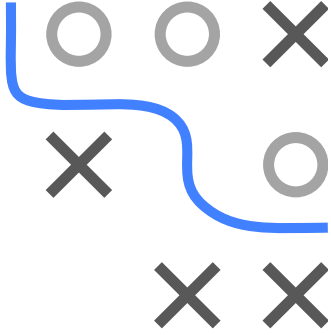
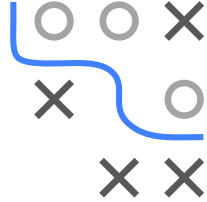


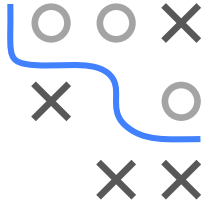


Introduction to Machine Learning

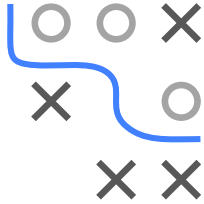
All slides

November 9, 2024

INTRODUCTION TO MACHINE LEARNING



Introduction to Machine Learning

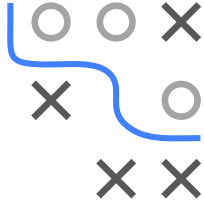


ML-Basics
What is Machine Learning?

Learning goals

- Understand basic terminology of and connections between ML, AI, DL and statistics
- Know the main directions of ML: Supervised, Unsupervised and Reinforcement Learning

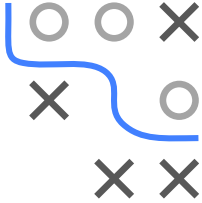
MACHINE LEARNING IS CHANGING OUR WORLD

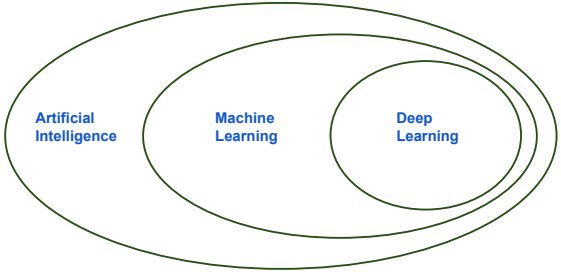


- Search engines learn what you want
- Recommender systems learn your taste in books, music, movies,...
- Algorithms do automatic stock trading
- Google Translate learns how to translate text
- Siri learns to understand speech
- DeepMind beats humans at Go
- Cars drive themselves
- Smart-watches monitor your health
- Election campaigns use algorithmically targeted ads to influence voters
- Data-driven discoveries are made in physics, biology, genetics, astronomy, chemistry, neurology,...
- ...

THE WORLD OF ARTIFICIAL INTELLIGENCE

... and the connections to Machine Learning and Deep Learning

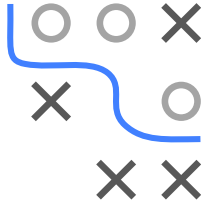




Many people are confused what these terms actually mean.

And what does all this have to do with statistics?

ARTIFICIAL INTELLIGENCE



- AI is a general term for a very large and rapidly developing field.
- There is no strict definition of AI, but it's often used when machines are trained to perform on tasks which until that time could only be solved by humans or are very difficult and assumed to require "intelligence".
- AI started in the 1940s - when the computer was invented. Scientists like Turing and John von Neumann immediately asked the question: If we can formalize computation, can we use computation to formalize "thinking"?
- AI includes machine learning, natural language processing, computer vision, robotics, planning, search, game playing, intelligent agents, and much more.
- Nowadays, AI is a "hype" term that many people use when they should probably say: ML or ... basic data analysis.

MACHINE LEARNING

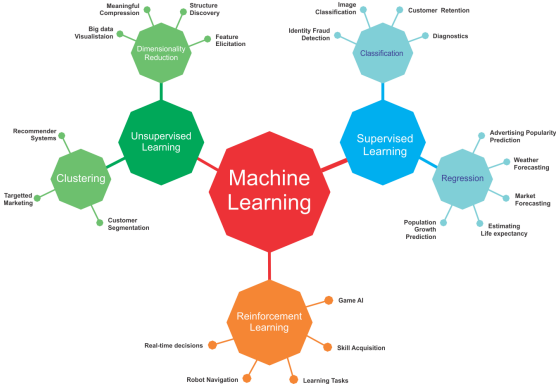
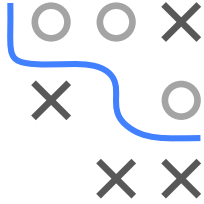


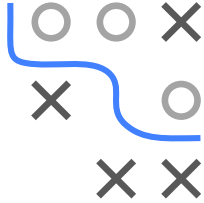
Image via `https://www.oreilly.com/library/view/java-deep-learning/9781788997454/assets/899ceaf3-c710-4675-ae99-33c76cd6ac2f.png`



- Mathematically well-defined and solves reasonably narrow tasks.
- ML algorithms usually construct predictive/decision models from data, instead of explicitly programming them.
- A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.
  Tom Mitchell, Carnegie Mellon University, 1998

DEEP LEARNING

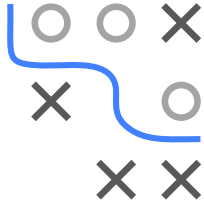
- DL is a subfield of ML which studies neural networks.
- Artificial neural networks (ANNs) might have been (roughly) inspired by the human brain, but they are simply a certain model class of ML.
- ANNs have been studied for decades. DL uses more layers, specific neurons were invented for images and tensors and many computational improvements allow training on large data.
- DL can be used on tabular data, but typical applications are images, texts or signals.
- The last 10-15 years have produced remarkable results and imitations of human ability, where the result looked intelligent.



"Any sufficiently advanced technology is indistinguishable from magic."
Arthur C. Clarke's 3rd law

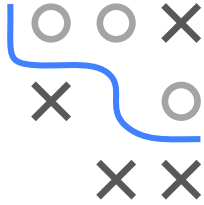
ML VS. STATS

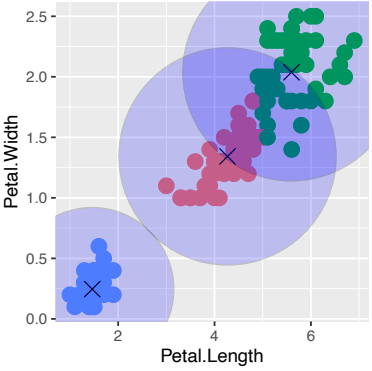
- ML and Statistics have historically been developed in different fields, but many methods and especially the mathematical foundations are equivalent.
- Traditionally, models from ML focused more on precise predictions whereas models from statistics focused more on the ability to interpret the patterns that generated the data and the ability to derive sound inference.
- Nowadays, ML and predictive modelling in statistics basically work on the same problems with the same tools.
- Unfortunately, the communities are still divided, don't talk to each other as much as they should and everyone is confused due to different terminology for the same concepts.
- Most parts of ML we could also call:
  Nonparametric statistics plus efficient numerical optimization.



UNSUPERVISED LEARNING

- Data without labels y
- Search for patterns within the inputs x
- *Unsupervised* as there is no “true” output we can optimize against

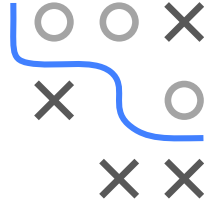


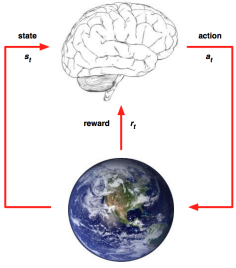


- Dimensionality reduction (PCA, Autoencoders ...); compress information in $\mathcal{X}$
- Clustering: group similar observations
- Outlier detection, anomaly detection
- Association rules

REINFORCEMENT LEARNING

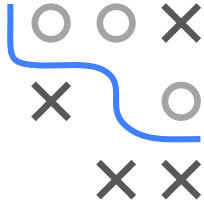
RL is a general-purpose framework for AI. At each time step an *agent* interacts with *environment*. It: observes state; receives reward; executes action.





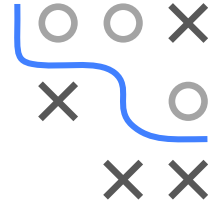
- Goal: Select actions to maximize future reward.
- Reward signals may be sparse, noisy and delayed.

WHAT COMES NEXT

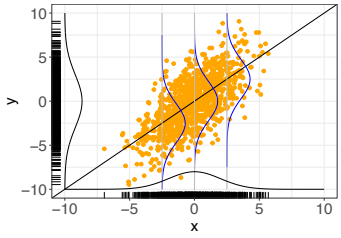


- We will deal with **supervised learning** for regression and classification: predicting labels y based on features x, using patterns that we learned from labeled data.
- First, we will go through fundamental concepts in supervised ML:
  - What kind of "data" do we learn from?
  - How can we formalize the goal of learning?
  - What is a "prediction model"?
  - How can we quantify "predictive performance"?
  - What is a "learning algorithm"
  - How can we operationalize learning?
- We will also look at a couple of fairly simple ML models to obtain a basic understanding.
- More complex stuff comes later.

Introduction to Machine Learning



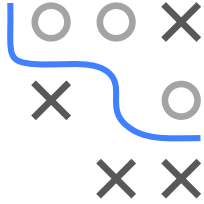
ML-Basics
Data



Learning goals

- Understand structure of tabular data in ML
- Understand difference between target and features
- Understand difference between labeled and unlabeled data
- Know concept of data-generating process

IRIS DATA SET



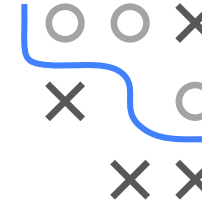
Introduced by the statistician Ronald Fisher and one of the most frequently used toy examples.

- Classify iris subspecies based on flower measurements.
- 150 iris flowers: 50 versicolor, 50 virginica, 50 setosa.
- Sepal length / width and petal length / width in [cm].

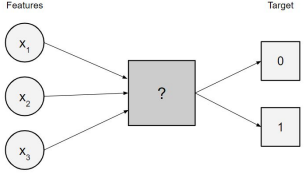
Source: `https://rpubs.com/vidhividhi/irisdataeda`

Word of warning: "iris" is a small, clean, low-dimensional data set, which is very easy to classify; this is not necessarily true in the wild.

DATA IN SUPERVISED LEARNING

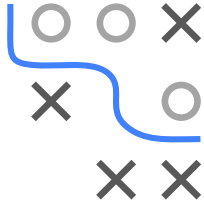


- The data we deal with in supervised learning usually consists of observations on different aspects of objects:
  - **Target**: the output variable / goal of prediction
  - **Features**: measurable properties that provide a concise description of the object
- We assume some kind of relationship between the features and the target, in a sense that the value of the target variable can be explained by a combination of the features.



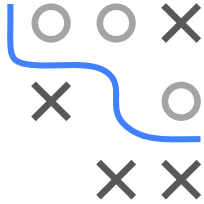
Features x				Target y
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
4.3	3.0	1.1	0.1	setosa
5.0	3.3	1.4	0.2	setosa
7.7	3.8	6.7	2.2	virginica
5.5	2.5	4.0	1.3	versicolor

ATTRIBUTE TYPES



- Both features and target variables may be of different data types
  - **Numerical** variables can have values in $\mathbb{R}$
  - **Integer** variables can have values in $\mathbb{Z}$
  - **Categorical** variables can have values in $\{C_1, ..., C_g\}$
  - **Binary** variables can have values in $\{0, 1\}$
- For the **target** variable, this results in different tasks of supervised learning: *regression* and *classification*.
- Most learning algorithms can only deal with numerical features, although there are some exceptions (e.g., decision trees can use integers and categoricals without problems). For other feature types, we usually have to pick or create an appropriate **encoding**, i.e., cast them to numerical values.
- If not stated otherwise, we assume numerical features.

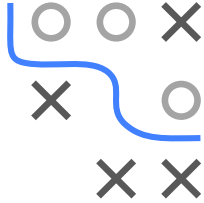
ENCODING FOR CATEGORICAL FEATURES



- We expand the representation of a feature x with k mutually exclusive categories from a scalar to a length-$\tilde{k}$ vector with at most one element being 1, and 0 otherwise: $\boldsymbol{o}(x) = [\mathbb{I}(x = j)]_{j=1,2,\ldots,\tilde{k}} \in \{0, 1\}^{\tilde{k}}$.
- Each entry of $\boldsymbol{o}(x)$ is treated as a separate feature.
- Two popular ways to do this are
  - **One-hot encoding**: $\tilde{k} = k$ dummies, so *exactly one* element is 1 ("hot"). E.g., $x \in \{a, b, c\} \mapsto \boldsymbol{o}(x) = (x_a, x_b, x_c)$, with $x_a = x_b = 0$, $x_c = 1$ and $\boldsymbol{o}(x) = (0, 0, 1)$ for $x = c$.
  - **Dummy encoding**: $\tilde{k} = k - 1$ dummies, so *at most one* element is 1, cutting the redundancy of one-hot encoding (necessary for learners that require non-singular input matrices, such as in linear regression). E.g., $x \in \{a, b, c\} \mapsto \boldsymbol{o}(x) = (x_a, x_b)$ for reference category c, with $x_a = x_b = 0$ and $\boldsymbol{o}(x) = (0, 0)$ for $x = c$.
- For features with a natural **order** in their categories we resort to encodings that reflect this ordinality, e.g., a sequence of integer values.

OBSERVATION LABELS

- We call the entries of the target column **labels**.
- We distinguish two basic forms our data may come in:
  - For **labeled** data we have already observed the target
  - For **unlabeled** data the target labels are unknown

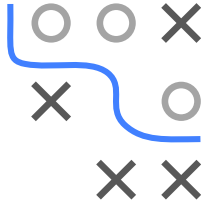


	Features x				Target y
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
labeled data	4.3	3.0	1.1	0.1	setosa
	5.0	3.3	1.4	0.2	setosa
	7.7	3.8	6.7	2.2	virginica
	5.5	2.5	4.0	1.3	versicolor
unlabeled data	5.9	3.0	5.1	1.8	?
	4.4	3.2	1.3	0.2	?

NOTATION FOR DATA

In formal notation, the data sets we are given are of the following form:

$$\mathcal{D} = \left(\left(\mathbf{x}^{(1)}, y^{(1)} \right), \ldots, \left(\mathbf{x}^{(n)}, y^{(n)} \right) \right) \in (\mathcal{X} \times \mathcal{Y})^n.$$



We call

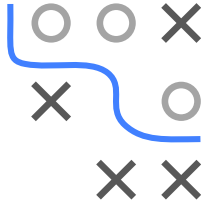
- $\mathcal{X}$ the input space with $p = \dim(\mathcal{X})$ (for now: $\mathcal{X} \subset \mathbb{R}^p$),
- $\mathcal{Y}$ the output / target space,
- the tuple $\left(\mathbf{x}^{(i)}, y^{(i)} \right) \in \mathcal{X} \times \mathcal{Y}$ the i-th observation,
- $\mathbf{x}_j = \left(x_j^{(1)}, \ldots, x_j^{(n)} \right)^\top$ the j-th feature vector.

We denote

- $(\mathcal{X} \times \mathcal{Y})^n$, i.e., the set of all data sets of size n, as $\mathbb{D}_n$,
- $\bigcup_{n \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^n$, i.e., the set of all finite data sets, as $\mathbb{D}$.

So we have observed n objects, described by p features.

DATA-GENERATING PROCESS



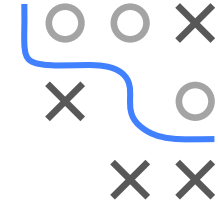
- We assume the observed data $\mathcal{D}$ to be generated by a process that can be characterized by some probability distribution

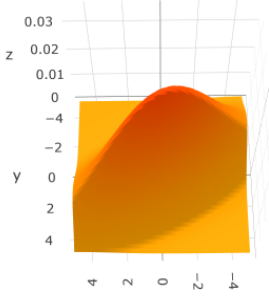
$$\mathbb{P}_{xy},$$

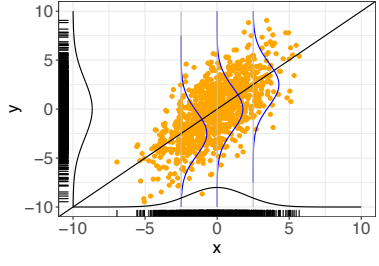
  defined on $\mathcal{X} \times \mathcal{Y}$.
- We denote the random variables following this distribution by lowercase **x** and y.
- It is important to understand that the true distribution is essentially **unknown** to us. In a certain sense, learning (part of) its structure is what ML is all about.

DATA-GENERATING PROCESS

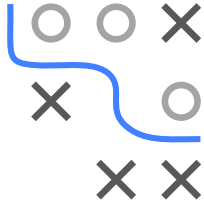
- We assume data to be drawn *i.i.d.* from the joint probability density function (pdf) / probability mass function (pmf) $p(\mathbf{x}, y)$.
  - i.i.d. stands for **i**ndependent and **i**dentically **d**istributed.
  - This means: We assume that all samples are drawn from the same distribution and are mutually independent – the i-th realization does not depend on the other $n - 1$ ones.
  - This is a strong yet crucial assumption that is precondition to most theory in (basic) ML.







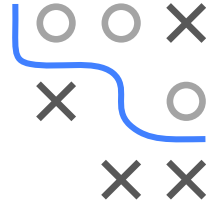
DATA-GENERATING PROCESS



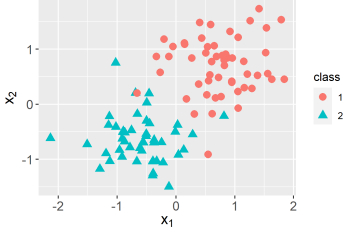
Remarks:

- With a slight abuse of notation we write random variables, e.g., $\mathbf{x}$ and y, in lowercase, as normal variables or function arguments. The context will make clear what is meant.
- Often, distributions are characterized by a parameter vector $\boldsymbol{\theta} \in \Theta$. We then write $p(\mathbf{x}, y \mid \boldsymbol{\theta})$.
- This lecture mostly takes a frequentist perspective. Distribution parameters $\boldsymbol{\theta}$ appear behind the $\mid$ for improved legibility, not to imply that we condition on them in a probabilistic Bayesian sense. So, strictly speaking, $p(\mathbf{x}|\boldsymbol{\theta})$ should usually be understood to mean $p_{\boldsymbol{\theta}}(\mathbf{x})$ or $p(\mathbf{x}, \boldsymbol{\theta})$ or $p(\mathbf{x}; \boldsymbol{\theta})$. On the other hand, this notation makes it very easy to switch to a Bayesian view.

