf{x}]$	$\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n y_i$
L1	$f^*(\mathbf{x}) = \text{med}_{y \mathbf{x}} [y \mathbf{x}]$	$\hat{f}(\mathbf{x}) = \text{med}(\hat{y}_1, \dots, \hat{y}_n)$
0-1	$h^*(\mathbf{x}) = \arg \max_{l \in \mathcal{Y}} \mathbb{P}(y = l \mathbf{x})$	$\hat{h}(\mathbf{x}) = \text{mode}(\hat{y}_1, \dots, \hat{y}_n)$
Brier	$\pi^*(\mathbf{x}) = \mathbb{P}(y = 1 \mathbf{x})$	$\hat{\pi}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \hat{y}_i$
Bernoulli (on probs)	$\pi^*(\mathbf{x}) = \mathbb{P}(y = 1 \mathbf{x})$	$\hat{\pi}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n \hat{y}_i$
Bernoulli (on scores)	$f^*(\mathbf{x}) = \log \left(\frac{\mathbb{P}(y=1 \mathbf{x})}{1 - \mathbb{P}(y=1 \mathbf{x})} \right)$	$\hat{f}(\mathbf{x}) = \log \frac{\hat{y}_1}{1 - \hat{y}_1}$

For Brier and Bernoulli, we predict the posterior probabilities (of DGP!). Losses that have this desirable property are called **proper scoring (rules)**.

Introduction to Machine Learning



Pseudo-Residuals and Gradient Desc



Learning goals

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

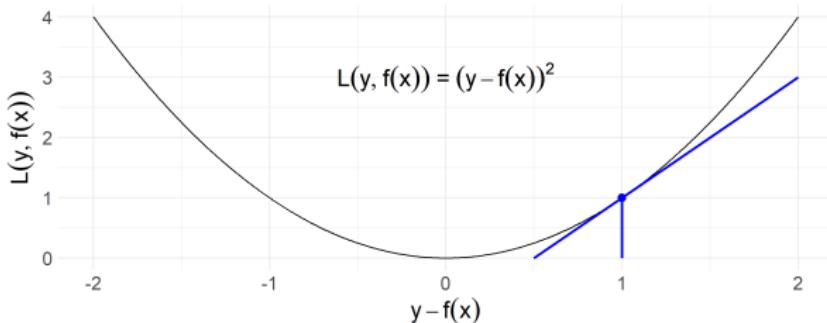
PSEUDO-RESIDUALS

- 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 classification.
- Note that \tilde{r} depends on y and $f(\mathbf{x})$ and L .



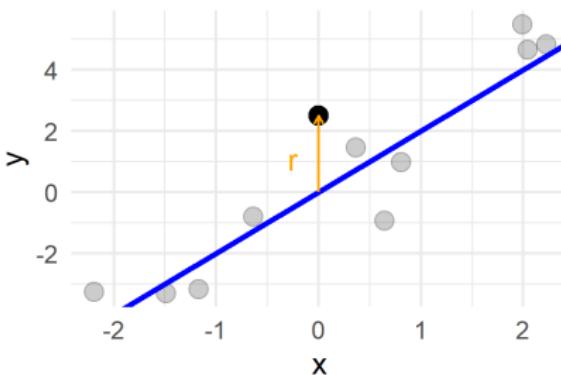
BEST POINT-WISE UPDATE

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} , the best point-wise update is the direction of the residual $r = y - f(\mathbf{x})$



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



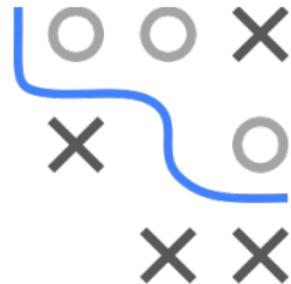
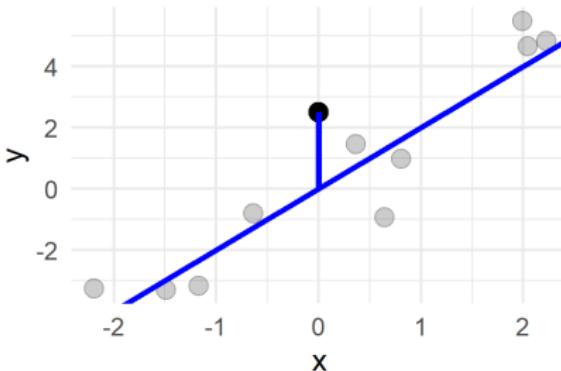
BEST POINT-WISE UPDATE

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} , 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.



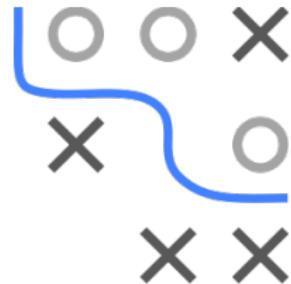
APPROXIMATE BEST POINT-WISE UPDATE

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-residual 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:

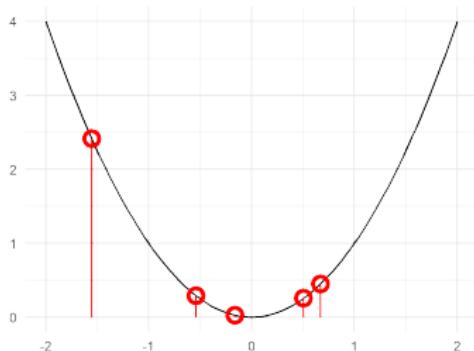
$$\begin{aligned}\nabla_{\theta} \mathcal{R}_{\text{emp}}(\theta) &= \sum_{i=1}^n \frac{\partial L(y^{(i)}, f)}{\partial f} \Bigg|_{f=f(\mathbf{x}^{(i)} | \theta)} \cdot \nabla_{\theta} f(\mathbf{x}^{(i)} | \theta) \\ &= - \sum_{i=1}^n \tilde{r}^{(i)} \cdot \nabla_{\theta} f(\mathbf{x}^{(i)} | \theta).\end{aligned}$$

- Hence the update is determined by a loss-optimal direction change of the model output and a loss-independent derivative. This is a very flexible, nearly loss-independent formulation.



Introduction to Machine Learning

Regression Losses: L2-loss



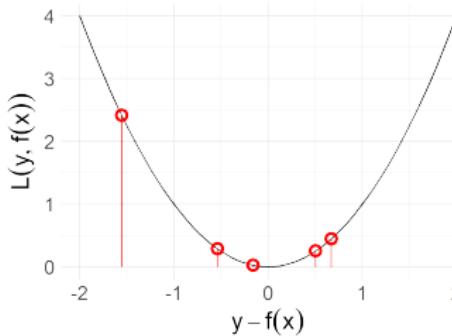
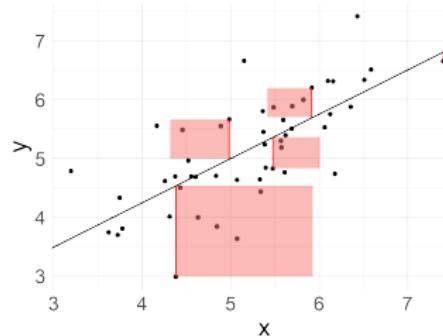
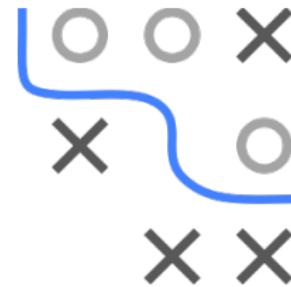
Learning goals

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

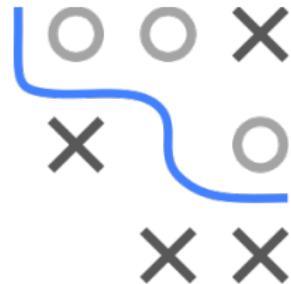
L2-LOSS

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

- Tries to reduce large residuals (if residual is twice as large 4 times as large), hence outliers in y can become problem
- Analytic properties: convex, differentiable (gradient no problem for loss minimization)
- Residuals = Pseudo-residuals: $\tilde{r} = -\frac{\partial 0.5(y-f(\mathbf{x}))^2}{\partial f(\mathbf{x})} = y - f(\mathbf{x}) = r$



L2-LOSS: RISK MINIMIZER



Let us consider the (true) risk for $\mathcal{Y} = \mathbb{R}$ and the L2-Loss
 $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$ with unrestricted $\mathcal{H} = \{f : \mathcal{X} \rightarrow \mathbb{R}^g\}$.

- By the law of total expectation

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

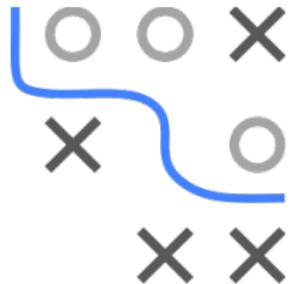
- Since \mathcal{H} is unrestricted we can choose f as we wish: At any $\mathbf{x} = \mathbf{x}$ we can predict any value c we want. The best point-prediction is the conditional mean

$$f^*(\mathbf{x}) = \operatorname{argmin}_c \mathbb{E}_{y|\mathbf{x}} [(y - c)^2 \mid \mathbf{x} = \mathbf{x}] \stackrel{(*)}{=} \mathbb{E}_{y|\mathbf{x}} [y \mid :$$

L2-LOSS: RISK MINIMIZER

(*) follows from:

$$\begin{aligned}\operatorname{argmin}_c \mathbb{E} [(y - c)^2] &= \operatorname{argmin}_c \underbrace{\mathbb{E} [(y - c)^2] - (\mathbb{E}[y] - c)^2}_{= \operatorname{Var}[y - c] = \operatorname{Var}[y]} + \\ &= \operatorname{argmin}_c \operatorname{Var}[y] + (\mathbb{E}[y] - c)^2 = \mathbb{E}[y].\end{aligned}$$



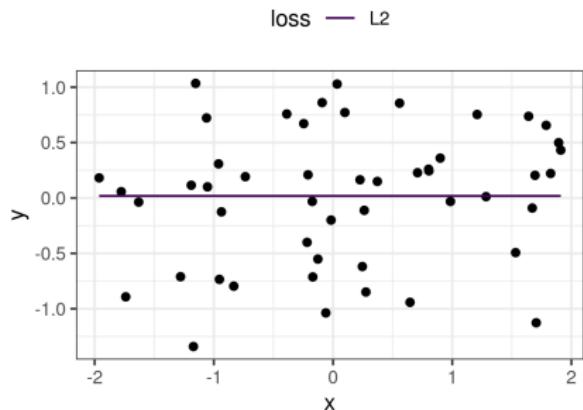
L2-LOSS: OPTIMAL CONSTANT MODEL

The optimal constant model in terms of the (theoretical) risk for loss is the expected value over y :

$$f(\mathbf{x}) = \mathbb{E}_{y | \mathbf{x}} [y | \mathbf{x}] \stackrel{\text{drop } \mathbf{x}}{=} \mathbb{E}_y [y]$$



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



L2-LOSS: OPTIMAL CONSTANT MODEL



Proof:

For the optimal constant model $f(\mathbf{x}) = \theta$ for the L2-loss $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$ 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:

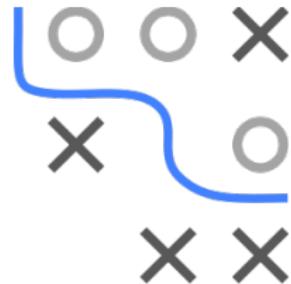
$$\frac{\partial \mathcal{R}_{\text{emp}}(\theta)}{\partial \theta} = -2 \sum_{i=1}^n (y^{(i)} - \theta) \stackrel{!}{=} 0$$

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

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

L2 LOSS MEANS MINIMIZING VARIANCE

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



What happens if we calculate the incurred loss of $\hat{\theta} = \bar{y}$

That's obviously $\mathcal{R}_{\text{emp}} = \sum_{i=1}^n (y^{(i)} - \bar{y})^2$.

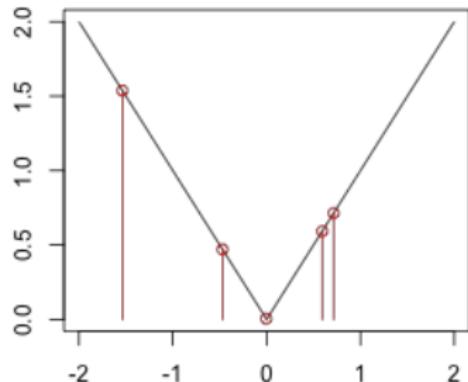
Average this sum by $\frac{1}{n}$ or $\frac{1}{n-1}$, and we get the empirical variance.

The same holds true for the pointwise construction / conditional distribution as considered in the slides before.



Introduction to Machine Learning

Regression Losses: L1-loss

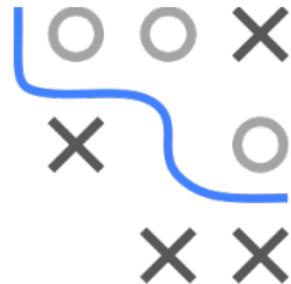


Learning goals

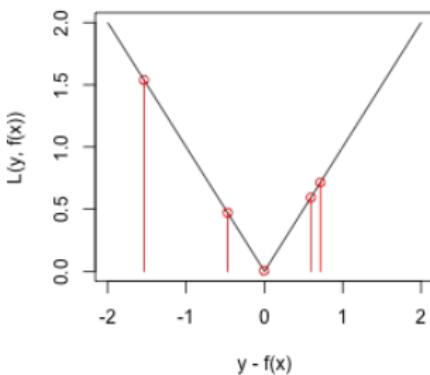
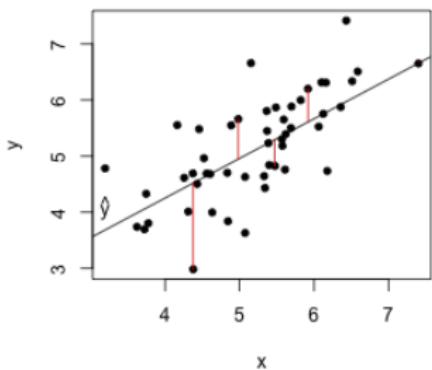
- Derive the risk minimizer of the L1-loss
- Derive the optimal constant model for the L1-loss

L1-LOSS

$$L(y, f(\mathbf{x})) = |y - f(\mathbf{x})|$$



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



L1-LOSS: RISK MINIMIZER

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



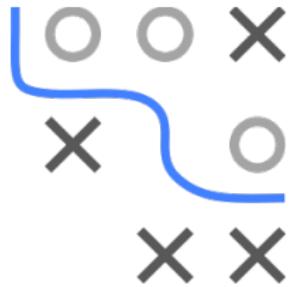
- We use the law of total expectation

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

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

$$\hat{f}(\mathbf{x}) = \operatorname{argmin}_c \mathbb{E}_{y|\mathbf{x}} [|y - c|] = \operatorname{med}_{y|\mathbf{x}} [y \mid \mathbf{x}] .$$

L1-LOSS: RISK MINIMIZER



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

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

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 $c = \operatorname{med}_y(y)$.

L1-LOSS: RISK MINIMIZER

* Note that since we are computing the derivative w.r.t. the integration bound 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

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

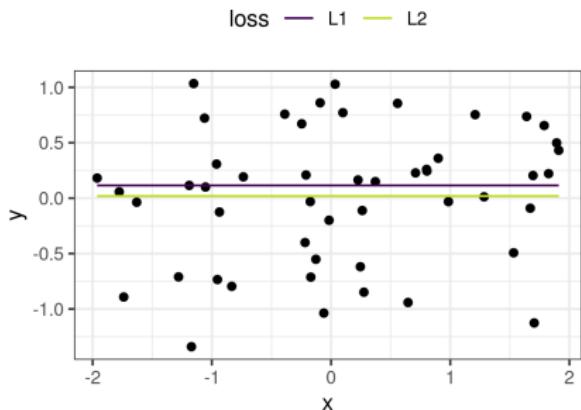
L1-LOSS: OPTIMAL CONSTANT MODEL

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



$$f(\mathbf{x}) = \text{med}_{y|x} [y | \mathbf{x}] \stackrel{\text{drop } \mathbf{x}}{=} \text{med}_y [y]$$

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



L1-LOSS: OPTIMAL CONSTANT MODEL

Proof:

- Firstly note that for $n = 1$ the median $\hat{\theta} = \text{med}(y^{(1)}) = y^{(1)}$ obviously minimizes the empirical risk \mathcal{R}_{emp} associated to loss L .
- Hence let $n > 1$ in the following: Let



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

for $a, b \in \mathbb{R}$. It holds that

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

Thus, any $\hat{\theta} \in [a, b]$ minimizes $S_{a,b}$.

L1-LOSS: OPTIMAL CONSTANT MODEL

W.l.o.g. assume now that all $y^{(i)}$ are sorted in increasing
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 $\mathcal{I}_{i_{\max}}$.
With this, \mathcal{R}_{emp} can be expressed as

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



L1-LOSS: OPTIMAL CONSTANT MODEL

From this follows that

- for “ n is even”: $\hat{\theta} \in \mathcal{I}_{i_{\max}} = [y^{(n/2)}, y^{(n/2+1)}]$ minimize 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 its minimal for $|y^{((n+1)/2)} - \theta|$ minimizes \mathcal{R}_{emp} ,

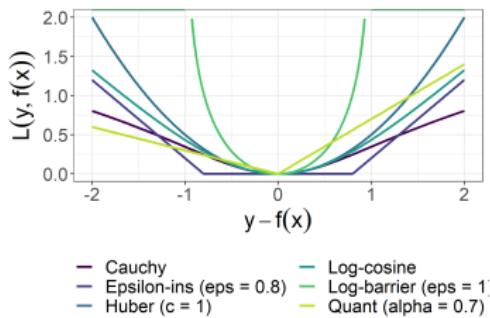
Since the median fulfills these conditions, we can conclude that the median minimizes the $L1$ loss.





Introduction to Machine Learning

Advanced Regression Losses

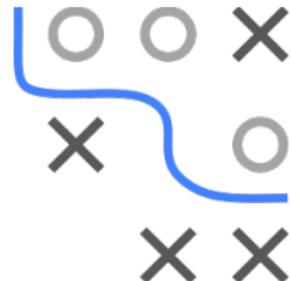


Learning goals

- 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 pc components, to measure errors in a custom way or are designed to have special properties like robustness.



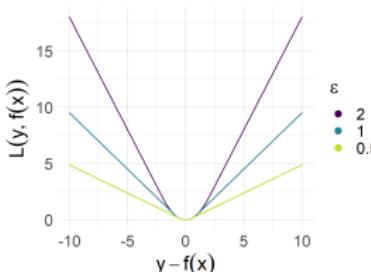
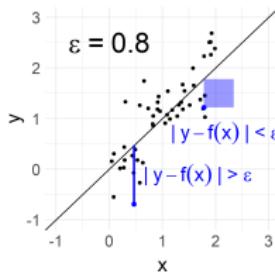
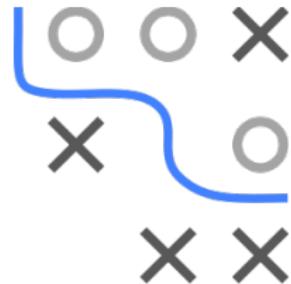
Examples:

- 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 product has no harm, larger deviations do.

HUBER LOSS

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

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

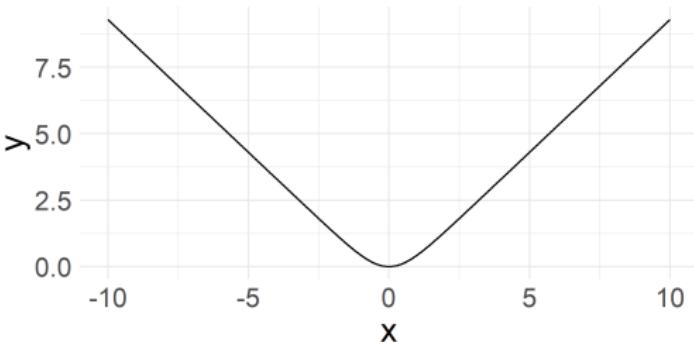


- Risk minimizer and optimal constant do not have a closed-form solution. To fit a model numerical optimization is necessary.
- Solution behaves like **trimmed mean**: a (conditional) mean based on (conditional) quantiles.

LOG-COSH LOSS

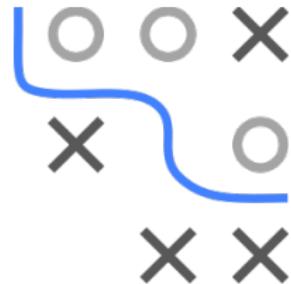
$$L(y, f(\mathbf{x})) = \log(\cosh(|y - f(\mathbf{x})|))$$

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

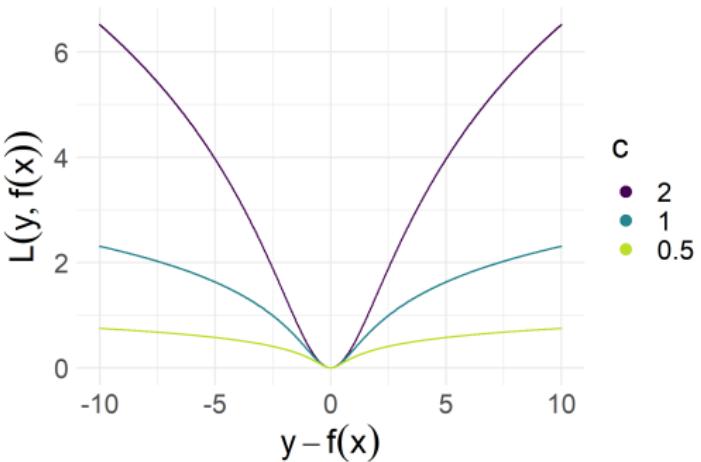


CAUCHY LOSS

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

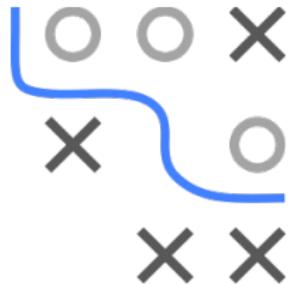


- Particularly robust toward outliers (controllable via c).
- Analytic properties: differentiable, but not convex!

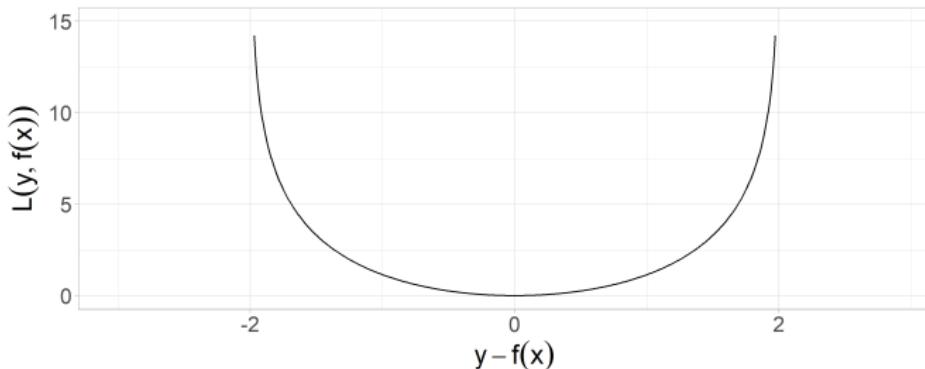


LOG-BARRIER LOSS

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

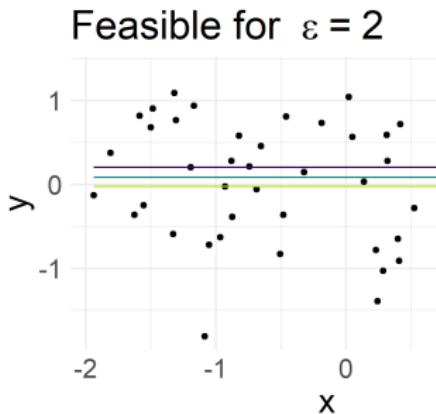
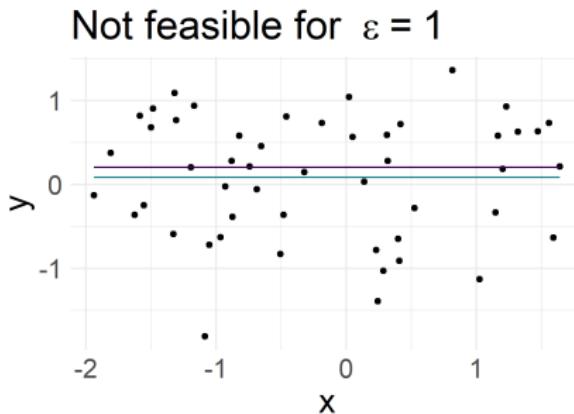


- Behaves like L_2 loss for small residuals.
- We use this if we don't want residuals larger than ϵ at all.
- No guarantee that the risk minimization problem has a solution.
- Plot shows log-barrier loss for $\epsilon = 2$:



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 .



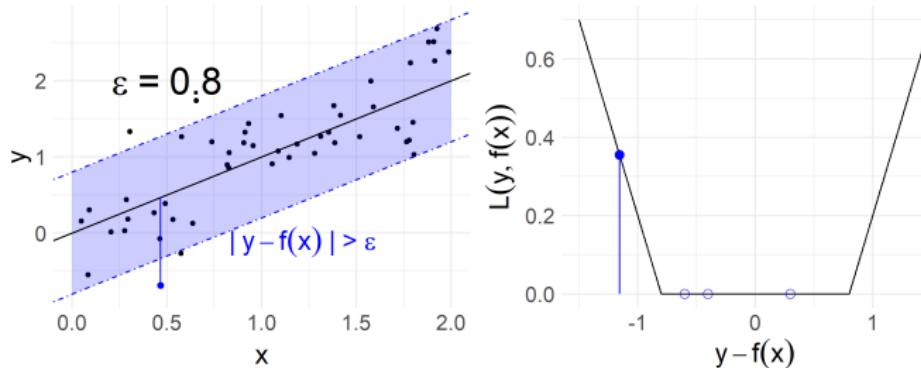
Loss — L1 — L2 — log-barrier

ϵ -INSENSITIVE LOSS

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



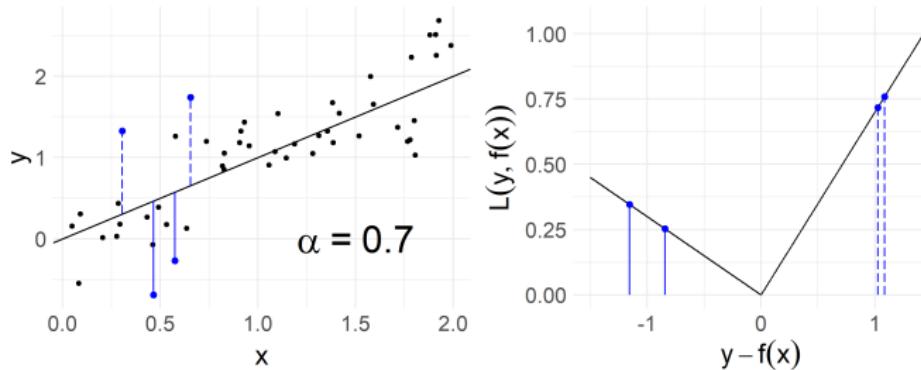
- Modification of L_1 loss, errors below ϵ accepted without penalty.
- Used in SVM regression.
- Properties: convex and not differentiable for $y - f(\mathbf{x}) \in \{-\infty, \infty\}$.



QUANTILE LOSS / PINBALL LOSS

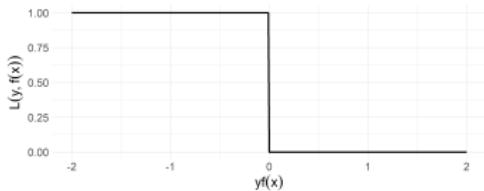
$$L(y, f(\mathbf{x})) = \begin{cases} (1 - \alpha)(f(\mathbf{x}) - y) & \text{if } y < f(\mathbf{x}) \\ \alpha(y - f(\mathbf{x})) & \text{if } y \geq f(\mathbf{x}) \end{cases}, \quad \alpha \in (0, 1]$$

- Extension of L_1 loss (equal to L_1 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



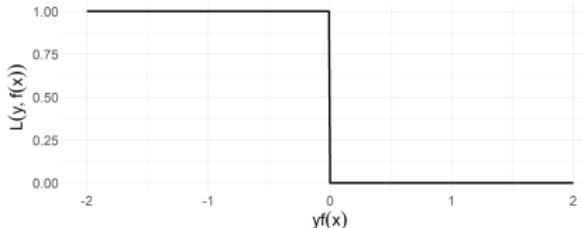
- Let us first consider a discrete classifier $h(\mathbf{x}) : \mathcal{X} \rightarrow \mathcal{Y}$.
- The most natural choice for $L(y, h(\mathbf{x}))$ is the 0-1-loss

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

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

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

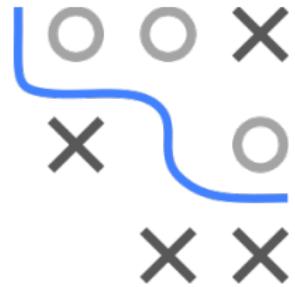
- Analytic properties: Not continuous, even for linear f the optimization problem is NP-hard and close to intractable.



0-1-LOSS: RISK MINIMIZER

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

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

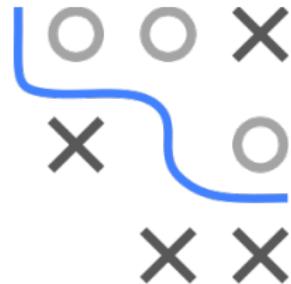


with $\mathbb{P}(y = k \mid \mathbf{x} = \mathbf{x})$ the posterior probability for class k . For the case we denote $\eta(\mathbf{x}) := \mathbb{P}(y = 1 \mid \mathbf{x} = \mathbf{x})$ and the expression becomes

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

0-1-LOSS: RISK MINIMIZER

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

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



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

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

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

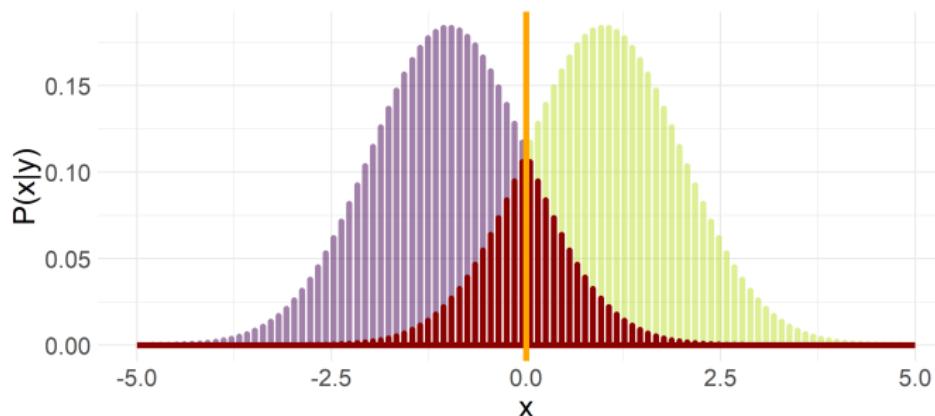
$$\mathcal{R}^* = \mathbb{E}_x [\min(\eta(\mathbf{x}), 1 - \eta(\mathbf{x}))] = 1 - \mathbb{E}_x [\max(\eta(\mathbf{x}), 1 - \eta(\mathbf{x}))]$$

0-1-LOSS: RISK MINIMIZER

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 orange and the Bayes error rate is highlighted as red area.



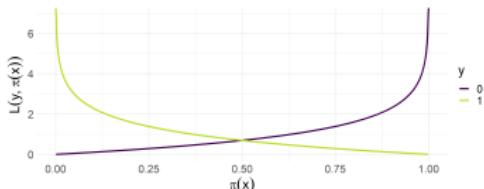


Introduction to Machine Learning

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



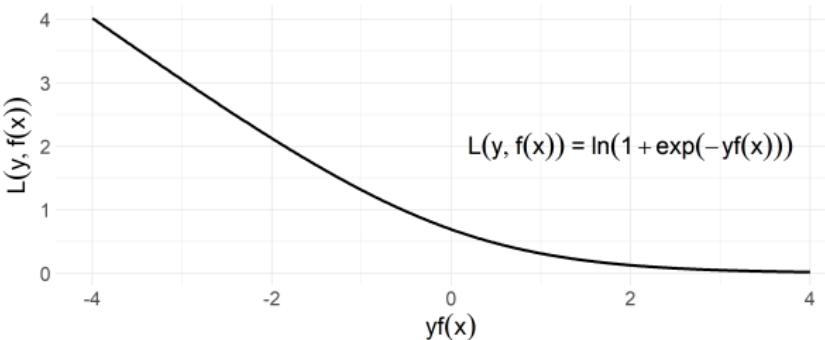
BERNOULLI LOSS

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

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



- Two equivalent formulations for different label encodings
- Negative log-likelihood of Bernoulli model, e.g., logistic regression
- Convex, differentiable
- Pseudo-residuals (0/1 case): $\tilde{r} = y - \frac{1}{1+\exp(-f(\mathbf{x}))}$
Interpretation: L_1 distance between 0/1-labels and posterior probabilities

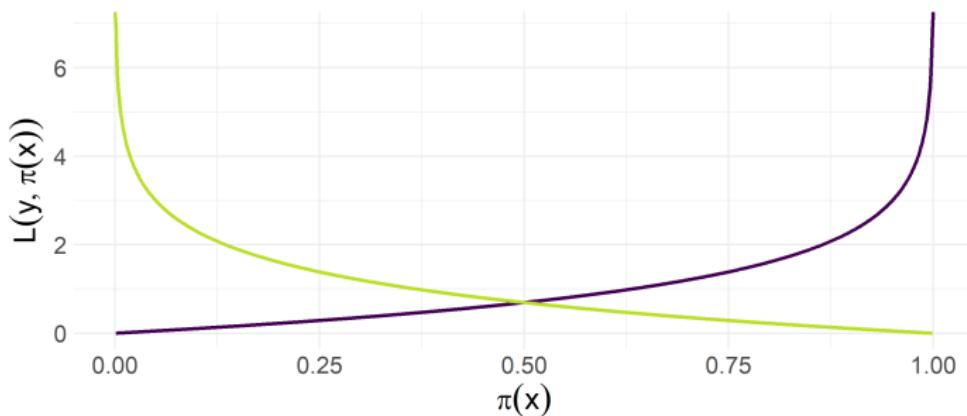


BERNOULLI LOSS ON PROBABILITIES

If scores are transformed into probabilities by the logistic function $\pi(\mathbf{x}) = (1 + \exp(-f(\mathbf{x})))^{-1}$ (or equivalently if $f(x) = \log\left(\frac{\pi(\mathbf{x})}{1-\pi(\mathbf{x})}\right)$, 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:

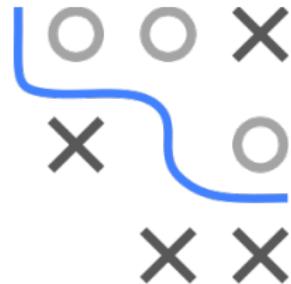
$$\mathcal{R}(f) = \mathbb{E}_{\mathbf{x}} [L(1, \pi(\mathbf{x})) \cdot \eta(\mathbf{x}) + L(0, \pi(\mathbf{x})) \cdot (1 - \eta(\mathbf{x}))],$$

with $\eta(\mathbf{x}) = \mathbb{P}(y = 1 \mid \mathbf{x} = \mathbf{x})$ (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

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

BERNOULLI LOSS: RISK MINIMIZER

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



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

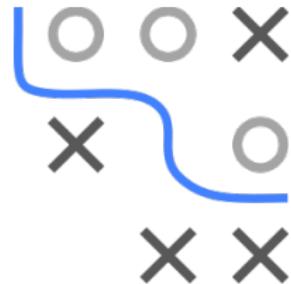
The function is undefined when $\mathbb{P}(y | \mathbf{x} = \mathbf{x}) = 1$ or $\mathbb{P}(y | \mathbf{x} = \mathbf{x}) = 0$, but predicts a smooth curve which grows when $\mathbb{P}(y | \mathbf{x} = \mathbf{x})$ increases and equals 0 when $\mathbb{P}(y | \mathbf{x} = \mathbf{x}) = 0.5$.

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

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



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

BERNOULLI: OPTIMAL CONSTANT MODEL

The optimal constant probability model $\pi(\mathbf{x}) = \theta$ w.r.t. the Bern 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 observe
We can simply prove this again by setting the derivative of the r
and solving for θ .

BERNOULLI: OPTIMAL CONSTANT MODEL

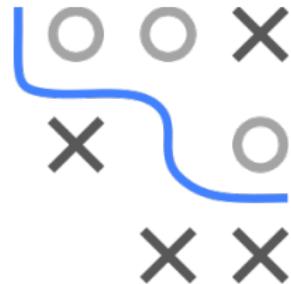
The optimal constant score model $f(\mathbf{x}) = \theta$ w.r.t. the Bernoulli labels from $\mathcal{Y} = \{-1, +1\}$ or $\mathcal{Y} = \{0, 1\}$ is:

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

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

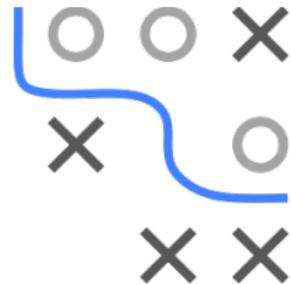
This again shows a tight (and unsurprising) connection of this to log-odds.

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 a distinction whether they are defined on labels $y \in \{0, 1\}$ or $y \in \{-1, +1\}$ 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 minimization and predict the optimal constant. Another approach 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 $\mathcal{N} \subseteq \mathcal{D}$ 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



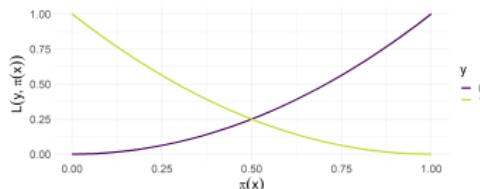
$$\begin{aligned}\mathcal{R}(\mathcal{N}) &= \sum_{(\mathbf{x}, y) \in \mathcal{N}} \left(-\sum_{k=1}^g [y = k] \log \pi_k(\mathbf{x}) \right) \\ &\stackrel{(*)}{=} -\sum_{k=1}^g \sum_{(\mathbf{x}, y) \in \mathcal{N}} [y = k] \log \pi_k^{(\mathcal{N})} \\ &= -\sum_{k=1}^g \log \pi_k^{(\mathcal{N})} \underbrace{\sum_{(\mathbf{x},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

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 pli



Introduction to Machine Learning

Brier Score



Learning goals

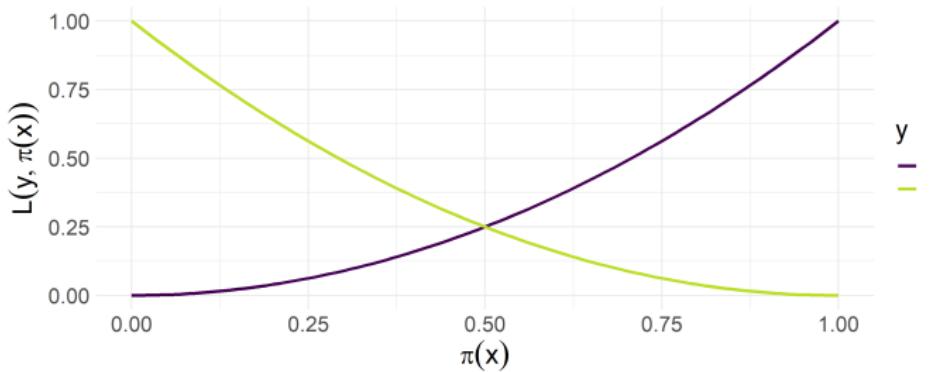
- Know the Brier score
- Derive the risk minimizer
- Derive the optimal constant model
- Understand the connection between Brier score and Gini splitting

BRIER SCORE

The binary Brier score is defined on probabilities $\pi(\mathbf{x}) \in [0, 1]$ & 0-1-encoded labels $y \in \{0, 1\}$ and measures their squared dist (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 predicted probability 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

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

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 MODE

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 minimization and predict the optimal constant. Another approach 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.

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

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) loss function

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



BRIER SCORE MINIMIZATION = GINI SPLITTI



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

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 SPLITTI

We further simplify the expression to

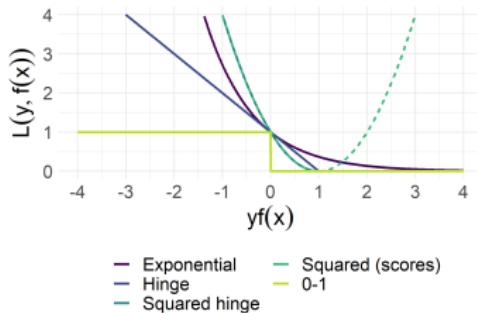
$$\begin{aligned}\mathcal{R}(\mathcal{N}) &= \sum_{k=1}^g n_{\mathcal{N},k} \left(\frac{n_{\mathcal{N}} - n_{\mathcal{N},k}}{n_{\mathcal{N}}} \right)^2 + (n_{\mathcal{N}} - n_{\mathcal{N},k}) \left(\frac{n_{\mathcal{N},k}}{n_{\mathcal{N}}} \right)^2 \\ &= \sum_{k=1}^g \frac{n_{\mathcal{N},k}}{n_{\mathcal{N}}}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



Introduction to Machine Learning



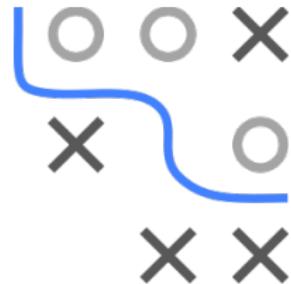
Advanced Classification Losses



Learning goals

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

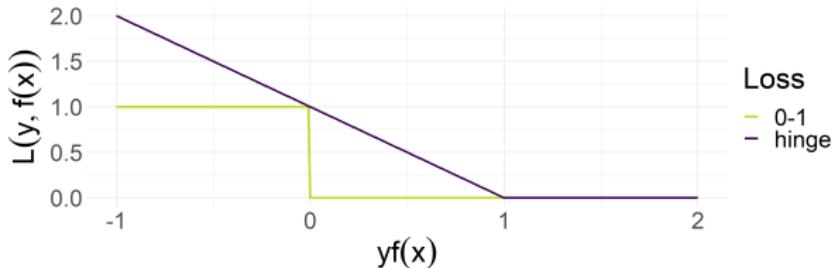
HINGE LOSS



- The intuitive appeal of the 0-1-loss is set off by its analytic 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:

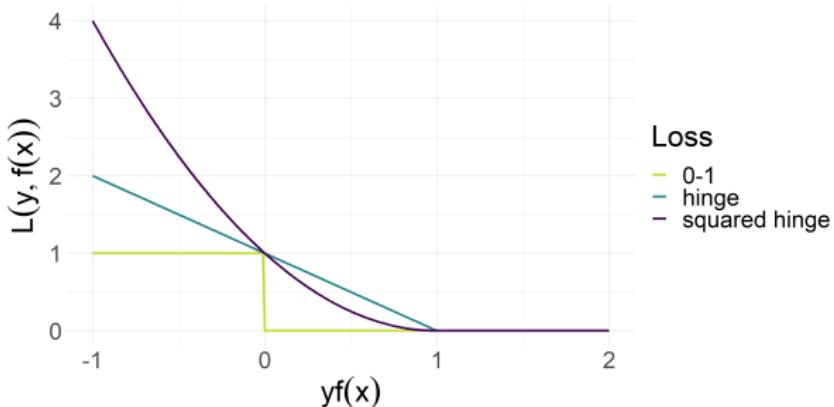
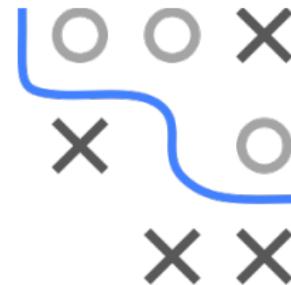


SQUARED HINGE LOSS

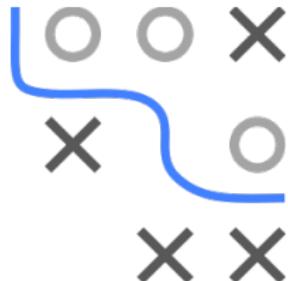
- We can also specify a **squared** version for the hinge loss:

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

- The L_2 form punishes margins $yf(\mathbf{x}) \in (0, 1)$ less severely than the non-squared hinge loss.
- Therefore, it is smoother yet more outlier-sensitive than the non-squared hinge loss.



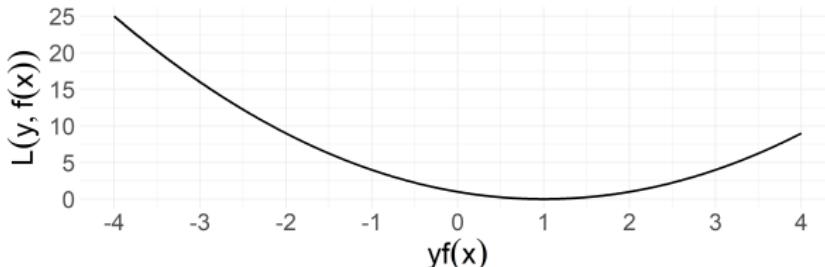
SQUARED LOSS ON SCORES



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

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

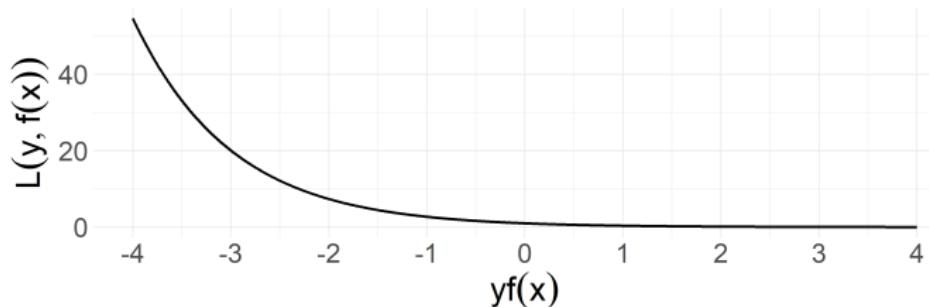
- This loss behaves just like the squared hinge loss for $yf(\mathbf{x})$ but is zero only for $yf(\mathbf{x}) = 1$ and actually increases again larger margins (which is in general not desirable!)



CLASSIFICATION LOSSES: EXPONENTIAL L

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 low confidence; if the prediction is right with a small confidence 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 classification.
- Let $y \in \{-1, +1\}$ with n_- negative and n_+ positive samples.
- The AUC can then be defined as

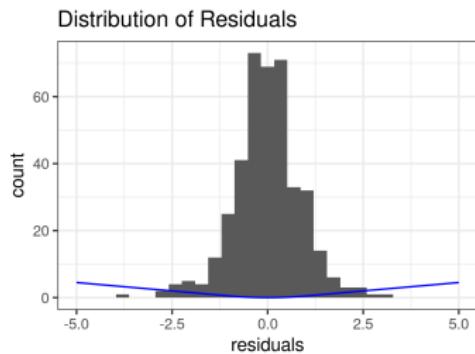
$$AUC = \frac{1}{n_+} \frac{1}{n_-} \sum_{i:y^{(i)}=1} \sum_{j:y^{(j)}=-1} [f^{(i)} > f^{(j)}]$$

- 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
- However, direct optimization of the AUC is numerically more difficult, and might not work as well as using a common loss function tuning for AUC in practice.



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 of between a Gaussian error distribution and the L2 loss

MAXIMUM LIKELIHOOD

Let us approach regression from a maximum likelihood perspective. In the following we assume that

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



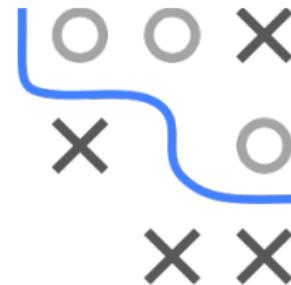
For example, we commonly consider a true underlying relationship with additive noise:

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

where f_{true} is a function that is parameterized by θ and ϵ being a noise term that follows some distribution \mathbb{P}_ϵ , with $\mathbb{E}[\epsilon] = 0$. Further, we assume ϵ to be independent of \mathbf{x} .

MAXIMUM LIKELIHOOD

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



- Then, given data

$$\mathcal{D} = \left(\left(\mathbf{x}^{(1)}, y^{(1)} \right), \dots, \left(\mathbf{x}^{(n)}, y^{(n)} \right) \right)$$

the maximum-likelihood principle is to maximize the **likelihood**

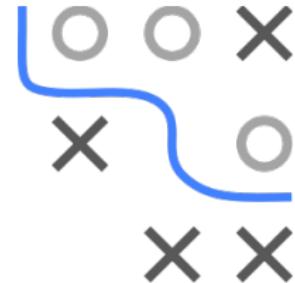
$$\mathcal{L}(\theta) = \prod_{i=1}^n p \left(y^{(i)} | \mathbf{x}^{(i)}, \theta \right)$$

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

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

MAXIMUM LIKELIHOOD

Let us take a machine learning perspective and assume our hypothesis space corresponds to the space of the (parameterized) f_{true} .



- Let us now simply define the negative log-likelihood as **loss function**

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

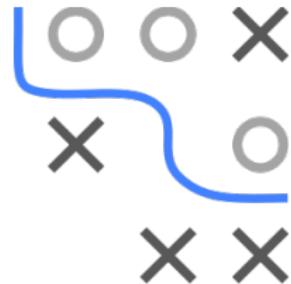
- Maximum-likelihood optimization can be formulated as an empirical risk minimization problem

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

- We can even disregard multiplicative or additive constants loss as they do not change the minimizer.

MAXIMUM LIKELIHOOD

- For every error distribution \mathbb{P}_ϵ we can derive an equivalent function, which leads to the same point estimator for the parameter vector θ as maximum-likelihood.
- **NB:** The other way around does not always work: We cannot derive a probability density function or error distribution corresponding to every loss function – the Hinge loss is a prominent example.



GAUSSIAN ERRORS - L2-LOSS

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

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

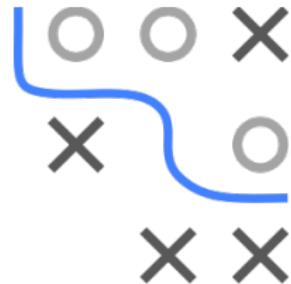
The likelihood is then

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



GAUSSIAN ERRORS - L2-LOSS

It is easy to see that minimizing the negative log-likelihood is equivalent to the *L2*-loss minimization approach since

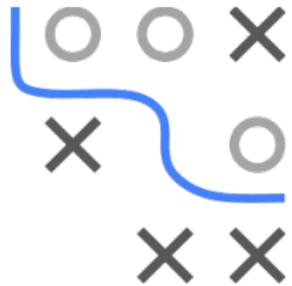


$$\begin{aligned}-\ell(\theta) &= -\log(\mathcal{L}(\theta)) \\&= -\log \left(\exp \left(-\frac{1}{2\sigma^2} \sum_{i=1}^n \left(y^{(i)} - f(\mathbf{x}^{(i)} | \theta) \right)^2 \right) \right) \\&\propto \sum_{i=1}^n \left(y^{(i)} - f(\mathbf{x}^{(i)} | \theta) \right)^2.\end{aligned}$$

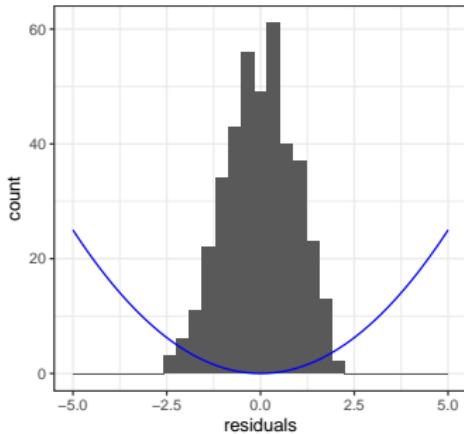
Note: We use \propto as “proportional to ... up to multiplicative and additive constants”

GAUSSIAN ERRORS - L2-LOSS

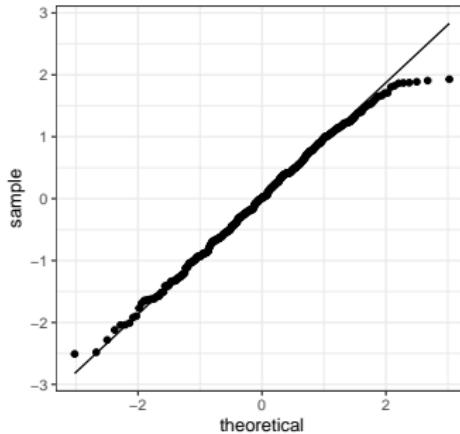
- We simulate data $y \mid \mathbf{x} \sim \mathcal{N}(f_{\text{true}}(\mathbf{x}), 1)$ with $f_{\text{true}} = 0.2 \cdot \mathbf{x}$.
- We can plot the empirical error distribution, i.e. the distribution of the residuals after fitting a regression model w.r.t. L2-loss.
- With the help of a Q-Q-plot we can compare the empirical residuals vs. theoretical quantiles of a Gaussian distribution.



Distribution of Residuals



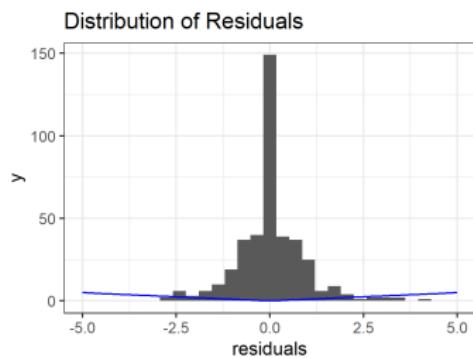
Residuals vs. Quantiles of Error Distribution



Introduction to Machine Learning



Maximum Likelihood Estimation vs. Empirical Risk Minimization



Learning goals

- Learn the correspondence between a Laplacian error distribution and the L1 loss
- Learn that there is no error distribution for the Huber loss
- Learn the correspondence between Bernoulli-distributed targets and the Bernoulli loss

LAPLACE ERRORS - L1-LOSS

Let us assume that errors are Laplacian, i.e. ϵ follows a Laplace distribution which has the density

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



Then

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

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

LAPLACE ERRORS - L1-LOSS

The likelihood is then

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



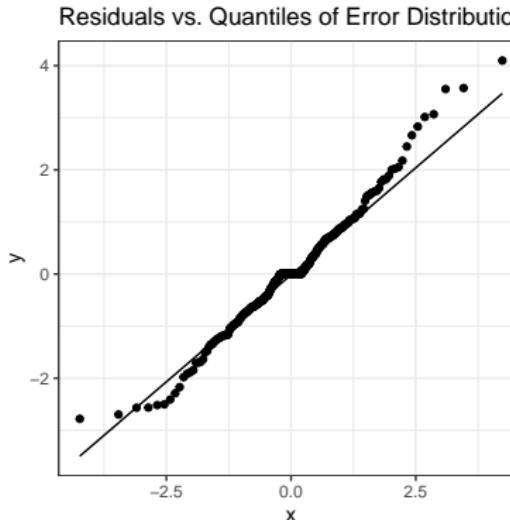
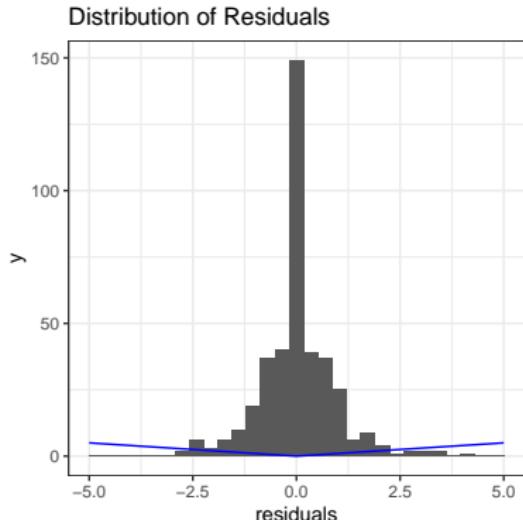
The negative log-likelihood is

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

Minimizing the negative log-likelihood for Laplacian error terms corresponds to empirical risk minimization with L1-loss.

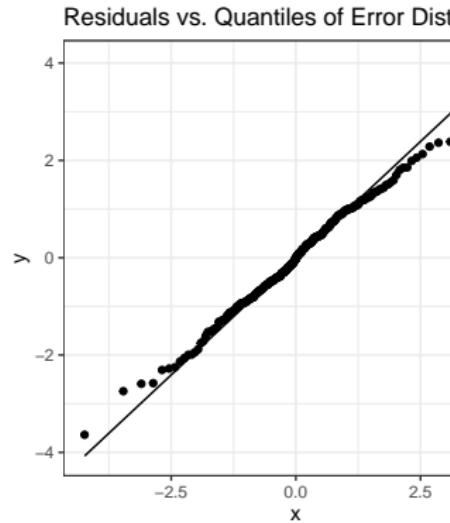
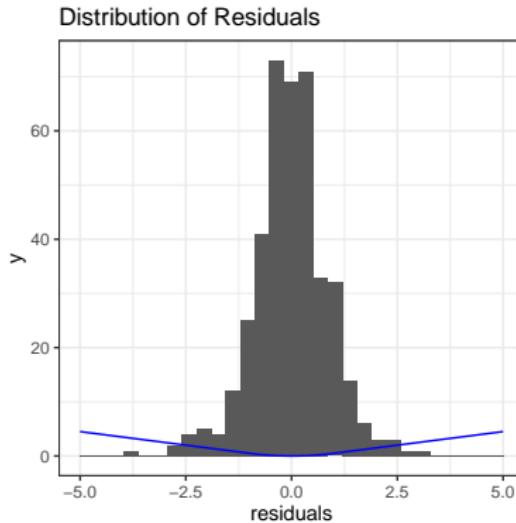
LAPLACE ERRORS - L1-LOSS

- 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. theoretical quantiles of a Laplacian distribution.



OTHER ERROR DISTRIBUTIONS

- There are losses that do not correspond to “real” error densities like the Huber loss. (In the QQ-plot below we show residuals against quantiles of a normal.)



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

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



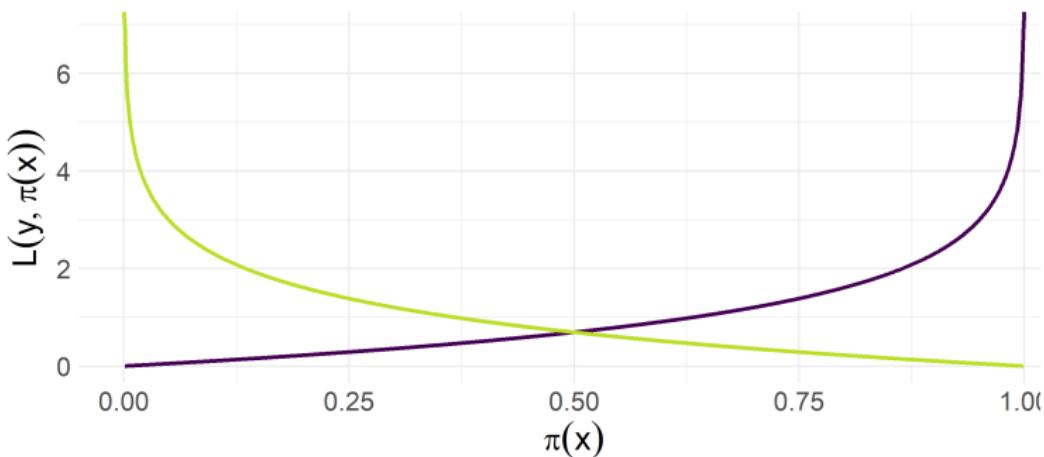
MAXIMUM LIKELIHOOD IN CLASSIFICATION

This gives rise to the following loss function

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



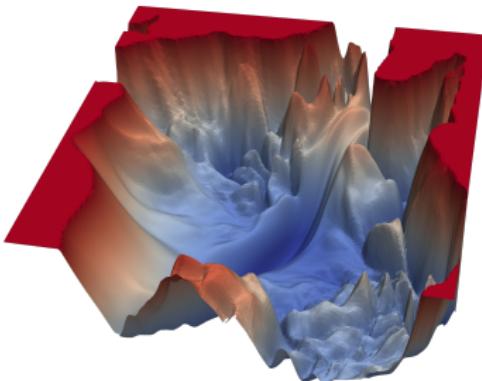
which we introduced as **Bernoulli** loss.





Introduction to Machine Learning

Properties of Loss Functions



Learning goals

- Know the concept of robustness
- Learn about analytical and computational properties of loss functions
- Understand that the loss function may influence convergence of the optimizer

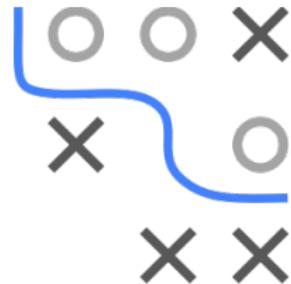
THE ROLE OF LOSS FUNCTIONS

Why should we care about how to choose the loss function $L(y)$

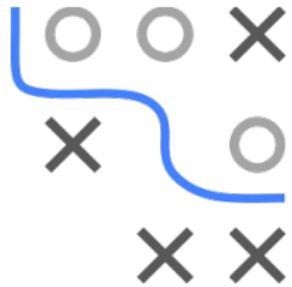
- **Statistical** properties: choice of loss implies statistical assumptions on the distribution of $y | \mathbf{x} = \mathbf{x}$ (see *maximum likelihood estimation vs. empirical risk minimization*).
- **Robustness** properties: some loss functions are more robust towards outliers than others.
- **Analytical** properties: the computational / optimization cost of the problem

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

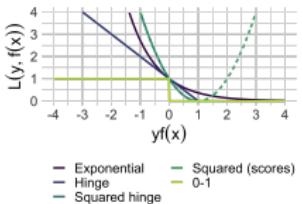
is influenced by the choice of the loss function.



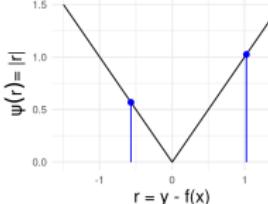
BASIC TYPES OF REGRESSION LOSSES



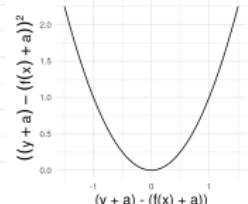
- Regression losses usually only depend on the **residuals** $r := y - f(\mathbf{x})$
- Classification losses are usually expressed in terms of the **margins** $\nu := y \cdot f(\mathbf{x})$.
- Losses are called **symmetric** if $L(y, f(\mathbf{x})) = L(f(\mathbf{x}), y)$.
- A loss is **translation-invariant** if $L(y + a, f(\mathbf{x}) + a) = L(y, f(\mathbf{x}))$
- A 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
 - $\psi(r) = 0 \Leftrightarrow r = 0$.



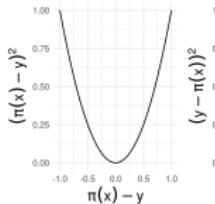
Margin-based losses



Distance-based: L_1



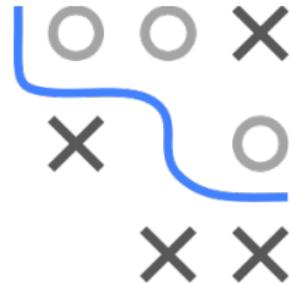
Translation-invariant: L_2



Symmetric: Brili

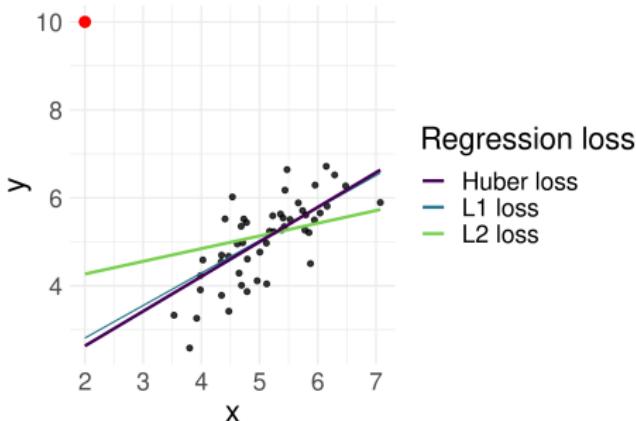
ROBUSTNESS

Outliers (in y) have large residuals $r = y - f(\mathbf{x})$. Some losses are more strongly affected by large residuals than others.



$y - \hat{f}(\mathbf{x})$	L_1	L_2	Huber ($\epsilon = 5$)
1	1	1	0.5
5	5	25	12.5
10	10	100	37.5
50	50	2500	237.5

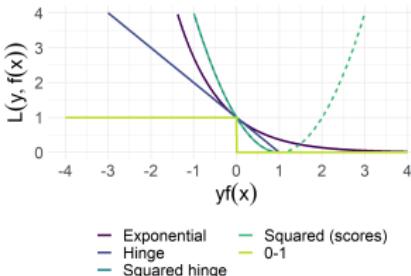
As a consequence, a loss influenced by outliers is less influenced by inliers if the loss is



L_2 is an example for a function that is not very robust towards outliers. It penalizes large residuals more than Huber loss, which are considered robust.

ANALYTICAL PROPERTIES: SMOOTHNESS

- **Smoothness** of a function is a property measured by the number of continuous derivatives.
- A function is said to be C^k if it is k times continuously differentiable. A function is C^∞ if it is continuously differentiable for all orders k .
- Derivative-based methods require a certain level of smoothness of the risk function $\mathcal{R}_{\text{emp}}(\theta)$.
 - If the loss function is not smooth, the risk minimization problem generally not be smooth either.
 - This may require the use of derivative-free optimization (which might not be desirable).
 - Smoothness of objective wrt θ not only depends on $L(y, f(x))$, it also requires smoothness of $f(x)$!



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.



ANALYTICAL PROPERTIES: SMOOTHNESS

Example: Lasso regression

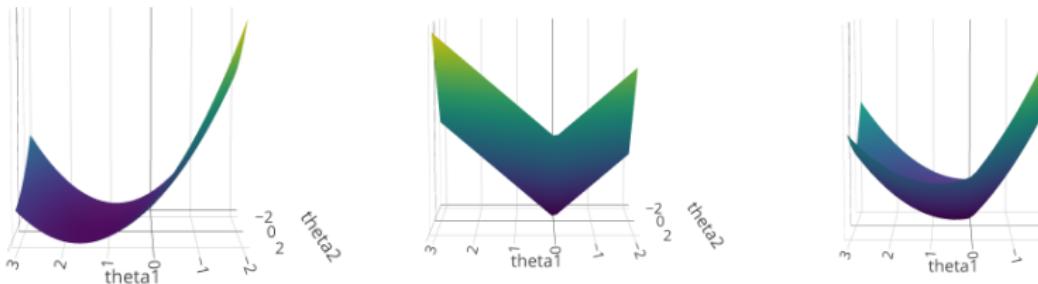
- Problem: Lasso has a non-differentiable objective function

$$\mathcal{R}_{\text{reg}}(\theta) = \|\mathbf{y} - \mathbf{X}\theta\|_2^2 + \lambda\|\theta\|_1 \in \mathcal{C}^0,$$



but many optimization methods are derivative-based, e.g.,

- Gradient descent: requires existence of gradient $\nabla \mathcal{R}_{\text{emp}}(\theta)$
- Newton-Raphson: requires existence of Hessian $\nabla^2 \mathcal{R}_{\text{emp}}(\theta)$
- We must therefore resort to alternative optimization techniques · instance, coordinate descent with subgradients.



Example: $y = x_1 + 1.2x_2 + \epsilon$. Left: unpenalized objective, middle: L1 penalty, right: penalized objective (all as functions of θ_1 and θ_2). We see how the L1 penalty nudges the optimum towards $(0, 0)$ and compromises the original objective's smoothness.

ANALYTICAL 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).

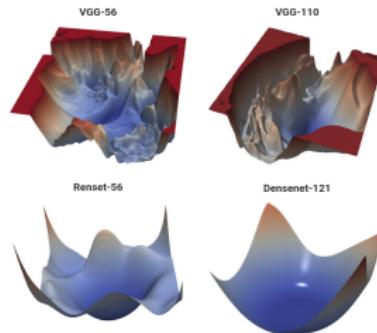
- In optimization, convex problems are desirable because they have a number of convenient properties. In particular, all local minima are global.
 - strictly convex function has at most **one** global minimum (uniqueness).
- For $\mathcal{R}_{\text{emp}} \in \mathcal{C}^2$, \mathcal{R}_{emp} is convex iff Hessian $\nabla^2 \mathcal{R}_{\text{emp}}(\theta)$ is positive semidefinite.

ANALYTICAL PROPERTIES: CONVEXITY

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



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).



ANALYTICAL PROPERTIES: CONVERGENCE

The choice of the loss function may also impact convergence behavior

In the extreme case of **complete separation**, optimization might even entirely. Consider the following scenario:

- Margin-based loss that is antitonic in $y \cdot f$ – for example, **Bernoulli loss**:

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



- Data perfectly separable by our learner

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

as every $\mathbf{x}^{(i)}$ is correctly classified: $f(\mathbf{x}^{(i)} | \theta) < 0$ for $y^{(i)} = -1, > 0$ for

$$\Rightarrow yf(\mathbf{x} | \theta) = |f(\mathbf{x} | \theta)|$$

- f linear in θ – for example, **logistic regression** with $f(\mathbf{x} | \theta) = :$

ANALYTICAL PROPERTIES: CONVERGENCE

- In optimization, e.g., with gradient descent, we can always find ϵ parameters θ' that classifies all samples perfectly.
- But taking a closer look at $\mathcal{R}_{\text{emp}}(\theta)$, we find that the same can be achieved with $2 \cdot \theta'$ – and at lower risk:

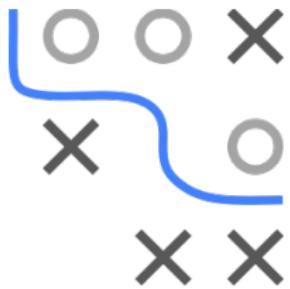
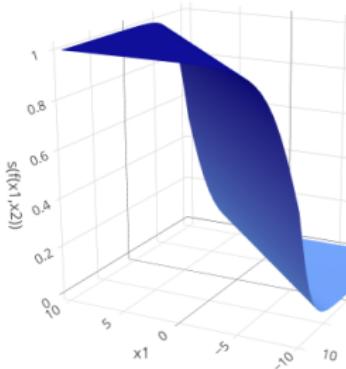
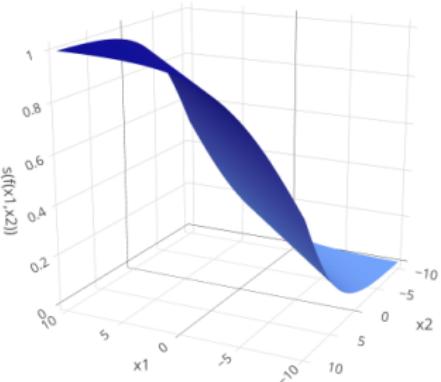
$$\begin{aligned}\mathcal{R}_{\text{emp}}(2 \cdot \theta) &= \sum_{i=1}^n L\left(\left|f\left(\mathbf{x}^{(i)} \mid 2 \cdot \theta\right)\right|\right) = \sum_{i=1}^n L\left(2 \cdot\left|f\left(\mathbf{x}^{(i)} \mid \theta\right)\right|\right) \\ &< \sum_{i=1}^n L\left(\left|f\left(\mathbf{x}^{(i)} \mid \theta\right)\right|\right) = \mathcal{R}_{\text{emp}}(\theta)\end{aligned}$$



- This actually holds true for every $a \cdot \theta$ with $a > 1$.
 - ⇒ By increasing $\|\theta\|$, our loss becomes smaller and smaller, optimization runs infinitely.

ANALYTICAL PROPERTIES: CONVERGENCE

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

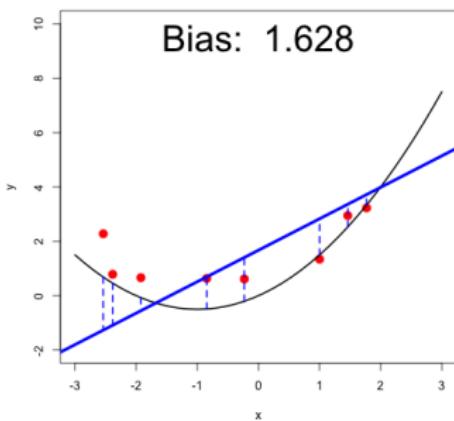


- In practice, data are seldom linearly separable and misclassified examples act as counterweights to increasing parameter values
- Besides, we can apply **regularization** to encourage convergence robust solutions.



Introduction to Machine Learning

Deep Dive: 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 $\hat{f}_{\mathcal{D}}$ 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 n , as well as the independent test observation.

We assume that the data is generated by

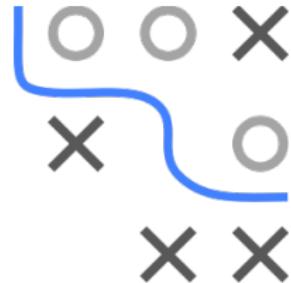
$$y = f_{\text{true}}(\mathbf{x}) + \epsilon,$$

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

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



Let us consider the error $(*)$ conditioned on one fixed test obse (\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



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

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

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

Plug in the definition of y

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

Reorder terms and use the binomial formula

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

BIAS-VARIANCE DECOMPOSITION



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

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

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

BIAS-VARIANCE DECOMPOSITION



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

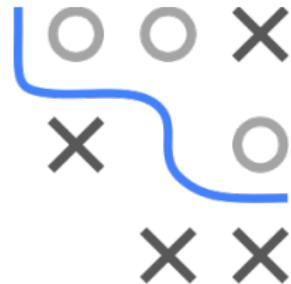
$$\underbrace{\sigma^2}_{\text{Variance of the data}} + \mathbb{E}_{xy} \left[\underbrace{\text{Var}_{\mathcal{D}_n} \left(\hat{f}_{\mathcal{D}_n}(\mathbf{x}) \mid \mathbf{x}, y \right)}_{\text{Variance of learner at } (\mathbf{x}, y)} \right] + \mathbb{E}_{xy} \left[\underbrace{\left(f_{\text{true}}(\mathbf{x})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

- ➊ The first term expresses the variance of the data. This is **noise** in the data. Also called Bayes, intrinsic or irreducible. 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 the locations (underfitting). Models with high capacity typically have low **bias** and *vice versa*.

