Deeplearning

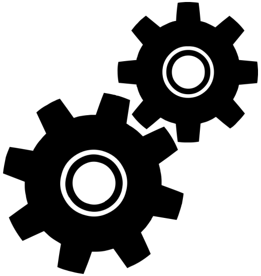
Chapter 0: Introduction to Machine Learning

Bernd Bischl

Department of Statistics – LMU Munich

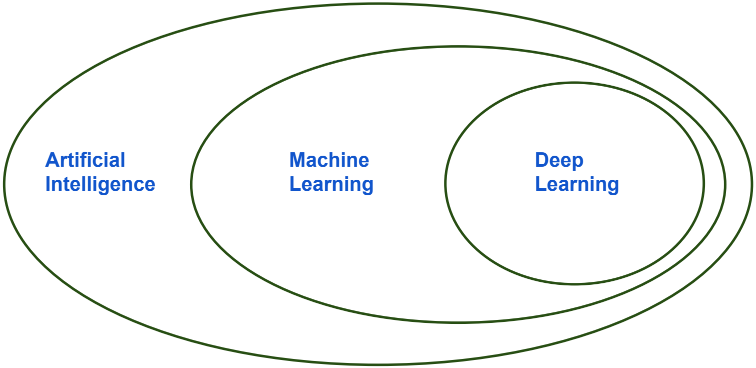
Winter term 2018

DATA SCIENCE AND MACHINE LEARNING



Machine learning is a branch of computer science and applied statistics covering algorithms that improves their performance at a given task based on sample data or experience.

DATA SCIENCE AND MACHINE LEARNING



MACHINE LEARNING AS BLACK-BOX MODELING

- Many concepts in ML can be explained without referring to the inner workings of a certain algorithm or model, especially things like model evaluation and tuning.
- ML also nowadays consists of dozens (or hundreds?) of different modeling techniques, where it is quite unclear which of these are really needed (outside of pure research) and which are really best.
- Understanding model-agnostic techniques is really paramount and can be achieved in a limited amount of time.

ML AS BLACK-BOX MODELING

- Really studying the inner workings of each and every ML model can take years. Do we even need to do this at all for some models?
- No: They exist in software. We can simply try them out, in the best case with an intelligent program that iterates over them and optimizes them.
- Yes: For practitioners: Some basic knowledge is often necessary to make right choices. And often stuff goes wrong, then understanding helps, too. For ML professionals: Deeper knowledge is needed to tailor methods to your specific problems, to sharpen intuition to create new ones, and as scientists of course we are interested in the fundamental theory.

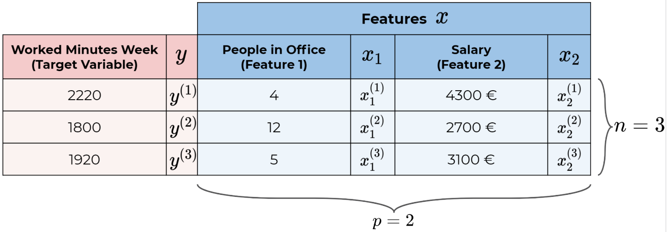
ML AS BLACK-BOX MODELING

- In the follwoing slides we will go through really fundamental terms and concepts in ML, that are relevant for everything that comes next.
- We will also learn a couple of extremely simple models to obtain a basic understanding.
- More complex stuff comes later.

Imagine you want to investigate how salary and workplace conditions affect productivity of employees. Therefore, you collect data about worked minutes per week (productivity), how many people work in the same office as the employees in question and the employees' salary.

Supervised Learning Scenario

DATA, TARGET AND INPUT FEATURES



		Features x			
Worked Minutes Week (Target Variable)	y	**People in Office (Feature 1)**	x_1	**Salary (Feature 2)**	x_2
2220	$y^{(1)}$	4	$x_1^{(1)}$	4300 €	$x_2^{(1)}$
1800	$y^{(2)}$	12	$x_1^{(2)}$	2700 €	$x_2^{(2)}$
1920	$y^{(3)}$	5	$x_1^{(3)}$	3100 €	$x_2^{(3)}$

$n = 3$

$p = 2$

The whole data set is expressed by

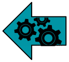
$$\mathcal{D} = \left\{ \left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(n)}, y^{(n)}\right) \right\}$$

with the *i*-th observation $\left(x^{(i)}, y^{(i)}\right) \in \mathcal{X} x \mathcal{Y}$.
$\mathcal{X}$ is called input and $\mathcal{Y}$ output or target space.

TARGET AND FEATURES RELATIONSHIP

- For our observed data we know which outcome is produced:

y
2200
1800
1920

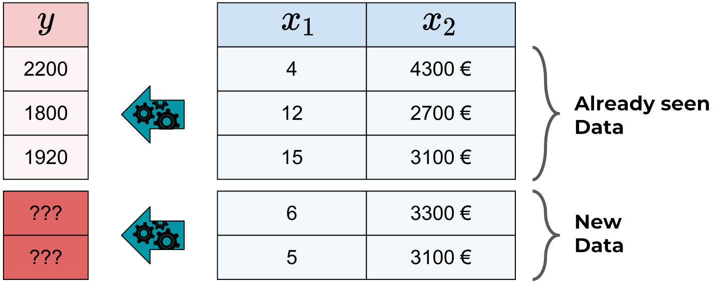


x_1	x_2
4	4300 €
12	2700 €
15	3100 €

Already seen Data

TARGET AND FEATURES RELATIONSHIP

- For new employees we can just observe the features:



y	x_1	x_2
2200	4	4300 €
1800	12	2700 €
1920	15	3100 €
???	6	3300 €
???	5	3100 €

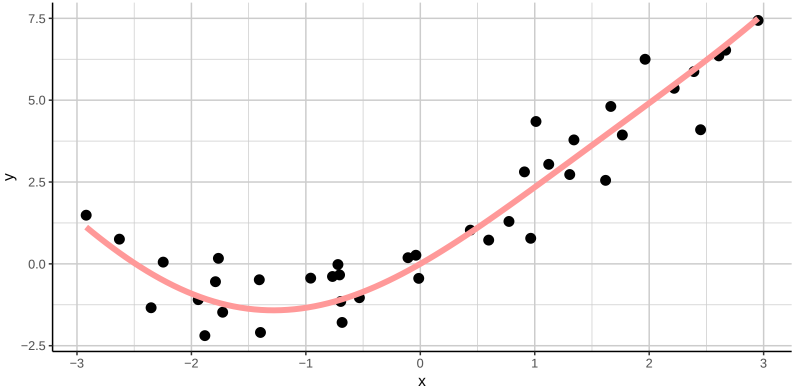
$\Rightarrow$ The goal is to predict the target variable for **unseen new data** by using a **model** trained on the already seen **train data**.

SUPERVISED LEARNING TASK

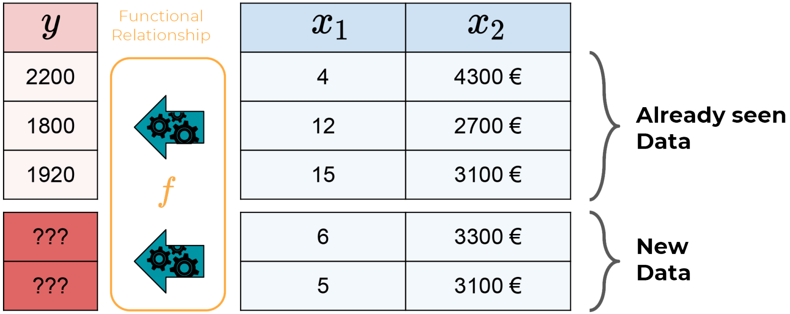
- **Regression**: Given an input x, predict corresponding output from $\mathcal{Y} \in \mathbb{R}^g, 1 \leq g < \infty$.
- **Classification**: Assigning an input x to one class of a finite set of classes $\mathcal{Y} = \{C_1, ..., C_g\}, 2 \leq g < \infty$.
- **Density estimation**: Given an input x, predict the probability distribution $p(y|x)$ on $\mathcal{Y}$.

REGRESSION TASK

- **Goal**: Predict a continuous output
- y is a metric variable (with values in $\mathbb{R}$)
- Regression model can be constructed by different methods, e.g., linear regression, trees or splines



TARGET AND FEATURES RELATIONSHIP



TARGET AND FEATURES RELATIONSHIP

- In ML, we want to be "lazy". We do not want to specify f manually.
- We want to learn it **automatically from labeled data**.
- Later we will see that we do have to specify something, like f's functional form and other stuff.
- Mathematically, we face a problem of function approximation: search for an f, such that, for all points in the training data and also all newly observed points

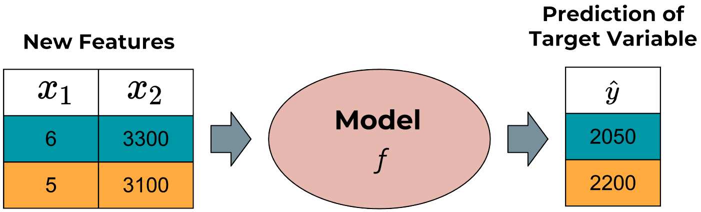
  $$\begin{tikzcd} \text{x} \arrow[r] & \boxed{f} \arrow[r] & \text{f(x)} \end{tikzcd}$$

  predicted outcomes are very close to real targets: $f(x) \approx y$

- We call this **supervised learning**.

WHAT IS A MODEL?

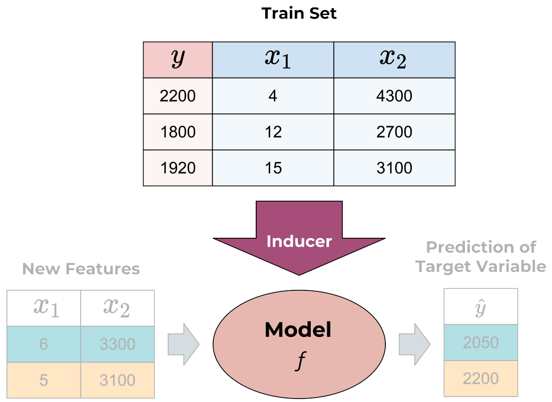
A model (or hypothesis) $f : \mathcal{X} \to \mathbb{R}^g$ maps inputs (or input features) to outputs (or targets).
A hypothesis class H is a set of such functions.



WHAT IS A LEARNER?