Introduction to Machine Learning



ML-Basics
Supervised Tasks

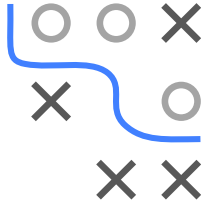


Learning goals

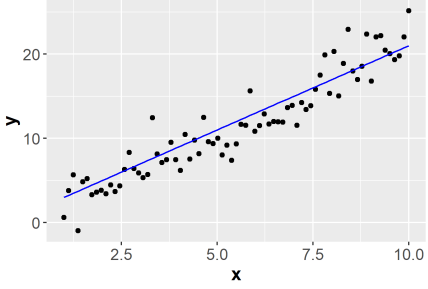
- Know definition and examples of supervised tasks
- Understand the difference between regression and classification

TASKS: REGRESSION VS CLASSIFICATION

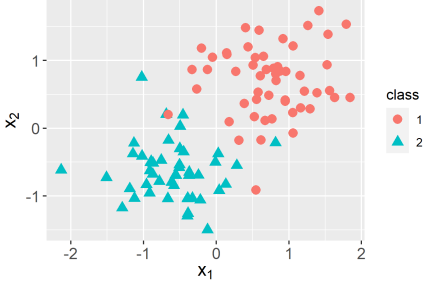
- Supervised tasks are data situations where learning the functional relationship between inputs (features) and output (target) is useful.
- The two most basic tasks are regression and classification, depending on whether the target is numerical or categorical.



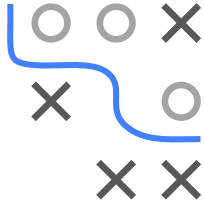
Regression: Our observed labels come from $\mathcal{Y} \subseteq \mathbb{R}$.



Classification: Observations are categorized: $y \in \mathcal{Y} = \{C_1, ..., C_g\}$.



PREDICT VS. EXPLAIN



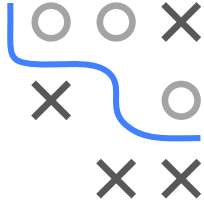
We can distinguish two main reasons to learn this relationship:

- **Learning to predict**. In such a case we potentially do not care how our model is structured or whether we can understand it.
  Example: predicting how a stock price will develop.
  Simply being able to use the predictor on new data is of direct benefit to us.
- **Learning to explain**. Here, our model is only a means to a better understanding of the inherent relationship in the data.
  Example: understanding which risk factors influence the probability to get a certain disease. We might not use the learned model on new observations, but rather discuss its implications, in a scientific or social context.

While ML was traditionally more interested in the former, classical statistics addressed the latter. In many tasks nowadays both are relevant – to different degrees.

REGRESSION EXAMPLE: HOUSE PRICES

Predict the price for a house in a certain area

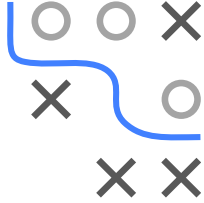


Features x				Target y
square footage of the house	number of bedrooms	swimming pool (yes/no)	...	house price in US$
1,180	3	0	...	221,900
2,570	3	1	...	538,000
770	2	0	...	180,000
1,960	4	1	...	604,000

Probably *learn to explain*. We might want to understand what influences a house price most. But maybe we are also looking for underpriced houses and the predictor is of direct use, too.

REGRESSION EXAMPLE: LENGTH-OF-STAY

Predict days a patient has to stay in hospital at time of admission

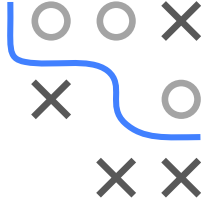


Features x					Target y
diagnosis category	admission type	gender	age	...	Length-of-stay in the hospital in days
heart disease	elective	male	75	...	4.6
injury	emergency	male	22	...	2.6
psychosis	newborn	female	0	...	8
pneumonia	urgent	female	67	...	5.5

Can be *learn to explain*, but *learn to predict* would help a hospital's planning immensely.

CLASSIFICATION EXAMPLE: RISK CATEGORY

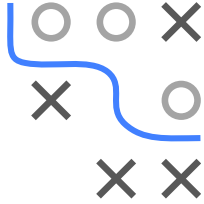
Predict one of five risk categories for a life insurance customer to determine the insurance premium



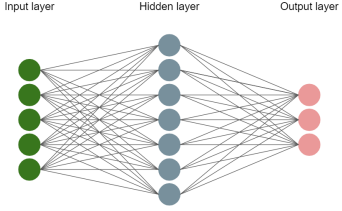
Features x				Target y
job type	age	smoker	...	risk group
carpenter	34	1	...	3
stuntman	25	0	...	5
student	23	0	...	1
white-collar worker	39	0	...	2

Probably *learn to predict*, but the company might be required to explain its predictions to its customers.

Introduction to Machine Learning



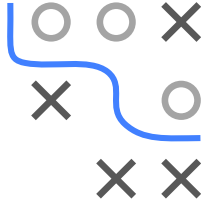
ML-Basics
Models & Parameters



Learning goals

- Understand that an ML model is simply a parametrized curve
- Understand that the hypothesis space lists all admissible models for a learner
- Understand the relationship between the hypothesis space and the parameter space

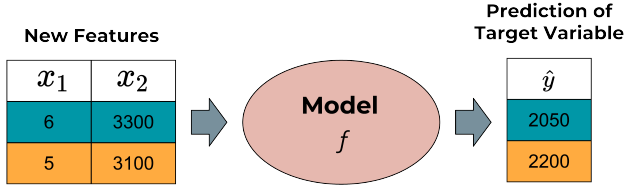
WHAT IS A MODEL?



- A **model** (or **hypothesis**)

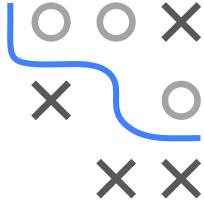
$$f : \mathcal{X} \to \mathbb{R}^g$$

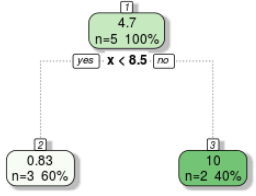
  is a function that maps feature vectors to predicted target values.
- In conventional regression: $g = 1$; for classification g is the number of classes, and output vectors are scores or class probabilities (details later).

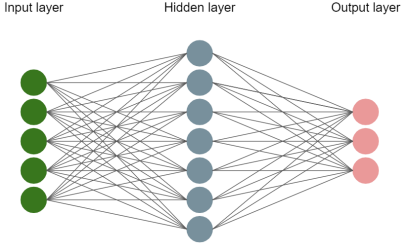


WHAT IS A MODEL?

- f is meant to capture intrinsic patterns of the data, the underlying assumption being that these hold true for *all* data drawn from $\mathbb{P}_{xy}$.
- It is easily conceivable how models can range from super simple (e.g., linear, tree stumps) to very complex (e.g., deep neural networks) and there are infinitely many choices how we can construct such functions.

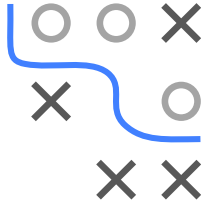






- In fact, ML requires **constraining** f to a certain type of functions.

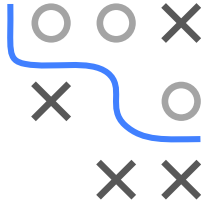
HYPOTHESIS SPACES



- Without restrictions on the functional family, the task of finding a "good" model among all the available ones is impossible to solve.
- This means: we have to determine the class of our model *a priori*, thereby narrowing down our options considerably. We could call that a **structural prior**.
- The set of functions defining a specific model class is called a **hypothesis space** $\mathcal{H}$:

$$\mathcal{H} = \{f : f \text{ belongs to a certain functional family}\}$$

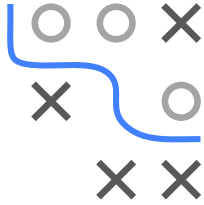
PARAMETRIZATION



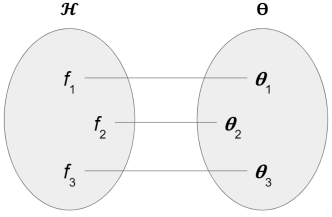
- All models within one hypothesis space share a common functional structure. We usually construct the space as **parametrized family of curves**.
- We collect all parameters in a **parameter vector** $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_d)$ from **parameter space** Θ.
- They are our means of fixing a specific function from the family. Once set, our model is fully determined.
- Therefore, we can re-write $\mathcal{H}$ as:

$$\mathcal{H} = \{f_{\boldsymbol{\theta}} : \quad f_{\boldsymbol{\theta}} \text{ belongs to a certain functional family parameterized by } \boldsymbol{\theta}\}$$

PARAMETRIZATION



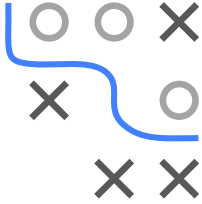
- This means: finding the optimal model is perfectly equivalent to finding the optimal set of parameter values.
- The relation between optimization over $f \in \mathcal{H}$ and optimization over $\boldsymbol{\theta} \in \Theta$ allows us to operationalize our search for the best model via the search for the optimal value on a d-dimensional parameter surface.

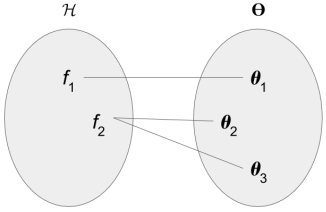


- $\boldsymbol{\theta}$ might be scalar or comprise thousands of parameters, depending on the complexity of our model.

PARAMETRIZATION

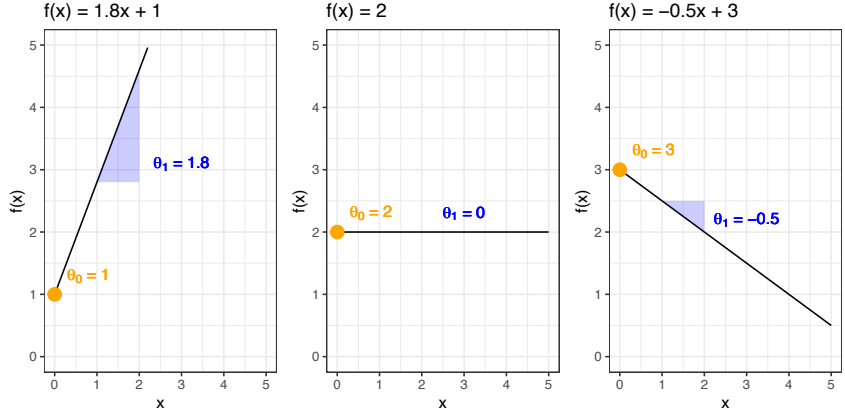
- Short remark: In fact, some parameter vectors, for some model classes, might encode the same function. So the parameter-to-model mapping could be non-injective.
- We call this then a non-identifiable model.
- But this shall not concern us here.





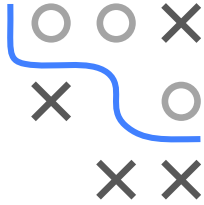
EXAMPLE: UNIVARIATE LINEAR FUNCTIONS

$$\mathcal{H} = \{f : f(\mathbf{x}) = \theta_0 + \theta_1 x, \boldsymbol{\theta} \in \mathbb{R}^2\}$$

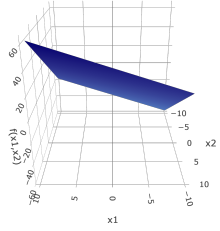


EXAMPLE: BIVARIATE QUADRATIC FUNCTIONS

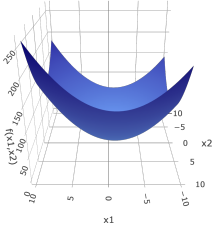
$$\mathcal{H} = \{f : f(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2, \boldsymbol{\theta} \in \mathbb{R}^6\},$$



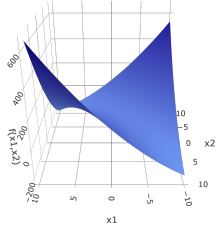
$f(x) = 3 + 2x_1 + 4x_2$



$f(x) = 3 + 2x_1 + 4x_2 +$
$+ 1x_1^2 + 1x_2^2$

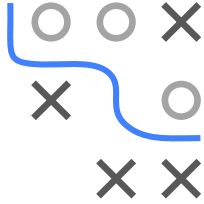


$f(x) = 3 + 2x_1 + 4x_2 +$
$+ 1x_1^2 + 1x_2^2 + 4x_1 x_2$



EXAMPLE: RBF NETWORK

Radial basis function networks with Gaussian basis functions

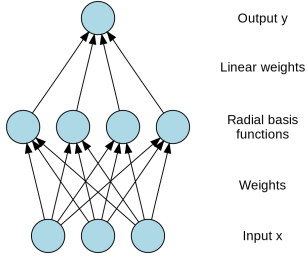


$$\mathcal{H} = \left\{ f : f(\mathbf{x}) = \sum_{i=1}^{k} a_i \rho(\|\mathbf{x} - \mathbf{c}_i\|) \right\},$$

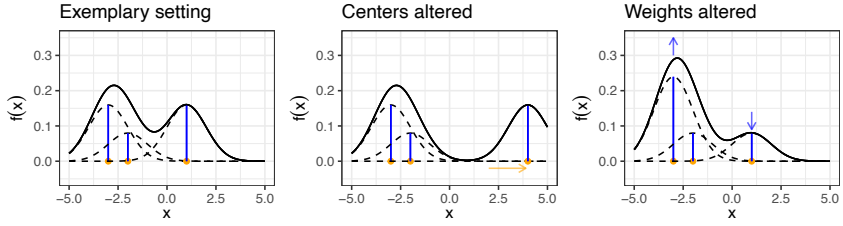
where

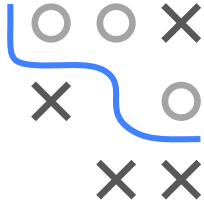
- a_i is the weight of the i-th neuron,
- $\mathbf{c}_i$ its center vector, and
- $\rho(\|\mathbf{x} - \mathbf{c}_i\|) = \exp(-\beta\|\mathbf{x} - \mathbf{c}_i\|^2)$ is the i-th radial basis function with bandwidth $\beta \in \mathbb{R}$.

Usually, the number of centers k and the bandwidth β need to be set in advance (so-called *hyperparameters*).



EXAMPLE: RBF NETWORK

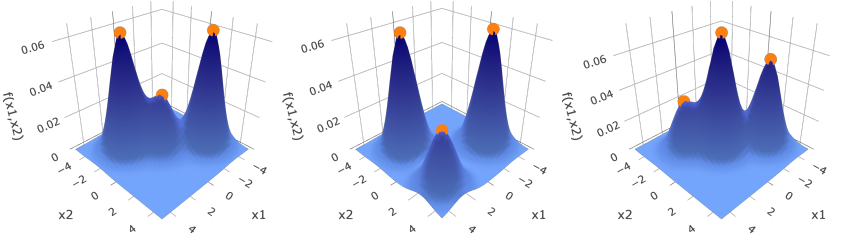




$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4$
$c_1 = -3, c_2 = -2, c_3 = 1$

$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4$
$c_1 = -3, c_2 = -2, c_3 = 4$

$a_1 = 0.6, a_2 = 0.2, a_3 = 0.2$
$c_1 = -3, c_2 = -2, c_3 = 1$

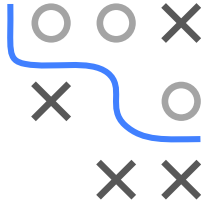


$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4$
$c_1 = (2, -2), c_2 = (0, 0),$
$c_3 = (-3, 2)$

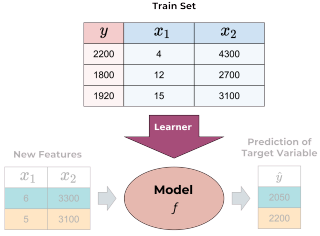
$a_1 = 0.4, a_2 = 0.2, a_3 = 0.4$
$c_1 = (2, -2), c_2 = (3, 3),$
$c_3 = (-3, 2)$

$a_1 = 0.2, a_2 = 0.45, a_3 = 0.35$
$c_1 = (2, -2), c_2 = (0, 0),$
$c_3 = (-3, 2)$

Introduction to Machine Learning



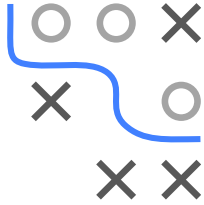
ML-Basics
Learner



Learning goals

- Understand that a supervised learner fits models automatically from training data
-

SUPERVISED LEARNING EXAMPLE



Imagine we want to investigate how working conditions affect productivity of employees.

- It is a **regression** task since the target *productivity* is continuous.
- We collect data about worked minutes per week (*productivity*), how many people work in the same office as the employee in question, and the employee's salary.

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

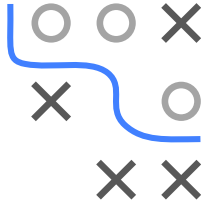
$n = 3$

$p = 2$

$x_1^{(2)}$ $x_2^{(1)}$ $y^{(3)}$

SUPERVISED LEARNING EXAMPLE

How could we construct a model from these data?



We could investigate the data manually and come up with a simple, hand-crafted rule such as:

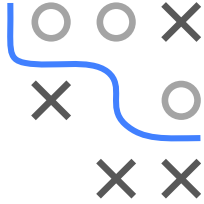
- The baseline productivity of an employee with salary 3000 and 7 people in the office is 1850 minutes
- A decrease of 1 person in the office increases productivity by 30
- An increase of the salary by 100 increases productivity by 10

=> Obviously, this is neither feasible nor leads to a good model

IDEA OF SUPERVISED LEARNING

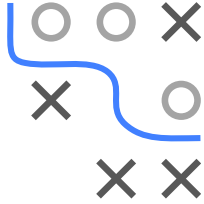
Goal: Automatically identify the fundamental functional relation in the data that maps an object's features to the target.

- **Supervised** learning means we make use of *labeled* data for which we observed the outcome.
- We use the labeled data to learn a model f.
- Ultimately, we use our model to compute predictions for **new** data whose target values are unknown.



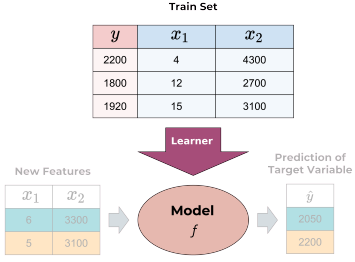
	x_1	x_2	Functional Relationship	y
Already seen Data	4	4300 €		2200
	12	2700 €		1800
	15	3100 €	f	1920
New Data	6	3300 €		???
	5	3100 €		???

LEARNER DEFINITION



- The algorithm for finding our *f* is called **learner**. It is also called **learning algorithm** or **inducer**.
- We prescribe a certain hypothesis space, the learner is our means of picking the best element from that space for our data set.
- Formally, it maps training data $\mathcal{D} \in \mathbb{D}$ (plus a vector of **hyperparameter** control settings $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$) to a model:

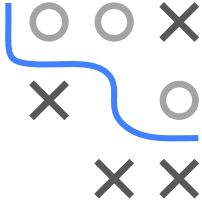
$$\mathcal{I} : \mathbb{D} \times \boldsymbol{\Lambda} \rightarrow \mathcal{H}$$

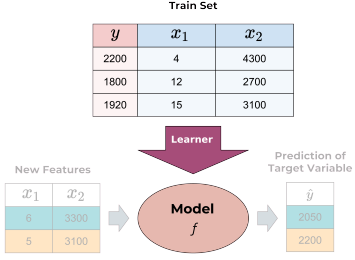


LEARNER DEFINITION

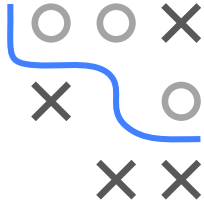
As pseudo-code template it would work like this:

- Learner has a defined model space of parametrized functions $\mathcal{H}$.
- User passes data set $\mathcal{D}_{\text{train}}$ and control settings $\boldsymbol{\lambda}$.
- Learner sets parameters so that model matches data best.
- Optimal parameters $\hat{\boldsymbol{\theta}}$ or function $\hat{f}$ is returned for later usage.