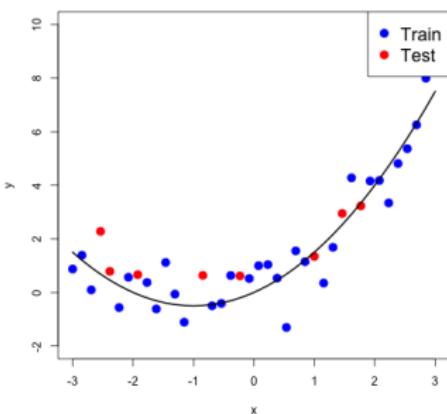
BIAS-VARIANCE DECOMPOSITION

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

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



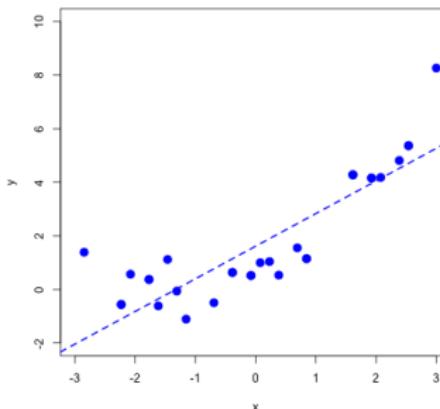
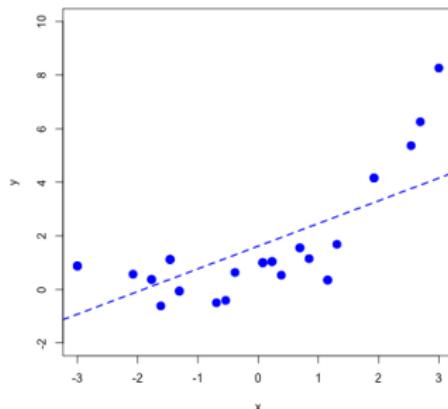
The data is then split into a training set and a test set.



BIAS-VARIANCE DECOMPOSITION

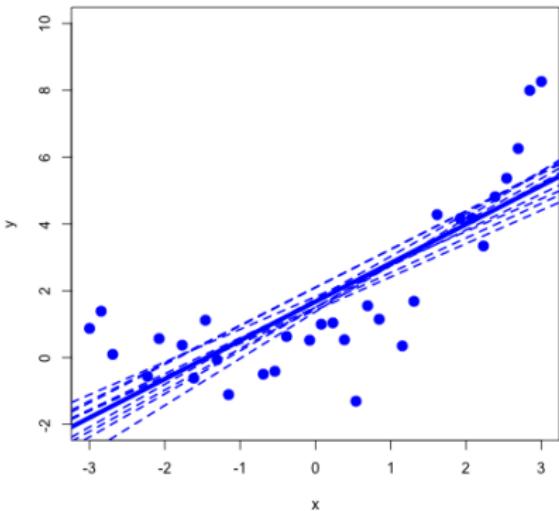
To obtain estimates for the bias and variance, we will train several models by sampling with replacement from the training data. This is commonly known as **bootstrapping**.

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



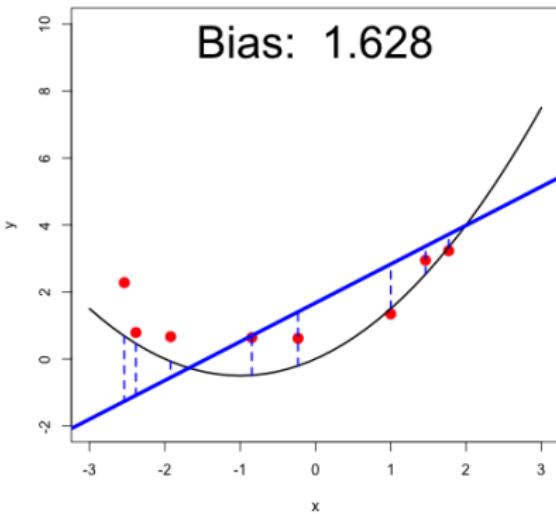
BIAS-VARIANCE DECOMPOSITION

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



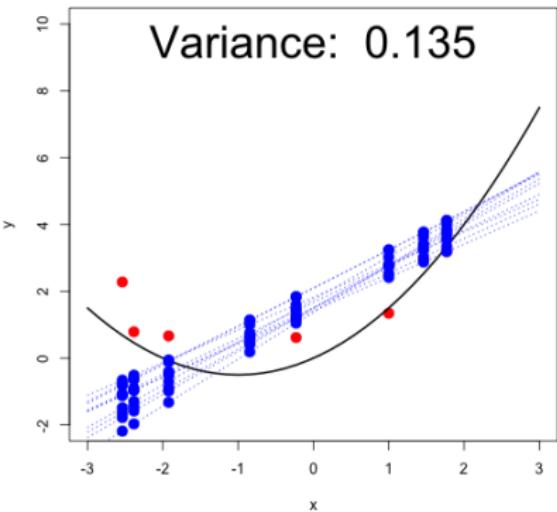
BIAS-VARIANCE DECOMPOSITION

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 model trained at the test point locations.

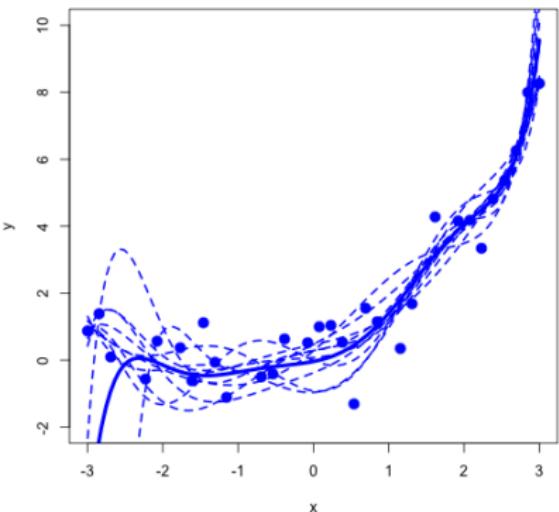


$$GE_n(\mathcal{I}_L) \approx 1 + 1.628 + 0.135 = 2.763$$

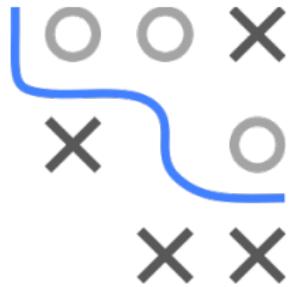
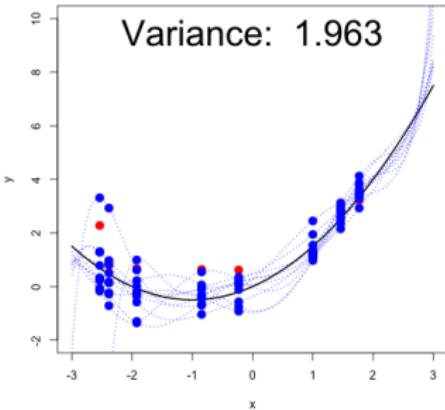
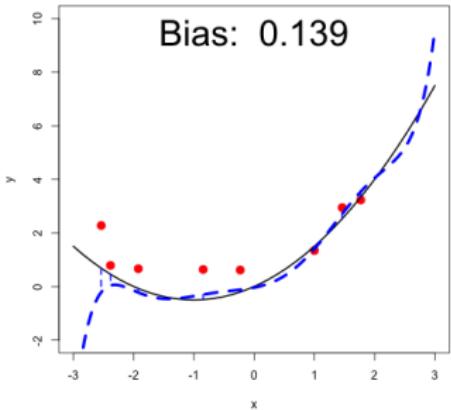
- The biggest component of the generalization error is the bias.

BIAS-VARIANCE DECOMPOSITION

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



BIAS-VARIANCE DECOMPOSITION

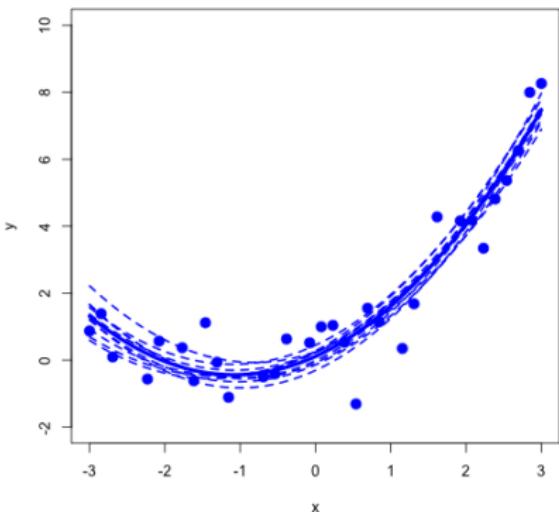


$$GE_n(\mathcal{I}_L) \approx 1 + 0.139 + 1.963 = 3.102$$

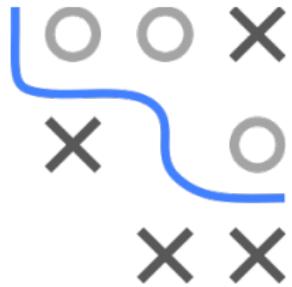
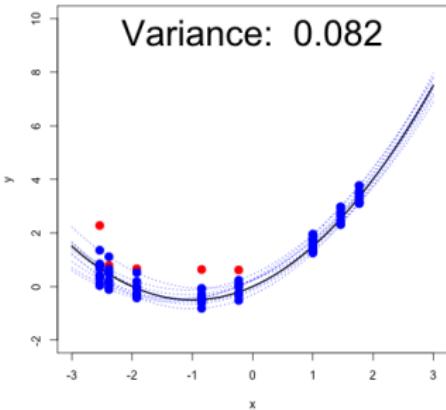
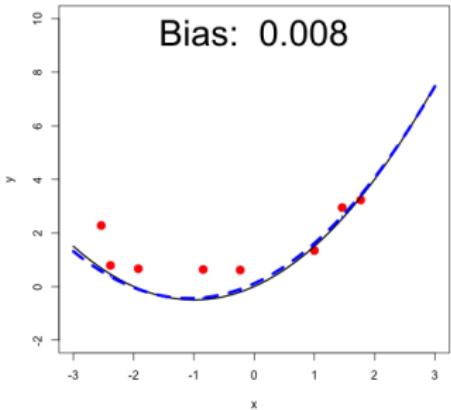
- The generalization error is higher than before
- Even though the bias is lower, the variance of the learner i

BIAS-VARIANCE DECOMPOSITION

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



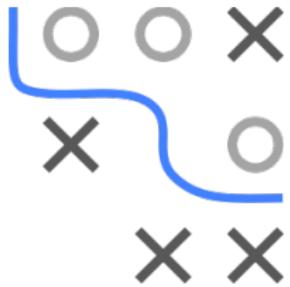
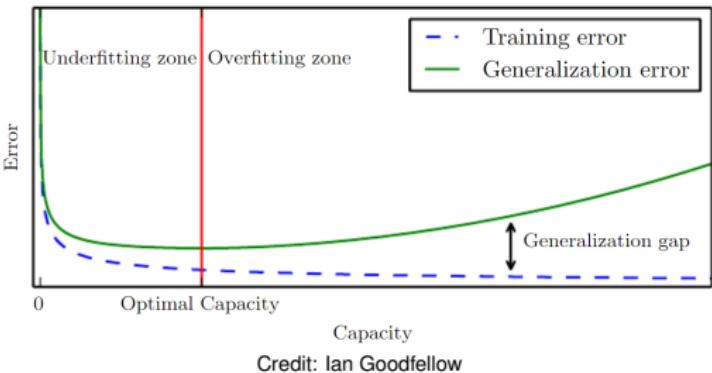
BIAS-VARIANCE DECOMPOSITION



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

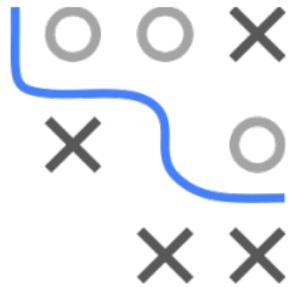
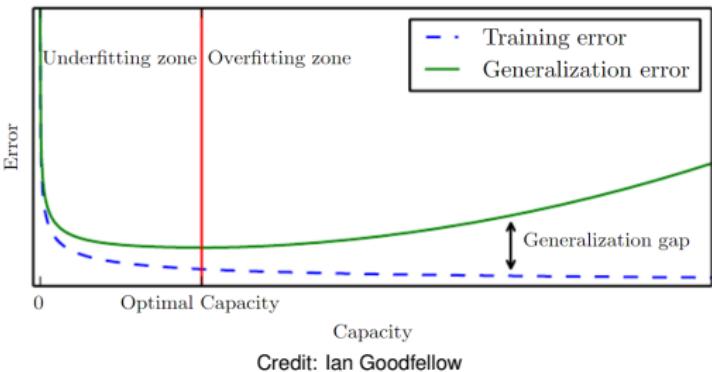
- The generalization error is the lowest at this complexity.
- The variance of the data acts as a lower bound.

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 capacity, determined by the type of hypotheses it can learn.
- The generalization error is minimized when it has the right capacity.
- Even for correctly specified models, the generalization error is lower-bounded by the irreducible noise σ^2 .

INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

Boosting

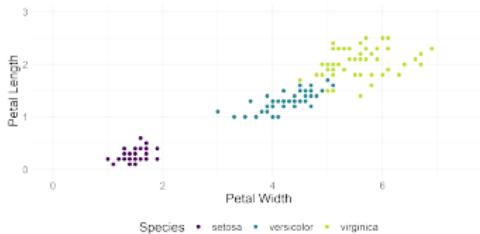
Feature Selection





Introduction to Machine Learning

Multiclass Classification and Losses



Learning goals

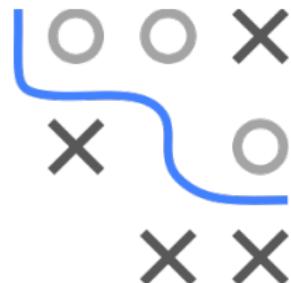
- Know what multiclass means & which types of classifiers exist
- Know the MC 0-1-loss
- Know the MC brier score
- Know the MC logarithmic loss

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 $(\mathbf{x}, y) \sim \mathbb{P}_{xy}$.



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

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 consider losses for **multiclass classification** tasks.
- For multiclass classification, loss functions will be defined
 - vectors of scores

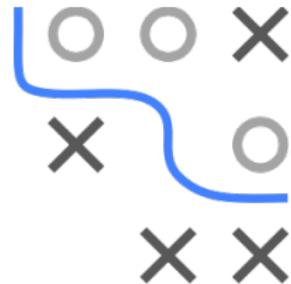
$$f(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_g(\mathbf{x}))$$

- 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	
virginica	0	0	
versicolor	0	1	
versicolor	0	1	
setosa	1	0	
setosa	1	0	





0-1-Loss

0-1-LOSS

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

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

is the Bayes optimal classifier, with

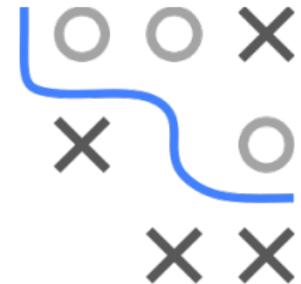
$$\hat{h}(\mathbf{x}) = \arg \max_{l \in \mathcal{Y}} \mathbb{P}(y = l \mid \mathbf{x} = \mathbf{x})$$

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 defined on a vector of class probabilities $(\pi_1(\mathbf{x}), \dots, \pi_g(\mathbf{x}))$

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



The optimal constant model $\pi(\mathbf{x}) = (\theta_1, \dots, \theta_g)$ (outputting a vector of constant class probabilities) is

$$\pi_k(\mathbf{x}) = \arg \min_{\theta_k} \mathcal{R}_{\text{emp}}(\theta) = \arg \min_{\theta_k} \left(\sum_{i=1}^n \sum_{k=1}^g (\mathbb{1}_{\{y^{(i)}=k\}} - \theta_k)^2 \right)$$

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

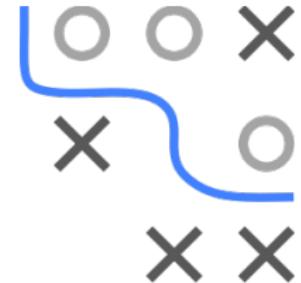
$$\frac{\partial \mathcal{R}_{\text{emp}}(\theta)}{\partial \theta_k} = -2 \cdot \sum_{i=1}^n (\mathbb{1}_{\{y^{(i)}=k\}} - \theta_k) = 0$$

$$\hat{\pi}_k(\mathbf{x}) = \hat{\theta}_k = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{\{y^{(i)}=k\}},$$

being the fraction of class- k observations.

MC BRIER SCORE

Claim: For $g = 2$ the MC Brier score is exactly twice as high as 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 $y = 0$:

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

For $y = 1$:

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

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



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 .

The optimal constant model $\pi(\mathbf{x}) = (\theta_1, \dots, \theta_g)$ (outputting a vector of constant class probabilities) is

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

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, def

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

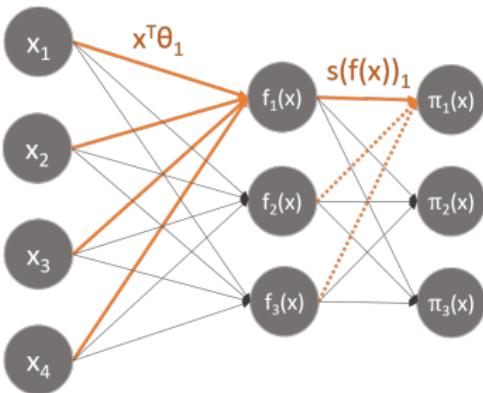
Proof:

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



Introduction to Machine Learning

Softmax Regression



Learning goals

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

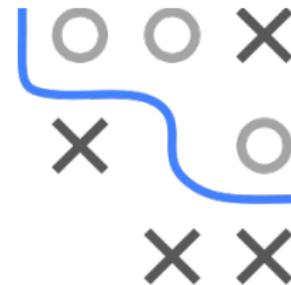
FROM LOGISTIC REGRESSION ...

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 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

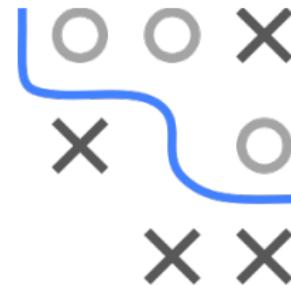
$$f_k(\mathbf{x}) = \boldsymbol{\theta}_k^\top \mathbf{x}, \quad k = 1, 2, \dots, g,$$

each indicating the confidence in class k .

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

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

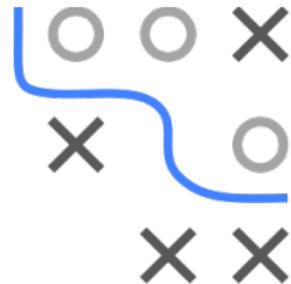
instead of the **logistic** function for $g = 2$. The probabilities are well-defined: $\sum \pi_k(\mathbf{x}) = 1$ and $\pi_k(\mathbf{x}) \in [0, 1]$ for all k .



... 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 the right element!).
- Furthermore, it is invariant to constant offsets in the input:

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

LOGISTIC VS. SOFTMAX REGRESSION



	Logistic Regression	Softmax Regre
\mathcal{Y}	$\{0, 1\}$	$\{1, 2, \dots, g\}$

Discriminant fun.

$$f(\mathbf{x}) = \theta^\top \mathbf{x}$$

$$f_k(\mathbf{x}) = \theta_k^\top \mathbf{x}, k =$$

Probabilities

$$\pi(\mathbf{x}) = \frac{1}{1 + \exp(-\theta^\top \mathbf{x})}$$

$$\pi_k(\mathbf{x}) = \frac{\exp(\theta_k^\top \mathbf{x})}{\sum_{j=1}^g \exp(\theta_j^\top \mathbf{x})}$$

$$L(y, \pi(\mathbf{x}))$$

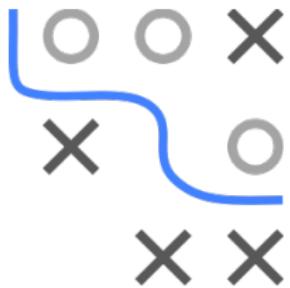
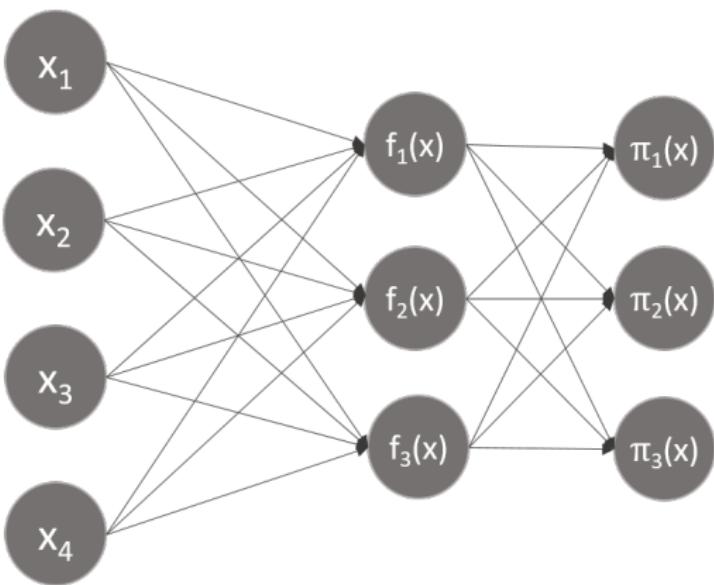
$$-\gamma \log(\pi(\mathbf{x})) - (1 - \gamma) \log(1 - \pi(\mathbf{x}))$$

Multiclass logarithmic loss

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

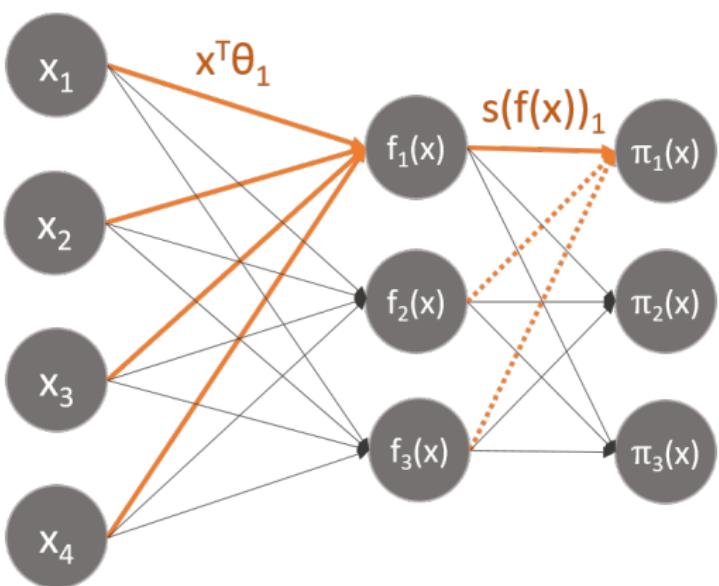
LOGISTIC VS. SOFTMAX REGRESSION

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 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 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 the hypothesis space, but the constrained problem is now convex: there exists exactly one parameter vector for every hypothesis.
- A similar approach is used in many ML models: multiclass logistic regression, naive Bayes, neural networks and boosting.



SOFTMAX: LINEAR DISCRIMINANT FUNCTION

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 $\exp(\cdot)$ (rank-preserving) and divides each element by **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.



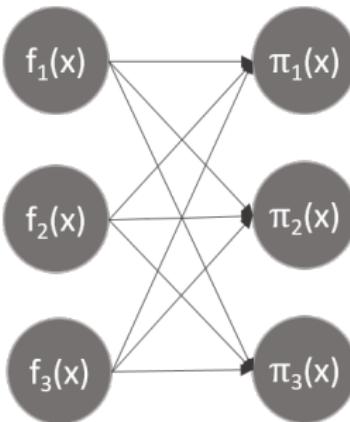
GENERALIZING SOFTMAX REGRESSION

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

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

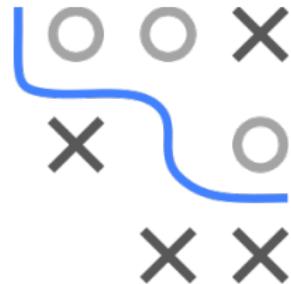
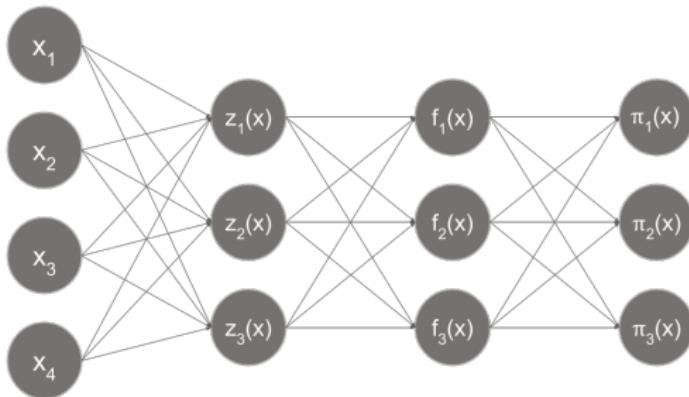


We can choose a multiclass loss and optimize the score function $f_k, k \in \{1, \dots, g\}$ by multivariate minimization. The scores can be transformed to probabilities by the **softmax** function.



GENERALIZING SOFTMAX REGRESSION

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

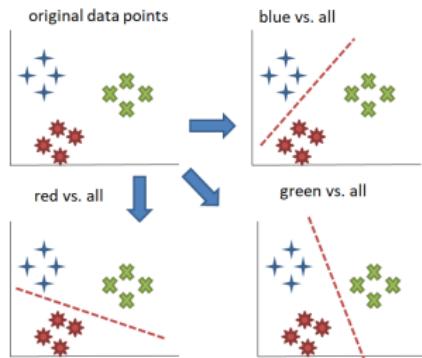


Remark: For more details about neural networks please refer to the lecture **Deep Learning**.



Introduction to Machine Learning

One-vs-Rest and One-vs-One

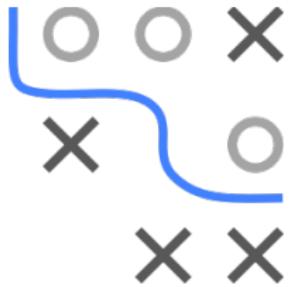


Learning goals

- 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.

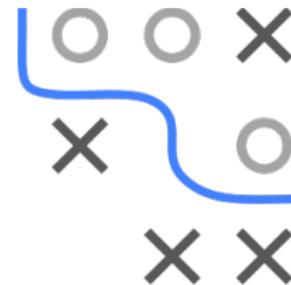


CODEBOOKS

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



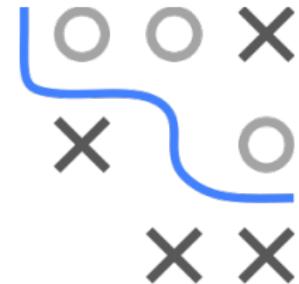
- The k -th column defines how classes of all observations are encoded in the binary subproblem / for binary classifier f_k
- Entry (m, i) takes values $\in \{-1, 0, +1\}$
 - if 0, observations of class $y^{(i)} = m$ are ignored.
 - if 1, observations of class $y^{(i)} = m$ are encoded as 1.
 - if -1,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



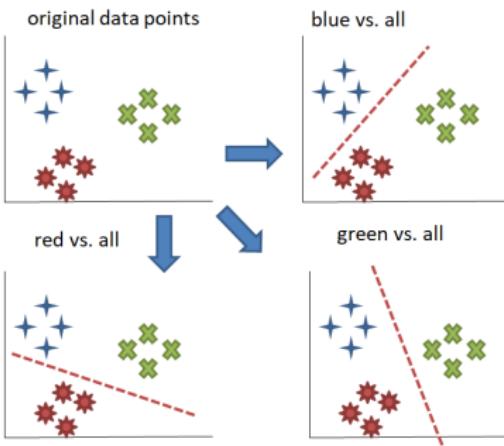
One-vs-Rest

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

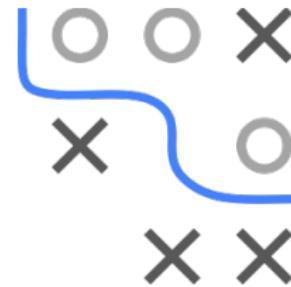


ONE-VS-REST

- Making decisions means applying all classifiers to a sample 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 complete. We could fit a second-stage, multinomial logistic regression model on output scores, so with inputs $(\hat{f}_1(\mathbf{x}^{(i)}), \dots, \hat{f}_g(\mathbf{x}^{(i)}))$ and output $y^{(i)}$ as training data.

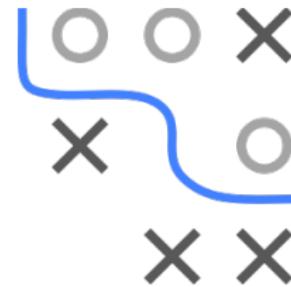




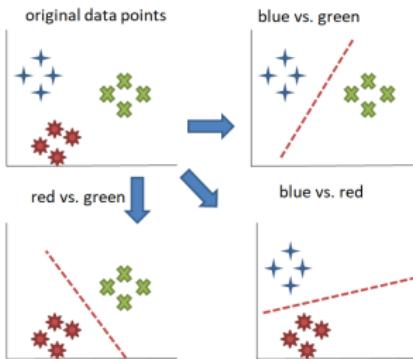
One-vs-One

ONE-VS-ONE

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

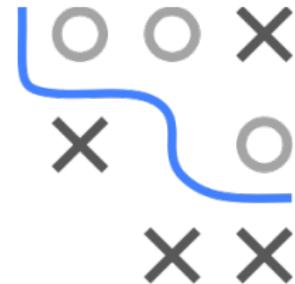


Class	$f_1(\mathbf{x})$	$f_2(\mathbf{x})$	$f_3(\mathbf{x})$
1	1	-1	0
2	-1	0	1
3	0	1	-1



ONE-VS-ONE

- Label prediction is done via **majority voting**. We predict the class of a new \mathbf{x} with all classifiers and select the class that occurs most often.
- **Pairwise coupling** (see *Hastie, T. and Tibshirani, R. (1996, Chapter 10) 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-R

- 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 require quadratic extra effort in g , but often only approximately linear effort in g .
- We experimentally investigate the train times of the one-vs-all and one-vs-one approaches for an increasing number of classes.
- 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.

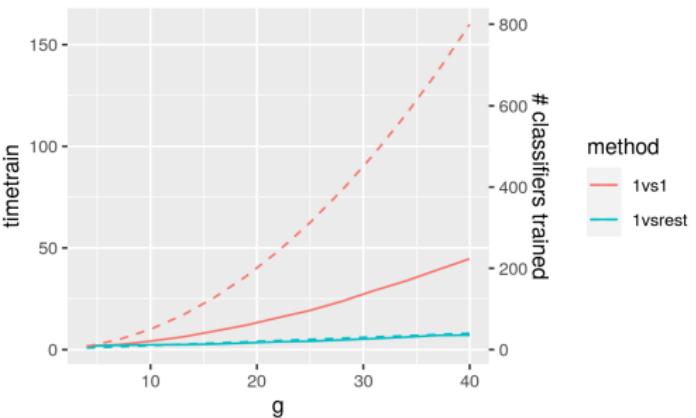


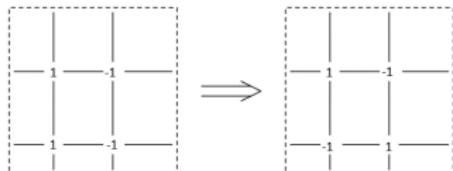
Figure: The number of classes vs. the training time (solid lines, left axis) and the number of learners (dashed lines, right axis) for each of the two approaches.

Introduction to Machine Learning

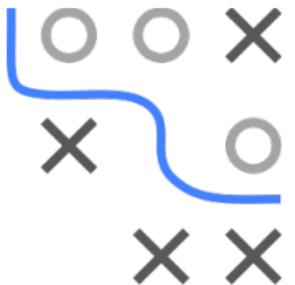
Designing Codebooks and ECOC



Learning goals



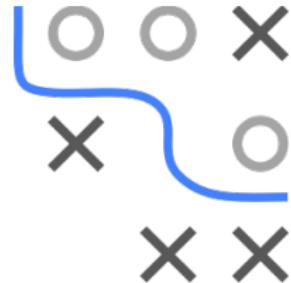
- Know what a codebook is
- Understand that codebooks generalize one-vs-one and one-vs-rest
- Know how to define a good codebook and error-correcting output codes (ECOC)
- Know how randomized hill-climbing algorithm is used to find good codebooks



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	1
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 decide the class \hat{y} for a new input \mathbf{x} ?

- When a new sample \mathbf{x} is going to be classified, all classifiers 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 $\text{sgn}(\hat{f}(\mathbf{x})) = (-1, -1, 0)$ 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} .

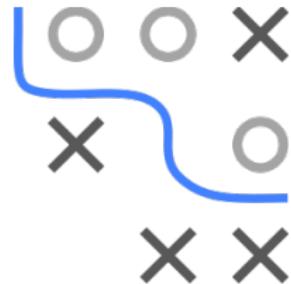


DEFINING GOOD CODEBOOKS

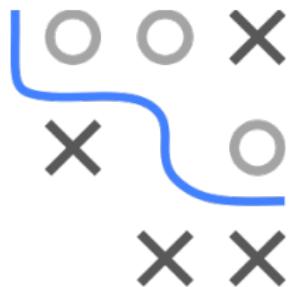
Question: How to define a good codebook?

- 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, it will be 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)

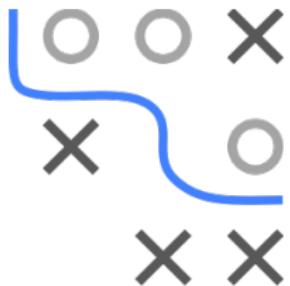
Another desirable property is **column separation**:

- 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 individual classifiers are relatively uncorrelated, so that the probability of simultaneous errors in many classifiers is small.
- Errors in classifiers f_k and f_l will also be highly correlated if errors in those columns are complementary.
- Try to ensure that columns are neither identical nor complementary.

→ We want to maximize distances between rows, and want 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 $2^k - 1$ 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 usable.

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

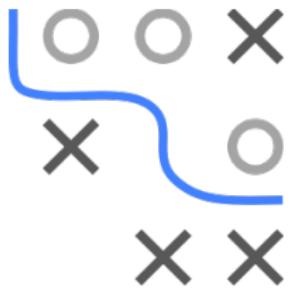
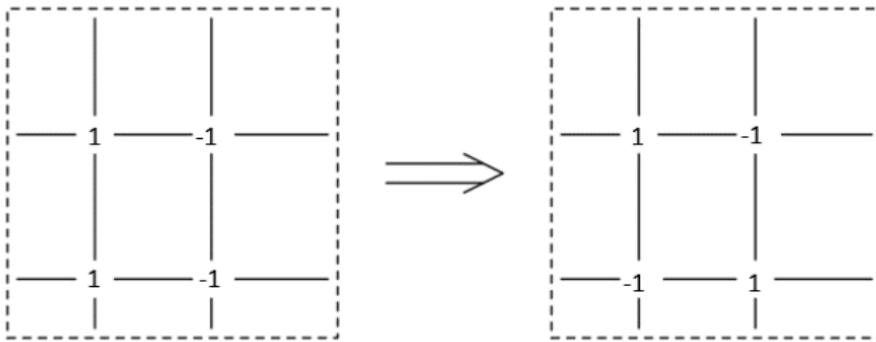
ERROR-CORRECTING CODES (ECOC)

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

- For only few classes $g \leq 11$, exhaustive search can be performed and a codebook that has good row and column separation can be 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.

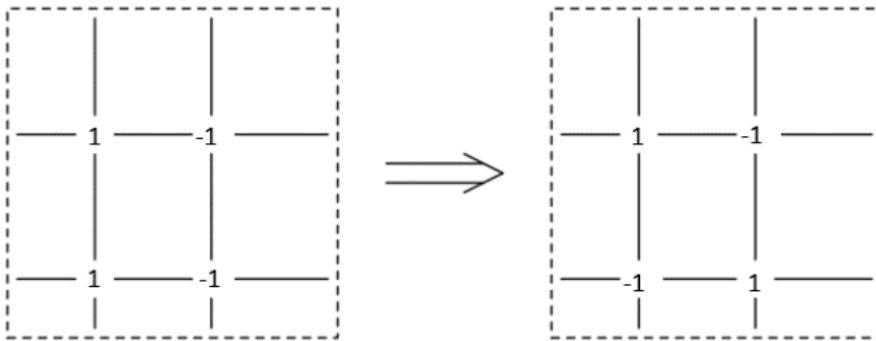


ECOC: RANDOMIZED HILL-CLIMBING ALGORITHM



- 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 rows and columns intersect and changes them to improve row and column separations
- When the procedure reaches a local maximum, the algorithm randomly chooses pairs of rows and columns and tries to increase 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

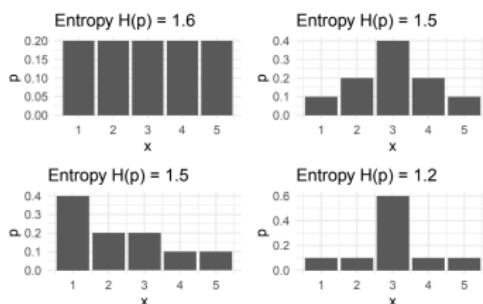
Feature Selection





Introduction to Machine Learning

Entropy



Learning goals

- Know that the entropy expresses expected information for discrete RVs
- Entropy for a single RV
- Joint entropy for two RVs
- Understand that the uniqueness theorem justifies the choice for the formula of the entropy

INFORMATION THEORY

- **Information Theory** is a field of study based on probability.
- The foundation of the field was laid by Claude Shannon in 1948, and it has since found applications in areas as diverse as communication theory, computer science, optimization, cryptography, machine learning and statistical inference.
- In addition to quantifying information, it also deals with efficient storing and transmitting the information.
- Information theory tries to quantify the "amount" of information gained or uncertainty reduced when a random variable is observed.

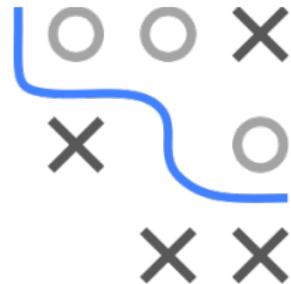


INFORMATION THEORY

- We introduce the basic concepts from a probabilistic perspective without referring too much to communication, channels or entropy.
- We will show some proofs, but not for everything. We recommend *Elements of Information Theory* by Cover and Thomas as 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.



ENTROPY



- We develop in this unit entropy as a measure of uncertainty
- Entropy is often introduced in IT as a measure of expected information or in terms of bits needed for efficient coding, but in stats and ML the first type of intuition seems most useful

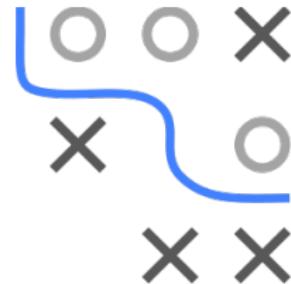
For a discrete random variable X with domain $x \in \mathcal{X}$ and pmf p

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

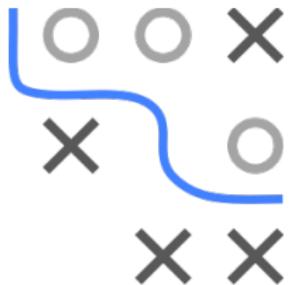
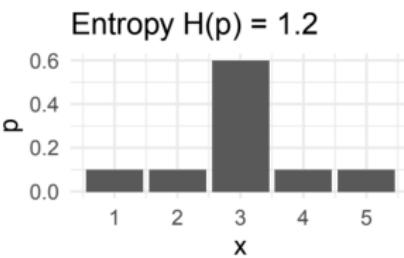
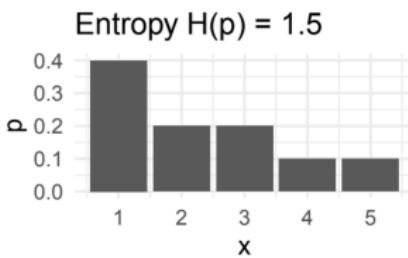
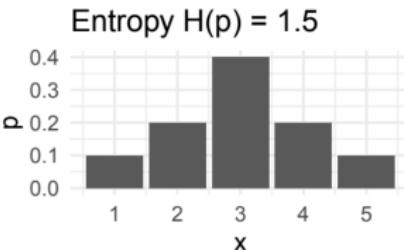
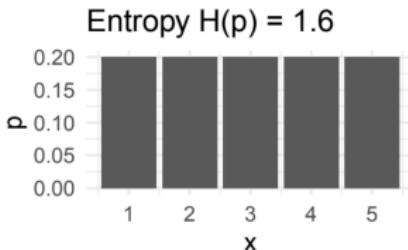
- **Definition:** Base 2 means the information is measured in bits, but you can use any number > 1 as base of the logarithm.
- **Note:** Because $\lim_{p \rightarrow 0} p \log_2 p = 0$, if $p(x) = 0$, $p(x) \log_2 p$ is taken to be zero for $x = 0$.

ENTROPY

- We have encountered various other measures of spread or uncertainty in statistics before.
- Furthermore, the formula does not seem to make intuitive sense.
- We will now do the following:
 - Approach it by various simple examples.
 - Note some basic properties - which seem desirable for an uncertainty measure.
 - Note that if we require these measures axiomatically, it drops out automatically as the result.
 - Note even more nice properties.



ENTROPY EXAMPLES

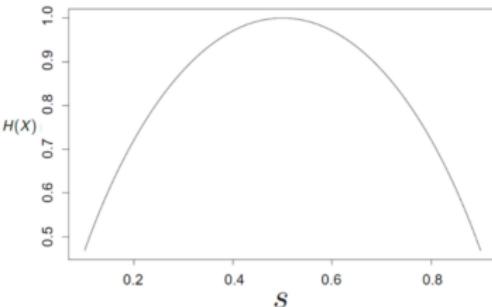
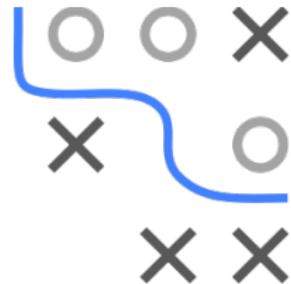


Naive observations: Uniform seems maximal, re-ordering does matter, and the more peaked, the less entropy.

ENTROPY OF BERNOUlli DISTRIBUTION

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

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

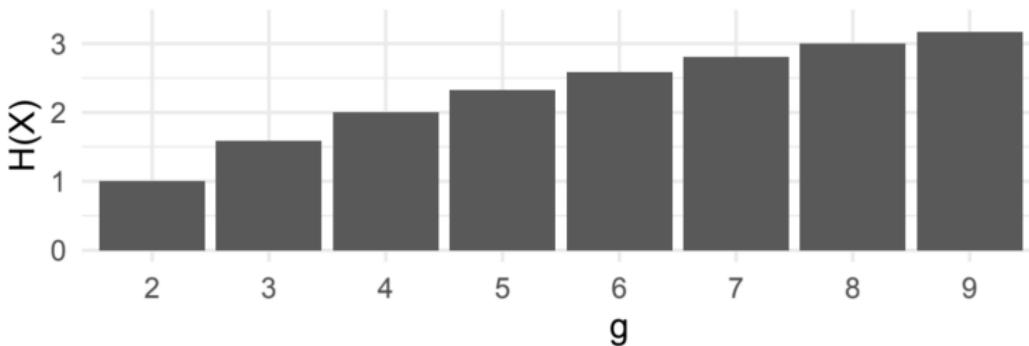


We note: If the coin is deterministic, so $s = 1$ or $s = 0$, then $H(s)$ is maximal for $s = 0.5$, a fair coin. $H(s)$ becomes monotone larger the closer we get to $s = 0.5$. This all seems plausible.

ENTROPY OF UNIFORM DISTRIBUTIONS

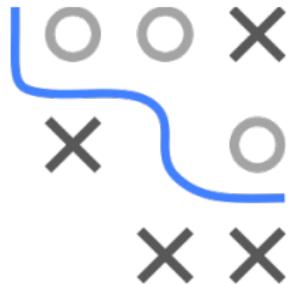
Let X be a uniform, discrete RV with g outcomes (g -sided fair d

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



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

PROPERTIES OF DISCRETE ENTROPY



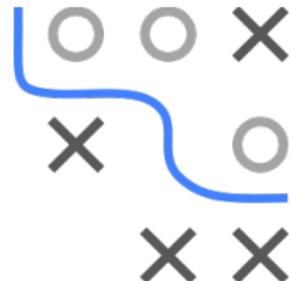
- ① Entropy is non-negative, so $H(X) \geq 0$.
- ② If one event has probability $p(x) = 1$, then $H(X) = 0$.
- ③ Symmetry. If the values $p(x)$ in the pmf are re-ordered, entropy does not change.
- ④ Adding or removing an event with $p(x) = 0$ does not change entropy.
- ⑤ $H(X)$ is continuous in probabilities $p(x)$.
- ⑥ Entropy is additive for independent RVs.
- ⑦ Entropy is maximal for a uniform distribution, so for a domain with g elements: $H(X) \leq -g \frac{1}{g} \log_2(\frac{1}{g}) = \log_2(g)$.

All properties except the last 2 follow trivially from the definition

JOINT ENTROPY

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

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



- Intuitively, the joint entropy is a measure of the total uncertainty of the two variables X and Y . In other words, it is simply the negative 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

Let X and Y be two independent RVs. Then:

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



ENTROPY OF UNIFORM DISTRIBUTIONS

Claim: The entropy of a discrete random variable X which take values in $\{x_1, x_2, \dots, x_g\}$ with associated probabilities $\{p_1, p_2, \dots\}$ is maximal when the distribution over X is uniform.



Proof: The entropy $H(X)$ is $-\sum_{i=1}^g p_i \log_2 p_i$ and our goal is to

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

subject to

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

ENTROPY OF UNIFORM DISTRIBUTIONS

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 for $\nabla L = 0$,

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

where the last step follows from the fact that all p_i are equal and the constraint.

THE UNIQUENESS THEOREM

Khinchin (1957) showed that the only family of functions satisfy

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

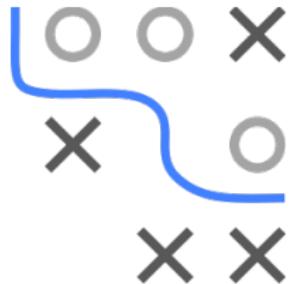
is of the following form:

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

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

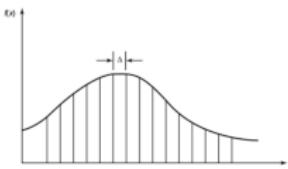


Introduction to Machine Learning



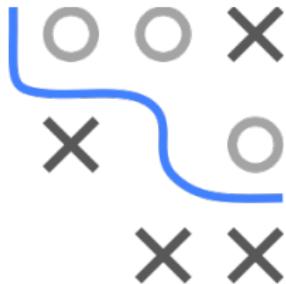
Differential Entropy

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 support \mathcal{X} , the analogue of entropy is **differential entropy**

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

- 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 some properties of discrete entropy
- $h(X) < 0$ is possible because $f(x) > 1$ is possible.

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

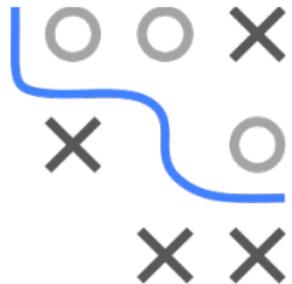
Let X be a Gaussian random variable $X \sim \mathcal{N}(\mu, \sigma^2)$ and let us measure in nats:

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



DIFF. ENTROPY OF GAUSSIAN

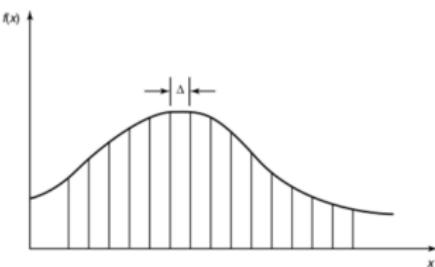
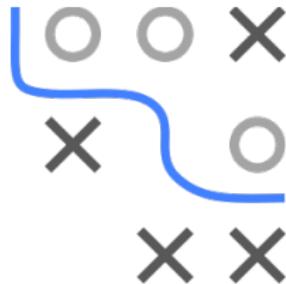
- $h(X)$ is not a function of μ (see translation invariance).
- As σ^2 increases, the differential entropy also increases.
- For $\sigma^2 < \frac{1}{2\pi e}$, it is negative.



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 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

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

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 $f(\cdot)$ on the support \mathcal{X} , differential entropy is also defined as:

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

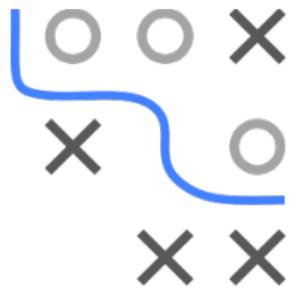
- Hence this also defines the joint differential entropy for a set of continuous RVs.

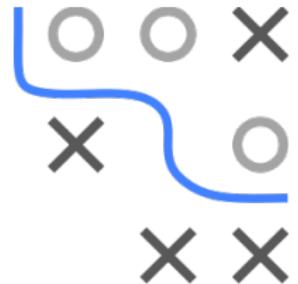
Entropy of a multivariate normal distribution: If $X \sim N(\mu, \Sigma)$ is multivariate Gaussian, then

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

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 distributions with the same (co)variance, so
$$h(X) \leq \frac{1}{2} \log(2\pi e)^n |\Sigma|.$$
- ④ Translation-invariant, $h(X + a) = h(X)$.
- ⑤ $h(aX) = h(X) + \log |a|$.
- ⑥ $h(AX) = h(X) + \log |A|$ for random vectors and matrix A.





Introduction to Machine Learning

Entropy and Optimal Code Length

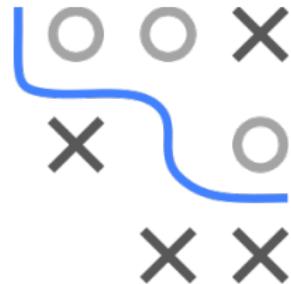
Learning goals

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

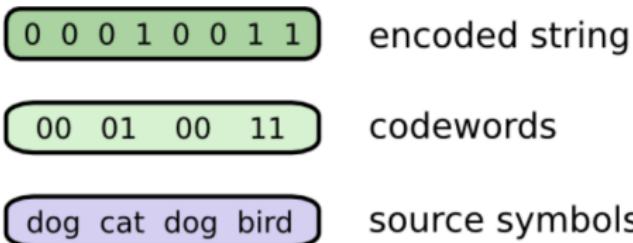
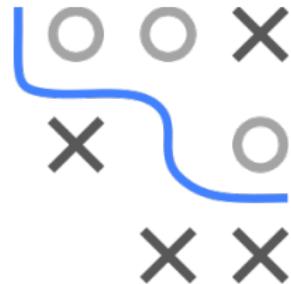
SOURCE CODING

- There is an interesting connection between entropy and a 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 stored or transmitted over a communication channel (such as fiber-optic cables).
- For example, one could use binary strings (0's and 1's) to represent messages.
- Because it may be expensive to transmit or store information, an important problem addressed by source coding is efficient schemes of minimal average length.



SOURCE CODING

- Formally, given a discrete alphabet/dictionary X of message symbols, a **binary code** is a mapping from symbols in X to 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 successive pair of digits with the associated word.

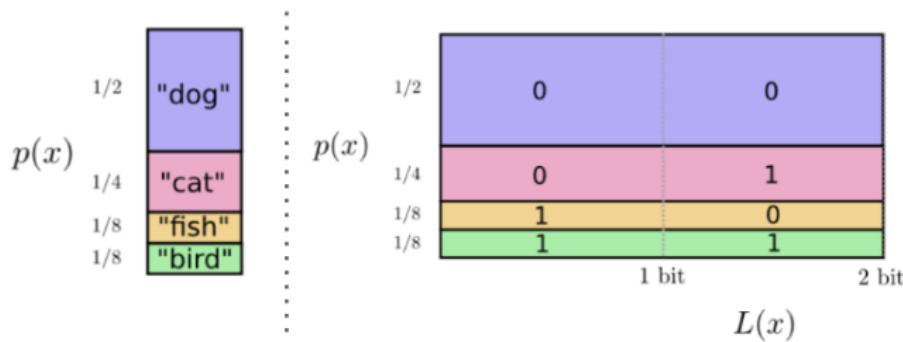
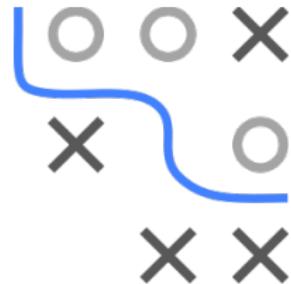


Credit: Chris Olah

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

SOURCE CODING

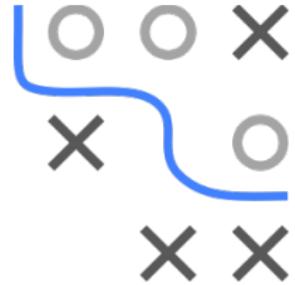
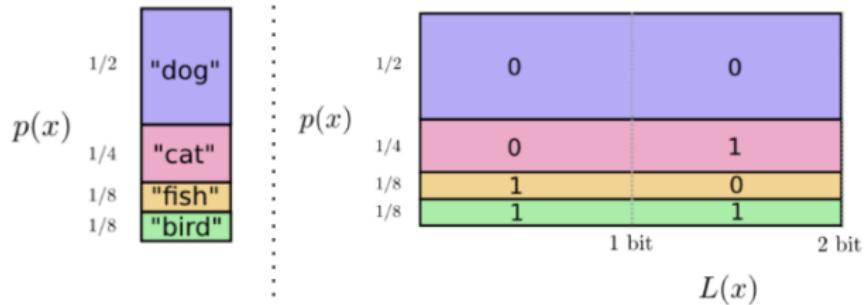
- Encoded messages are emitted by a source which can be modeled as a probability distribution over the message symbols in the dictionary.
- Let X be a random variable that represents a symbol from the source and let $p(x) = \mathbb{P}(X = x)$, for symbol x in our dictionary.



Credit: Chris Olah

- The length $L(x)$ is simply the number of bits in the corresponding codeword. In this example, all codewords have length 2.

SOURCE CODING



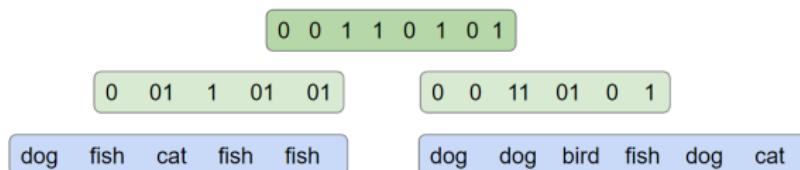
- For this code, the expected length of a message emitted by the source is, naturally:

$$\mathbb{E}[L(X)] = \frac{1}{2} \cdot 2 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 2 + \frac{1}{8} \cdot 2 = 2 \text{ bits.}$$

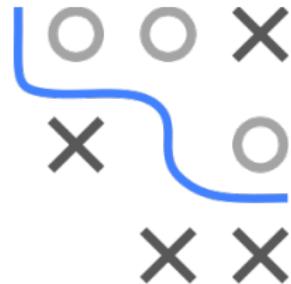
- The area of a rectangle in the image on the right reflects the value of the corresponding term in the expectation.

SOURCE CODING

- Maybe we can create better average-length coding scheme **variable-length** codes by assigning shorter codes to more 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:

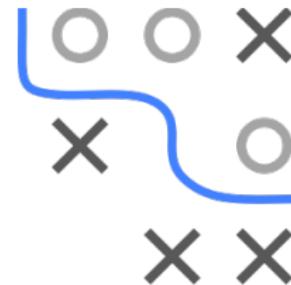


- One way to make variable-length messages unambiguous is ensuring that no codeword is a prefix (initial segment) of another codeword. Such a code is known as a **prefix code**.



SOURCE CODING

- 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
		0	1
0		1	0
		1	1
1		0	0
		1	1
		1	0
			1

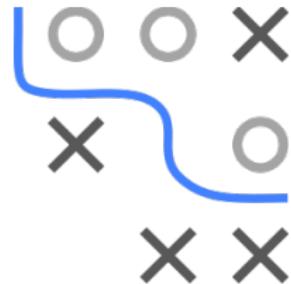
bit 1 bit 2 bit 3

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

SOURCE CODING

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

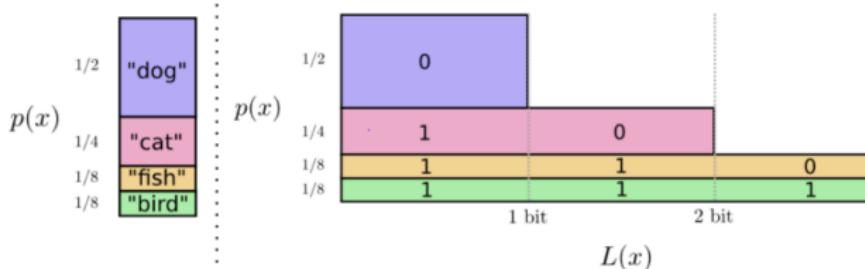


- Here, if the codeword **01** is assigned to a symbol, then **010** and **011** cannot be assigned to any other symbol because that would break the prefix property.
- If a codeword of length L is assigned to a symbol, then $\frac{1}{2^L}$ possible codewords of length $> L$ must be discarded.
- If some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

SOURCE CODING

- An example of prefix code:

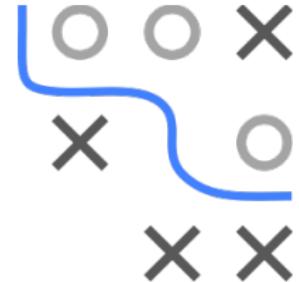
"dog" → 0, "cat" → 10, "fish" → 110 and "bird" → 111.



- Here, the expected code length is :

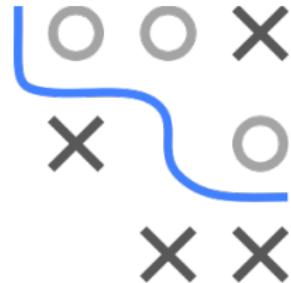
$$\begin{aligned}\mathbb{E}[L(X)] &= \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 \\ &= -\frac{1}{2} \cdot \log_2 \left(\frac{1}{2} \right) - \frac{1}{4} \cdot \log_2 \left(\frac{1}{4} \right) - \frac{1}{8} \cdot \log_2 \left(\frac{1}{8} \right) - \frac{1}{8}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

- Actually, this coding scheme is the most efficient way to store and transmit these messages. It is simply not possible to do better!



SOURCE CODING

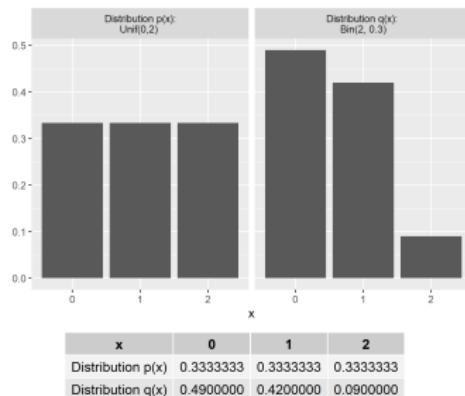
- In fact, Shannon's **source coding theorem** (or **noiseless coding theorem**) tells us that the optimal trade-off is made when the cost of a symbol with probability p is $\log(1/p)$.
- In other words, the entropy of the source distribution is the theoretical lower bound on the average code length.
- If it is any lower, some information will be distorted or lost.
- In practice, algorithms such as Huffman Coding can be used to generate variable-length codes that are close (in terms of expected length) to the theoretical limit.





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 continuous) distributions with the same support:

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

or:

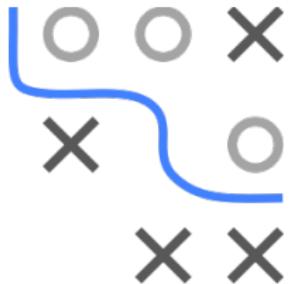
$$D_{KL}(p\|q) = \mathbb{E}_p \left[\log \frac{p(X)}{q(X)} \right] = \int_{x \in \mathcal{X}} p(x) \cdot \log \frac{p(x)}{q(x)}.$$

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



KULLBACK-LEIBLER DIVERGENCE

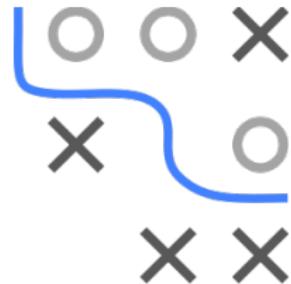
$$D_{KL}(p\|q) = \mathbb{E}_p \left[\log \frac{p(X)}{q(X)} \right]$$



- 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 distribution and q as something that we want to measure the quality of in reference to p .

INFORMATION INEQUALITY

$D_{KL}(p\|q) \geq 0$ holds always true for any pair of distributions, and 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 happen if $q(x)/p(x)$ is constant everywhere. That implies $p =$

KL AS LOG-DIFFERENCE

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



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

$$D_{KL}(p\|q) = \mathbb{E}_p(\log(p(x)) - \log(q(x))).$$

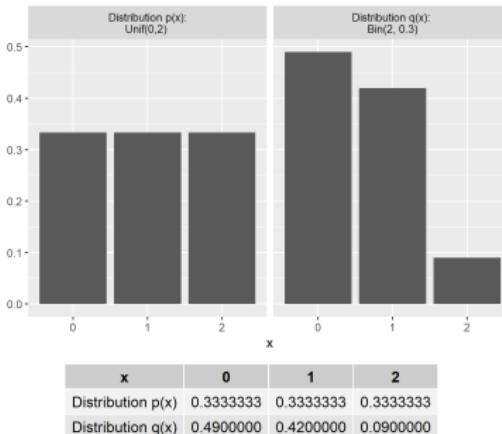
This is why we integrate out with respect to the data distribution – “good” approximation $q(x)$ should minimize the difference to $p(x)$.

KL AS LOG-DIFFERENCE

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



Example: Let $q(x)$ be a binomial distribution with $N = 2$ and $p = 0.3$, and let $p(x)$ be a discrete uniform distribution. Both distributions have the same support $\mathcal{X} = \{0, 1, 2\}$.



KL AS LOG-DIFFERENCE



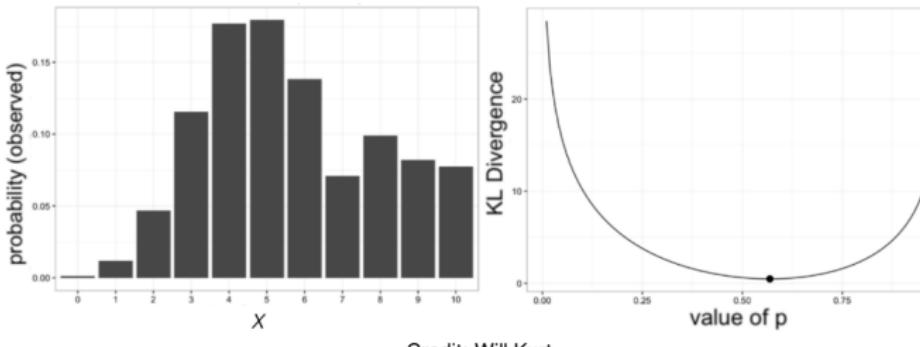
$$\begin{aligned} D_{KL}(p\|q) &= \sum_{x \in \mathcal{X}} p(x) \log \left(\frac{p(x)}{q(x)} \right) \\ &= 0.333 \log \left(\frac{0.333}{0.49} \right) + 0.333 \log \left(\frac{0.333}{0.42} \right) + 0.333 \log \left(\frac{0.333}{0.25} \right) \\ &= 0.23099 \text{ (nats)} \end{aligned}$$

$$\begin{aligned} D_{KL}(q\|p) &= \sum_{x \in \mathcal{X}} q(x) \log \left(\frac{q(x)}{p(x)} \right) \\ &= 0.49 \log \left(\frac{0.49}{0.333} \right) + 0.42 \log \left(\frac{0.42}{0.333} \right) + 0.09 \log \left(\frac{0.09}{0.25} \right) \\ &= 0.16801 \text{ (nats)} \end{aligned}$$

Again, note that $D_{KL}(p\|q) \neq D_{KL}(q\|p)$.

KL IN FITTING

Because KL quantifies the difference between distributions, it can be used as a loss function to find a good fit for the observed data.



Credit: Will Kurt

Figure: Left: Histogram of observed frequencies of a random variable X which takes integer values between 0 and 10. Right: The KL divergence between the observed distribution and $\text{Binom}(10,p)$ is minimized when $p \approx 0.57$.

Will Kurt (2017): Kullback-Leibler Divergence Explained.

<https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained>



KL IN FITTING

Because KL quantifies the difference between distributions, it can be used as a loss function to find a good fit for the observed data.

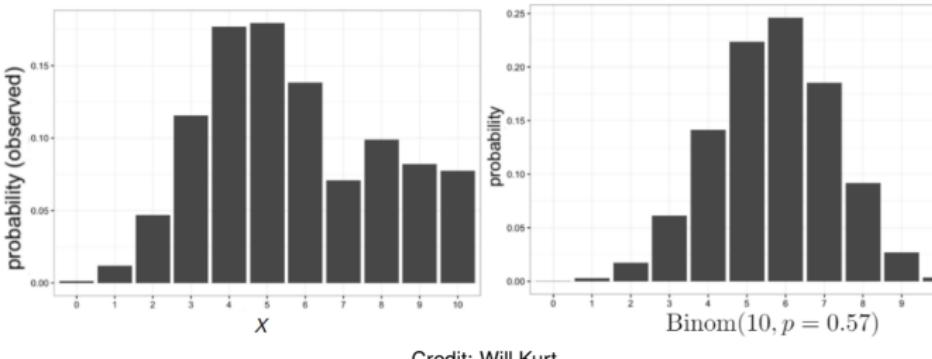
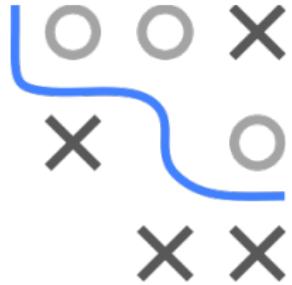


Figure: Left: Histogram of observed frequencies of a random variable which takes values between 0 and 10. Right: Fitted Binomial distribution ($p \approx 0.57$).

On the right is the Binomial distribution that minimizes the KL divergence.

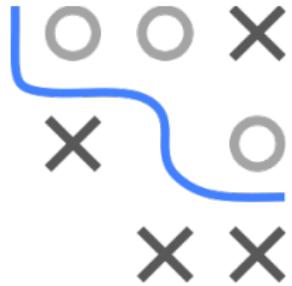


KL AS LIKELIHOOD RATIO

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

$$LR = \frac{p(x)}{q(x)}$$

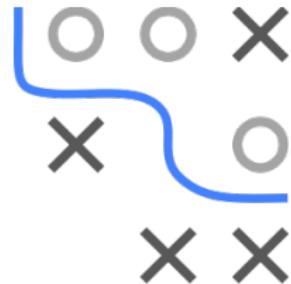
In the above, if for x we have $LR > 1$, then p seems better, for $LR < 1$, q seems better.



KL AS LIKELIHOOD RATIO

Or we can compute LR for a complete set of data (as always, to our life easier):

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



Now let us assume that our data already come from p . It does not make sense anymore to ask whether p or q fit the data better. But maybe we want to pose the question "How different is q from p ?" by formulating it as: "If we sample many data from p , how easily can we see that p is better than q through LR, on average?"

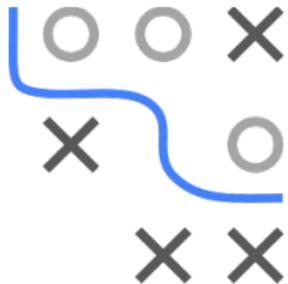
$$\mathbb{E}_p \left[\log \frac{p(x)}{q(x)} \right]$$

That expected LR is really KL!

KL AS LIKELIHOOD RATIO

In summary we could say for KL:

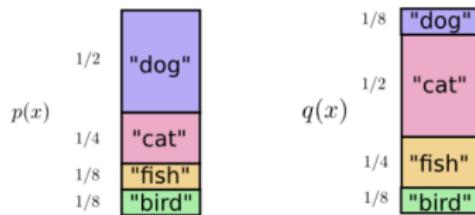
- It measures how much "evidence" each sample provides on average to distinguish p from q , if you sample from p .
- If p and q are very similar, most samples will not help much vice versa for very different distributions.
- In practice, we often want to make the approximation q as indistinguishable from the real p (our data) as possible. We already did that when we fitted (in our log-difference persp





Introduction to Machine Learning

Cross-Entropy, KL and Source Coding

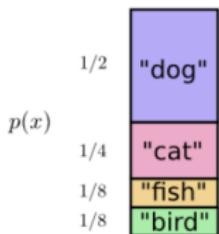


Learning goals

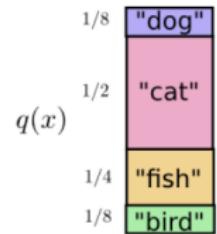
- Know the cross-entropy
- Understand the connection between entropy, cross-entropy and KL divergence

CROSS-ENTROPY - DISCRETE CASE

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



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



$$H_q(p) = 2.375 \text{ bits}$$

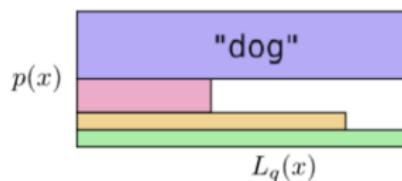
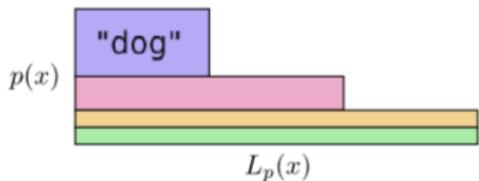


Figure: $L_p(x)$, $L_q(x)$ are the optimal code lengths for $p(x)$ and $q(x)$

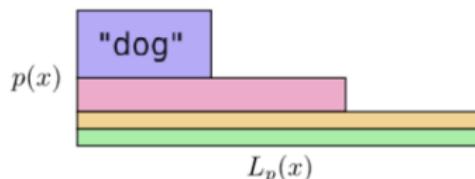
CROSS-ENTROPY - DISCRETE CASE

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

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



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



$$H_q(p) = 2.375 \text{ bits}$$

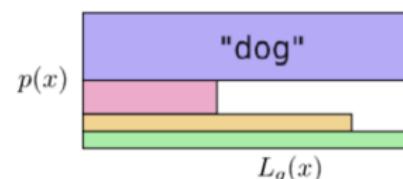
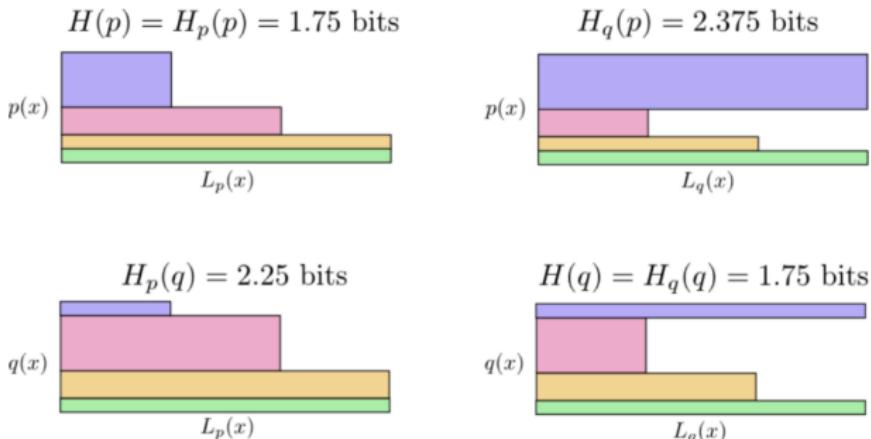


Figure: $L_p(x)$, $L_q(x)$ are the optimal code lengths for $p(x)$ and $q(x)$

We directly see: cross-entropy of p with itself is entropy: $H_p(p)$

CROSS-ENTROPY - DISCRETE CASE



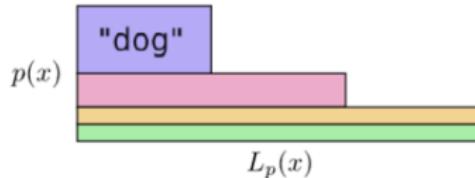
Credit: Chris Olah



- In top, $H_q(p)$ is greater than $H(p)$ primarily because the blue event 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 $H_q(p) \neq H_p(q)$.

CROSS-ENTROPY - DISCRETE CASE

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



$$H_q(p) = 2.375 \text{ bits}$$

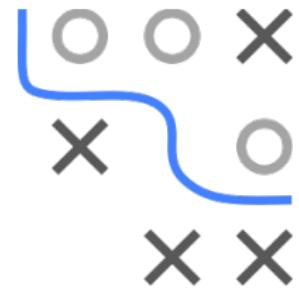
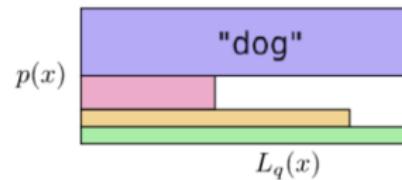


Figure: $L_p(x), L_q(x)$ are the optimal code lengths for $p(x)$ and $q(x)$

- Let x' denote the symbol "dog". The difference in code len

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

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

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

CROSS-ENTROPY - DISCRETE CASE

- Entropy = Avg. nr. of bits if we optimally encode p
- Cross-Entropy = Avg. nr. of bits if we suboptimally encode
- $D_{KL}(p\|q)$: Difference in bits between the two



We can summarize this also through this identity:

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

This is because:

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

CROSS-ENTROPY - CONTINUOUS CASE

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

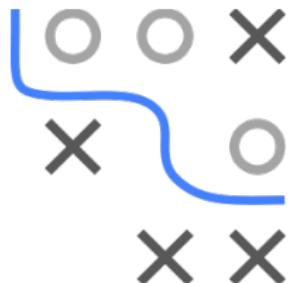
$$H_p(q) = \int q(x) \log \left(\frac{1}{p(x)} \right) dx = - \int q(x) \log(p(x))$$



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

PROOF: MAXIMUM OF DIFFERENTIAL ENTR

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



Proof: Let $g(x)$ be a Gaussian with mean μ and variance σ^2 and let $f(x)$ be 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

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

PROOF: MAXIMUM OF DIFFERENTIAL ENTR

The second term in (1) is,

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



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

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

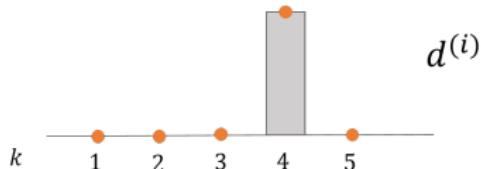
with equality when $f(x) = g(x)$ (following from the properties o
Kullback-Leibler divergaence).