The **learner** (inducer, learning algorithm) takes our labeled data set (**training set**) and produces a model (which again is a function): Applying a learning algorithm means coming up with a hypothesis given sample data. Formally, it maps:

$$\{((x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}))\} \subset \mathcal{X} \times \mathcal{Y} \to H$$



Train Set

y	x_1	x_2
2200	4	4300
1800	12	2700
1920	15	3100

New Features

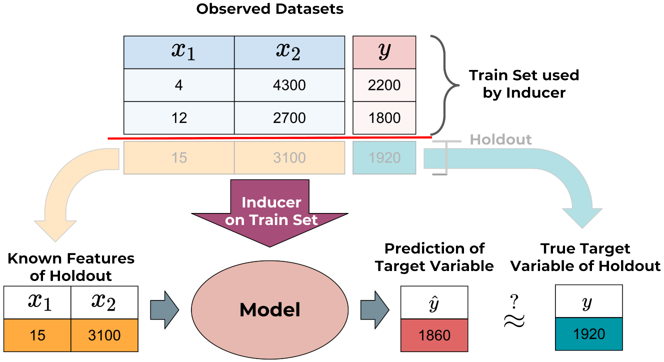
x_1	x_2
6	3300
5	3100

Prediction of Target Variable

$\hat{y}$
2050
2200

HOW TO EVALUATE MODELS

- Simply compare predictions from model with truth:



LEARNER DECOMPOSITION

Nearly all ML supervised learning training algorithms can be described by three components:

Learning = Hypothesis Space + Evaluation + Optimization

- **Hypothesis Space:** Defines functional structures of f we can learn.
- **Evaluation:** How well does a certain hypothesis score on a given data set? Allows us to choose better candidates over worse ones.
- **Optimization:** How do we search the hypothesis space? Guided by the evaluation metric.
- All of these components represent important choices in ML which can have drastic effects:
  If we make smart choices here, we can tailor our learner to our needs - but that usually requires quite a lot of experience and deeper insights into ML.

LEARNER DECOMPOSITION

Hypothesis Space	**Evaluation**	**Optimization**
Instances / Neighbours	Squared error	Gradient descent
Linear functions	Likelihood	Stochastic gradient descent
Decision trees	Information gain	Quadratic programming
Set of rules	K-L divergence	Greedy optimization
Neural networks		Combinatorial optimization
Graphical models		

Note: What is on the same line above does not belong together!

RISK MINIMIZATION

Let $\mathbb{P}_{xy}$ be the joint distribution of x and y. This defines all aspects of the generating process where our data comes from.

The quality of the prediction $y = f(x)$ of a model is measured by a *loss function* $L(y, f(x))$.

A function $L : \mathcal{Y} \times \mathcal{Y} \to [0, \infty[$ with the property $L(y, y) = 0$ for $y \in \mathcal{Y}$ is called a loss function.

A typical loss function is the squared error: $L(y, f(x)) = (y - f(x|\theta))^2$,

Its expectation is the so-called *risk*,

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

Obvious aim: Minimize $\mathcal{R}(f)$ over f.

RISK MINIMIZATION

But minimizing $\mathcal{R}(f)$ over f is not (in general) practical:

- $\mathbb{P}_{xy}$ is unknown.
- We could estimate $\mathbb{P}_{xy}$ in non-parametric fashion from the data D, e.g., by kernel density estimation, but this really does not scale to higher dimensions (see curse of dimensionality).
- We can efficiently estimate $\mathbb{P}_{xy}$, if we place rigorous assumptions on its distributional form, and methods like discriminant analysis work exactly this way. ML usually studies more flexible models.

RISK MINIMIZATION

An alternative (without directly assuming something about $\mathbb{P}_{xy}$) is to approximate $\mathcal{R}(f)$ based on the data $\mathcal{D}$, by means of the *empirical risk*

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

Learning then amounts to *empirical risk minimization*

$$\hat{f} = \arg\min_{f \in H} \mathcal{R}_{\text{emp}}(f).$$

RISK MINIMIZATION

When f is parameterized by θ, this becomes:

$$\begin{aligned}\mathcal{R}_{\text{emp}}(\theta) &= \frac{1}{n}\sum_{i=1}^{n} L\left(y^{(i)}, f\left(x^{(i)}|\theta\right)\right)\\ \hat{\theta} &= \arg\min_{\theta\in\Theta} \mathcal{R}_{\text{emp}}(\theta)\end{aligned}$$

Thus learning (often) means solving the above *optimization problem*. Which implies a tight connection between ML and optimization.
Note that (with a slight abuse of notation), if it is more convenient, and as there is no difference w.r.t. the minimizer, we might also define the $\mathcal{R}_{\text{emp}}$ in its non-average-but-instead-summed version as:

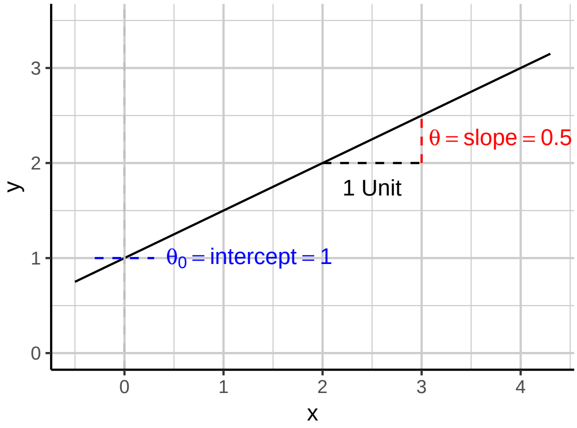
$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(x^{(i)}|\theta\right)\right)$$

RISK MINIMIZATION

- For regression, the loss usually only depends on residuals: $L(y, f(x)) = L(y - f(x)) = L(\epsilon)$, this is a *translation invariant* loss
- Choice of loss decides statistical properties of f: Robustness, error distribution
- Choice of loss decides computational / optimization properties of minimization of $\mathcal{R}_{\text{emp}}(\theta)$: Smoothness of objective, can gradient methods be applied, uni- or multimodality.
  If $L(y, f(x))$ is convex in its second arguments, and $f(x|\theta)$ is linear in θ, then $\mathcal{R}_{\text{emp}}(\theta)$ is convex. Hence every local minimum of $\mathcal{R}_{\text{emp}}(\theta)$ is a global one. If $L(y, f(x))$ not convex, R might have multiple local minima (bad!).

(Linear) Regression

STRAIGHT LINE / UNIVARIATE LINEAR MODEL



$$y = \theta_0 + \theta \cdot x$$

LINEAR REGRESSION: HYPOTHESIS SPACE

We want to learn a numerical target variable, by a linear transformation of the features. So with $\theta \in \mathbb{R}^p$ this mapping can be written:

$$y = f(x) = \theta_0 + \theta^T x$$

This restricts the hypothesis space H to all linear functions in θ:

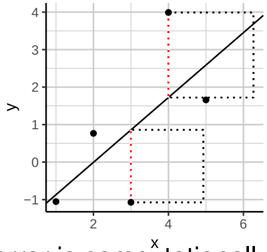
$$H = \{\theta_0 + \theta^T x \mid (\theta_0, \theta) \in \mathbb{R}^{p+1}\}$$

Given observed labeled data $\mathcal{D}$, how to find (θ, θ_0)? This is learning or parameter estimation.
NB: We assume here and from now on that θ_0 is included in θ.

LINEAR REGRESSION: EVALUATION

We now measure training error as sum-of-squared errors. This is also called the L2 loss, or L2 risk:

$$\mathcal{R}_{\text{emp}}(\theta) = \text{SSE}(\theta) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(x^{(i)}|\theta\right)\right) = \sum_{i=1}^{n}\left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

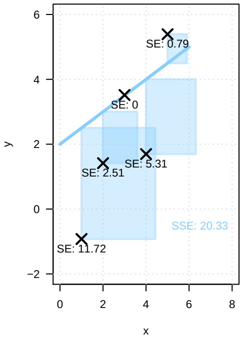


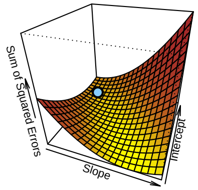
Optimizing the squared error is computationally much simpler than optimizing the absolute differences (this would be called L1 loss).

LINEAR REGRESSION: OPTIMIZATION

We want to find the parameters of the LM / an element of the hypothesis space H that best suits the data. So we evaluate different candidates for θ.

- A first (random) try yields an SSE of 20.33 (**Evaluation**).

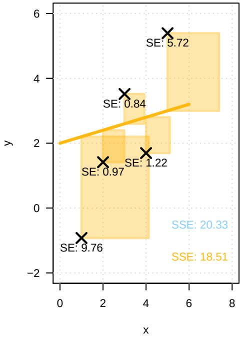


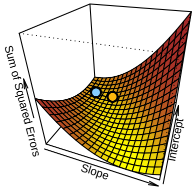


LINEAR REGRESSION: OPTIMIZATION

We want to find the parameters of the LM / an element of the hypothesis space H that best suits the data. So we evaluate different candidates for θ.

- Another line yields an lower SSE of 18.51 (**Evaluation**).





LINEAR REGRESSION: OPTIMIZATION

Instead of guessing parameters we could also use gradient descent to iteratively find the optimal θ. Here, we can even find the optimal value analytically:

$$\hat{\theta} = \arg\min_{\theta} \mathcal{R}_{\text{emp}}(\theta) = \arg\min_{\theta} \|y - X\theta\|_2^2$$

X is the $n \times (p+1)$-feature-data-matrix, also called design matrix. This yields the so called normal equations for the LM:

$$\frac{\delta}{\delta\theta}\mathcal{R}_{\text{emp}}(\theta) = 0 \qquad \Rightarrow \qquad \hat{\theta} = \left(X^T X\right)^{-1} X^T y$$

LINEAR REGRESSION: OPTIMIZATION

Differentiating $\mathcal{R}_{\text{emp}}(\theta) = \text{SSE}(\theta)$ w.r.t θ yields the so-called *normal equations*:

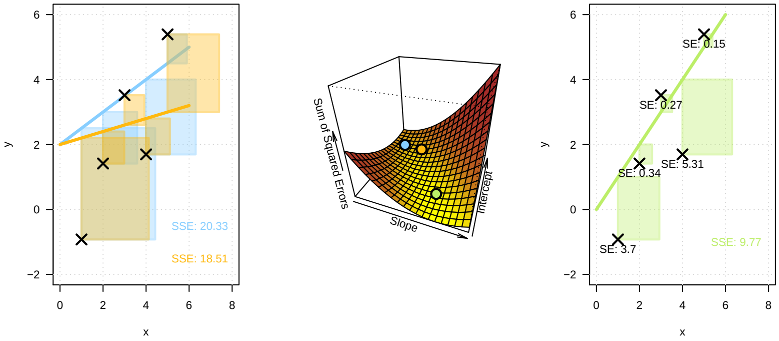
$$X^T(\mathbf{y} - X\theta) = 0$$

The optimal θ is

$$\hat{\theta} = (X^T X)^{-1} X^T \mathbf{y}$$

LINEAR REGRESSION: OPTIMIZATION

The optimal line has an SSE of 9.77 (**Evaluation**)