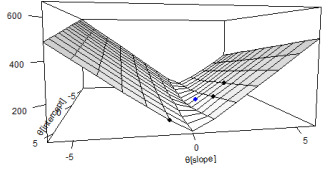




Introduction to Machine Learning



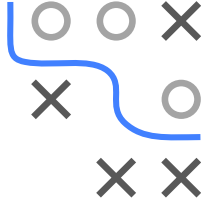
ML-Basics
Losses & Risk Minimization



Learning goals

- Know the concept of loss
- Understand the relationship between loss and risk
- Understand the relationship between risk minimization and finding the best model

HOW TO EVALUATE MODELS



- When training a learner, we optimize over our hypothesis space, to find the function which matches our training data best.
- This means, we are looking for a function, where the predicted output per training point is as close as possible to the observed label.

Features x		Target y		Prediction $\hat{y}$
People in Office (Feature 1) x_1	Salary (Feature 2) x_2	Worked Minutes Week (Target Variable)	$\overset{?}{\approx}$	Worked Minutes Week (Target Variable)
4	4300 €	2220		2588
12	2700 €	1800		1644
5	3100 €	1920		1870

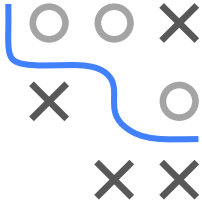
$\mathcal{D}_{\text{train}}$

- To make this precise, we need to define now how we measure the difference between a prediction and a ground truth label pointwise.

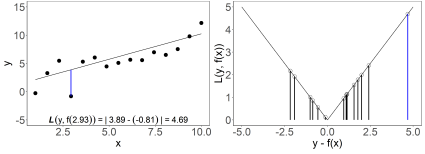
LOSS

The **loss function** $L(y, f(\mathbf{x}))$ quantifies the "quality" of the prediction $f(\mathbf{x})$ of a single observation $\mathbf{x}$:

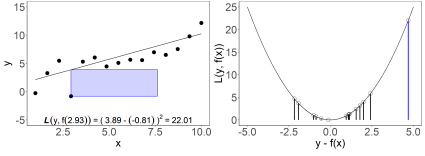
$$L : \mathcal{Y} \times \mathbb{R}^g \to \mathbb{R}.$$



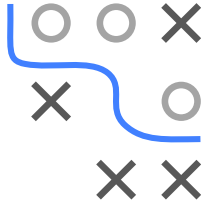
In regression, we could use the absolute loss $L(y, f(\mathbf{x})) = |f(\mathbf{x}) - y|$;



or the L2-loss $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$:



RISK OF A MODEL



- The (theoretical) **risk** associated with a certain hypothesis $f(\mathbf{x})$ measured by a loss function $L(y, f(\mathbf{x}))$ is the **expected loss**

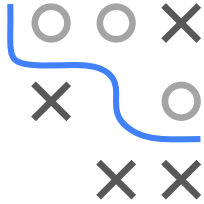
$$\mathcal{R}(f) := \mathbb{E}_{xy}[L(y, f(\mathbf{x}))] = \int L(y, f(\mathbf{x}))\, \mathrm{d}\mathbb{P}_{xy}.$$

- This is the average error we incur when we use f on data from $\mathbb{P}_{xy}$.
- Goal in ML: Find a hypothesis $f(\mathbf{x}) \in \mathcal{H}$ that **minimizes** risk.

RISK OF A MODEL

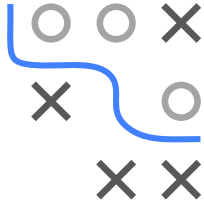
Problem: Minimizing $\mathcal{R}(f)$ over f is not feasible:

- $\mathbb{P}_{xy}$ is unknown (otherwise we could use it to construct optimal predictions).
- We could estimate $\mathbb{P}_{xy}$ in non-parametric fashion from the data $\mathcal{D}$, 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.



But as we have n i.i.d. data points from $\mathbb{P}_{xy}$ available we can simply approximate the expected risk by computing it on $\mathcal{D}$.

EMPIRICAL RISK

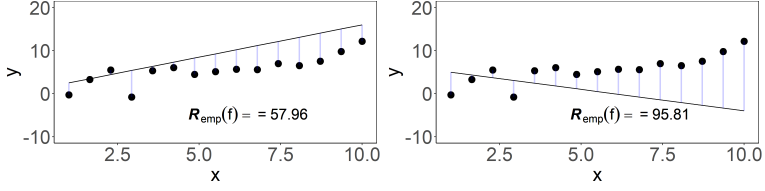


To evaluate, how well a given function f matches our training data, we now simply sum-up all f's pointwise losses.

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

This gives rise to the **empirical risk function** which allows us to associate one quality score with each of our models, which encodes how well our model fits our training data.

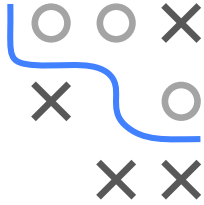
$$\mathcal{R}_{\text{emp}} : \mathcal{H} \to \mathbb{R}$$



EMPIRICAL RISK

- The risk can also be defined as an average loss

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



The factor $\frac{1}{n}$ does not make a difference in optimization, so we will consider $\mathcal{R}_{\text{emp}}(f)$ most of the time.

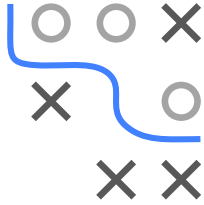
- Since f is usually defined by **parameters** $\boldsymbol{\theta}$, this becomes:

$$\mathcal{R} : \mathbb{R}^d \to \mathbb{R}$$

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

EMPIRICAL RISK MINIMIZATION

The best model is the model with the smallest risk.

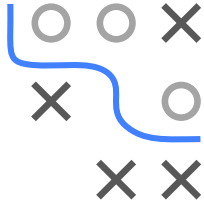


If we have a finite number of models f, we could simply tabulate them and select the best.

Model	$\theta_{intercept}$	θ_{slope}	$\mathcal{R}_{\text{emp}}(\theta)$
f_1	2	3	194.62
f_2	3	2	127.12
f_3	6	-1	95.81
f_4	1	1.5	57.96

EMPIRICAL RISK MINIMIZATION

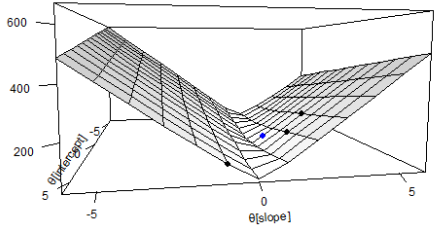
But usually $\mathcal{H}$ is infinitely large.



Instead we can consider the risk surface w.r.t. the parameters $\boldsymbol{\theta}$.
(By this I simply mean the visualization of $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$)

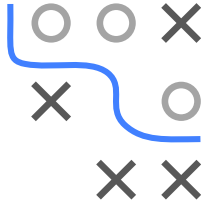
$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) : \mathbb{R}^d \to \mathbb{R}.$$

Model	$\boldsymbol{\theta}_{intercept}$	$\boldsymbol{\theta}_{slope}$	$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$
f_1	2	3	194.62
f_2	3	2	127.12
f_3	6	-1	95.81
f_4	1	1.5	57.96



EMPIRICAL RISK MINIMIZATION

Minimizing this surface is called **empirical risk minimization** (ERM).

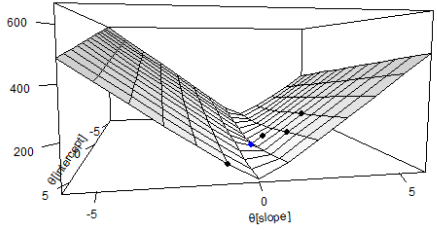


$$\hat{\theta} = \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}).$$

Usually we do this by numerical optimization.

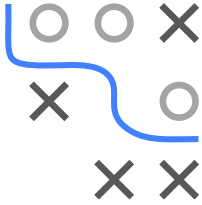
$$\mathcal{R} : \mathbb{R}^d \to \mathbb{R}.$$

Model	$\boldsymbol{\theta}_{intercept}$	$\boldsymbol{\theta}_{slope}$	$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$
f_1	2	3	194.62
f_2	3	2	127.12
f_3	6	-1	95.81
f_4	1	1.5	57.96
f_5	1.25	0.90	23.40

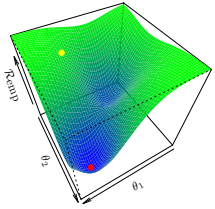


In a certain sense, we have now reduced the problem of learning to **numerical parameter optimization**.

Introduction to Machine Learning



ML-Basics
Optimization

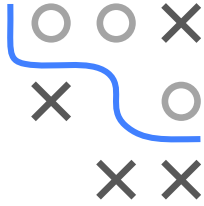


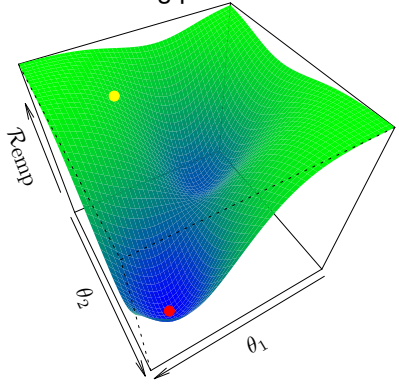
Learning goals

- Understand how the risk function is optimized to learn the optimal parameters of a model
- Understand the idea of gradient descent as a basic risk optimizer

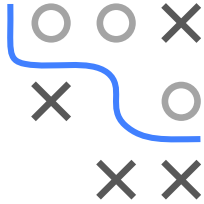
LEARNING AS PARAMETER OPTIMIZATION

- We have seen, we can operationalize the search for a model f that matches training data best, by looking for its parametrization $\boldsymbol{\theta} \in \Theta$ with lowest empirical risk $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$.
- Therefore, we usually traverse the error surface downwards; often by local search from a starting point to its minimum.





LEARNING AS PARAMETER OPTIMIZATION



The ERM optimization problem is:

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

For a **(global) minimum** $\hat{\boldsymbol{\theta}}$ it obviously holds that

$$\forall \boldsymbol{\theta} \in \Theta: \quad \mathcal{R}_{\text{emp}}(\hat{\boldsymbol{\theta}}) \leq \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}).$$

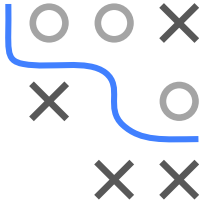
This does not imply that $\hat{\boldsymbol{\theta}}$ is unique.

Which kind of numerical technique is reasonable for this problem strongly depends on model and parameter structure (continuous params? uni-modal $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$?). Here, we will only discuss very simple scenarios.

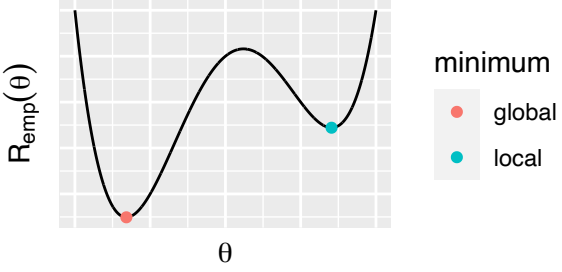
LOCAL MINIMA

If $\mathcal{R}_{\text{emp}}$ is continuous in $\boldsymbol{\theta}$ we can define a **local minimum** $\hat{\boldsymbol{\theta}}$:

$$\exists \epsilon > 0 \; \forall \boldsymbol{\theta} \text{ with } \left\| \hat{\boldsymbol{\theta}} - \boldsymbol{\theta} \right\| < \epsilon : \quad \mathcal{R}_{\text{emp}}(\hat{\boldsymbol{\theta}}) \leq \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}).$$



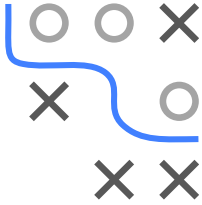
Clearly every global minimum is also a local minimum. Finding a local minimum is easier than finding a global minimum.



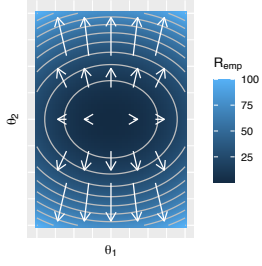
LOCAL MINIMA AND STATIONARY POINTS

If $\mathcal{R}_{\text{emp}}$ is continuously differentiable in $\boldsymbol{\theta}$ then a **sufficient condition** for a local minimum is that $\hat{\boldsymbol{\theta}}$ is **stationary** with 0 gradient, so no local improvement is possible:

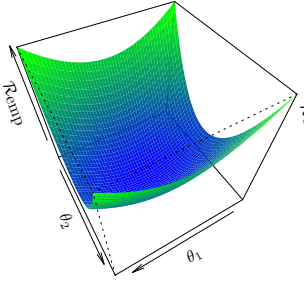
$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\hat{\boldsymbol{\theta}}) = 0$$



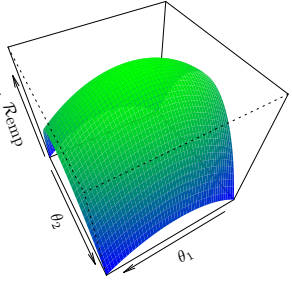
and the Hessian $\frac{\partial^2}{\partial \boldsymbol{\theta}^2} \mathcal{R}_{\text{emp}}(\hat{\boldsymbol{\theta}})$ is positive definite. While the neg. gradient points into the direction of fastest local decrease, the Hessian measures local curvature of $\mathcal{R}_{\text{emp}}$.



$\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$

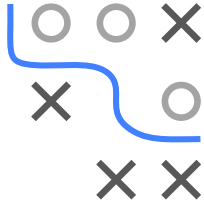


const. pos. def. Hessian



const. neg. def. Hessian

LEAST SQUARES ESTIMATOR



Now, for given features $\mathbf{X} \in \mathbb{R}^{n \times p}$ and target $\mathbf{y} \in \mathbb{R}^n$, we want to find the best linear model regarding the squared error loss, i.e.,

$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 = \sum_{i=1}^{n} (\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)})^2 \ .$$

With the sufficient condition for continously differentiable functions it can be shown that the **least squares estimator**

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

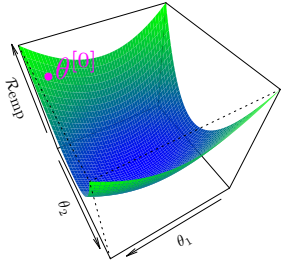
is a local minimum of $\mathcal{R}_{\text{emp}}$. If $\mathbf{X}$ is full-rank, $\mathcal{R}_{\text{emp}}$ is strictly convex and there is only one local minimum - which is also global.

Note: Often such analytical solutions in ML are not possible, and we rather have to use iterative numerical optimization.

GRADIENT DESCENT

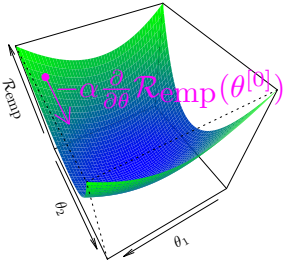
The simple idea of GD is to iteratively go from the current candidate $\boldsymbol{\theta}^{[t]}$ in the direction of the negative gradient, i.e., the direction of the steepest descent, with learning rate α to the next $\boldsymbol{\theta}^{[t+1]}$:

$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \alpha \frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[t]}).$$

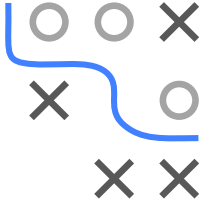


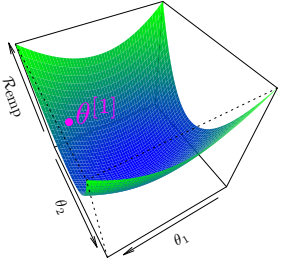
We choose a random start $\boldsymbol{\theta}^{[0]}$ with risk $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[0]}) = 76.25$.

GRADIENT DESCENT - EXAMPLE



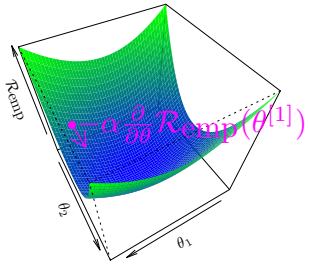
Now we follow in the direction of the negative gradient at $\boldsymbol{\theta}^{[0]}$.



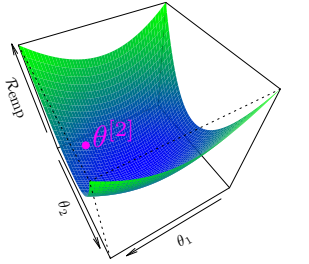


We arrive at $\boldsymbol{\theta}^{[1]}$ with risk $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[1]}) \approx 42.73$.
We improved:
$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[1]}) < \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[0]})$.

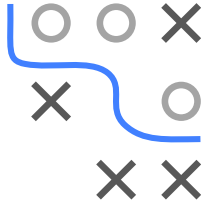
GRADIENT DESCENT - EXAMPLE



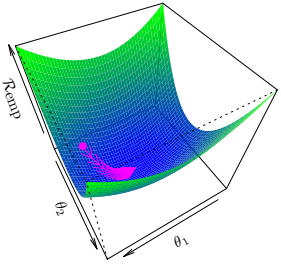
Again we follow in the direction of the negative gradient, but now at $\theta^{[1]}$.



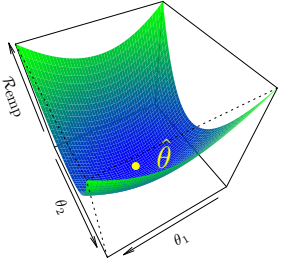
Now $\theta^{[2]}$ has risk $\mathcal{R}_{\text{emp}}(\theta^{[2]}) \approx 25.08$.



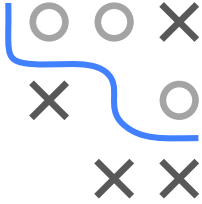
GRADIENT DESCENT - EXAMPLE



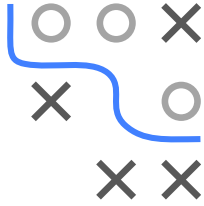
We iterate this until some form of convergence or termination.



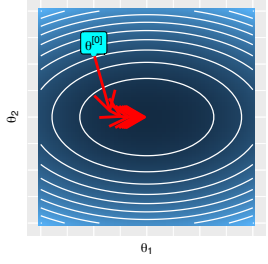
We arrive close to a stationary $\hat{\boldsymbol{\theta}}$ which is hopefully at least a local minimum.