Introduction to Machine Learning



Information Theory for Machine Learr

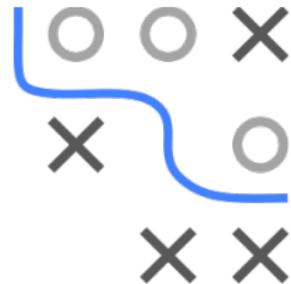
Learning goals



- Minimizing KL is equivalent to maximizing the log-likelihood
- Minimizing KL is equivalent to minimizing cross-entropy
- Minimizing cross-entropy between modeled and observed probabilities is equivalent to log-loss minimization

KL VS MAXIMUM LIKELIHOOD

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



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

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

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

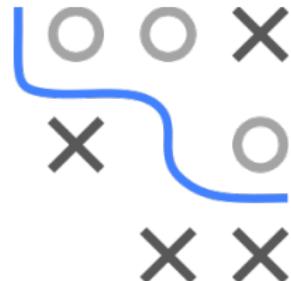
For a finite dataset of n samples from p , this is approximated as

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

KL VS CROSS-ENTROPY

From this here we can actually see much more:

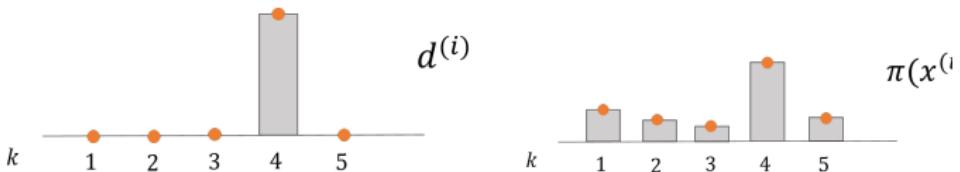
$$\arg \min_{\theta} D_{KL}(p \| q_{\theta}) = \arg \min_{\theta} -\mathbb{E}_{x \sim p} \log q(x | \theta) = \arg \min_{\theta}$$



- So minimizing with respect to KL is the same as minimizing with respect to cross-entropy!
- That implies minimizing with respect to cross-entropy is the same as maximum likelihood!
- Remember, how we only characterized cross-entropy through source coding / bits? We could now motivate cross-entropy as a "relevant" term that you have to minimize, when you minimize after you drop $\mathbb{E}_p \log p(x)$, which is simply the neg. entropy
- Or we could say: Cross-entropy between p and q is simply the expected negative log-likelihood of q , when our data come from p .

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 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 categorical distribution over the classes.



CROSS-ENTROPY VS. LOG-LOSS

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

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



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

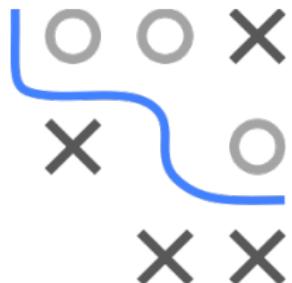
$$\begin{aligned} R &= \sum_{i=1}^n H_{\pi_k(\mathbf{x}^{(i)}|\theta)}(d^{(i)}) \\ &= \sum_{i=1}^n \left(- \sum_k d_k^{(i)} \log \pi_k(\mathbf{x}^{(i)}|\theta) \right) \\ &= \sum_{i=1}^n \underbrace{\left(- \sum_{k=1}^g [y^{(i)} = k] \log \pi_k(\mathbf{x}^{(i)}|\theta) \right)}_{\text{log loss}} \\ &=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

CROSS-ENTROPY VS. BERNOULLI LOSS

For completeness sake:

Let us use the Bernoulli loss for binary classification:

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

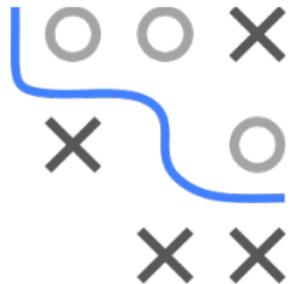


If p represents a $\text{Ber}(y)$ distribution (so deterministic, where the label receives probability mass 1) and we also interpret $\pi(\mathbf{x})$ as Bernoulli distribution $\text{Ber}(\pi(\mathbf{x}))$, the Bernoulli loss $L(y, \pi(\mathbf{x}))$ is cross-entropy $H_{\pi(\mathbf{x})}(p)$.

ENTROPY AS PREDICTION LOSS

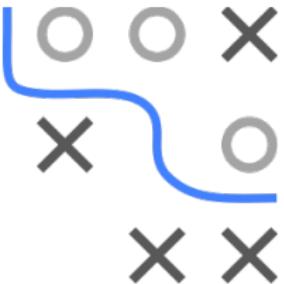
Assume log-loss for a situation where you only model with a categorical probability vector π . We know the optimal model under that loss:

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



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

ENTROPY AS PREDICTION LOSS



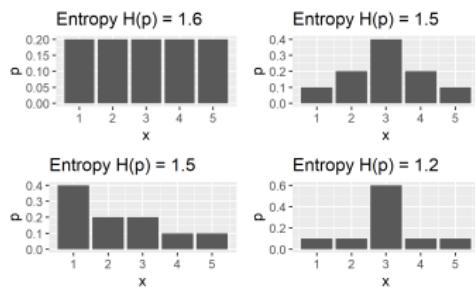
$$\begin{aligned}\mathcal{R} &= \frac{1}{n} \sum_{i=1}^n \left(- \sum_{k=1}^g [y^{(i)} = k] \log \pi_k \right) \\ &= - \frac{1}{n} \sum_{k=1}^g \sum_{i=1}^n [y^{(i)} = k] \log \pi_k \\ &= - \sum_{k=1}^g \frac{n_k}{n} \log \pi_k \\ &= - \sum_{k=1}^g \pi_k \log \pi_k = H(\pi)\end{aligned}$$

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



Learning goals

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

JOINT ENTROPY

- The **joint entropy** of two discrete random variables X and Y with a joint distribution $p(x, y)$ is:

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

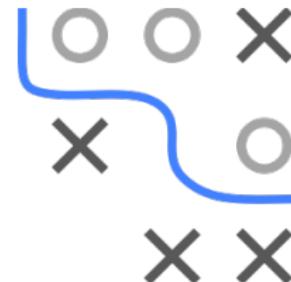
which can also be expressed as

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

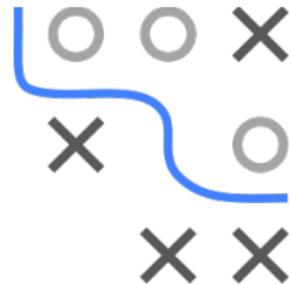
- For continuous random variables X and Y with joint density $p(x, y)$, the differential joint entropy is:

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

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



CONDITIONAL ENTROPY



- The **conditional entropy** $H(Y|X)$ quantifies the uncertainty 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 X .
- If $(X, Y) \sim p(x, y)$, the conditional entropy $H(Y|X)$ is defi

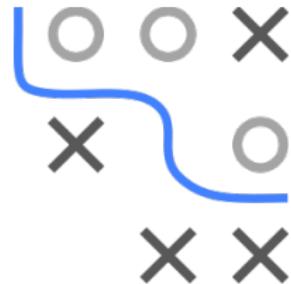
$$\begin{aligned} H(Y|X) &= \mathbb{E}_X[H(Y|X = x)] = \sum_{x \in \mathcal{X}} p(x)H(Y|X = x) \\ &= -\sum_{x \in \mathcal{X}} p(x) \sum_{y \in \mathcal{Y}} p(y|x) \log p(y|x) \\ &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(y|x) \\ &= -\mathbb{E}[\log p(Y|X)]. \end{aligned}$$

- 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, in fact, 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:

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

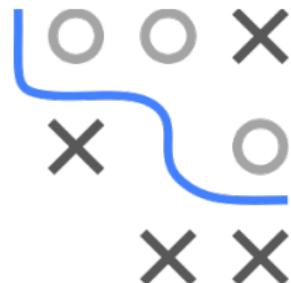
JOINT AND CONDITIONAL ENTROPY

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 consideratio

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

MUTUAL INFORMATION



- The MI describes the amount of information about one random variable obtained through the other one or how different the distribution is from pure independence.
- Consider two random variables X and Y with a joint probability mass function $p(x, y)$ and marginal probability mass functions $p(x)$ and $p(y)$. The MI $I(X; Y)$ is the Kullback-Leibler distance 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

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

Thus, mutual information $I(X; Y)$ is the reduction in the uncertainty due to the knowledge of Y .



MUTUAL INFORMATION

The following relations hold:

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

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

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

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

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

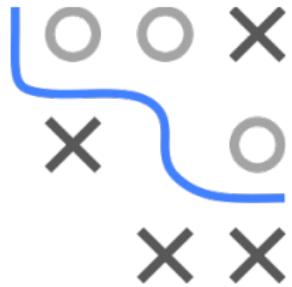


All of the above are trivial to prove.

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



The marginal distribution of X is $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$ and the 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 bits.

MUTUAL INFORMATION - EXAMPLE

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

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

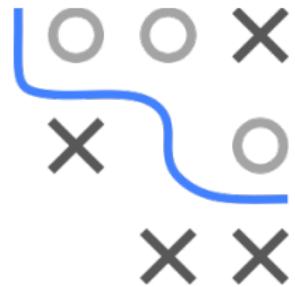
Similarly, $H(Y|X) = \frac{13}{8}$ bits and $H(X, Y) = \frac{27}{8}$ bits.



MUTUAL INFORMATION - COROLLARIES

Non-negativity of mutual information: For any two random variables $I(X; Y) \geq 0$, 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 reduce the uncertainty in X . Note that this is true only on the average

MUTUAL INFORMATION - COROLLARIES

Independence bound on entropy: Let X_1, X_2, \dots, X_n be drawn according to $p(x_1, x_2, \dots, x_n)$. Then

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

with equality if and only if the X_i are independent.

Proof: With the chain rule for entropies,

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

where the inequality follows directly from above. We have equality if and only if X_i is independent of X_{i-1}, \dots, X_1 for all i (i.e., if and only if the X_i 's are independent).



MUTUAL INFORMATION PROPERTIES

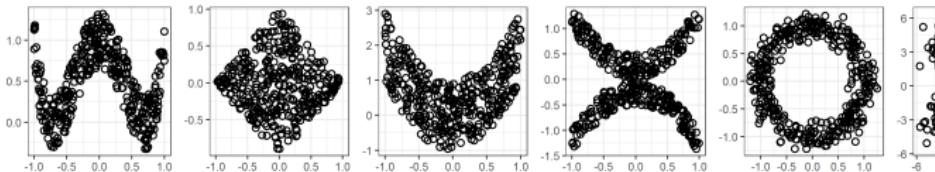
- MI is a measure of the amount of "dependence" between variables. It is zero if and only if the variables are independent.
- On the other hand, if one of the variables is a deterministic function of the other, the mutual information is maximal, i.e. entropy of the first.
- Unlike (Pearson) correlation, mutual information is not limited to real-valued random variables.
- Mutual information can be used to perform **feature selection**. Quite simply, each variable X_i is rated according to $I(X_i; Y)$, sometimes called information gain.
- The same principle can also be used in decision trees to select a feature to split on. Splitting on MI/IG is then equivalent to reduction with log-loss.



MUTUAL INFORMATION VS. CORRELATION



- If two variables are independent, their correlation is 0.
- However, the reverse is not necessarily true. It is possible for dependent variables to have 0 correlation because correlation measures linear dependence.



- The figure above shows various scatterplots where, in each case, the correlation is 0 even though the two variables are strongly dependent, and MI is large.
- Mutual information can therefore be seen as a more general measure of dependence between variables 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

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

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

Boosting

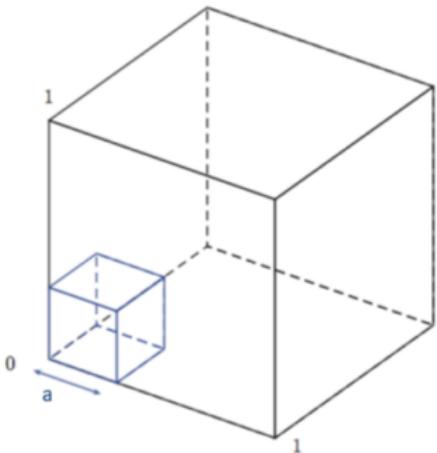
Feature Selection





Introduction to Machine Learning

Curse of Dimensionality

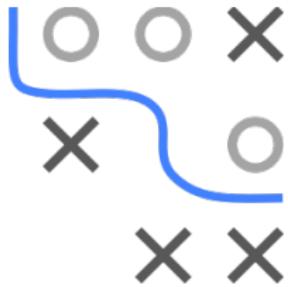


Learning goals

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

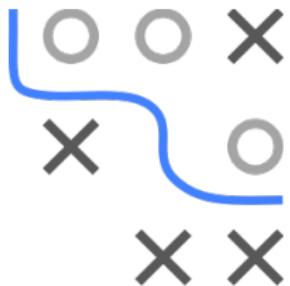
CURSE OF DIMENSIONALITY

- 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 occur in low-dimensional spaces.
- Our intuition about the geometry of a space is formed in two or 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:



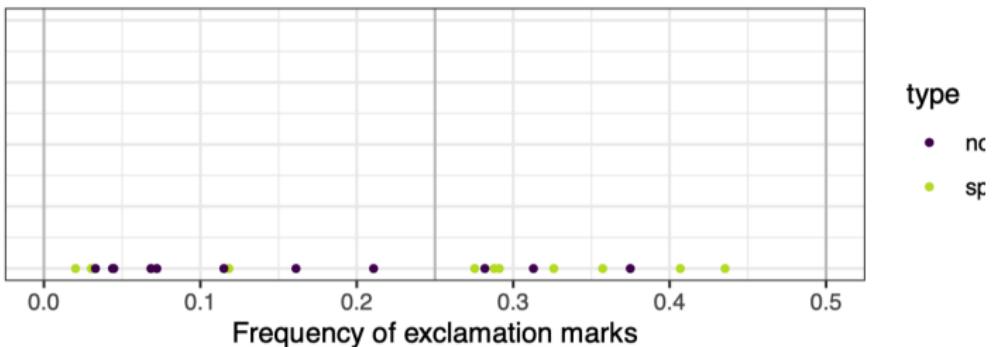
We are given 20 emails, 10 of them are spam and 10 are not. Our goal is to predict if a new incoming mail is spam or not.

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

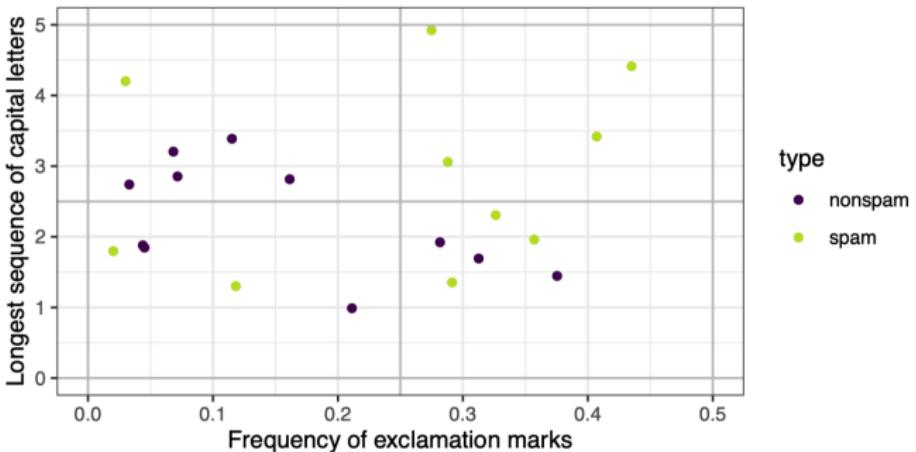
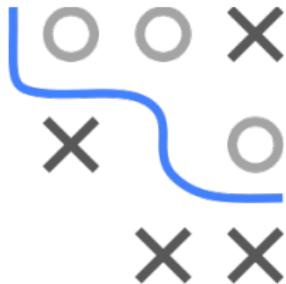


Based on the frequency of exclamation marks, we train a very simple classifier (a decision stump with split point $x = 0.25$):

- 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

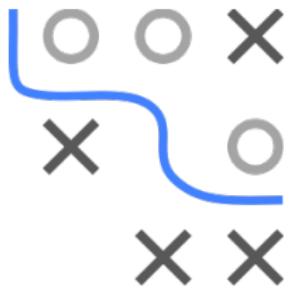
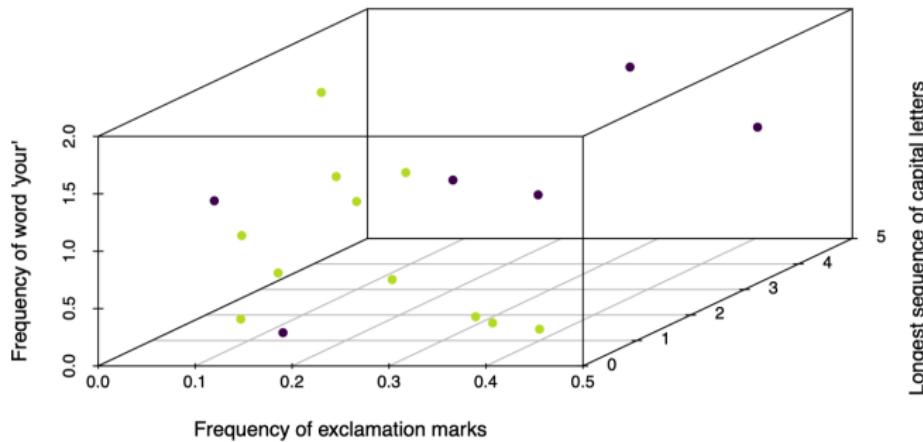
Let us feed more information into our classifier. We include a feature that contains the length of the longest sequence of capital letters.



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

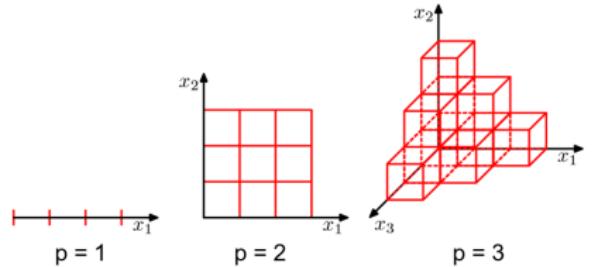
CURSE OF DIMENSIONALITY: EXAMPLE

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
- As dimensionality increases, the data become **sparse**; so 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.



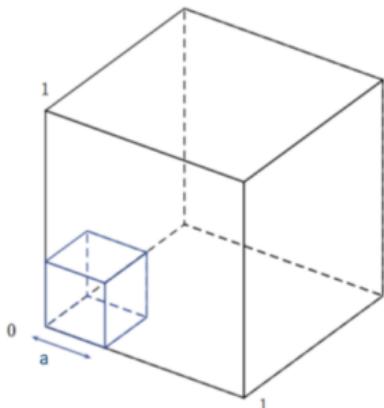
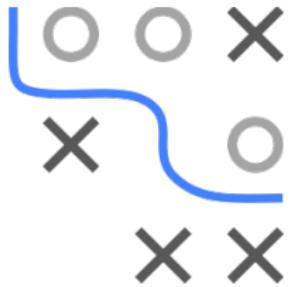
Bishop, Pattern Recognition and Machine Learning, 2006



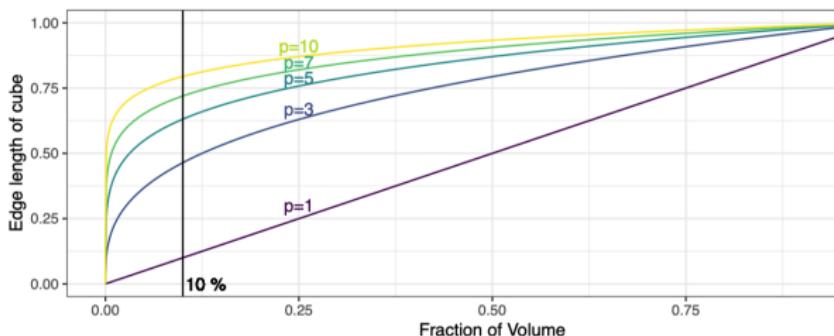
Geometry of High-Dimensional Space

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



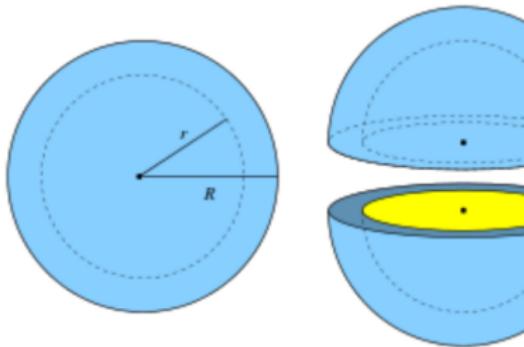
$$a^p = \frac{1}{10} \Leftrightarrow a = \frac{1}{\sqrt[p]{10}}$$

- 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 the ϵ -“edge”, $\epsilon := R - r$, of this hypersphere can be calculated using the formula

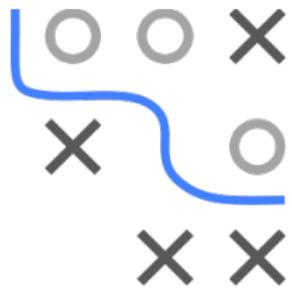
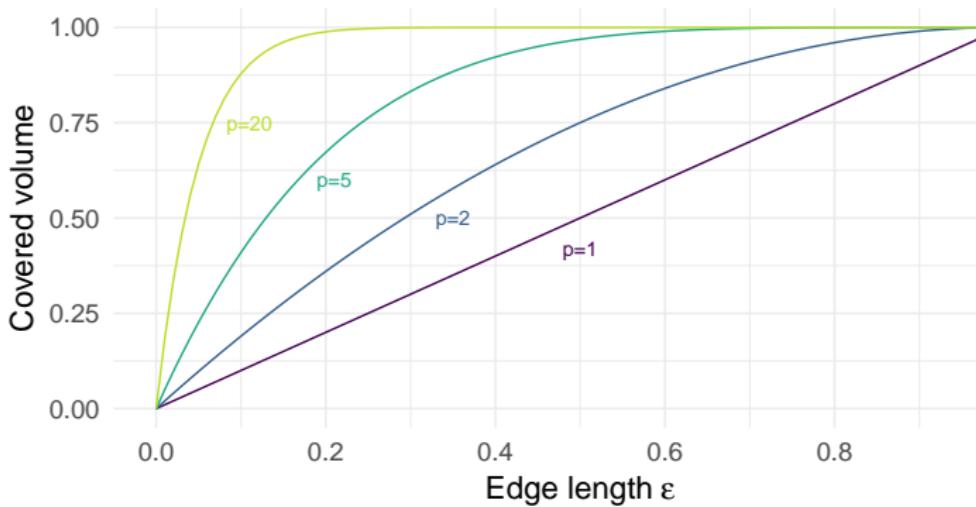
$$1 - \left(1 - \frac{\epsilon}{R}\right)^P.$$



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 the outer shell of thickness 0.2:

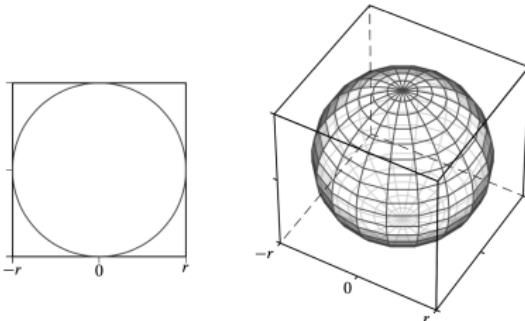
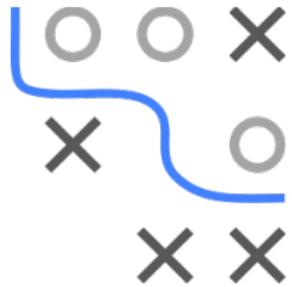


HYPERSPHERE WITHIN HYPERCUBE

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

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

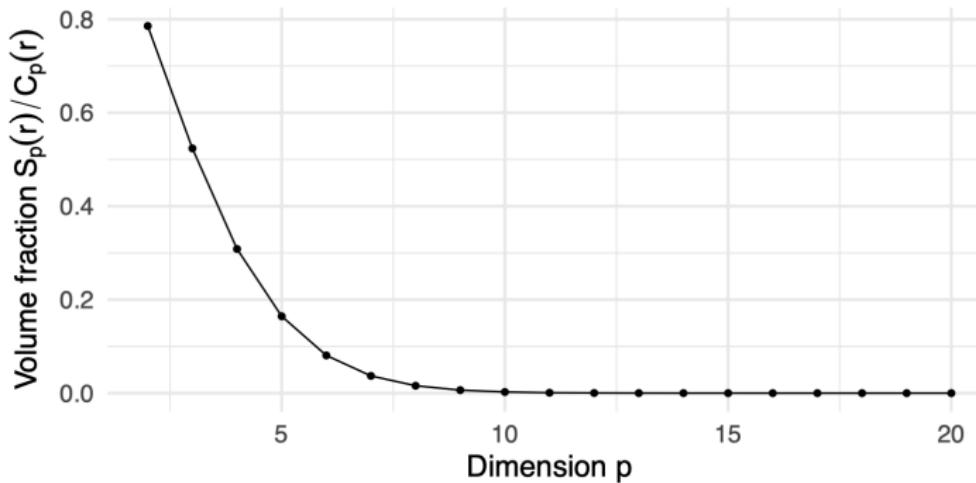
i.e., as the dimensionality increases, most of the volume of the hypercube can be found in its corners.



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:

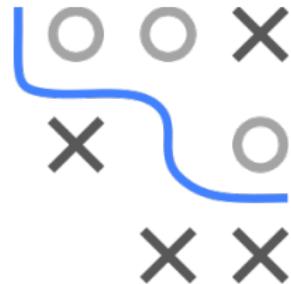


Note: For $r > 0$, the volume fraction $\frac{S_p(r)}{C_p(r)}$ is independent of r .

UNIFORMLY DISTRIBUTED DATA

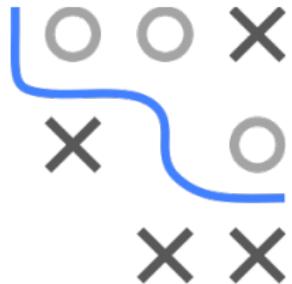
The consequences of the previous results for uniformly distributed data in the high-dimensional hypercube are:

- Most of the data points will lie on the boundary of the space.
- The points will be mainly scattered on the large number of faces of the hypercube, which themselves will become very long.
- 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 Euclidean distance from the origin), s.t.

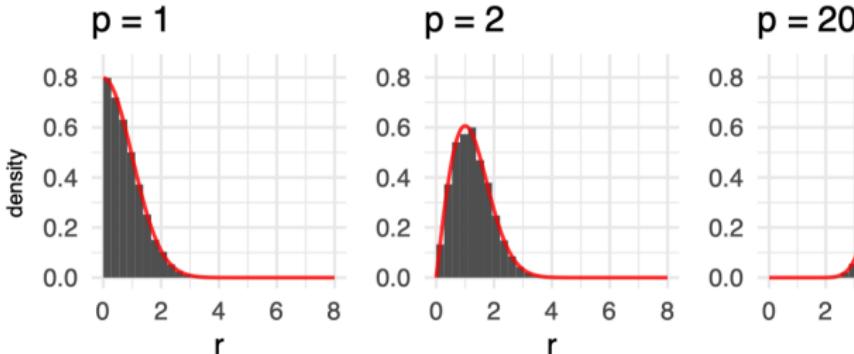
$$p(r) = \frac{S_p(r)r^{p-1}}{(2\pi\sigma^2)^{p/2}} \exp\left(-\frac{r^2}{2\sigma^2}\right),$$

where $S_p(r)$ is the surface of the p -dimensional hypersphere of radius r .

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

GAUSSIANS IN HIGH DIMENSIONS

- To verify this functional relationship empirically, we draw 100 points from the p -dimensional standard normal distribution and plot them over the histogram of the points' distances to the origin:



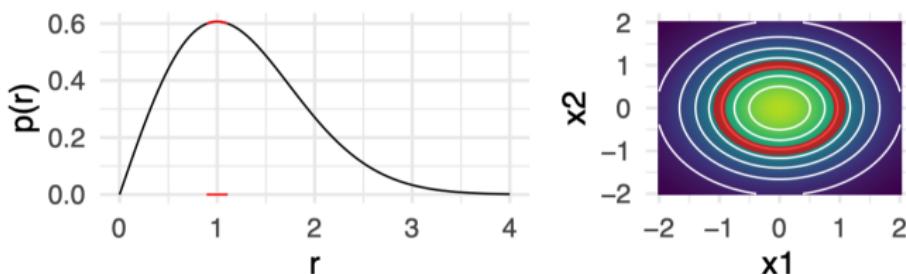
- We can see that for large p the probability mass of the Gaussian is concentrated in a fairly thin “shell” rather far away from the origin. This may seem counterintuitive, but:

GAUSSIANS IN HIGH DIMENSIONS

- For the probability mass of a hyperspherical shell it follows

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

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

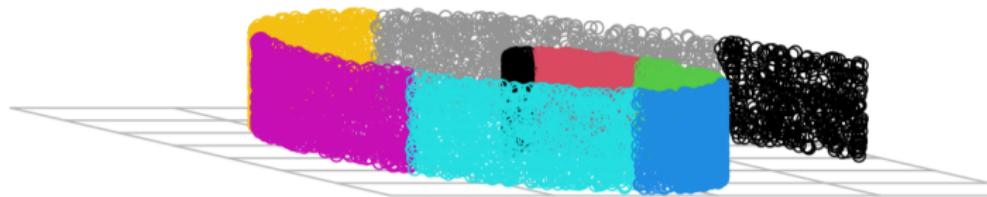
- While f_p becomes smaller with increasing r , the region of the integral -the hyperspherical shell- becomes bigger.

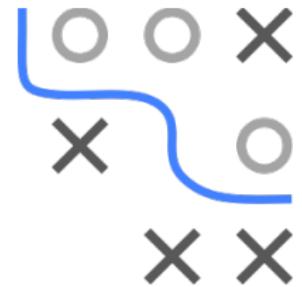


INTERMEDIATE REMARKS

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

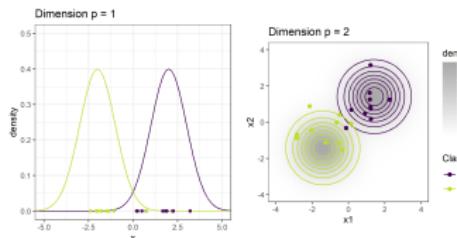
- Often the data is restricted to a manifold of a lower dimension
(Or at least the directions in the feature space over which significant changes in the target variables occur may be constrained)
- At least locally small changes in the input variables usually result in small changes in the target variables.





Introduction to Machine Learning

Curse of Dimensionality - Examples Learning Algorithms



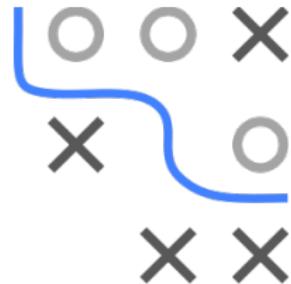
Learning goals

- See how the performance of k-NN and the linear model deteriorates in high-dimension 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 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** 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 $[0, 1]^p$:

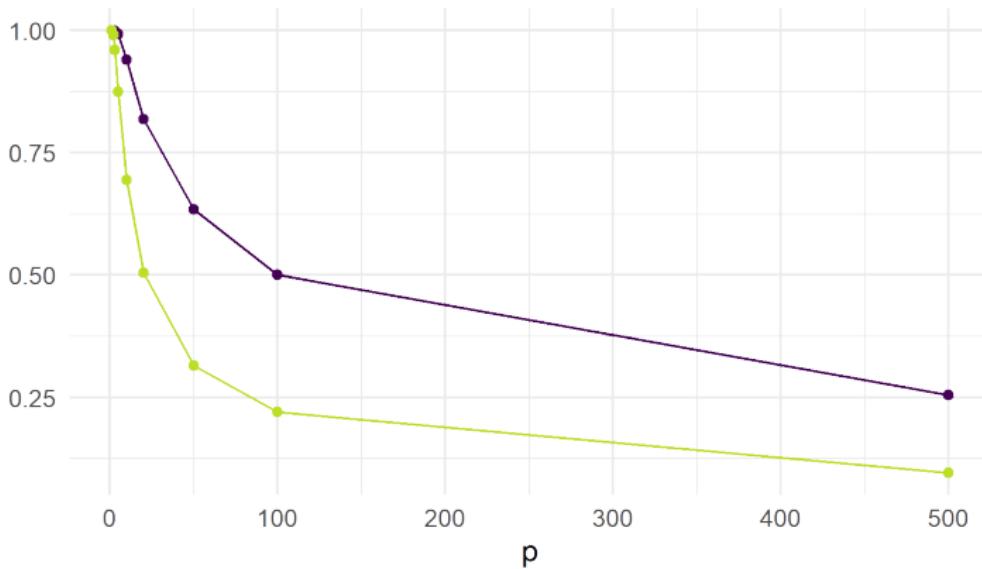
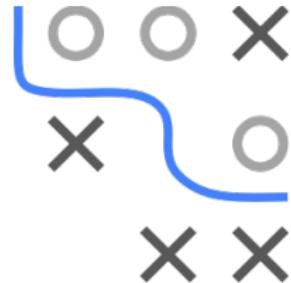
p	$\min d(\mathbf{x}, \tilde{\mathbf{x}})$	$\overline{d(\mathbf{x}, \tilde{\mathbf{x}})}$	$\max d(\mathbf{x}, \tilde{\mathbf{x}})$	$\overline{d_{NN1}(\mathbf{x})}$	$\max d_{NN1}(\mathbf{x})$
1	1.2e-08	0.33	1	5e-05	0.00042
2	0.00011	0.52	1.4	0.0051	0.02
3	0.0021	0.66	1.7	0.026	0.073
5	0.016	0.88	2	0.11	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 $c := \frac{\max(d(\mathbf{x}, \tilde{\mathbf{x}})) - \min(d(\mathbf{x}, \tilde{\mathbf{x}}))}{\max(d(\mathbf{x}, \tilde{\mathbf{x}}))}$

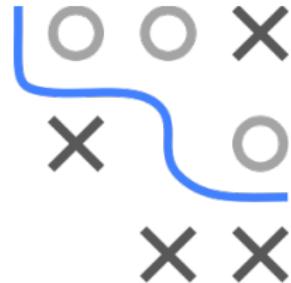
"locality"² $l := \frac{d(\mathbf{x}, \tilde{\mathbf{x}}) - d_{NN1}(\mathbf{x})}{d(\mathbf{x}, \tilde{\mathbf{x}})}$ with increasing number of dimensions



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 away 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 with fewer observations we average, the higher the variance of the estimate.
 - k-NN estimates get more inaccurate with increasing dimensionality of the data.



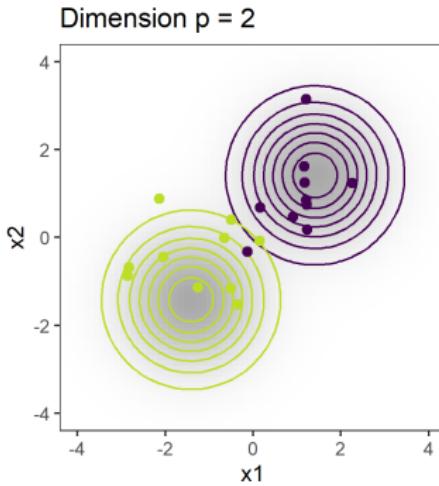
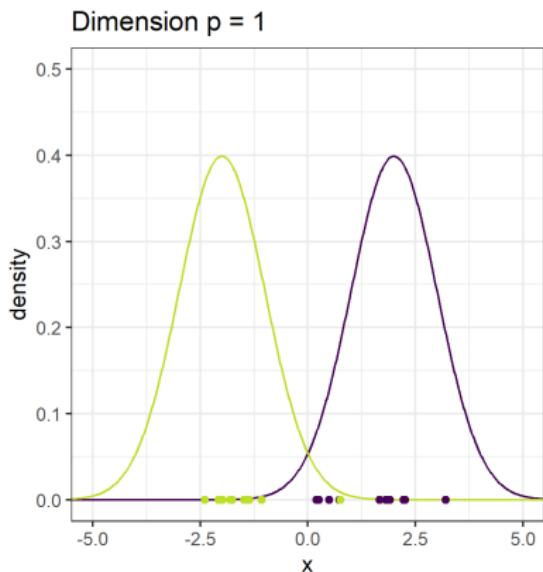
EXAMPLE: K-NN

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 selecting a point 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 selecting a point from a Gaussian with mean $-\mu = (-a, -a, \dots, -a)$ and unit covariance matrix

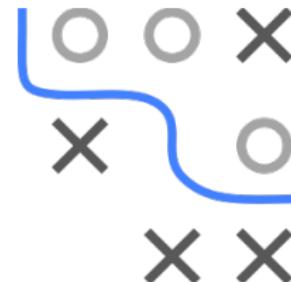


EXAMPLE: K-NN



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 classifier predicts $\hat{y} = 1$ iff

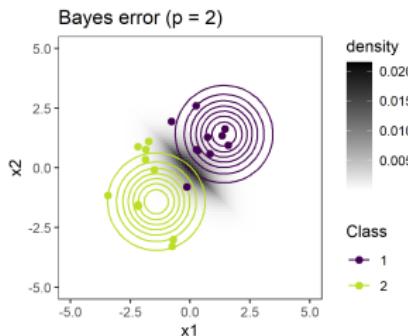
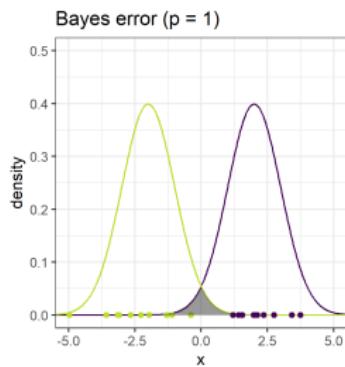
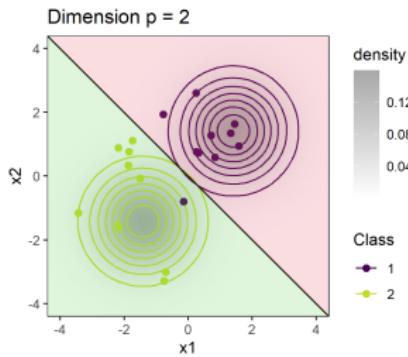
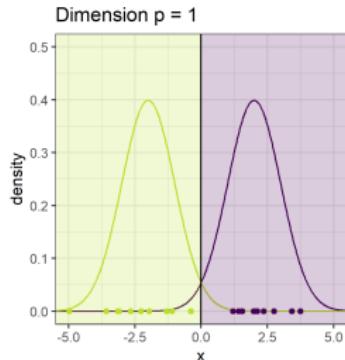
$$\begin{aligned}\mathbb{P}(y = 1 \mid \mathbf{x}) &= \frac{p(\mathbf{x} \mid y = 1)\mathbb{P}(y = 1)}{p(\mathbf{x})} = \frac{1}{2} \cdot \frac{p(\mathbf{x} \mid y = 1)}{p(\mathbf{x})} \\ &\geq \frac{1}{2} \cdot \frac{p(\mathbf{x} \mid y = 2)}{p(\mathbf{x})} \\ &= \frac{p(\mathbf{x} \mid y = 2)\mathbb{P}(y = 2)}{p(\mathbf{x})} = \mathbb{P}(y = 2 \mid \mathbf{x}).\end{aligned}$$

This is equivalent to

$$\begin{aligned}\hat{y} = 1 &\Leftrightarrow \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top(\mathbf{x} - \boldsymbol{\mu})\right) \geq \exp\left(-\frac{1}{2}(\mathbf{x} + \boldsymbol{\mu})^\top(\mathbf{x} + \boldsymbol{\mu})\right) \\ &\Leftrightarrow \mathbf{x}^\top \boldsymbol{\mu} \geq 0.\end{aligned}$$

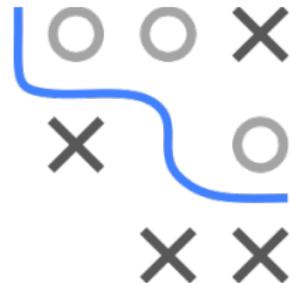
EXAMPLE: K-NN

Optimal Bayes classifier and Bayes error (shaded area):



EXAMPLE: K-NN

We can calculate the corresponding expected misclassification
(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 $\mathbf{x} \mid y$ follows a $\mathcal{N}(\boldsymbol{\mu}, \mathbf{I})$ distribution with $\boldsymbol{\mu} = (a, \dots, a)$).

EXAMPLE: K-NN

We get for the Bayes error:

$$\begin{aligned} &= p \left(\frac{\sum_{i=1}^p \mathbf{x}_i - p \cdot a}{\sqrt{p}} \leq \frac{-p \cdot a}{\sqrt{p}} \mid y = 1 \right) \\ &= \Phi(-\sqrt{p}a) \stackrel{a=\frac{2}{\sqrt{p}}}{=} \Phi(-2) \approx 0.0228, \end{aligned}$$

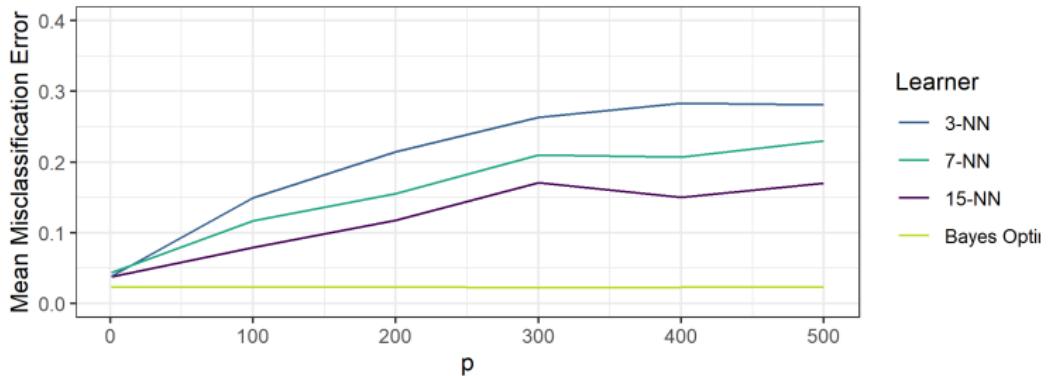
where Φ is the distribution function of a standard normal random variable.

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).

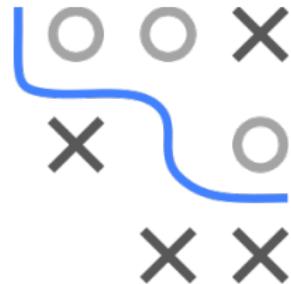


→ k-NN deteriorates quickly with increasing dimension

EXAMPLE: LINEAR MODEL

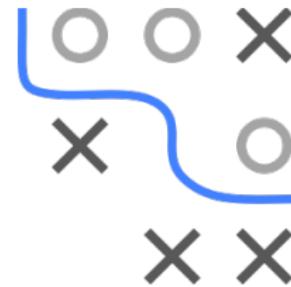
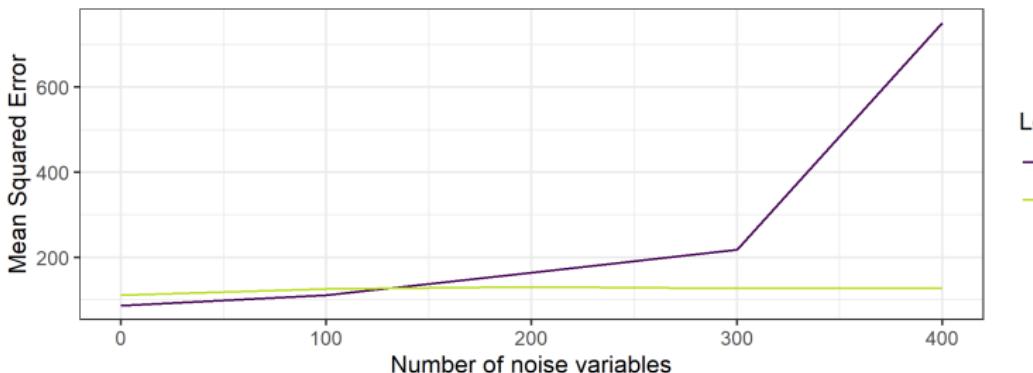
We also investigate how the linear model behaves in high dimensional spaces.

- We take the Boston Housing data set, where the value of the house 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) and look at the performance of a linear model trained on the 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.

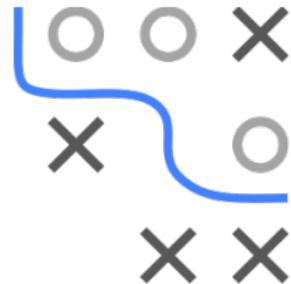
EXAMPLE: LINEAR MODEL

- The regression coefficients of the noise features can not be estimated precisely as zero in the unregularized LM due to 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.

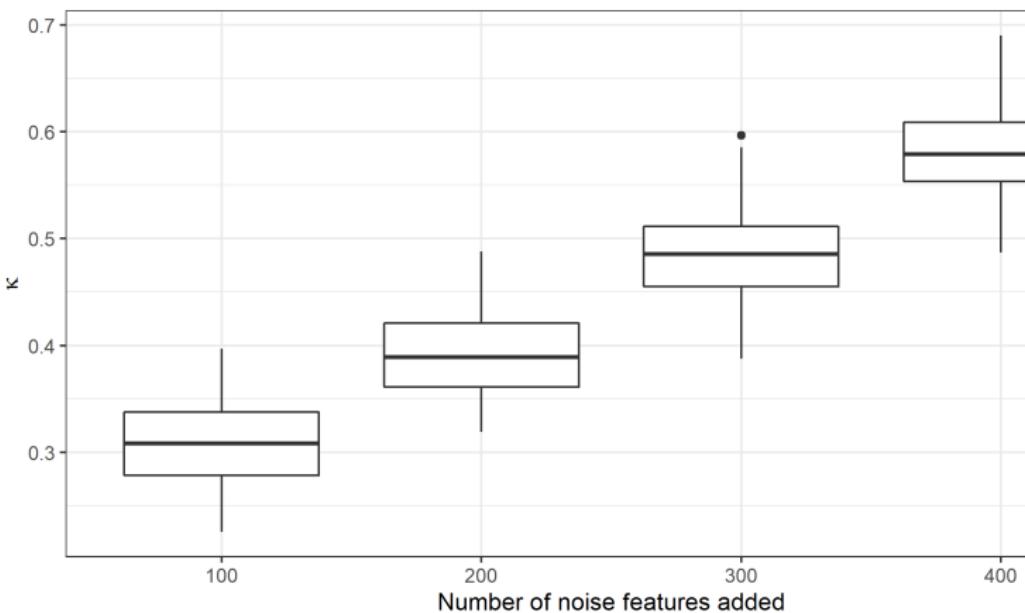
Therefore we decompose the response $\hat{y}^{(i)}$ of each iteration i into $\hat{y}_{\text{true}}^{(i)}$ (predicted with noise features set to 0) and $\hat{y}_{\text{noise}}^{(i)}$ (predicted with true features set to 0), s.t.

$$\hat{y}^{(i)} = \hat{y}_{\text{true}}^{(i)} + \hat{y}_{\text{noise}}^{(i)} + \text{intercept}.$$

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



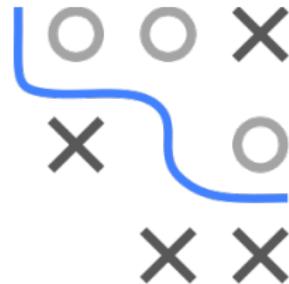
When we add 400 noise features to the model, most of the time average, over 50% of the flexible part of the prediction ($\hat{y}^{(i)}$) – it is determined by the noise features.

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 & 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. based on 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

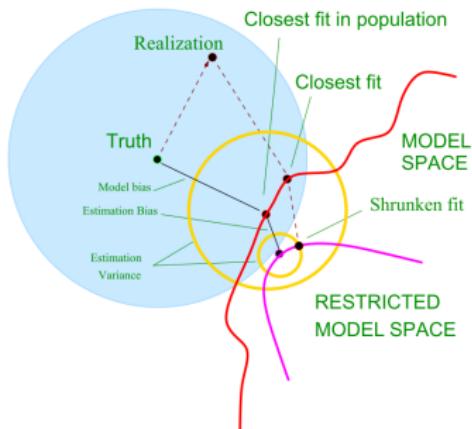
Feature Selection





Introduction to Machine Learning

Introduction to Regularization



Learning goals

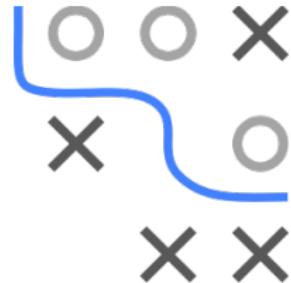
- Understand why overfitting happens
- Know how overfitting can be avoided
- Know regularized empirical risk minimization



Motivation for Regularization

EXAMPLE: OVERFITTING

- Assume we want to predict the daily maximum ozone level given a data set containing 50 observations.
- The data set contains 12 features describing time conditions (e.g., weekday, month), the weather (e.g., temperature at different weather stations, humidity, wind speed) or geographic variables (e.g., the pressure gradient).
- We fit a linear regression model using **all** of the features



$$f(\mathbf{x} | \boldsymbol{\theta}) = \boldsymbol{\theta}^T \mathbf{x} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_{12} x_{12}$$

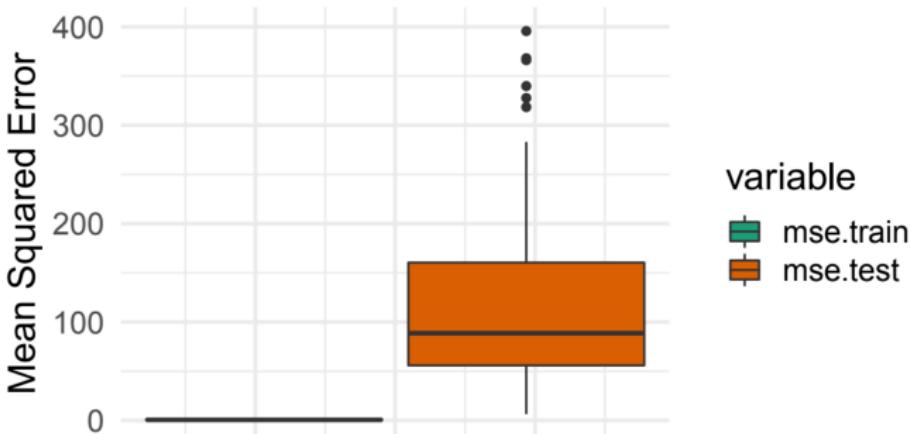
with the L_2 loss.

- We evaluate the performance with 10 times 10-fold CV.

We use (a subset of) the Ozone data set from the mlbench package. This will artificially create a “high-dimensional” dataset by reducing the number of observations drastically while keeping the number of features fixed.

EXAMPLE: OVERFITTING

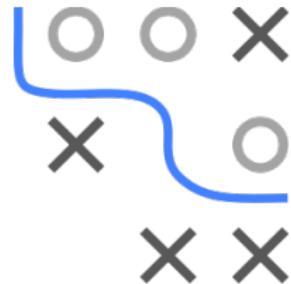
While our model fits the training data almost perfectly (left), it generalizes poorly to new test data (right). We overfitted.



AVOID OVERFITTING

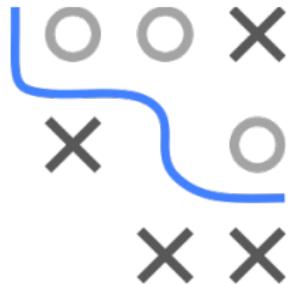
Why can **overfitting** happen? And how to avoid it?

- ➊ Not enough data
→ collect **more data**
- ➋ Data is noisy
→ collect **better data** (reduce noise)
- ➌ Models are too complex
→ use **less complex models**
- ➍ Aggressive loss optimization
→ **optimize less**

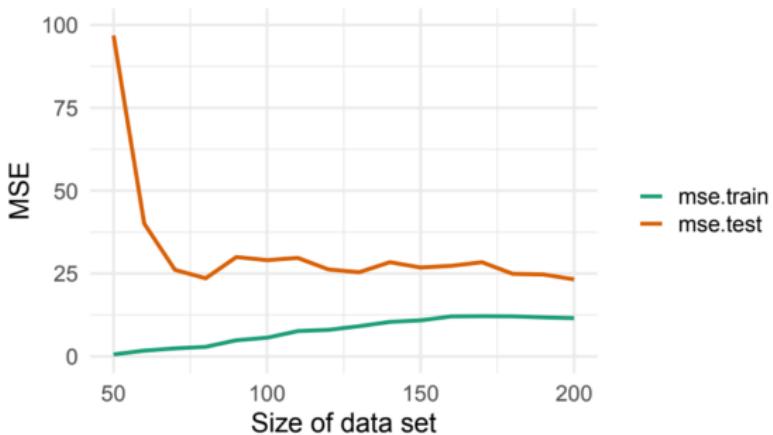


AVOID OVERFITTING

Approach 1: Collect more data



We explore our results for increased dataset size by 10 times 10 CV. The fit worsens slightly, but the test error decreases.



Good idea, but often not feasible in practice.

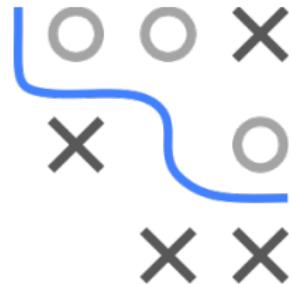
AVOID OVERFITTING

Approach 3: Reduce complexity

We try the simplest model we can think of: the constant model.
L2 loss, the optimal constant model is

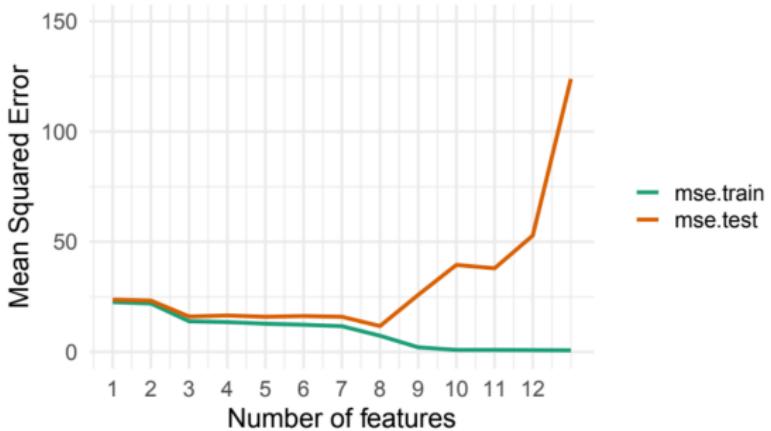
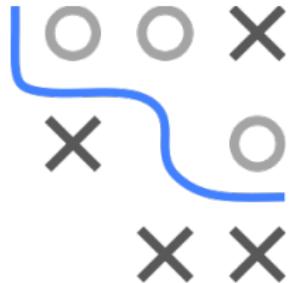
$$f(\mathbf{x} \mid \theta) = \frac{1}{n} \sum_{i=1}^n y^{(i)}$$

We then increase the complexity of the model step-by-step by adding one feature at a time.



AVOID OVERFITTING

We can control the complexity of the model by including/excluding features. We can try out all feature combinations and investigate model fit.



Note: For simplicity, we added the features in one specific (clever) order, so we can do this quickly. Also note there are $2^{12} = 4096$ potential feature combinations.

AVOID OVERFITTING

Approach 4: Optimize less

Now we use polynomial regression with temperature as the only variable to predict the ozone level, i.e.,



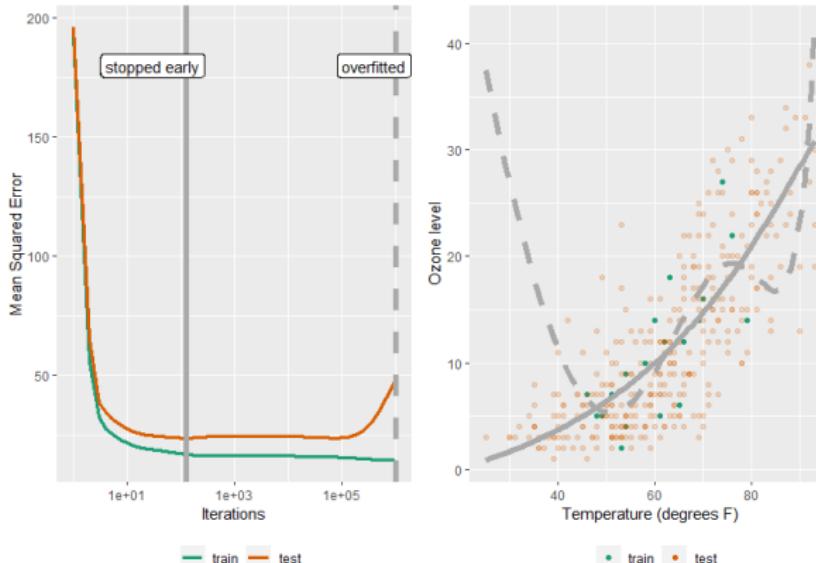
$$f(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{i=0}^d \theta_i (x_T)^i.$$

We choose $d = 15$, for which we get a very flexible model, which can be prone to overfitting for small data sets.

In this example, we don't solve for $\hat{\boldsymbol{\theta}}$ directly, but instead, we use a gradient descent algorithm to find $\hat{\boldsymbol{\theta}}$ stepwise.

AVOID OVERFITTING

We want to stop the optimization early when the generalization starts to degrade.



Note: For polynomial regression, gradient descent usually needs many iterations to converge. It starts to overfit. Hence a very small training set was chosen to accelerate the process.

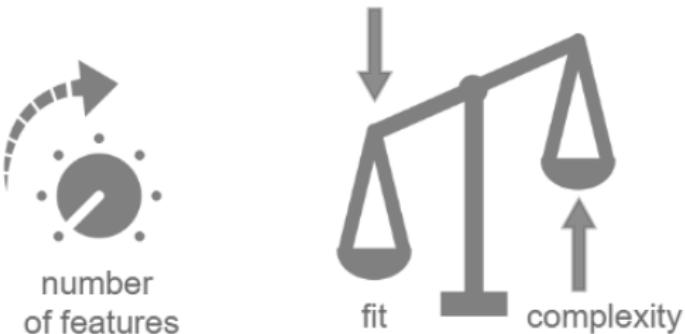
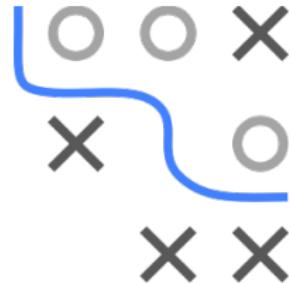


AVOID OVERFITTING

We have contradictory goals

- **maximizing the fit** (minimizing the train loss)
- **minimizing the complexity** of the model.

We need to find the “sweet spot”.



AVOID OVERFITTING

Until now, we can either add a feature completely or not at all.

Instead of controlling the complexity in a discrete way by specifying the number of features, we might prefer to control the complexity **on a continuum** from simple to complex.



complexity
parameter



Regularized Empirical Risk Minimization

REGULARIZED EMPIRICAL RISK MINIMIZATION

Recall, empirical risk minimization with a complex hypothesis set can lead to overfit. A major tool to handle overfitting is **regularization**.



In the broadest sense, regularization refers to any modification of a learning algorithm that is intended to reduce its generalization error but not its training error.

Explicitly or implicitly, such modifications represent the preferences we have regarding the elements of the hypothesis set.

REGULARIZED EMPIRICAL RISK MINIMIZATION

Commonly, regularization takes the following form:



$$\mathcal{R}_{\text{reg}}(f) = \mathcal{R}_{\text{emp}}(f) + \lambda \cdot J(f) = \sum_{i=1}^n L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) + \lambda \cdot$$

- $J(f)$ is called **complexity penalty**, **roughness penalty** or **regularizer**.
- $\lambda > 0$ is called **complexity control** parameter.
- It measures the “complexity” of a model and penalizes it in
- As for \mathcal{R}_{emp} , often \mathcal{R}_{reg} and J are defined on θ instead of $\mathcal{R}_{\text{reg}}(\theta) = \mathcal{R}_{\text{emp}}(\theta) + \lambda \cdot J(\theta)$.

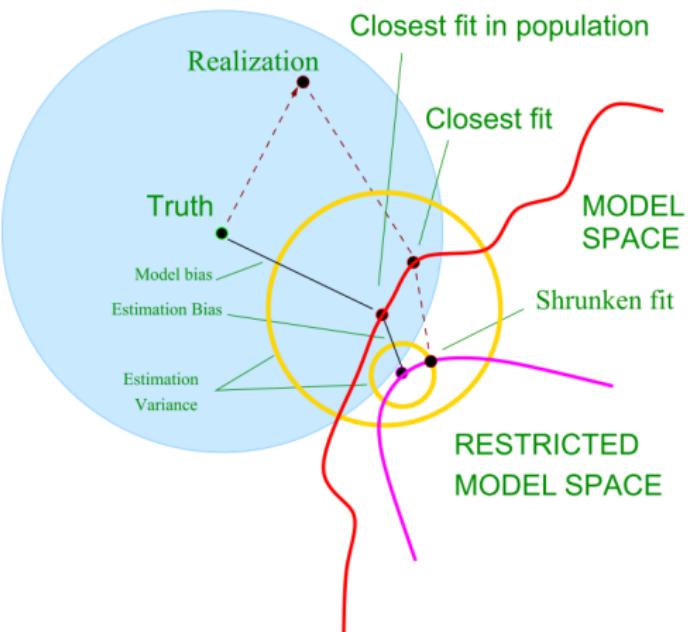
REGULARIZED EMPIRICAL RISK MINIMIZATION

Remarks:

- Note that we now face an optimization problem with two constraints:
 - ➊ models should fit well (low empirical risk),
 - ➋ but not be too complex (low $J(f)$).
- We decide to combine the two in a weighted sum and to control the trade-off via the complexity control parameter λ .
- λ is hard to set manually and is usually selected via cross-validation (see later).
- $\lambda = 0$: The regularized risk $\mathcal{R}_{\text{reg}}(f)$ reduces to the simple empirical $\mathcal{R}_{\text{emp}}(f)$.
- If λ goes to infinity, we stop caring about the loss/fit and f becomes as “simple” as possible.



REGULARIZED EMPIRICAL RISK MINIMIZATION



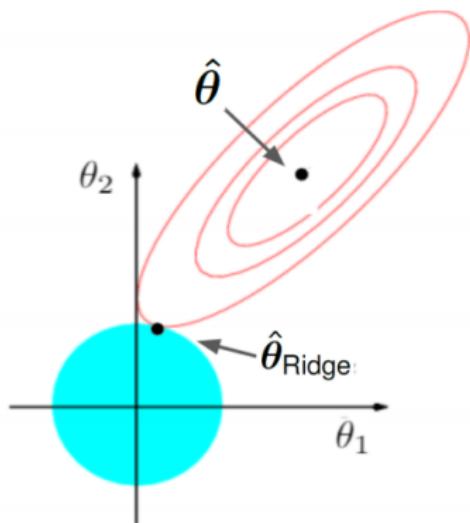
Hastie, The Elements of Statistical Learning, 2009 (p. 225)



Introduction to Machine Learning



Lasso and Ridge Regression



Learning goals

- Know the regularized linear model
- Know Ridge regression (L_2 penalty)
- Know Lasso regression (L_1 penalty)

REGULARIZATION IN THE LINEAR MODEL

- Linear models can also overfit if we operate in a high-dimensional space with not that many observations.
- OLS usually require a full-rank design matrix.
- When features are highly correlated, the least-squares estimate becomes highly sensitive to random errors in the observed response, producing a large variance in the fit.
- We now add a complexity penalty to the loss:

$$\mathcal{R}_{\text{reg}}(\theta) = \sum_{i=1}^n \left(y^{(i)} - \theta^\top \mathbf{x}^{(i)} \right)^2 + \lambda \cdot J(\theta).$$

- Intuitive to measure model complexity as deviation from the 0-origin, as the 0-model is empty and contains no effects. Models close to this either have few active features or only weak effects.
- So we measure $J(\theta)$ through a vector norm. This shrinks coefficients closer 0, hence the term **shrinkage methods**.



RIDGE REGRESSION



Ridge regression uses a simple $L2$ penalty:

$$\begin{aligned}\hat{\theta}_{\text{Ridge}} &= \arg \min_{\theta} \sum_{i=1}^n \left(y^{(i)} - \theta^T \mathbf{x}^{(i)} \right)^2 + \lambda \|\theta\|_2^2 \\ &= \arg \min_{\theta} (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) + \lambda \theta^T \theta.\end{aligned}$$

Optimization is possible (as in the normal LM) in analytical form

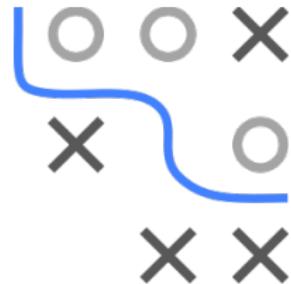
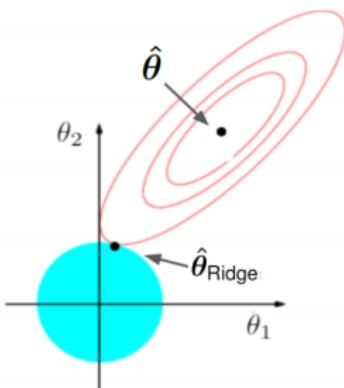
$$\hat{\theta}_{\text{Ridge}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Name comes from the fact that we add positive entries along the diagonal "ridge" $\mathbf{X}^T \mathbf{X}$.

RIDGE REGRESSION

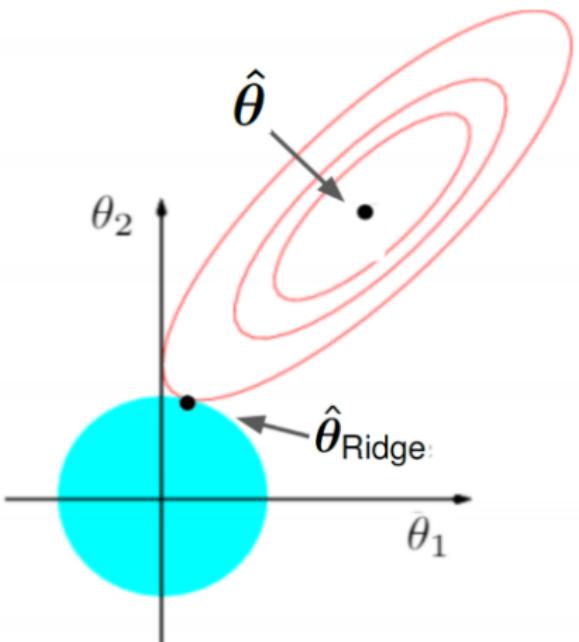
We understand the geometry of these 2 mixed components in c regularized risk objective much better, if we formulate the optim as a constrained problem (see this a Lagrange multipliers in rev

$$\begin{aligned} \min_{\theta} \quad & \sum_{i=1}^n \left(y^{(i)} - f(\mathbf{x}^{(i)} | \theta) \right)^2 \\ \text{s.t.} \quad & \|\theta\|_2^2 \leq t \end{aligned}$$



NB: Relationship between λ and t will be explained later.

RIDGE REGRESSION



- We still optimize the \mathcal{R}_{emp} cannot leave a ball around $\hat{\theta}$.
- $\mathcal{R}_{\text{emp}}(\theta)$ grows monotonically as we move away from $\hat{\theta}$.
- Inside constraints perspective: From origin, jump from contour line to contour line (better) until infeasible, stop before.
- Outside constraints perspective: From $\hat{\theta}$, jump from contour line to contour line (worse) until feasible, stop then.
- So our new optimum will be on the boundary of that ball.

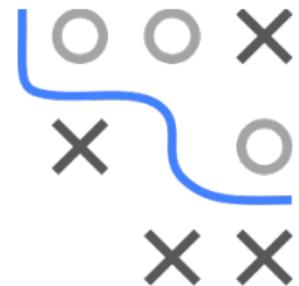


EXAMPLE: POLYNOMIAL RIDGE REGRESSION

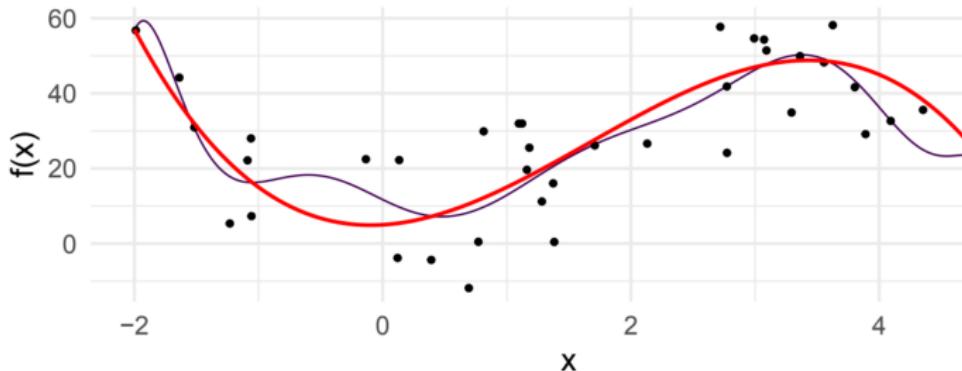
True (unknown) function is $f(x) = 5 + 2x + 10x^2 - 2x^3 + \epsilon$ (in noise)

Let us consider a d th-order polynomial

$$f(x) = \theta_0 + \theta_1 x + \cdots + \theta_d x^d = \sum_{j=0}^d \theta_j x^j.$$

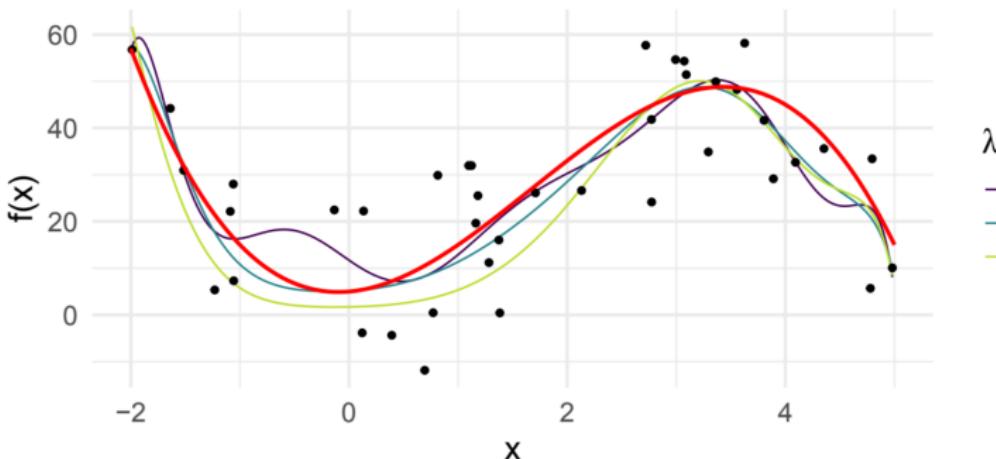


Using model complexity $d = 10$ overfits:



EXAMPLE: POLYNOMIAL RIDGE REGRESSION

With an L_2 penalty we can now select d "too large" but regularize the model by shrinking its coefficients. Otherwise we have to optimize the discrete d .

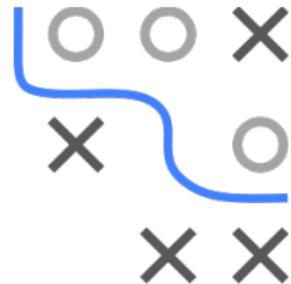


λ	β_0	β_1	β_2	β_3	β_4	β_5	β_6	β_7	β_8	β_9
0.00	12.00	-16.00	4.80	23.00	-5.40	-9.30	4.20	0.53	-0.63	0.13
10.00	5.20	1.30	3.70	0.69	1.90	-2.00	0.47	0.20	-0.14	0.03
100.00	1.70	0.46	1.80	0.25	1.80	-0.94	0.34	-0.01	-0.06	0.02

LASSO REGRESSION

Another shrinkage method is the so-called **Lasso regression**, uses an $L1$ penalty on θ :

$$\begin{aligned}\hat{\theta}_{\text{Lasso}} &= \arg \min_{\theta} \sum_{i=1}^n \left(y^{(i)} - \theta^T \mathbf{x}^{(i)} \right)^2 + \lambda \|\theta\|_1 \\ &= \arg \min_{\theta} (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) + \lambda \|\theta\|_1.\end{aligned}$$

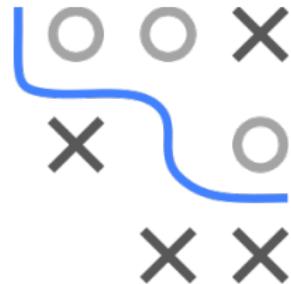


Note that optimization now becomes much harder. $\mathcal{R}_{\text{reg}}(\theta)$ is not convex, but we have moved from an optimization problem with an analytical solution towards a non-differentiable problem.