LINEAR REGRESSION: OPTIMIZATION

In statistics, we would start from a maximum-likelihood perspective

$$y^{(i)} = f\left(x^{(i)}\right) + \epsilon^{(i)} \sim N(f\left(x^{(i)}\right), \sigma^2)$$

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} p(y^{(i)}|f\left(x^{(i)}|\theta\right), \sigma^2) \propto \exp(-\frac{\sum_{i=1}^{n}(f\left(x^{(i)}|\theta\right) - y^{(i)})^2}{2\sigma^2})$$

If we minimize the neg. log-likelihood, we see that this is equivalent to our loss minimization approach!

$$\ell(\theta) \propto \sum_{i=1}^{n}(f\left(x^{(i)}|\theta\right) - y^{(i)})^2 = min!$$

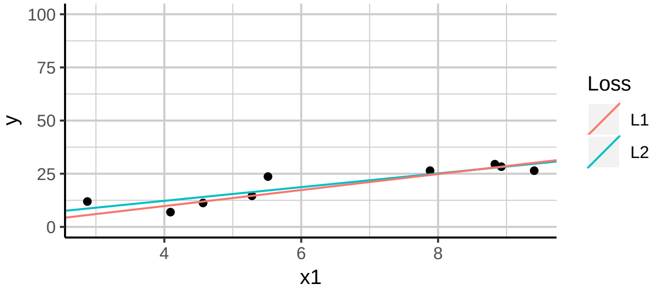
EXAMPLE: LINEAR REGR. WITH L1 VS L2 LOSS

We could also minimize the L1 loss. This changes the evaluation and optimization step:

$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(x^{(i)}|\theta\right)\right) = \sum_{i=1}^{n} |y^{(i)} - \theta^T x^{(i)}| \qquad \text{(Evaluation)}$$

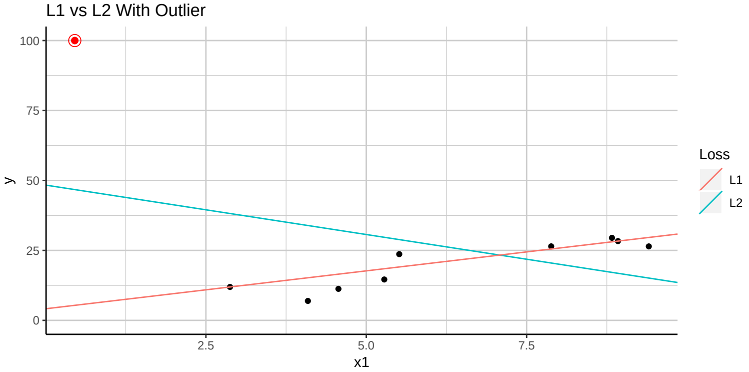
Much harder to optimize, but the model is less sensitive to outliers.

L1 vs L2 Without Outlier



EXAMPLE: LINEAR REGR. WITH L1 VS L2 LOSS

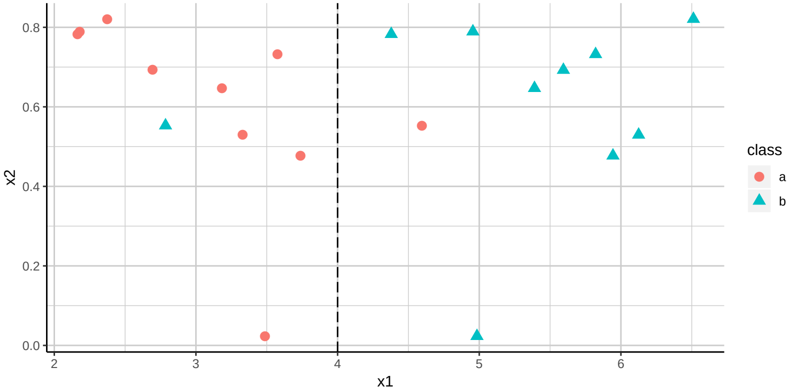
Adding an outlier (highlighted red) pulls the line fitted with L2 into the direction of the outlier:



Classification

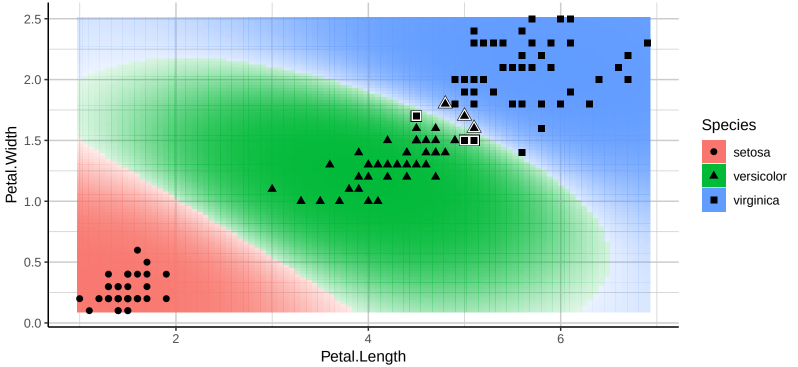
BINARY CLASSIFICATION TASK

- y is a categorical variable (with two values)
- E.g., sick/healthy, or credit/no credit
- **Goal**: Predict a class (or membership probabilities)



MULTICLASS CLASSIFICATION TASK

- *y* is a categorical variable with > 2 unordered values
- Each instance belongs to only one class
- **Goal**: Predict a class (or membership probabilities)



CLASSIFICATION: POSTERIOR PROBABILITIES

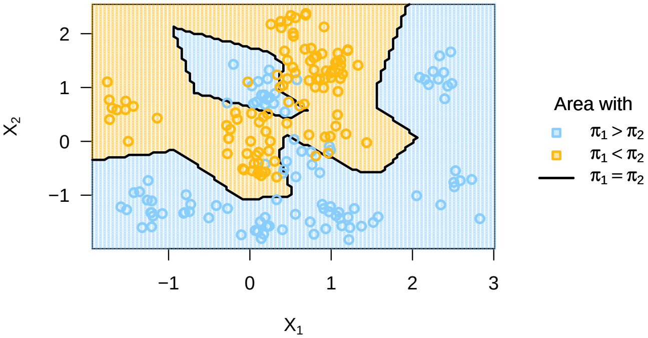
- Often estimating class probabilities for observations is more informative than predicting labels.
- Posterior probabilities are the probability of a class *k* occuring, given the feature *x*

$$\pi_k(x) = P(Y = k|x)$$

- Many models can directly output and optimize for these probabilities.
- Class estimation for x via $\arg\max_k \pi_k(x)$ - just take most likely class.

CLASSIFICATION: DECISION BOUNDARY

- Area in feature space where there is no unique best class with maximum posterior probability are equal, e.g., for binary classification $\pi(x)_1 = \pi(x)_2$
- Separates areas in which a specific class is estimated / chosen.



BINARY LABEL CODING

Remark: Notation in binary classification can be sometimes confusing because of different coding styles, and as we have to talk about predicted scores, classes and probabilities.

A binary variable can take only two possible values. For probability / likelihood-based model derivations a 0-1-coding, for geometric / loss-based models the -1+1-coding is often preferred.

- $\mathcal{Y} = \{0, 1\}$. Here, the approach often models $\pi(x)$, the posterior probability for class 1 given x. Usually, we then define $h(x) = [\pi(x) \geq 0.5] \in \mathcal{Y}$.
- $\mathcal{Y} = \{-1, 1\}$. Here, the approach often models $f(x)$, a real-valued score from $\mathbb{R}$ given x. Usually, we define $h(x) = \text{sign}(f(x)) \in \mathcal{Y}$, and we interpret |f(x)| as “confidence” for the predicted class $h(x)$.

LOGISTIC REGRESSION

A *discriminant* approach for directly modeling the posterior probabilities of the classes is *Logistic regression*. For now, we will only look at the binary case $y \in \{0, 1\}$. Note that we will supress the intercept in notation.

$$\pi(x) = \mathbb{P}(y = 1|x) = \frac{\exp(\theta^T x)}{1 + \exp(\theta^T x)} = \tau(\theta^T x)$$

We can either directly assume the above form, or arrive at it by requiring that the log-odds are linear in θ:

$$\log \frac{\pi(x)}{1 - \pi(x)} = \log \frac{\mathbb{P}(y = 1|x)}{\mathbb{P}(y = 0|x)} = \theta^T x.$$

The logistic function $\tau(t) = \frac{\exp(t)}{1+\exp(t)}$ transforms the scores / log-odds $\theta^T x$ into a probability.

LOGISTIC REGRESSION

Logistic regression is usually fitted by maximum likelihood.

$$\begin{aligned}
\mathcal{L}(\theta) &= \prod_{i=1}^{n} \mathbb{P}(y = y^{(i)} | x^{(i)}, \theta) \\
&= \prod_{i=1}^{n} \pi(x^{(i)}, \theta)^{y^{(i)}} [1 - \pi(x^{(i)}, \theta)]^{1 - y^{(i)}}.
\end{aligned}$$

$$\begin{aligned}
\ell(\theta) &= \sum_{i=1}^{n} y^{(i)} \log[\pi(x^{(i)}, \theta)] + (1 - y^{(i)}) \log[1 - \pi(x^{(i)}, \theta)] \\
&= \sum_{i=1}^{n} y^{(i)} \log[\exp(\theta^T x^{(i)})] - y^{(i)} \log[1 + \exp(\theta^T x^{(i)})] \\
&+ \quad (1 - y^{(i)}) \log\left[1 - \frac{\exp(\theta^T x^{(i)})}{1 + \exp(\theta^T x^{(i)})}\right] \\
&= \sum_{i=1}^{n} y^{(i)} \theta^T x^{(i)} - \log[1 + \exp(\theta^T x^{(i)})]
\end{aligned}$$

LOGISTIC REGRESSION

We can minimize our loss, by maximizing $\ell(\theta)$.
This now cannot be solved analytically, but is at least concave, so we have to refer to numerical optimization, e.g., gradient ascent.