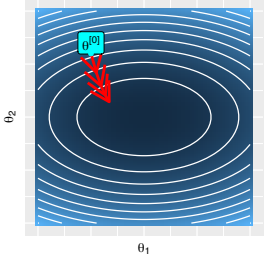
GRADIENT DESCENT - LEARNING RATE



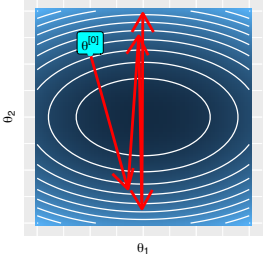
- The negative gradient is a direction that looks locally promising to reduce $\mathcal{R}_{\text{emp}}$.
- Hence it weights components higher in which $\mathcal{R}_{\text{emp}}$ decreases more.
- However, the length of $-\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}}$ measures only the local decrease rate, i.e., there are no guarantees that we will not go "too far".
- We use a learning rate α to scale the step length in each iteration. Too much can lead to overstepping and no converge, too low leads to slow convergence.
- Usually, a simple constant rate or rate-decrease mechanisms to enforce local convergence are used



good convergence for α_1



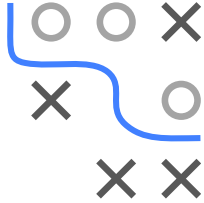
poor convergence for $\alpha_2 (< \alpha_1)$



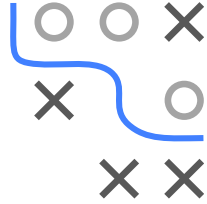
no convergence for $\alpha_3 (> \alpha_1)$

FURTHER TOPICS

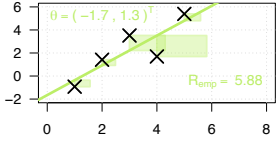
- GD is a so-called first-order method. Second-order methods use the Hessian to refine the search direction for faster convergence.
- There exist many improvements of GD, e.g., to smartly control the learn rate, to escape saddle points, to mimic second order behavior without computing the expensive Hessian.
- If the gradient of GD is not derived from the empirical risk of the whole data set, but instead from a randomly selected subset, we call this **stochastic gradient descent** (SGD). For large-scale problems this can lead to higher computational efficiency.



Introduction to Machine Learning



ML-Basics
Components of Supervised Learning

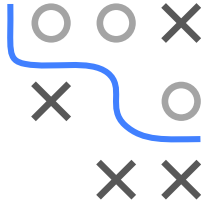


Learning goals

- Know the three components of a learner: Hypothesis space, risk, optimization
- Understand that defining these separately is the basic design of a learner
- Know a variety of choices for all three components

COMPONENTS OF SUPERVISED LEARNING

Summarizing what we have seen before, many supervised learning algorithms can be described in terms of three components:

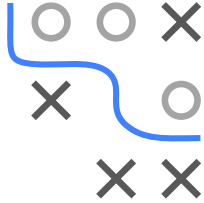


Learning = Hypothesis Space + Risk + Optimization

- **Hypothesis Space:** Defines (and restricts!) what kind of model f can be learned from the data.
- **Risk:** Quantifies how well a specific model performs on a given data set. This allows us to rank candidate models in order to choose the best one.
- **Optimization:** Defines how to search for the best model in the **hypothesis space**, i.e., the model with the smallest **risk**.

COMPONENTS OF SUPERVISED LEARNING

This concept can be extended by the concept of **regularization**, where the model complexity is accounted for in the risk:

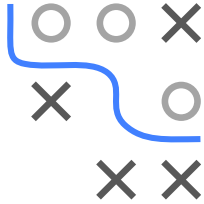


Learning = Hypothesis Space + Risk + Optim
Learning = Hypothesis Space + Loss (+ Regularization) + Optim

- For now you can just think of the risk as sum of the losses.
- While this is a useful framework for most supervised ML problems, it does not cover all special cases, because some ML methods are not defined via risk minimization and for some models, it is not possible (or very hard) to explicitly define the hypothesis space.

VARIETY OF LEARNING COMPONENTS

The framework is a good orientation to not get lost here:

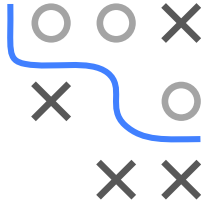


Hypothesis Space :
- Step functions
- Linear functions
- Sets of rules
- Neural networks
- Voronoi tesselations
- ...

Risk / **Loss** :
- Mean squared error
- Misclassification rate
- Negative log-likelihood
- Information gain
- ...

Optimization :
- Analytical solution
- Gradient descent
- Combinatorial optimization
- Genetic algorithms
- ...

SUPERVISED LEARNING, FORMALIZED



A **learner** (or **inducer**) $\mathcal{I}$ is a *program* or *algorithm* which

- receives a **training set** $\mathcal{D} \in \mathbb{D}$, and,
- for a given **hypothesis space** $\mathcal{H}$ of **models** $f : \mathcal{X} \to \mathbb{R}^g$,
- uses a **risk** function $\mathcal{R}_{\text{emp}}(f)$ to evaluate $f \in \mathcal{H}$ on $\mathcal{D}$;
  or we use $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$ to evaluate f's parametrization $\boldsymbol{\theta}$ on $\mathcal{D}$
- uses an **optimization** procedure to find

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \mathcal{R}_{\text{emp}}(f) \qquad \text{or} \qquad \hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}).$$

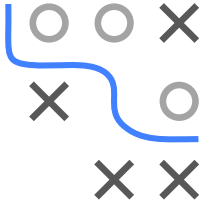
So the inducer mapping (including hyperparameters $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$) is:

$$\mathcal{I} : \mathbb{D} \times \boldsymbol{\Lambda} \to \mathcal{H}$$

We can also adapt this concept to finding $\hat{\boldsymbol{\theta}}$ for parametric models:

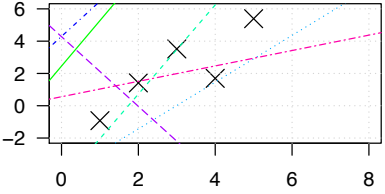
$$\mathcal{I} : \mathbb{D} \times \boldsymbol{\Lambda} \to \Theta$$

EXAMPLE: LINEAR REGRESSION ON 1D



- The **hypothesis space** in univariate linear regression is the set of all linear functions, with $\boldsymbol{\theta} = (\theta_0, \theta_1)^\top$:

$$\mathcal{H} = \{f(\mathbf{x}) = \theta_0 + \theta \mathbf{x} : \theta_0, \theta_1 \in \mathbb{R}\}$$

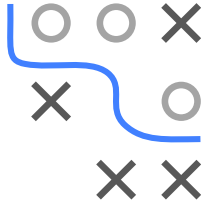


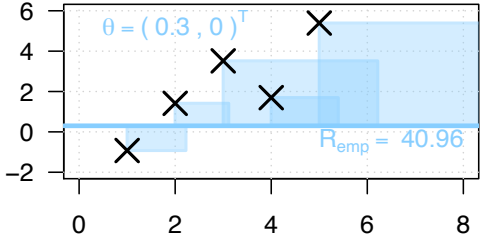
Design choice: We could add more flexibility by allowing polynomial effects or by using a spline basis.

EXAMPLE: LINEAR REGRESSION ON 1D

- We might use the squared error as loss function to our **risk**, punishing larger distances more severely:

$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} (y^{(i)} - \theta_0 - \theta_1 \mathbf{x}^{(i)})^2$$

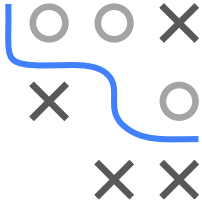


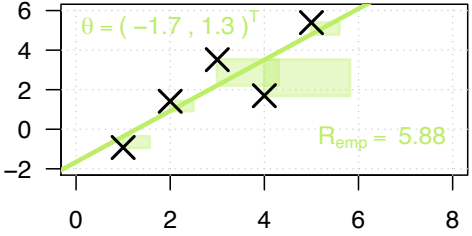


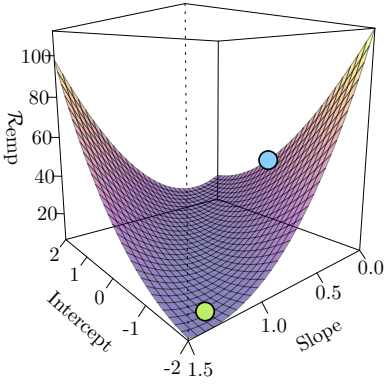
Design choice: Use absolute error / the $L1$ loss to create a more robust model which is less sensitive regarding outliers.

EXAMPLE: LINEAR REGRESSION ON 1D

- **Optimization** will usually mean deriving the ordinary-least-squares (OLS) estimator $\hat{\theta}$ analytically.



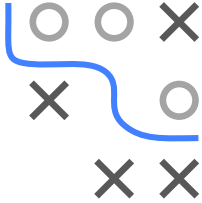




Design choice: We could use stochastic gradient descent to scale better to very large or out-of-memory data.

SUMMARY

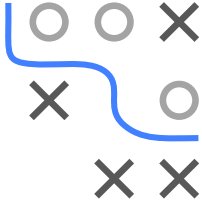
By decomposing learners into these building blocks:



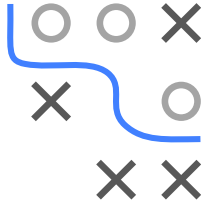
- we have a framework to better understand how they work,
- we can more easily evaluate in which settings they may be more or less suitable, and
- we can tailor learners to specific problems by clever choice of each of the three components.

Getting this right takes a considerable amount of experience.

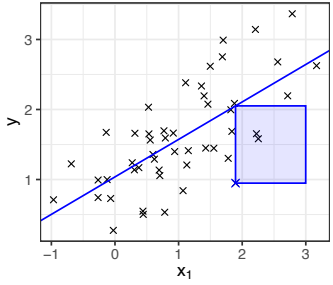
INTRODUCTION TO MACHINE LEARNING



Introduction to Machine Learning



Supervised Regression
Linear Models with *L2* Loss



Learning goals

- Grasp the overall concept of linear regression
- Understand how *L2* loss optimization results in SSE-minimal model
- Understand this as a general template for ERM in ML

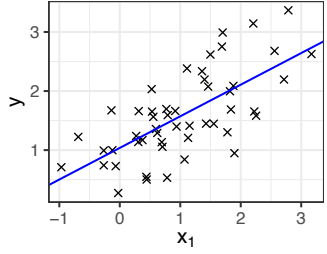
LINEAR REGRESSION

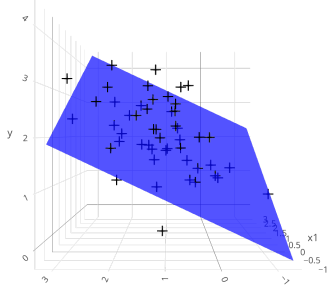
- Idea: predict $y \in \mathbb{R}$ as **linear** combination of features[1]:

$$\hat{y} = f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} = \theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p$$

  $\rightsquigarrow$ find loss-optimal params to describe relation $y|\mathbf{x}$

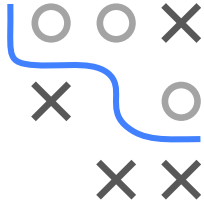
- Hypothesis space: $\mathcal{H} = \{f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} \mid \boldsymbol{\theta} \in \mathbb{R}^{p+1}\}$





[1] Actually special case of linear model, which is linear combo of *basis functions* of features $\rightsquigarrow$ Polynomial Regression Models

DESIGN MATRIX



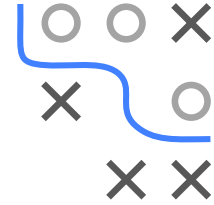
- Mismatch: $\boldsymbol{\theta} \in \mathbb{R}^{p+1}$ vs $\mathbf{x} \in \mathbb{R}^{p}$ due to intercept term
- Trick: pad feature vectors with leading 1, s.t.
  - $\mathbf{x} \mapsto \mathbf{x} = (1, x_1, \ldots, x_p)^\top$, and
  - $\boldsymbol{\theta}^\top \mathbf{x} = \theta_0 \cdot 1 + \theta_1 x_1 + \cdots + \theta_p x_p$
- Collect all observations in **design matrix X** $\in \mathbb{R}^{n \times (p+1)}$
  $\rightsquigarrow$ more compact: single param vector incl. intercept
- Resulting linear model:

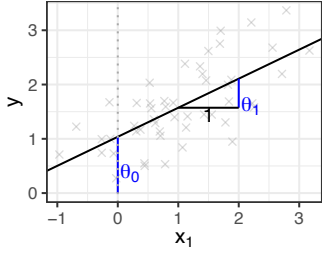
$$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\theta} = \begin{pmatrix} 1 & x_1^{(1)} & \ldots & x_p^{(1)} \\ 1 & x_1^{(2)} & \ldots & x_p^{(2)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(n)} & \ldots & x_p^{(n)} \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{pmatrix} = \begin{pmatrix} \theta_0 + \theta_1 x_1^{(1)} + \cdots + \theta_p x_p^{(1)} \\ \theta_0 + \theta_1 x_1^{(2)} + \cdots + \theta_p x_p^{(2)} \\ \vdots \\ \theta_0 + \theta_1 x_1^{(n)} + \cdots + \theta_p x_p^{(n)} \end{pmatrix}$$

- We will make use of this notation in other contexts

EFFECT INTERPRETATION

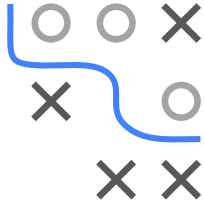
- Big plus of LM: immediately **interpretable** feature effects
- "Marginally increasing x_j by 1 unit increases y by θ_j units"
  $\rightsquigarrow$ *ceteris paribus* assumption: $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_p$ fixed





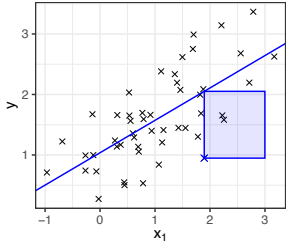

```
Call:
lm(formula = y ~ x_1, data = dt_univ)

Residuals:
     Min       1Q   Median       3Q      Max
-1.10346 -0.34727 -0.00766  0.31500  1.04284

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.03727    0.11360   9.131 4.55e-12 ***
x_1          0.53521    0.08219   6.512 4.13e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5327 on 48 degrees of freedom
Multiple R-squared:  0.469,     Adjusted R-squared:  0.458
F-statistic:  42.4 on 1 and 48 DF,  p-value: 4.129e-08
```

MODEL FIT



- How to determine LM fit? $\rightsquigarrow$ define risk & optimize
- Popular: *L2* **loss** / **quadratic loss** / **squared error**

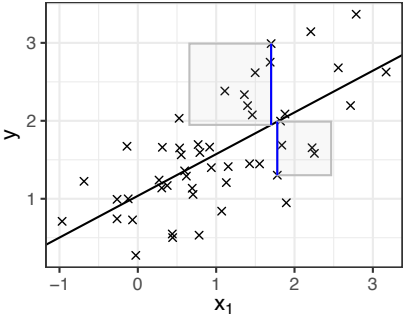
$$L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2 \quad \text{or} \quad L(y, f(\mathbf{x})) = 0.5 \cdot (y - f(\mathbf{x}))^2$$

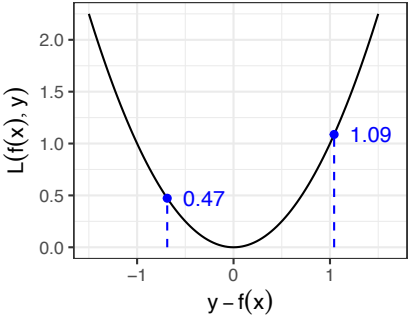


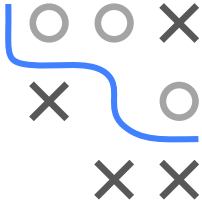
- Why penalize **residuals** $r = y - f(\mathbf{x})$ quadratically?
  - Easy to optimize (convex, differentiable)
  - Theoretically appealing (connection to classical stats LM)

LOSS PLOTS

We will often visualize loss effects like this:







- Data as $y \sim x_1$
- Prediction hypersurface
  $\rightsquigarrow$ here: line
- Residuals $r = y - f(\mathbf{x})$
  $\rightsquigarrow$ squares to illustrate loss

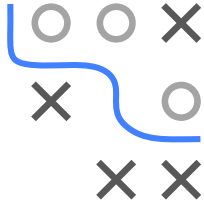
- Loss as function of residuals
  $\rightsquigarrow$ strength of penalty?
  $\rightsquigarrow$ symmetric?
- Highlighted: loss for residuals shown on LHS

OPTIMIZATION

- Resulting risk equivalent to **sum of squared errors (SSE)**:

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

- Consider example with $n = 5 \rightsquigarrow$ different models with varying SSE

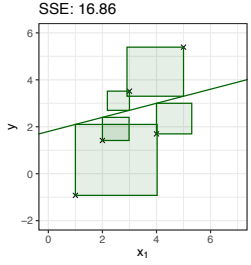


OPTIMIZATION

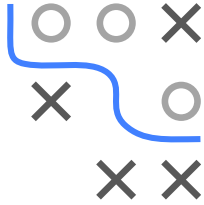
- Resulting risk equivalent to **sum of squared errors (SSE)**:

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

- Consider example with $n = 5 \rightsquigarrow$ different models with varying SSE



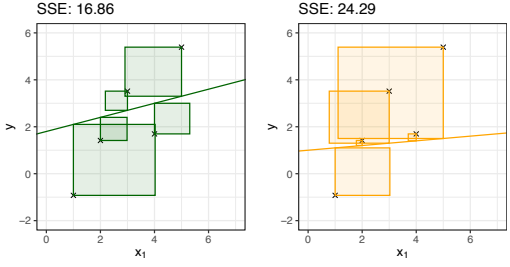
OPTIMIZATION



- Resulting risk equivalent to **sum of squared errors (SSE)**:

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

- Consider example with $n = 5 \rightsquigarrow$ different models with varying SSE

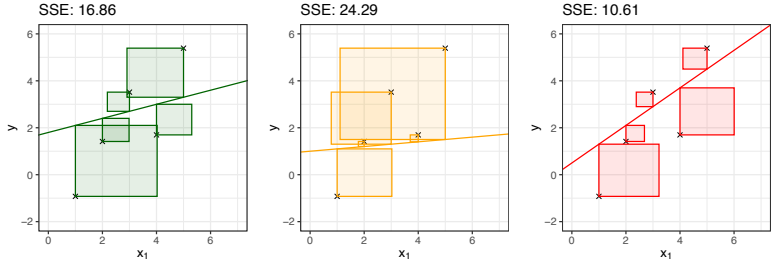


OPTIMIZATION

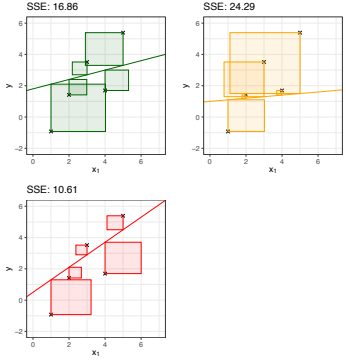
- Resulting risk equivalent to **sum of squared errors (SSE)**:

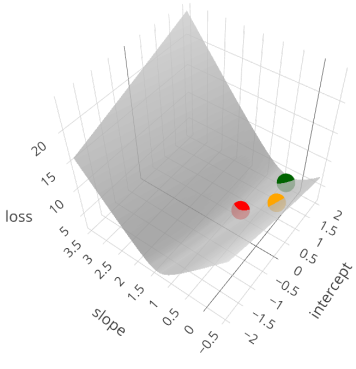
$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)} \right)^2$$

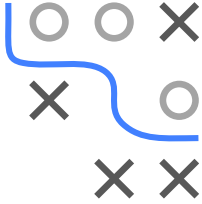
- Consider example with $n = 5 \rightsquigarrow$ different models with varying SSE



OPTIMIZATION

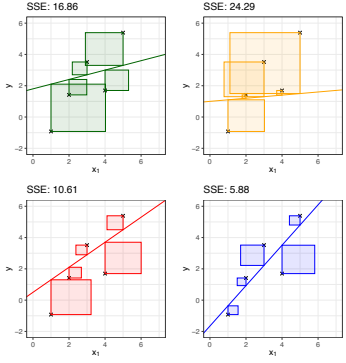


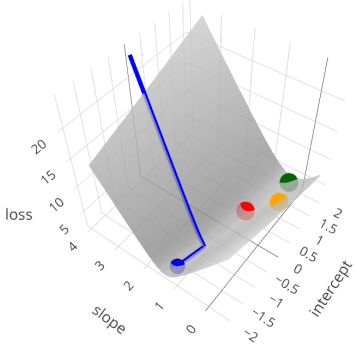


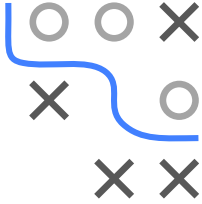


Intercept θ_0	Slope θ_1	SSE
1.80	0.30	16.86
1.00	0.10	24.29
0.50	0.80	10.61

OPTIMIZATION





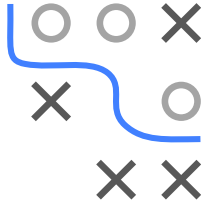


Intercept θ_0	Slope θ_1	SSE
1.80	0.30	16.86
1.00	0.10	24.29
0.50	0.80	10.61
-1.65	1.29	5.88

Instead of guessing, of course, use **optimization**!