Name: least absolute shrinkage and selection operator.

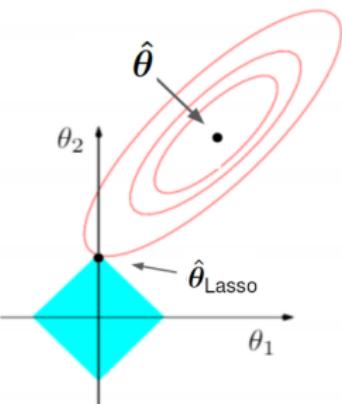
LASSO REGRESSION

We can also rewrite this as a constrained optimization problem.
penalty results in the constrained region to look like a diamond



$$\min_{\theta} \quad \sum_{i=1}^n \left(y^{(i)} - f(\mathbf{x}^{(i)} | \theta) \right)^2$$

subject to: $\|\theta\|_1 \leq t$



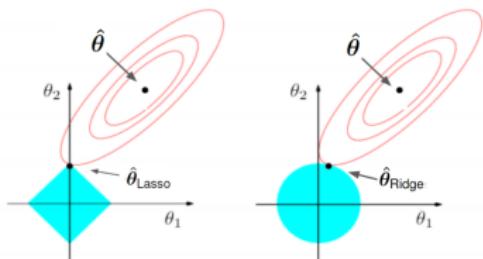


Introduction to Machine Learning

Lasso vs. Ridge Regression

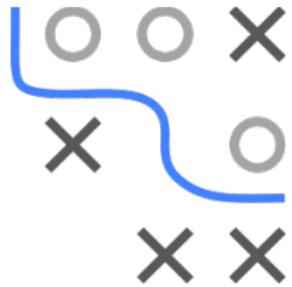
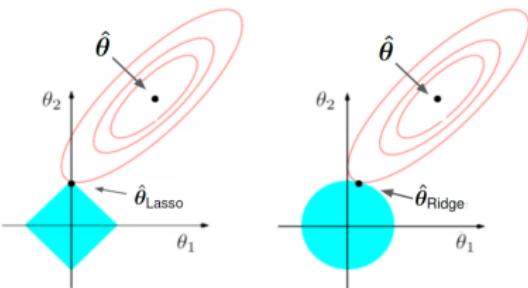
Learning goals

- Know the geometry of Ridge v Lasso regularization
- Understand the effects of the methods on model coefficients
- Understand that Lasso creates sparse solutions



LASSO VS. RIDGE GEOMETRY

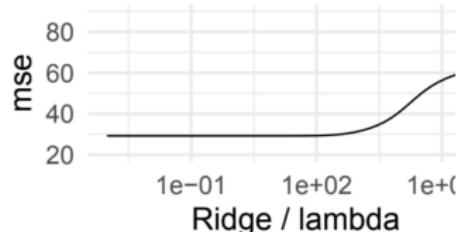
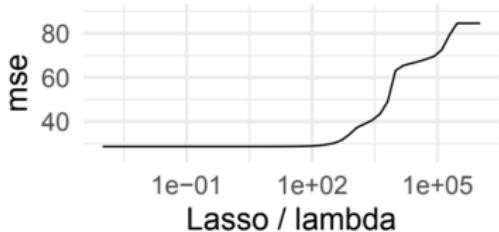
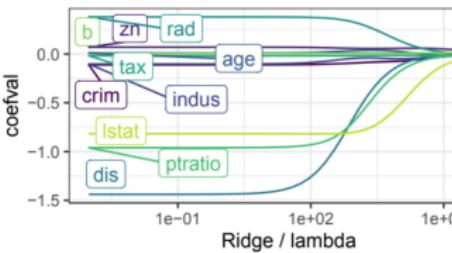
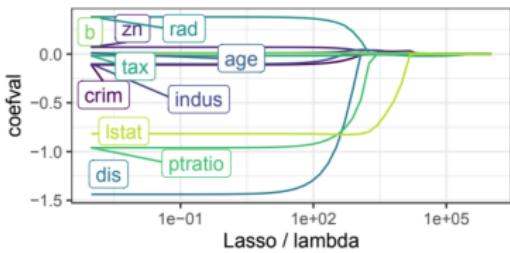
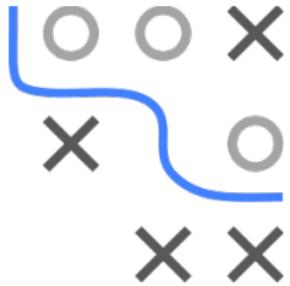
$$\min_{\theta} \sum_{i=1}^n \left(y^{(i)} - f(\mathbf{x}^{(i)} | \theta) \right)^2 \quad \text{s.t. } \|\theta\|_p^p \leq t$$



- In both cases, the solution which minimizes $\mathcal{R}_{\text{reg}}(\theta)$ is always on the boundary of the feasible region (for sufficiently large λ).
- As expected, $\hat{\theta}_{\text{Lasso}}$ and $\hat{\theta}_{\text{Ridge}}$ have smaller parameter norms than $\hat{\theta}$.
- For Lasso, the solution likely touches vertices of the constraint region. This induces sparsity and is a form of variable selection.
- In the $p > n$ case, the Lasso selects at most n features (due to the nature of the convex optimization problem).

COEFFICIENT PATHS AND 0-SHRINKAGE

Example 1: Boston Housing (few features removed for readability)
We cannot overfit here with an unregularized linear model as the data is so low-dimensional. But we see how only Lasso shrinks to sparse paths.



Coefficient paths and cross-val. MSE for λ values for Ridge and Lasso

COEFFICIENT PATHS AND 0-SHRINKAGE



Example 2: High-dimensional simulated data

We simulate a continuous, correlated dataset with 50 features, observations $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(100)} \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{0}, \Sigma)$ and

$$y = 10 \cdot (x_1 + x_2) + 5 \cdot (x_3 + x_4) + \sum_{j=5}^{14} x_j + \epsilon$$

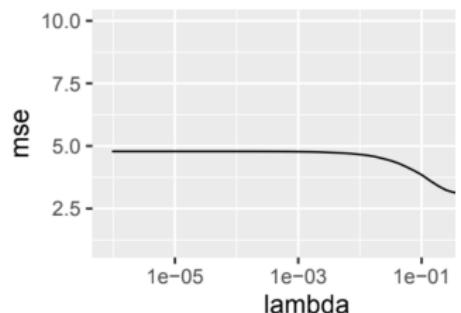
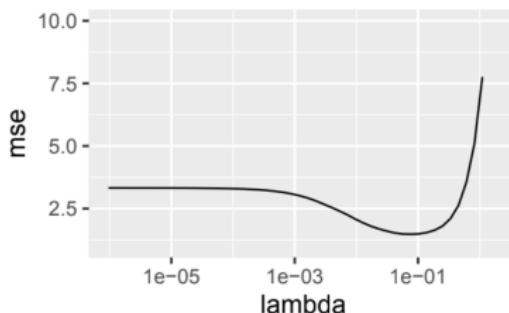
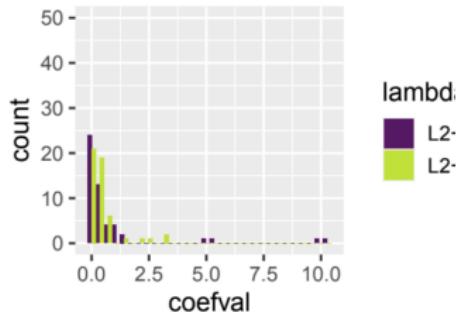
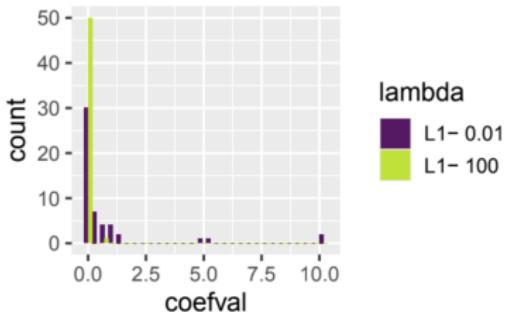
where $\epsilon \sim \mathcal{N}(0, 1)$ and $\forall k, l \in \{1, \dots, 50\}$:

$$\text{Cov}(x_k, x_l) = \begin{cases} 0.7^{|k-l|} & \text{for } k \neq l \\ 1 & \text{else} \end{cases}.$$

Note that 36 of the 50 features are noise variables.

COEFFICIENT PATHS AND 0-SHRINKAGE

Coefficient histograms for different λ values for Ridge and Lass high-dimensional data along with the cross-validated MSE.



REGULARIZATION AND FEATURE SCALING

- Note that very often we do not include θ_0 in the penalty term (but this can be implementation-dependent).
- These methods are typically not equivariant under scaling inputs, so one usually standardizes the features.
- Note that for a normal LM, if you scale some features, we simply "anti-scale" the coefficients the same way. The risk not change. For regularized models this is not so simple. If scale features to smaller values, coefficients have to become larger to counteract. They now are penalized more heavily. Such a scaling would make some features less attractive without changing anything relevant in the data.



REGULARIZATION AND FEATURE SCALING



Example:

- Let the true data generating process be

$$y = x_1 + \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1).$$

- Let there be 5 features $x_1, \dots, x_5 \sim \mathcal{N}(0, 1)$.
- Using the Lasso (package `glmnet`), we get

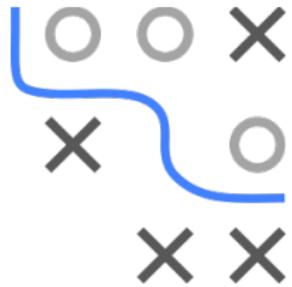
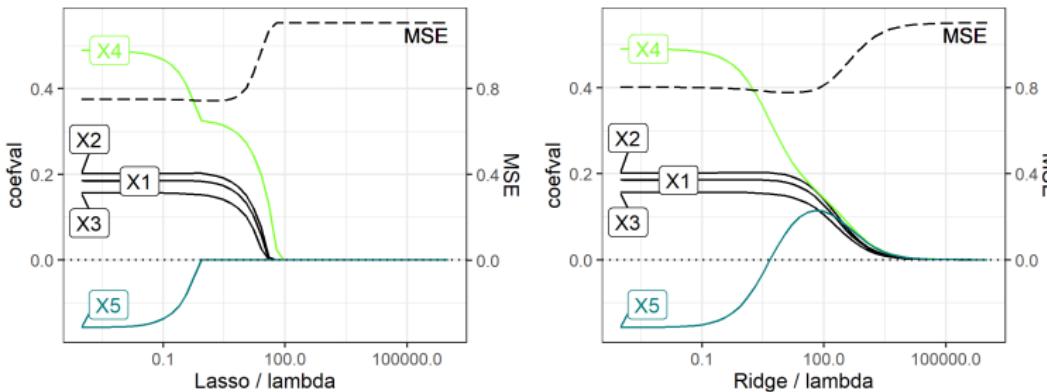
	(Intercept)	x1	x2	x3	x4
	-0.056	0.489	0.000	0.000	0.000

- But if we rescale any of the noise features, say $x_2 = 1000x_2$ and don't use standardization, we get

	(Intercept)	x1	x2	x3	x4
	-0.106830	0.000000	-0.000013	0.000000	0.000000

- This is due to the fact, that the coefficient of x_2 will live on a small scale as the covariate itself is large. The feature will be less penalized by the L_1 -norm and is favored by Lasso.

CORRELATED FEATURES



Fictional example for the model

$y = 0.2X_1 + 0.2X_2 + 0.2X_3 + 0.2X_4 + 0.2X_5 + \epsilon$ of 100 observations
 $\epsilon \sim \mathcal{N}(0, 1)$. X_1-X_4 are independently drawn from different normal distributions: $X_1, X_2, X_3, X_4 \sim \mathcal{N}(0, 2)$. While X_1-X_4 have pairwise correlation coefficients of 0, X_4 and X_5 are nearly perfectly correlated: $X_5 = X_4 + \delta, \delta \sim \mathcal{N}(0, 0.3), \rho(X_4, X_5) = 0.98$.

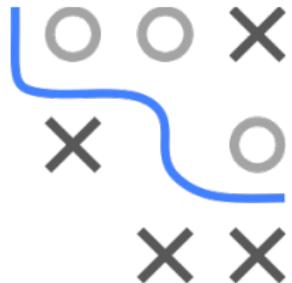
We see that Lasso shrinks the coefficient for X_5 to zero early or Ridge assigns similar coefficients to X_4, X_5 for larger λ .

SUMMARIZING COMMENTS

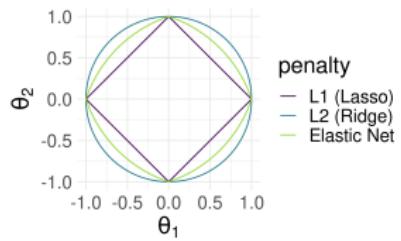
- Neither one can be classified as overall better.
- Lasso is likely better if the true underlying structure is sparse: only few features influence y . Ridge works well if there are influential features.
- Lasso can set some coefficients to zero, thus performing variable selection, while Ridge regression usually leads to smaller estimated coefficients, but still dense θ vectors.
- Lasso has difficulties handling correlated predictors. For high correlation Ridge dominates Lasso in performance.
- For Lasso one of the correlated predictors will have a large coefficient, while the rest are (nearly) zeroed. The respective feature is, however, selected randomly.
- For Ridge the coefficients of correlated features are similar.



Introduction to Machine Learning



Elastic Net and Regularization for GL



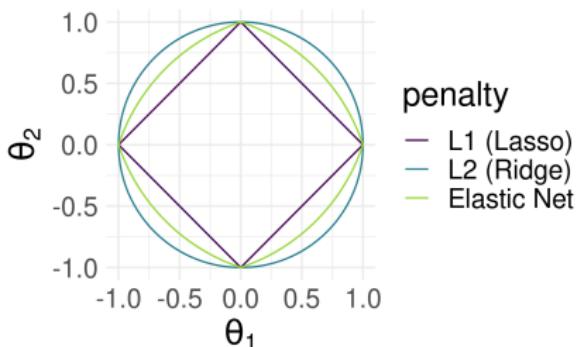
Learning goals

- Know the elastic net as compromise between Ridge and Lasso regression
- Know regularized logistic regression

ELASTIC NET

Elastic Net combines the L_1 and L_2 penalties:

$$\mathcal{R}_{\text{elnet}}(\boldsymbol{\theta}) = \sum_{i=1}^n (y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)})^2 + \lambda_1 \|\boldsymbol{\theta}\|_1 + \lambda_2 \|\boldsymbol{\theta}\|_2^2.$$



- Correlated predictors tend to be either selected or zeroed together.
- Selection of more than n features possible for $p > n$.

ELASTIC NET

Simulating two examples with each 50 data sets and 100 observations each:

$$\mathbf{y} = \mathbf{X}\beta + \sigma\epsilon, \quad \epsilon \sim N(0, 1), \quad \sigma = 1$$



Ridge performs better for:

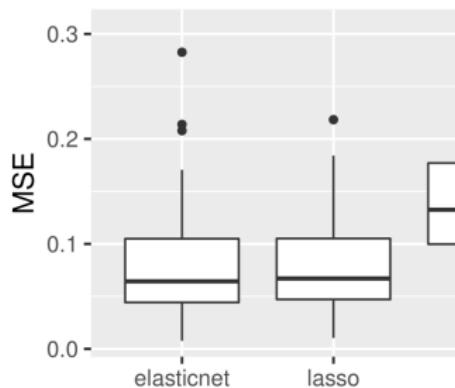
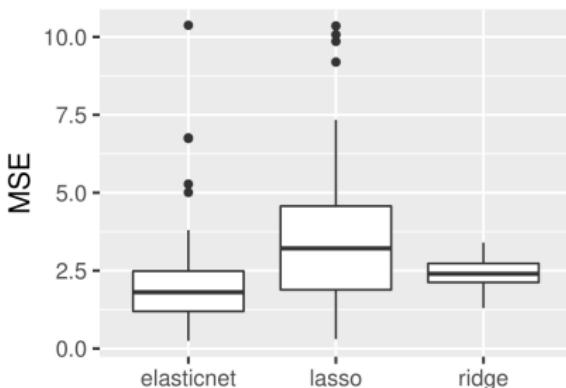
$$\beta = (\underbrace{2, \dots, 2}_{5}, \underbrace{0, \dots, 0}_{5})$$

$$\text{corr}(\mathbf{X}_i, \mathbf{X}_j) = 0.8^{|i-j|} \text{ for all } i \text{ and } j$$

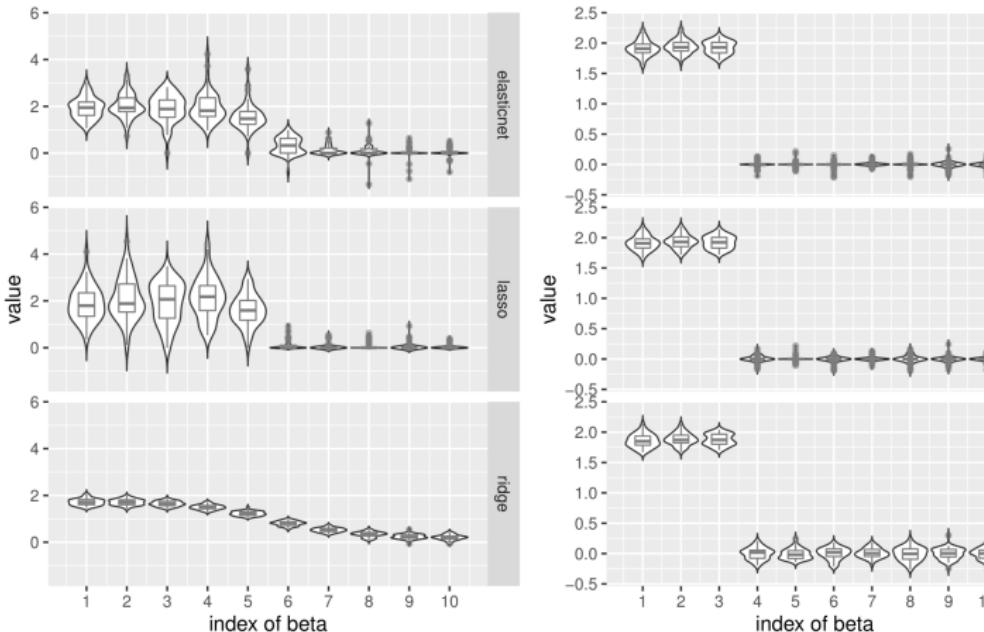
Lasso performs better

$$\beta = (2, 2, 2, \underbrace{0, \dots, 0}_{7})$$

$$\text{corr}(\mathbf{X}_i, \mathbf{X}_j) = 0 \text{ for all } i \neq j, c$$



ELASTIC NET

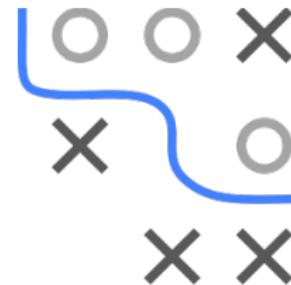


Since Elastic Net offers a compromise between Ridge and Lasso, it is suitable for both data situations.

REGULARIZED LOGISTIC REGRESSION

Regularizers can be added very flexibly to basically any model based on ERM.

Hence, we can, e.g., construct L_1 - or L_2 -penalized logistic regression to enable coefficient shrinkage and variable selection in this mode

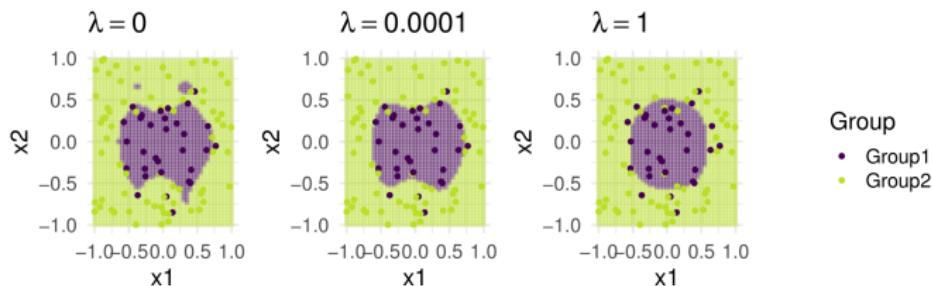


$$\begin{aligned}\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) &= \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + \lambda \cdot J(\boldsymbol{\theta}) \\ &= \sum_{i=1}^n \log \left[1 + \exp \left(-2y^{(i)} f \left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta} \right) \right) \right] + \lambda \cdot J\end{aligned}$$

REGULARIZED LOGISTIC REGRESSION

We fit a logistic regression model using polynomial features for x_2 with maximum degree of 7. We add an L_2 penalty. We see fo

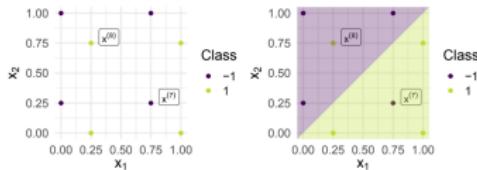
- $\lambda = 0$: The unregularized model seems to overfit.
- $\lambda = 0.0001$: Regularization helps to learn the underlying mechanism.
- $\lambda = 1$: The real data-generating process is captured very '





Introduction to Machine Learning

Regularization for Underdetermined Problems



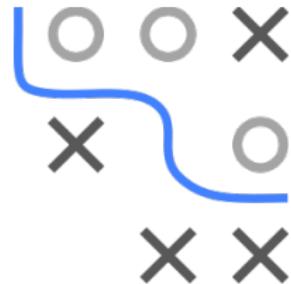
Learning goals

- Understand that regularization used to make ill-posed problem well defined
- Know that regularization can guarantee convergence for logistic regression on a linearly separable dataset

UNDERDETERMINED PROBLEMS

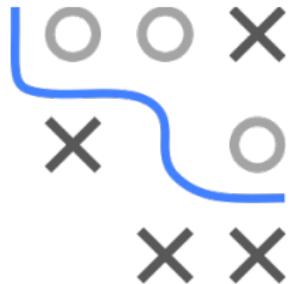
Regularization can also be motivated from a numerical perspective

- Regularization can sometimes be necessary to make certain ill-posed problems well defined. Linear models such as (linear) regression and PCA depend "inverting" / solving a linear system which not always works.
- When we solve linear systems like $\mathbf{X}\theta = \mathbf{y}$, there are 3 cases:
 - ① \mathbf{X} is of square form and has full rank. This is normal linear system and irrelevant for us here, now.
 - ② \mathbf{X} has more rows than columns. The system is "overdetermined". Try to solve $\mathbf{X}\theta \approx \mathbf{y}$, by minimizing $||\mathbf{X}\theta - \mathbf{y}||$. Ideally, this difference would be zero, but due to the too many rows this is often not possible. This is equivalent to linear regression!
 - ③ \mathbf{X} has more columns than rows / linear dependence between columns exists. Now there are usually an infinite number of solutions. We define a "preference" for them to make the problem well-defined (familiar?). Such problems are called "underdetermined".



UNDERDETERMINED PROBLEMS

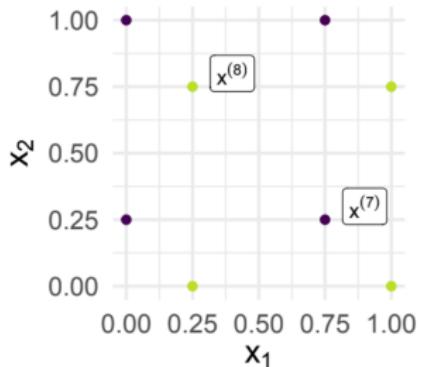
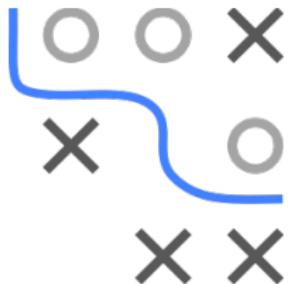
- A very old and well-known approach in underdetermined problems is to still reduce the problem to optimization by minimizing $\|\mathbf{y} - \mathbf{X}\mathbf{x}\|$ but adding a small positive constant to the diagonal of $\mathbf{X}^T\mathbf{X}$.
- In optimization / numerical analysis this is known as **Tikhonov** regularization.
- But as you should be able to see now: This is completely equivalent to Ridge regression!



UNDETERMINED PROBLEMS

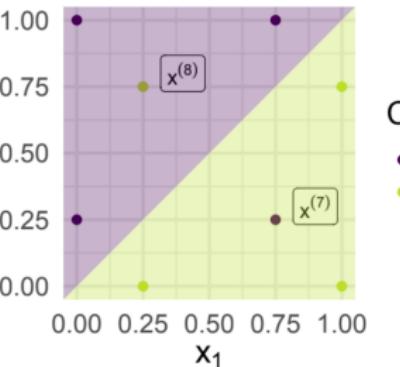
We now study not the normal LM (which we could), but logistic regression applied to a linearly separable dataset for a more subtle example:

First, we take a look at logistic regression for an almost linearly separable dataset consisting of the observations $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(8)}$.



Class

- 1
- 1



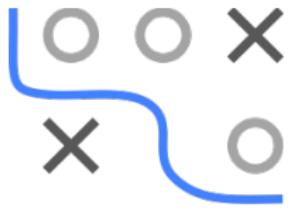
Note: WLOG we estimate the model without intercept, s.t. we can visualize the regression coefficient θ in 2D. Also, the symmetry of the data does not influence the generality of our conclusions.

UNDETERMINED PROBLEMS

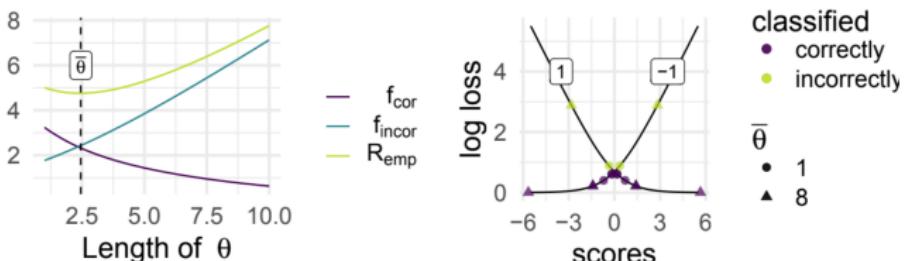
Because of the symmetry of the data, the direction¹ of θ is $\tilde{\theta} := (\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}})^\top$

To find $\bar{\theta} := \|\theta\|_2$, we consider the empirical risk \mathcal{R}_{emp} along $\tilde{\theta}$:

$$\begin{aligned}\mathcal{R}_{\text{emp}} &= \sum_{i=1}^8 \log \left[1 + \exp \left(-y^{(i)} \theta^\top \mathbf{x}^{(i)} \right) \right] \\ &= \underbrace{\sum_{i=1}^6 \log \left[1 + \exp \left(-\bar{\theta} |\tilde{\theta}^\top \mathbf{x}^{(i)}| \right) \right]}_{=: f_{\text{cor}}(\bar{\theta}) \text{ (correctly classified)}} + \underbrace{\sum_{i=7}^8 \log \left[1 + \exp \left(\bar{\theta} |\tilde{\theta}^\top \mathbf{x}^{(i)}| \right) \right]}_{=: f_{\text{incor}}(\bar{\theta}) \text{ (incorrectly classified)}}\end{aligned}$$



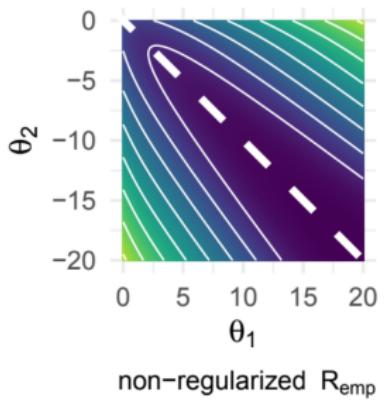
Clearly, $f_{\text{cor}} / f_{\text{incor}}$ are monotonically decreasing/increasing with rising length c



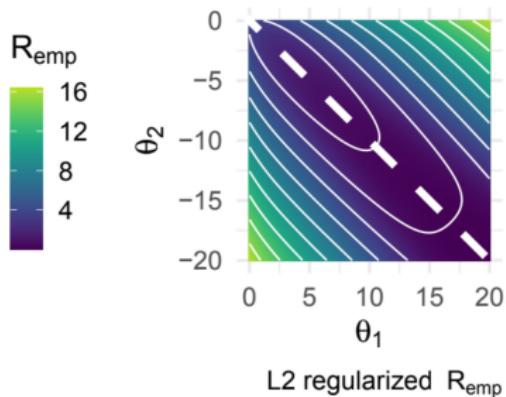
¹ θ is perpendicular to the decision boundary and points to the "1"-space

UNDERCONSTRAINED PROBLEMS

- By removing the 7th and 8th observation, we get a linearly separable dataset.
- This also means that we lose our "counterweight", i.e., if a parameter vector is able to classify the samples perfectly, the vector 2θ also classifies the samples perfectly, with decreased risk.
- Therefore, an iterative optimizer such as stochastic gradient descent (SGD) will continually increase θ and never halt (in theory).
- In such cases, regularization can guarantee convergence:



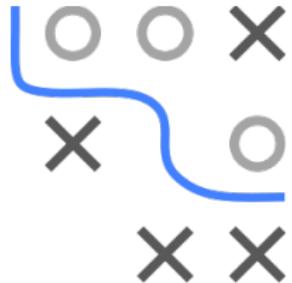
non-regularized R_{emp}



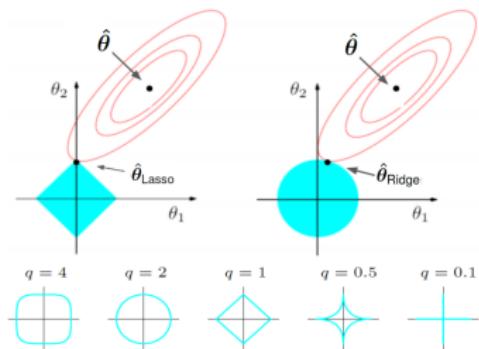
L2 regularized R_{emp}



Introduction to Machine Learning



L0 Regularization



Learning goals

- Know LQ norm regularization
- Understand that L0 norm regularization simply counts the number of non-zero parameters

LQ NORM REGULARIZATION

Besides L_1 and L_2 norm we could use any L_q norm for regularization

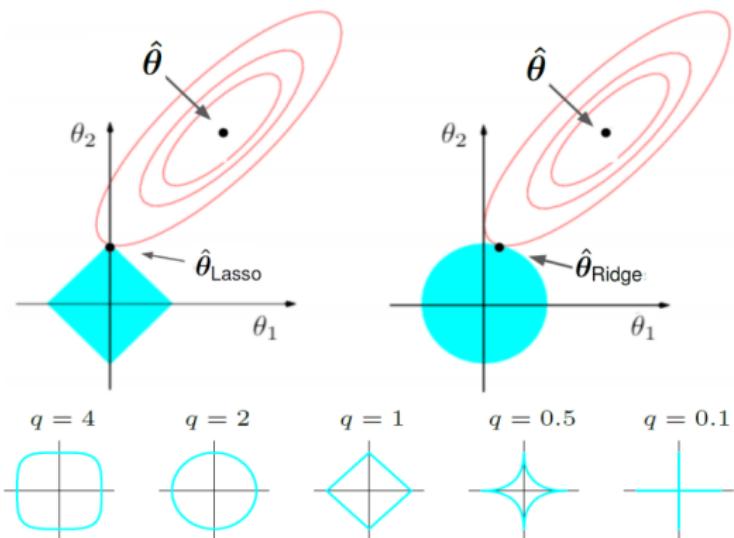
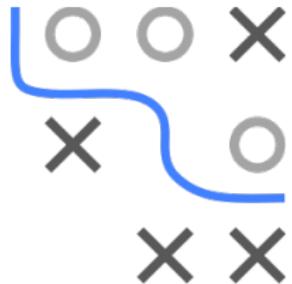


Figure: Top: Ridge and Lasso loss contours and feasible regions. Bottom: Different feasible region shapes for L_q norms $\sum_j |\theta_j|^q$.

L0 REGULARIZATION

- Consider the L_0 -regularized risk of a model $f(\mathbf{x} | \theta)$

$$\mathcal{R}_{\text{reg}}(\theta) = \mathcal{R}_{\text{emp}}(\theta) + \lambda \|\theta\|_0 := \mathcal{R}_{\text{emp}}(\theta) + \lambda \sum_j |\theta_j|$$



- Unlike the L_1 and L_2 norms, the L_0 "norm" simply counts the number of non-zero parameters in the model.

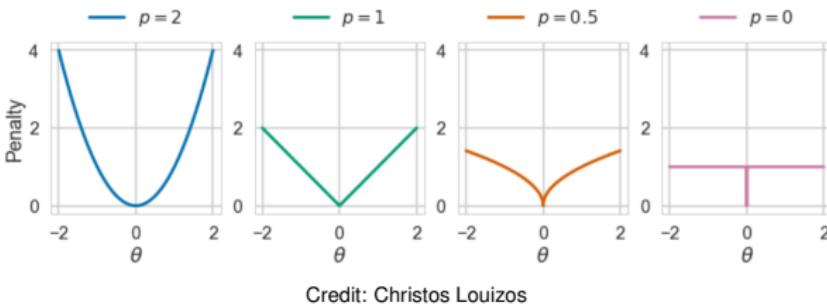
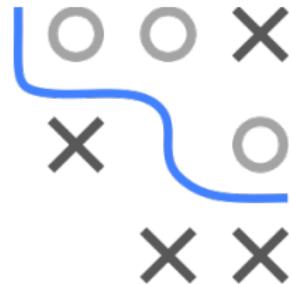
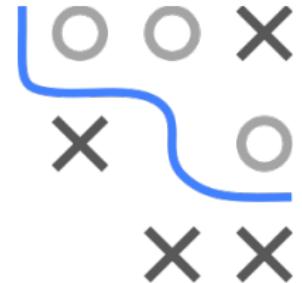


Figure: L_p norm penalties for a parameter θ according to different

L₀ REGULARIZATION

- For any parameter θ , the L_0 penalty is zero for $\theta = 0$ ($0^0 := 0$) and is constant for any $\theta \neq 0$, no matter how large or small it is.
- L_0 regularization induces sparsity in the parameter vector more aggressively than L_1 regularization, but does not shrink coefficient values as L_1 and L_2 does.
- Model selection criteria such as Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) are special cases of L_0 regularization (corresponding to specific values of λ).
- The L_0 -regularized risk is neither continuous, differentiable nor convex.
- It is computationally hard to optimize (NP-hard) and likely intractable. For smaller n and p we might be able to solve it nowadays directly, for larger scenarios efficient approximations of the L_0 are still topic of current research.



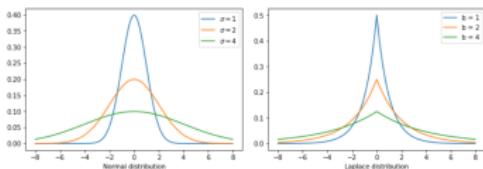


Introduction to Machine Learning

Regularization in Non-Linear Models Bayesian Priors

Learning goals

- Understand that regularization and parameter shrinkage can be applied to non-linear models
- Know structural risk minimization
- Know how regularization risk minimization is the same as MLE in a Bayesian perspective, where the penalty corresponds to the parameter prior.



SUMMARY: REGULARIZED RISK MINIMIZATION

If we should define ML in only one line, this might be it:

$$\min_{\theta} \mathcal{R}_{\text{reg}}(\theta) = \min_{\theta} \left(\sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)} | \theta)) + \lambda \cdot J(\theta) \right)$$



We can choose for a task at hand:

- the **hypothesis space** of f , which determines how feature influence the predicted y
- the **loss** function L , which measures how errors should be
- the **regularization** $J(\theta)$, which encodes our inductive bias preference for certain simpler models

By varying these choices one can construct a huge number of different ML models. Many ML models follow this construction principle and can be interpreted through the lens of regularized risk minimization.

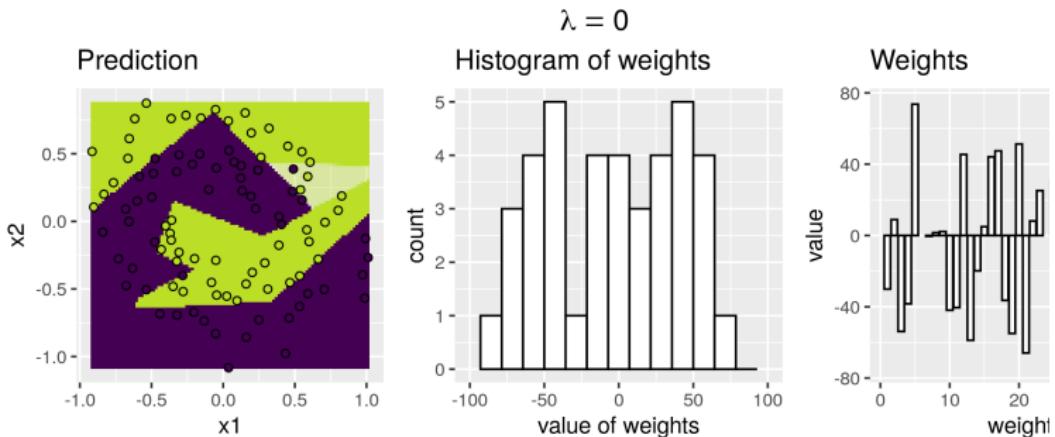
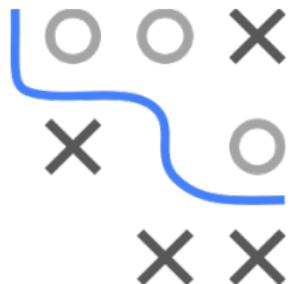
REGULARIZATION IN NONLINEAR MODELS

- So far we have mainly considered regularization in LMs.
- Can also be applied to non-linear models (with numeric parameters), where it is often important to prevent overfitting.
- Here, we typically use L_2 regularization, which still results in parameter shrinkage and weight decay.
- By adding regularization, prediction surfaces in regression and classification become smoother.
- Note: In the chapter on non-linear SVMs we will study the effect of regularization on a non-linear model in detail.



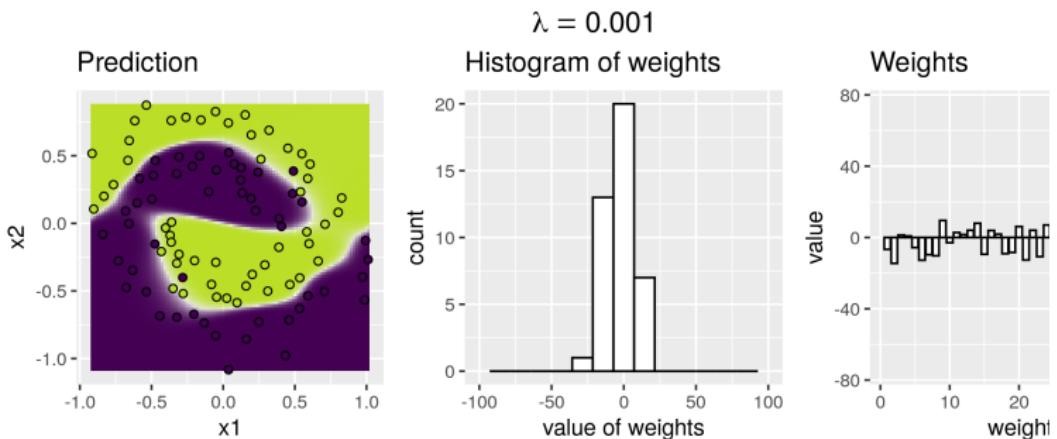
REGULARIZATION IN NONLINEAR MODELS

Setting: Classification for the spirals data. Neural network with sin hidden layer containing 10 neurons and logistic output activation, reg with L_2 penalty term for $\lambda > 0$. Varying λ affects smoothness of the c boundary and magnitude of network weights:



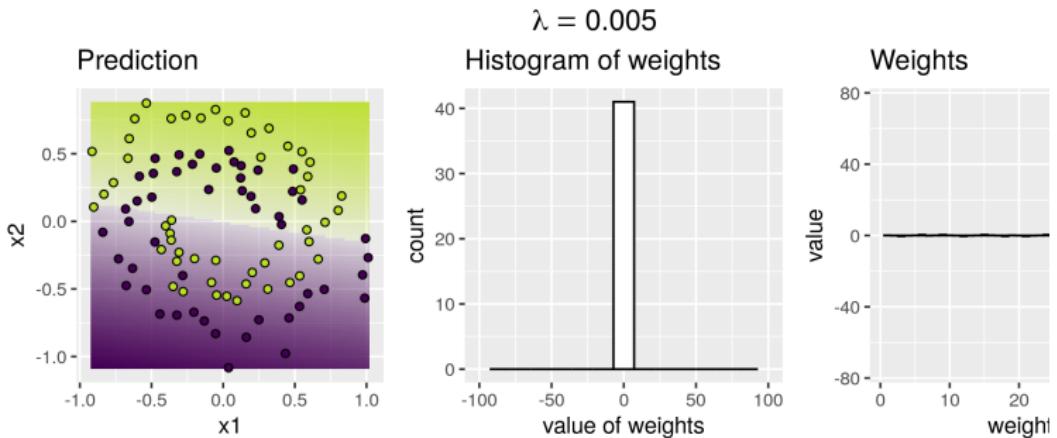
REGULARIZATION IN NONLINEAR MODELS

Setting: Classification for the spirals data. Neural network with sin hidden layer containing 10 neurons and logistic output activation, reg with L_2 penalty term for $\lambda > 0$. Varying λ affects smoothness of the c boundary and magnitude of network weights:



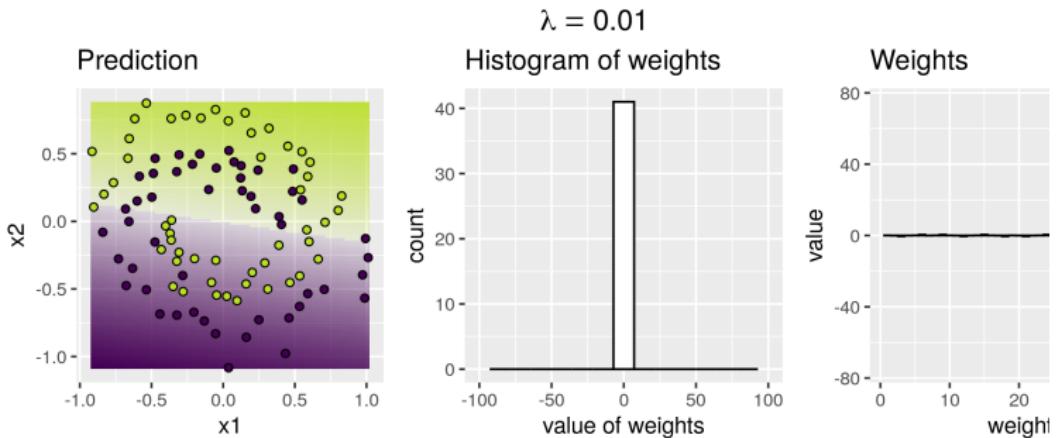
REGULARIZATION IN NONLINEAR MODELS

Setting: Classification for the spirals data. Neural network with sin hidden layer containing 10 neurons and logistic output activation, reg with L_2 penalty term for $\lambda > 0$. Varying λ affects smoothness of the c boundary and magnitude of network weights:



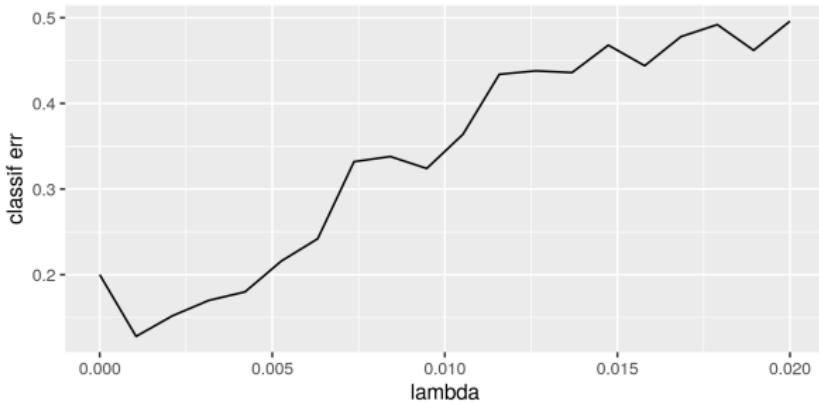
REGULARIZATION IN NONLINEAR MODELS

Setting: Classification for the spirals data. Neural network with sin hidden layer containing 10 neurons and logistic output activation, reg with L_2 penalty term for $\lambda > 0$. Varying λ affects smoothness of the c boundary and magnitude of network weights:



REGULARIZATION IN NONLINEAR MODELS

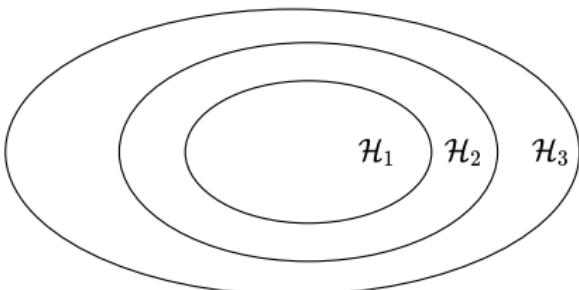
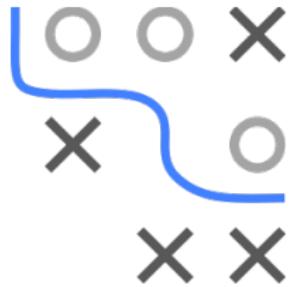
The prevention of overfitting can also be seen in CV. Same setting as before, but each λ is evaluated with repeated CV (10 folds, 5 repeats).



We see the typical U-shape with the sweet spot between overfit (LHS, low λ) and underfitting (RHS, high λ) in the middle.

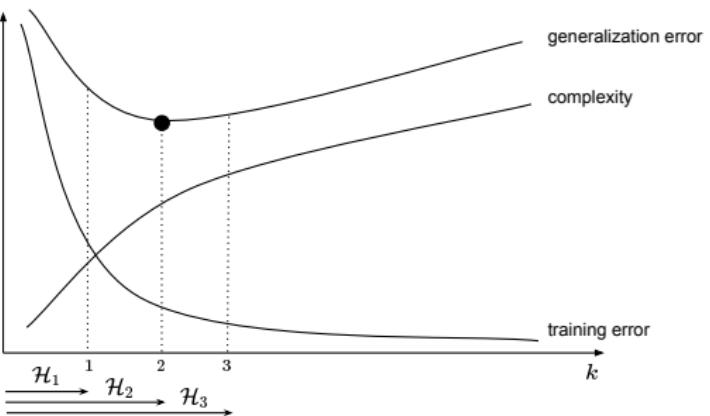
STRUCTURAL RISK MINIMIZATION

- Thus far, we only considered adding a complexity penalty to empirical risk minimization.
- Instead, structural risk minimization (SRM) assumes that the hypothesis space \mathcal{H} can be decomposed into increasingly complex hypotheses (size or capacity): $\mathcal{H} = \cup_{k \geq 1} \mathcal{H}_k$.
- Complexity parameters can be the, e.g. the degree of poly in linear models or the size of hidden layers in neural netw



STRUCTURAL RISK MINIMIZATION

- SRM chooses the smallest k such that the optimal model found by ERM or RRM cannot significantly be outperformed by a model from a \mathcal{H}_m with $m > k$.
- By this, the simplest model can be chosen, which minimizes generalization bound.
- One challenge might be choosing an adequate complexity measure, as for some models, multiple complexity measur



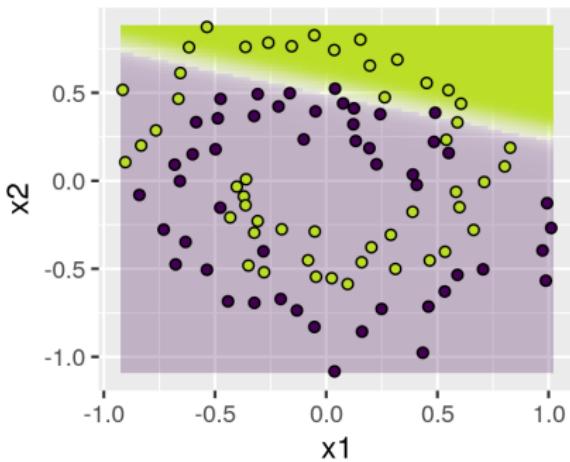
STRUCTURAL RISK MINIMIZATION

Setting: Classification for the spirals data. Neural network with sin hidden layer containing k neurons and logistic output activation, L2 re with $\lambda = 0.001$. So here SRM and RRM are both used. Varying the s hidden layer affects smoothness of the decision boundary:



size of hidden layer = 1

Prediction



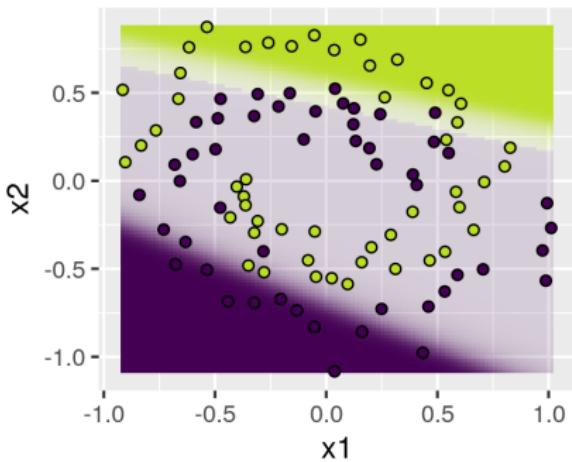
STRUCTURAL RISK MINIMIZATION

Setting: Classification for the spirals data. Neural network with sin hidden layer containing k neurons and logistic output activation, L2 re with $\lambda = 0.001$. So here SRM and RRM are both used. Varying the s hidden layer affects smoothness of the decision boundary:



size of hidden layer = 2

Prediction



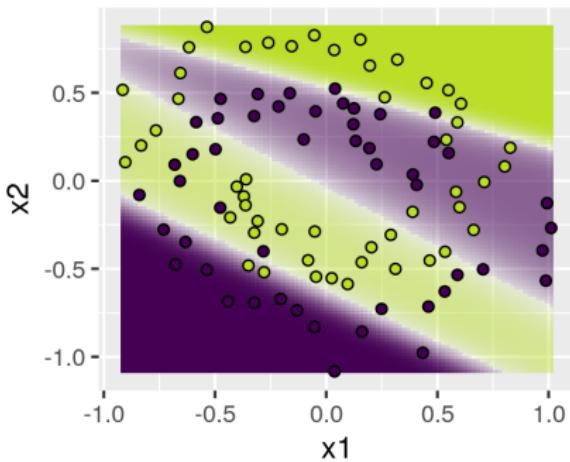
STRUCTURAL RISK MINIMIZATION

Setting: Classification for the spirals data. Neural network with sin hidden layer containing k neurons and logistic output activation, L2 re with $\lambda = 0.001$. So here SRM and RRM are both used. Varying the size of hidden layer affects smoothness of the decision boundary:



size of hidden layer = 3

Prediction



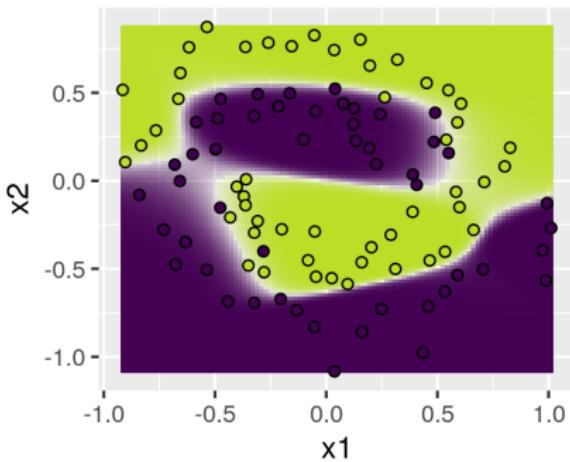
STRUCTURAL RISK MINIMIZATION

Setting: Classification for the spirals data. Neural network with sin hidden layer containing k neurons and logistic output activation, L2 re with $\lambda = 0.001$. So here SRM and RRM are both used. Varying the s hidden layer affects smoothness of the decision boundary:



size of hidden layer = 5

Prediction



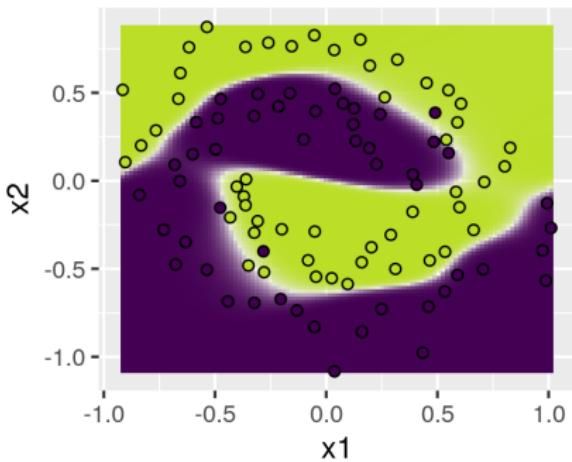
STRUCTURAL RISK MINIMIZATION

Setting: Classification for the spirals data. Neural network with sin hidden layer containing k neurons and logistic output activation, L2 re with $\lambda = 0.001$. So here SRM and RRM are both used. Varying the size of hidden layer affects smoothness of the decision boundary:



size of hidden layer = 10

Prediction



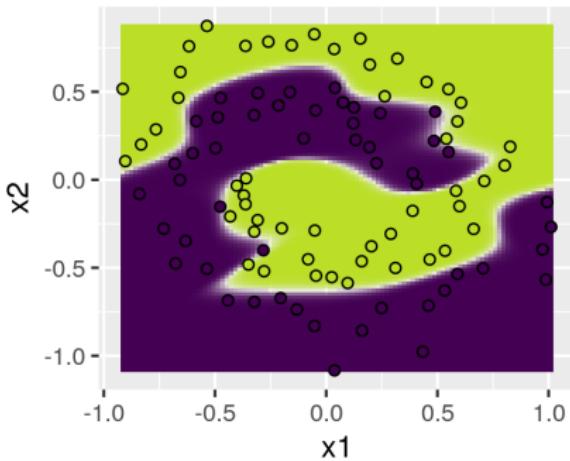
STRUCTURAL RISK MINIMIZATION

Setting: Classification for the spirals data. Neural network with sin hidden layer containing k neurons and logistic output activation, L2 re with $\lambda = 0.001$. So here SRM and RRM are both used. Varying the size of hidden layer affects smoothness of the decision boundary:



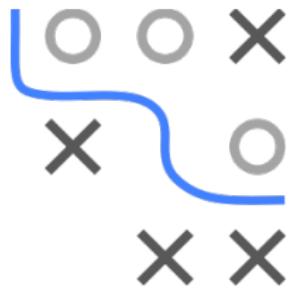
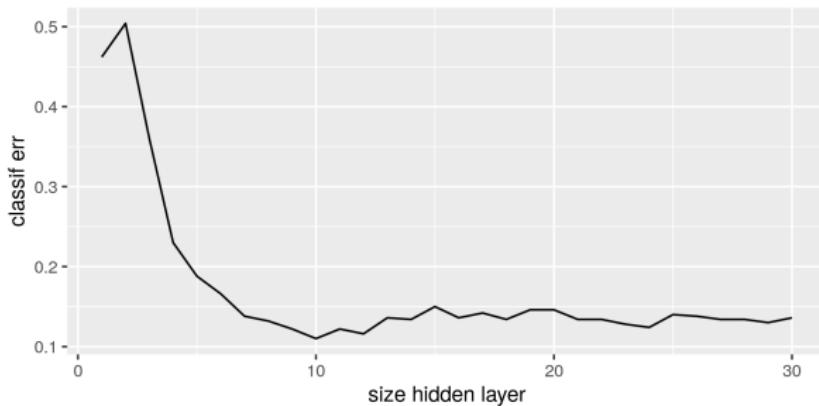
size of hidden layer = 100

Prediction



STRUCTURAL RISK MINIMIZATION

Again, complexity vs CV score.

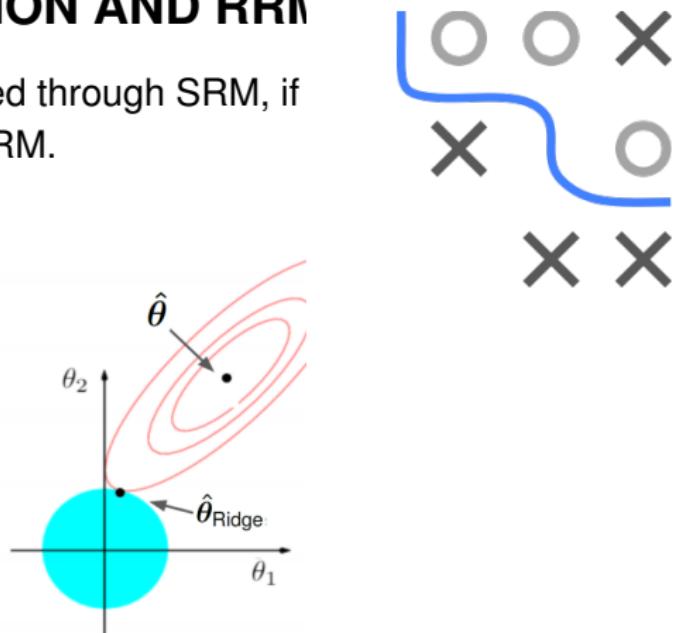


A minimal model with good generalization seems to have ca. 6-hidden neurons.

STRUCTURAL RISK MINIMIZATION AND RRM

Note that normal RRM can also be interpreted through SRM, if rewrite the penalized ERM as constrained ERM.

$$\begin{aligned} \min_{\theta} \quad & \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)} | \theta)) \\ \text{s.t.} \quad & \|\theta\|_2^2 \leq t \end{aligned}$$



We can interpret going through λ from large to small as through small to large. This constructs a series of ERM problems with hypothesis spaces \mathcal{H}_λ , where we constrain the norm of θ to un of growing size.

RRM VS. BAYES

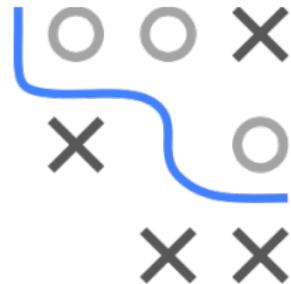
We already created a link between max. likelihood estimation and RRM.

Now we will generalize this for RRM.

Assume we have a parameterized distribution $p(y|\theta, \mathbf{x})$ for our data y given features \mathbf{x} , and a prior $q(\theta)$ over our parameter space, all in the Bayesian framework.

From the Bayes theorem we know:

$$p(\theta|\mathbf{x}, y) = \frac{p(y|\theta, \mathbf{x})q(\theta)}{p(y|\mathbf{x})} \propto p(y|\theta, \mathbf{x})q(\theta)$$



RRM VS. BAYES

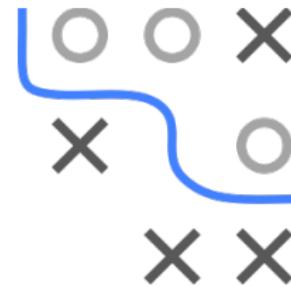
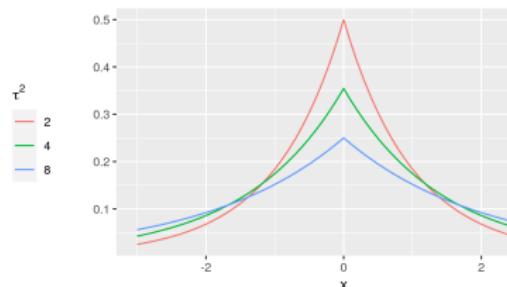
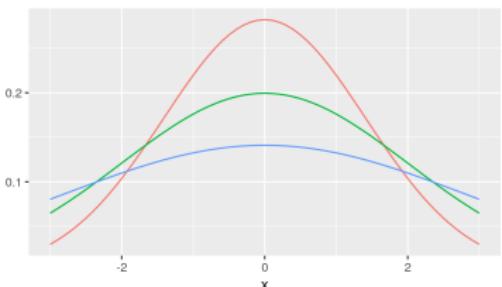
The maximum a posteriori (MAP) estimator of θ is now the min

$$-\log p(y | \theta, \mathbf{x}) - \log q(\theta).$$

- Again, we identify the loss $L(y, f(\mathbf{x} | \theta))$ with $-\log(p(y|\theta,$
- If $q(\theta)$ is constant (i.e., we used a uniform, non-informative prior), then 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: $\theta_j \sim \text{Laplace}(\mu, b)$ has density $\frac{1}{2b} \exp(-\frac{|\mu-x|}{b})$, with scale parameter μ and variance $2b^2$.
- In both cases, regularization strength increases as the variance of the prior decreases: a prior probability mass more narrowly concentrated around 0 encourages shrinkage.

EXAMPLE: BAYESIAN L2 REGULARIZATION

We can easily see the equivalence of L_2 regularization and a Gaussian prior.

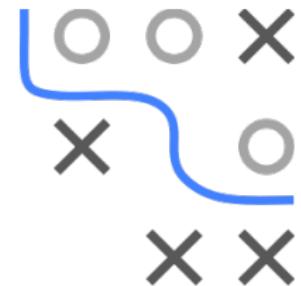
- We define a Gaussian prior with uncorrelated components for θ

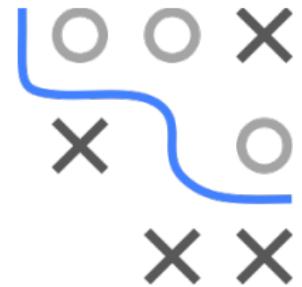
$$q(\theta) = \mathcal{N}_d(\mathbf{0}, \text{diag}(\tau^2)) = \prod_{j=1}^d \mathcal{N}(0, \tau^2) = (2\pi\tau^2)^{-\frac{d}{2}} \exp\left(-\frac{1}{2\tau^2} \sum_j \theta_j^2\right)$$

- With this, the MAP estimator becomes

$$\begin{aligned}\hat{\theta}^{\text{MAP}} &= \arg \min_{\theta} (-\log p(y | \theta, \mathbf{x}) - \log q(\theta)) \\ &= \arg \min_{\theta} \left(-\log p(y | \theta, \mathbf{x}) + \frac{d}{2} \log(2\pi\tau^2) + \frac{1}{2\tau^2} \sum_{j=1}^d \theta_j^2 \right) \\ &= \arg \min_{\theta} \left(-\log p(y | \theta, \mathbf{x}) + \frac{1}{2\tau^2} \|\theta\|_2^2 \right).\end{aligned}$$

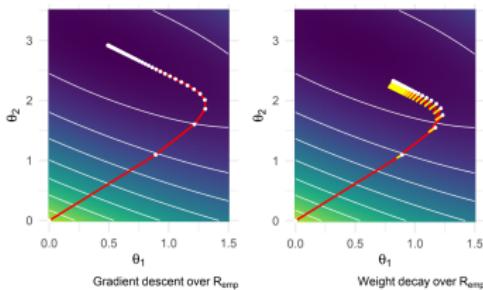
- We see how the inverse variance (precision) $1/\tau^2$ controls shrinkage.





Introduction to Machine Learning

Geometric Analysis of L2 Regularization and Weight Decay



Learning goals

- Have a geometric understanding of $L2$ regularization
- Understand why $L2$ regularization in combination with gradient descent is called weight decay

WEIGHT DECAY VS. L2 REGULARIZATION

Let us optimize the L_2 -regularized risk of a model $f(\mathbf{x} \mid \theta)$

$$\min_{\theta} \mathcal{R}_{\text{reg}}(\theta) = \min_{\theta} \mathcal{R}_{\text{emp}}(\theta) + \frac{\lambda}{2} \|\theta\|_2^2$$

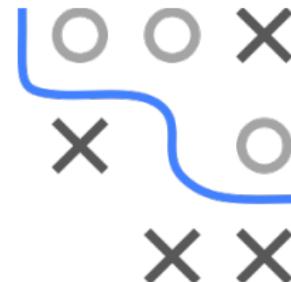
by gradient descent. The gradient is

$$\nabla_{\theta} \mathcal{R}_{\text{req}}(\theta) = \nabla_{\theta} \mathcal{R}_{\text{emp}}(\theta) + \lambda \theta.$$

We iteratively update θ by step size α times the negative gradient.

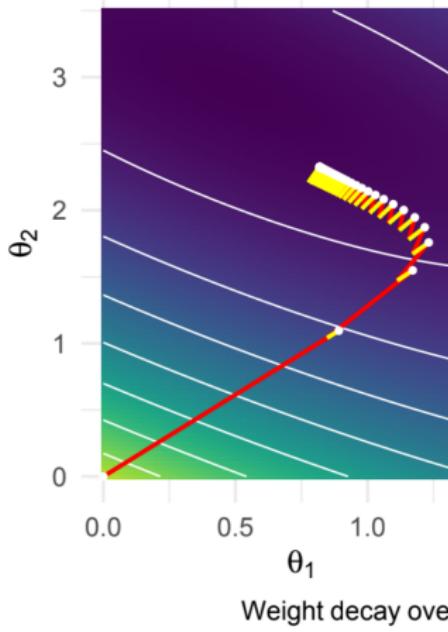
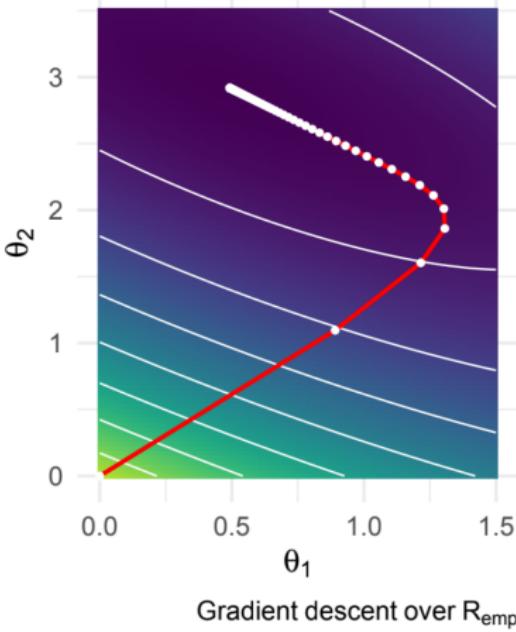
$$\begin{aligned}\boldsymbol{\theta}^{[\text{new}]} &= \boldsymbol{\theta}^{[\text{old}]} - \alpha \left(\nabla_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[\text{old}]}) + \lambda \boldsymbol{\theta}^{[\text{old}]} \right) \\ &= \boldsymbol{\theta}^{[\text{old}]} (1 - \alpha \lambda) - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[\text{old}]}).\end{aligned}$$

The term $\lambda\theta^{[old]}$ causes the parameter (**weight**) to **decay** in proportion to its size. This is a very well-known technique in deep learning and is simply *L2 regularization* in disguise.



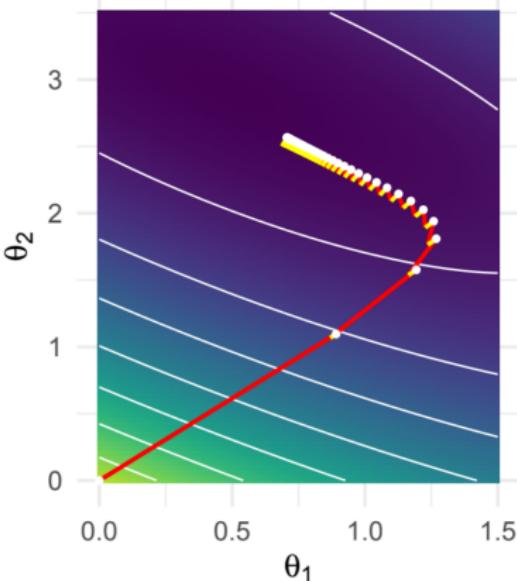
WEIGHT DECAY VS. L2 REGULARIZATION

When we use weight decay, we follow the steepest slope of \mathcal{R}_{el} gradient descent, but in every step, we are pulled back to the or

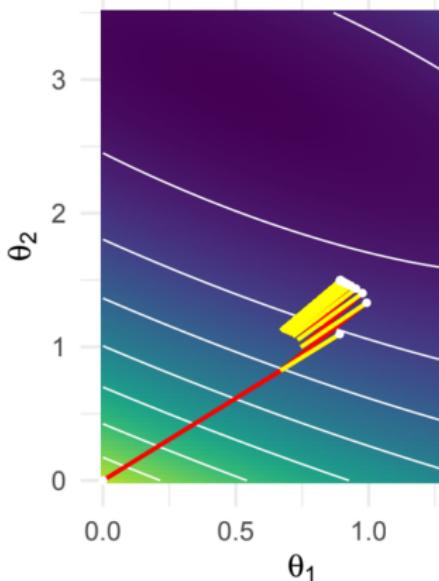


WEIGHT DECAY VS. L2 REGULARIZATION

How strongly we are pulled back to the origin for a fixed stepsize depends only on λ (as long as the procedure converges):



Weight decay (small λ) over R_{emp}



Weight decay (large λ) over R_{emp}



WEIGHT DECAY VS. L2 REGULARIZATION

Weight decay can be interpreted **geometrically**.

Let's use a quadratic Taylor approximation of the unregularized objective $\mathcal{R}_{\text{emp}}(\theta)$ in the neighborhood of 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 matrix of $\mathcal{R}_{\text{emp}}(\theta)$ evaluated at $\hat{\theta}$.



- The first-order term is 0 in the expression above because the gradient is 0 at the minimizer.
- \mathbf{H} is positive semidefinite.

WEIGHT DECAY VS. L2 REGULARIZATION

The minimum of $\tilde{\mathcal{R}}_{\text{emp}}(\theta)$ occurs where $\nabla_{\theta} \tilde{\mathcal{R}}_{\text{emp}}(\theta) = \mathbf{H}(\theta -$



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 $\hat{\theta}_{\text{Ridge}}$:

$$\nabla_{\theta} \tilde{\mathcal{R}}_{\text{reg}}(\theta) = 0,$$

$$\lambda\theta + \mathbf{H}(\theta - \hat{\theta}) = 0,$$

$$(\mathbf{H} + \lambda\mathbf{I})\theta = \mathbf{H}\hat{\theta},$$

$$\hat{\theta}_{\text{Ridge}} = (\mathbf{H} + \lambda\mathbf{I})^{-1} \mathbf{H}\hat{\theta},$$

This give us a formula to see how the minimizer of the *L2*-regularized version is a transformation of the minimizer of the unpenalized

WEIGHT DECAY VS. L2 REGULARIZATION

- As λ approaches 0, the regularized solution $\hat{\theta}_{\text{Ridge}}$ approaches the unregularized solution $\hat{\theta}$. 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:

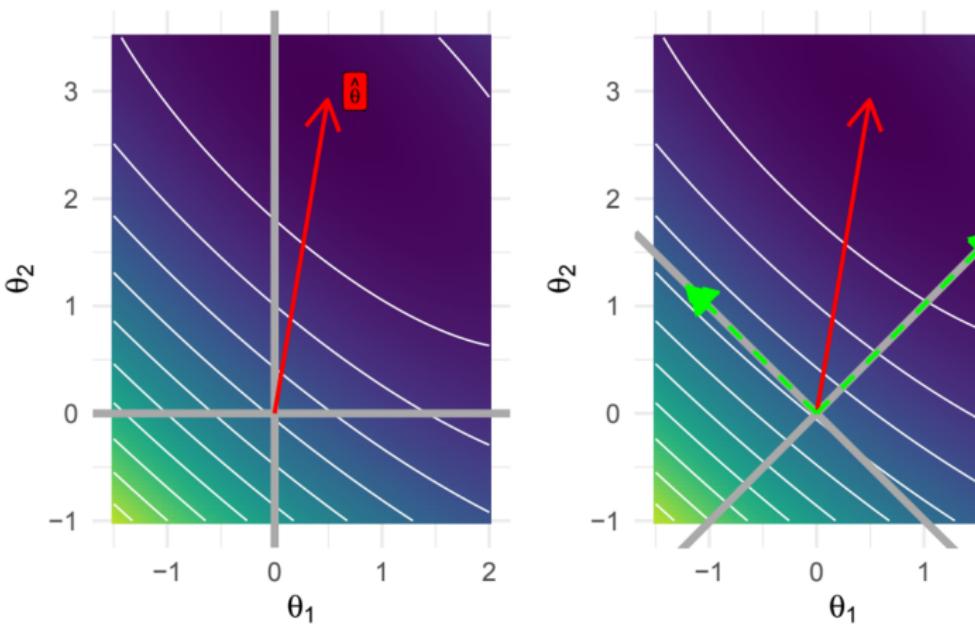
$$\begin{aligned}\hat{\theta}_{\text{Ridge}} &= (Q\Sigma Q^T + \lambda I)^{-1} Q\Sigma Q^T \hat{\theta} \\ &= [Q(\Sigma + \lambda I)Q^T]^{-1} Q\Sigma Q^T \hat{\theta} \\ &= Q(\Sigma + \lambda I)^{-1} \Sigma Q^T \hat{\theta}\end{aligned}$$

- Therefore, weight decay rescales $\hat{\theta}$ along the axes defined by the eigenvectors of H . The component of $\hat{\theta}$ that is aligned with the eigenvector Q_j is rescaled by a factor of $\frac{\sigma_j}{\sigma_j + \lambda}$, where σ_j is the corresponding eigenvalue.