To predicted class labels with minimal errors, we should probably predict $y = 1$, iff

$$\pi(x|\theta) = \frac{\exp(\theta^T x)}{1 + \exp(\theta^T x)} \geq 0.5,$$

which is equal to

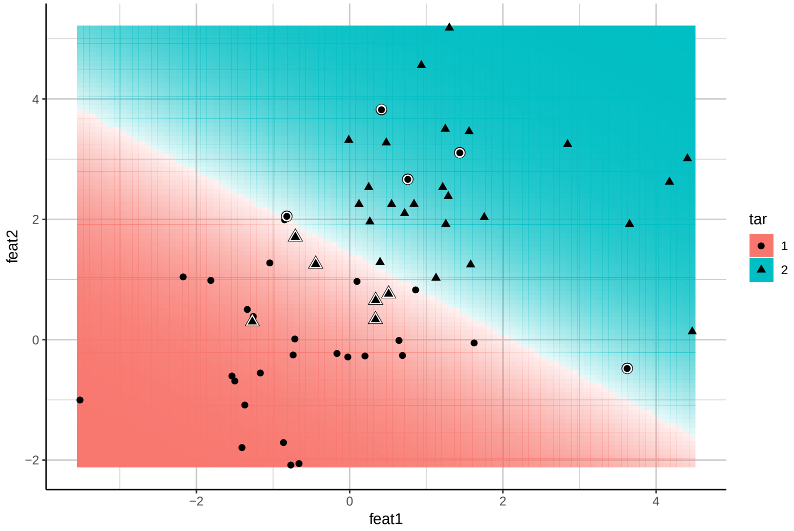
$$\theta^T x \geq 0.$$

So logistic regression gives us a *linear classifier*:

$$\hat{y} = h(\theta^T x) = \begin{cases} 1 & \text{for } x^T \theta \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

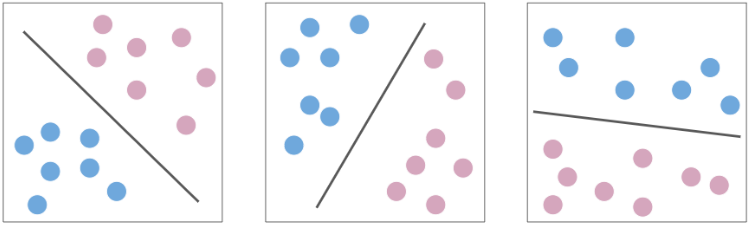
LOGISTIC REGRESSION

logreg: model=FALSE
Train: mmce=0.183; CV: mmce.test.mean=0.233



TYPES OF BOUNDARIES THAT CAN BE LEARNED

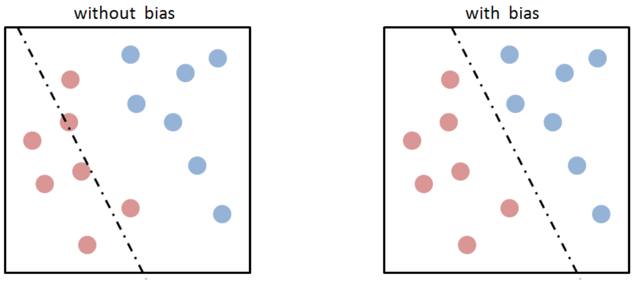
- Different values of θ map to different linear decision boundaries separating the two classes.



Note : Only two dimensions pictured here

TYPES OF BOUNDARIES THAT CAN BE LEARNED

- The intercept θ_0, is in ML often (unfortunately) called bias.
- It produces an affine shift on the linear decision boundary.



LIKELIHOOD AND LOSSES

Let us generalize what we have observed from the comparison between the maximum-likelihood and the risk-based approach linear regression. The maximum-likelihood principle is to maximize

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

or to minimize the neg. log-likelihood:

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

Now let us define a new loss as:

$$L\left(y, f(x|\theta)\right) = -\log p(y|x, \theta)$$

LIKELIHOOD AND LOSSES

And consider the empirical risk

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

Then the maximum-likelihood estimator $\hat{\theta}$, which we obtain by optimizing $\mathcal{L}(\theta)$ is identical to the loss-minimal $\hat{\theta}$ we obtain by minimizing $\mathcal{R}_{\text{emp}}(\theta)$. This implies that we can identify for every error distribution and its pdf $p(y|x, \theta)$ an equivalent loss function which leads to the same point estimator for the parameter vector θ. We can even disregard multiplicative or additive constants in the loss, as they do not change the minimizer.
The other way around does not always work: We cannot identify for every loss function and associated pdf for the distribution.

LIKELIHOOD AND LOSSES

Let us reconsider the logistic regression maximum likelihood fit. The neg. log-likelihood for the pdf is:

$$-\log p(y|x, \theta) = -y \log[\pi(x)] - (1 - y) \log[1 - \pi(x)]$$

This is the cross-entropy loss. Logistic regression minimizes this, and we could use this loss for any other model which directly models $\pi(x)$.

Now lets assume we have a score function $f(x)$ instead of $\pi(x)$. We can transform the score to a probability via the logistic transformation:

$$\begin{aligned} \pi(x) &= \frac{\exp(f(x))}{1 + \exp(f(x))} = \frac{1}{1 + \exp(-f(x))} \\ 1 - \pi(x) &= \frac{\exp(-f(x))}{1 + \exp(-f(x))} = \frac{1}{1 + \exp(f(x))} \end{aligned}$$

LIKELIHOOD AND LOSSES

The loss now becomes

$$\begin{aligned} -\log p(y|x,\theta) &= -y\log[\pi(x)] - (1-y)\log[1-\pi(x)] \\ &= y\log[1+\exp(-f(x))] + (1-y)\log[1+\exp(f(x)] \end{aligned}$$

For y=0 and y=1 this is:

$$\begin{aligned} y=0 &: \log[1+\exp(f(x)] \\ y=1 &: \log[1+\exp(-f(x))] \end{aligned}$$

If we would encode now with $\mathcal{Y} = \{-1,+1\}$, we can unify this like this:

$$L(y, f(x)) = \log[1+\exp(-yf(x)]$$

So we have recovered the Bernoulli loss. LR minimizes this, and we could use this loss for any other model with directly models $f(x)$.

ESTIMATING PROBABILITIES

In logistic regression, the logistic function transforms the log-odds $t = \theta^T x$ into a probability: $\tau(t) = \frac{\exp(t)}{1+\exp(t)}$.

$\tau(t)$ is a so-called sigmoid function. The above approach can be generalized, so other score-generating models $f(x)$ can be turned into probability estimators by using $\tau(f(x))$.

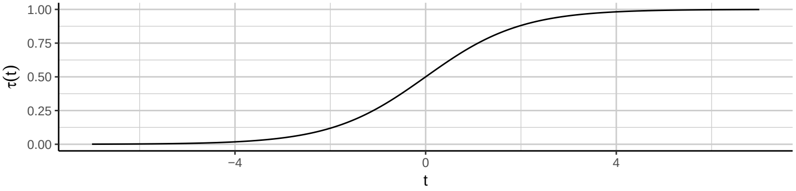
- f could be another model that outputs a score
- τ could be another sigmoid-function (i.e. the cdf of the standard normal distribution which leads to the *probit*-model)
- We have to choose an appropriate loss, e.g. binomial or cross-entropy loss. One (older) approach is also to simply use $\tau(f(x))$ as the model and then minimize the squared loss between $\hat{y}$ and the $(0, 1)$ labels.

ESTIMATING PROBABILITIES - SIGMOID

A *sigmoid* is a bounded, differentiable, real-valued function $\tau : \mathbb{R} \to (0, 1)$, that has a non-neg. derivative at each point.

Properties of logistic function $\tau(t)$:

- $\lim_{t\to\ -\infty} \tau(t) = 0$ and $\lim_{t\to\infty} \tau(t) = 1$
- $\frac{\partial\tau(t)}{\partial t} = \frac{\exp(t)}{(1+\exp(t))^2} = \tau(t)(1 - \tau(t))$
- $\tau(t)$ is symmetrical about the point $(0, 1/2)$



MULTINOMIAL REGRESSION AND SOFTMAX

For a categorical response variable $y \in \{1, \dots, g\}$ with $g > 2$ the model extends to

$$\pi_k(x) = \mathbb{P}(y = k|x) = \frac{\exp(\theta_k^T x)}{\sum_{j=1}^{g} \exp(\theta_j^T x)}.$$

The latter function is called the *softmax*, and defined on a numerical vector z:

$$\tau(z)_k = \frac{\exp(z_k)}{\sum_j \exp(z_j)}$$

It is a generalization of the logistic function (check for g=2). It "squashes" a g-dim. real-valued vector z to a vector of the same dimension, with every entry in the range [0, 1] and all entries adding up to 1.

MULTINOMIAL REGRESSION AND SOFTMAX

By comparing the posterior probabilities of two categories *k* and *l* we end up in a linear function (in *x*),

$$\log \frac{\pi_k(x)}{\pi_l(x)} = (\theta_k - \theta_l)^T x.$$

Remark:

- θ_j are vectors here.
- Well-definedness: $\pi_k(x) \in [0, 1]$ and $\sum_k \pi_k(x) = 1$

MULTINOMIAL REGRESSION AND SOFTMAX

This approach can be extended in exactly the same fashion for other score based models. For each class k we define a binary score model $f_k(x)$ with parameter vector θ_k. We then combine these models through the softmax function

$$\mathbb{P}(y = k|x) = \pi_k(x) = s_k(f_1(x), \dots f_g(x))$$

and optimize all parameter vectors of the f_k jointly. Maximum likelihood:

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} \prod_{k=1}^{g} \pi_k(x|\theta)^{[y=k]}$$

Negative log-likelihood turns this into empirical risk minimization:

$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{g} 1_{[y=k]} \log s_k(f_1(x|\theta_k), \dots f_g(x|\theta_g))$$

MULTINOMIAL REGRESSION AND SOFTMAX

Further comments:

- We can now, e.g., calculate gradients and optimize this with standard numerical optimization software.
- For linear $f(x|\theta) = \theta^T x$ this is also called *softmax regression*.
- Has an unusual property that it has a "redundant" set of parameters. If we substract a fixed vector from all θ_k, the predictions do not change at all. I.e., our model is "overparameterized", and 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 (but $\mathcal{R}_{\text{emp}}(\theta)$ is convex)! Hence, a numerical trick is to set $\theta_g = 0$ and only optimize the other θ_k.
- A similar approach is used in many ML models: multiclass LDA, naive Bayes, neural networks and boosting.