ANALYTICAL OPTIMIZATION

- Special property of LM with *L2* loss: **analytical solution** available



$$\hat{\boldsymbol{\theta}} \in \arg\min_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)} \right)^2$$
$$= \arg\min_{\boldsymbol{\theta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2$$

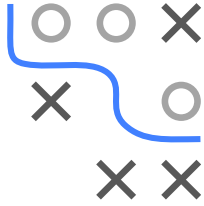
- Find via **normal equations**

$$\frac{\partial \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = 0$$

- Solution: **ordinary-least-squares (OLS)** estimator

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

STATISTICAL PROPERTIES



- LM with *L2* loss intimately related to classical stats LM
- Assumptions
  - $\mathbf{x}^{(i)}$ **iid** for $i \in \{1, \ldots, n\}$
  - **Homoskedastic** (equivariant) **Gaussian** errors

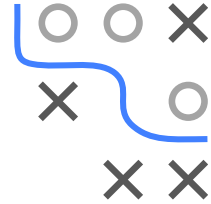
$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\epsilon}, \ \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$$

    $\rightsquigarrow$ y_i conditionally independent & normal: $\mathbf{y}|\mathbf{X} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\theta}, \sigma^2 \boldsymbol{I})$
  - Uncorrelated features

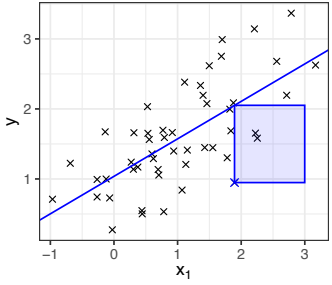
    $\rightsquigarrow$ multicollinearity destabilizes effect estimation
- If assumptions hold: statistical **inference** applicable
  - Hypothesis tests on significance of effects, incl. *p*-values
  - Confidence & prediction intervals via student-*t* distribution
  - Goodness-of-fit measure $R^2 = 1 - \text{SSE} \ / \ \underbrace{\text{SST}}_{\sum_{i=1}^{n} (y^{(i)} - \bar{y})^2}$

    $\rightsquigarrow$ SSE = part of data variance *not* explained by model

Introduction to Machine Learning



Supervised Regression
Deep Dive: Proof OLS Regression

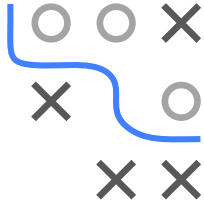


Learning goals

- Understand analytical derivation of OLS estimator for LM

ANALYTICAL OPTIMIZATION

- Special property of LM with *L2* loss: **analytical solution** available



$$\hat{\boldsymbol{\theta}} \in \arg\min_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)} \right)^2$$
$$= \arg\min_{\boldsymbol{\theta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2$$

- Find via **normal equations**

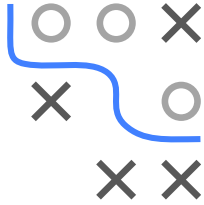
$$\frac{\partial \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = 0$$

- Solution: **ordinary-least-squares (OLS)** estimator

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

ANALYTICAL OPTIMIZATION – PROOF

$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \big(\underbrace{y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)}}_{=:\epsilon_i} \big)^2 = \| \underbrace{\mathbf{y} - \mathbf{X}\boldsymbol{\theta}}_{=:\boldsymbol{\epsilon}} \|_2^2; \quad \boldsymbol{\theta} \in \mathbb{R}^{\tilde{p}} \text{ with } \tilde{p} := p + 1$$

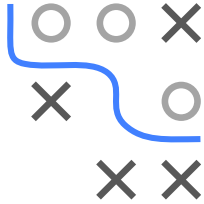


$$\begin{aligned}
0 &= \frac{\partial \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \quad \text{(sum notation)} \\
0 &= \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{i=1}^{n} \epsilon_i^2 \quad \big| \text{ sum \& chain rule} \\
0 &= \sum_{i=1}^{n} \frac{\partial \epsilon_i^2}{\partial \epsilon_i} \frac{\partial \epsilon_i}{\partial \boldsymbol{\theta}} \\
0 &= \sum_{i=1}^{n} 2\epsilon_i (-1) (\mathbf{x}^{(i)})^\top \\
0 &= \sum_{i=1}^{n} (y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)}) (\mathbf{x}^{(i)})^\top \\
\boldsymbol{\theta}^\top \sum_{i=1}^{n} \mathbf{x}^{(i)} (\mathbf{x}^{(i)})^\top &= \sum_{i=1}^{n} y^{(i)} (\mathbf{x}^{(i)})^\top \quad \big| \text{ transpose} \\
\left(\sum_{i=1}^{n} \underbrace{\mathbf{x}^{(i)} (\mathbf{x}^{(i)})^\top}_{\tilde{p} \times \tilde{p}} \right) \boldsymbol{\theta} &= \sum_{i=1}^{n} \mathbf{x}^{(i)} y^{(i)} \\
(\mathbf{X}^\top \mathbf{X}) \boldsymbol{\theta} &= \mathbf{X}^\top \mathbf{y}
\end{aligned}$$

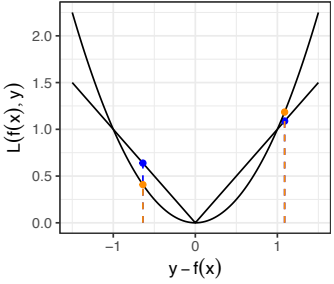
$$\begin{aligned}
0 &= \frac{\partial \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \quad \text{(matrix notation)} \\
0 &= \frac{\partial \|\boldsymbol{\epsilon}\|_2^2}{\partial \boldsymbol{\theta}} \\
0 &= \frac{\partial \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon}}{\partial \boldsymbol{\theta}} \quad \big| \text{ chain rule} \\
0 &= \frac{\partial \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon}}{\partial \boldsymbol{\epsilon}} \cdot \frac{\partial \boldsymbol{\epsilon}}{\partial \boldsymbol{\theta}} \\
0 &= 2\boldsymbol{\epsilon}^\top \cdot (-1 \cdot \mathbf{X}) \\
0 &= (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top \mathbf{X} \\
0 &= \mathbf{y}^\top \mathbf{X} - \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X} \\
\boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X} &= \mathbf{y}^\top \mathbf{X} \quad \big| \text{ transpose} \\
\mathbf{X}^\top \mathbf{X} \boldsymbol{\theta} &= \mathbf{X}^\top \mathbf{y} \\
\boldsymbol{\theta} &= \underbrace{\underbrace{(\mathbf{X}^\top \mathbf{X})^{-1}}_{\tilde{p} \times \tilde{p}} \underbrace{\mathbf{X}^\top}_{\tilde{p} \times n} \underbrace{\mathbf{y}}_{n \times 1}}_{\tilde{p} \times 1}
\end{aligned}$$

NB: $\sum_{i=1}^{n} \mathbf{x}^{(i)} (\mathbf{x}^{(i)})^\top = \mathbf{X}^\top \mathbf{X}$ is easy to show (try it!) – and good to remember (this is basically the estimation of Cov(X))

Introduction to Machine Learning



Supervised Regression
Linear Models with $L1$ Loss

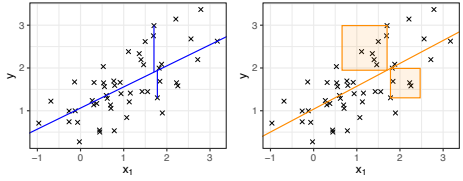


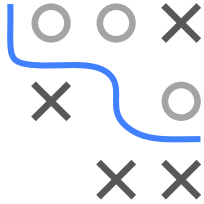
Learning goals

- Understand difference between $L1$ and $L2$ regression
- See how choice of loss affects optimization & robustness

ABSOLUTE LOSS

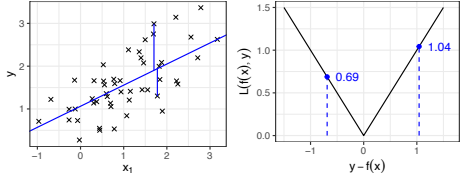
- *L*2 regression minimizes quadratic residuals – wouldn't **absolute** residuals seem more natural?



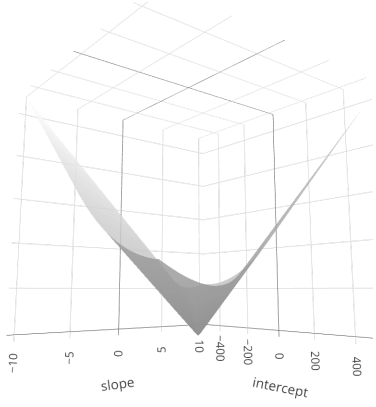


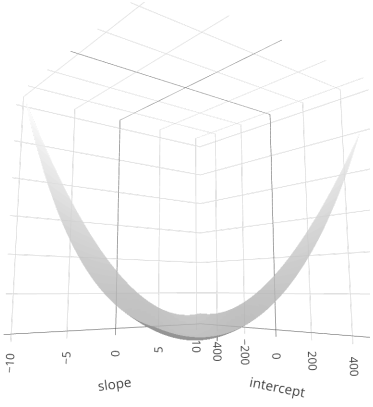
- *L*1 **loss** / **absolute error** / **least absolute deviation (LAD)**

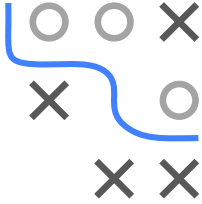
$$L(y, f(\mathbf{x})) = |y - f(\mathbf{x})|$$



*L*1 **VS** *L*2 **– LOSS SURFACE**





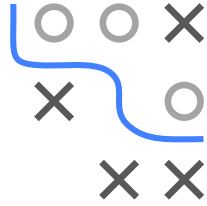


*L*1 loss (left) harder to optimize than *L*2 loss (right)

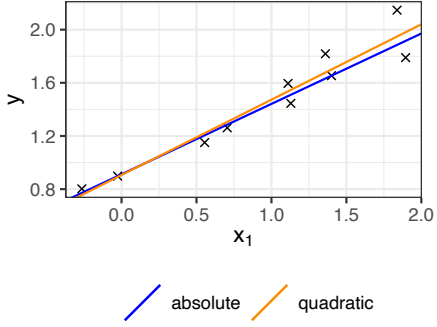
- Convex but **not differentiable** in $y - f(\mathbf{x}) = 0$
- No analytical solution

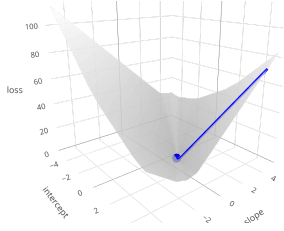
*L*1 **VS** *L*2 **– ESTIMATED PARAMETERS**

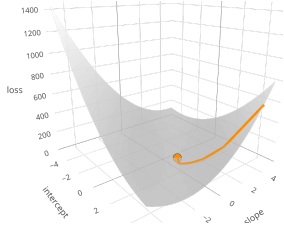
- Results of *L*1 and *L*2 regression often not that different
- Simulated data: $y^{(i)} = 1 + 0.5x_1^{(i)} + \epsilon^{(i)}, \quad \epsilon^{(i)} \overset{i.i.d}{\sim} \mathcal{N}(0, 0.01)$



	intercept	slope
*L*1	0.91	0.53
*L*2	0.91	0.57

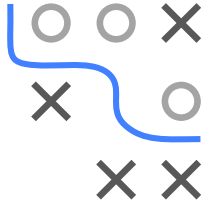


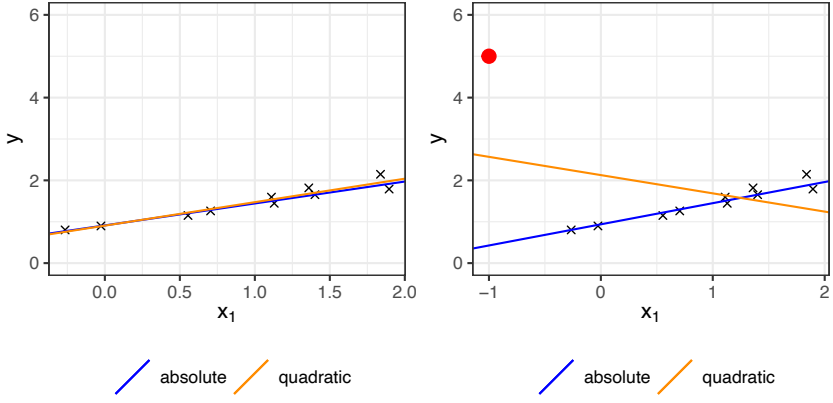




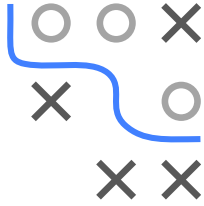
*L*1 **VS** *L*2 **– ROBUSTNESS**

- *L*2 quadratic in residuals $\rightsquigarrow$ outlying points carry lots of weight
- E.g., 3$\times$ residual $\Rightarrow$ 9$\times$ loss contribution
- *L*1 more **robust** in presence of outliers (example ctd.):

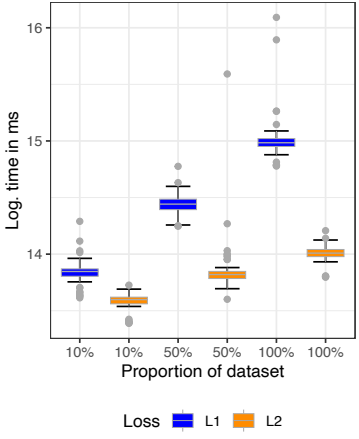




*L*1 **VS** *L*2 **– OPTIMIZATION COST**



- Real-world `weather` problem $\rightsquigarrow$ predict mean temperature
- Compare **time** to fit *L*1 (`quantreg::rq()`) vs *L*2 (`lm::lm()`) for different dataset proportions (repeat 50$\times$)



Loss

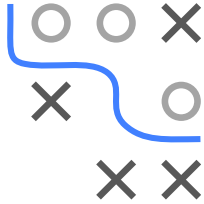
	Fitted: *L*1	Fitted: *L*2
Total *L*1 loss	8.98×10^4	8.99×10^4
Total *L*2 loss	5.83×10^6	5.81×10^6

Estimated coefficients

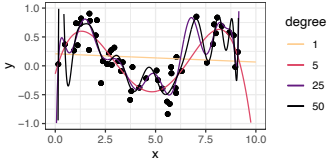
x_j	*L*1: $\hat{\theta}_j$	*L*2: $\hat{\theta}_j$
`Max_temperature`	0.553	0.563
`Min_temperature`	0.441	0.427
`Visibility`	0.026	0.041
`Wind_speed`	0.002	0.010
`Max_wind_speed`	−0.026	−0.039
`(Intercept)`	−0.380	−0.102

*L*1 **slower** to optimize!

Introduction to Machine Learning



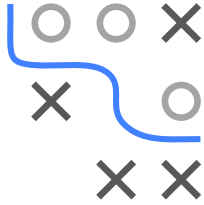
Supervised Regression
Polynomial Regression Models



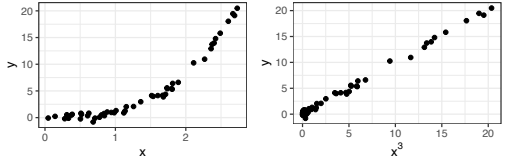
Learning goals

- Learn about general form of linear model
- See how to add flexibility by using polynomials
- Understand that more flexibility is not necessarily better

INCREASING FLEXIBILITY



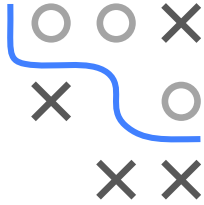
- Recall our definition of LM: model y as linear combo of features
- But: isn't that pretty **inflexible**?
- E.g., here, y does not seem to be a linear function of x...



... but relation to x^3 looks pretty linear!