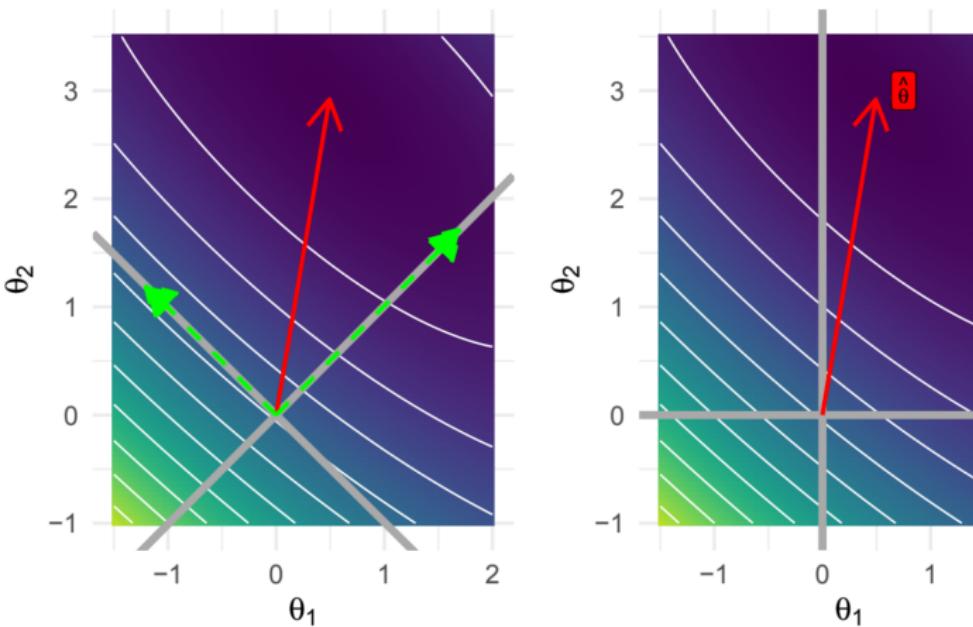
WEIGHT DECAY VS. L2 REGULARIZATION

Firstly, $\hat{\theta}$ is rotated by \mathbf{Q}^T , which we can interpret as a projection on the rotated coordinate system defined by the principal directions \mathbf{H} :



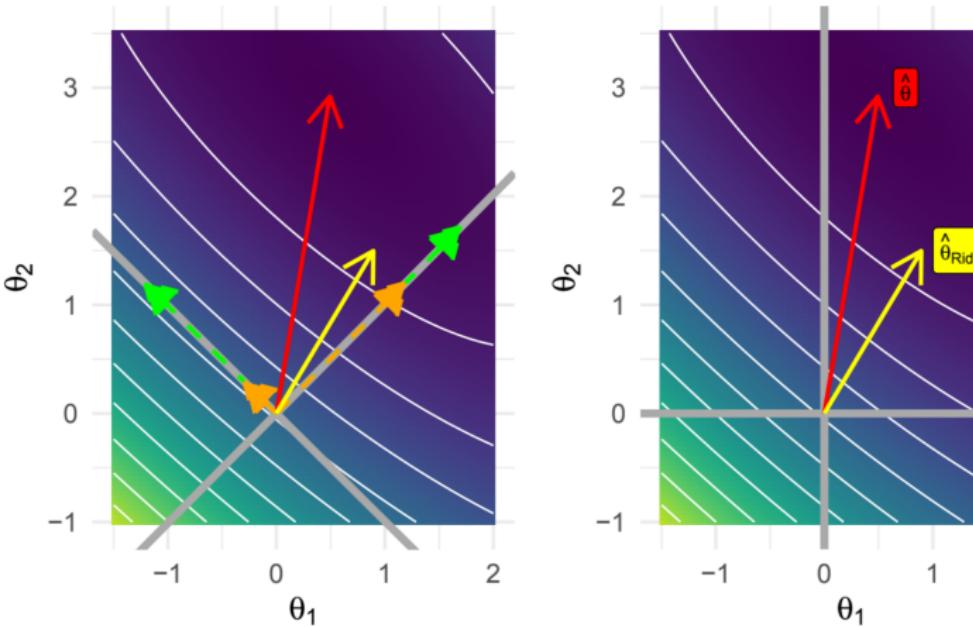
WEIGHT DECAY VS. L2 REGULARIZATION

Since, for $\lambda = 0$, the transformation matrix $(\Sigma + \lambda I)^{-1}\Sigma = \Sigma^-$ we simply arrive at $\hat{\theta}$ again after projecting back.



WEIGHT DECAY VS. L2 REGULARIZATION

If $\lambda > 0$, the component projected on the j -th axis gets rescaled $\frac{\sigma_j}{\sigma_j + \lambda}$ before $\hat{\theta}_{\text{Ridge}}$ is rotated back.

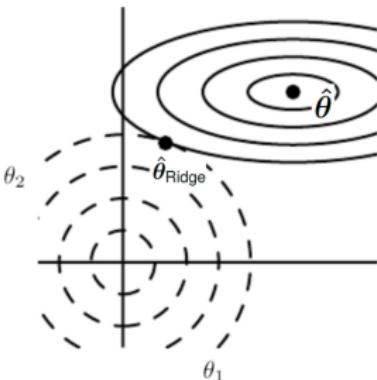


WEIGHT DECAY VS. L2 REGULARIZATION

- Along directions where the eigenvalues of H are relatively large, for example, where $\sigma_j \gg \lambda$, the effect of regularization is quite small.
- On the other hand, components with $\sigma_j \ll \lambda$ will be shrunk towards zero, and thus have nearly zero magnitude.
- In other words, only directions along which the parameters contribute significantly to reducing the objective function are preserved relatively intact.
- In the other directions, a small eigenvalue of the Hessian means that moving in this direction will not significantly increase the value of the gradient. For such unimportant directions, the corresponding components of θ are decayed away.



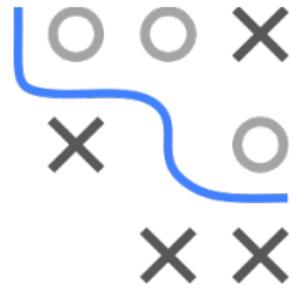
WEIGHT DECAY VS. L2 REGULARIZATION



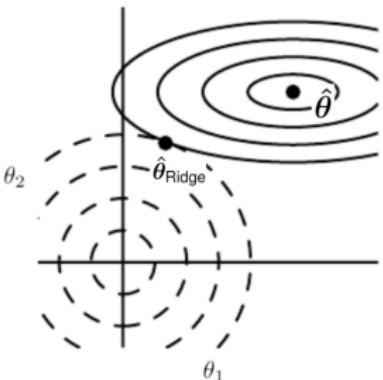
Credit: Goodfellow et al. (2016), ch. 7

Figure: The solid ellipses represent the contours of the unregularized objective function; the dashed circles represent the contours of the L_2 penalty. At $\hat{\theta}_{\text{Ridge}}$, the two objectives reach an equilibrium.

In the first dimension, the eigenvalue of the Hessian of $\mathcal{R}_{\text{emp}}(\theta)$ is small; the objective function does not increase much when moving horizontally from $\hat{\theta}$. Therefore, the regularizer has a strong effect on this axis and pulled close to zero.



WEIGHT DECAY VS. L2 REGULARIZATION



Credit: Goodfellow et al. (2016), ch. 7

Figure: The solid ellipses represent the contours of the unregularized objective; the dashed circles represent the contours of the L^2 penalty. At $\hat{\theta}_{\text{Ridge}}$, the two objectives reach an equilibrium.

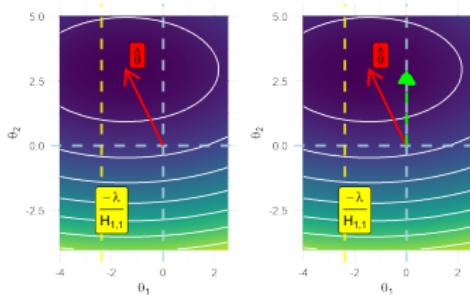
In the second dimension, the corresponding eigenvalue is large indicating high curvature. The objective function is very sensitive to movement along this axis, and, as a result, the position of θ_2 is less affected by the regularization.





Introduction to Machine Learning

Geometric Analysis of L1-regularization



Learning goals

- Have a geometric understanding of L1-regularization
- Understand geometrically how L1-regularization induces sparsity

L1-REGULARIZATION

- The L1-regularized risk of a model $f(\mathbf{x} | \boldsymbol{\theta})$ is

$$\min_{\boldsymbol{\theta}} \mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) = \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1$$

and the (sub-)gradient is:

$$\nabla_{\boldsymbol{\theta}} \mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) = \lambda \text{sign}(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$$

- Note that, unlike in the case of L2, the contribution of the L penalty to the gradient doesn't scale linearly with each θ_j .
- Let us now make (again) a quadratic Taylor approximation $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$ around its minimizer $\hat{\boldsymbol{\theta}}$. To get a clean algebraic expression, we further assume the Hessian of $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$ is diagonal, i.e. $\mathbf{H} = \text{diag}([H_{1,1}, \dots, H_{d,d}])$, where each $H_{j,j}$:
- This assumption holds, for example, if the input features for a linear regression task have been decorrelated using PCA.



L1-REGULARIZATION

- If we plug this approximation into $\mathcal{R}_{\text{reg}}(\theta)$, the result nicely decomposes into a sum over the parameters:

$$\tilde{\mathcal{R}}_{\text{reg}}(\theta) = \mathcal{R}_{\text{emp}}(\hat{\theta}) + \sum_j \left[\frac{1}{2} H_{j,j} (\theta_j - \hat{\theta}_j)^2 \right] + \sum_j \lambda$$



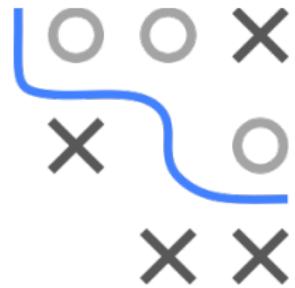
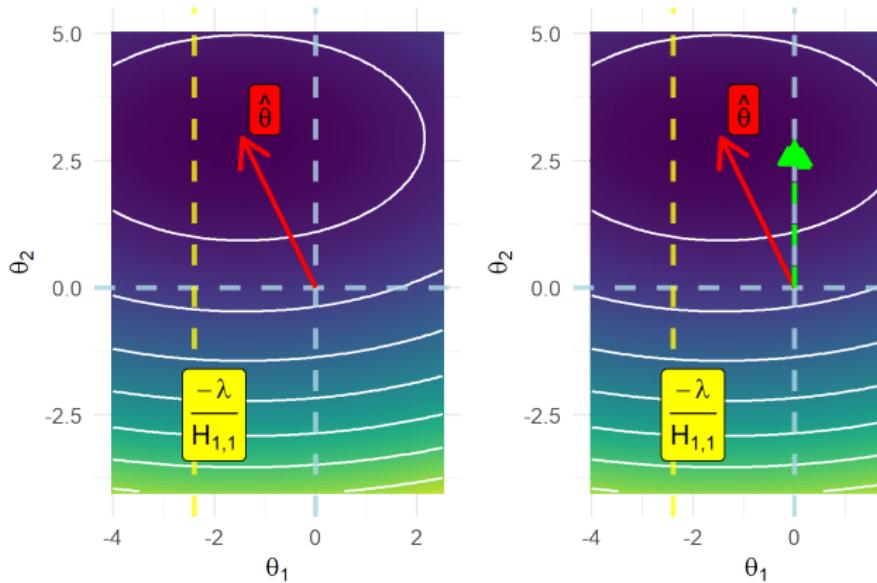
- We can minimize analytically:

$$\begin{aligned}\hat{\theta}_{\text{Lasso},j} &= \text{sign}(\hat{\theta}_j) \max \left\{ |\hat{\theta}_j| - \frac{\lambda}{H_{j,j}}, 0 \right\} \\ &= \begin{cases} \hat{\theta}_j + \frac{\lambda}{H_{j,j}} & , \text{if } \hat{\theta}_j < -\frac{\lambda}{H_{j,j}} \\ 0 & , \text{if } \hat{\theta}_j \in [-\frac{\lambda}{H_{j,j}}, \frac{\lambda}{H_{j,j}}] \\ \hat{\theta}_j - \frac{\lambda}{H_{j,j}} & , \text{if } \hat{\theta}_j > \frac{\lambda}{H_{j,j}} \end{cases}\end{aligned}$$

- If $H_{j,j} = 0$ exactly, $\hat{\theta}_{\text{Lasso},j} = 0$.

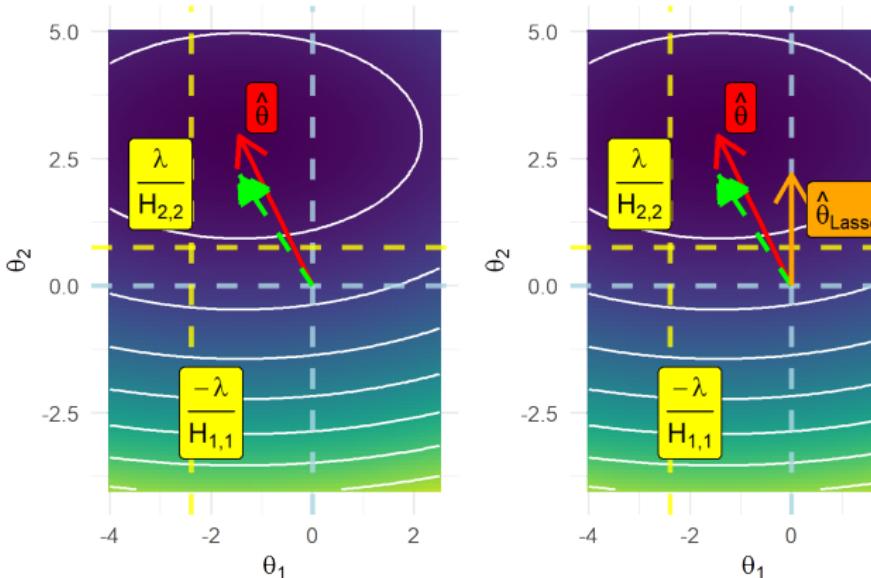
L1-REGULARIZATION

- 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 regularized risk) is 0 because the contribution of $\mathcal{R}_{\text{emp}}(\theta)$ / $\mathcal{R}_{\text{reg}}(\theta)$ is overwhelmed by the L1 penalty, which forces it to be at the boundary.



L1-REGULARIZATION

- 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 value of θ_j toward 0 by the amount $\frac{\lambda}{H_{j,j}}$.

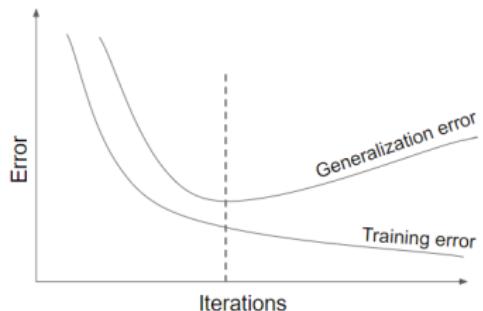


- Therefore, the L1 penalty induces sparsity in the parameters

Introduction to Machine Learning



Early Stopping



Learning goals

- Know how early stopping work
- Understand how early stopping acts as a regularizer

EARLY STOPPING

- When training with an iterative optimizer such as SGD, it is commonly the case that, after a certain number of iterations, generalization error begins to increase even though training error continues to decrease.
- Early stopping** refers to stopping the algorithm early before generalization error increases.

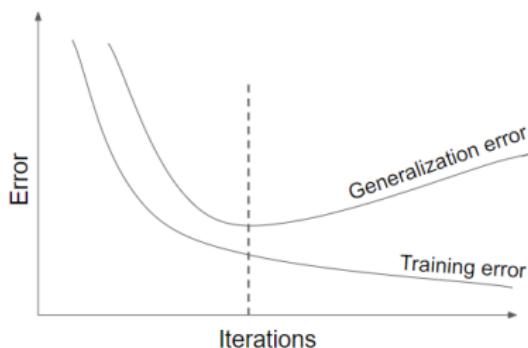
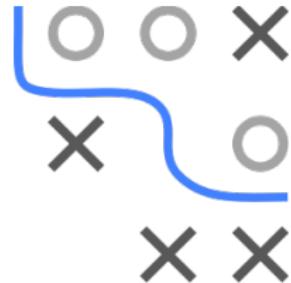


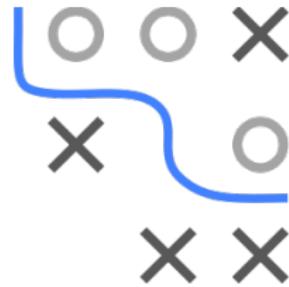
Figure: After a certain number of iterations, the algorithm begins to

EARLY STOPPING

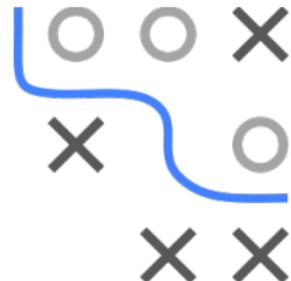
How early stopping works:

- ❶ Split training data $\mathcal{D}_{\text{train}}$ into $\mathcal{D}_{\text{subtrain}}$ and \mathcal{D}_{val} (e.g. with a 2:1).
- ❷ Train on $\mathcal{D}_{\text{subtrain}}$ and evaluate model using the validation set.
- ❸ Stop training when validation error stops decreasing (after *“patience”* steps).
- ❹ Use parameters of the previous step for the actual model.

More sophisticated forms also apply cross-validation.



EARLY STOPPING



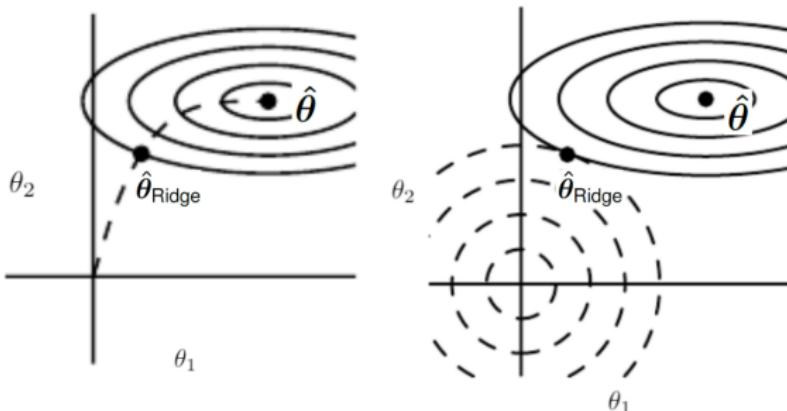
Strengths	Weaknesses
Effective and simple	Periodical evaluation of validation
Applicable to almost any model without adjustment	Temporary copy of θ (we have to retrain the whole model each time validation error improves)
Combinable with other regularization methods	Less data for training \rightarrow includes validation afterwards

- Relation between optimal early-stopping iteration T_{stop} and weight-decay penalization parameter λ for step-size α (see Goodfellow et al. (2016) page 251-252 for proof):

$$T_{\text{stop}} \approx \frac{1}{\alpha \lambda} \Leftrightarrow \lambda \approx \frac{1}{T_{\text{stop}} \alpha}$$

- Small λ (low penalization) \Rightarrow high T_{stop} (complex model / many updates).

EARLY STOPPING



Credit: Goodfellow et al. (2016)



Figure: An illustration of the effect of early stopping. *Left:* The solid contour lines indicate the contours of the negative log-likelihood. The dashed line indicates the trajectory taken by SGD beginning from the origin. Rather than stopping at the point that minimizes the risk, early stopping results in the trajectory stopping at an intermediate point $\hat{\theta}_{\text{Ridge}}$. *Right:* An illustration of the effect of L_2 regularization for comparison. The dashed circles indicate the contours of the L_2 penalty which causes the minimum of the total cost to lie closer to the origin than the minimum of the unregularized cost.

INTRODUCTION TO MACHINE LEARNING

Advanced Risk Minimization

Multiclass Classification

Information Theory

Curse of Dimensionality

Regularization

Linear Support Vector Machine

Nonlinear Support Vector Machine

Boosting

Feature Selection

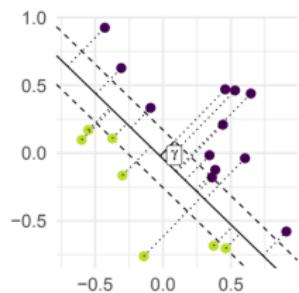




Introduction to Machine Learning

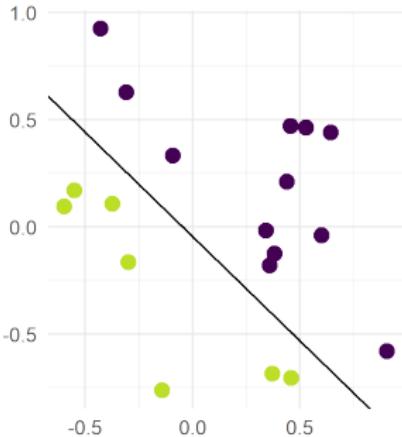
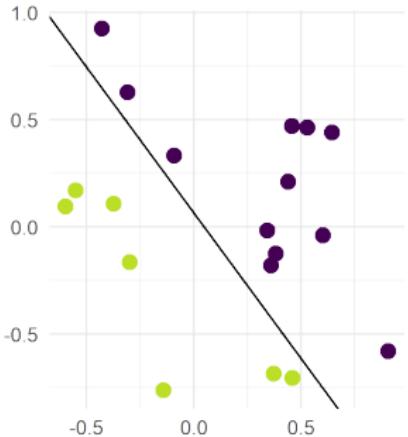
Linear Hard Margin SVM

Learning goals



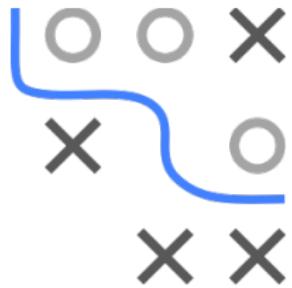
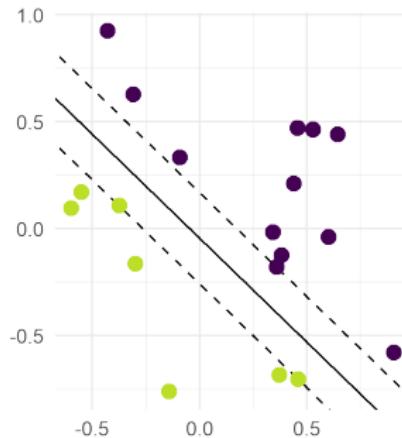
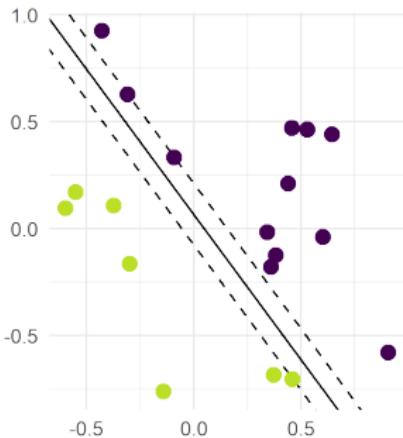
- Know that the hard-margin SV maximizes the margin between data points and hyperplane
- Know that this is a quadratic program
- Know that support vectors are the data points closest to the separating hyperplane

LINEAR CLASSIFIERS



- We want study how to build a binary, linear classifier from : geometrical principles.
- Which of these two classifiers is “better”?

LINEAR CLASSIFIERS



- We want study how to build a binary, linear classifier from : 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 $f(\mathbf{x}) = \theta^\top \mathbf{x} + \theta_0$, such that all +--observations are in the positive halfspace

$$\{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) > 0\}$$

and all --observations are in the negative halfspace

$$\{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) < 0\}.$$

- For a linear separating hyperplane, we have

$$y^{(i)} \underbrace{\left(\theta^\top \mathbf{x}^{(i)} + \theta_0 \right)}_{=f(\mathbf{x}^{(i)})} > 0 \quad \forall i \in \{1, 2, \dots, n\}.$$

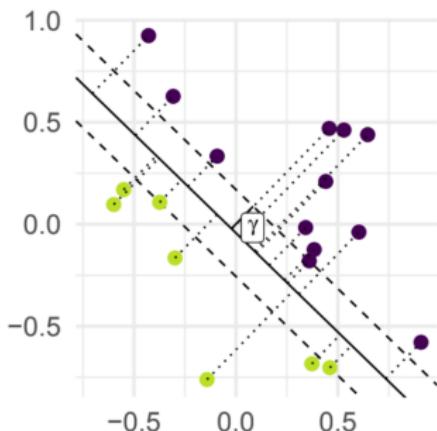


SUPPORT VECTOR MACHINES: GEOMETRY

-

$$d(f, \mathbf{x}^{(i)}) = \frac{y^{(i)} f(\mathbf{x}^{(i)})}{\|\theta\|} = y^{(i)} \frac{\theta^T \mathbf{x}^{(i)} + \theta_0}{\|\theta\|}$$

- computes the (signed) distance to the separating hyperplane
 $f(\mathbf{x}) = 0$, positive for correct classifications, negative for incorrect classifications
- This expression becomes negative for misclassified points

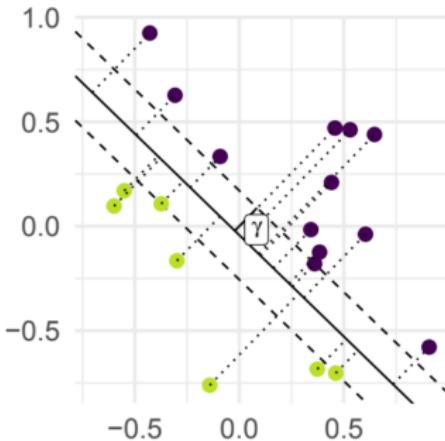


SUPPORT VECTOR MACHINES: GEOMETRY

- The distance of f to the whole dataset \mathcal{D} is the smallest distance to any point.

$$\gamma = \min_i \left\{ d(f, \mathbf{x}^{(i)}) \right\}.$$

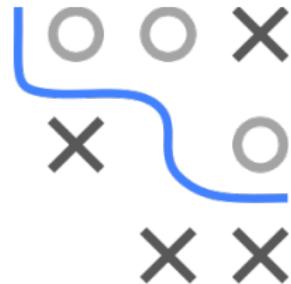
- This represents the “safety margin”, it is positive if f separates the classes. we want to maximize it.



MAXIMUM MARGIN SEPARATION

We formulate the desired property of a large “safety margin” as 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 given by $f (= \theta^T \mathbf{x} + \theta_0) = 0$.
- Our objective is to maximize the “safety” distance.

MAXIMUM MARGIN SEPARATION

We reformulate the problem:



$$\max_{\theta, \theta_0} \gamma$$

$$\text{s.t. } \frac{y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0)}{\|\theta\|} \geq \gamma \quad \forall i \in \{1, \dots, n\}.$$

- The inequality is rearranged by multiplying both sides with

$$\max_{\theta, \theta_0} \gamma$$

$$\text{s.t. } y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq \|\theta\| \gamma \quad \forall i \in \{1, \dots, n\}$$

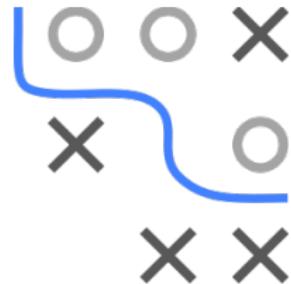
MAXIMUM MARGIN SEPARATION

- 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 – 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 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:

$$\begin{aligned} \min_{\theta, \theta_0} \quad & \frac{1}{2} \|\theta\|^2 \\ \text{s.t.} \quad & y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq 1 \quad \forall i \in \{1, \dots, n\} \end{aligned}$$



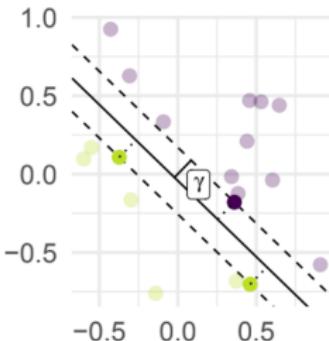
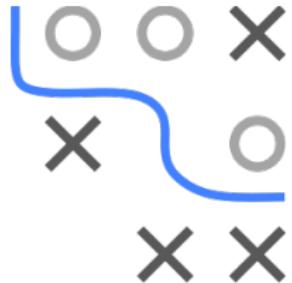
This turns out to be a **convex optimization problem** – particularly a **quadratic program**: The objective function is quadratic, and the constraints are linear inequalities.

This is called the **primal** problem. We will later show that we can derive a dual problem from it.

We will call this the **linear hard-margin SVM**.

SUPPORT VECTORS

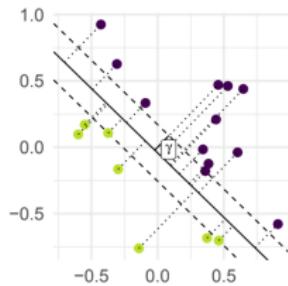
- There exist instances $(\mathbf{x}^{(i)}, y^{(i)})$ with minimal margin $y^{(i)} f(\mathbf{x}^{(i)}) = 1$, fulfilling the inequality constraints with equality.
- They are called **support vectors (SVs)**. They are located at a distance of $\gamma = 1/\|\theta\|$ from the separating hyperplane.
- It 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 via 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

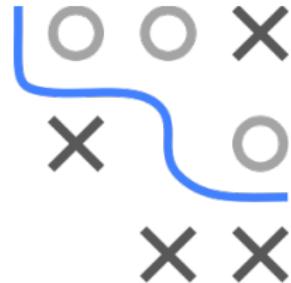
$$\begin{aligned} L(\theta, \theta_0, \alpha) = \quad & \frac{1}{2} \|\theta\|^2 - \sum_{i=1}^n \alpha_i [y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) - 1] \\ \text{s.t.} \quad & \alpha_i \geq 0 \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$

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 decision variables α , as constraints turned into variables and vice versa. Now each data point has an associated non-negative weight.



$$\begin{aligned} L(\theta, \theta_0, \alpha) = & \frac{1}{2} \|\theta\|^2 - \sum_{i=1}^n \alpha_i \left[y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) - 1 \right] \\ \text{s.t.} \quad & \alpha_i \geq 0 \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$

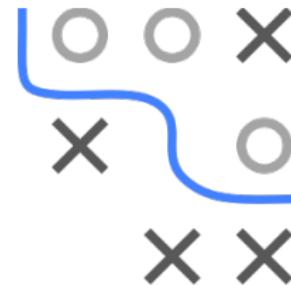
We find the stationary point of $L(\theta, \theta_0, \alpha)$ w.r.t. θ, θ_0 and obtain

$$\begin{aligned} \theta &= \sum_{i=1}^n \alpha_i y^{(i)} \mathbf{x}^{(i)}, \\ 0 &= \sum_{i=1}^n \alpha_i y^{(i)} \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$

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 (duality).

Under these conditions, and if we solve the dual problem and obtain α , we know that θ is a linear combination of our data points:

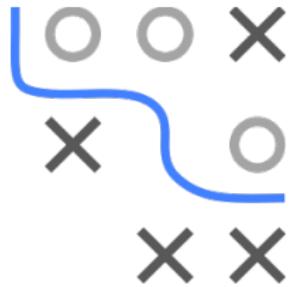
$$\hat{\theta} = \sum_{i=1}^n \hat{\alpha}_i y^{(i)} \mathbf{x}^{(i)}$$

Complementary slackness means:

$$\hat{\alpha}_i [y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) - 1] = 0 \quad \forall i \in \{1, \dots, n\}.$$



HARD MARGIN SVM DUAL



$$\hat{\theta} = \sum_{i=1}^n \hat{\alpha}_i y^{(i)} \mathbf{x}^{(i)}$$

$$\hat{\alpha}_i [y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) - 1] = 0 \quad \forall i \in \{1, \dots, n\}.$$

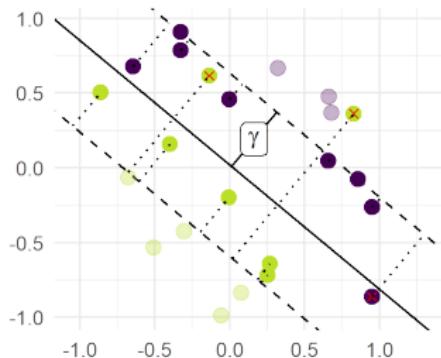
- So either $\hat{\alpha}_i = 0$, and is not active in the linear combination, or i then $y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) = 1$, and $(\mathbf{x}^{(i)}, y^{(i)})$ has minimal margin support vector!
- We see that we can directly extract the support vectors from the variables and the θ solution only depends on them.
- We can reconstruct the bias term θ_0 from any support vector:

$$\theta_0 = y^{(i)} - \langle \theta, \mathbf{x}^{(i)} \rangle.$$

Introduction to Machine Learning



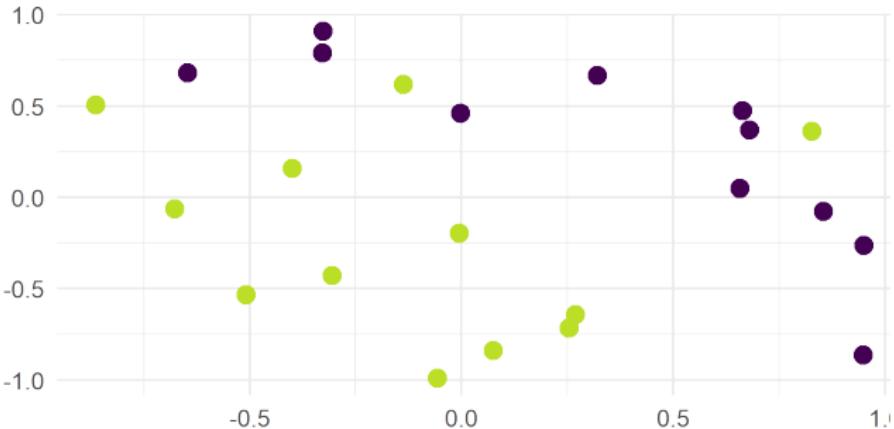
Soft-Margin SVM



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



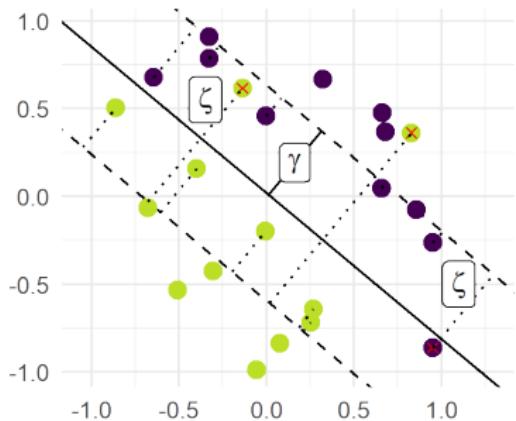
- Assume that dataset \mathcal{D} is not linearly separable.
- Margin maximization becomes meaningless because the hard-margin SVM optimization problem has contradictory constraints and thus an empty **feasible region**.

MARGIN VIOLATIONS

- We still want a large margin for most of the examples.
- We allow violations of the margin constraints via slack vars

$$y^{(i)} \left(\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0 \right) \geq 1 - \zeta^{(i)}$$

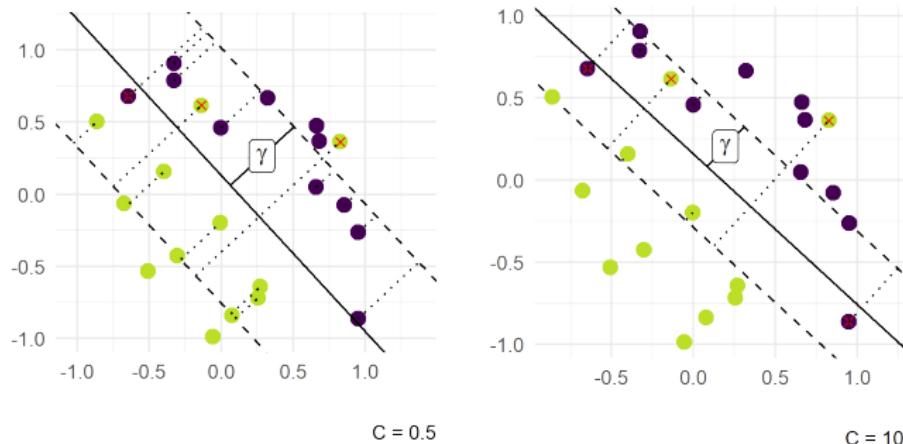
- Even for separable data, a decision boundary with a few violations and a large average margin may be preferable to one with many violations and a small average margin.



We assume $\gamma =$
further complicates
sentation.

MARGIN VIOLATIONS

- Now we have two distinct and contradictory goals:
 - Maximize the margin.
 - Minimize margin violations.
- Let's minimize a weighted sum of them: $\frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n$
- Constant $C > 0$ controls the relative importance of the two



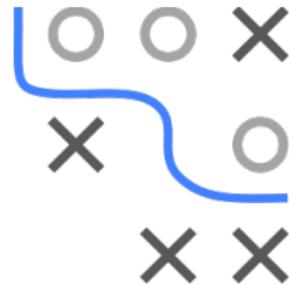
SOFT-MARGIN SVM

The linear **soft-margin** SVM is the convex quadratic program:

$$\min_{\theta, \theta_0, \zeta^{(i)}} \frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n \zeta^{(i)}$$

$$\text{s.t. } y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq 1 - \zeta^{(i)} \quad \forall i \in \{1, \dots,$$

$$\text{and } \zeta^{(i)} \geq 0 \quad \forall i \in \{1, \dots, n\}.$$



This is called “soft-margin” SVM because the “hard” margin constraint is replaced with a “softened” constraint that can be violated by an amount $\zeta^{(i)}$.

SOFT-MARGIN SVM DUAL FORM

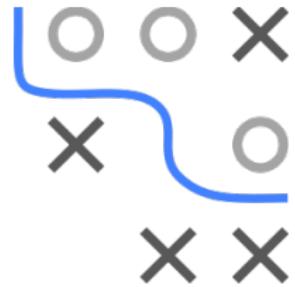
Can be derived exactly as for the hard margin case.

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \\ & \sum_{i=1}^n \alpha_i y^{(i)} = 0, \end{aligned}$$

or, in matrix notation:

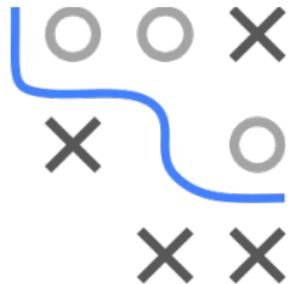
$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & \mathbf{1}^T \alpha - \frac{1}{2} \alpha^T \text{diag}(\mathbf{y}) \mathbf{K} \text{diag}(\mathbf{y}) \alpha \\ \text{s.t.} \quad & \alpha^T \mathbf{y} = 0, \\ & 0 \leq \alpha \leq C, \end{aligned}$$

with $\mathbf{K} := \mathbf{X} \mathbf{X}^T$.



COST PARAMETER C

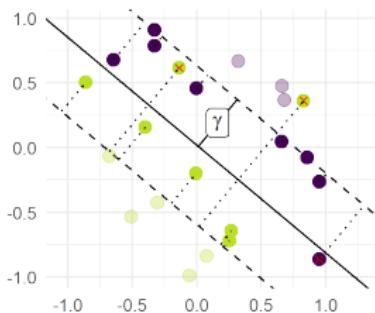
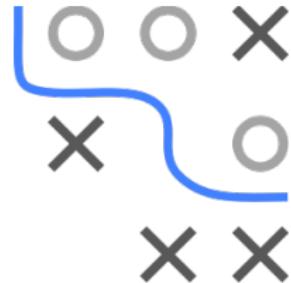
- The parameter C controls the trade-off between the two objectives of maximizing the size of the margin and minimizing frequency and size of margin violations.
- It is known under different names, such as “trade-off parameter”, “regularization parameter”, and “complexity control parameter”.
- For sufficiently large C margin violations become extremely small and the optimal solution does not violate any margins if the data is separable. The hard-margin SVM is obtained as a special case for $C \rightarrow \infty$.



SUPPORT VECTORS

There are three types of training examples:

- Non-SVs have a margin > 1 and can be removed from the problem without changing the solution.
- Some SVs are located exactly on the margin and have $yf(\mathbf{x}) = 1$.
- Other SVs are margin violators, with $yf(\mathbf{x}) < 1$, and have an associated positive slack $\zeta^{(i)} > 0$. They are misclassified if $\zeta^{(i)} \geq 1$.

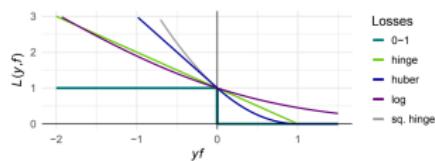




Introduction to Machine Learning

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 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 L

We derived this QP for the soft-margin SVM:

$$\begin{aligned} \min_{\theta, \theta_0, \zeta^{(i)}} \quad & \frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n \zeta^{(i)} \\ \text{s.t.} \quad & y^{(i)} (\langle \theta, \mathbf{x}^{(i)} \rangle + \theta_0) \geq 1 - \zeta^{(i)} \quad \forall i \in \{1, \dots, n\}, \\ \text{and} \quad & \zeta^{(i)} \geq 0 \quad \forall i \in \{1, \dots, n\}. \end{aligned}$$



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 be 0 (we do not get a bonus for points beyond the margin on the side). So we can rewrite the above:

$$\frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)})); \quad L(y, f) = \begin{cases} 1 - yf & \text{if } y \neq 0 \\ 0 & \text{if } y = 0 \end{cases}$$

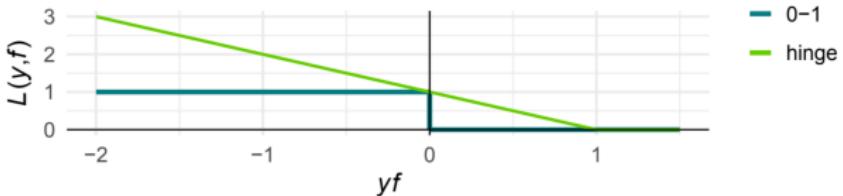
We can also write $L(y, f) = \max(1 - yf, 0)$.

SOFT-MARGIN SVM WITH ERM AND HINGE I

$$\mathcal{R}_{\text{emp}}(\theta) = \frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)})) ; \quad L(y, f) = \max(1 - yf, 0)$$



- 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 certain sense it is the best upper convex relaxation of the 0-1 loss.



SOFT-MARGIN SVM WITH ERM AND HINGE LOSS

$$\frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)})) ; L(y, f) = \max(1 - yf,$$

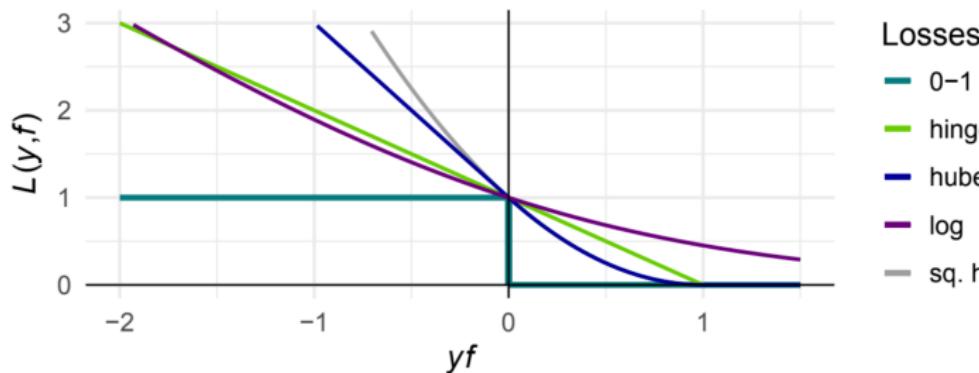
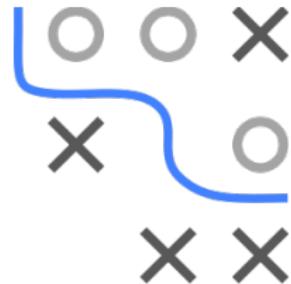


- The ERM interpretation does not require any of the terms – loss or the regularizer – to be geometrically meaningful.
- The above form is a very compact form to define the convex optimization problem of the SVM.
- It is "well-behaved" due to convexity, every minimum is global.
- The above is convex, without constraints! We might see that "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"

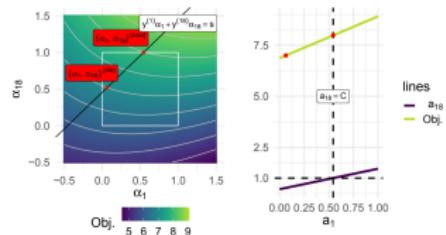




Introduction to Machine Learning

Support Vector Machine Training

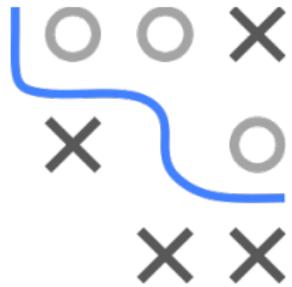
Learning goals



- 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

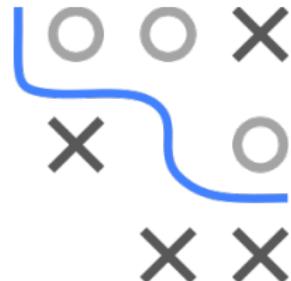
- Until now, we have ignored the issue of solving the various optimization problems.
- The first question is whether we should solve the **primal** or **dual problem**.
- In the literature SVMs are usually trained in the dual.
- However, SVMs can be trained both in the primal and the dual; each approach 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 the basic ideas here.



TRAINING SVM IN THE PRIMAL

Unconstrained formulation of soft-margin SVM:

$$\min_{\theta, \theta_0} \quad \frac{\lambda}{2} \|\theta\|^2 + \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)} | \theta))$$



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:

- ➊ Use smoothed loss (squared hinge, huber), then do GD.
NB: Will not create a sparse SVM if we do not add extra tri
- ➋ Use **subgradient** methods.
- ➌ Do stochastic subgradient descent.
Pegasos: Primal Estimated sub-GrAdient SOlver for SVM.

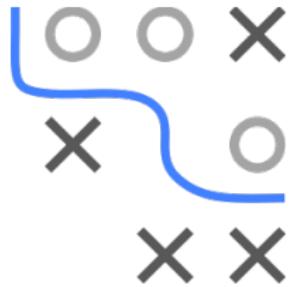
PEGASOS: SSGD IN THE PRIMAL

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**
-

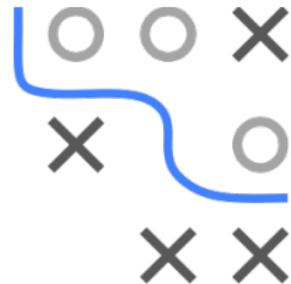
Note the weight decay due to the L2-regularization.

TRAINING SVM IN THE DUAL

The dual problem of the soft-margin SVM is

$$\max_{\alpha} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \quad \sum_{i=1}^n \alpha_i y^{(i)} = 0$$



We 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

$$\max_{\alpha} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \quad \sum_{i=1}^n \alpha_i y^{(i)} = 0$$



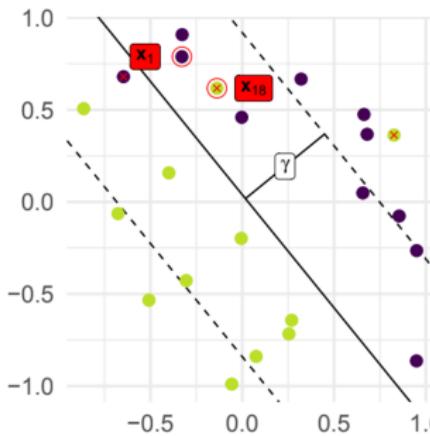
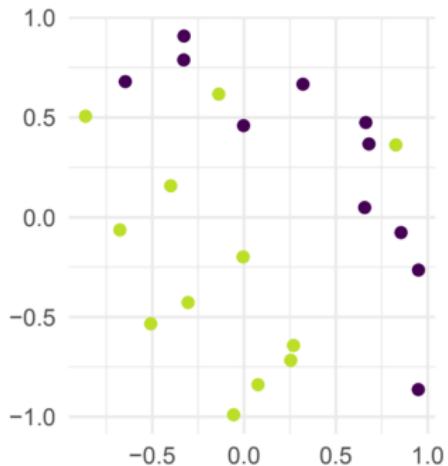
Pairwise coordinate ascent in the dual

- 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 sign 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.



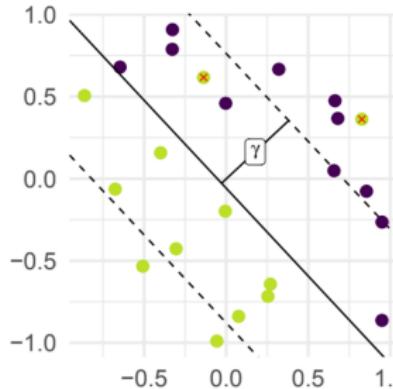
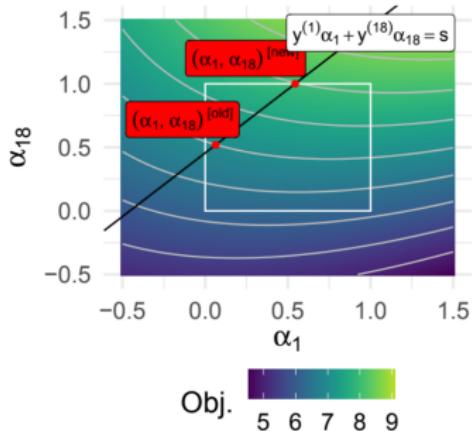
TRAINING SVM IN THE DUAL

$$\max_{\alpha} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \rangle$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \quad \sum_{i=1}^n \alpha_i y^{(i)} = 0$$

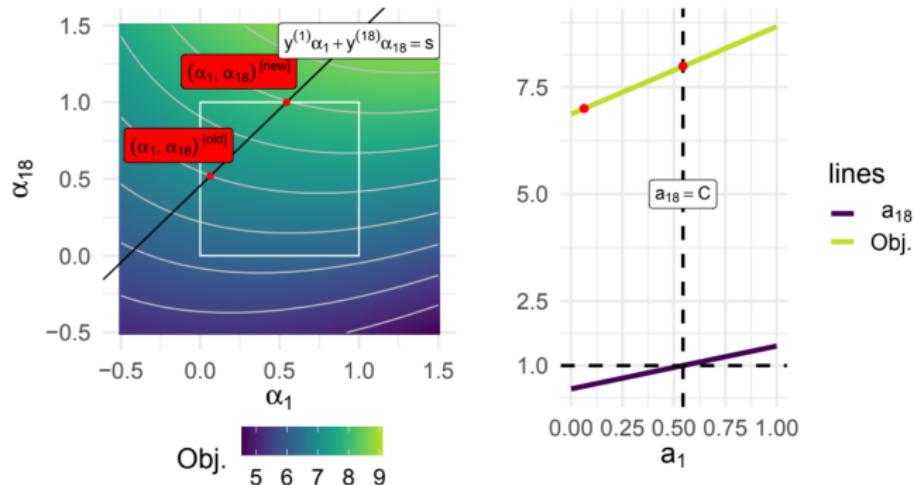
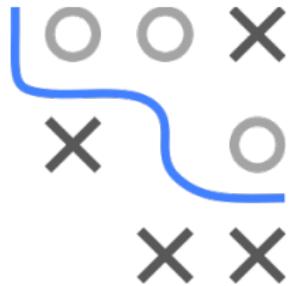


We move on the linear constraint until the pair-optimum or the bounday (here



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 an 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

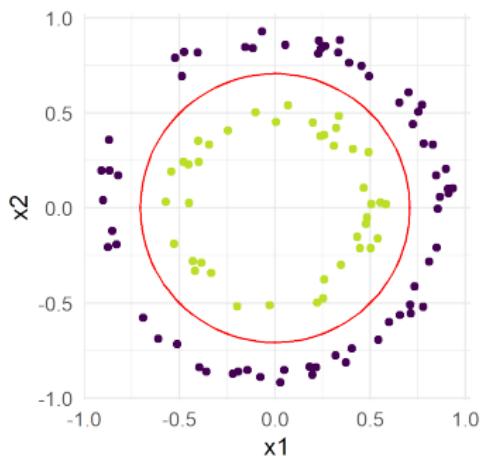
Feature Selection





Introduction to Machine Learning

Feature Generation for Nonlinear Separation

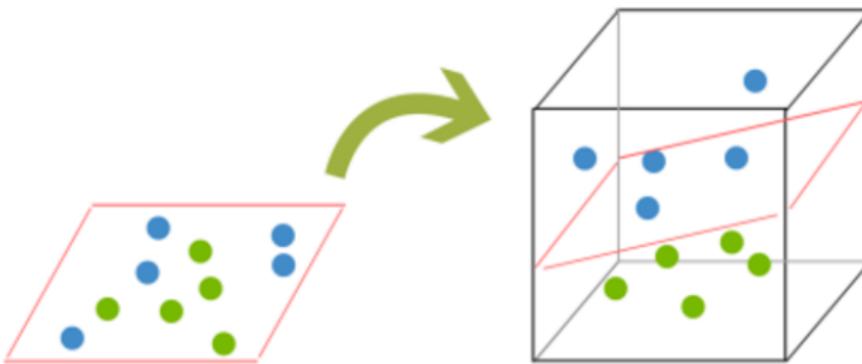


Learning goals

- Understand how nonlinearity can be introduced via feature maps in SVMs
- Know the limitation of feature maps

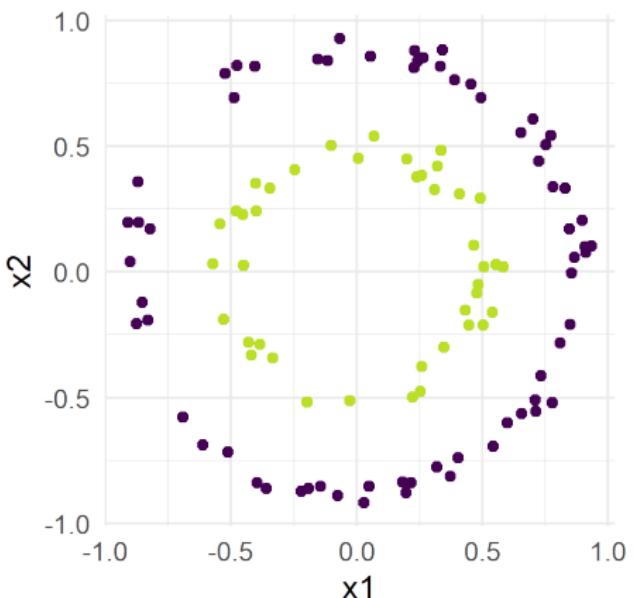
NONLINEARITY VIA FEATURE MAPS

- 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

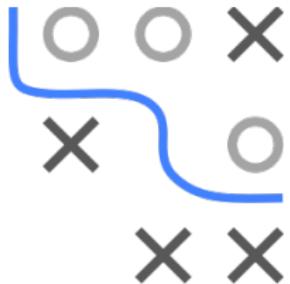


NONLINEARITY VIA FEATURE MAPS

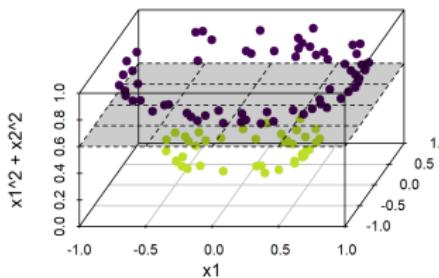
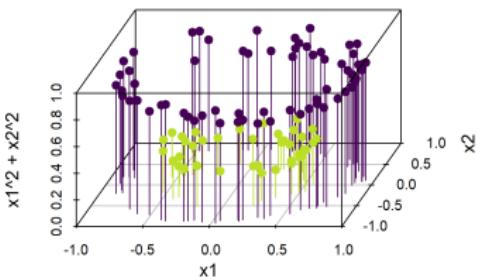
In order to “lift” the data points into a higher dimension, we have a suitable **feature map** $\phi : \mathcal{X} \rightarrow \Phi$. Let us consider another example where the classes lie on two concentric circles:



NONLINEARITY VIA FEATURE MAPS

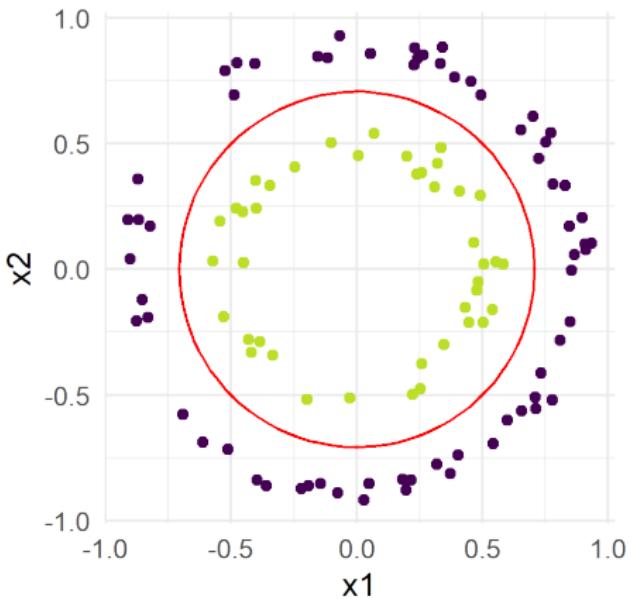


We apply the feature map $\phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$ to map our points into a 3D space. Now our data can be separated by a hyperplane.



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 LIMITATI

Let us have a look at a similar nonlinear feature map $\phi : \mathbb{R}^2 \rightarrow$ 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 .

$$k_1 = \binom{d + p - 1}{d} \quad k_2 = \binom{d + p}{d} - 1$$

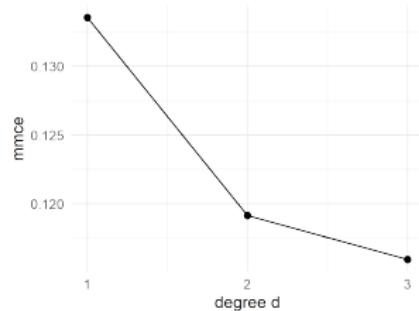
Which is quite a lot, if p is large.



FEATURE MAPS: COMPUTATIONAL LIMITATION

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 of degree d and $C = 1$:

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
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
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
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
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

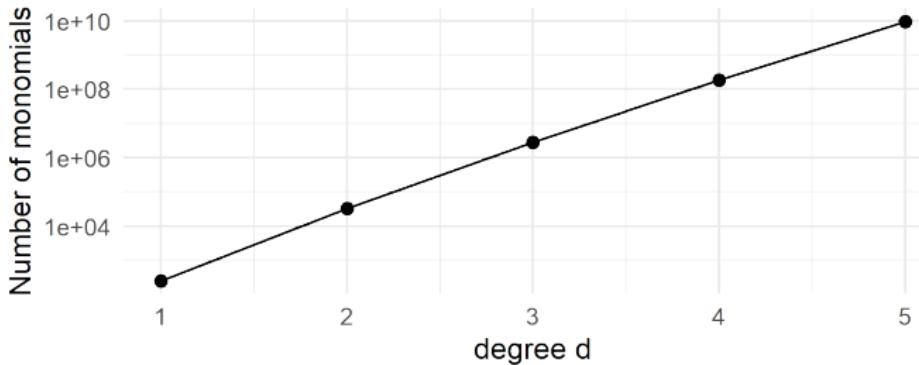


For this scenario, with increasing degree d the test mmce decreases.

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 don't do any tuning - as we always should for the SVM!

FEATURE MAPS: COMPUTATIONAL LIMITATION

However, even a 16×16 -pixel input image results in infeasible dimensions for our extracted features (monomials up to degree



In this case, training classifiers like a linear SVM via dataset transformations will incur serious **computational and memory problems**.

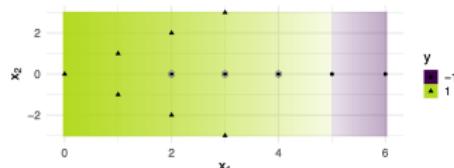
Are we at a “dead end”?

Answer: No, this is why kernels exist!



Introduction to Machine Learning

The Kernel Trick



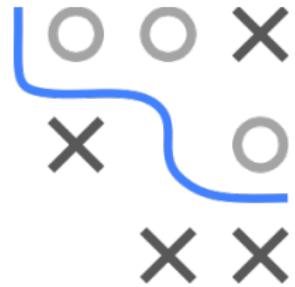
Learning goals

- 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:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \\ & \sum_{i=1}^n \alpha_i y^{(i)} = 0, \end{aligned}$$



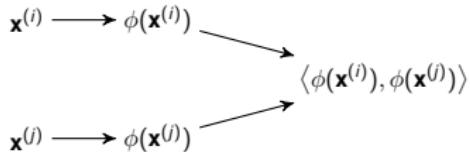
Here we replaced all features $\mathbf{x}^{(i)}$ with feature-generated, transformed versions $\phi(\mathbf{x}^{(i)})$.

We see: The optimization problem only depends on **pair-wise inner products** of the inputs.

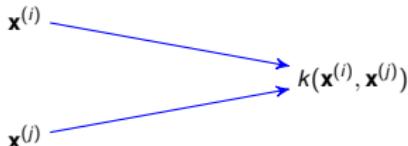
This now allows a trick to enable efficient solving.

KERNEL = FEATURE MAP + INNER PRODUC

Instead of first mapping the features to the higher-dimensional space and calculating the inner products afterwards,



it would be nice to have an efficient “shortcut” computation:



We will see: **Kernels** give us such a “shortcut”.

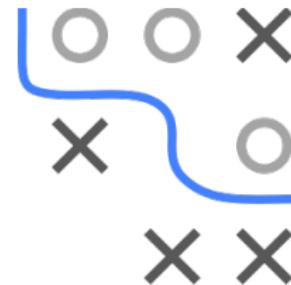
MERCER KERNEL

Definition: A (**Mercer**) **kernel** on a space \mathcal{X} is a continuous fu

$$k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$$

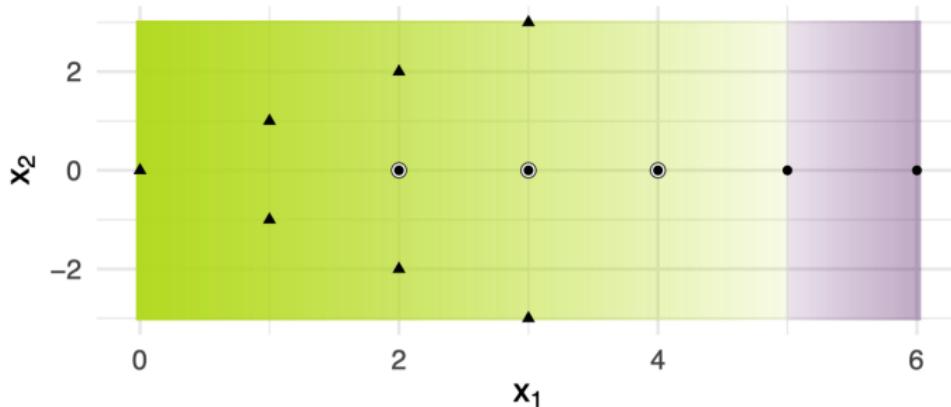
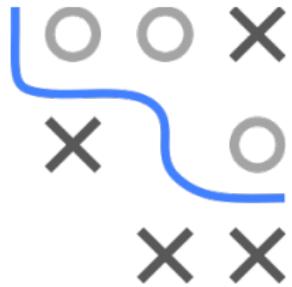
of two arguments with the properties

- 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 $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$ kernel Gram matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ with entries $K_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ 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.

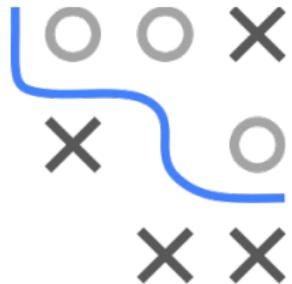


SUM AND PRODUCT KERNELS

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).

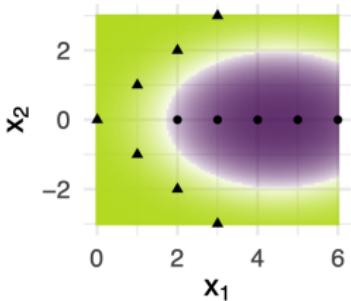
The proofs remain as (simple) exercises.



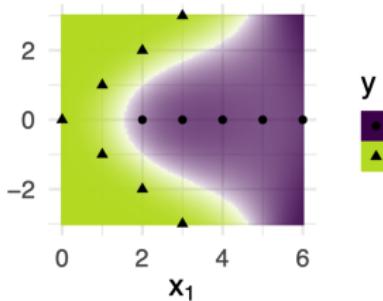
POLYNOMIAL KERNEL

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = (\mathbf{x}^\top \tilde{\mathbf{x}} + b)^d, \text{ for } b \geq 0, d \in \mathbb{N}$$

$d = 2$



$d = 3$



From the sum-product rules it directly follows that this is a kernel

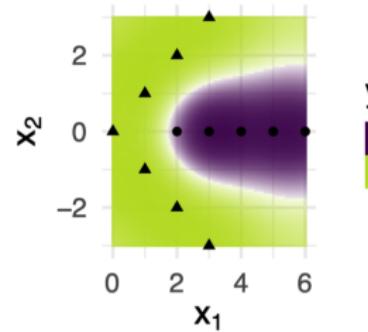
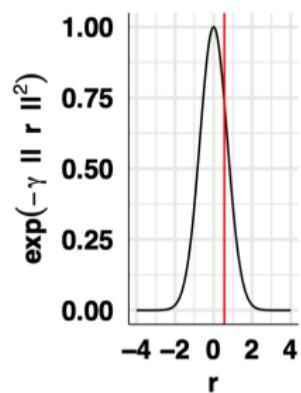
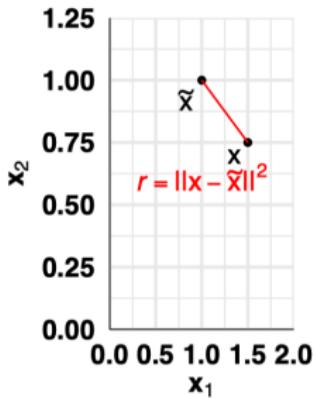
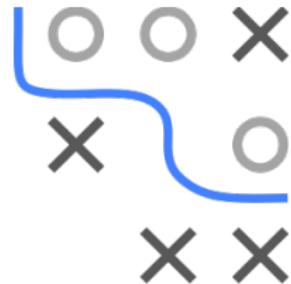
RBF KERNEL

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 inner products $\langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle$ by kernels $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$



$$\max_{\alpha} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C,$$

$$\sum_{i=1}^n \alpha_i y^{(i)} = 0.$$

This problem is still convex because \mathbf{K} is psd!

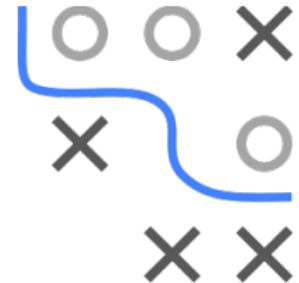
KERNEL SVM

We kernelize the dual (soft-margin) SVM problem by replacing dot 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 dot products $\langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \rangle$ by kernels $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$



$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^{(i)} y^{(j)} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \\ & \sum_{i=1}^n \alpha_i y^{(i)} = 0. \end{aligned}$$

In more compact matrix notation with \mathbf{K} denoting the kernel matrix:

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & \mathbf{1}^\top \alpha - \frac{1}{2} \alpha^\top \text{diag}(\mathbf{y}) \mathbf{K} \text{diag}(\mathbf{y}) \alpha \\ \text{s.t.} \quad & \alpha^\top \mathbf{y} = 0, \\ & 0 \leq \alpha \leq C. \end{aligned}$$

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:

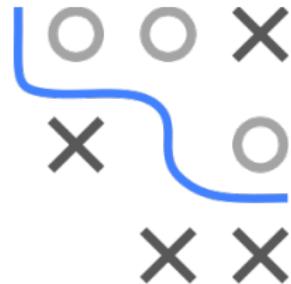
$$\begin{aligned} \langle \hat{\theta}, \phi(\mathbf{x}) \rangle &= \left\langle \sum_{i=1}^n \alpha_i y^{(i)} \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}) \right\rangle = \sum_{i=1}^n \alpha_i y^{(i)} \langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}) \rangle \\ &= \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}), \quad \text{so:} \quad f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) \end{aligned}$$

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 digits and use the rest of the data for testing; and use $C=1$.

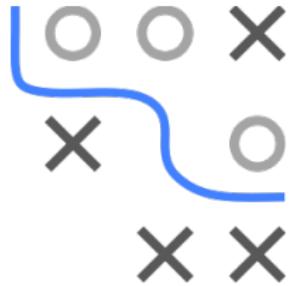
0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9

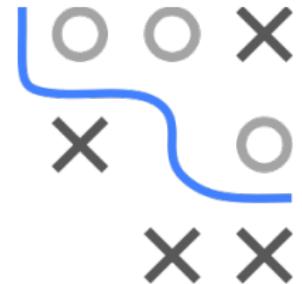
	Error
linear	0.134
poly (d = 2)	0.119
RBF (gamma = 0.001)	0.12
RBF (gamma = 1)	0.184



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 prediction is 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 the 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 representation as long as it is efficiently computable.

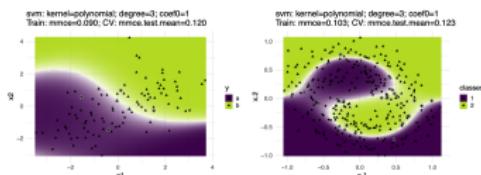




Introduction to Machine Learning

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 similar calculation and using the multinomial formula

$$(\mathbf{x}_1 + \dots + \mathbf{x}_p)^d = \sum_{k_i \geq 0, \sum_i k_i = d} \binom{d}{k_1, \dots, k_p} x_1^{k_1} \dots x_p^{k_p}$$

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 $p = d = 2$: $\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$.

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 co term in the original space, with

$$(\mathbf{x}^T \tilde{\mathbf{x}} + b)^d = (x_1 \tilde{x}_1 + \dots + x_p \tilde{x}_p + \sqrt{b} \sqrt{b})^d$$

The feature map contains all monomials up to order d .

$$\phi(\mathbf{x}) = \left(\sqrt{\binom{d}{k_1, \dots, k_{p+1}}} x_1^{k_1} \dots x_p^{k_p} b^{k_{p+1}/2} \right)_{k_i \geq 0, \sum_i k_i =}$$

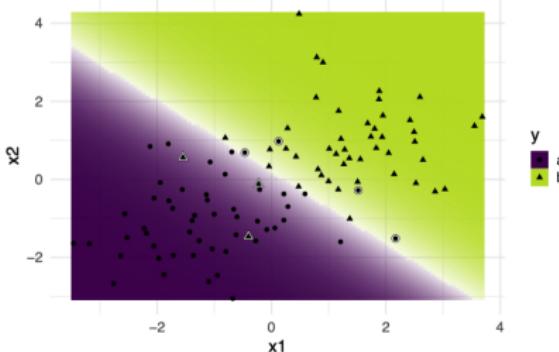
The map $\phi(\mathbf{x})$ has $\binom{p+d}{d}$ dimensions. For $p = d = 2$:

$$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2b}x_1, \sqrt{2b}x_2, \sqrt{b})$$

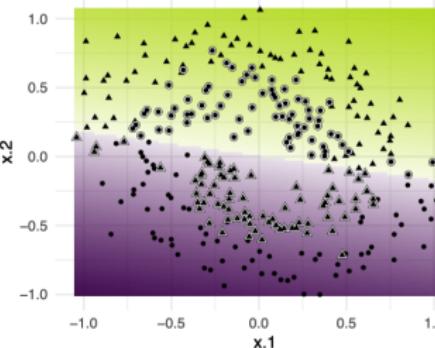
POLYNOMIAL KERNEL

Degree $d = 1$ yields a linear decision boundary.

svm: kernel=polynomial; degree=1; coef0=1
Train: mmce=0.070; CV: mmce.test.mean=0.110



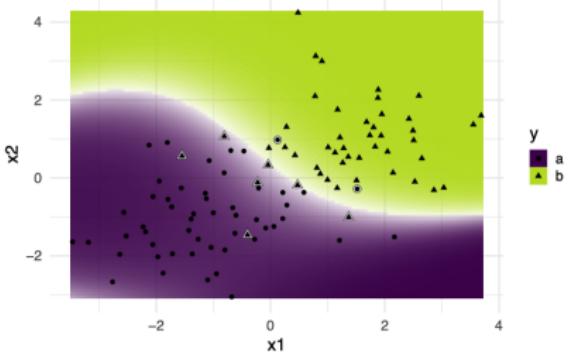
svm: kernel=polynomial; degree=1; coef0=
Train: mmce=0.500; CV: mmce.test.mean=



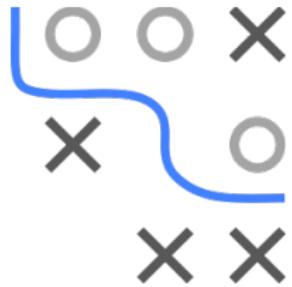
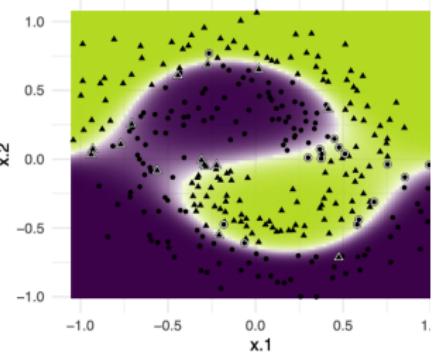
POLYNOMIAL KERNEL

The higher the degree, the more nonlinearity in the decision boundary.

svm: kernel=polynomial; degree=3; coef0=1
Train: mmce=0.090; CV: mmce.test.mean=0.120



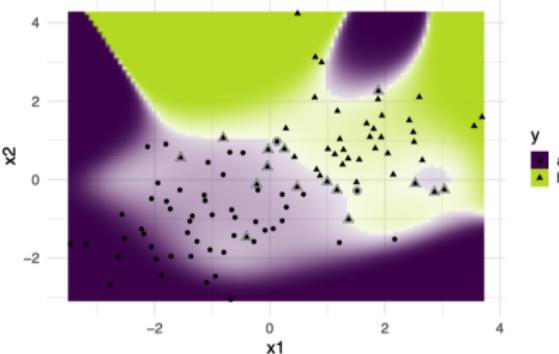
svm: kernel=polynomial; degree=3; coef0=1
Train: mmce=0.103; CV: mmce.test.mean=0.120



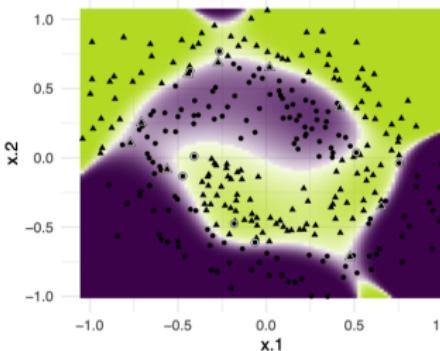
POLYNOMIAL KERNEL

The higher the degree, the more nonlinearity in the decision boundary.

svm: kernel=polynomial; degree=9; coef0=1
Train: mmce=0.170; CV: mmce.test.mean=0.250



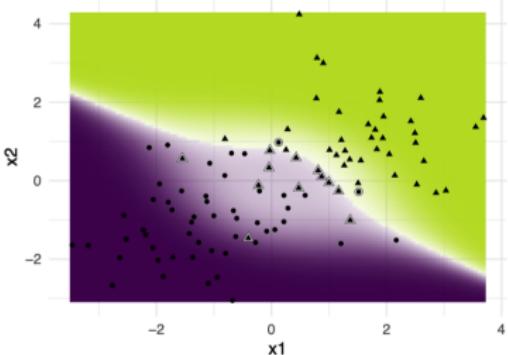
svm: kernel=polynomial; degree=9; coef0:
Train: mmce=0.057; CV: mmce.test.mean:



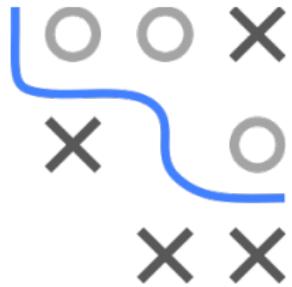
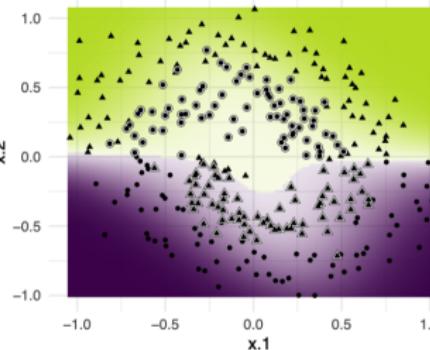
POLYNOMIAL KERNEL

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=0.140; CV: mmce.test.mean=0.180



svm: kernel=polynomial; degree=3; coef0=0
Train: mmce=0.473; CV: mmce.test.mean=

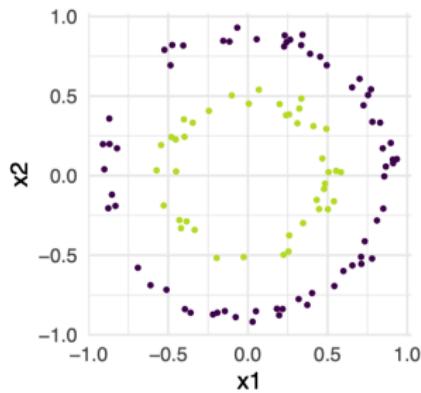




Introduction to Machine Learning

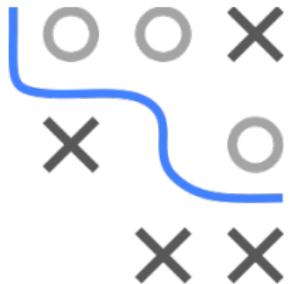
Reproducing Kernel Hilbert Space and Representer Theorem

Learning goals



- Know that for every kernel there is an associated feature map a space (Mercer's Theorem)
- Know that this feature map is unique, and the reproducing kernel Hilbert space (RKHS) is 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 with 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 $\mathcal{X} \rightarrow \mathbb{R}$.
“features” of the RKHS are the kernel functions evaluated at

- 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:

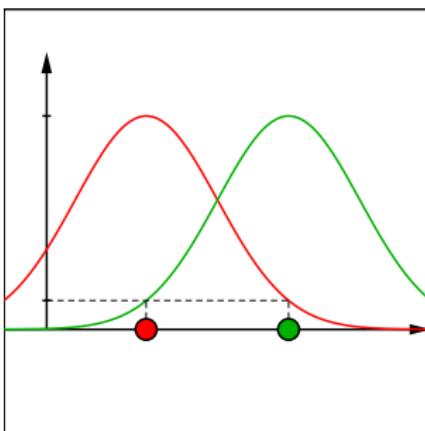
$$\langle k(\mathbf{x}, \cdot), k(\tilde{\mathbf{x}}, \cdot) \rangle = \langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle = k(\mathbf{x}, \tilde{\mathbf{x}}).$$

REPRODUCING KERNEL HILBERT SPACE

- The RKHS provides us with a useful interpretation:
an input $\mathbf{x} \in \mathcal{X}$ mapped to the **basis function** $\phi(\mathbf{x}) = k(\mathbf{x})$
- 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 space of basis functions.