Performance Measures and Generalization

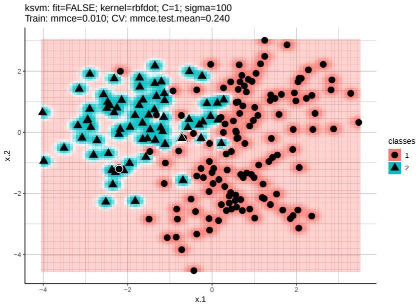
OVERFITTING

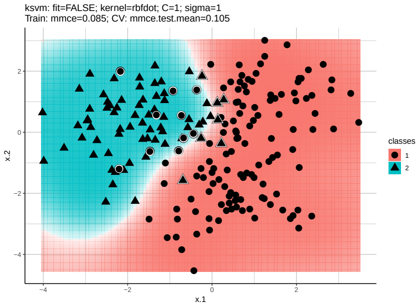
- Learner finds a pattern in the data that is not actually true in the real world: *overfits* the data
- Every powerful learner can “hallucinate” patterns
- Happens when you have too many hypotheses and not enough data to tell them apart
- The more data, the more “bad” hypotheses are eliminated
- If the hypothesis space is not constrained, there may never be enough data
- There is often a parameter that allows you to constrain (*regularize*) the learner

OVERFITTING

Overfitting learner

Non-overfitting learner





Better training set performance (seen examples)

Better test set performance (unseen examples)

OVERFITTING AND NOISE

- Overfitting is seriously exacerbated by *noise* (errors in the training data)
- An unconstrained learner will start to model that noise
- It can also arise when relevant features are missing in the data
- In general it’s better to make some mistakes on training data (“ignore some observations”) than trying to get all correct
- To avoid overfitting:
  - Some learners can do “early stopping” before perfectly fitting (i.e., overfitting) the training data
  - In general, prefer simpler hypotheses altogether when they work well (e.g. regularization, pruning)

TRIPLE TRADE-OFF

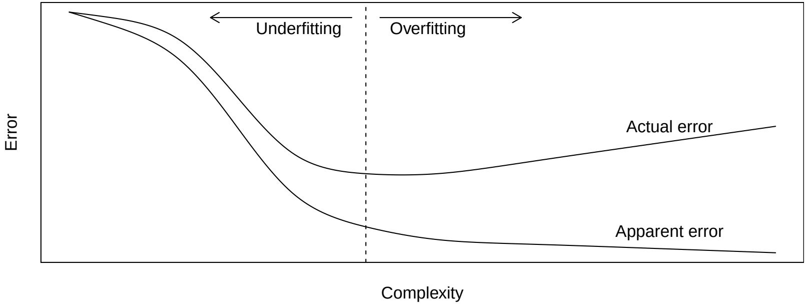
In all learning algorithms that are trained from data, there is a trade-off between three factors:

- The complexity of the hypothesis we fit to the training data
- The amount of training data (in terms of both instances and informative features)
- The generalization error on new examples

If the capacity of the learning algorithm is large enough to a) approximate the data generating process and b) exploit the information contained in the data, the generalization error will decrease as the amount of training data increases.
For a fixed size of training data, the generalization error decreases first and then starts to increase (overfitting) as the complexity of the hypothesis space H increases.

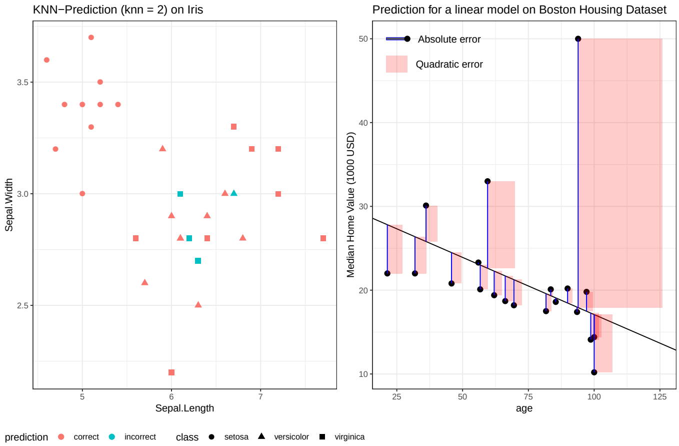
TRADE-OFF BETWEEN GENERALIZATION ERROR AND COMPLEXITY

Apparent error (on the training data) and real error (prediction error on new data) evolve in the opposite direction with increasing complexity:



$\Rightarrow$ Optimization regarding the model complexity is desirable: Find the right amount of complexity for the given amount of data where generalization error becomes minimal.

PERFORMANCE MEASURES



To compare the performance of two ML models on a single observation or a new data set, we would ideally like to boil them down to a **single** numeric measure of the performance.

PERFORMANCE MEASURES

Performance measures are related to losses, but are not the same.

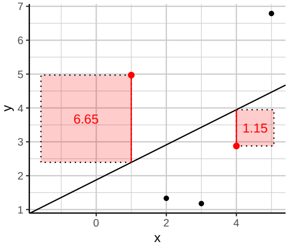
- A performance measure is simply applied at the end, to a fitted model, and some test data under consideration. A loss function is optimized by a learning algorithm.
- A performance measure is usually dictated by our prediction problem. There are no technical restrictions on it, except that it needs to be computable and appropriate, as it is only measured on a model and data set, to evaluate the model. A loss function on the other hand needs to be “handable” by the optimizer, so a certain degree of smoothness is usually required.
- Sometimes, performance mearures arer called “outer losses”, and “innner loss” is used for the loss minimized by the learner.
- Sometime, one can make the inner loss match the outer loss, so the learning algorithm optimizes exactly the metric we are interested in. But often we use approximations for the inner loss, e.g., for efficiency.

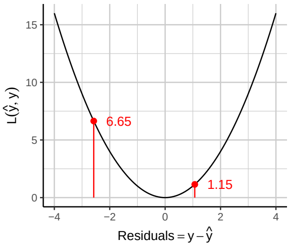
REGRESSION: MSE

The **Mean Squared Error** compares the mean of the squared distances between the target variable y and the predicted target $\hat{y}$.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y})^2 \in [0; \infty]$$

Single observations with a large prediction error heavily influence the **MSE**, as they enter quadratically.



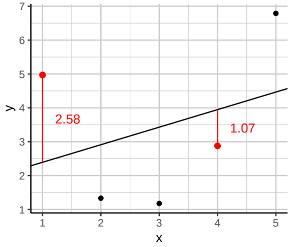


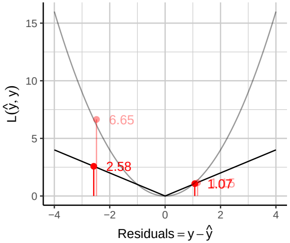
REGRESSION: MAE

A more robust (but not neccessarily better) way to compute a performance measure is the **Mean Absolute Error**:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y^{(i)} - \hat{y}| \in [0; \infty]$$

The absolute error is more robust. On the other hand, the absolute error is not differentiable at 0.





REGRESSION: R^2

Another often used measure is R^2. It measures the *fraction of variance explained* by the model.

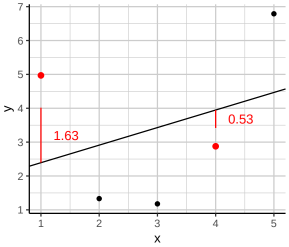
$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y^{(i)} - \hat{y})^2}{\sum_{i=1}^{n}(y^{(i)} - \bar{y})^2} = \frac{SSE_{Model}}{SSE_{Intercept}}$$

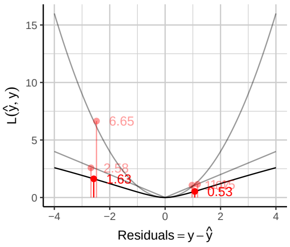
- If R^2 is measured on the training data: $R^2 \in [0; 1]$.
- If R^2 is measured on held out data: $R^2 \in [-\infty; 1]$.

REGRESSION: DEFINING A CUSTOM LOSS

Assume a use case, where the target variable can have a wide range of values across different orders of magnitude. A possible solution would be to use a loss functions that allows for better model evaluation and comparison. The **Mean Squared Logarithmic Absolute Error** is not strongly influenced by large values due to the logarithm.

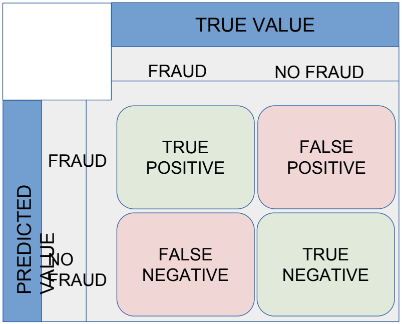
$$\frac{1}{n}\sum_{i=1}^{n}(\log(|y^{(i)} - \hat{y}| + 1))^2$$





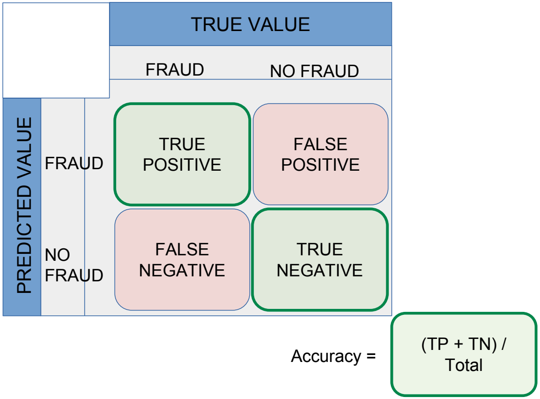
PERFORMANCE MEASURES - CLASSIFICATION

The confusion matrix is an intuitive metric used for establishing the correctness and accuracy of a model. It is used for classification problems where the output can be of two or more types of classes.



CLASSIFICATION: ACCURACY / ERROR

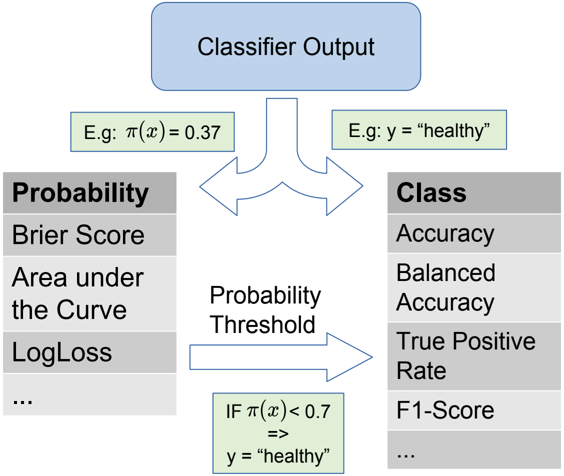
The confusion matrix is not a performance measure as such, but most performance metrics for binary classification are based on it.