- Many other trafos conceivable, e.g., $\sin(x_1)$, $\max(0, x_2)$, $\sqrt{x_3}, \ldots$
- Turns out we can use LM much more **flexibly** (and: it's still linear)
  $\rightsquigarrow$ interpretation might get less straightforward, though

THE LINEAR MODEL



- Recall what we previously defined as LM:

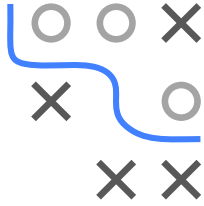
$$f(x) = \theta_0 + \sum_{j=1}^{p} \theta_j x_j = \theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p \qquad (1)$$

- Actually, just special case of "true" LM
- **The linear model** with **basis functions** ϕ_j:

$$f(\mathbf{x}) = \theta_0 + \sum_{j=1}^{p} \theta_j \phi_j(x_j) = \theta_0 + \theta_1 \phi_1(x_1) + \cdots + \theta_p \phi_p(x_p)$$

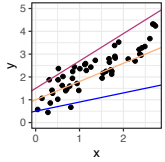
- In Eq. 1, we implicitly use identity trafo: $\phi_j = \mathrm{id}_x : x \mapsto x \quad \forall j$
  $\rightsquigarrow$ we often say LM and imply $\phi_j = \mathrm{id}_x$

THE LINEAR MODEL



- Are models like $f(\mathbf{x}) = \theta_0 + \theta_1 x^2$ **really linear**?
  - Certainly not in covariates:

$$\begin{aligned} a \cdot f(x, \boldsymbol{\theta}) + b \cdot f(x_*, \boldsymbol{\theta}) &= \theta_0(a+b) + \theta_1(ax^2 + bx_*^2) \\ &\neq \theta_0 + \theta_1(ax + bx_*)^2 \\ &= f(ax + bx_*, \boldsymbol{\theta}) \end{aligned}$$



$\boldsymbol{\theta} = (0.5, 0.4)^\top$
$\boldsymbol{\theta} = (1.0, 0.8)^\top$
$\boldsymbol{\theta} = (1.5, 1.2)^\top$

  - Crucially, however, **linear in params**:

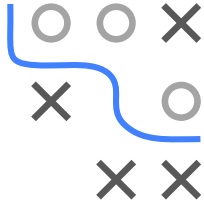
$$\begin{aligned} a \cdot f(x, \boldsymbol{\theta}) + b \cdot f(x, \boldsymbol{\theta}^*) &= a\theta_0 + b\theta_0^* + (a\theta_1 + b\theta_1^*)x^2 \\ &= f(x, a\boldsymbol{\theta} + b\boldsymbol{\theta}^*) \end{aligned}$$

- NB: we still call design matrix $\mathbf{X}$, incorporating possible trafos:

$$\mathbf{X} = \begin{pmatrix} 1 & \phi_1(x_1^{(1)}) & \dots & \phi_p(x_p^{(1)}) \\ \vdots & \vdots & & \vdots \\ 1 & \phi_1(x_1^{(n)}) & \dots & \phi_p(x_p^{(n)}) \end{pmatrix}$$

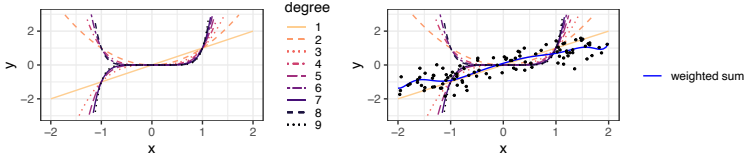
$\rightsquigarrow$ solution via normal equations as usual

POLYNOMIAL REGRESSION



- Simple & flexible choice for basis funs: ***d*-polynomials**
- Idea: map x_j to (weighted) sum of its monomials up to order $d \in \mathbb{N}$

$$\phi^{(d)} : \mathbb{R} \to \mathbb{R}, \quad x_j \mapsto \sum_{k=1}^{d} \beta_k x_j^k$$

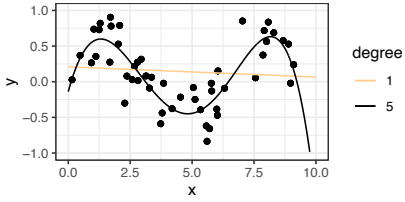


- How to estimate coefficients β_k?
  - Both LM & polynomials **linear** in their params $\rightsquigarrow$ merge
  - E.g., $f(\mathbf{x}) = \theta_0 + \theta_1 \phi^{(d)}(x) = \theta_0 + \sum_{k=1}^{d} \theta_{1,k} x^k$

$$\rightsquigarrow \mathbf{X} = \begin{pmatrix} 1 & x^{(1)} & (x^{(1)})^2 & \dots & (x^{(1)})^d \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x^{(n)} & (x^{(n)})^2 & \dots & (x^{(n)})^d \end{pmatrix}, \quad \boldsymbol{\theta} \in \mathbb{R}^{d+1}$$

POLYNOMIAL REGRESSION – EXAMPLES

Univariate regression, $d \in \{1, 5\}$

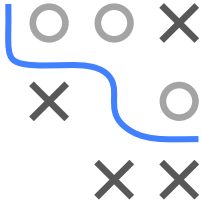


- Data-generating process:

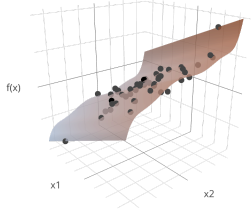
$$y = 0.5 \sin(x) + \epsilon,$$
$$\epsilon \sim \mathcal{N}(0, 0.3^2)$$

- Model:

$$f(x) = \theta_0 + \sum_{k=1}^{d} \theta_{1,k} x^k$$



Bivariate regression, $d = 7$



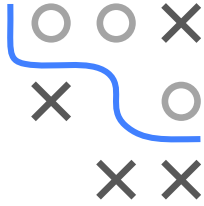
- Data-generating process:

$$y = 1 + 2x_1 + x_2^3 + \epsilon,$$
$$\epsilon \sim \mathcal{N}(0, 0.5^2)$$

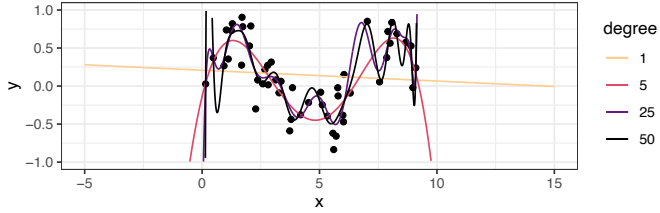
- Model:

$$f(x) = \theta_0 + \theta_1 x_1 + \sum_{k=1}^{7} \theta_{2,k} x_2^k$$

COMPLEXITY OF POLYNOMIALS



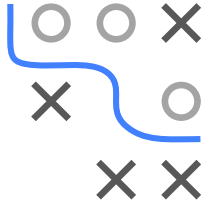
- Higher d allows to learn more complex functions
  $\rightsquigarrow$ richer hyp space / higher **capacity**



- Should we then simply let $d \to \infty$?
  - **No**: data contains random **noise** – not part of true DGP
  - Model with overly high capacity learns all those spurious patterns $\rightsquigarrow$ poor generalization to new data
  - Also, higher d can lead to oscillation esp. at bounds (Runge's phenomenon[1])

[1] Interpolation of m equidistant points with d-polynomial only well-conditioned for $d < 2\sqrt{m}$. Plot: 50 points, models with $d > 14$ instable (under equidistance assumption)

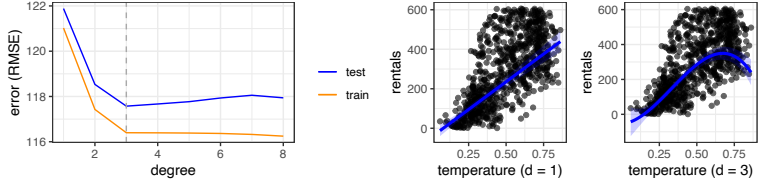
BIKE RENTAL EXAMPLE



- OpenML task `dailybike`: predict `rentals` from weather conditions
- Hunch: non-linear effect of `temperature` $\rightsquigarrow$ include with polynomial:

$$f(\mathbf{x}) = \sum_{k=1}^{d} \theta_{\text{temperature},k} x_{\text{temperature}}^{k} + \theta_{\text{season}} x_{\text{season}} + \theta_{\text{humidity}} x_{\text{humidity}}$$

- Test error[2] confirms suspicion $\rightsquigarrow$ minimal for $d = 3$

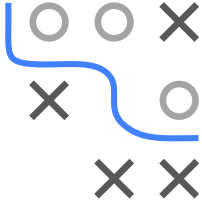


d	θ_0	$\theta_{\text{temp},1}$	$\theta_{\text{temp},2}$	$\theta_{\text{temp},3}$	θ_{sSPRING}	θ_{sSUMMER}	θ_{sFALL}	θ_{hum}	**test error**
1	377.3	2778.2			101.0	57.0	80.1	-230.0	**121.9**
3	419.3	2645.8	-963.1	-430.9	71.9	75.8	56.6	-283.8	**117.6**

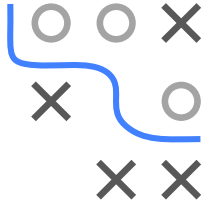
- Conclusion: flexible effects can improve fit/performance

[2] Reliable insights about model performance only via separate test dataset not used during training (here computed via 10-fold *cross validation*). Much more on this in Evaluation chapter

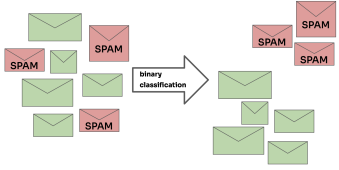
INTRODUCTION TO MACHINE LEARNING



Introduction to Machine Learning



Classification Tasks

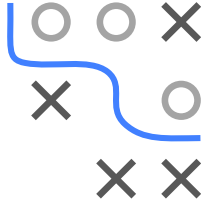


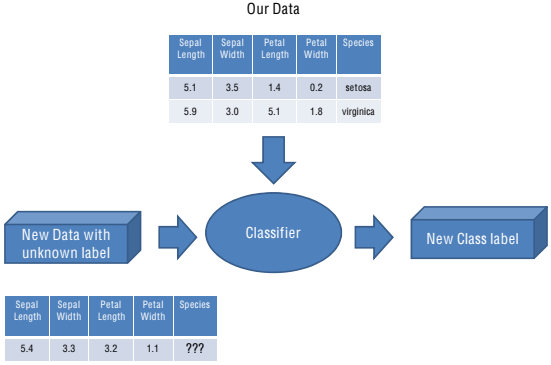
Learning goals

- Classification is supervised learning with categorical labels
- Binary vs. multiclass
- Some examples of classification tasks

CLASSIFICATION

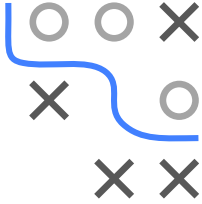
Learn functions that assign class labels to observation / feature vectors. Each observation belongs to exactly one class. The main difference to regression is that the target is categorical.

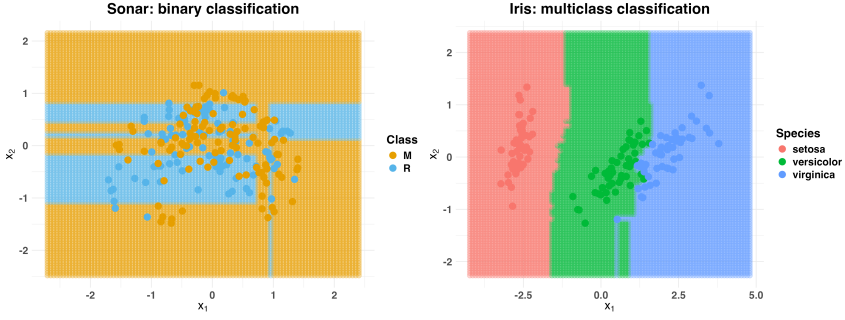




BINARY AND MULTICLASS TASKS

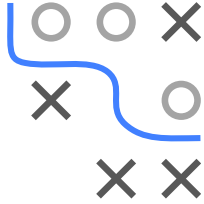
Tasks have a finite number of (unordered) classes.
They can be **binary** or **multiclass**.

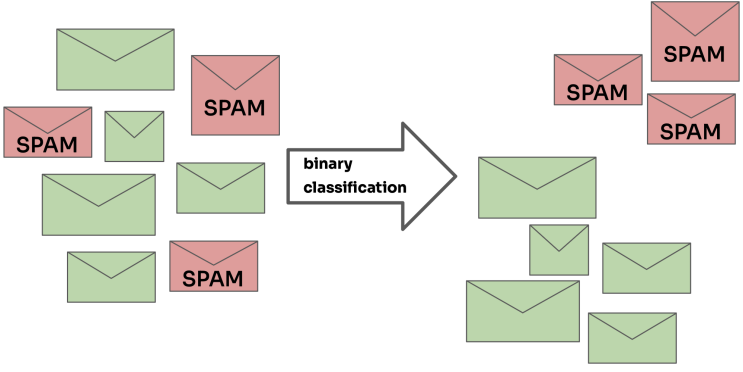




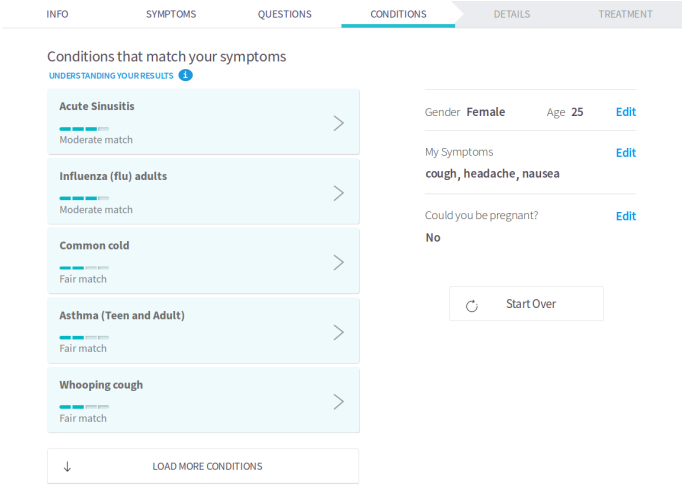
BINARY CLASSIFICATION TASK - EXAMPLES

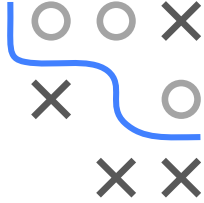
- Credit risk prediction, based on personal data and transactions
- Spam detection, based on textual features
- Churn prediction, based on customer behavior
- Predisposition for specific illness, based on genetic data





MULTICLASS TASK - MEDICAL DIAGNOSIS

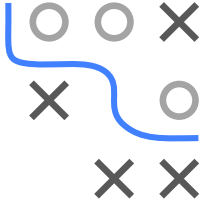




https://symptoms.webmd.com

MULTICLASS TASK - IRIS

The iris dataset was introduced by the statistician Ronald Fisher and is one of the most frequent used data sets. Originally, it was designed for linear discriminant analysis.



Setosa

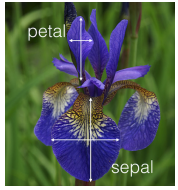
Versicolor

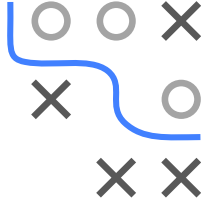
Virginica

Source:
`https://en.wikipedia.org/wiki/Iris_flower_data_set`

MULTICLASS TASK - IRIS

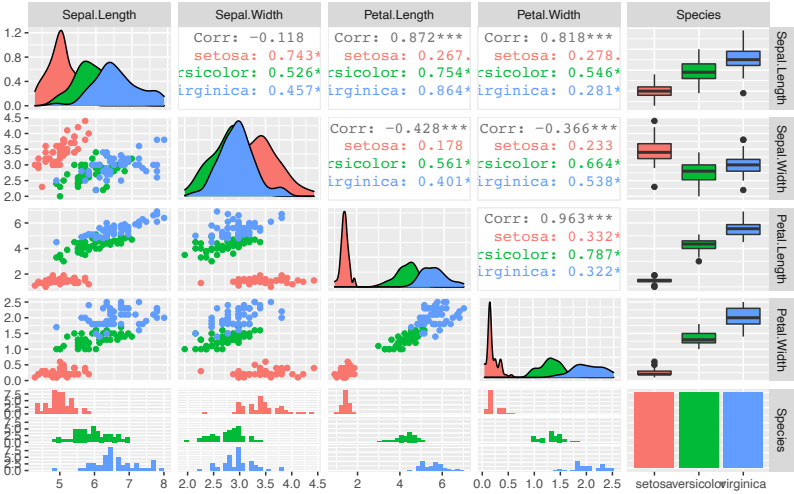
- 150 iris flowers
- Predict subspecies
- Based on sepal and petal length / width in [cm]

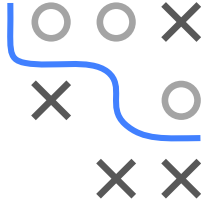




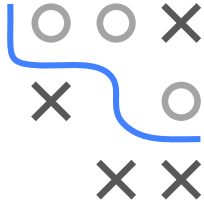
```
##       Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
##    1:          5.1         3.5          1.4         0.2    setosa
##    2:          4.9         3.0          1.4         0.2    setosa
##    3:          4.7         3.2          1.3         0.2    setosa
##    4:          4.6         3.1          1.5         0.2    setosa
##    5:          5.0         3.6          1.4         0.2    setosa
##   ---
##  146:          6.7         3.0          5.2         2.3 virginica
##  147:          6.3         2.5          5.0         1.9 virginica
##  148:          6.5         3.0          5.2         2.0 virginica
##  149:          6.2         3.4          5.4         2.3 virginica
##  150:          5.9         3.0          5.1         1.8 virginica
```

MULTICLASS TASK - IRIS

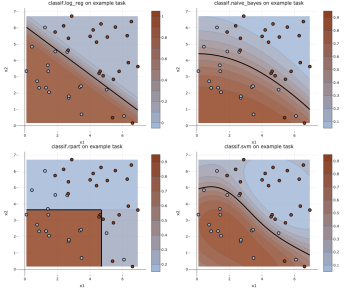




Introduction to Machine Learning



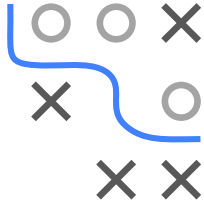
Classification
Basic Definitions



Learning goals

- Basic notation
- Hard labels vs. probabilities vs. scores
- Decision regions and boundaries
- Generative vs. discriminant approaches

NOTATION AND TARGET ENCODING



- In classification, we aim at predicting a discrete output

$$y \in \mathcal{Y} = \{C_1, ..., C_g\}$$

  with $2 \leq g < \infty$, given data $\mathcal{D}$
- For convenience, we often encode these classes differently
- **Binary case,** $g = 2$: Usually use $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, +1\}$
- **Multiclass case,** $g \geq 3$: Could use $\mathcal{Y} = \{1, \ldots, g\}$, but often use **one-hot encoding** $o(y)$, i.e., g-length vector with $o_k(y) = \mathbb{I}(y = k) \in \{0, 1\}$:

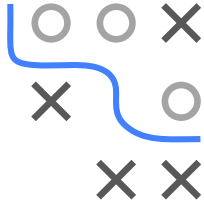
ID	Features	Species
1	...	Setosa
2	...	Setosa
3	...	Versicolor
4	...	Virginica
5	...	Setosa

one-hot encoding

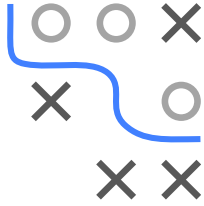
o(Species)
(1, 0, 0)
(1, 0, 0)
(0, 1, 0)
(0, 0, 1)
(1, 0, 0)

CLASSIFICATION MODELS

- While for regression the model $f : \mathcal{X} \to \mathbb{R}$ simply maps to the label space $\mathcal{Y} = \mathbb{R}$, classification is slightly more complicated.
- We sometimes like our models to output (hard) classes, sometimes probabilities, sometimes class scores. The latter 2 are vectors.
- The most basic / common form is the score-based classifier, this is why we defined models already as $f : \mathcal{X} \to \mathbb{R}^g$.
- To minimize confusion, we distinguish between all 3 in notation: $h(\mathbf{x})$ for hard labels, $\pi(\mathbf{x})$ for probabilities and $f(\mathbf{x})$ for scores
- Why all of that and not only hard labels? a) Scores / probabilities are more informative than hard class predictions; b) from an optimization perspective, it is much (!) easier to work with continuous values.