REPRESENTER THEOREM

The **representer theorem** tells us that the solution of a support machine problem



$$\begin{aligned} \min_{\theta, \theta_0, \zeta^{(i)}} \quad & \frac{1}{2} \theta^\top \theta + C \sum_{i=1}^n \zeta^{(i)} \\ \text{s.t.} \quad & y^{(i)} \left(\langle \theta, \phi(\mathbf{x}^{(i)}) \rangle + \theta_0 \right) \geq 1 - \zeta^{(i)} \quad \forall i \in \{1, \dots, n\} \\ \text{and} \quad & \zeta^{(i)} \geq 0 \quad \forall i \in \{1, \dots, n\} \end{aligned}$$

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 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:

$$\langle \theta, \phi(\mathbf{x}^{(i)}) \rangle = \underbrace{\langle v, \phi(\mathbf{x}^{(i)}) \rangle}_{=0} + \underbrace{\langle v^\perp, \phi(\mathbf{x}^{(i)}) \rangle}_{\forall i \in \{1, 2, \dots, n\}}$$

Thus, we have two independent optimization problems, namely the standard problem for v and the unconstrained minimization problem of $\|v^\perp\|^2$ for v^\perp , the 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

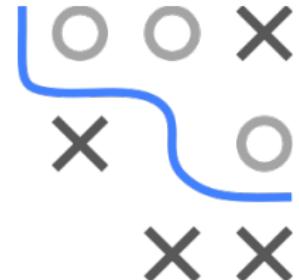
$$\boldsymbol{\theta} = \sum_{j=1}^n \beta_j \cdot \phi(\mathbf{x}^{(j)})$$

for the weight vector $\boldsymbol{\theta} \in \Phi$.

- The SVM prediction on $\mathbf{x} \in \mathcal{X}$ can be computed as

$$f(\mathbf{x}) = \sum_{j=1}^n \beta_j \langle \phi(\mathbf{x}^{(j)}), \phi(\mathbf{x}) \rangle + \theta_0 .$$

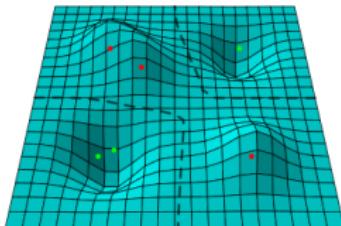
It can be shown that the sum is **sparse**: $\beta_j = 0$ for non-support vectors.





Introduction to Machine Learning

The Gaussian RBF Kernel



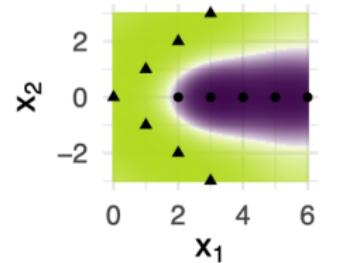
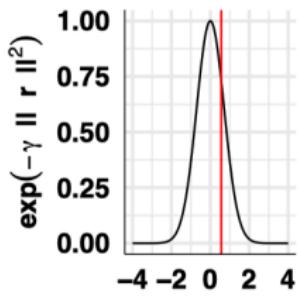
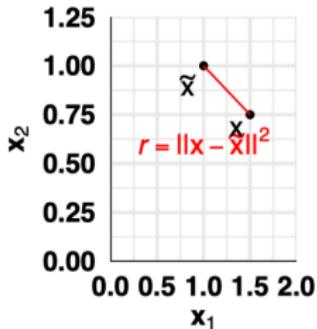
Learning goals

- Know the Gaussian (RBF) kernel
- Understand that all data sets are not separable with this kernel
- Understand the effect of the kernel hyperparameter σ

RBF KERNEL

The “radial” **Gaussian kernel** is defined as

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = \exp\left(-\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|^2}{2\sigma^2}\right) \quad \text{or} \quad k(\mathbf{x}, \tilde{\mathbf{x}}) = \exp(-\gamma \|\mathbf{x} - \tilde{\mathbf{x}}\|^2)$$



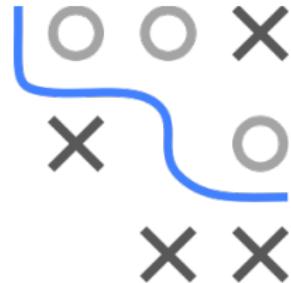
A straightforward extension is

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = \exp\left(-(\mathbf{x} - \tilde{\mathbf{x}})^T C (\mathbf{x} - \tilde{\mathbf{x}})\right)$$

for a symmetric, positive definite matrix C .

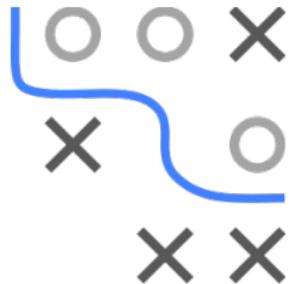
RBF KERNEL

- With a Gaussian kernel, all RKHS basis functions $\phi(\mathbf{x}) = \dots$ 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 constraint 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 encounter some margin violators because we should not trust nonlinear data. A hard-margin SVM with Gaussian kernels may fail to separate any dataset but will usually overfit.

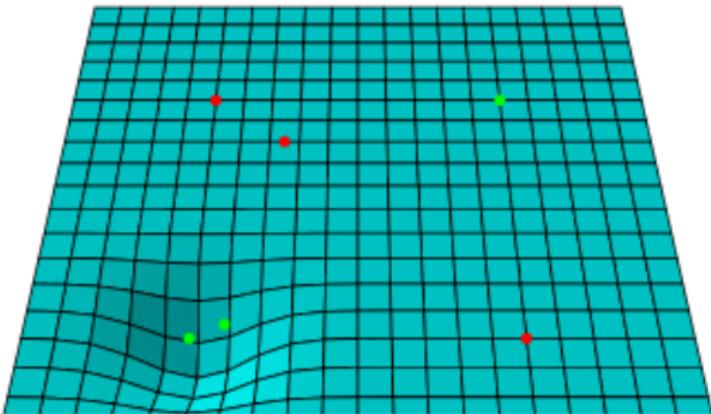


WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the ϵ the RBF kernel much better as a local model.



$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$



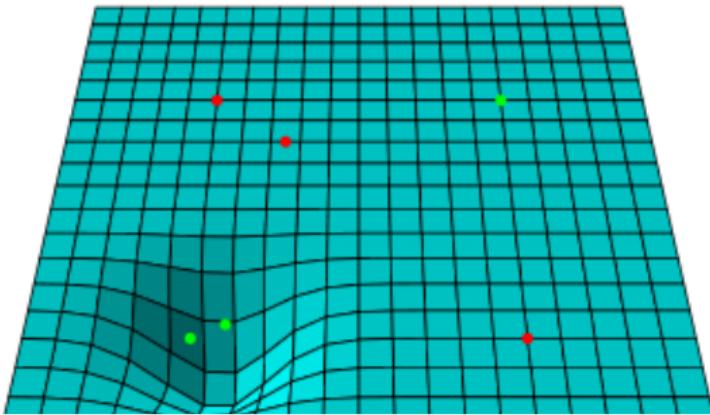
All support vectors :
RBF "bumps", these
with the dual variab
multipliers α_i and la
then "mix" these bu
to form the decision
Which becomes a k

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the ϵ the RBF kernel much better as a local model.



$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$



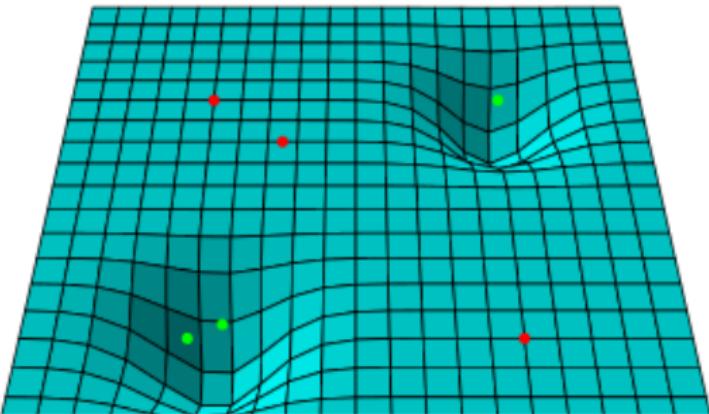
All support vectors :
RBF "bumps", these
with the dual variab
multipliers α_i and la
then "mix" these bu
to form the decision
Which becomes a k

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the ϵ the RBF kernel much better as a local model.



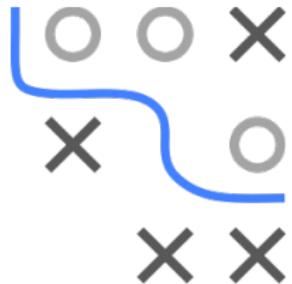
$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$



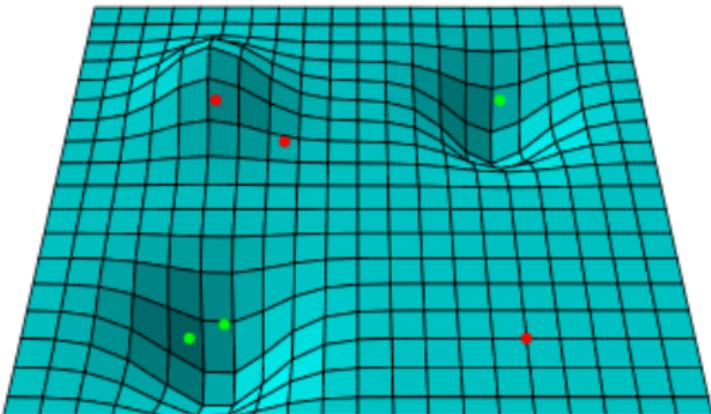
All support vectors :
RBF "bumps", these
with the dual variab
multipliers α_i and la
then "mix" these bu
to form the decision
Which becomes a k

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the ϵ the RBF kernel much better as a local model.



$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$



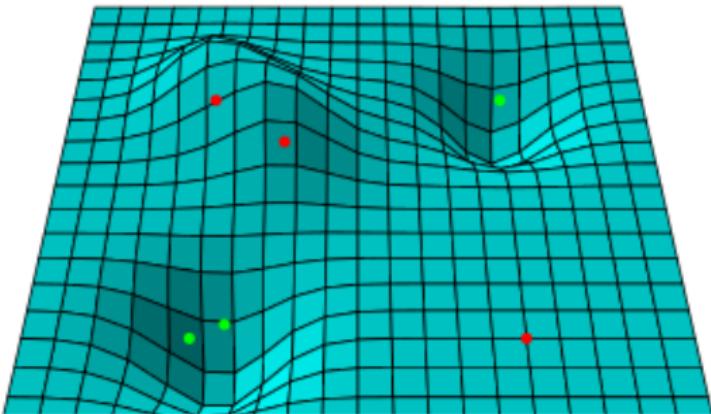
All support vectors :
RBF "bumps", these
with the dual variab
multipliers α_i and la
then "mix" these bu
to form the decision
Which becomes a k

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the ϵ the RBF kernel much better as a local model.



$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$



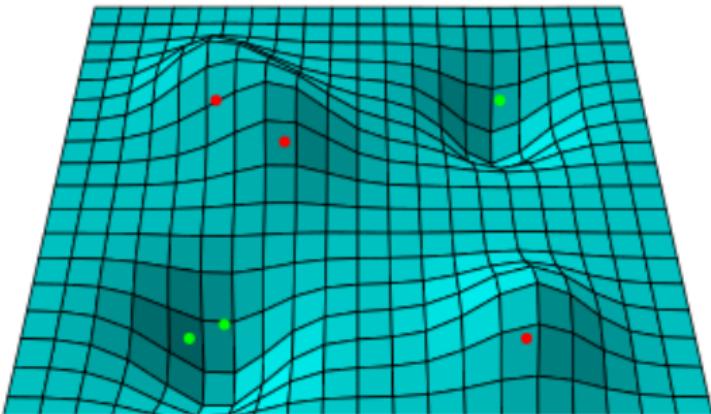
All support vectors :
RBF "bumps", these
with the dual variab
multipliers α_i and la
then "mix" these bu
to form the decision
Which becomes a k

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the ϵ the RBF kernel much better as a local model.



$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$



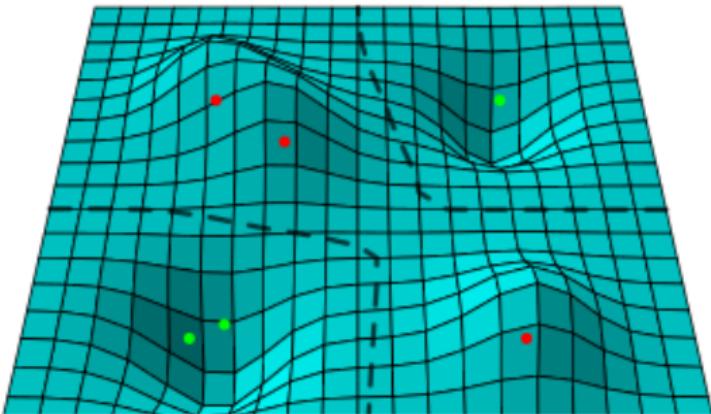
All support vectors :
RBF "bumps", these
with the dual variab
multipliers α_i and la
then "mix" these bu
to form the decision
Which becomes a k

WEIGHTED MIXTURE OF GAUSSIANS

Via the RKHS / basis function intuition we can understand the ϵ the RBF kernel much better as a local model.



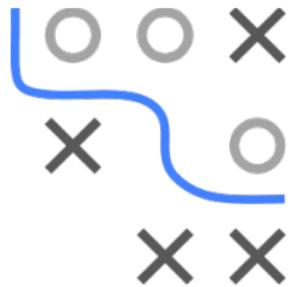
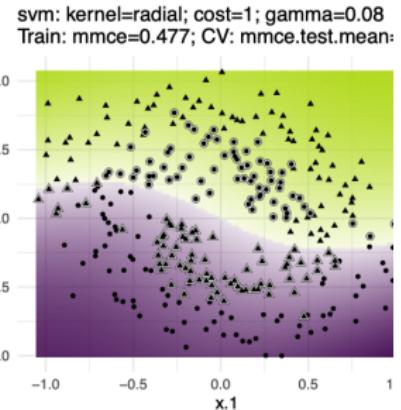
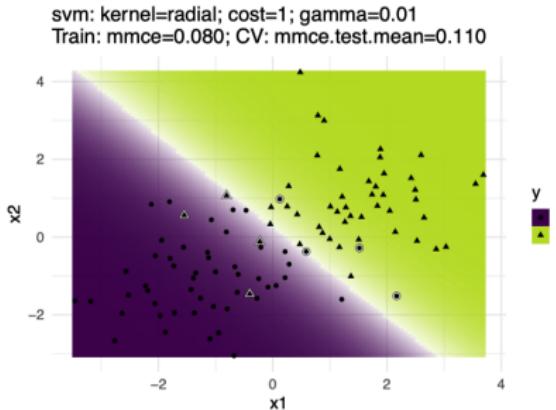
$$f(\mathbf{x}) = \sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0$$



All support vectors :
RBF "bumps", these
with the dual variab
multipliers α_i and la
then "mix" these bu
to form the decision
Which becomes a b

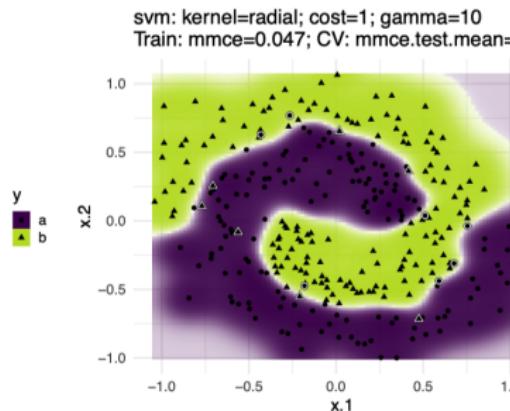
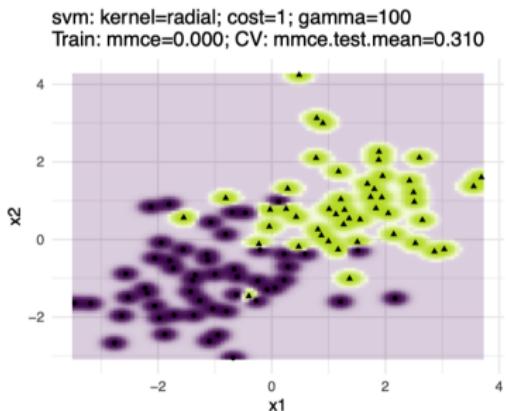
RBF KERNEL WIDTH

A large σ (or a small γ) will make the decision boundary very smooth and in the limit almost linear.



RBF KERNEL WIDTH

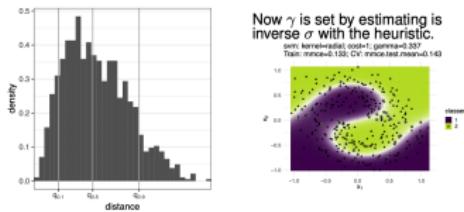
A small σ parameter makes the function more “wiggly”, in the limit it will totally over fit the data by basically modelling each training data point with maximal uncertainty at all other test points.





Introduction to Machine Learning

SVM Model Selection

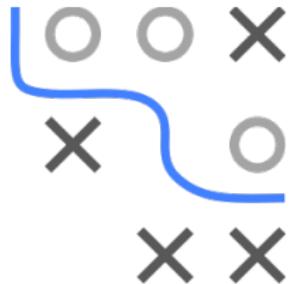


Learning goals

- Know that the SVM is sensitive to hyperparameter choices
- Understand the effect of different (kernel) hyperparameters

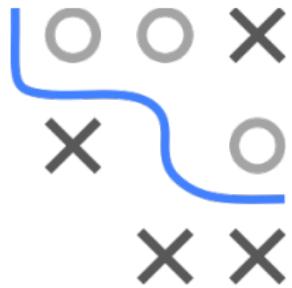
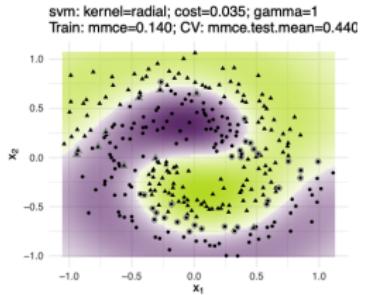
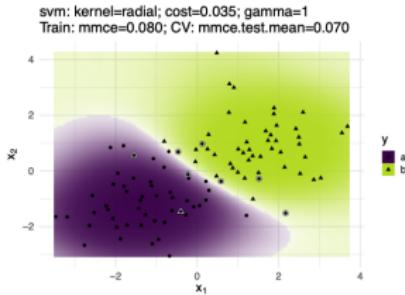
MODEL SELECTION FOR KERNEL SVMS

- “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 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 advantage that kernel parameters are directly estimated during training.

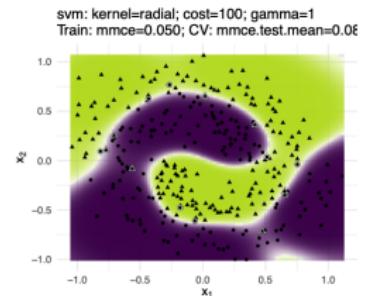
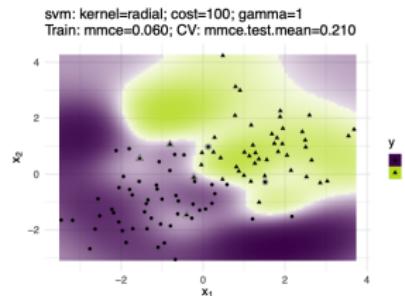


SVM HYPERPARAMETERS

Small C “allows” for margin-violating points in favor of a large margin

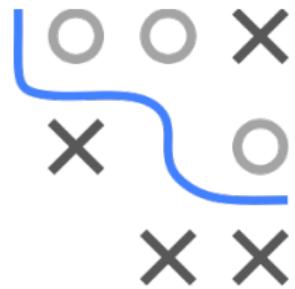
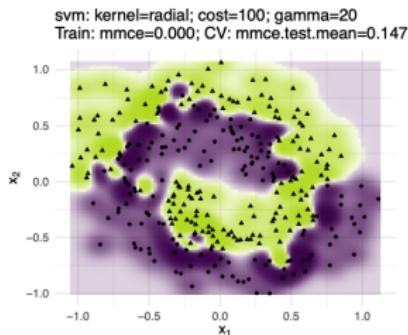
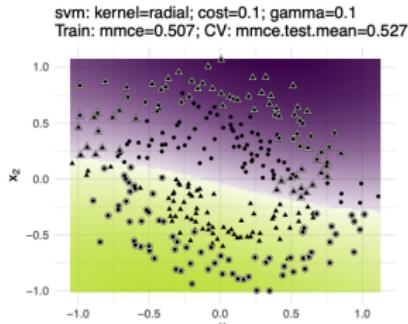
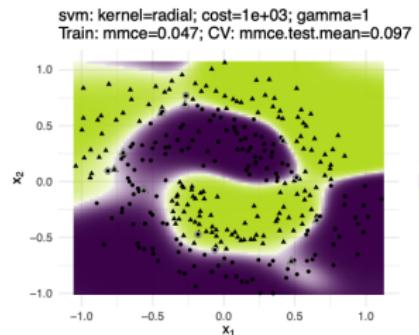
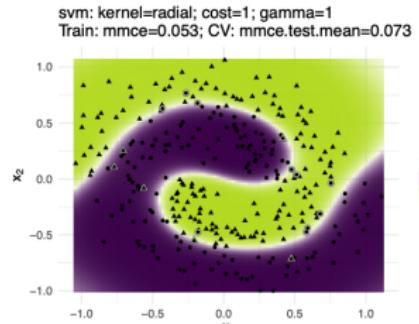


Large C penalizes margin violators, decision boundary is more complex



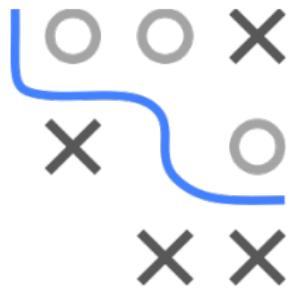
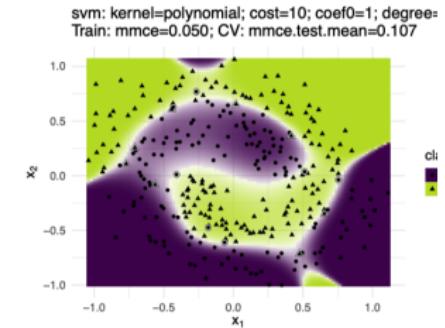
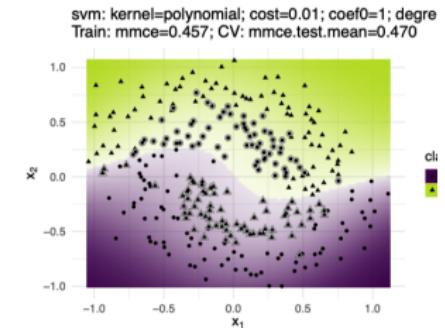
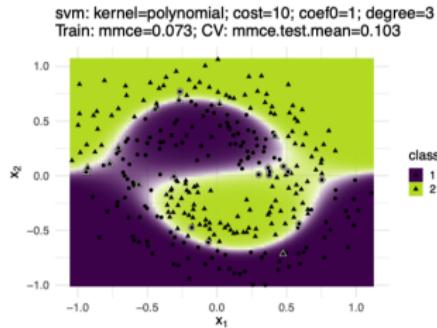
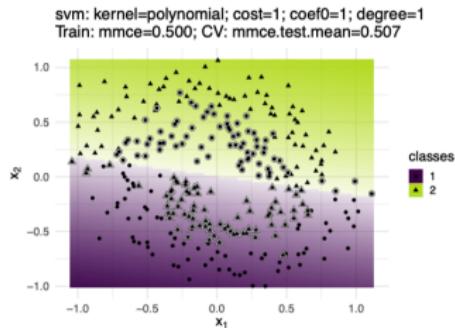
SVM HYPERPARAMETERS

Hyperparameters strongly influence the model: RBF kernel.



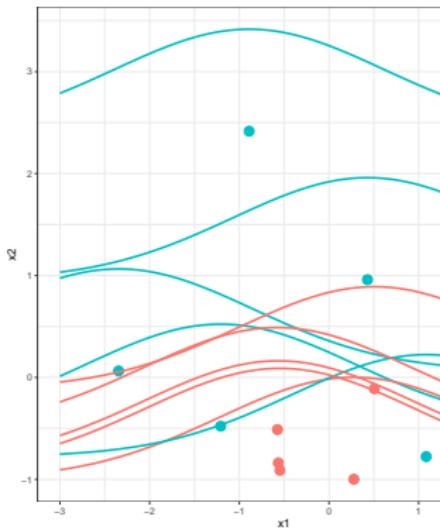
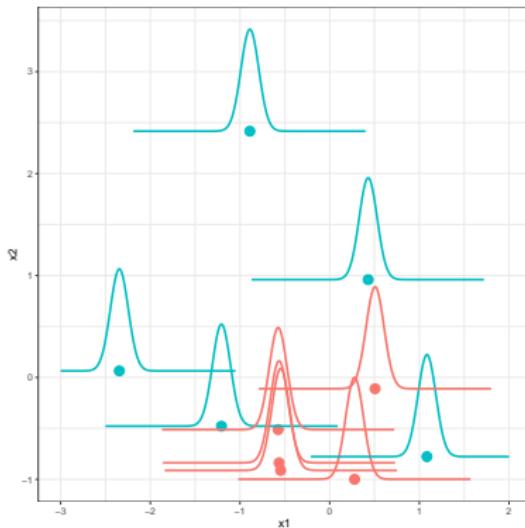
SVM HYPERPARAMETERS

Hyperparameters strongly influence the model: Polynomial kerr



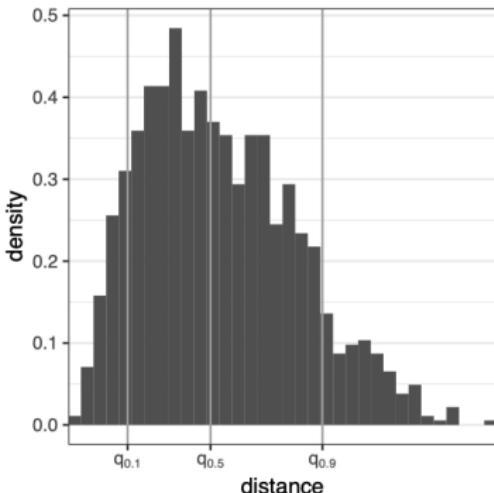
RBF SIGMA HEURISTIC

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 for the width hyperparameter σ^2 .



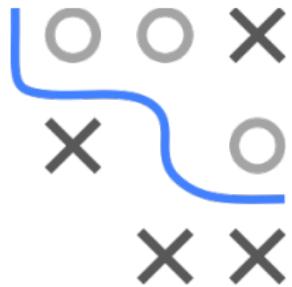
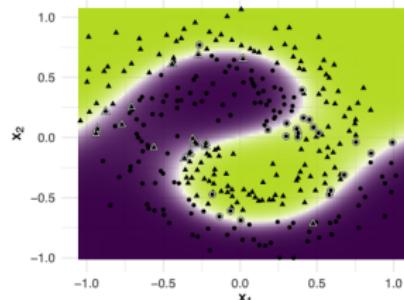
RBF SIGMA HEURISTIC

- 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.



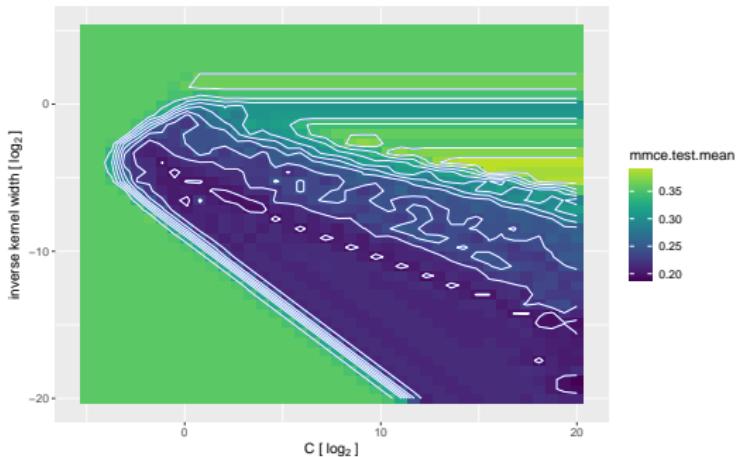
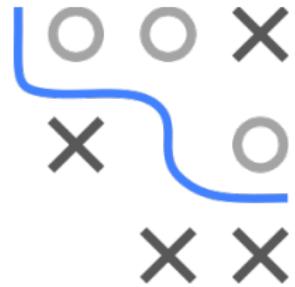
Now γ is set by estimating inverse σ with the heuristic

svm: kernel=radial; cost=1; gamma=0.337
Train: mmce=0.133; CV: mmce.test.mean=0.



SVM HYPERPARAMETERS

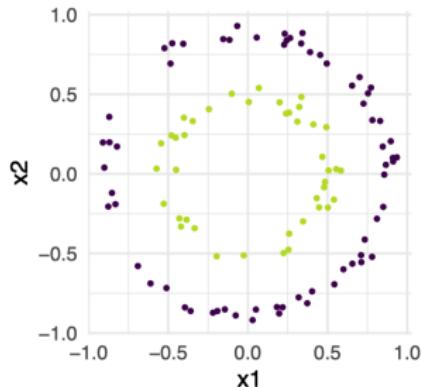
- RBF-SVM parameters are often optimized on log-scale, as we want to explore large values and values close to 0.
- E.g.: $C \in [2^{-15}, 2^{15}]$, $\gamma \in [2^{-15}, 2^{15}]$
- The cross-validated performance landscape often forms a characteristic "ridge" with a larger area of equally good val





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

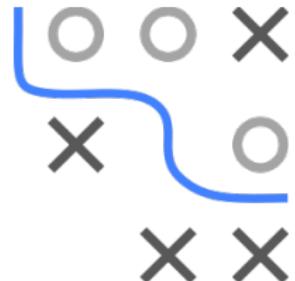
SVMS AS NON-PARAMETRIC MODELS

- In contrast to linear models, for an SVM we do not have to specify the number of coefficients of the decision function before training.
- The number of coefficients depends on the size of the data set, not on the number of support vectors.
- Such models are called **non-parametric**.
- The big advantage of non-parametric models is that their representational 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 such 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 infinitely many 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 $C(\mathcal{X})$; i.e., for all $g \in C(\mathcal{X})$, $\varepsilon > 0$ 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 $\mathcal{X} \subset \mathbb{I}$ define $C(n) = C_0 \cdot n^{q-1}$ for some $C_0 > 0$ and $0 < q < 1/p$. Fix a 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.

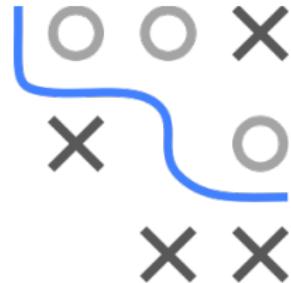
ASYMPTOTIC PERFORMANCE

- Convergence of the risk to the Bayes risk for all distributions 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 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 changes at the right rate with n :
 - $n \cdot C(n)$ tends to infinity, which means that the relative 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 detailed 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.

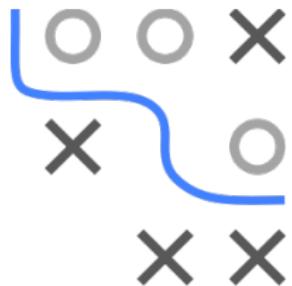


Kernels on Infinite-Dimensional Vect Spaces



KERNELS ON INFINITE-DIMENSIONAL VECT 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 len of graphs, or of trees).
- Such data structures are natural representations for, e.g, documents.
- There are many examples of data that do not naturally con vector form.
- Most often meaningful and cheap-to-compute kernels can defined directly on the input data structures – they simply 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

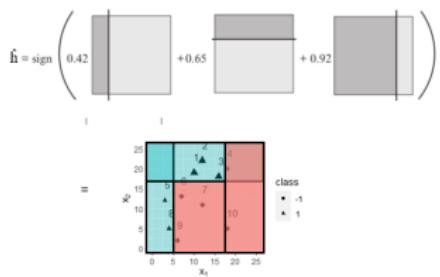
Feature Selection





Introduction to Machine Learning

Introduction to Boosting / AdaBoost

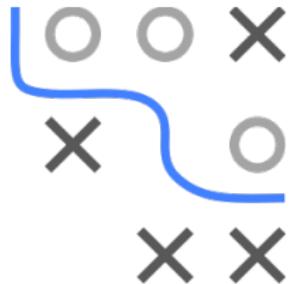


Learning goals

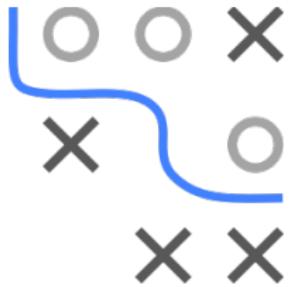
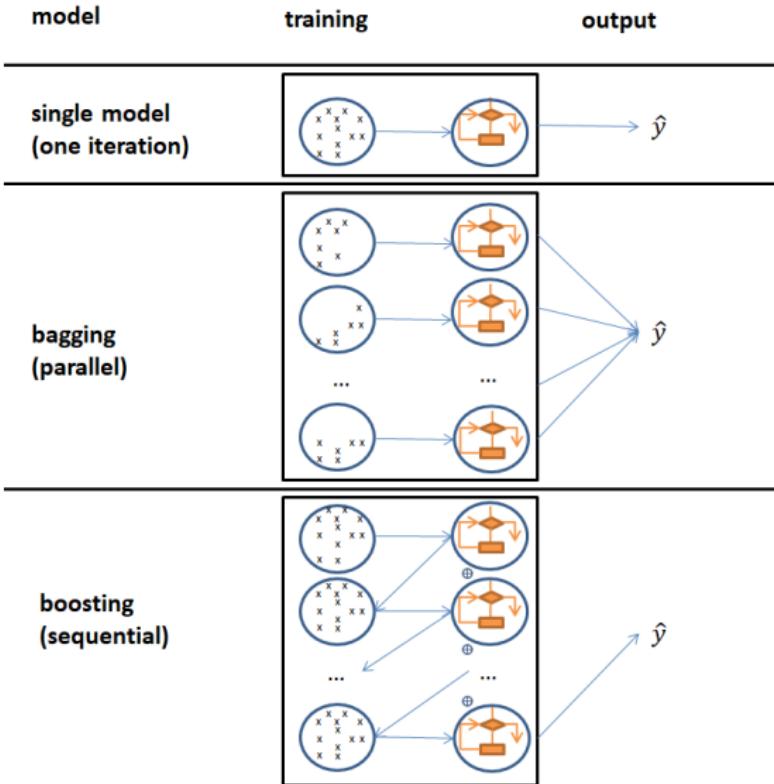
- Understand general idea of boosting
- Learn AdaBoost algorithm
- Understand difference between bagging and boosting

INTRODUCTION TO BOOSTING

- Boosting is considered to be one of the most powerful ideas within the last twenty years.
- Originally designed for classification, (especially gradient) handles regression (and many other supervised tasks) 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”.



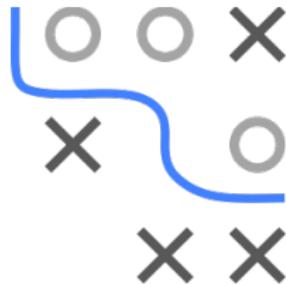
BOOSTING VS. BAGGING



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.

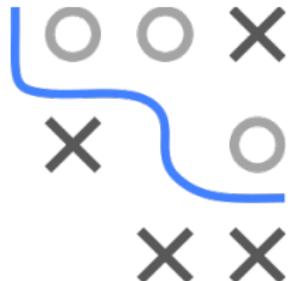
THE BOOSTING ANSWER - ADABOOST

Any weak (base) learner can be iteratively boosted to become a strong learner (Schapire and Freund, 1990). 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 (1990).
- 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 on the 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 we learn 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. The purpose is to give higher weights to base learners with high 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 h.
 classifier $\hat{b}^{[m]}$
- 4: Calculate weighted in-sample misclassification rate

$$\text{err}^{[m]} = \sum_{i=1}^n w^{[m](i)} \cdot \mathbb{1}_{\{y^{(i)} \neq \hat{b}^{[m]}(\mathbf{x}^{(i)})\}}$$

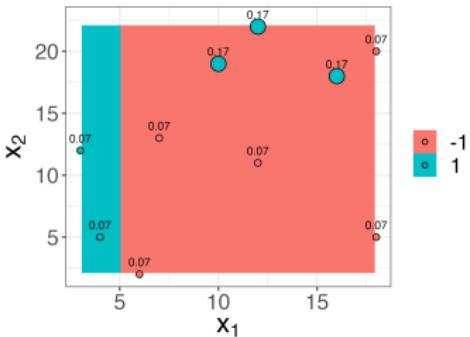
- 5: Compute: $\hat{\beta}^{[m]} = \frac{1}{2} \log \left(\frac{1-\text{err}^{[m]}}{\text{err}^{[m]}} \right)$
- 6: Set: $w^{[m+1](i)} = w^{[m](i)} \cdot \exp \left(-\hat{\beta}^{[m]} \cdot y^{(i)} \cdot \hat{b}^{[m]}(\mathbf{x}^{(i)}) \right)$
- 7: Normalize $w^{[m+1](i)}$ such that $\sum_{i=1}^n w^{[m+1](i)} = 1$
- 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 $w^{[1](i)} = \frac{1}{10} \quad \forall i \in 1, \dots, 10.$



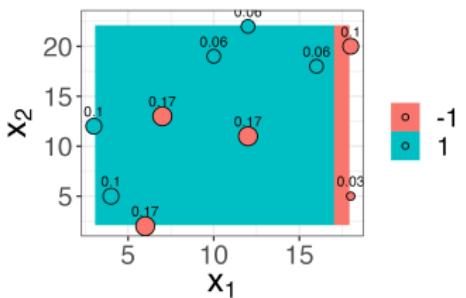
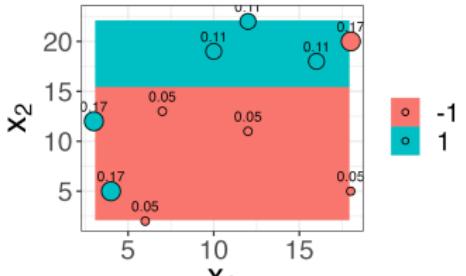
New observation weights:

- 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: $w^{[2](i)}$
 - misclassified: $w^{[2](i)} \approx 0.$

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



New observation weights (before norm):

- For misclassified observations:
 $w^{[2](i)} \cdot \exp(-\hat{\beta}^{[2]} \cdot (-1)) \approx \nu$
- For correctly classified observations:
 $w^{[3](i)} = w^{[2](i)} \cdot \exp(-\hat{\beta}^{[2]} \cdot 1)$

Iteration $m = 3$:

- $\text{err}^{[3]} \approx 3 \cdot 0.05 = 0.15$
- $\hat{\beta}^{[3]} \approx 0.92$

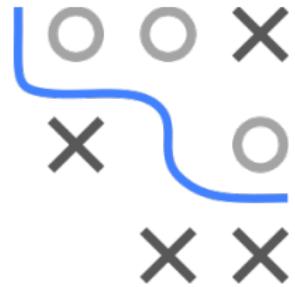
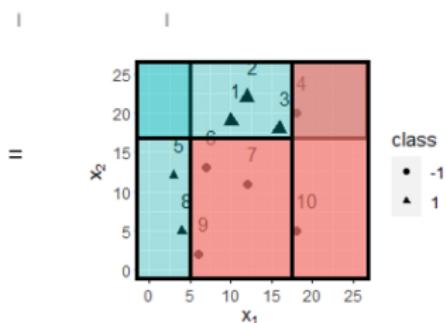
Note: the smaller the error rate of a base learner, the larger the weight, e.g., $\text{err}^{[3]} \approx 0.15 < \text{err}^{[1]} \approx 0.3$ and $\hat{\beta}^{[3]} \approx 0.92 > \hat{\beta}^{[1]} \approx 0.42$.



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:

$$\hat{h} = \text{sign} \left(0.42 \begin{array}{|c|} \hline \text{---} \\ \hline \end{array} + 0.65 \begin{array}{|c|} \hline \text{---} \\ \hline \end{array} + 0.92 \begin{array}{|c|} \hline \text{---} \\ \hline \end{array} \right)$$



Hence, when all three base classifiers are combined, all samples are classified correctly.

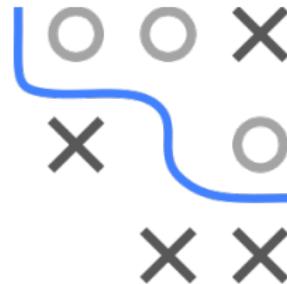
BAGGING VS BOOSTING

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

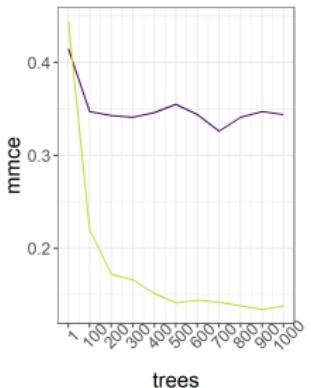
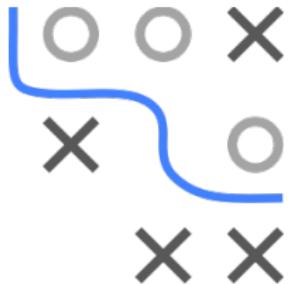
AdaBoost

- Base learners are weak learners, e.g., only single nodes
- Base learners have different weights depending on their predictive accuracy
- Sequential algorithm, order matters
- Aim: bias and variance reduction
- Tends to overfit



BAGGING VS BOOSTING STUMPS

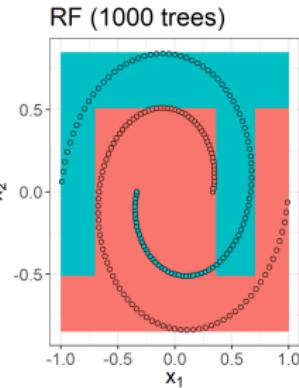
Random forest versus AdaBoost (both with stumps) on Spiral from `mlbench` ($n = 200$, $sd = 0$), with 5×5 repeated CV.



learner

— rf

— ada



● 1

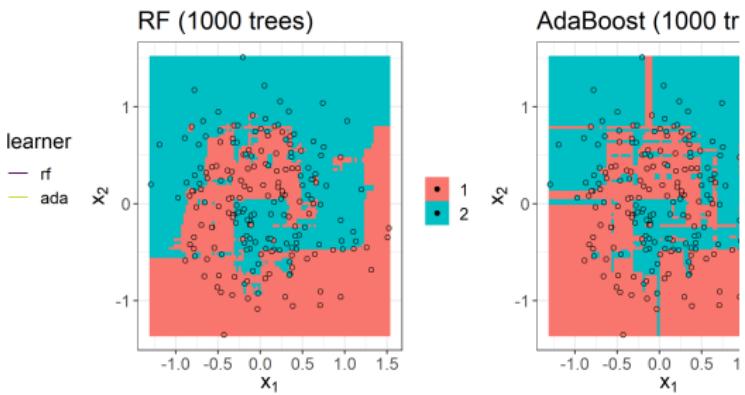
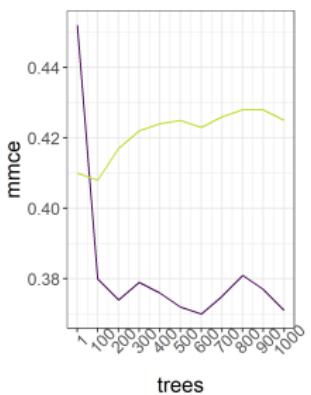
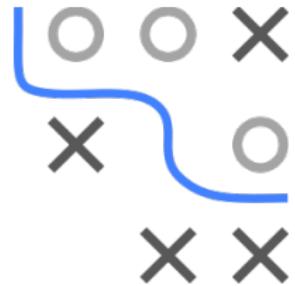
● 2



Weak learners do not work well with bagging as only variance, bias reduction happens.

OVERFITTING BEHAVIOR

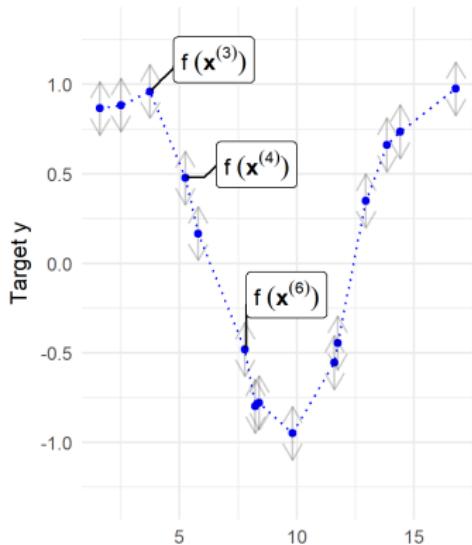
Historically, the overfitting behavior of AdaBoost was often discussed. Increasing standard deviation to $sd = 0.3$ and allowing for more flexibility in the base learners, AdaBoost overfits with increasing number of trees while the RF only saturates. The overfitting of AdaBoost is quite typical as data is very noisy.





Introduction to Machine Learning

Gradient Boosting



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:

$$f(\mathbf{x}) = \sum_{m=1}^M \alpha^{[m]} b(\mathbf{x}, \theta^{[m]}).$$

Hence, we minimize the empirical risk:

$$\mathcal{R}_{\text{emp}}(f) = \sum_{i=1}^n L \left(y^{(i)}, f \left(\mathbf{x}^{(i)} \right) \right) = \sum_{i=1}^n L \left(y^{(i)}, \sum_{m=1}^M \alpha^{[m]} b(\mathbf{x}, \theta^{[m]}) \right)$$

FORWARD STAGEWISE ADDITIVE MODELING

Why is gradient boosting a good choice for this problem?

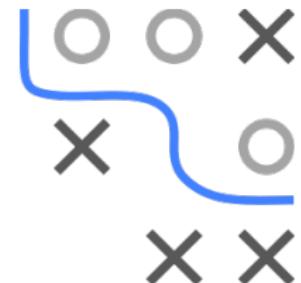
- Because of the additive structure it is difficult to jointly minimize $\mathcal{R}_{\text{emp}}(f)$ w.r.t. $((\alpha^{[1]}, \theta^{[1]}), \dots, (\alpha^{[M]}, \theta^{[M]}))$, which is a very high-dimensional parameter space (though this is less of a problem nowadays, especially in the case of numeric parallel spaces).
- Considering trees as base learners is worse as we would have to grow M trees in parallel so they work optimally together as ensemble.
- Stagewise additive modeling has nice properties, which we can 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 1 Forward Stagewise Additive Modeling.

- 1: Initialize $\hat{f}^{[0]}(\mathbf{x})$ with loss optimal constant model
 - 2: **for** $m = 1 \rightarrow M$ **do**
 - 3: $(\hat{\alpha}^{[m]}, \hat{\theta}^{[m]}) = \arg \min_{\alpha, \theta} \sum_{i=1}^n L \left(y^{(i)}, \hat{f}^{[m-1]} \left(\mathbf{x}^{(i)} \right) + \alpha b \left(\mathbf{x}^{(i)}, \theta \right) \right)$
 - 4: Update $\hat{f}^{[m]}(\mathbf{x}) \leftarrow \hat{f}^{[m-1]}(\mathbf{x}) + \hat{\alpha}^{[m]} b \left(\mathbf{x}, \hat{\theta}^{[m]} \right)$
 - 5: **end for**
-

GRADIENT BOOSTING

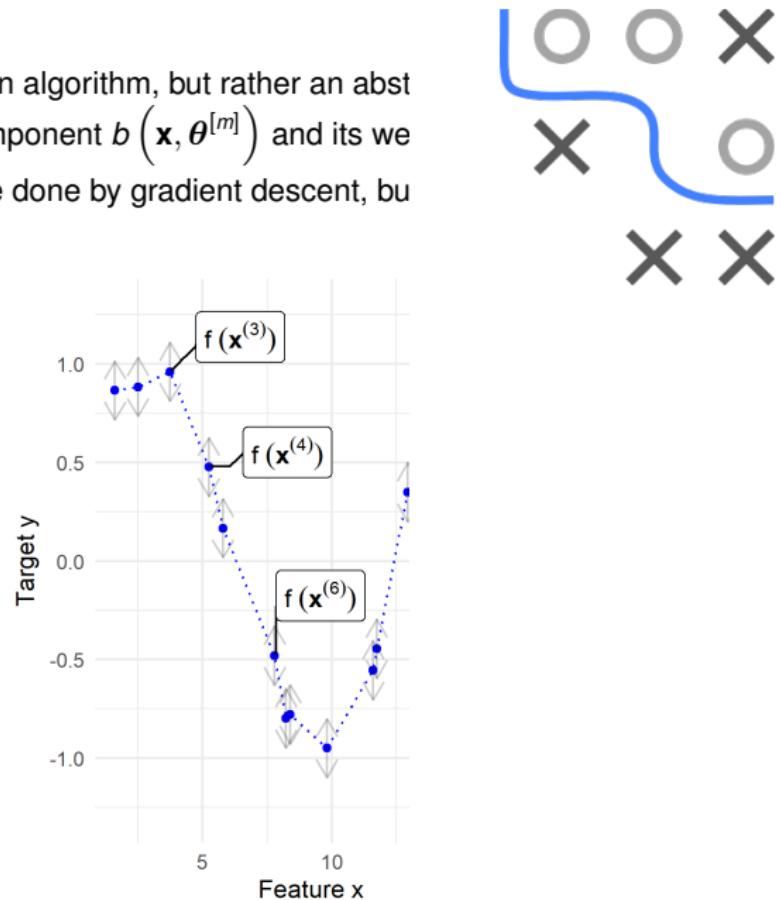
The algorithm we just introduced is not really an algorithm, but rather an abst principle. We need to find the new additive component $b(\mathbf{x}, \theta^{[m]})$ and its weight coefficient $\alpha^{[m]}$ in each iteration m . This can be done by gradient descent, but in function space.

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.

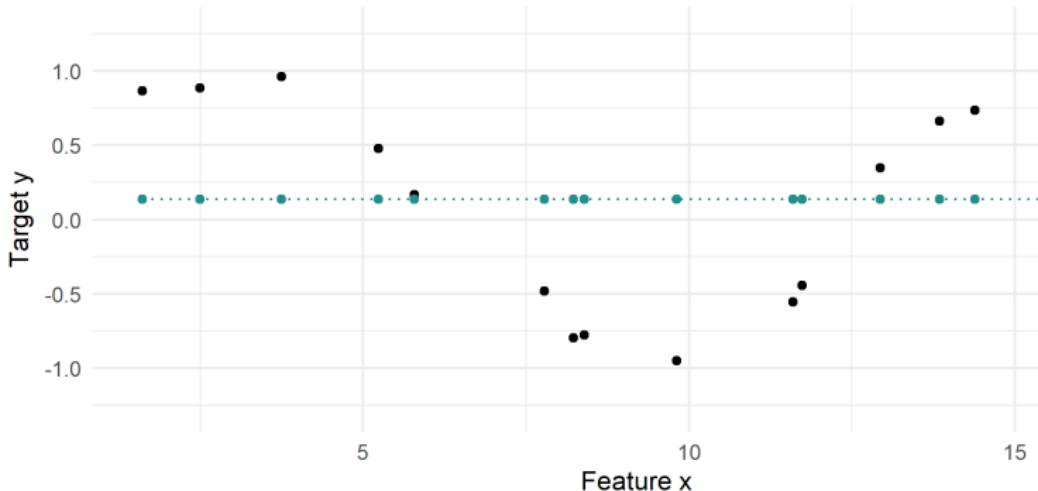
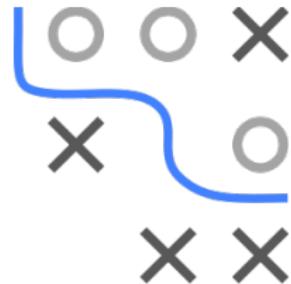


GRADIENT BOOSTING

Aim: Define a movement in function space so we can push our function towards the data points.

Given: Regression problem with one feature x and target varia

Initialization: Set all parameters to the optimal constant value
the mean of y for $L2$.



PSEUDO RESIDUALS

How do we have to distort this function to move it towards the observations a loss down?

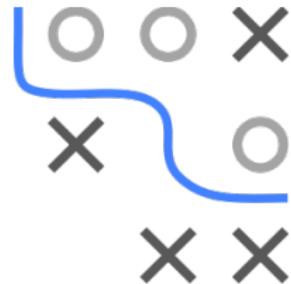
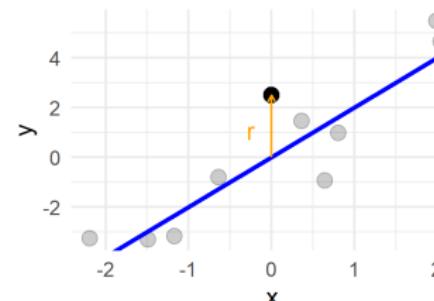
We minimize the risk of such a model with gradient descent (yes, this makes suspend all doubts for a few seconds).

So, we calculate the gradient at a point of the parameter space, that is, the de w.r.t. each component of the parameter vector (which is 0 for all terms with i :

$$\tilde{r}^{(i)} = -\frac{\partial \mathcal{R}_{\text{emp}}}{\partial f(\mathbf{x}^{(i)})} = -\frac{\partial \sum_j L(y^{(j)}, f(\mathbf{x}^{(j)}))}{\partial f(\mathbf{x}^{(i)})} = -\frac{\partial L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)}{\partial f(\mathbf{x}^{(i)})}.$$

Reminder: The pseudo-residuals $\tilde{r}(f)$ match the usual residuals for the squared loss:

$$\begin{aligned}-\frac{\partial L(y, f(\mathbf{x}))}{\partial f(\mathbf{x})} &= -\frac{\partial 0.5(y - f(\mathbf{x}))^2}{\partial f(\mathbf{x})} \\ &= y - f(\mathbf{x})\end{aligned}$$



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, 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:

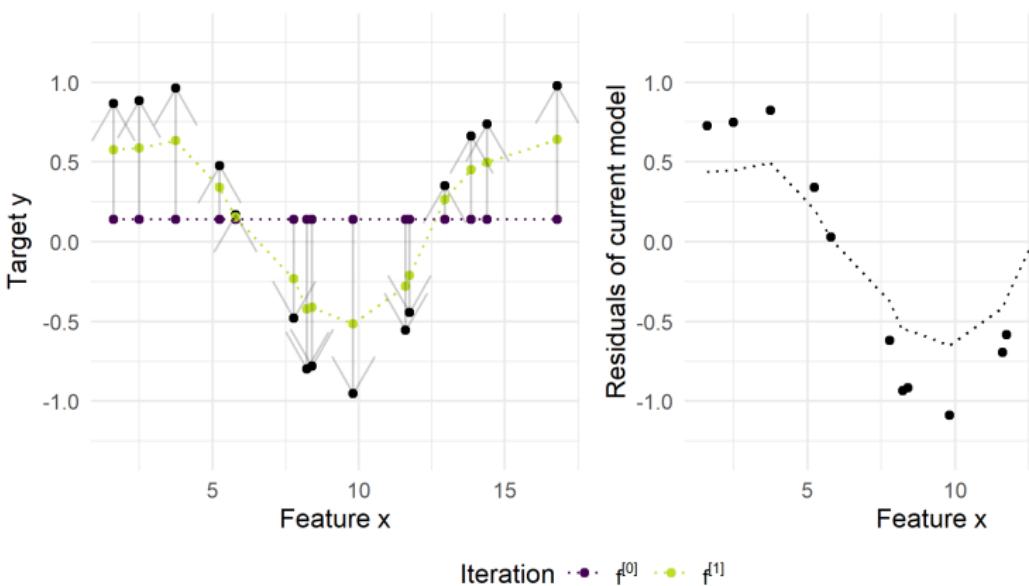
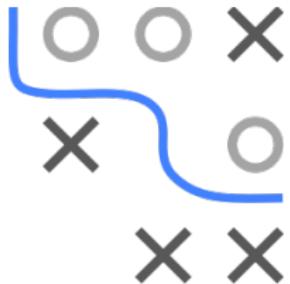
$$f^{[m]}(\mathbf{x}^{(i)}) = f^{[m-1]}(\mathbf{x}^{(i)}) - \alpha \frac{\partial L(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f^{[m-1]}(\mathbf{x}^{(i)})}.$$

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:

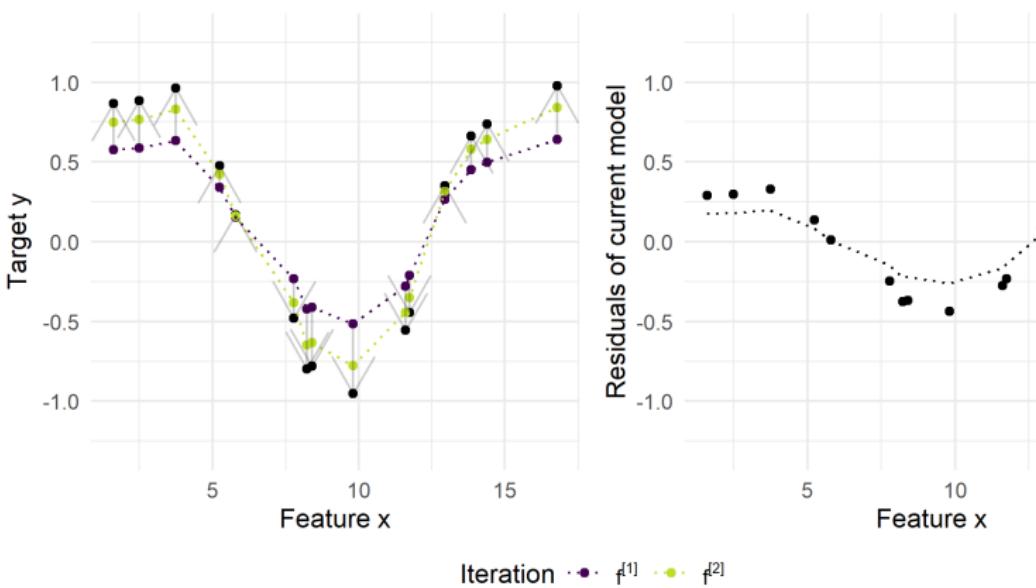
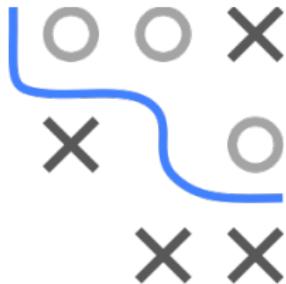
Let's move our function $f(\mathbf{x}^{(i)})$ a fraction towards the pseudo-residuals with a learning rate of $\alpha = 0.6$.



GRADIENT BOOSTING

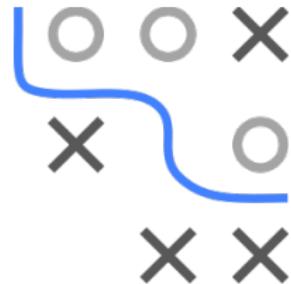
Iteration 2:

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.

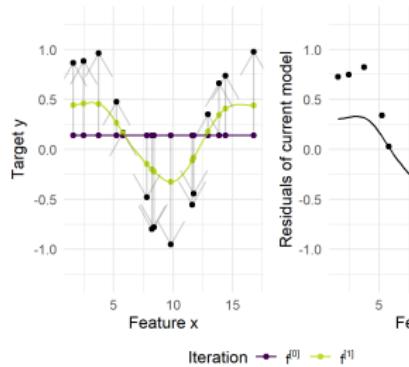


So, we restrict our additive components to $b(\mathbf{x}, \theta^{[m]}) \in \mathcal{B}$.

The pseudo-residuals are calculated exactly as stated above, then we fit a simple model $b(\mathbf{x}, \theta^{[m]})$ to them:

$$\hat{\theta}^{[m]} = \arg \min_{\theta} \sum_{i=1}^n \left(\tilde{r}^{[m](i)} - b(\mathbf{x}^{(i)}, \theta) \right)^2.$$

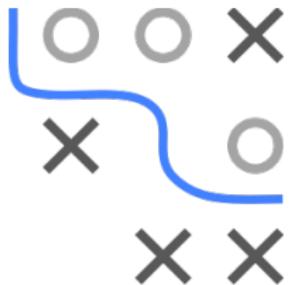
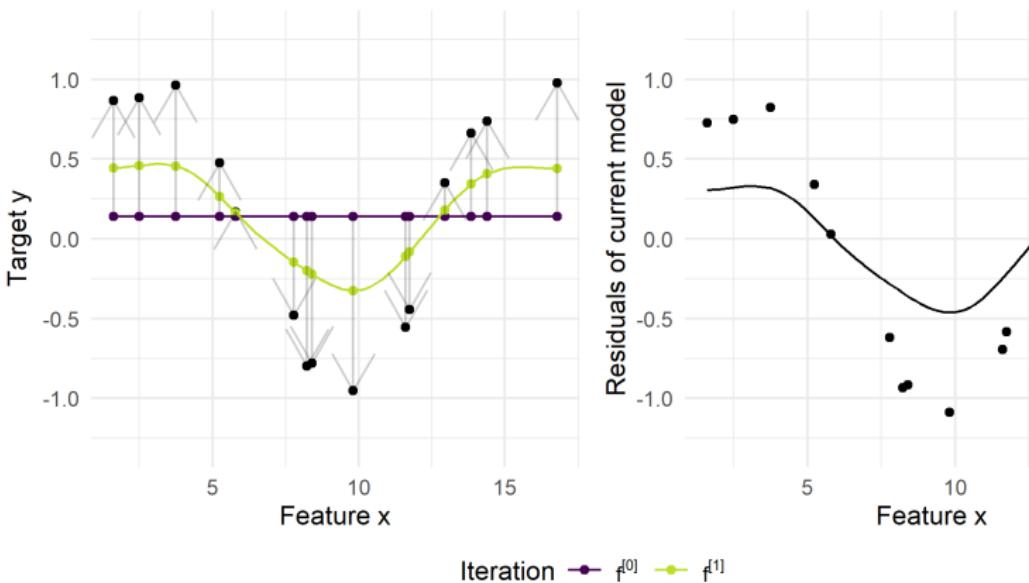
So, evaluated on the training data, our $b(\mathbf{x}, \theta^{[m]})$ corresponds as closely as possible to the negative loss function gradient and generalizes over the whole space.



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

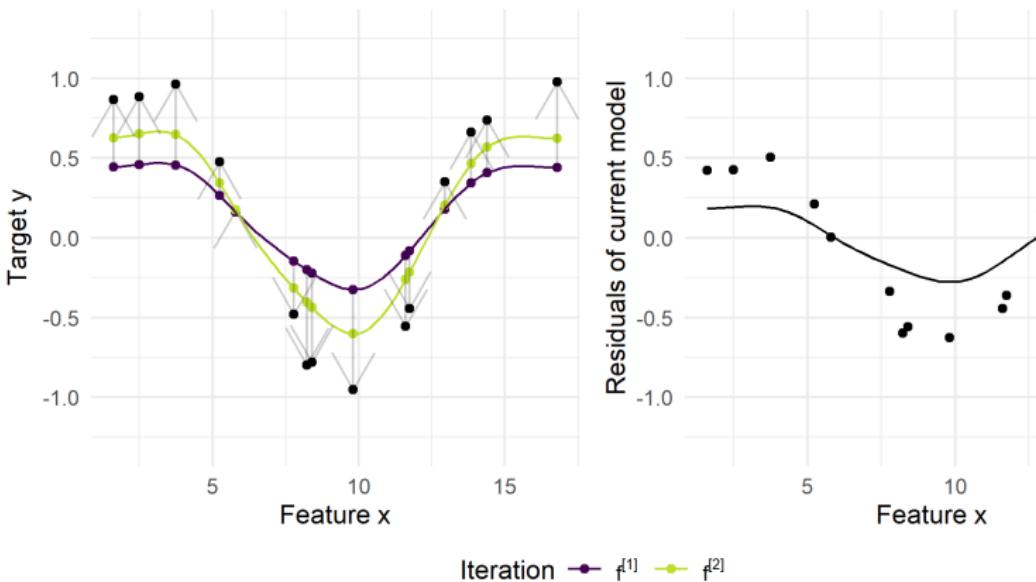
Iteration 1:



GRADIENT BOOSTING

Instead of moving the function values for each observation by a fraction close to the observed data, we fit a regression base learner to the pseudo-residuals (right).

Iteration 2:

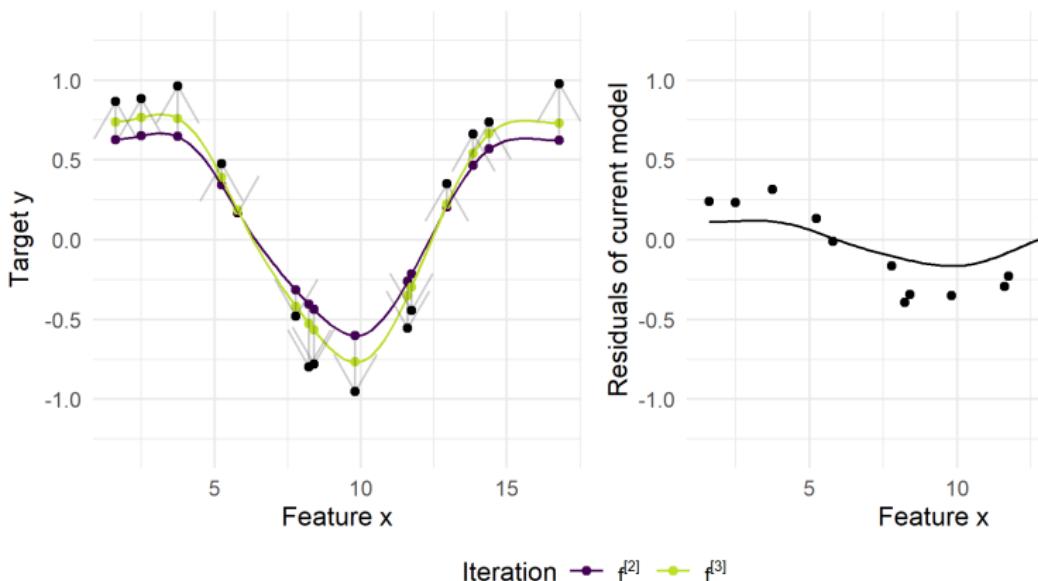


GRADIENT BOOSTING

This base learner is then added to the current state of the ensemble weighted 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 2 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 common in practice, it can also be replaced by a line search.



Line search is an iterative approach to find a local minimum. In order to set the learning rate, the following one-dimensional optimization problem has to be solved:

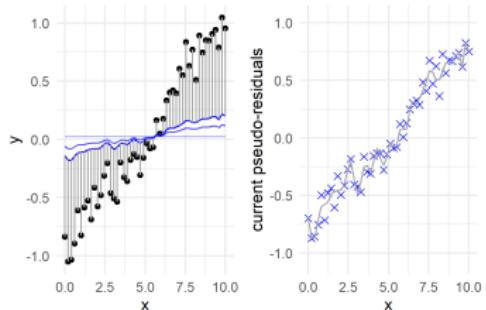
$$\hat{\alpha}^{[m]} = \arg \min_{\alpha} \sum_{i=1}^n L(y^{(i)}, f^{[m-1]}(\mathbf{x}) + \alpha b(\mathbf{x}, \hat{\theta}^{[m]}))$$

Optionally, an (inexact) backtracking line search can be used to find $\alpha^{[m]}$ 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.