Classification Error = 1 - Accuracy

THRESHOLD

Thresholding flexibly converts measured probabilities to labels.

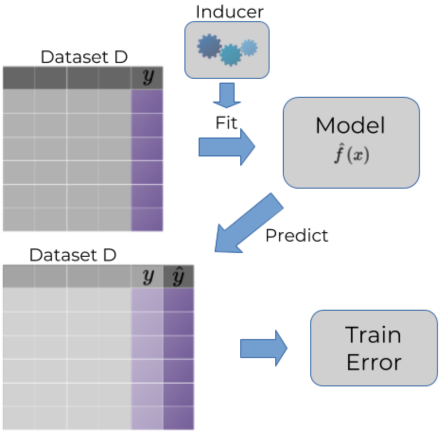


CLASSIFICATION PERFORMANCE MEASURES

Classification	**Explanation**
Accuracy	Fraction of correct classifications
Balanced Accuracy	Fraction of correct classifications in each class
Recall	Fraction of positives a classifier captures
Precision	Fraction of positives in instances predicted as positive
F1-Score	Tradeoff between precision and recall
AUC	Measures calibration of predicted Probabilities
Brier Score	Squared difference between predicted probability and true label
Logloss	Emphasizes errors for predicted probabilities close to 0 and 1

TRAINING ERROR

The *training error* (also called apparent error or resubstitution error) is estimated by the average error over the training set $\mathcal{D}_{\text{train}}$:



TRAINING ERROR

- The training error is usually a very unreliable and overly optimistic estimator of future performance.
- Goodness-of-fit measures like R^2, likelihood, AIC, BIC, deviance are all based on the training error.

EXAMPLE: POLYNOMIAL REGRESSION

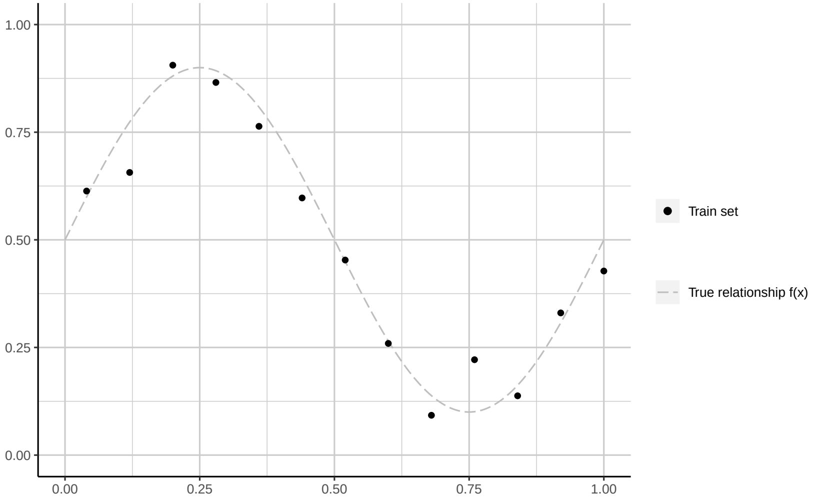
Assume we have a single, univariate x and that y can be approximated by a dth-order polynomial

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

θ_j are the coefficients and d is called the degree.
This is still linear regression / a linear model, and for a fixed d, the θ can easily be obtained through the normal equations.

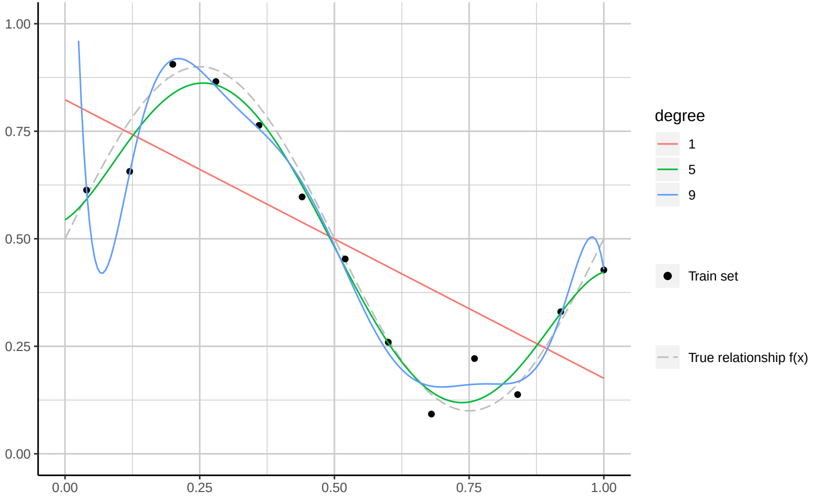
EXAMPLE: POLYNOMIAL REGRESSION

True relationship is $f(x) = 0.5 + 0.4 \cdot \sin(2\pi x) + \epsilon$



EXAMPLE: POLYNOMIAL REGRESSION

Models of different *complexity*, i.e., of different orders of the polynomial are fitted. How should we choose *d*?



EXAMPLE: POLYNOMIAL REGRESSION

The performance of the models on the training data can be measured by calculating the *training error* according to the mean squared error (MSE) criterion:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

d = 1: 0.03, d = 5: 0.002, d = 9: 0.001
Simply using the training error seems to be a bad idea.

TRAINING ERROR PROBLEMS

- Extend any ML training in the following way: On top of normal fitting, we also store $\mathcal{D}_{\text{train}}$. During prediction, we first check whether x is already stored in this set. If so, we replicate its label. The training error of such an (unreasonable) procedure will always be zero.
- The training error of 1-NN is always zero.
- The training error of an interpolating spline in regression is always zero.
- For models of severely restricted capacity, and given enough data, the training error might provide reliable information. E.g. consider a linear model in 5d, with 10.000 training points. But: What happens if we have less data? And p becomes larger? Can you precisely define where the training error becomes unreliable?

TEST ERROR AND HOLD-OUT SPLITTING

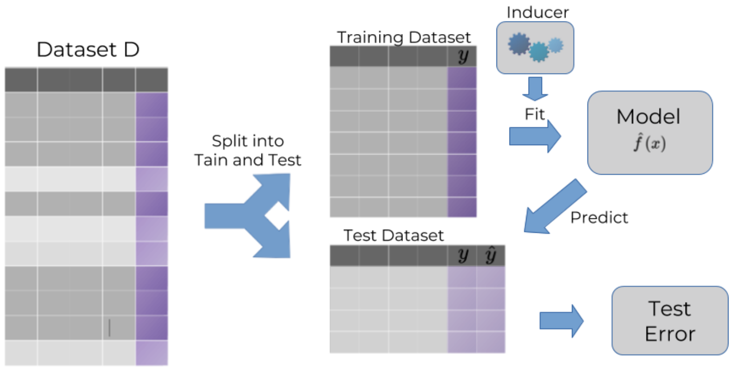
To reliably assess a model, we need to define

- how to simulate the scenario of “new unseen data” and
- how to estimate the generalization error.

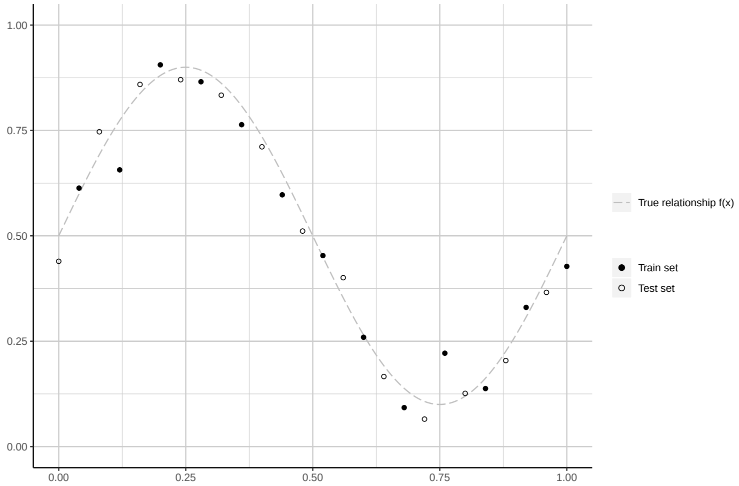
Hold-out splitting and evaluation

- Straightforward idea: To measure performance, let’s simulate how our model will be applied on new, unseen data
- So, to evaluate a given model: predict and evaluate only on data not used during training
- For a given set $\mathcal{D}$, we have to preserve some data for testing that we cannot use for training, hence we need to define a training set $\mathcal{D}_{\text{train}}$ and a test set $\mathcal{D}_{\text{test}}$, e.g. by randomly partitioning the original $\mathcal{D}$

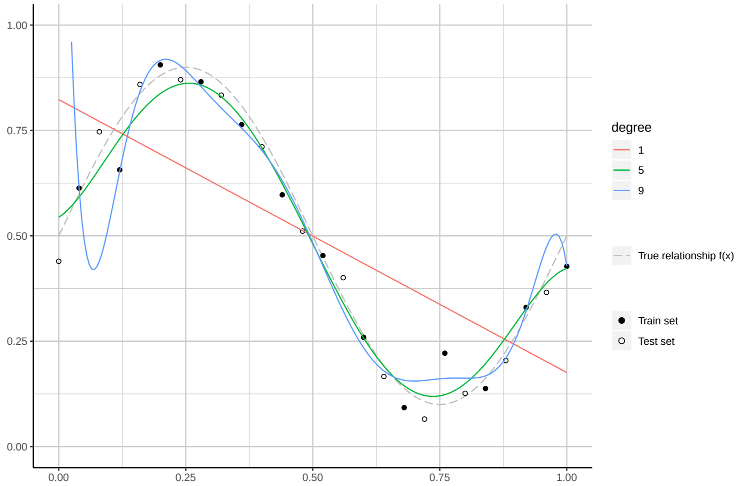
TEST ERROR AND HOLD-OUT SPLITTING



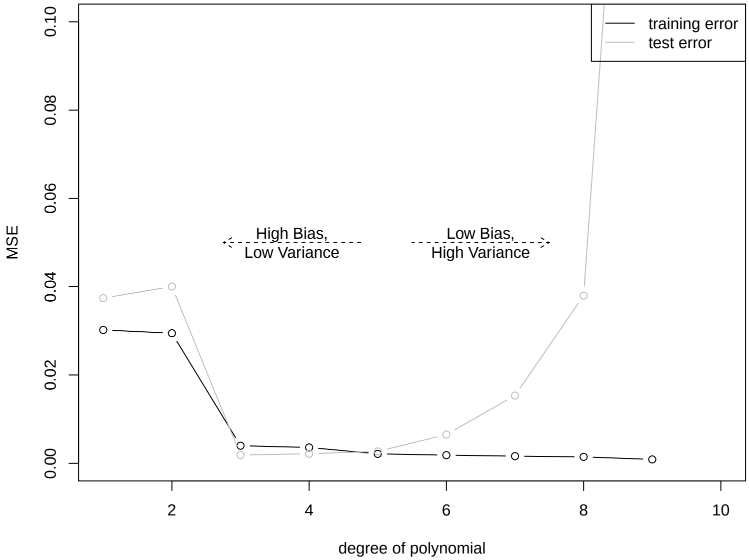
TEST ERROR AND HOLD-OUT SPLITTING



TEST ERROR AND HOLD-OUT SPLITTING



TEST ERROR AND HOLD-OUT SPLITTING



Test error is best for $d = 3$

TRAINING VS. TEST ERROR

The training error

- is an over-optimistic (biased) estimator as the performance is measured on the same data the learned prediction function $\hat{f}_{\mathcal{D}_{\text{train}}}(x)$ was trained for.
- decreases with smaller training set size as it is easier for the model to learn the underlying structure in the training set perfectly.
- decreases with increasing model complexity as the model is able to learn more complex structures.