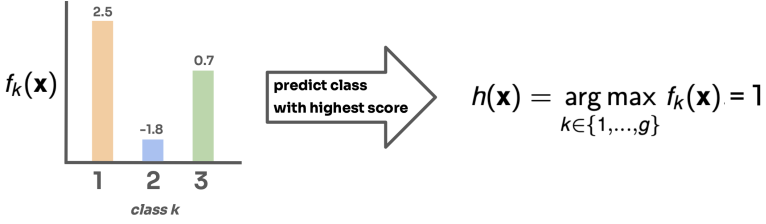
SCORING CLASSIFIERS



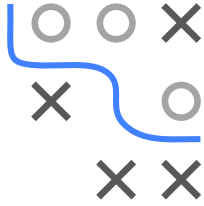
- Construct g **discriminant / scoring functions** $f_1, ..., f_g : \mathcal{X} \to \mathbb{R}$
- Predicted class is usually the one with max score

$$h(\mathbf{x}) = \underset{k \in \{1,...,g\}}{\arg\max}\ f_k(\mathbf{x})$$

- For $g = 2$, a single discriminant function $f(\mathbf{x}) = f_1(\mathbf{x}) - f_{-1}(\mathbf{x})$ is sufficient (here, it's natural to label classes with $\{-1, +1\}$ and we used slight abuse of notation for the subscripts), class labels are constructed by $h(\mathbf{x}) = \text{sgn}(f(\mathbf{x}))$
- $|f(\mathbf{x})|$ or $|f_k(\mathbf{x})|$ is loosely called "confidence"



PROBABILISTIC CLASSIFIERS



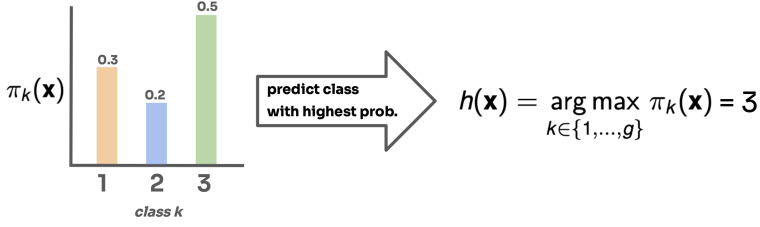
- Construct g **probability functions**

$$\pi_1, ..., \pi_g : \mathcal{X} \to [0, 1], \sum_{k=1}^{g} \pi_k(\mathbf{x}) = 1$$

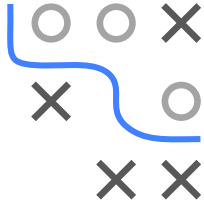
- Predicted class is usually the one with max probability

$$h(\mathbf{x}) = \underset{k \in \{1,...,g\}}{\arg\max}\ \pi_k(\mathbf{x})$$

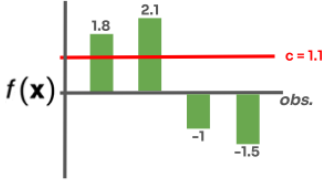
- For $g = 2$, single $\pi(\mathbf{x})$ is constructed, which models the predicted probability for the positive class (natural to encode $\mathcal{Y} = \{0, 1\}$)

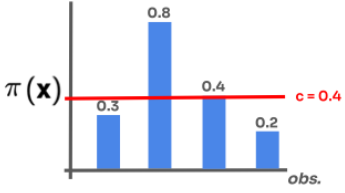


THRESHOLDING

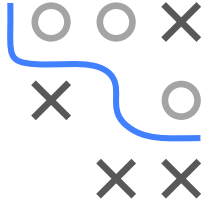


- For imbalanced cases or class with costs, we might want to deviate from the standard conversion of scores to classes
- Introduce basic concept (for binary case) and add details later
- Convert scores or probabilities to class outputs by thresholding: $h(\mathbf{x}) := [\pi(\mathbf{x}) \geq c]$ or $h(\mathbf{x}) := [f(\mathbf{x}) \geq c]$ for some threshold c
- Standard thresholds: $c = 0.5$ for probabilities, $c = 0$ for scores

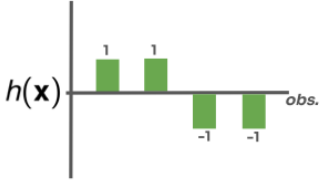


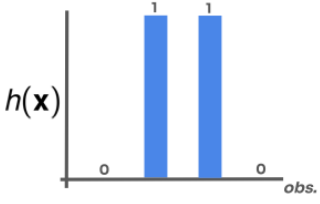


THRESHOLDING



- For imbalanced cases or class with costs, we might want to deviate from the standard conversion of scores to classes
- Introduce basic concept (for binary case) and add details later
- Convert scores or probabilities to class outputs by thresholding: $h(\mathbf{x}) := [\pi(\mathbf{x}) \geq c]$ or $h(\mathbf{x}) := [f(\mathbf{x}) \geq c]$ for some threshold c
- Standard thresholds: $c = 0.5$ for probabilities, $c = 0$ for scores

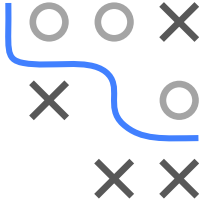


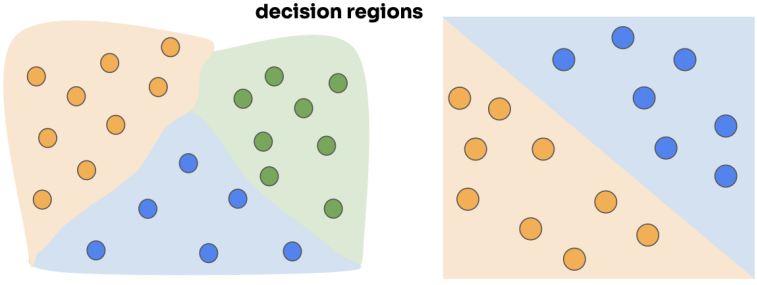


DECISION REGIONS

Set of points **x** where class k is predicted:

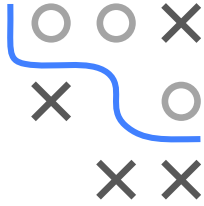
$$\mathcal{X}_k = \{\mathbf{x} \in \mathcal{X} : h(\mathbf{x}) = k\}$$





DECISION BOUNDARIES

Points in space where classes with maximal score are tied and the corresponding hypersurfaces are called **decision boundaries**

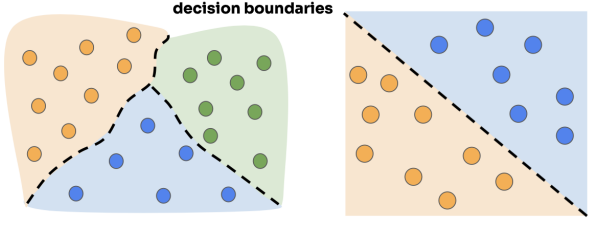


$$\{\mathbf{x} \in \mathcal{X} : \exists\, i \neq j \text{ s.t. } f_i(\mathbf{x}) = f_j(\mathbf{x}) \wedge f_i(\mathbf{x}), f_j(\mathbf{x}) \geq f_k(\mathbf{x})\ \forall k \neq i, j\}$$

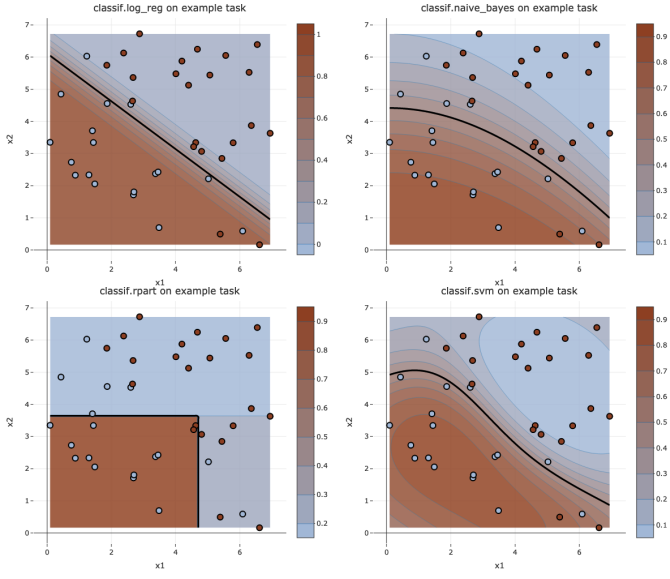
In binary case we can simply use the threshold:

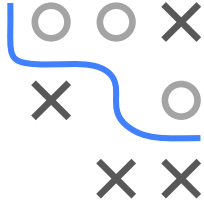
$$\{\mathbf{x} \in \mathcal{X} : f(\mathbf{x}) = c\}$$

$c = 0$ for scores and $c = 0.5$ for probs is consistent with the above.



DECISION BOUNDARY EXAMPLES

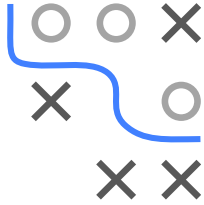




GENERATIVE APPROACH

Models class-conditional $p(\mathbf{x}|y = k)$, and employs Bayes' theorem:

$$\pi_k(\mathbf{x}) \approx \mathbb{P}(y = k \mid \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y = k)\mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})} = \frac{p(\mathbf{x}|y = k)\pi_k}{\sum_{j=1}^{g} p(\mathbf{x}|y = j)\pi_j}$$



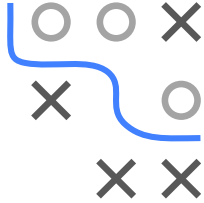
Prior probs $\pi_k = \mathbb{P}(y = k)$ can easily be estimated from training data as relative frequencies of each class:

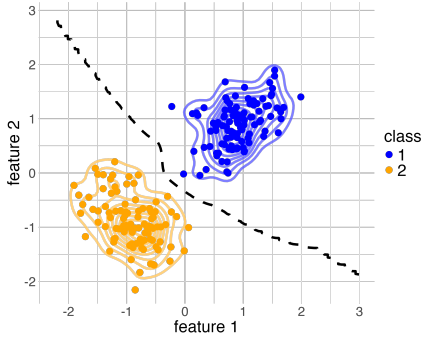
ID	Sex	Age	Class	Survived the Titanic
1	male	49	2nd	no
2	female	23	1st	yes
3	male	32	3rd	no
4	male	51	2nd	no
5	female	49	1st	yes

$$\hat{\pi} = \frac{2}{5}$$

GENERATIVE APPROACH

Decision boundary implicitly defined via the conditional distributions



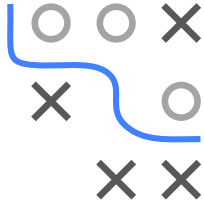


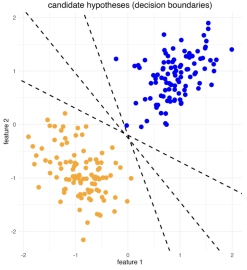
Examples are Naive Bayes, LDA and QDA.
NB: LDA and QDA have 'discriminant' in their name, but are generative!

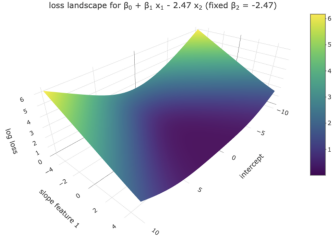
DISCRIMINANT APPROACH

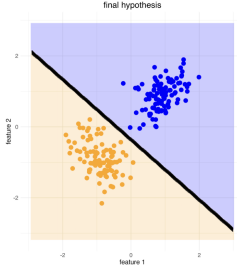
Here we optimize the discriminant functions (or better: their parameters) directly, usually via ERM:

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



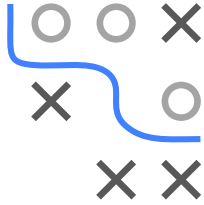




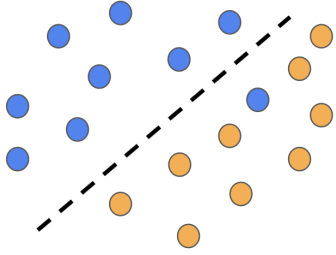


Examples are neural networks, logistic regression and SVMs

Introduction to Machine Learning



Classification
Linear Classifiers

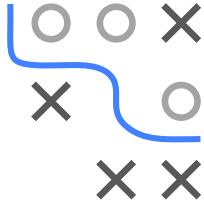


Learning goals

- Linear classifier
- Linear decision boundaries
- Linear separability

LINEAR CLASSIFIERS

Important subclass of classification models.



Definition: If discriminant(s) $f_k(\mathbf{x})$ can be written as affine linear function(s) (possibly through a rank-preserving, monotone transformation g):

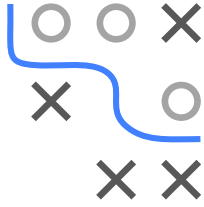
$$g(f_k(\mathbf{x})) = \boldsymbol{w}_k^\top \mathbf{x} + b_k,$$

we will call the classifier **linear**.

- $\boldsymbol{w}_k$ and b_k do not necessarily refer to parameters θ_k, although they often coincide; discriminant simply must be writable in an affine-linear way
- reasons for the transformation is that we only care about the position of the decision boundary

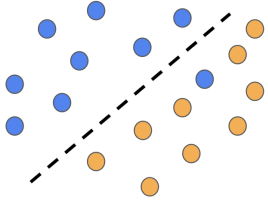
LINEAR DECISION BOUNDARIES

We can also easily show that the decision boundary between classes i and j is a hyperplane. For every $\mathbf{x}$ where there is a tie in scores:



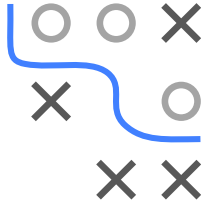
$$\begin{aligned} f_i(\mathbf{x}) &= f_j(\mathbf{x}) \\ g(f_i(\mathbf{x})) &= g(f_j(\mathbf{x})) \\ \boldsymbol{w}_i^\top \mathbf{x} + b_i &= \boldsymbol{w}_j^\top \mathbf{x} + b_j \\ (\boldsymbol{w}_i - \boldsymbol{w}_j)^\top \mathbf{x} + (b_i - b_j) &= 0 \end{aligned}$$

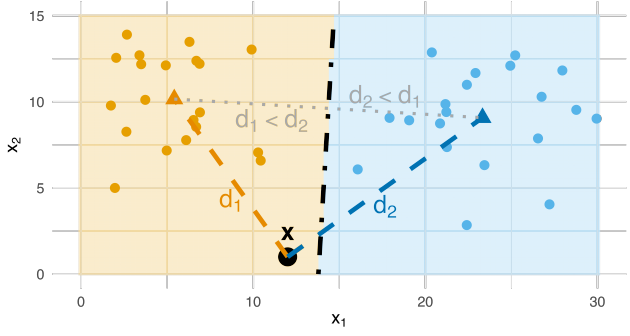
This represents a **hyperplane** separating two classes:



EXAMPLE: 2 CLASSES WITH CENTROIDS

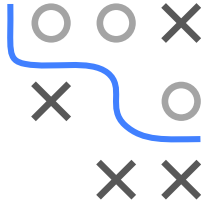
- Model binary problem with centroid $\boldsymbol{\mu_k}$ per class as "parameters"
- Don't really care how the centroids are estimated;
  could use class means, but the following doesn't depend on it
- Classify point **x** by assigning it to class *k* of nearest centroid





EXAMPLE: 2 CLASSES WITH CENTROIDS

Let's calculate the decision boundary:



$$d_1 = ||\mathbf{x}-\boldsymbol{\mu_1}||^2 = \mathbf{x}^\top\mathbf{x}-2\mathbf{x}^\top\boldsymbol{\mu_1}+\boldsymbol{\mu_1}^\top\boldsymbol{\mu_1} = \mathbf{x}^\top\mathbf{x}-2\mathbf{x}^\top\boldsymbol{\mu_2}+\boldsymbol{\mu_2}^\top\boldsymbol{\mu_2} = ||\mathbf{x}-\boldsymbol{\mu_2}||^2$$

Where d is measured using Euclidean distance. This implies:

$$-2\mathbf{x}^\top\boldsymbol{\mu_1} + \boldsymbol{\mu_1}^\top\boldsymbol{\mu_1} = -2\mathbf{x}^\top\boldsymbol{\mu_2} + \boldsymbol{\mu_2}^\top\boldsymbol{\mu_2}$$

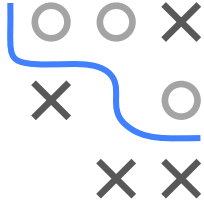
Which simplifies to:

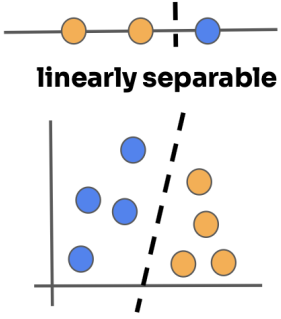
$$2\mathbf{x}^\top(\boldsymbol{\mu_2} - \boldsymbol{\mu_1}) = \boldsymbol{\mu_2}^\top\boldsymbol{\mu_2} - \boldsymbol{\mu_1}^\top\boldsymbol{\mu_1}$$

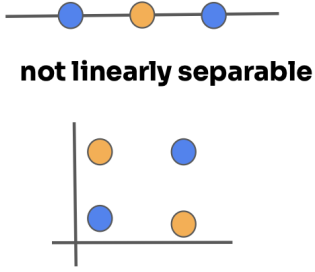
Thus, it's a linear classifier!

LINEAR SEPARABILITY

If there exists a linear classifier that perfectly separates the classes of some dataset, the data are called **linearly separable**.

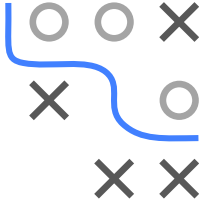


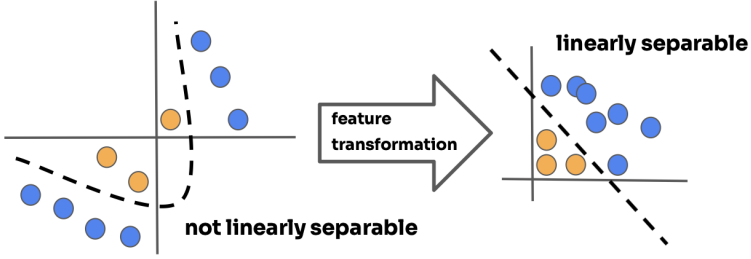




FEATURE TRANSFORMATIONS

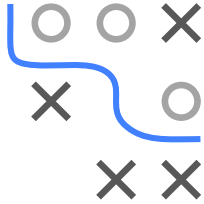
Note that linear classifiers can represent **non-linear** decision boundaries in the original input space if we use derived features like higher order interactions, polynomial features, etc.



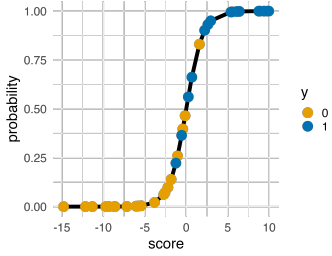


Here we used absolute values to find suitable derived features.