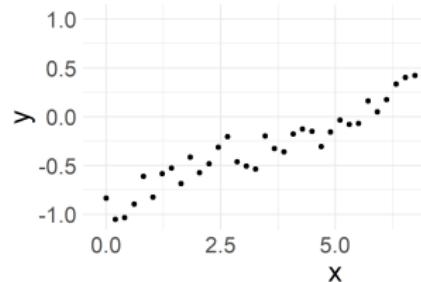


- L2: Init = optimal constant = $\text{mean}(y)$; for L1 it's $\text{median}(y)$
- BLs are cubic B -splines with 40 knots.
- PRs L_2 : $\tilde{r}(f) = r(f) = y - f(\mathbf{x})$
- PRs L_1 : $\tilde{r}(f) = \text{sign}(y - f(\mathbf{x}))$
- Constant learning rate 0.2

Univariate toy data:

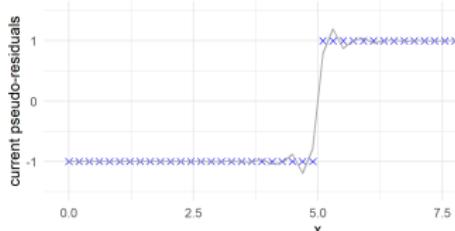
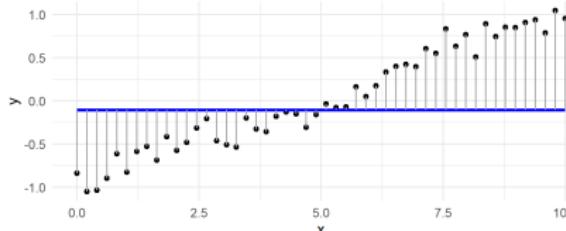
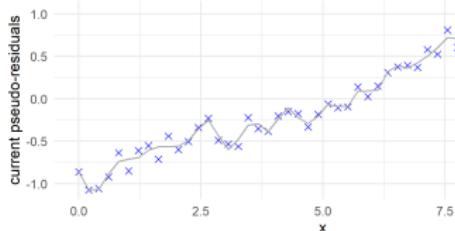
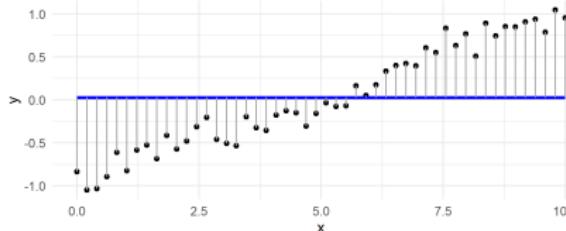
$$y^{(i)} = -1 + 0.2 \cdot x^{(i)} + 0.1 \cdot \sin(x^{(i)}) + \epsilon^{(i)}$$

$$n = 50 ; \epsilon^{(i)} \sim \mathcal{N}(0, 0.1)$$



GAM WITH L_2 VS L_1 LOSS

Top: L_2 loss, bottom: L_1 loss

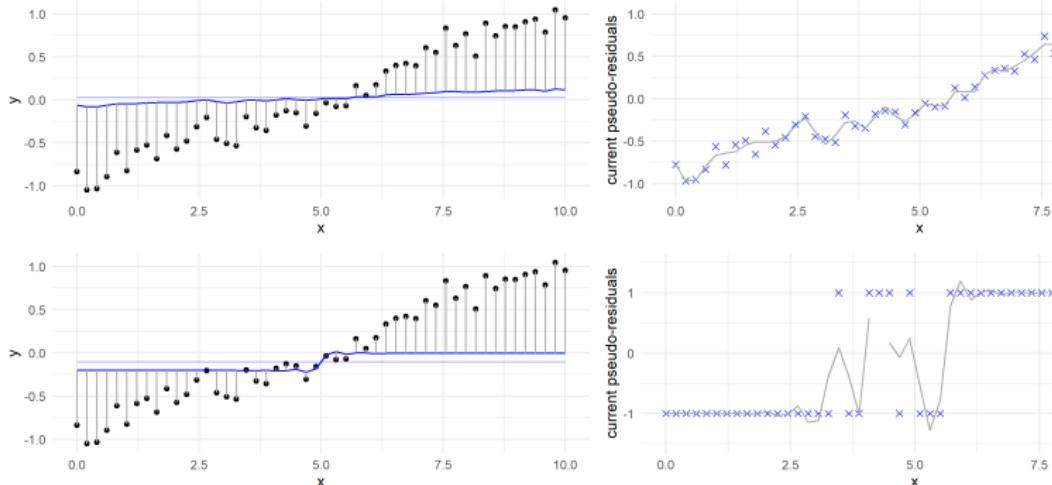


Iteration 1

Shape of PRs affects gradual model fit: L_1 only sees resids' sign, BLs are no 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

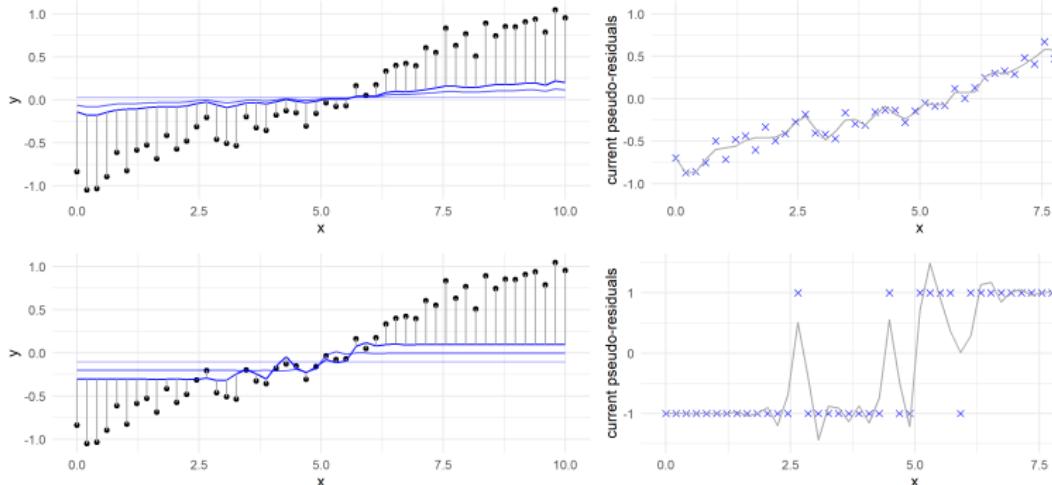


Iteration 2

Shape of PRs affects gradual model fit: L_1 only sees resids' sign, BLs are no 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

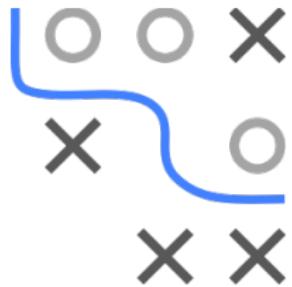
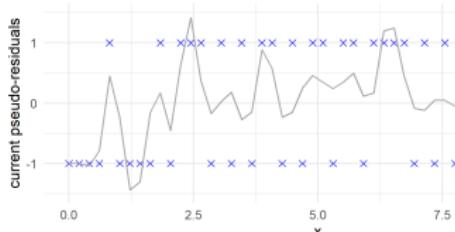
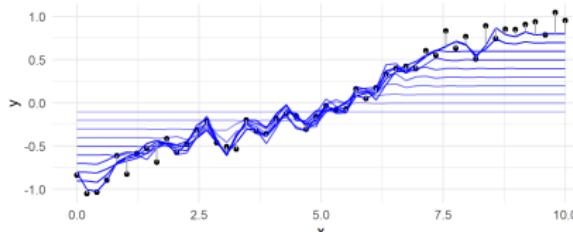
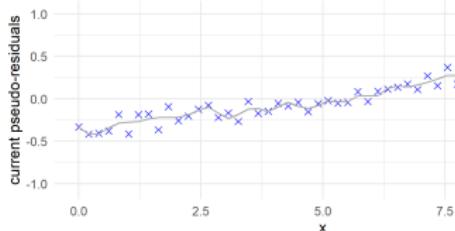
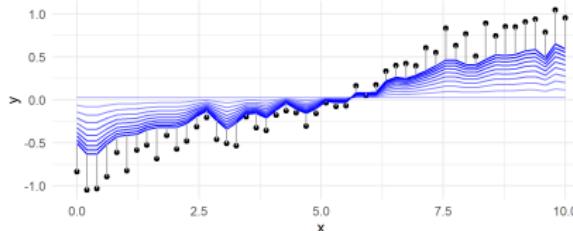


Iteration 3

Shape of PRs affects gradual model fit: L_1 only sees resids' sign, BLs are no size of values as in L_2 and hence lead to more moderate changes.

GAM WITH $L2$ VS $L1$ LOSS

Top: $L2$ loss, bottom: $L1$ loss

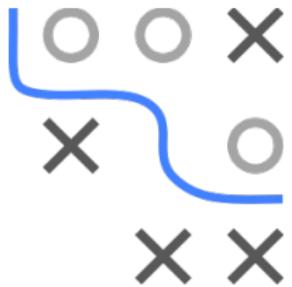
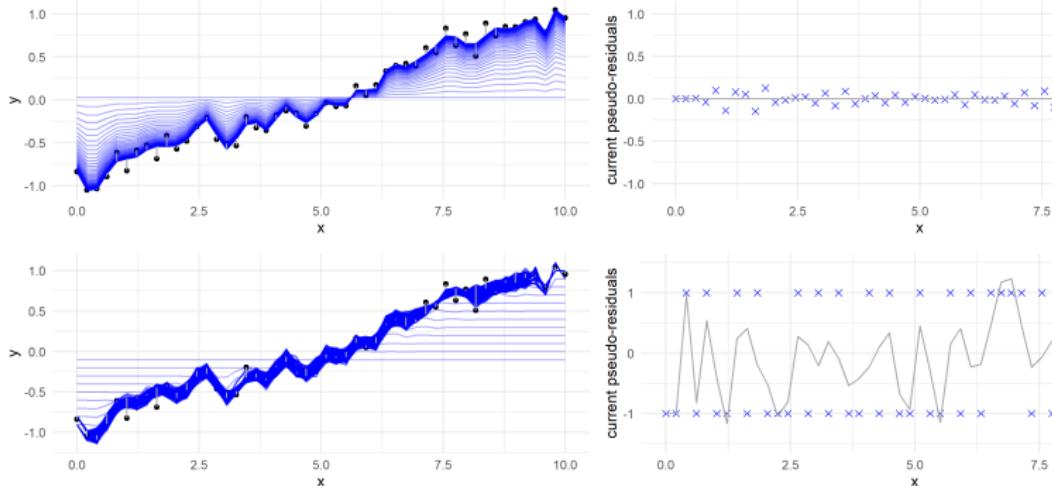


Iteration 10

Shape of PRs affects gradual model fit: $L1$ only sees resids' sign, BLs are no size of values as in $L2$ and hence lead to more moderate changes.

GAM WITH L_2 VS L_1 LOSS

Top: L_2 loss, bottom: L_1 loss

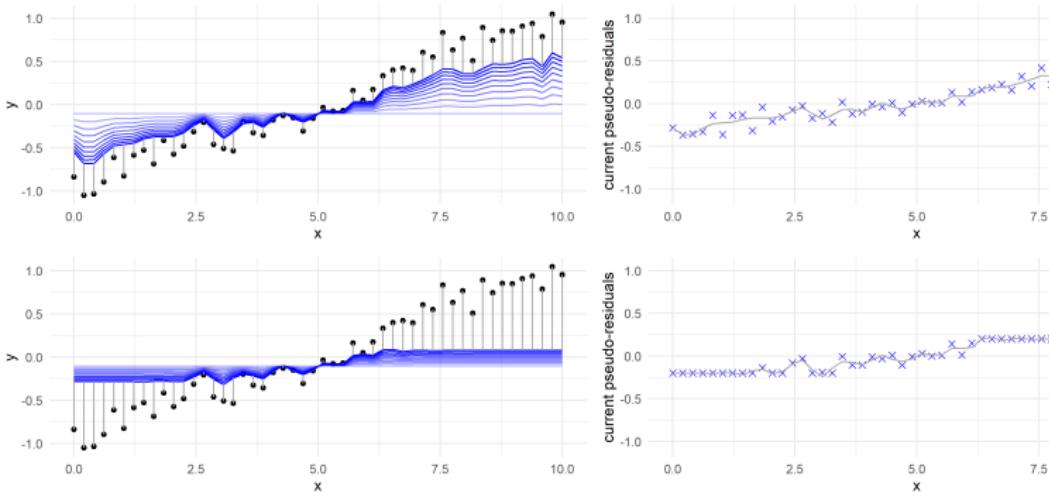


Iteration 100

Shape of PRs affects gradual model fit: L_1 only sees resids' sign, BLs are no size of values as in L_2 and hence lead to more moderate changes.

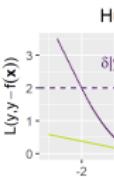
GAM WITH HUBER LOSS

Top: $\delta = 2$, bottom: $\delta = 0.2$.



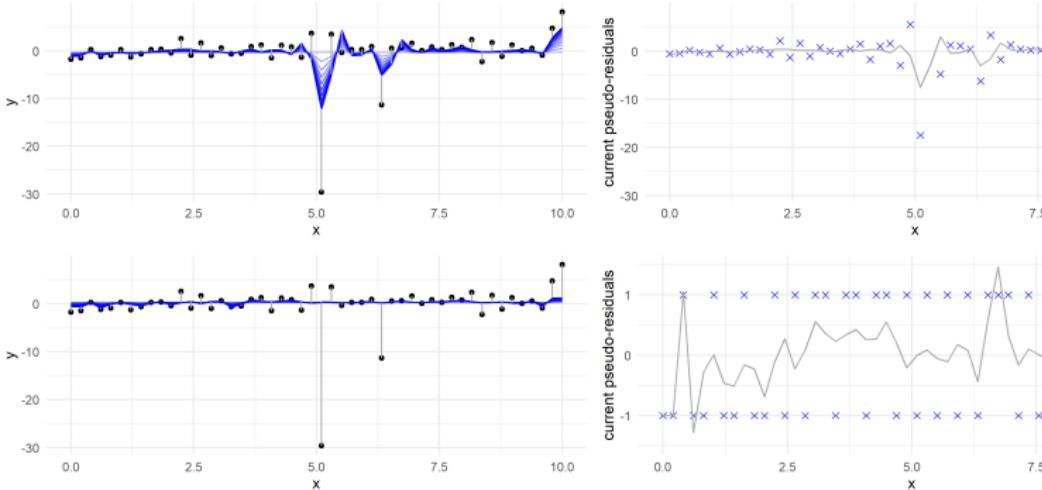
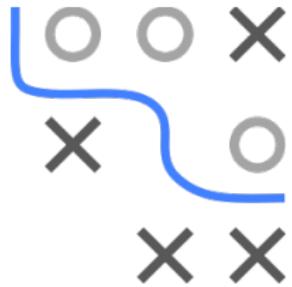
Iteration 10

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.
Top: L_2 , bottom: L_1 .

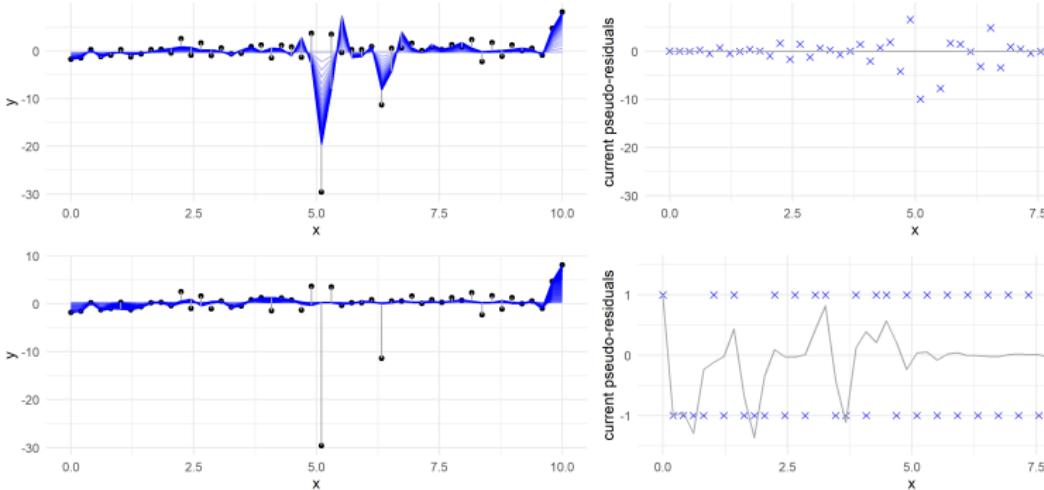
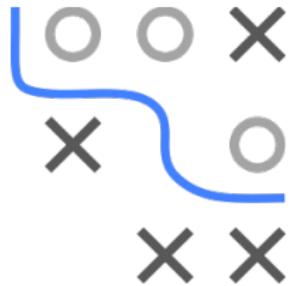


Iteration 10

L_2 loss is affected by outliers rather strongly, whereas L_1 solely considers residual 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
Top: L_2 , bottom: L_1 .

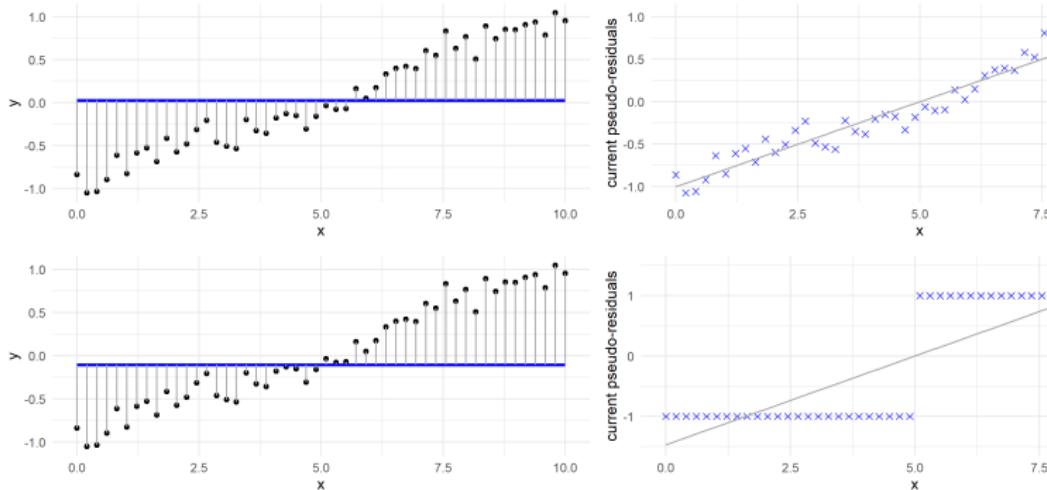


Iteration 100

L_2 loss is affected by outliers rather strongly, whereas L_1 solely considers residual 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 .

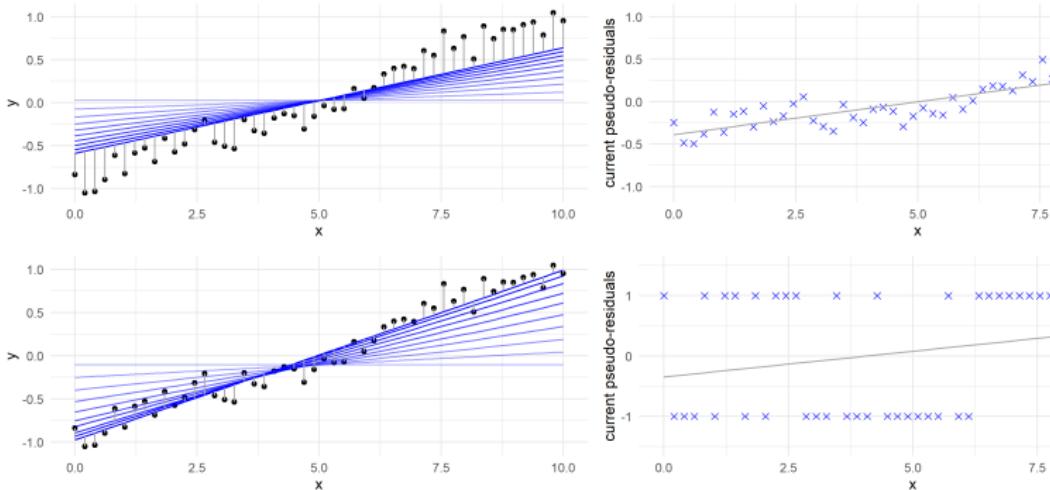


Iteration 1

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 .

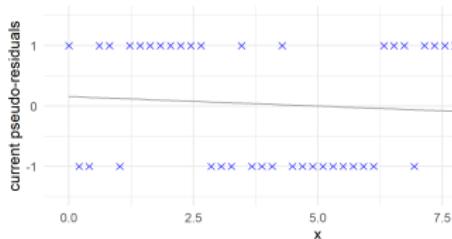
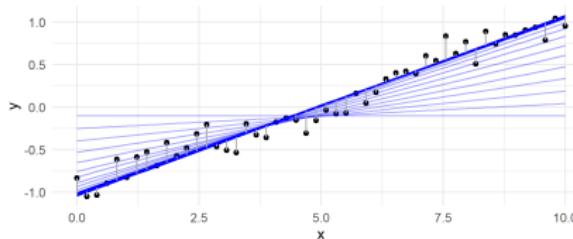
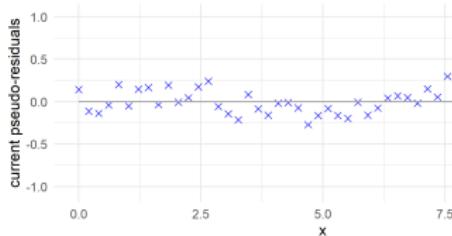
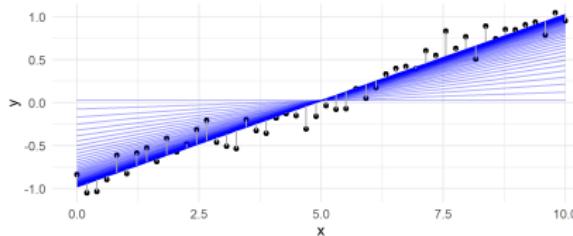


Iteration 10

L_2 : as $\tilde{r}(f) = r(f)$, BL of 1st iter already optimal; but learn rate slows us down

LM WITH L_2 VS L_1 LOSS

Top: L_2 , bottom: L_1 .



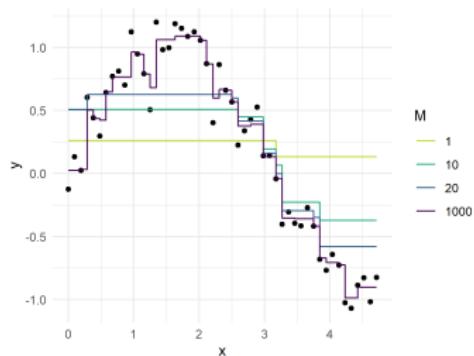
Iteration 100

L_2 : 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 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:

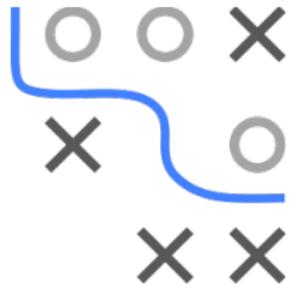
- Limit nr of iters M , i.e., additive components (“early stopping”).
- Limit depth of trees. Can also be interpreted as choosing the number of interactions (see later).
- Use a small learn rate α for only mild model updates.
 α a.k.a. shrinkage.

Practical hints:

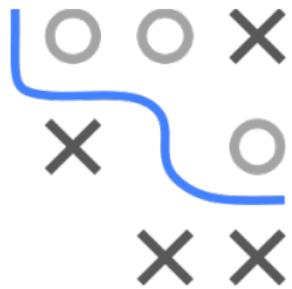
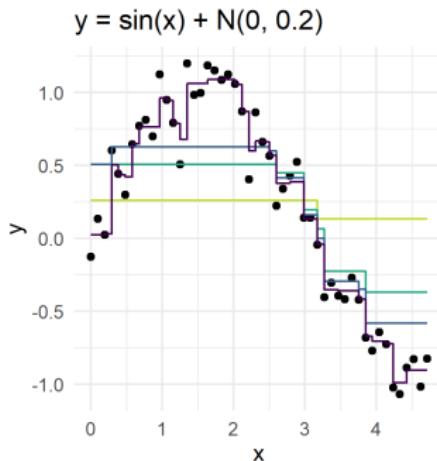
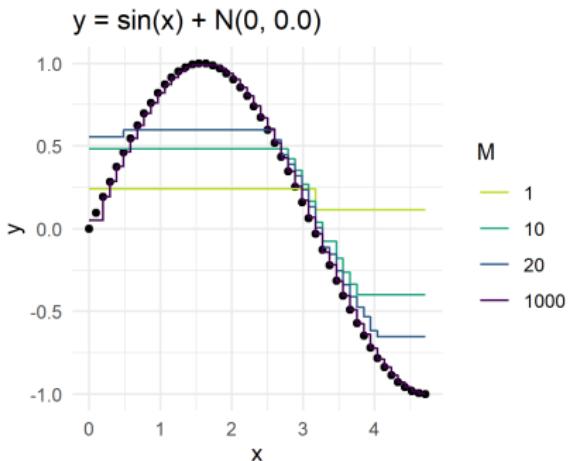
- Optimal values for M and α strongly depend on each other: if one increases M one can use a smaller value for α and vice versa.
- Fast option = Make α small and choose M by CV.
- Probably best to tune all 3 hyperpars jointly via, e.g., CV.

STOCHASTIC GRADIENT BOOSTING

- Minor modification to incorporate the advantages of bagging
- In each iter, we only fit on a random subsample of the train
- Especially for small train sets, this often leads to better results
- Size of random sets = new hyperparameter



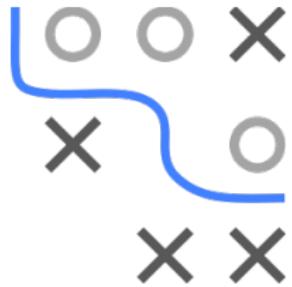
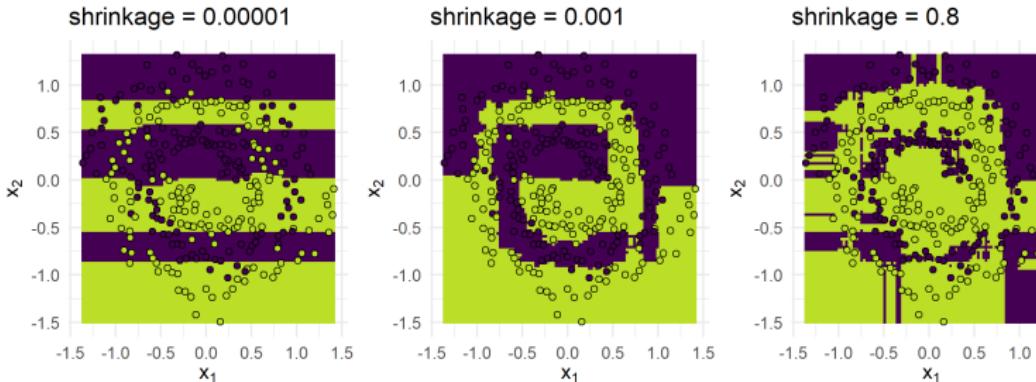
EXAMPLE: SINUSOIDAL WITH TREE STUMP



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



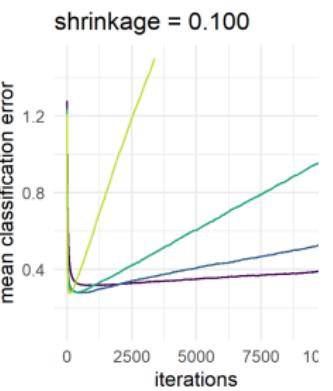
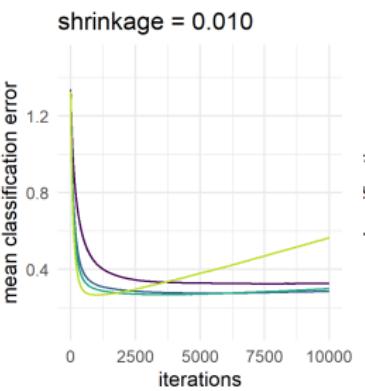
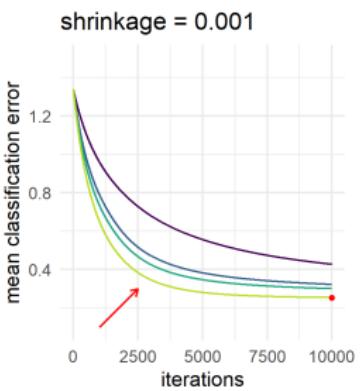
We observe an oversmoothing effect in the left scenario with strong regularization (i.e., very small learning rate) and overfitting when regularization is too weak (right). $\alpha = 0.001$ yields a pretty goo

EXAMPLE: SPAM DETECTION WITH TREES



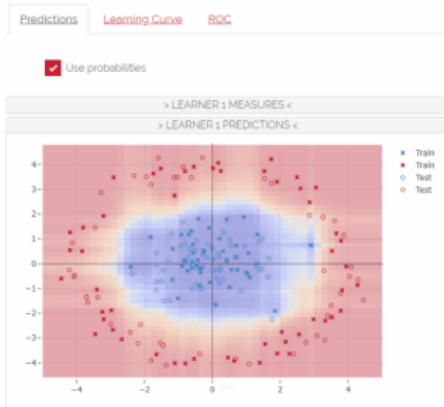
Hyperpar	Range
Loss	Bernoulli (for classification)
Number of trees M	$\{0, 1, \dots, 10000\}$
Shrinkage α	$\{0.001, 0.01, 0.1\}$
Max. tree depth	$\{1, 3, 5, 20\}$

Use 3-CV in grid search; optimal config in red:



Introduction to Machine Learning

Gradient Boosting for 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 effectively, the pseudo-residuals are $y - \pi(\mathbf{x})$.

Through $\pi(\mathbf{x}) = s(f(\mathbf{x}))$ we can also estimate posterior probab

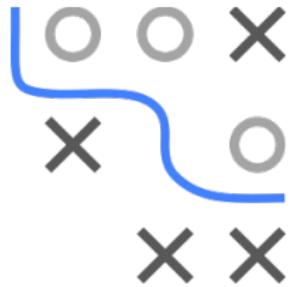
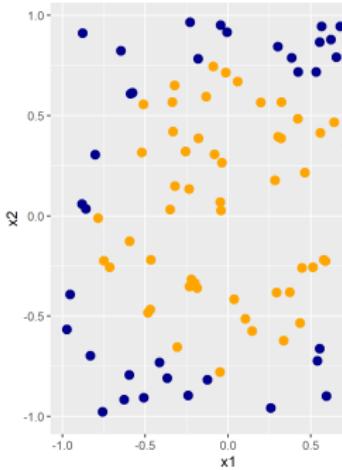
BINARY CLASSIFICATION

- Rest works as in regression.
- NB: We fit regression BLs against the PRs with $L2$ loss.
- Exponential loss works too. In practice there is no big difference although Bernoulli loss makes a bit more sense from a the (maximum likelihood) perspective.
- It can be shown GB with exp loss is basically equivalent to generalizes AdaBoost.



EXAMPLE: 2D CIRCLE DATA

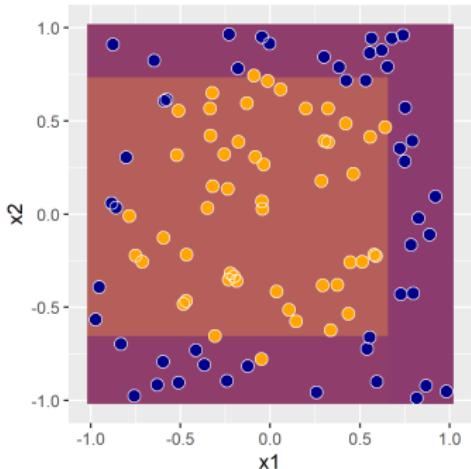
- mlbench circle data with $n = 100$
- Bernoulli loss
- BL = shallow tree with max. depth of 3
- We initialized with $f^{[0]} = 0$.



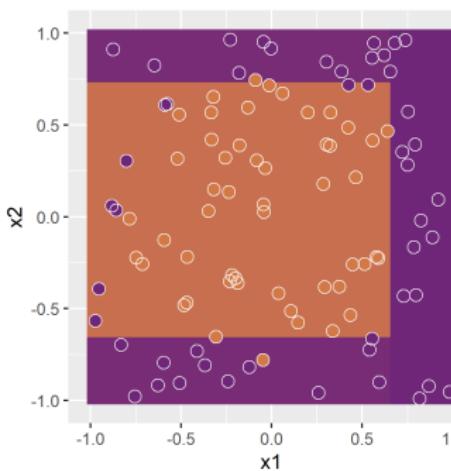
EXAMPLE: 2D CIRCLE DATA

BG color is predicted probs on LHS on RHS we show and pred

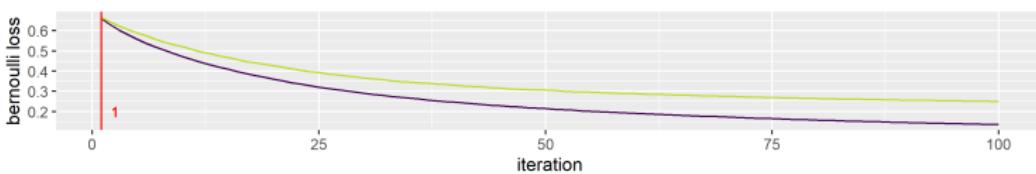
y and $\hat{\pi}(x)$



\tilde{r} and $\hat{b}^{[m]}$



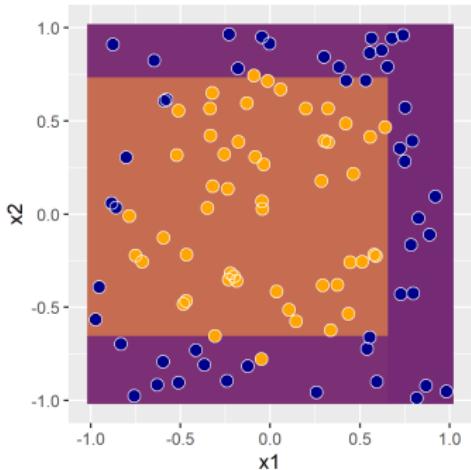
beroulli loss



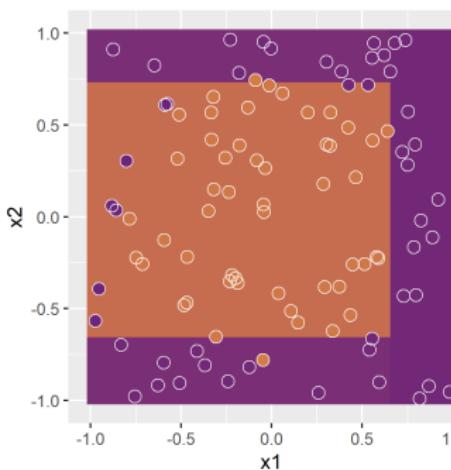
EXAMPLE: 2D CIRCLE DATA

BG color is predicted probs on LHS on RHS we show and pred

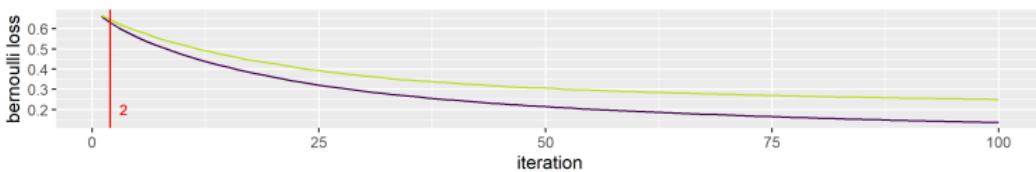
y and $\hat{\pi}(x)$



\tilde{r} and $\hat{b}^{[m]}$



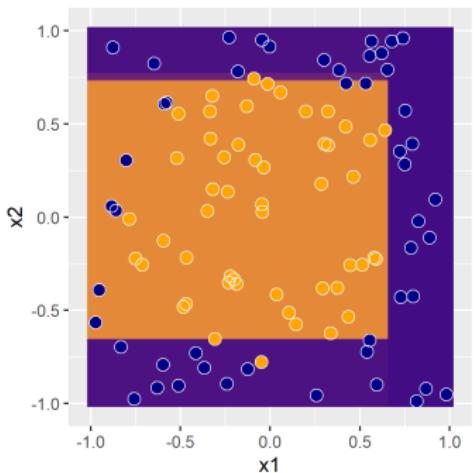
beroulli loss



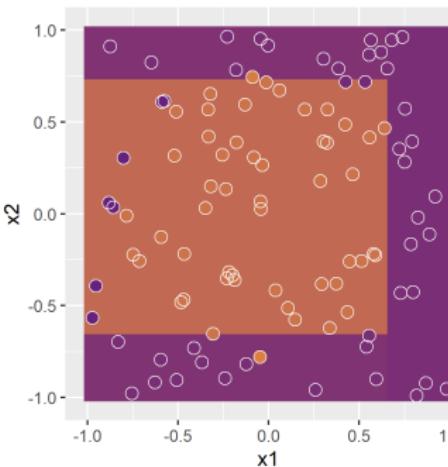
EXAMPLE: 2D CIRCLE DATA

BG color is predicted probs on LHS on RHS we show and pred

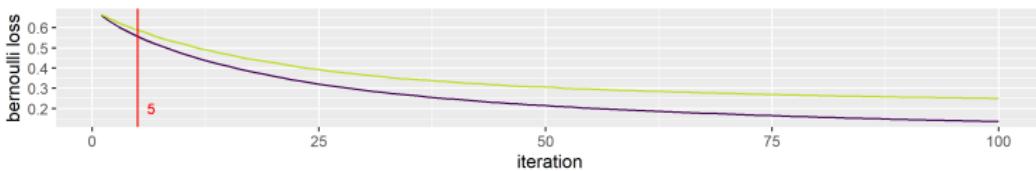
y and $\hat{\pi}(x)$



\tilde{r} and $\hat{b}^{[m]}$



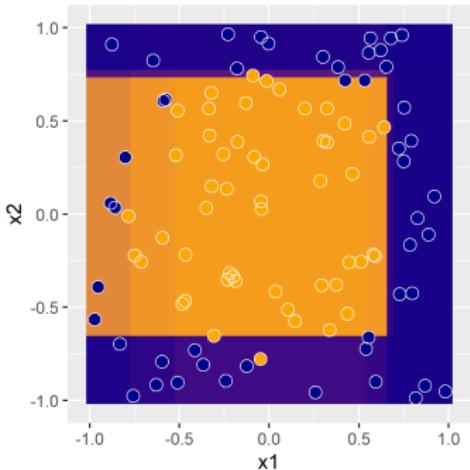
beroulli loss



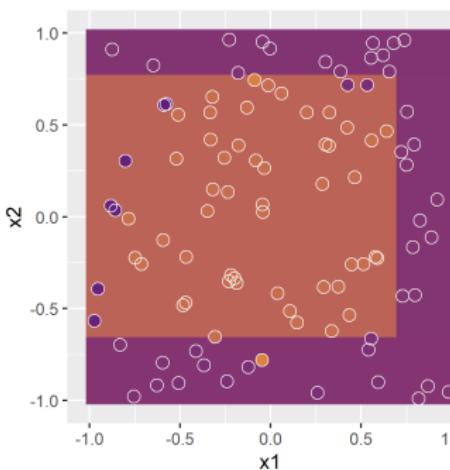
EXAMPLE: 2D CIRCLE DATA

BG color is predicted probs on LHS on RHS we show and pred

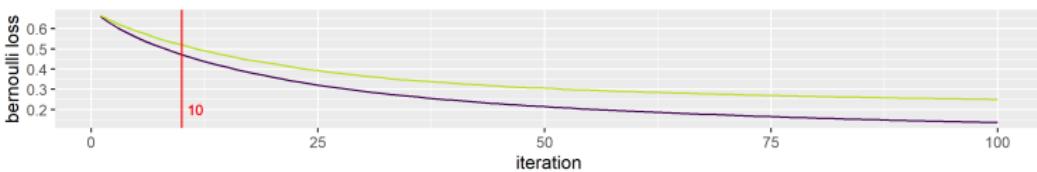
y and $\hat{\pi}(x)$



\tilde{r} and $\hat{b}^{[m]}$



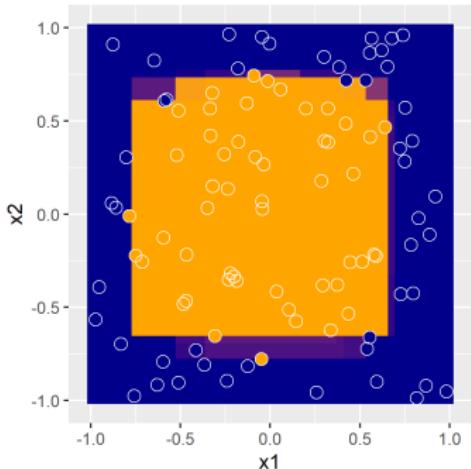
beroulli loss



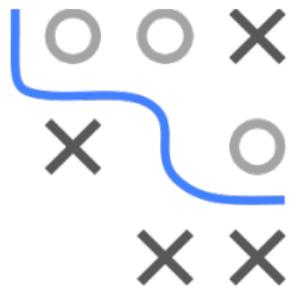
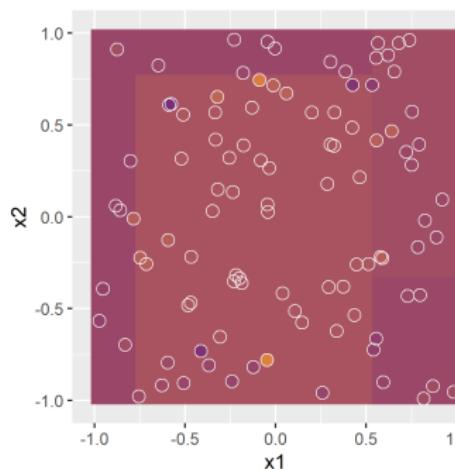
EXAMPLE: 2D CIRCLE DATA

BG color is predicted probs on LHS on RHS we show and pred

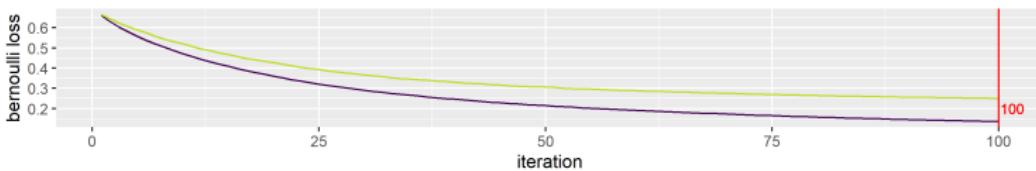
y and $\hat{\pi}(x)$



\tilde{r} and $\hat{b}^{[m]}$



beroulli loss



MULTICLASS PROBLEMS

We proceed as in softmax regression and model a categorical distribution with multinomial / log loss. For $\mathcal{Y} = \{1, \dots, g\}$, we have discriminant functions $f_k(\mathbf{x})$, one for each class and each one based on 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