TRAINING VS. TEST ERROR

The test error

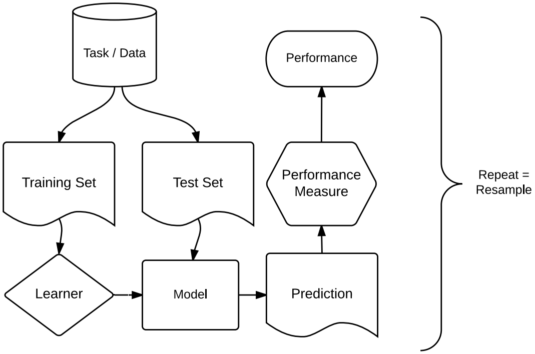
- will typically decrease when the training set increases as the model generalizes better with more data.
- will have higher variance with decreasing test set size.
- will have higher variance with increasing model complexity.

RESAMPLING

- We need to construct a better performance estimator through *resampling*, that uses the data more efficiently.
- All resampling variants operate similar: The data set is split repeatedly into training and tests sets, and we later aggregate (e.g. average) the results.
- The usual trick is to make training sets quite larger (to keep the pessimistic bias small), and to handle the variance introduced by smaller test sets through many repetitions and averaging of results.

RESAMPLING

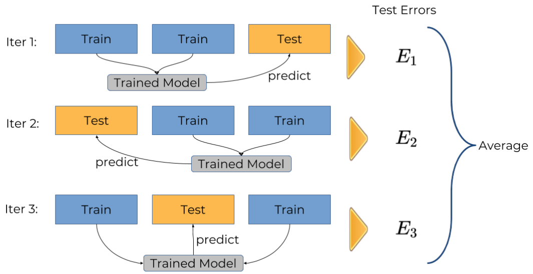
- Aim: Assess the performance of learning algorithm.
- Uses the data more efficiently then simple train-test.
- Repeatedly split in train and test, then aggregate results.



CROSS-VALIDATION

- Split the data into k roughly equally-sized partitions.
- Use each part once as test set and joint $k-1$ other as train.
- Obtain k test errors and average.

Example: 3-fold cross-validation:



CROSS-VALIDATION - COMMENTS

- $k = n$ is known as leave-one-out (LOO) cross-validation or jackknife.
- The performance estimates for each fold are NOT independent, because of the structured overlap of the training sets. Hence, the variance of the estimator increases again for very large k (close to LOO), when training sets nearly completely overlap.
- LOO is nearly unbiased, but has high variance.
- Repeated k-fold CV (multiple random partitions) can improve error estimation for small sample size.

SUMMARY OF CONCEPTS

- In ML we fit, at the end, a model on all our given data.
- Problem: We need to know how well this model performs in the future. But no data now is left to reliably do this. But we really need this.
- In order to approximate this, we do the next best thing. We estimate how well the learner works when it sees nearly *n* points from the same data distribution.
- Holdout, CV, resampling estimate exactly this number. The “pessimistic bias” refers to when use much less data in fitting than *n*. Then we “hurt” our learner unfairly.

SUMMARY OF CONCEPTS

- Strictly speaking, resampling only produces one number, the performance estimator. It does NOT produce models, paramaters, etc. These are intermediate results and discarded.
- The model and parameters are obtained when we fit the learner finally on the complete data.
- This is a bit weird and complicated, but we have to live with this.

COMMENTS

- 5CV or 10CV have become standard, 10-times repeated 10CV for small sample sizes, but THINK about whether that makes sense in your application.
- Do not use Hold-Out, CV with few iterations, or subsampling with a low subsampling rate for small samples, since this can cause the estimator to be extremely biased, with large variance.
- A $\mathcal{D}$ with $|\mathcal{D}| = 100.000$ can have small sample size properties if one class has only 100 observations ...

Regularization

CURSE OF DIMENSIONALITY

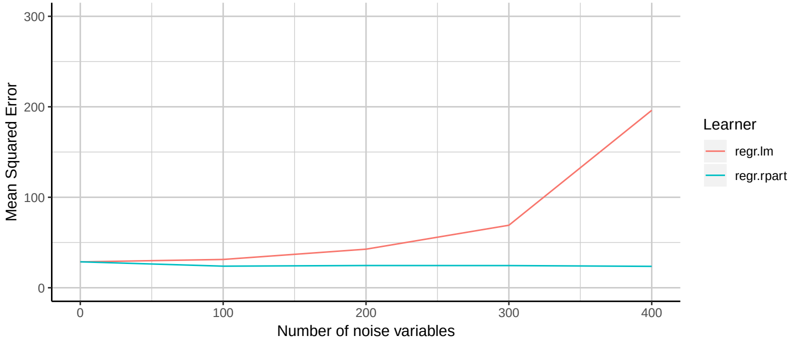
We investigate how the linear model behaves in high dimensional spaces.

We take the Boston Housing data set, where the value of houses in the area around Boston is predicted based on 14 features describing the region (e.g., crime rate, status of the population, etc.).

We train a linear model on the data. We then add $100, 200, 300, ...$ noise variables (containing no information at all) and look at the performance of a linear model trained on this modified data (10 times repeated 10-fold CV).

CURSE OF DIMENSIONALITY

We compare the performance of an LM to that of a regression tree.



$\rightarrow$ The unregularized LM struggles with the added noise features, while our tree seems to nicely filter them out

CURSE OF DIMENSIONALITY

Many basic ML methods with the curse of dimensionality. A large part of ML is concerned with dealing with this problem and finding ways around it.

Possible approaches are:

- Increasing the space coverage by gathering more observations (not always viable in practice!)
- Reducing the number of dimensions before training (e. g. PCA or feature selection)
- Regularization

REGULARIZATION

Assume we want to predict the daily maximum **ozone level** in LA 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 use (a subset of) the `Ozone` data set from the `mlbench` package.

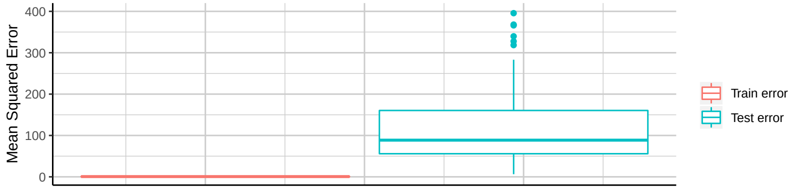
REGULARIZATION

We fit a linear regression model using **all** of the features

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

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

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



REGULARIZATION

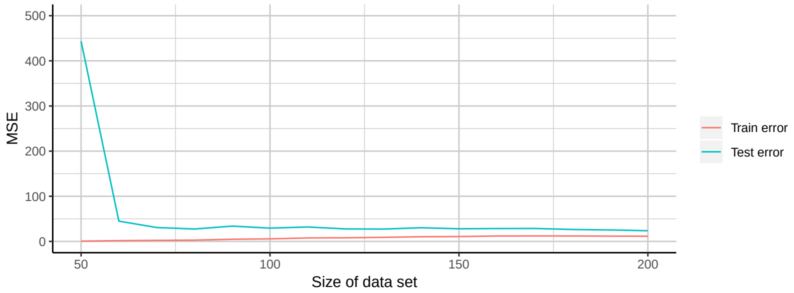
Why can **overfitting** happen? And how to avoid it?

1. Not enough data $\rightarrow$ collect **more data**
2. Data is noisy $\rightarrow$ collect **better data** (reduce noise)
3. Models are too complex $\rightarrow$ use **less complex models**
4. Aggressive loss optimization $\rightarrow$ **optimize less**

REGULARIZATION

Approach 1: Collect more data

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



Good idea, but often not feasible in practice.

REGULARIZATION

Approach 2: Reduce **complexity**

We try the simplest model we can think of: We do not use any features. We always predict the empirical mean

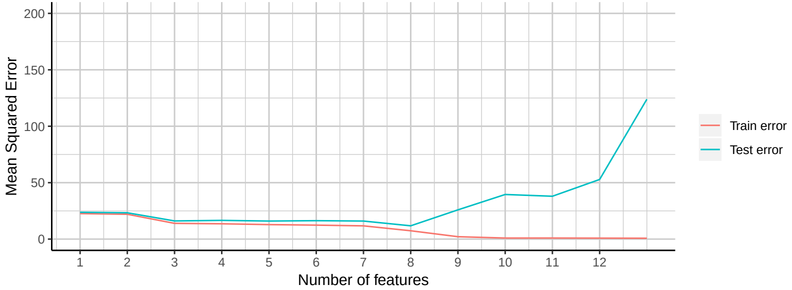
$$f(x|\theta) = \theta_0 = \frac{1}{n}\sum_{i=1}^{n} y^{(i)}$$

(intercept only model, featureless predictor).

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

REGULARIZATION

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



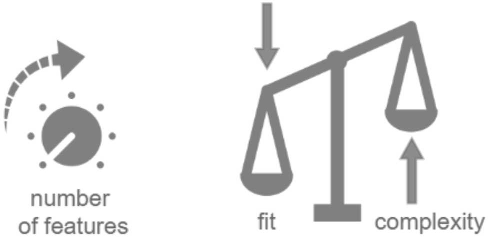
Note: For simplicity, we added the features in one specific order - but there are $2^{12} = 4096$ potential feature combinations.

REGULARIZATION: COMPLEXITY VS. FIT

We have have contradictory goals

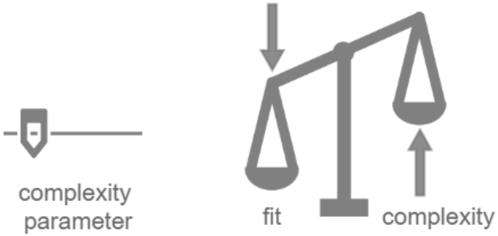
- **maximizing the fit** (minimize the train loss)
- **minimizing the complexity** of the model.

We need to find the “sweet spot”.



REGULARIZATION: COMPLEXITY VS. FIT

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.



REGULARIZATION IN THE LINEAR MODEL

Instead of pure **empirical risk minimization**, we add a penalty for complex (read: large) parameters θ_j:

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

The parameter λ is the complexity control parameter. If $\lambda = 0$, $\mathcal{R}_{\text{reg}}(\theta)$ reduces to simple empirical risk minimization, so here a simple LM.

RIDGE REGRESSION

If we choose $J(\theta)$ to be the L_2-penalty$^{(*)}$, the model is called **Ridge Regression**.

It tries to *shrink* the regression coefficients θ towards 0. We minimize:

$$\mathcal{R}_{ridge}(\theta) = \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)})^2 + \lambda ||\theta||_2^2$$

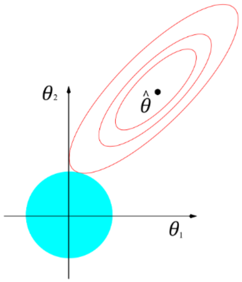
$^{(*)}$ The L_2 norm is $||\theta||_2^2 = \theta_1^2 + \theta_2^2 + ... + \theta_p^2$

RIDGE REGRESSION

Equivalent formulation as constrained optimization problem:

$$\min_\theta \qquad \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)})^2$$

$$\text{subject to:} \quad ||\theta||_2^2 \leq \lambda$$



LASSO REGRESSION

Another shrinkage method is the so-called **Lasso regression**, which uses an L_1 penalty$^{(*)}$ on θ. We minimize:

$$\mathcal{R}_{lasso}(\theta) = \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)})^2 + \lambda ||\theta||_1$$

Note that optimization becomes harder now, because the problem is not differentiable anymore.

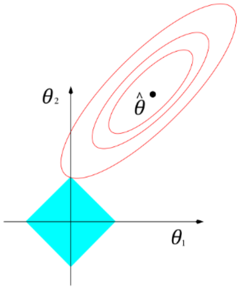
$^{(*)}$ The L_1 norm is $||\theta||_1 = |\theta_1| + |\theta_2| + ... + |\theta_p|$

LASSO REGRESSION

Equivalent formulation as constrained optimization problem:

$$\min_\theta \quad \sum_{i=1}^{n}(y^{(i)} - \theta^T x^{(i)})^2$$

$$\text{subject to:} \quad ||\theta||_1 \leq \lambda$$



RIDGE VS. LASSO REGRESSION

Sparsity of the solution:

- Lasso regression ensures a **sparse** solution (it selects features)
- Ridge regression includes all features

High-dimensional feature spaces ($p \gg n$):

- Lasso selects at most n features (even if more features are associated with the target)
- Ridge regression performs well in high-dimensional feature spaces

Correlated features:

- If features are highly correlated, Lasso selects only one of these and ignores the others

EXAMPLE: POLYNOMIAL RIDGE REGRESSION

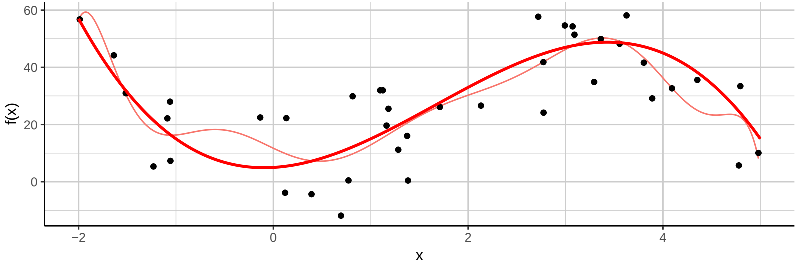
Again, let us consider a *d*th-order polynomial

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

True relationship is $f(x) = 5 + 2x + 10x^2 - 2x^3 + \epsilon$

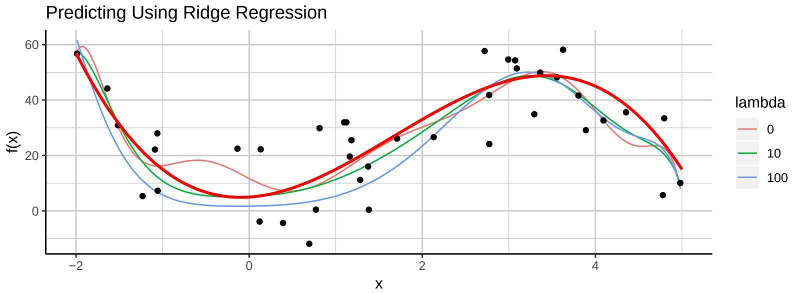
Making predictions for $d = 10$ with linear regression tends to overfit:

Predicting Using Linear Regression



EXAMPLE: POLYNOMIAL RIDGE REGRESSION

Using Ridge Regression with different penalty terms approximates the real data generating process better:



REGULARIZATION

We now generalize the concept of regularization:

- Instead of the LM, any model f could be considered
- Instead of the MSE, any loss function L can be considered
- Instead of the L_1 or L_2 penalty on θ, we add a penalty for complex models $J(\theta)$ (*complexity penalty* or *regularizer*)

$$\mathcal{R}_{\text{reg}}(\theta) = \mathcal{R}_{\text{emp}}(\theta) + \lambda \cdot J(\theta)$$

REGULARIZATION

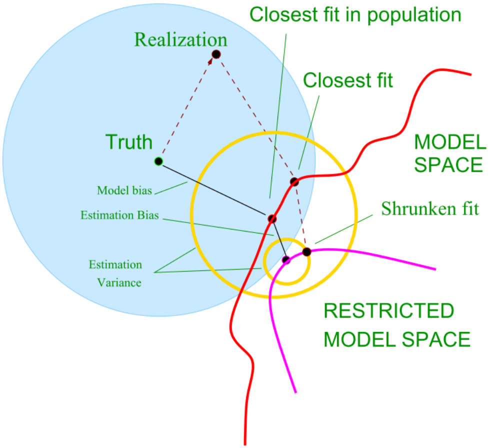


Figure: Hastie, The Elements of Statistical Learning, 2009

REGULARIZED RISK MINIMIZATION

We now have extreme flexibility to make appropriate choices in $\mathcal{R}_{\text{reg}}$:

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

- the **model structure** for f, affects how features influence y
- the **loss function** that measures how error should be treated
- **regularization** $J(\theta)$ that encodes our inductive bias and preference for certain simpler models

REG. RISK MIN.: WEIGHT DECAY

Let's optimize the L2-regularized risk of a model $f(x|\theta)$

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

by gradient descent.

$$\nabla \mathcal{R}_{\text{reg}}(\theta) = \nabla \mathcal{R}_{\text{emp}}(\theta) + 2\lambda\theta$$

and iteratively update θ by step size α times the negative gradient

$$\theta^{(new)} = \theta^{(old)} - \alpha \left(\nabla \mathcal{R}_{\text{emp}}(\theta) + \lambda\theta^{(old)}\right) = \theta^{(old)}(1 - \alpha\lambda) - \alpha \nabla \mathcal{R}_{\text{emp}}(\theta)$$

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

REGULARIZED LOGISTIC REGRESSION

We can also add a regularizer to the risk of logistic regression

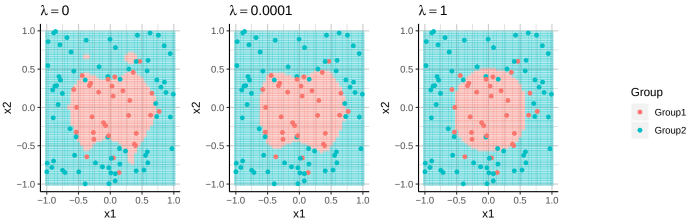
$$
\begin{aligned}
\mathcal{R}_{\text{reg}}(\theta) &= \mathcal{R}_{\text{emp}}(\theta) + \lambda \cdot J(\theta) \\
&= \sum_{i=1}^{n} \log\left(1 + \exp\left(-2y^{(i)} f\left(x^{(i)} \middle| \theta\right)\right)\right) + \lambda \cdot J(\theta)
\end{aligned}
$$

The other parts of the logistic regression remains exactly the same except for the fitting algorithm to find $\hat{\theta}$.

REGULARIZED LOGISTIC REGRESSION

We fit a logistic regression model using polynomial features for x_1 and x_2 with maximum degree of 7. We add an L2 penalty. We see

- $\lambda = 0$: the unregularized model seems to overfit
- $\lambda = 0.0001$: regularization helps to learn the underlying mechanism
- $\lambda = 1$ displays the real data generating process very well



FURTHER COMMENTS

- Note that very often we do not include θ_0 in the penalty term $J(\theta)$ (but this can be implementation-dependent).
- These methods are not equivariant under scaling of the inputs, so one usually standardizes the features.
- The regularization parameter λ is a hyperparameter that needs to be chosen carefully. One way is to tune it cross-validation.

REGULARIZED RISK MINIMIZATION VS. BAYES

We have already created a link between maximum likelihood estimation and empirical risk minimization. Now we will generalize this for regularized risk minimization.

Assume we have a parametrized distribution $p(x|\theta)$ for our data and a prior $p(\theta)$ over our parameter space, all in the Bayesian framework. With Bayes theorem we know:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} \propto p(x|\theta)p(\theta)$$

REGULARIZED RISK MINIMIZATION VS. BAYES

The maximum a-posterio estimator (MAP) of θ is now the minimizer of

$$-\sum_{i=1}^{n} \log p(x^{(i)}|\theta) - \log p(\theta).$$

Again, we identify the loss $L\left(y, f(x|\theta)\right)$ with $-\log(p(x|\theta))$. If $p(\theta)$ is constant, so we have a uniform, non-informative prior, we again arrive at empirical risk minimization.

If not, we can identify $-log(p(\theta))$, our prior, with our regularizer $J(\theta)$. If we introduce a further control constant λ in front of the prior, we arrive at regularized risk minimization.

Example: Rigde regression with linear model $f(x|\theta)$

- L2 loss = normal distribution of error terms
- L2 regularizer = Gaussian prior on θ