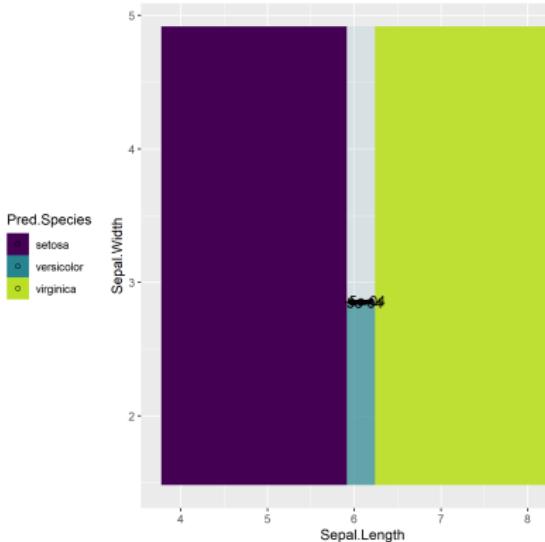
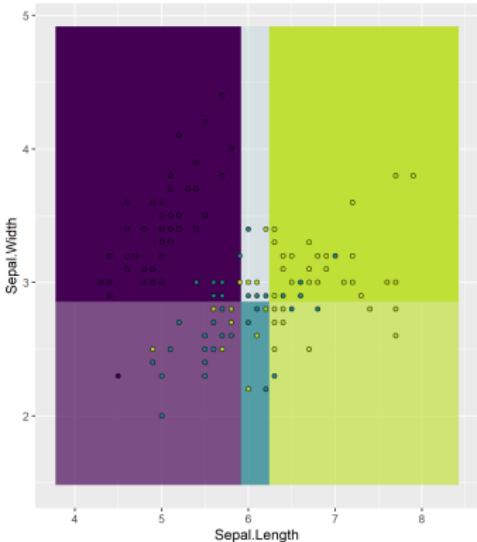
Algorithm 1 GB for Multiclass

```
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

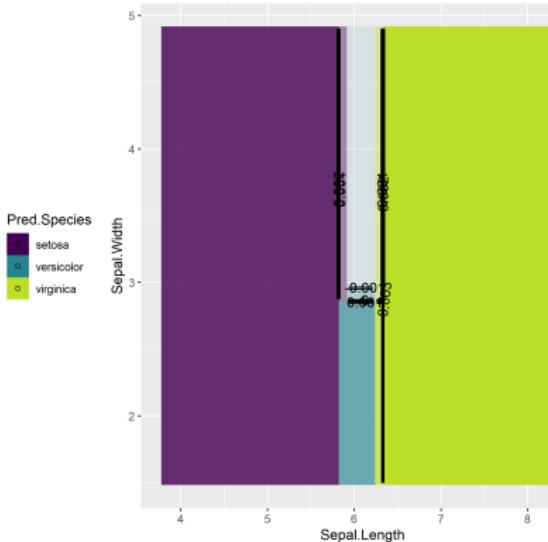
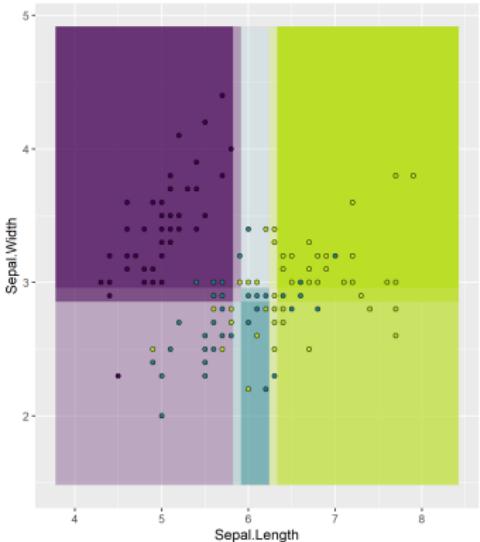
LHS: BG color is predicted probs and point col is true label; RH
Contour lines of discriminant functions.



Iteration=1

EXAMPLE: 2D IRIS

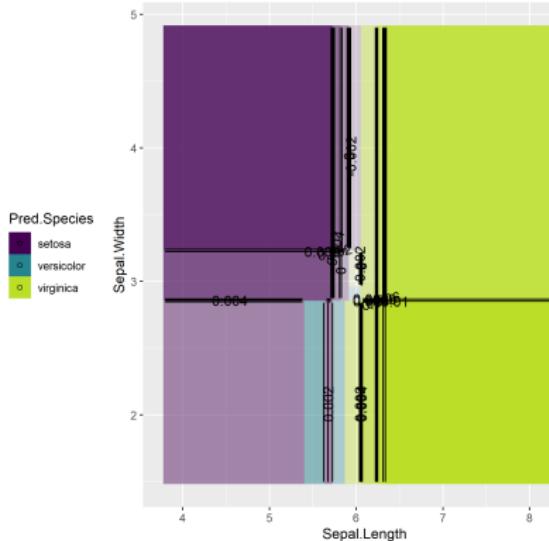
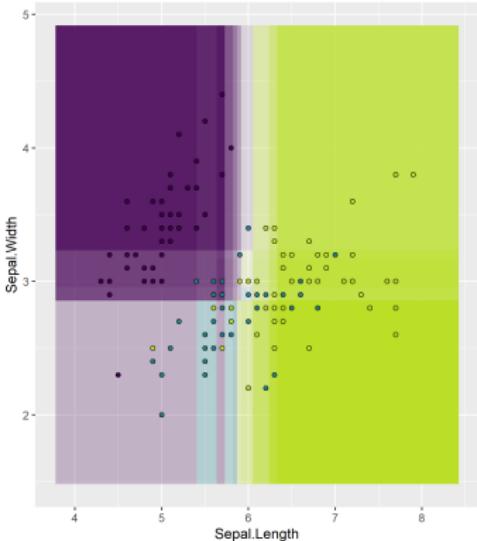
LHS: BG color is predicted probs and point col is true label; RH
Contour lines of discriminant functions.



Iteration=2

EXAMPLE: 2D IRIS

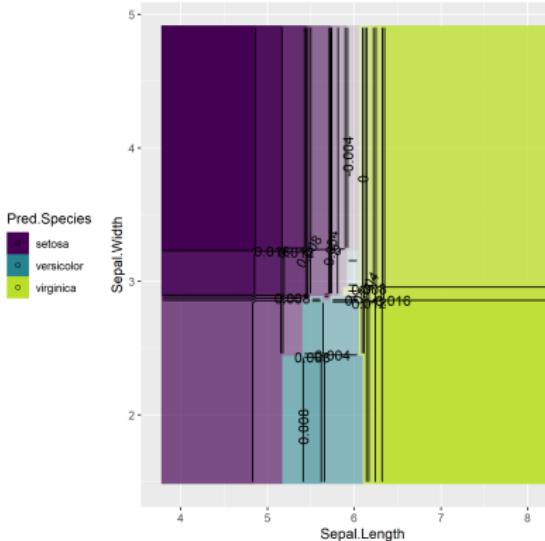
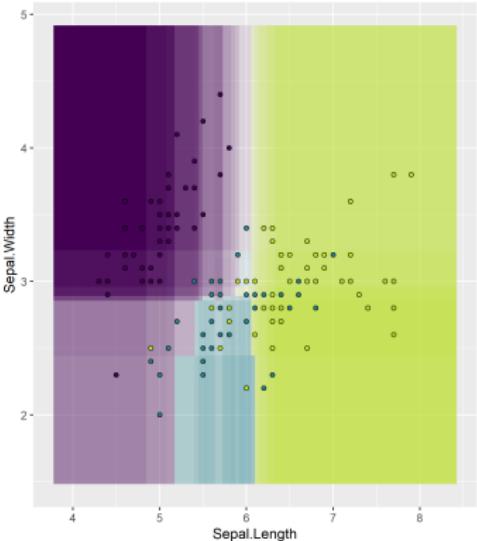
LHS: BG color is predicted probs and point col is true label; RH
Contour lines of discriminant functions.



Iteration=5

EXAMPLE: 2D IRIS

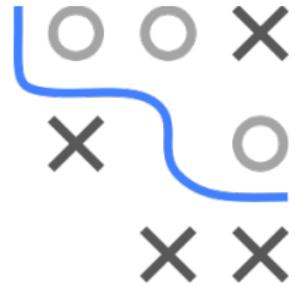
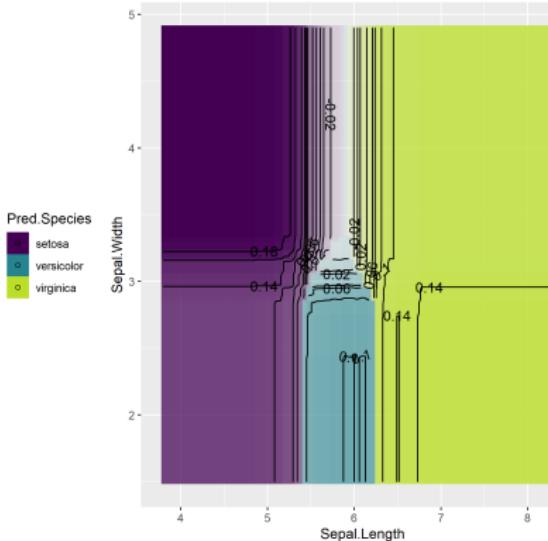
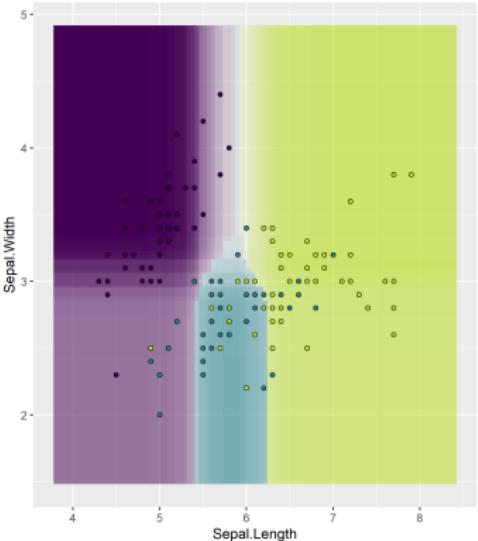
LHS: BG color is predicted probs and point col is true label; RH
Contour lines of discriminant functions.



Iteration=10

EXAMPLE: 2D IRIS

LHS: BG color is predicted probs and point col is true label; RH
Contour lines of discriminant functions.

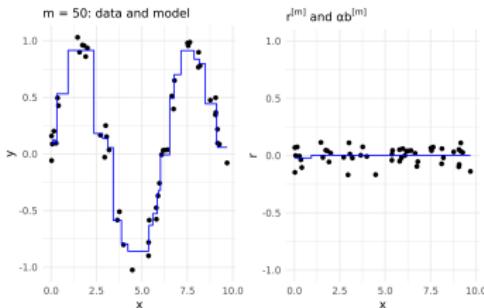


Iteration=100



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.

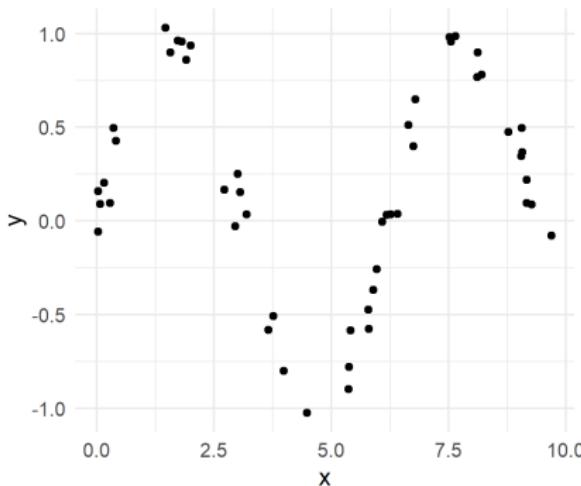
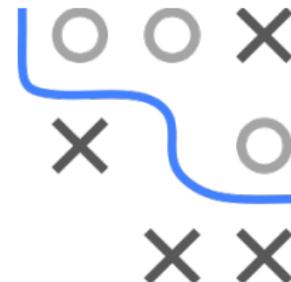
GB with trees inherits these, and strongly improves predictive p



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 $\epsilon^{(i)} \sim \mathcal{N}(0, 0.01)$
 $\forall i \in \{1, \dots, 50\}$.



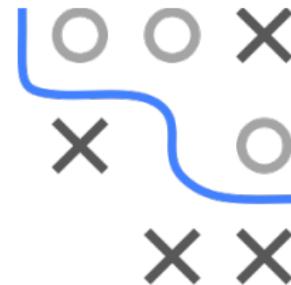
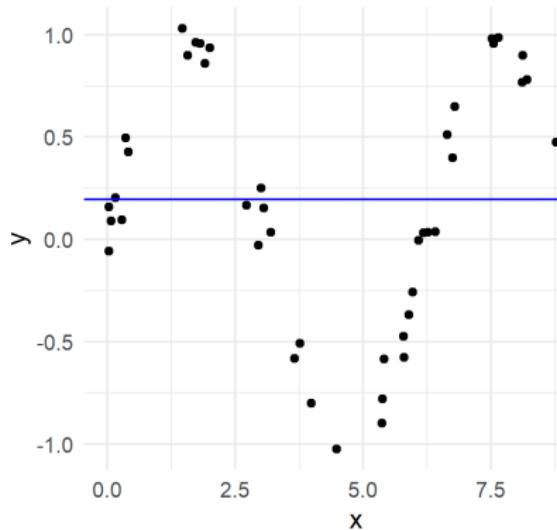
Aim: we want to fit a gradient boosting model to the data using stumps as base learners

Since we are facing a regression problem, we use L_2 loss.

EXAMPLE 1

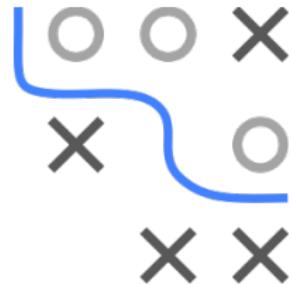
Iteration 0: initialization by optimal constant (mean) prediction $\hat{f}^{[0]}(x) = \bar{y}$:

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

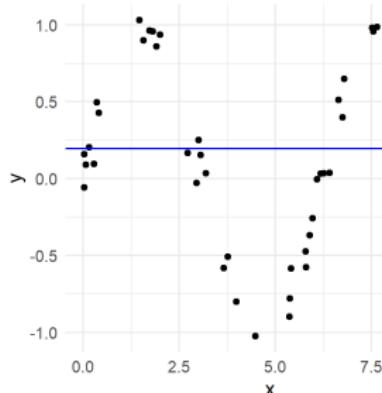


EXAMPLE 1

Iteration 1: (1) Calculate pseudo-residuals $\tilde{r}^{[m](i)}$ and (2) fit a regression stn



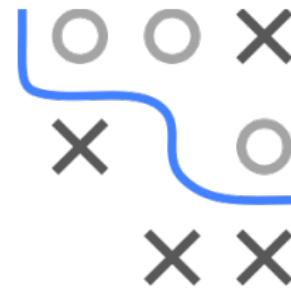
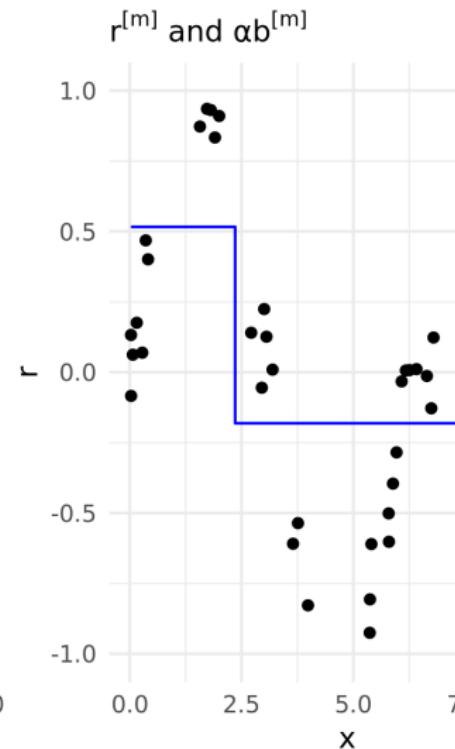
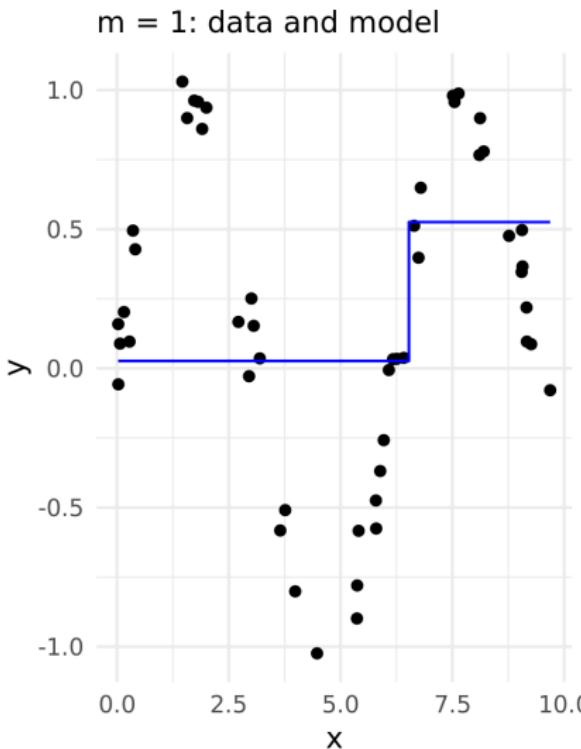
i	$x^{(i)}$	$y^{(i)}$	$\hat{f}^{[0]}$	$\tilde{r}^{[1](i)}$	$\hat{b}^{[1](i)}$
1	0.03	0.16	0.20	-0.04	-0.17
2	0.03	-0.06	0.20	-0.25	-0.17
3	0.07	0.09	0.20	-0.11	-0.17
:	:	:	:	:	:
50	9.69	-0.08	0.20	-0.27	0.33



(3) Update model by $\hat{f}^{[1]}(x) = \hat{f}^{[0]}(x) + \hat{b}^{[1]}$.

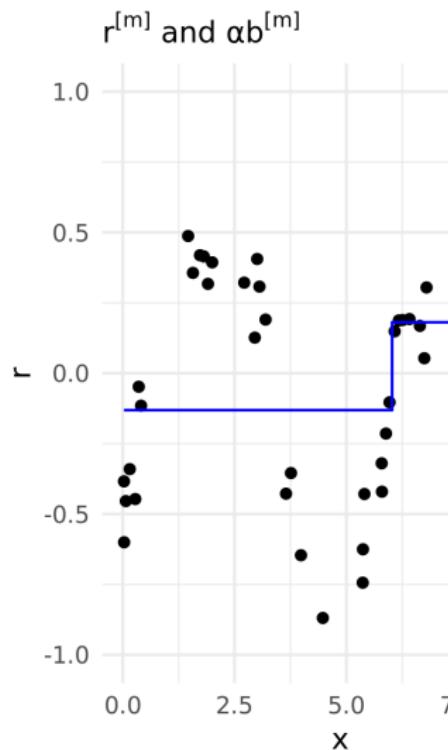
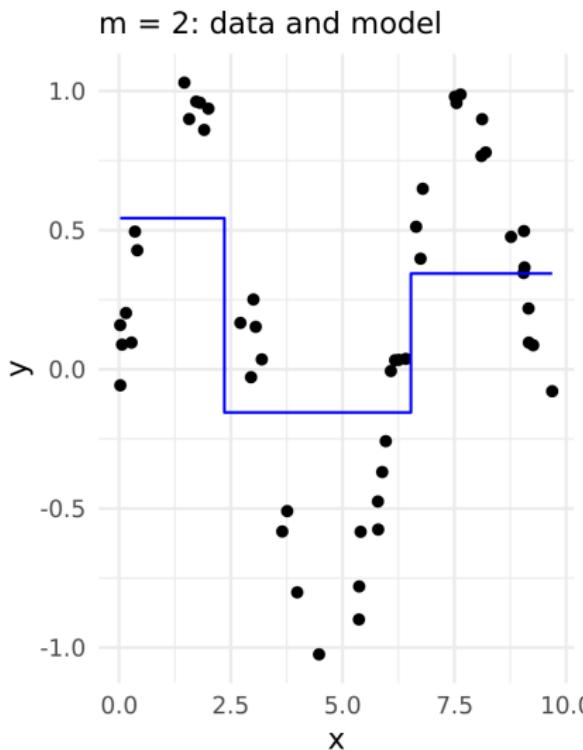
EXAMPLE 1

Repeat step (1) to (3):



EXAMPLE 1

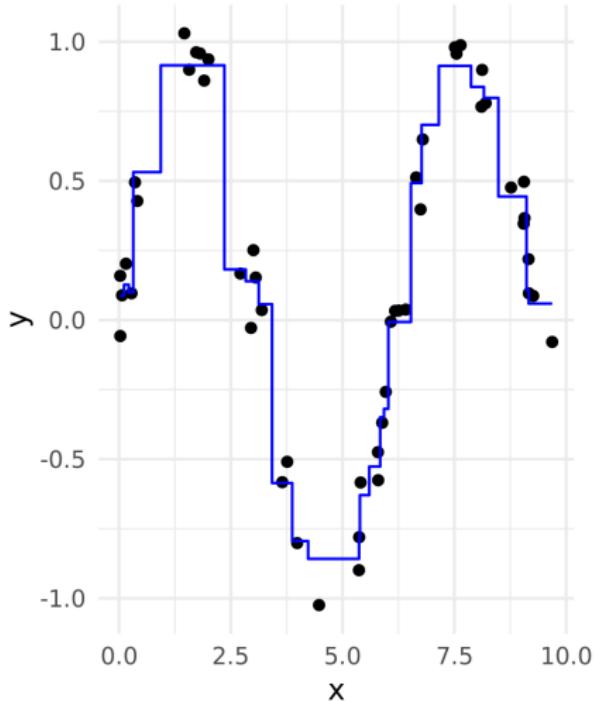
Repeat step (1) to (3):



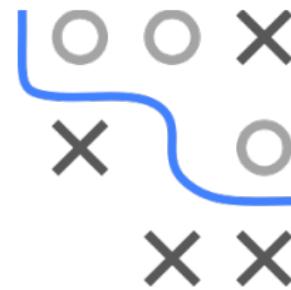
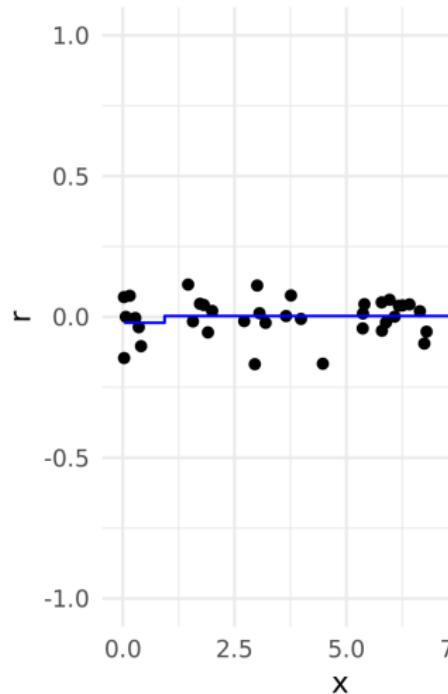
EXAMPLE 1

Repeat step (1) to (3):

$m = 50$: data and model

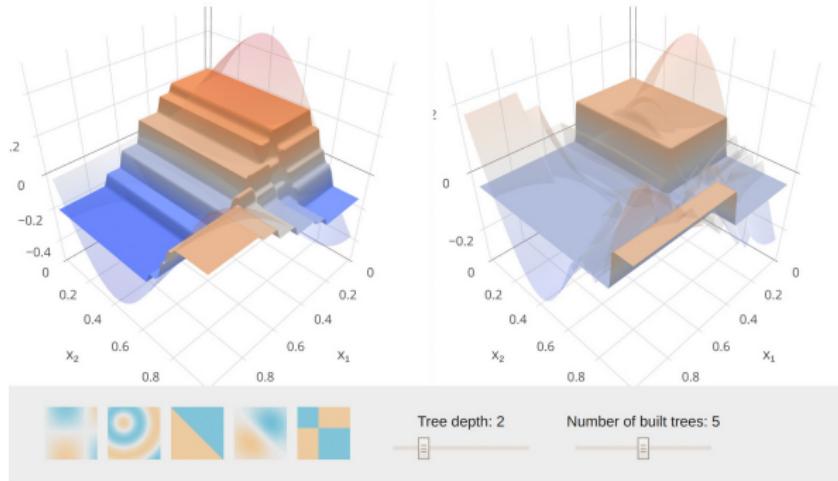


$r^{[m]}$ and $\alpha b^{[m]}$



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 DE

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:

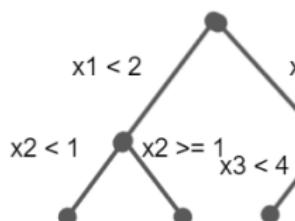
$$f(\mathbf{x}) = f_0 + \sum_{j=1}^p f_j(x_j)$$



With trees of depth 2, we have two-way interactions:

$$f(\mathbf{x}) = f_0 + \sum_{j=1}^p f_j(x_j) + \sum_{j \neq k} f_{j,k}(x_j, x_k)$$

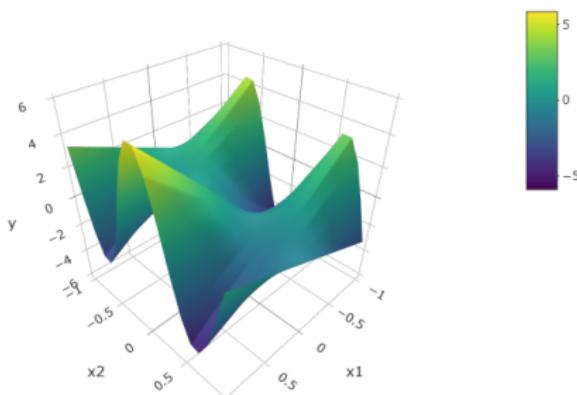
with f_0 being a constant intercept.



MODEL STRUCTURE AND INTERACTION DE

Simulation setting:

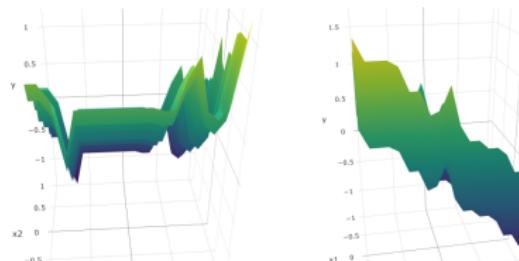
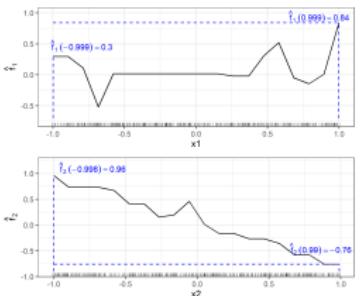
- Features x_1 and x_2 and numeric y ; with $n = 500$
- x_1 and x_2 are uniformly distributed between -1 and 1
- $y^{(i)} = x_1^{(i)} - x_2^{(i)} + 5 \cos(5x_2^{(i)}) \cdot x_1^{(i)} + \epsilon^{(i)}$ with $\epsilon^{(i)} \sim \mathcal{N}(0, 1)$
- We fit 2 GB models, with tree depth 1 and 2, respectively.



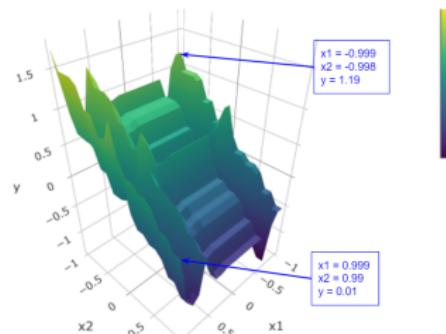
MODEL STRUCTURE AND INTERACTION DE

GBM with interaction depth of 1 (GAM)

No interactions are modelled: Marginal effects of x_1 and x_2 add joint effect (plus the constant intercept $\hat{f}_0 = -0.07$).



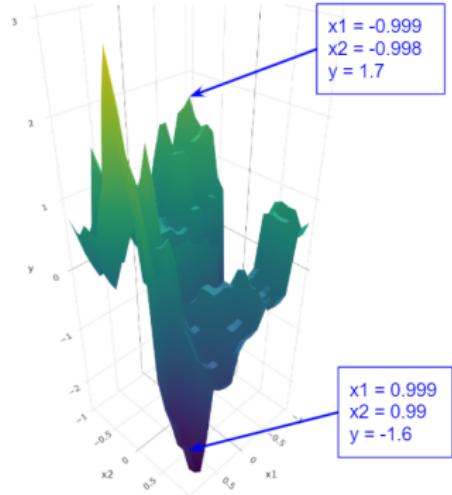
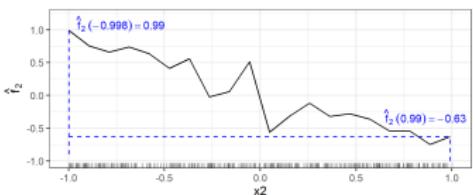
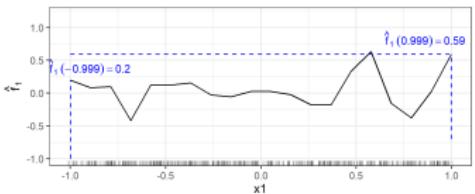
$$\begin{aligned}\hat{f}(-0.999, -0.998) &= \hat{f}_0 + \hat{f}_1(-0.999) + \hat{f}_2(-0.998) \\ &= -0.07 + 0.3 + 0.96 = 1.19\end{aligned}$$



MODEL STRUCTURE AND INTERACTION DE

GBM with interaction depth of 2

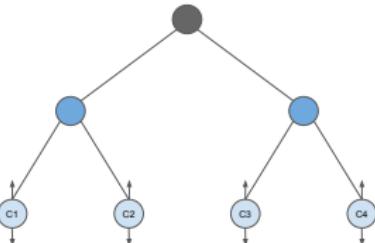
Interactions between x_1 and x_2 are modelled: Marginal effects of x_1 and x_2 do NOT add up to joint effect due to interaction effects.





Introduction to Machine Learning

Gradient Boosting with Trees 2



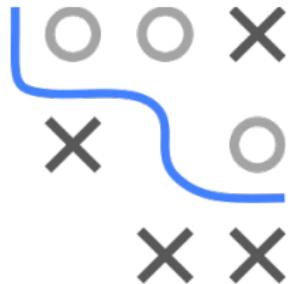
Learning goals

- 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 learnir

ADAPTING TERMINAL COEFFICIENTS

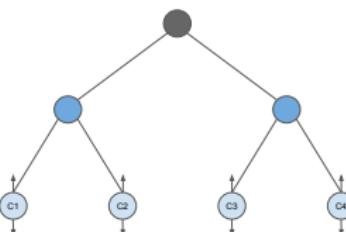
After the tree has been fitted against the PRs, we can adapt terminal constants in a second step to become more loss optimal.



$$f^{[m]}(\mathbf{x}) = f^{[m-1]}(\mathbf{x}) + \sum_{t=1}^{T^{[m]}} \tilde{c}_t^{[m]} \mathbb{1}_{\{\mathbf{x} \in R_t^{[m]}\}}.$$

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. Ris data in a node:

$$\mathcal{R}(\mathcal{N}') = \sum_{i \in \mathcal{N}'} L(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + c)$$

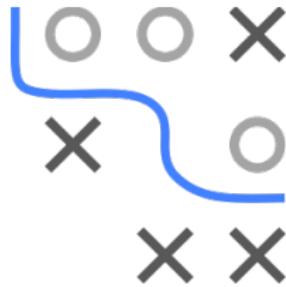


with \mathcal{N}' being the index set of a specific (left or right) node after and c the constant of the node.

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 tree 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 $\hat{c}_{tk}^{[m]}$. Additionally, the regions corresponding to the different classes 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

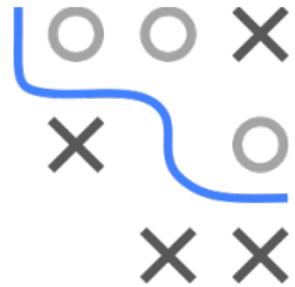
$$\hat{c}_{tk}^{[m]} = \frac{g-1}{g} \frac{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} \tilde{r}_k^{[m](i)}}{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} |\tilde{r}_k^{[m](i)}| \left(1 - |\tilde{r}_k^{[m](i)}|\right)}.$$



GB MULTICLASS WITH TREES

Algorithm 1 Gradient Boosting for g -class Classification.

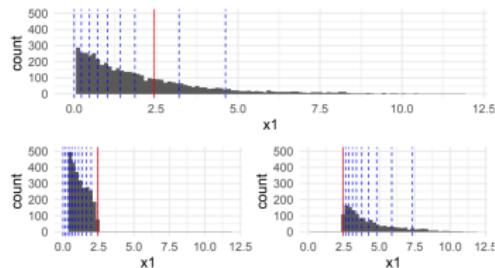
- 1: Initialize $f_k^{[0]}(\mathbf{x}) = 0$, $k = 1, \dots, g$
- 2: **for** $m = 1 \rightarrow M$ **do**
- 3: Set $\pi_k(\mathbf{x}) = \frac{\exp(f_k^{[m]}(\mathbf{x}))}{\sum_j \exp(f_j^{[m]}(\mathbf{x}))}$, $k = 1, \dots, g$
- 4: **for** $k = 1 \rightarrow g$ **do**
- 5: For all i : Compute $\tilde{r}_k^{[m](i)} = \mathbb{1}_{\{y^{(i)}=k\}} - \pi_k(\mathbf{x}^{(i)})$
- 6: Fit regr. tree to the $\tilde{r}_k^{[m](i)}$ giving terminal regions $R_{tk}^{[m]}$
- 7: Compute
$$\hat{c}_{tk}^{[m]} = \frac{g-1}{g} \frac{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} \tilde{r}_k^{[m](i)}}{\sum_{\mathbf{x}^{(i)} \in R_{tk}^{[m]}} |\tilde{r}_k^{[m](i)}| \left(1 - |\tilde{r}_k^{[m](i)}|\right)}$$
- 8: Update $\hat{f}_k^{[m]}(\mathbf{x}) = \hat{f}_k^{[m-1]}(\mathbf{x}) + \sum_t \hat{c}_{tk}^{[m]} \mathbb{1}_{\{\mathbf{x} \in R_{tk}^{[m]}\}}$
- 9: **end for**
- 10: **end for**
- 11: **end for**
- 12: Output $\hat{f}_1^{[M]}, \dots, \hat{f}_g^{[M]}$





Introduction to Machine Learning

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

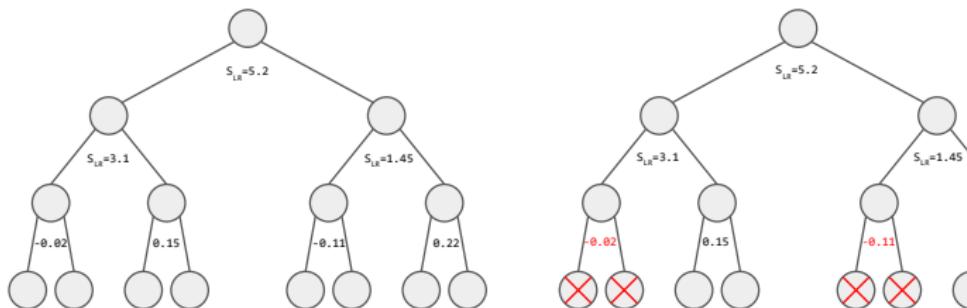
$$\begin{aligned}\mathcal{R}_{\text{reg}}^{[m]} = \sum_{i=1}^n L & \left(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + b^{[m]}(\mathbf{x}^{(i)}) \right) \\ & + \lambda_1 J_1(b^{[m]}) + \lambda_2 J_2(b^{[m]}) + \lambda_3 J_3(b^{[m]})\end{aligned}$$



- $J_1(b^{[m]}) = T^{[m]}$: 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

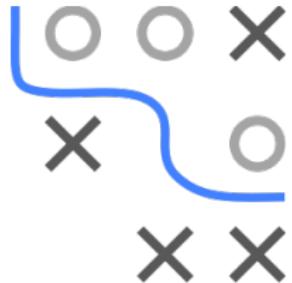
TREE GROWING

- 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 gradient GB.



Feature Subsampling: Similar to `mtry` in a random forest only random subset of features is used for split finding.

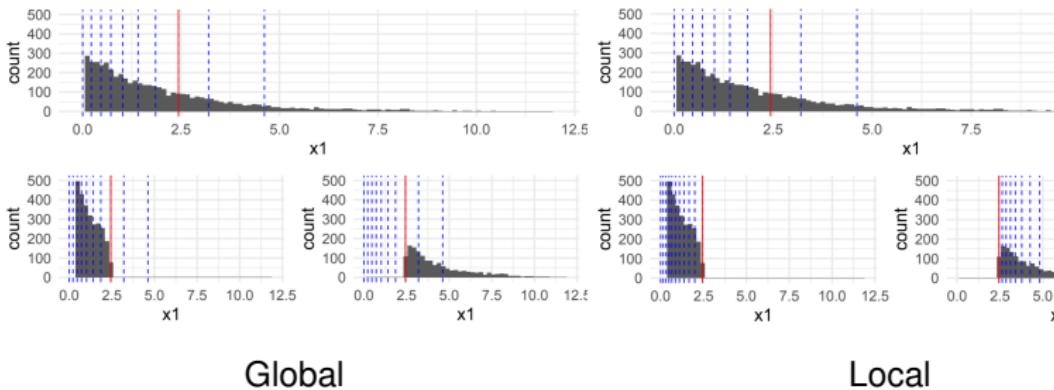
The fraction of features for a split can be randomly sampled for

- ➊ tree
- ➋ level of a tree
- ➌ split

Feature subsampling speeds up training even further and can create more diverse ensembles that will often perform better.

APPROXIMATE SPLIT-FINDING ALGORITHMS

- Speeds up tree building for large data
- Considers not all, but only ℓ 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**



Blue lines are percentiles and red = selected split

DROPOUT ADDITIVE-REGRESSION TREES

DART introduces idea of *dropout* regularization used in DL to b



- 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 BL 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 w 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 controls contribution of each data point)
nrounds	I	$\{1, \dots, 5000\}$	—	—	number of boosting rounds (can also be optimized via early stopping).
gamma	R	$[-7, 6]$	2^x	0	minimum loss reduction required to make a further partition at a node of the tree
max_depth	I	$\{1, \dots, 20\}$	—	6	maximum depth of the tree
colsample_bytree	R	$[0.1, 1]$	—	1	subsample ratio of features used for each tree
colsample_bylevel	R	$[0.1, 1]$	—	1	subsample ratio of features used for each depth level
lambda	R	$[-10, 10]$	2^x	1	L_2 regularization term
alpha	R	$[-10, 10]$	2^x	0	L_1 regularization term
subsample	R	$[0.1, 1]$	—	1	subsample ratio of training instances

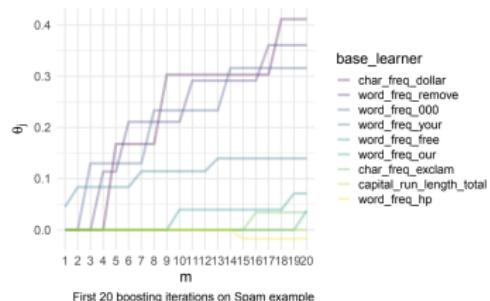


Introduction to Machine Learning

Componentwise Gradient Boosting

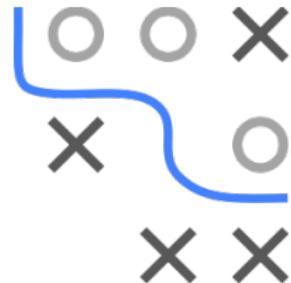
Learning goals

- 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

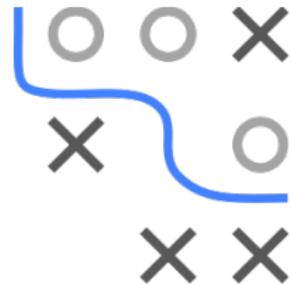
- has strong predictive performance,
- has components that are still interpretable,
- does automatic selection of components,
- is sparser than a model fitted with maximum-likelihood estimation

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

$$\{b_j^{[m]}(\mathbf{x}, \boldsymbol{\theta}^{[m]}) : j = 1, 2, \dots, J\},$$

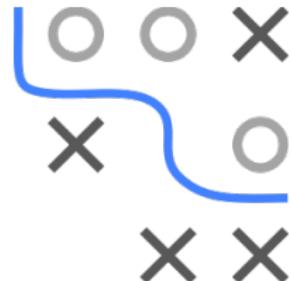
where j indexes the type of base learner.

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



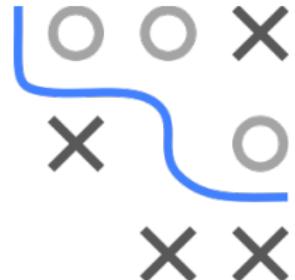
Algorithm Componentwise Gradient Boosting.

- 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 model 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 apply models to all kinds of other distributions and losses, as long as we can calculate (or approximate) a derivative of the loss.

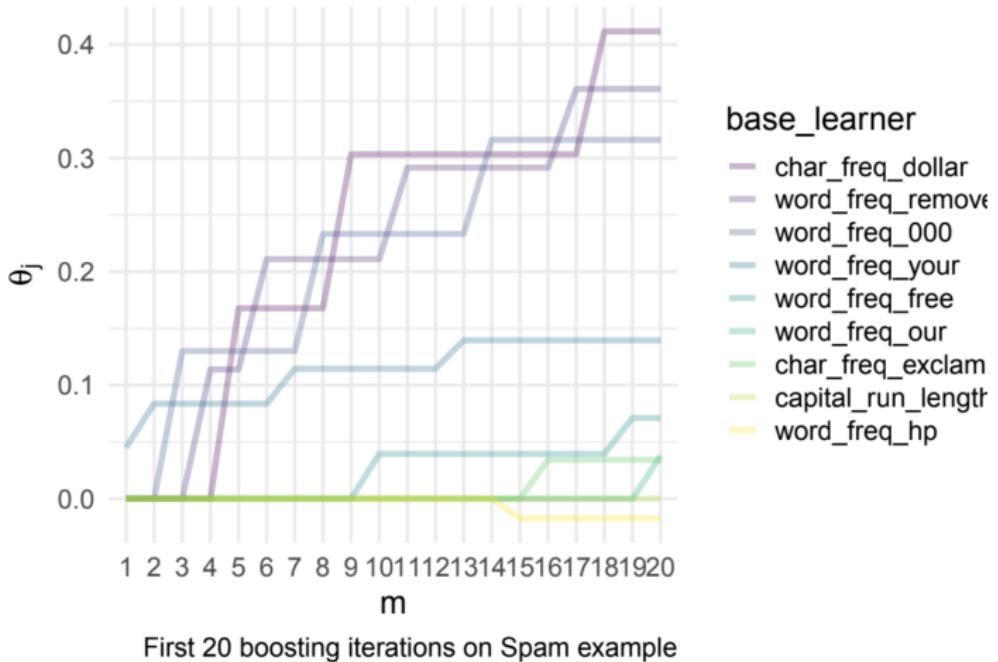
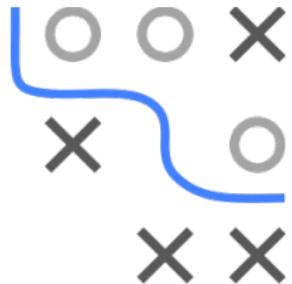


Usually we do not let the boosting model converge fully, but **stop** early for the sake of regularization and feature selection.

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.

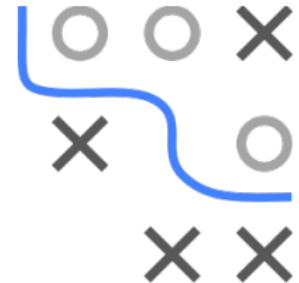


FEATURE SELECTION

- 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 $1 < m_{\text{stop}} \leq$

$$VI_j = \sum_{m=1}^{m_{\text{stop}}} \left(\mathcal{R}_{\text{emp}} \left(f^{[m-1]}(\mathbf{x}) \right) - \mathcal{R}_{\text{emp}} \left(f^{[m]}(\mathbf{x}) \right) \right) \cdot \mathbb{I}_{j \in s}$$

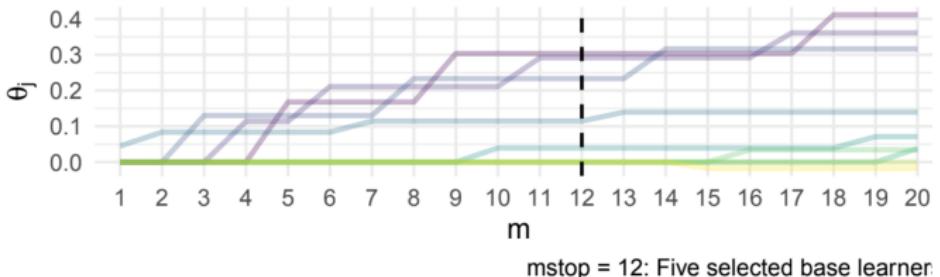
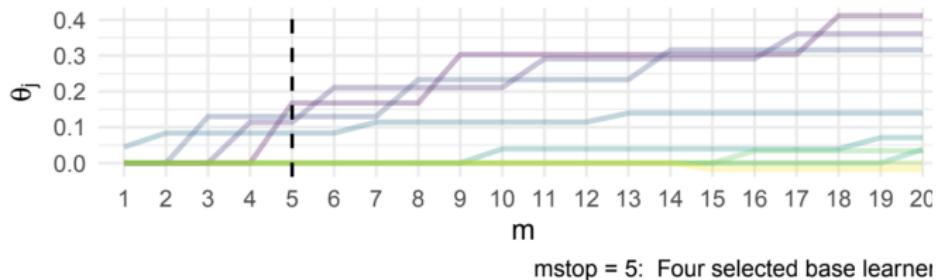
where $s(m)$ denotes the index set of features selected in m -th iteration.



FEATURE SELECTION

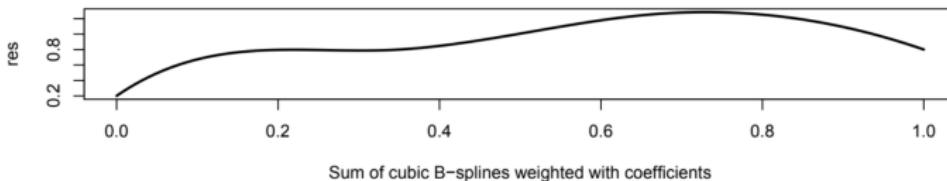
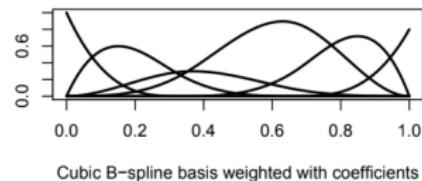
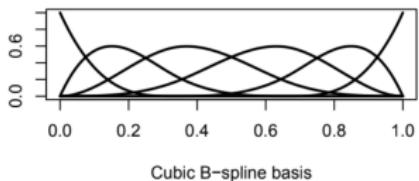
The number of features effectively included in the final model depends on the value of M .

→ A sparse logistic regression is fitted.



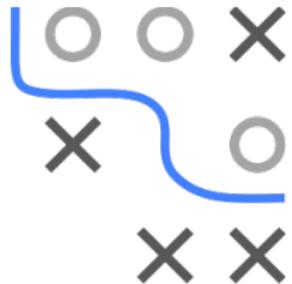
NONLINEAR BASE LEARNERS

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:

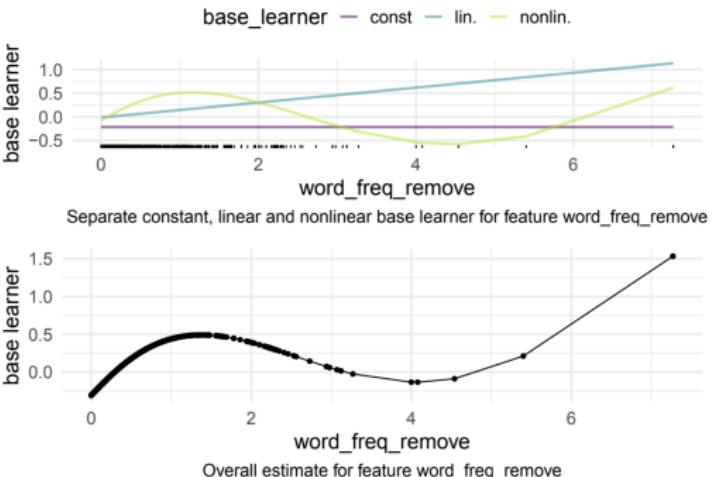
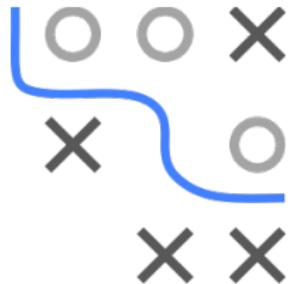
$$\begin{aligned} b_j(x_j, \theta^{[m]}) &= b_{j,\text{const}}(x_j, \theta^{[m]}) + b_{j,\text{lin}}(x_j, \theta^{[m]}) + b_{j,\text{nonlin}}(x_j, \theta^{[m]}) \\ &= \theta_{\text{const}}^{[m]} + x_j \cdot \theta_{j,\text{lin}}^{[m]} + s_j(x_j, \theta_{j,\text{nonlin}}^{[m]}), \end{aligned}$$

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.

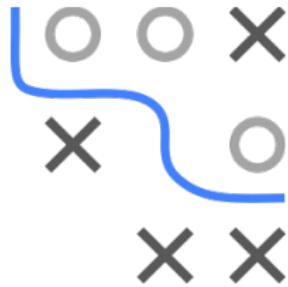
SPAM EXAMPLE: CONTINUED

- 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 1 feature and let the boosting model decide which to select.
- The model selects 17 different base learners (out of 114):



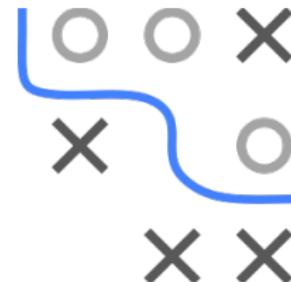
FAIR BASE LEARNER SELECTION

- 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 pe 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 freec

$$df(\lambda_j) = \text{tr} (2\mathbf{S}_j - \mathbf{S}_j^T \mathbf{S}_j).$$

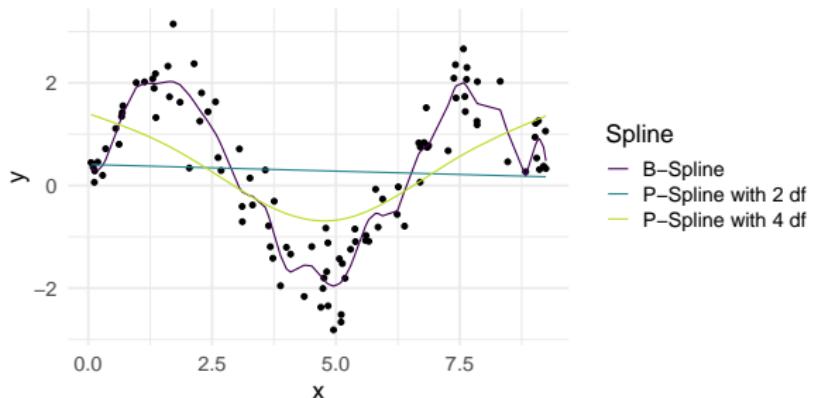
Note: With $\lambda_j = 0$, \mathbf{S}_j is the projection matrix into the target space, with $\text{tr}(\mathbf{S}_j) = \text{rank}(\mathbf{X})$, which corresponds to the number of parameters in the model.

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:

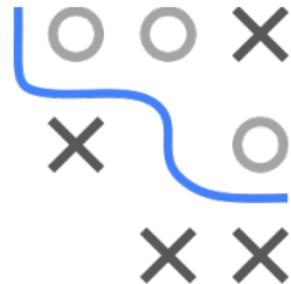
- Setting $df = 24$ gives B -splines with $\lambda_j = 0$.
- Setting $df = 4$ gives P -splines with $\lambda_j = 418$.
- Setting $df = 2$ gives P -splines with $\lambda_j = 42751174892$.



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

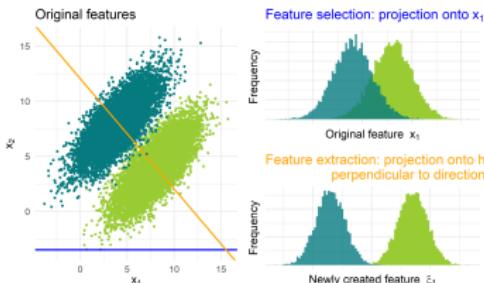
Boosting

Feature Selection



Supervised Learning

Feature Selection



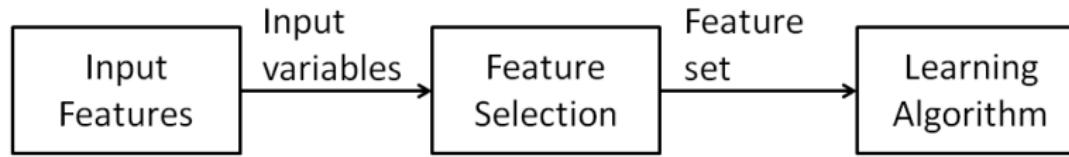
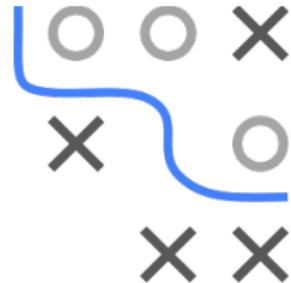
Learning goals

- Understand that adding more features can be detrimental to prediction performance.
- Understand the benefits of keeping only informative features for the model.

INTRODUCTION

Feature selection deals with

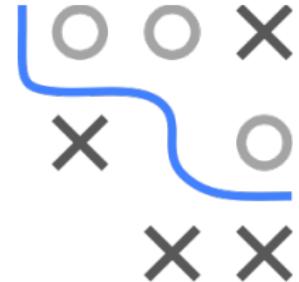
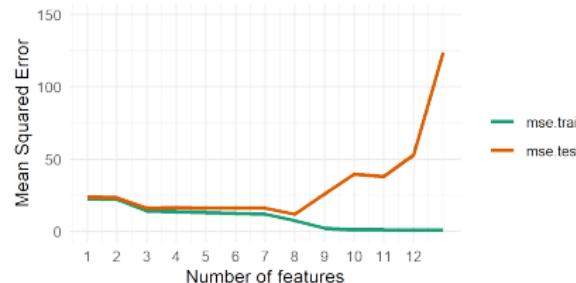
- techniques for choosing a suitable subset of features
- evaluating the influence of features on the model



Feature selection can be performed relying on domain knowledge and expert input, or using a data-driven algorithmic approach.

MOTIVATION

- Naive view:
 - More features → more information → discriminant power ↑
 - Model is not harmed by irrelevant features since their parameters can simply be estimated as 0.
- In practice, irrelevant and redundant features can “confuse” learners (see **curse of dimensionality**) and worsen performance.
- Example: In linear regression, R^2 is monotonically increasing in p , but adding irrelevant features leads to overfitting (capturing noise).



MOTIVATION

- In high-dimensional data sets, we often have prior information that many features are either irrelevant or redundant.
- Feature selection is critical for
 - reducing noise and overfitting,
 - improving performance/generalization,
 - interpretability by identifying most informative features.
- Feature selection can also remedy problems arising in small n regimes or under limited computational resources.
- Many models require $n > p$ data. Thus, we either need to
 - adapt models to high-dimensional data (e.g. regularization),
 - design entirely new procedures for $p > n$ data, or
 - use the preprocessing methods addressed in this lecture.



SIZE OF DATASETS

The increasingly automatized collection of information makes data sets with extremely high dimensionality available, while classical models were developed for small p data.



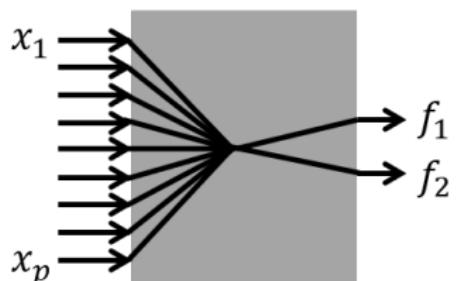
- **Classical setting:** Up to around 10^2 features, feature selection might be relevant, but benefits often negligible.
- **Datasets of medium to high dimensionality:** At around 10^2 to 10^3 features, classical approaches can still work well, while principled feature selection helps in many cases.
- **High-dimensional data:** 10^3 to 10^9 or more features. Examples are e.g. micro-array / gene expression data and text categorization (bag-of-words features). If, in addition, observations are few, the scenario is called $p \gg n$.

FEATURE SELECTION VS. EXTRACTION

Feature selection

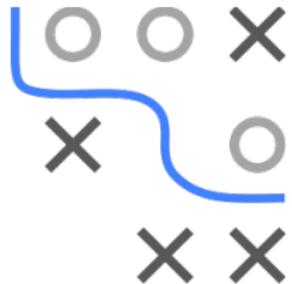


Feature extraction



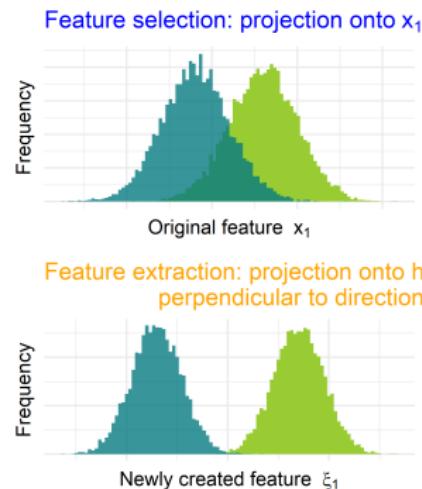
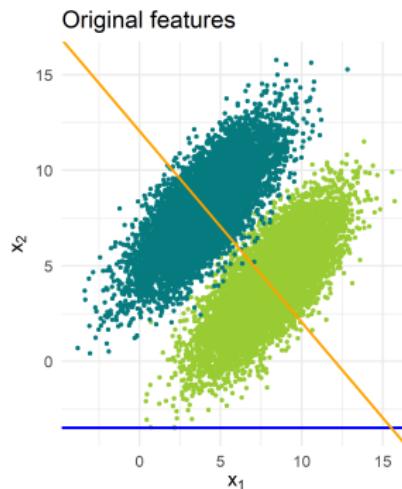
- Creates a subset of original features \mathbf{x} by selecting $\tilde{p} < p$ features \mathbf{f} .
- Retains information on selected individual features.

- Maps p features in \mathbf{x} to \tilde{p} extracted features \mathbf{f} .
- Info on individual features can be lost through (non-)linear combination.



FEATURE SELECTION VS. EXTRACTION

- Both FS and FE contribute to 1) dimensionality reduction, and 2) simplicity of classification rules.
- FE can be unsupervised (PCA, Multidimensional Scaling, Manifold Learning) or supervised (supervised PCA, partial least squares).
- FE can produce lower dim projections which can be more informative than FS.

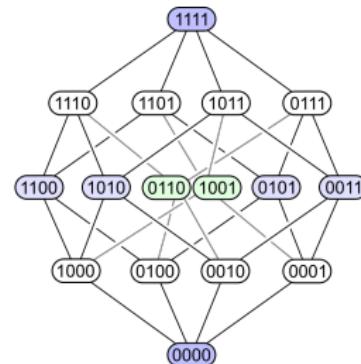
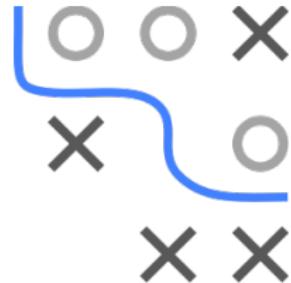


OBJECTIVE FUNCTION

Given p features, the **best-subset selection problem** is to find a subset $S \subseteq \{1, \dots, p\}$ optimizing objective $\Psi : \Omega \rightarrow \mathbb{R}$:

$$S^* \in \arg \min_{S \in \Omega} \{\Psi(S)\}$$

- Ω = search space of all feature subsets $S \subseteq \{1, \dots, p\}$. Usually we encode this by bit vectors, i.e., $\Omega = \{0, 1\}^p$ (1 = feat. selected)
- Objective Ψ can be different functions, e.g., AIC/BIC for LM or cross-validated performance of a learner.



HOW DIFFICULT IS BEST-SUBSET SELECTION?

- Size of search space = 2^p , i.e., grows exponentially in p as it is the power set of $\{1, \dots, p\}$.
- Finding best subset is discrete combinatorial optimization problem also known as L_0 regularization.
- It can be shown that this problem unfortunately can not be solved efficiently in general (NP-hard; see, e.g., Natarajan, 1995).
- We can avoid having to search the entire space by employing efficient search strategies, moving through the search space in a smart way that finds performant feature subsets.



Supervised Learning

Filter Methods

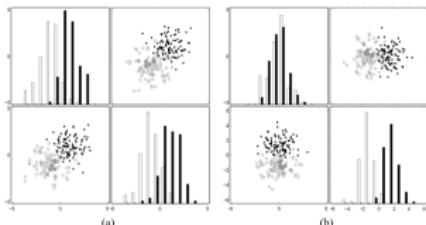
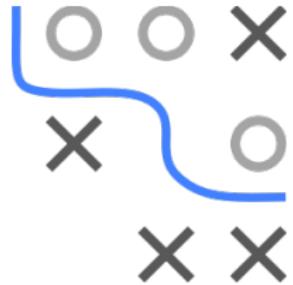


Figure 1: **Information gain from presumably redundant variables.** (a) A two class problem with independently and identically distributed (i.i.d.) variables. Each class has a Gaussian distribution with no covariance. (b) The same example after a 45 degree rotation showing that a combination of the two variables yields a separation improvement by a factor $\sqrt{2}$. I.i.d. variables are not truly redundant.

Learning goals

- Understand how filter methods work
- Understand how to apply them for feature selection
- Understand advantages and disadvantages, and how to overcome them.

INTRODUCTION

- Mostly, **filter methods** construct a measure that describes the strength of the (univariate) dependency between a feature and the target variable.
- Typically, this yields a numeric score for each feature j . That is what is known as **variable-ranking**.
- Filters are in general independent of a specific classification learner and can be applied generically.
- Filter methods are strongly related to methods for determining variable importance.



EXAMPLES, TAKEN FROM GUYON (2003)

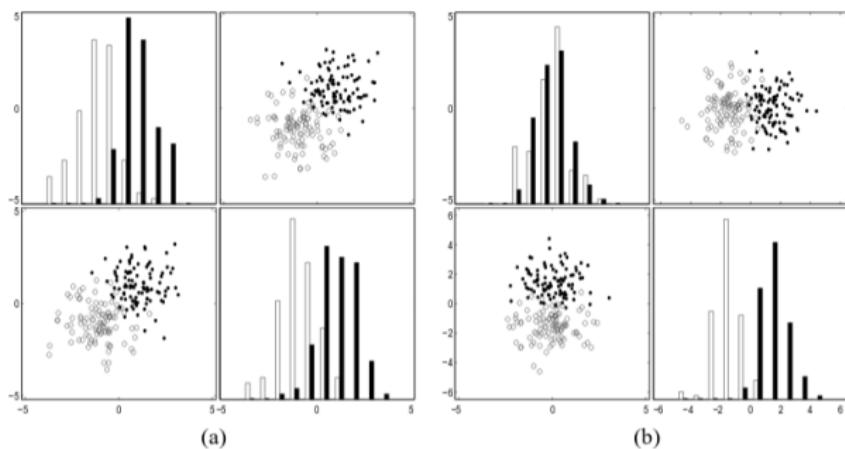


Figure 1: **Information gain from presumably redundant variables.** (a) A two class problem with independently and identically distributed (i.i.d.) variables. Each class has a Gaussian distribution with no covariance. (b) The same example after a 45 degree rotation showing that a combination of the two variables yields a separation improvement by a factor $\sqrt{2}$. I.i.d. variables are not truly redundant.



Isabelle Guyon, André Elisseeff (2003). An Introduction to Variable and Feature Selection. Journal of Machine Learning Research (3) p. 1157-1182.

EXAMPLES, TAKEN FROM GUYON (2003)

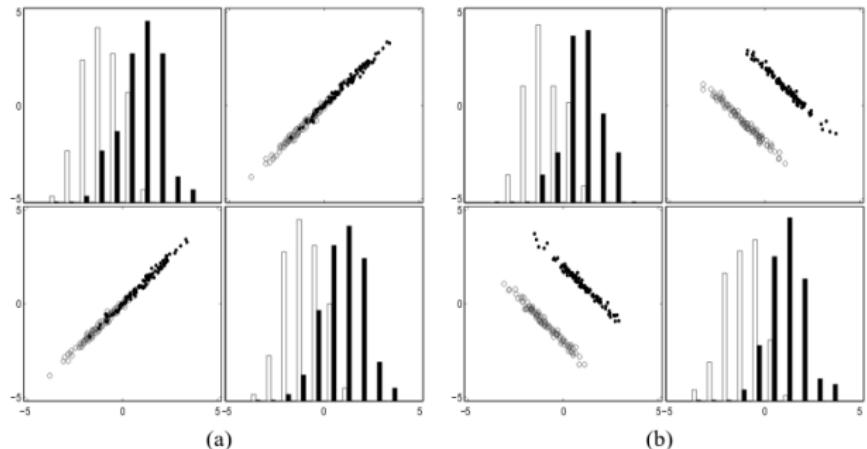


Figure 2: **Intra-class covariance.** In projection on the axes, the distributions of the two variables are the same as in the previous example. (a) The class conditional distributions have a high covariance in the direction of the line of the two class centers. There is no significant gain in separation by using two variables instead of just one. (b) The class conditional distributions have a high covariance in the direction perpendicular to the line of the two class centers. An important separation gain is obtained by using two variables instead of one.

Isabelle Guyon, André Elisseeff (2003). An Introduction to Variable and Feature Selection. Journal of Machine Learning Research (3) p. 1157-1182.



EXAMPLES, TAKEN FROM GUYON (2003)

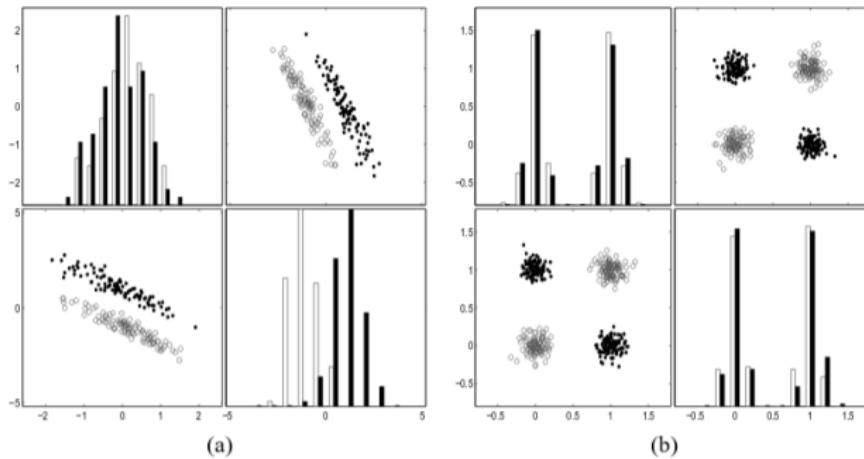


Figure 3: A variable useless by itself can be useful together with others. (a) One variable has completely overlapping class conditional densities. Still, using it jointly with the other variable improves class separability compared to using the other variable alone. (b) XOR-like or chessboard-like problems. The classes consist of disjoint clumps such that in projection on the axes the class conditional densities overlap perfectly. Therefore, individual variables have no separation power. Still, taken together, the variables provide good class separability .



Isabelle Guyon, André Elisseeff (2003). An Introduction to Variable and Feature Selection. Journal of Machine Learning Research (3) p. 1157-1182.

FILTER: χ^2 -STATISTIC



- Test for independence between the j -th feature and the target y .
- Numeric features or targets need to be discretized.
- Hypotheses:

$$H_0 : p(x_j = l, y = k) = p(x_j = l) p(y = k), \forall j = 1, \dots, k_1 \\ \forall k = 1, \dots, k_2$$

$$H_1 : \exists j, k : p(x_j = l, y = k) \neq p(x_j = l) p(y = k)$$

- Calculate the χ^2 -statistic for each feature-target combination:

$$\chi^2 = \sum_{j=1}^{k_1} \sum_{k=1}^{k_2} \left(\frac{e_{jk} - \tilde{e}_{jk}}{\tilde{e}_{jk}} \right) \underset{\text{approx.}}{\overset{H_0}{\sim}} \chi^2((k_1 - 1)(k_2 - 1))$$

where e_{jk} is the observed relative frequency of pair (j, k) and $\tilde{e}_{jk} = \frac{e_j \cdot e_k}{n}$ is the expected relative frequency.

- The greater χ^2 , the more dependent is the feature-target combination, the more relevant is the feature.

FILTER: PEARSON & SPEARMAN CORRELATION

Pearson correlation $r(\mathbf{x}_j, y)$:

- For numeric features and targets only.
- Most sensitive for linear or monotonic relationships.

$$\bullet \quad r(\mathbf{x}_j, y) = \frac{\sum_{i=1}^n (x_j^{(i)} - \bar{x}_j)(y^{(i)} - \bar{y})}{\sqrt{\sum_{i=1}^n (x_j^{(i)} - \bar{x}_j)^2} \sqrt{\sum_{i=1}^n (y^{(i)} - \bar{y})^2}}, \quad -1 \leq r \leq 1$$

Spearman correlation $r_{SP}(\mathbf{x}_j, y)$:

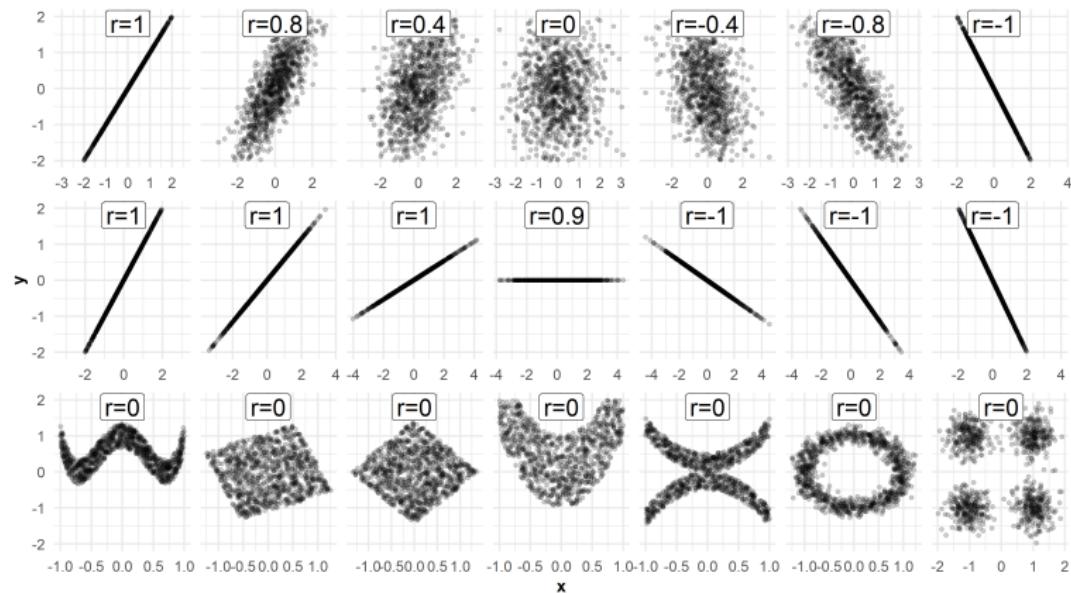
- For features and targets at least on an ordinal scale.
- Equivalent to Pearson correlation computed on the ranks.
- Assesses monotonicity of the dependency relationship.

Use absolute values $|r(\mathbf{x}_j, y)|$ for feature ranking: higher score indicates a higher relevance.



FILTER: PEARSON & SPEARMAN CORRELATION

Only **linear** dependency structure, non-linear (non-monotonic) aspects are not captured by r :



FILTER: DISTANCE CORRELATION



$$r_D(\mathbf{x}_j, y) = \sqrt{\frac{c_D^2(\mathbf{x}_j, y)}{\sqrt{c_D^2(\mathbf{x}_j, \mathbf{x}_j)c_D^2(y, y)}}} :$$

Normed version of **distance covariance**

$$c_D(\mathbf{x}_j, y) = \frac{1}{n^2} \sum_{i=1}^n \sum_{k=1}^n D_{\mathbf{x}_j}^{(ik)} D_y^{(ik)} \text{ for}$$

- distances of observations: $d(x_j^{(i)}, x_j^{(k)})$,
- mean distances $\bar{d}_{xj}^{(i\cdot)} = \frac{1}{n} \sum_{k=1}^n d(x_j^{(i)}, x_j^{(k)})$,
- and centered pairwise distances
$$D_x^{(ik)} = d(x_j^{(i)}, x_j^{(k)}) - (\bar{d}_{xj}^{(i\cdot)} + \bar{d}_{xj}^{(\cdot k)} - \bar{d}_{xj}^{(\cdot\cdot)})$$

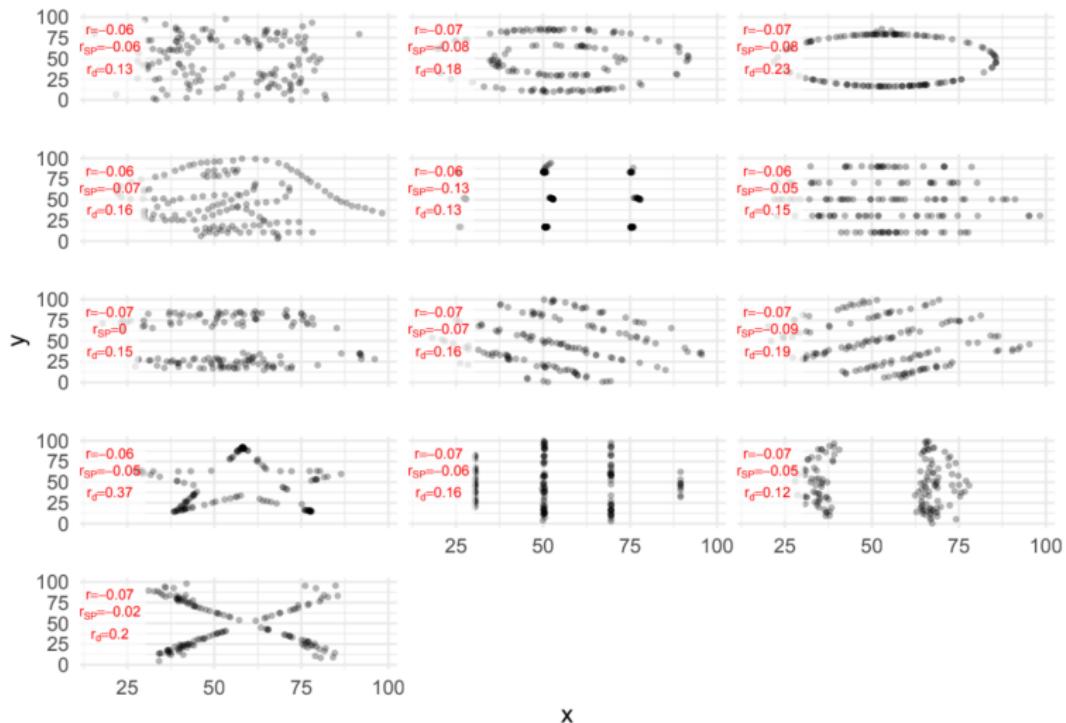
FILTER: DISTANCE CORRELATION

Properties:

- $0 \leq r_D(\mathbf{x}_j, y) \leq 1 \quad \forall j \in \{1, \dots, p\}$
- $r_D(\mathbf{x}_j, y) = 0$ only if \mathbf{x} and y are empirically independent (!)
- $r_D(\mathbf{x}_j, y) = 1$ for exact linear dependencies
- also assesses strength of **non-monotonic, non-linear** dependencies
- very generally applicable since it only requires distance measures on \mathcal{X} and \mathcal{Y} : can also be used for ranking multivariate features, feature combinations or non-tabular inputs (text, images, audio, etc.)
- expensive to compute for large data (i.e., use a subsample)



FILTER: DISTANCE CORRELATION



FILTER: WELCH'S t-TEST

- For binary classification with numeric features.
- Test for unequal means of the j -th feature.
- For notational purposes let $\mathcal{Y} \in \{0, 1\}$. Then the subscript j_0 refers to the j -th feature where $y = 0$ and j_1 where $y = 1$.
- Hypotheses:
 $H_0: \mu_{j_0} = \mu_{j_1}$ vs. $H_1: \mu_{j_0} \neq \mu_{j_1}$
- Calculate Welch's t-statistic for every feature \mathbf{x}_j

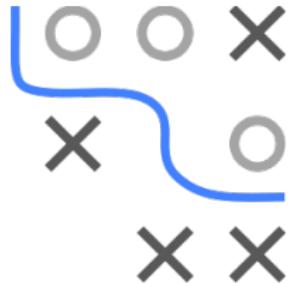
$$t_j = \frac{\bar{x}_{j_0} - \bar{x}_{j_1}}{\sqrt{\left(\frac{S_{x_{j_0}}^2}{n_0} + \frac{S_{x_{j_1}}^2}{n_1}\right)}}$$

where \bar{x}_{j_0} , $S_{x_{j_0}}^2$ and n_0 are the sample mean, the population variance and the sample size for $y = 0$, respectively.

- A higher t-score indicates higher relevance of the feature.



FILTER: F-TEST



- For multiclass classification ($g \geq 2$) and numeric features.
- Assesses whether the expected values of a feature \mathbf{x}_j within the classes of the target differ from each other.
- Hypotheses:
 $H_0 : \mu_{j_0} = \mu_{j_1} = \dots = \mu_{j_g}$ vs. $H_1 : \exists k, l : \mu_{j_k} \neq \mu_{j_l}$
- Calculate the F-statistic for each feature-target combination:

$$F = \frac{\text{between-group variability}}{\text{within-group variability}}$$

$$F = \frac{\sum_{k=1}^g n_k (\bar{x}_{j_k} - \bar{x}_j)^2 / (g - 1)}{\sum_{k=1}^g \sum_{i=1}^{n_k} (x_{j_k}^{(i)} - \bar{x}_{j_k})^2 / (n - g)}$$

where \bar{x}_{j_k} is the sample mean of feature \mathbf{x}_j where $y = k$ and \bar{x}_j is the overall sample mean of feature \mathbf{x}_j .

- A higher F-score indicates higher relevance of the feature.

FILTER: MUTUAL INFORMATION

$$I(X; Y) = \mathbb{E}_{p(x,y)} \left[\log \frac{p(X, Y)}{p(X)p(Y)} \right]$$



- Each variable x_j is rated according to $I(x_j; y)$, this is sometimes called information gain.
- MI is a measure of the amount of "dependence" between variables. It is zero if and only if the variables are independent.
- On the other hand, if one of the variables is a deterministic function of the other, the mutual information is maximal.
- Unlike (Pearson) correlation, mutual information is not limited to real-valued random variables.
- Mutual information can be seen as a more general measure of dependence between variables than correlation.

FILTER

How to combine a filter with a model:

- Calculate filter-values.
- Sort features by value.
- Train model on \tilde{p} best features.



How to choose \tilde{p} :

- It can be prescribed by the application.
- Eyeball estimation: Read from filter plots (i.e., Scree plots).
- Use resampling.

FILTER

Advantages:

- Easy to calculate.
- Typically scales well with the number of features p .
- Mostly interpretable intuitively.
- Combination with every model possible.

Disadvantages:

- Often univariate (not always) ignores multivariate dependencies.
- Redundant features will have similar weights.
- Ignores the learning algorithm.



MINIMUM REDUNDANCY MAXIMUM RELEVANCY

- Most filter type methods select top-ranked features based on a certain filter method (χ^2 , rank-correlation, t-test,...) without considering relationships among the features.
- Problems:
 - Features may be correlated and hence, may cause redundancy.
 - Selected features cover narrow regions in space.
- Goal: In addition to maximum relevancy of the features we want the features to be maximally dissimilar to each other (minimum redundancy).
- Features can be either continuous or categorical.



mRMR: CRITERION FUNCTIONS

- Let $S \subset \{1, \dots, p\}$ be a subset of features we want to find.
- Minimize redundancy

$$\min \text{Red}(S), \quad \text{Red}(S) = \frac{1}{|S|^2} \sum_{j,l \in S} I_{xx}(\mathbf{x}_j, \mathbf{x}_l)$$

- Maximize relevancy

$$\max \text{Rel}(S), \quad \text{Rel}(S) = \frac{1}{|S|} \sum_{j \in S} I_{xy}(\mathbf{x}_j, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(n)}\})$$

- I_{xx} measures the strength of the dependency between two features.
- I_{xy} measures the strength of the dependency between a feature and the target.



mRMR: CRITERION FUNCTIONS

- Examples for I_{xx} :

- Two discrete features: mutual information $I(\mathbf{x}_j, \mathbf{x}_l)$
- Two numeric features: correlation $|r(\mathbf{x}_j, \mathbf{x}_l)|$

- Examples for I_{xy} :

- Discrete feature, discrete target: mutual information $I(\mathbf{x}_j, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(n)}\})$
- Continuous feature, discrete target: F-statistic $F(\mathbf{x}_j, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(n)}\})$



mRMR: CRITERION FUNCTIONS

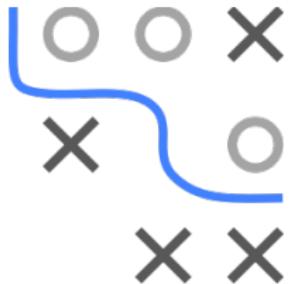
- The mRMR feature set is obtained by optimizing the relevancy and the redundancy criterion simultaneously.
- This requires combining the criteria into a single objective function:

$$\Psi(S) = (\text{Rel}(S) - \text{Red}(S)) \quad \text{or} \quad \Psi(S) = (\text{Rel}(S)/\text{Red}(S))$$

- Exact solution requires $\mathcal{O}(|\mathcal{X}|^{|S|})$ searches, where $|\mathcal{X}|$ is the number of features in the whole feature set and $|S|$ is the number of features selected.

In practice, incremental search methods are used to find near-optimal feature sets defined by Ψ :

- Suppose we already have a feature set with $m - 1$ features S_{m-1} .



mRMR: CRITERION FUNCTIONS

- Next, we select the m -th feature from the set \bar{S}_{m-1} by selecting the feature that maximizes:

$$\max_{j \in \bar{S}_{m-1}} [I_{xy}(\mathbf{x}_j, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(n)}\}) - \frac{1}{|\bar{S}_{m-1}|} \sum_{l \in \bar{S}_{m-1}} I_{xx}(\mathbf{x}_j, \mathbf{x}_l)]$$

- The complexity of this incremental algorithm is $\mathcal{O}(|p| \cdot |S|)$.



mRMR: ALGORITHM

Algorithm mRMR algorithm

1: Set $S = \emptyset$, $R = \{1, \dots, p\}$

2: Find the feature with maximum relevancy:

$$j^* := \arg \max_j I_{xy}(\mathbf{x}_j, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(n)}\})$$

3: Set $S = \{j^*\}$ and update $R \leftarrow R \setminus \{j^*\}$

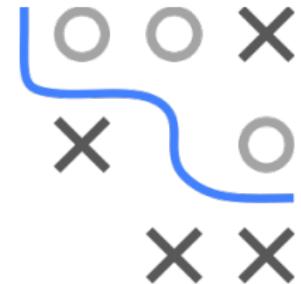
4: **repeat**

5: Find feature \mathbf{x}_j that maximizes:

$$\text{missing formula}$$

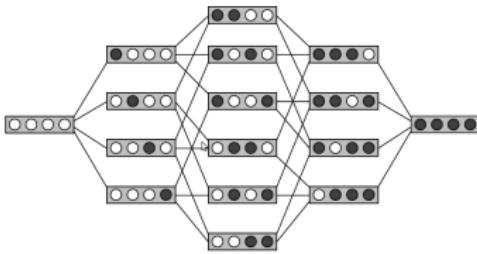
6: Update $S \leftarrow S \cup \{j^*\}$ and $R \leftarrow R \setminus \{j^*\}$

7: **until** Expected number of features have been obtained or some other constraints are satisfied.



Supervised Learning

Wrapper methods



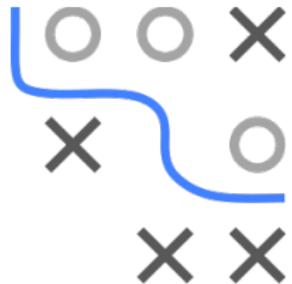
Learning goals

- Understand how wrapper methods work
- Understand how they could help in feature selection
- Know their advantages and disadvantages



INTRODUCTION

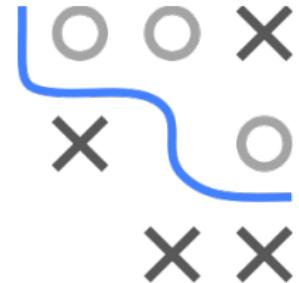
- Wrapper methods emerged from the idea that different sets of features can be optimal for different classification learners.
- Given a set of features, we can use the classifier itself to assess their quality.
- We could just evaluate on the test set or use resampling techniques to achieve this.
- A wrapper is nothing else than a discrete search strategy for S , where the cross-validated test error of a learner as a function of S is now the objective criterion.



INTRODUCTION

There are a lot of varieties of wrappers. To begin with we have to determine the following components:

- A set of starting values
- Operators to create new points out of the given ones
- A termination criterion



INTRODUCTION

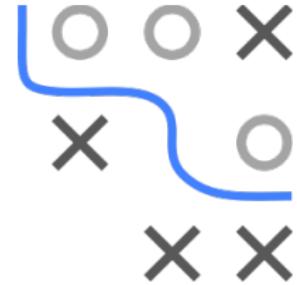
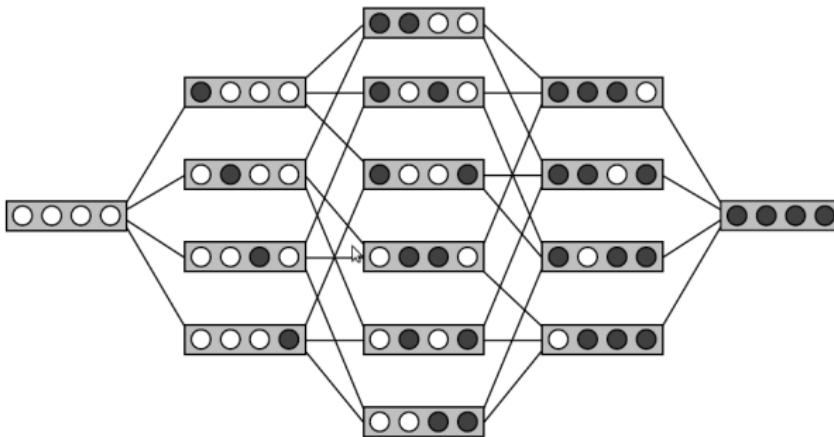


Figure: Space of all feature sets for 4 features. The indicated relationships between the sets insinuate a greedy search strategy which either adds or removes a feature.

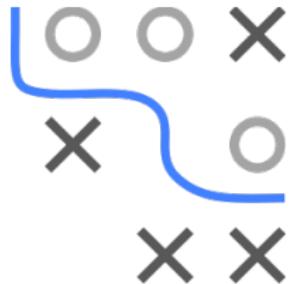
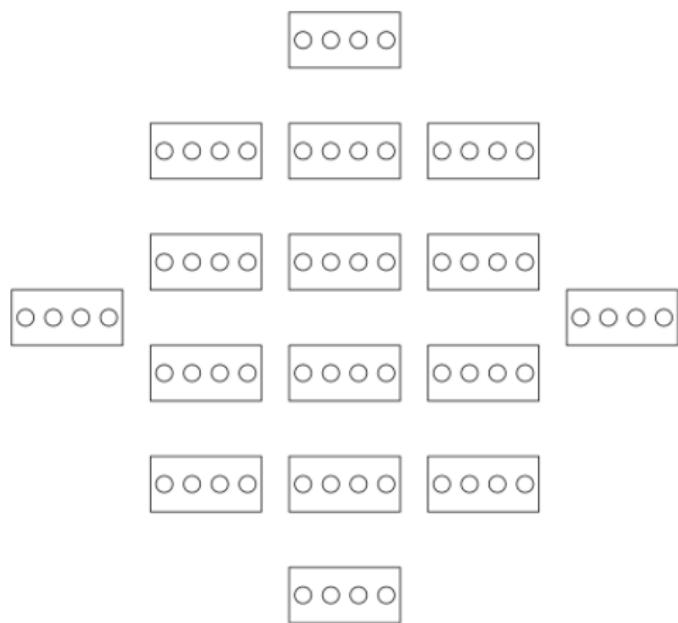
GREEDY FORWARD SEARCH

- Let $S \subset \{1, \dots, p\}$, where $\{1, \dots, p\}$ is an index set of all features.
- Start with the empty feature set $S = \emptyset$.
- For a given set S , generate all $S_j = S \cup \{j\}$ with $j \notin S$.
- Evaluate the classifier on all S_j and use the best S_j .
- Iterate over this procedure.
- Terminate if:
 - the performance measure no longer shows relevant improvement,
 - a maximum number of features is used, or
 - a given performance value is reached.

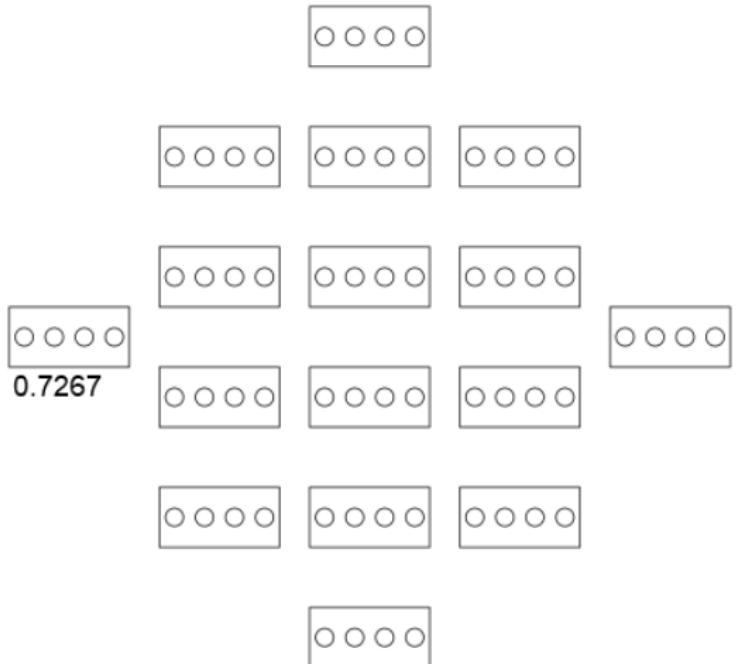


GREEDY FORWARD SEARCH

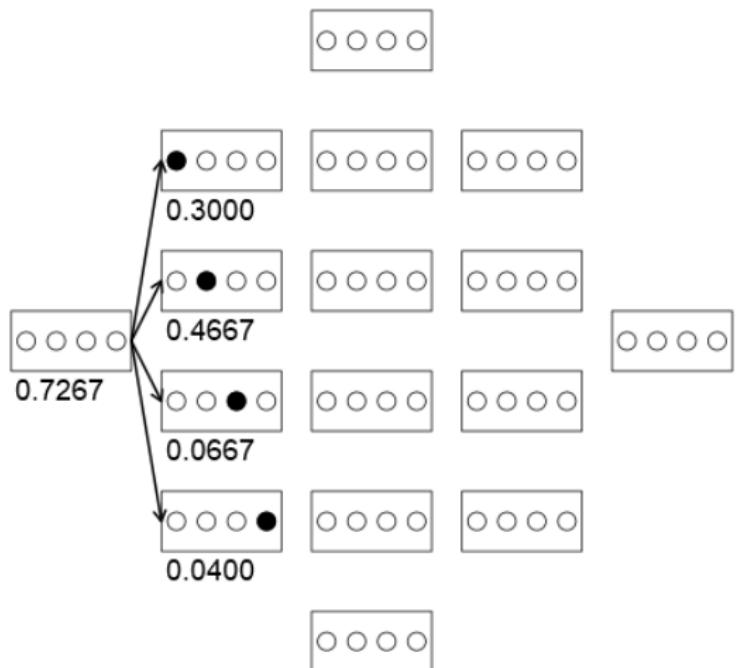
Example for greedy forward search on iris data:



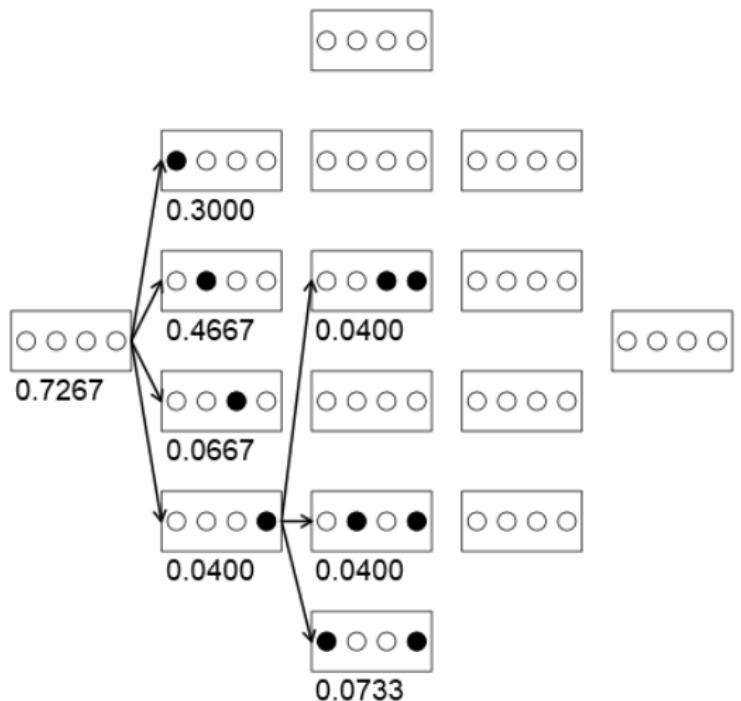
GREEDY FORWARD SEARCH



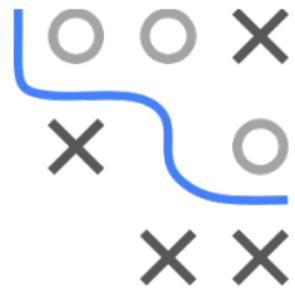
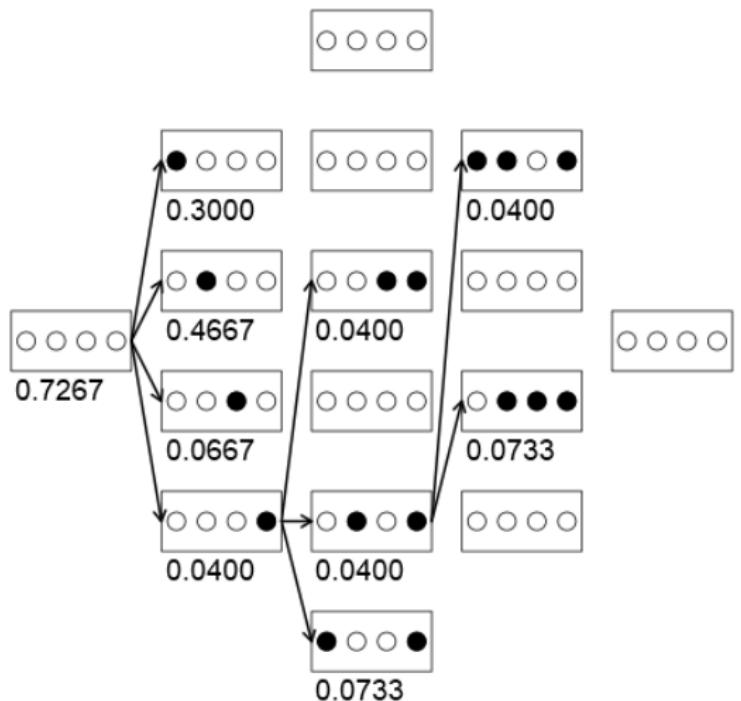
GREEDY FORWARD SEARCH



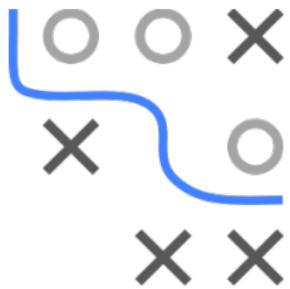
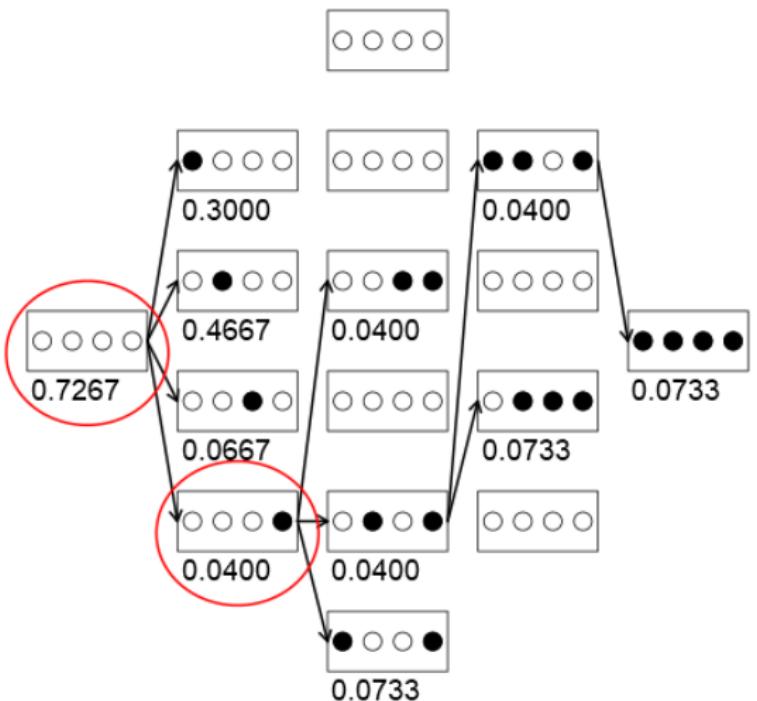
GREEDY FORWARD SEARCH



GREEDY FORWARD SEARCH



GREEDY FORWARD SEARCH



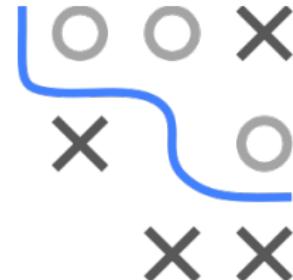
GREEDY BACKWARD SEARCH

- Start with the full index set of features $S = \{1, \dots, p\}$.
- For a given set S generate all $S_j = S \setminus \{j\}$ with $j \in S$.
- Evaluate the classifier on all S_j and use the best S_j .
- Iterate over this procedure.
- Terminate if:
 - the performance drops drastically, or
 - a given performance value is undershot.



EXTENSIONS

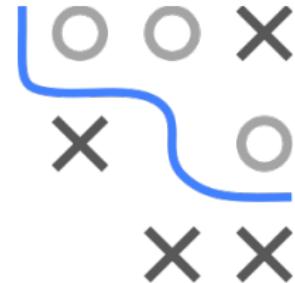
- Eliminate or add several features at once to increase speed.
- Allow alternating forward and backward search.
- Randomly create candidate feature sets in each iteration.
- Continue search based on the set of features where an improvement is present.
- Use improvements of earlier iterations.



EXTENSIONS

Algorithm A simple 1+1 genetic algorithm

- 1: Start with a random set of features S (bit vector b).
- 2: **repeat**
- 3: Flip a couple of bits in b with probability p .
- 4: Generate set S' and bit vector b' .
- 5: Measure the classifier's performance on S' .
- 6: If S' performs better than S , update $S \leftarrow S'$, otherwise $S \leftarrow S$.
- 7: **until** One of the following conditions is met:
 - A given performance value is reached.
 - Budget is exhausted.



WRAPPERS



Advantages:

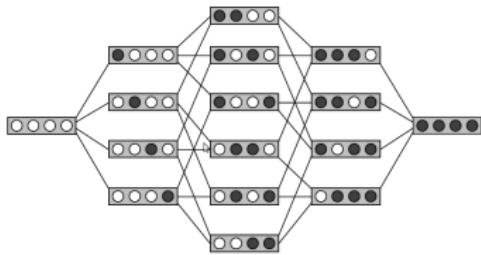
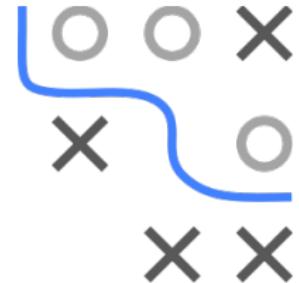
- Can be combined with every learner.
- Can be combined with every performance measure.
- Optimizes the desired criterion directly.

Disadvantages:

- Evaluating the target function is expensive.
- Does not scale well if number of features becomes large.
- Does not use much structure or available information from our model.

Supervised Learning

Embedded Feature Selection

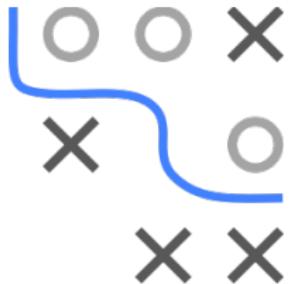


Learning goals

- Add Learning goals

EMBEDDED FEATURE SELECTION

- Embedded techniques are methods that integrate feature selection directly into the learning process.
- They use an internal criterion of the applied learner to have better control over the search for useful features.
- Embedded techniques usually need to be tailored to each learner.



SVM: RECURSIVE FEATURE ELIMINATION (RFE)

- RFE is a popular backward search technique for the linear SVM and the 2-class problem.
- Here we will always assume standardized features. (This should be the case for an SVM anyway!)
- Coefficient size $|\theta_j|$ tells us how much impact feature j has on our classification, since $f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} + \theta_0$ is the decision function.

Idea: Sequentially drop the feature with the smallest $|\theta_j|$.



SVM: RECURSIVE FEATURE ELIMINATION (RFE)

Recursive feature elimination

- Standardize the data.
- Start with full set of features S .
- Fit a linear SVM, using the features in S , and estimate coefficients θ .
- Remove feature(s) j with minimal $|\theta_j|$ from S .
- Iterate using the reduced S for the SVM.



SVM: RECURSIVE FEATURE ELIMINATION (RFE)

Some notes:

- Strictly speaking, this procedure does not perform selection, but rather constructs a ranking of the features.
- As for filters, we need an extra criterion for termination / selection.
- To improve speed one can drop k features in each iteration.
- Extensions to other kernels or multi-class tasks are not trivial.

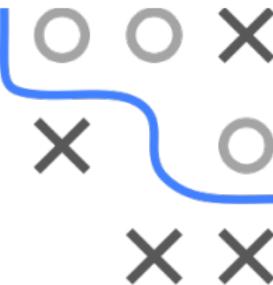


L1 PENALIZATION / LASSO

LASSO: least absolute shrinkage and selection operator

- As introduced before, linear methods that regularize the coefficients of the model with an $L1$ penalty in the empirical risk

$$\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) = \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1 = \sum_{i=1}^n \left(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)} \right)^2 + \lambda \sum_{j=1}^p |\theta_j|$$

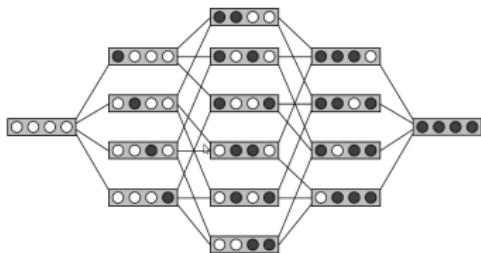
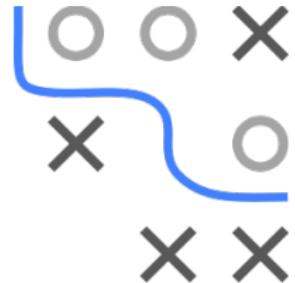


are very popular for high-dimensional data.

- The penalty summand shrinks the coefficients towards 0 in the final model.
- Many (improved) variants: group LASSO, adaptive LASSO, ElasticNet, ...
- Has some very nice optimality results: e.g., compressed sensing.

Supervised Learning

Practical Tips for Feature Selection



Learning goals

- Add learning goals

SOLVING A FEATURE SELECTION PROBLEM (TAKEN FROM GUYON (2003))

① Do you have domain knowledge?

If yes, construct a better set of “ad hoc” features.

② Are your features commensurate?

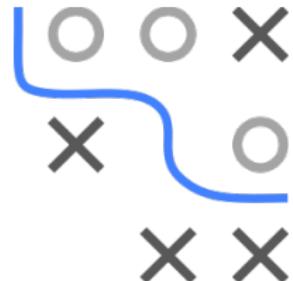
If no, consider normalizing them.

③ Do you suspect interdependence of features?

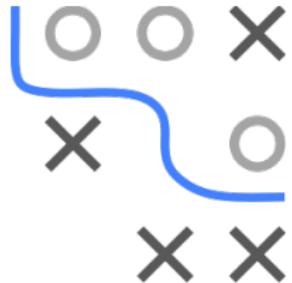
If yes, expand your feature set by constructing conjunctive features or products of features, as many as your computer resources allow you to.

④ Do you need to prune the input features (e.g. for cost, speed or data-understanding reasons)?

If no, construct disjunctive features or weighted sums of features (e.g. by clustering or matrix factorization).



SOLVING A FEATURE SELECTION PROBLEM (TAKEN FROM GUYON (2003))



- ⑤ Do you need to assess features individually (e.g. to understand their influence on the system, or because their number is so large that you need to do a first filtering)?
If yes, use a variable-ranking method. Otherwise, do it anyway to get baseline results.

- ⑥ Do you need a predictor?

If no, stop.

- ⑦ Do you suspect your data is “dirty” (has a few meaningless input patterns and/or noisy outputs or wrong class labels)?
If yes, detect the outlier examples using the top-ranking features obtained in step 5 as representation; check and/or discard them.

SOLVING A FEATURE SELECTION PROBLEM (TAKEN FROM GUYON (2003))

⑧ Do you know what to try first?

If no, use a linear predictor. Use a forward selection method with the “probe” method as a stopping criterion or use the L_0 norm embedded method. For comparison, following the ranking of step 5, construct a sequence of predictors from the same family, using increasing subsets of features. Can you match or improve performance with a smaller subset? If yes, try a nonlinear predictor with that subset.

⑨ Do you have new ideas, time, computational resources, and enough examples?

If yes, compare several feature selection methods, including your new idea, correlation coefficients, backward selection and embedded methods. Use linear and nonlinear predictors. Select the best approach via model selection.



SOLVING A FEATURE SELECTION PROBLEM (TAKEN FROM GUYON (2003))

- ⑩ Do you want a stable solution (to improve performance and/or understanding)?

If yes, sub-sample your data and redo your analysis for several bootstraps.



OPEN POINTS / PROBLEMS

- In general, it is difficult to give suggestions on when to use which feature selection method.
- Most of the time, it is reasonable to start with a simple, fast method. If this yields unsatisfactory results, one can gradually move to more expensive methods.
- Not every introduced method can be generalized to multi-class problems in an easy fashion.
- Combining the choice of an appropriate classifier and parameter tuning with feature selection is not simple.