Introduction to Machine Learning



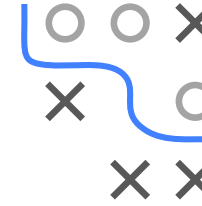
Classification
Logistic Regression



Learning goals

- Hypothesis space of LR
- Log-Loss derivation
- Intuition for loss
- LR as linear classifier

MOTIVATION



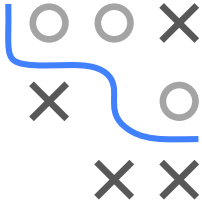
- Let's build a **discriminant** approach, for binary classification, as a probabilistic classifier $\pi(\mathbf{x} \mid \boldsymbol{\theta})$
- We encode $y \in \{0, 1\}$ and use ERM:

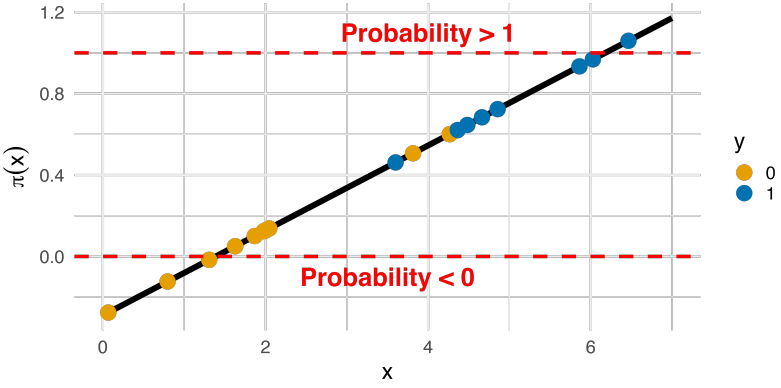
$$\arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^{n} L\left(y^{(i)}, \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

- We want to "copy" over ideas from linear regression
- In the above, our model structure should be "mainly" linear and we need a loss function

DIRECT LINEAR MODEL FOR PROBABILITIES

We could directly use an LM to model $\pi(\mathbf{x} \mid \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}$.
And use L2 loss in ERM.

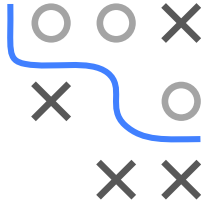




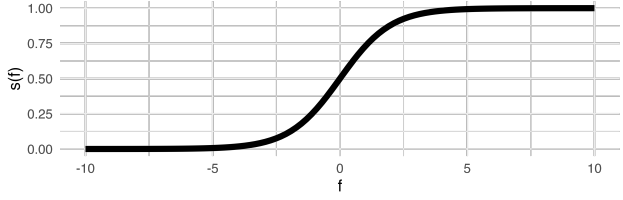
But: This obviously will result in predicted probabilities $\pi(\mathbf{x} \mid \boldsymbol{\theta}) \notin [0, 1]$!

HYPOTHESIS SPACE OF LR

To avoid this, logistic regression "squashes" the estimated linear scores $\boldsymbol{\theta}^\top \mathbf{x}$ to $[0, 1]$ through the **logistic function** s:



$$\pi(\mathbf{x} \mid \boldsymbol{\theta}) = \frac{\exp\left(\boldsymbol{\theta}^\top \mathbf{x}\right)}{1 + \exp\left(\boldsymbol{\theta}^\top \mathbf{x}\right)} = \frac{1}{1 + \exp\left(-\boldsymbol{\theta}^\top \mathbf{x}\right)} = s\left(\boldsymbol{\theta}^\top \mathbf{x}\right) = s(f(\mathbf{x}))$$

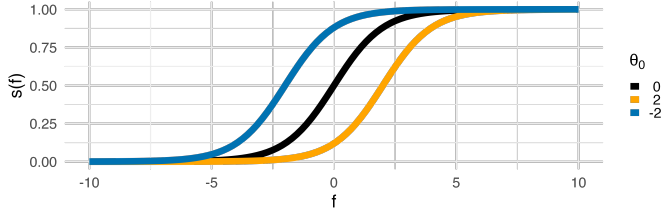


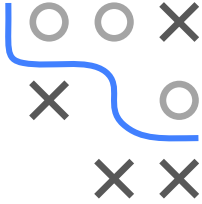
$\Rightarrow$ **Hypothesis space** of LR:

$$\mathcal{H} = \left\{ \pi : \mathcal{X} \to [0, 1] \mid \pi(\mathbf{x} \mid \boldsymbol{\theta}) = s(\boldsymbol{\theta}^\top \mathbf{x}) \mid \boldsymbol{\theta} \in \mathbb{R}^{p+1} \right\}$$

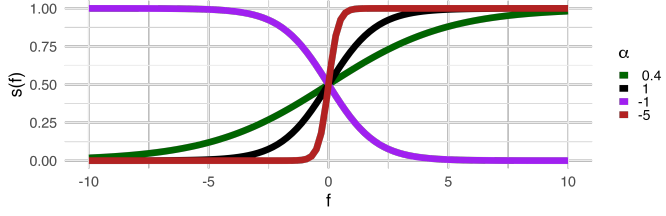
LOGISTIC FUNCTION

Intercept θ_0 shifts $\pi = s(\theta_0 + f) = \frac{\exp(\theta_0+f)}{1+\exp(\theta_0+f)}$ horizontally



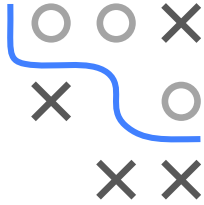


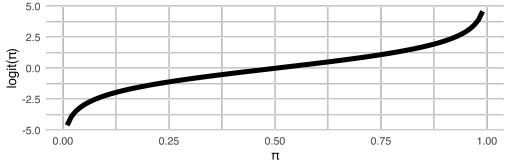
Scaling f like $s(\alpha f) = \frac{\exp(\alpha f)}{1+\exp(\alpha f)}$ controls slope and direction



THE LOGIT

The inverse $s^{-1}(\pi) = \log\left(\frac{\pi}{1-\pi}\right)$ where π is a probability is called **logit** (also called **log odds** since it is equal to the logarithm of the odds $\frac{\pi}{1-\pi}$)



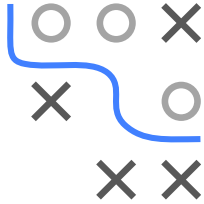


- Positive logits indicate probabilities > 0.5 and vice versa
- E.g.: if $p = 0.75$, odds are $3:1$ and logit is $log(3) \approx 1.1$
- Features $\mathbf{x}$ act linearly on logits, controlled by coefficients $\boldsymbol{\theta}$:

$$s^{-1}(\pi(\mathbf{x})) = \log\left(\frac{\pi(\mathbf{x})}{1-\pi(\mathbf{x})}\right) = \boldsymbol{\theta}^T x$$

DERIVING LOG-LOSS

We need to find a suitable loss function for **ERM**. We look at likelihood which multiplies up $\pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)$ for positive examples, and $1 - \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)$ for negative.



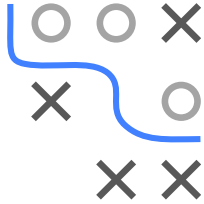
$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i \text{ with } y^{(i)}=1} \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right) \prod_{i \text{ with } y^{(i)}=0} \left(1 - \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

We can now cleverly combine the 2 cases by using exponents (note that only one of the 2 factors is not 1 and "active"):

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{n} \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)^{y^{(i)}} \left(1 - \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)^{1-y^{(i)}}$$

DERIVING LOG-LOSS / 2

Taking the log to convert products into sums:



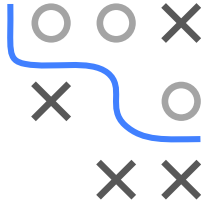
$$\ell(\boldsymbol{\theta}) = \log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log \left(\pi \left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta} \right)^{y^{(i)}} \left(1 - \pi \left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta} \right) \right)^{1 - y^{(i)}} \right)$$

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

Since we want to minimize the risk, we work with the negative $\ell(\boldsymbol{\theta})$:

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

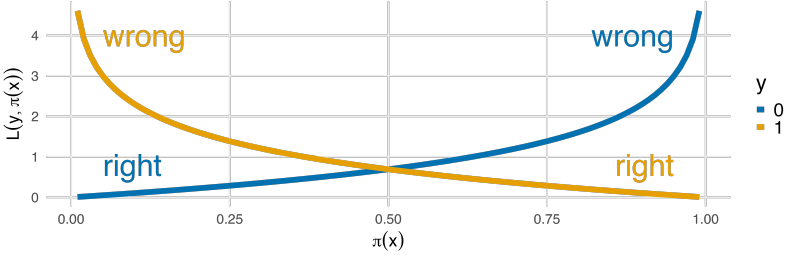
BERNOULLI / LOG LOSS



The resulting loss

$$L(y, \pi) = -y \log(\pi) - (1 - y) \log(1 - \pi)$$

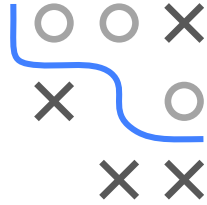
is called **Bernoulli, binomial, log** or **cross-entropy** loss

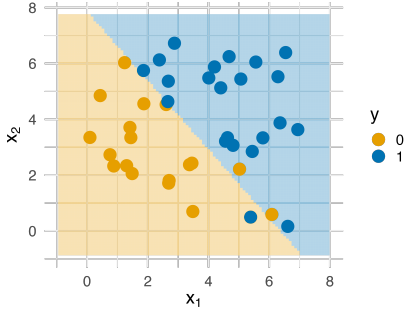


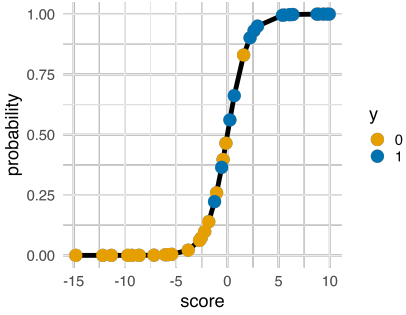
- Penalizes confidently wrong predictions heavily
- Is used for many other classifiers, e.g., in NNs or boosting

LOGISTIC REGRESSION IN 2D

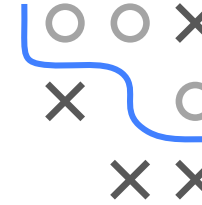
LR is a linear classifier, as $\pi(\mathbf{x} \mid \boldsymbol{\theta}) = s\left(\boldsymbol{\theta}^\top \mathbf{x}\right)$ and s is isotonic.







OPTIMIZATION



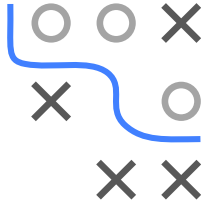
- Log-Loss is convex, under regularity conditions LR has a unique solution (because of its linear structure), but not an analytical one
- To fit LR we use numerical optimization, e.g., Newton-Raphson
- If data is linearly separable, the optimization problem is unbounded and we would not find a solution; way out is regularization
- Why not use least squares on $\pi(\mathbf{x}) = s(f(\mathbf{x}))$?
  Answer: ERM problem is not convex anymore :(
- We can also write the ERM as

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

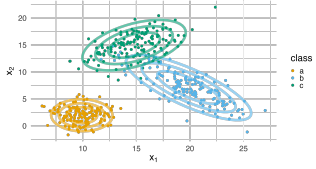
With $f(\mathbf{x} \mid \boldsymbol{\theta}) = \boldsymbol{\theta}^T\mathbf{x}$ and $L(y, f) = -yf + \log(1 + \exp(f))$

This combines the sigmoid with the loss and shows a convex loss directly on a linear function

Introduction to Machine Learning



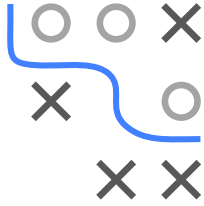
Classification
Discriminant Analysis



Learning goals

- LDA and QDA construction principle based on generative approach
- How are their parameters estimated
- Linear and quadratic decision boundaries

LINEAR DISCRIMINANT ANALYSIS



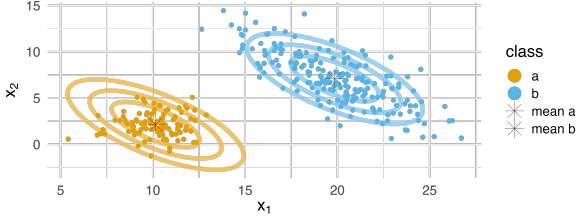
Generative approach, following Bayes' theorem:

$$\pi_k(\mathbf{x}) \approx \mathbb{P}(y = k \mid \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y = k)\mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})} = \frac{p(\mathbf{x}|y = k)\pi_k}{\sum_{j=1}^{g} p(\mathbf{x}|y = j)\pi_j}$$

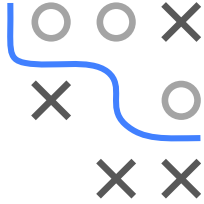
Assume that distribution $p(\mathbf{x}|y = k)$ per class is **multivariate Gaussian**:

$$p(\mathbf{x}|y = k) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu_k})^T\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu_k})\right)$$

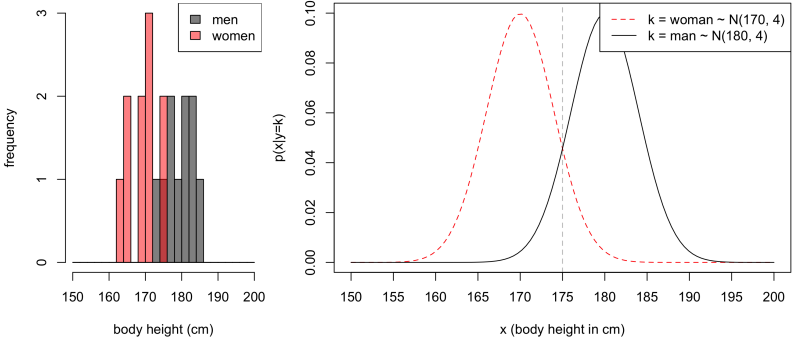
with **equal covariance structure**, so $\Sigma_k = \Sigma \quad \forall k$



UNIVARIATE EXAMPLE



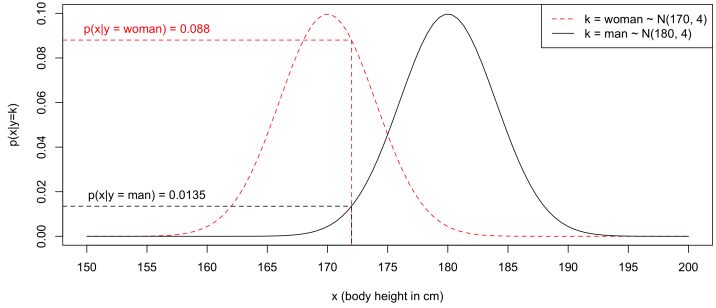
- Classify a new person as male or female based on their height (naive toy example, unrealistic in many ways)
- We will compute in the true DGP, so we assume we know all distributions and their params; we use the LDA setup

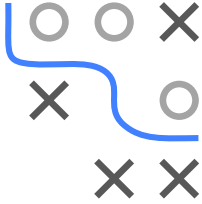


Optimal separation is located at the intersection (= decision boundary)!

UNIVARIATE EXAMPLE: EQUAL CLASS SIZES

Let's compute posterior probability that a 172 cm tall person is male



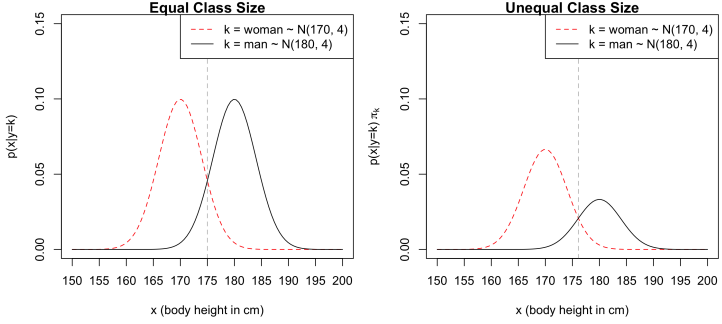


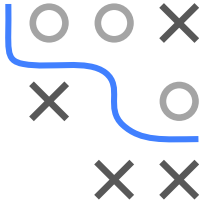
Assuming equal class sizes, prior probs π_k cancel out (since $\pi_{man} = \pi_{woman}$):

$$\mathbb{P}(y = \text{man} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid y = \text{man})}{p(\mathbf{x} \mid y = \text{man}) + p(\mathbf{x} \mid y = \text{woman})} = \frac{0.0135}{0.0135 + 0.088} = 0.13$$

UNIVARIATE EXAMPLE: UNEQUAL CLASS SIZES

For unequal class sizes (e.g., $\pi_{woman} = 2\pi_{man}$), the prior probs matter and cause a shift of the decision boundary towards the smaller class



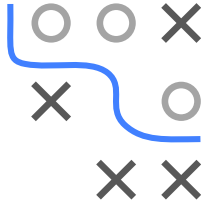


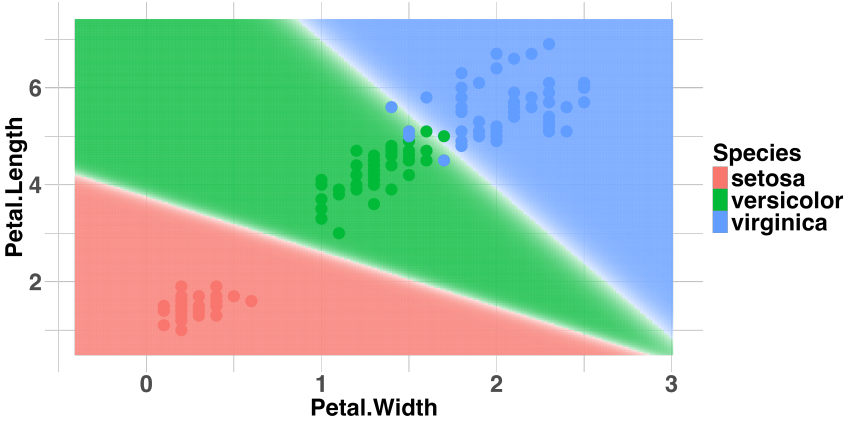
$$\mathbb{P}(y = \text{man} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid y = \text{man})\pi_{man}}{p(\mathbf{x} \mid y = \text{man})\pi_{man} + p(\mathbf{x} \mid y = \text{woman})\pi_{woman}}$$

$$= \frac{0.0135 \cdot \frac{1}{3}}{0.0135 \cdot \frac{1}{3} + 0.088 \cdot \frac{2}{3}} = 0.0712$$

LDA AS LINEAR CLASSIFIER

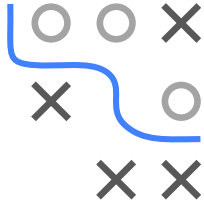
Because of the equal covariance structure of all class-specific Gaussians, the decision boundaries of LDA are always linear





LDA AS LINEAR CLASSIFIER

Can easily prove this by showing that posteriors can be written as affine-linear functions - up to rank-preserving transformation:



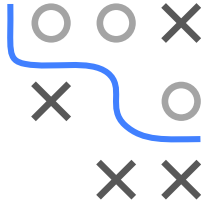
$$\pi_k(\mathbf{x}) = \frac{\pi_k \cdot p(\mathbf{x}|y=k)}{p(\mathbf{x})} = \frac{\pi_k \cdot p(\mathbf{x}|y=k)}{\sum_{j=1}^{g} \pi_j \cdot p(\mathbf{x}|y=j)}$$

As the denominator is the same for all classes we only need to consider

$$\pi_k \cdot p(\mathbf{x}|y=k)$$

and show that this can be written as a linear function of $\mathbf{x}$.

LDA AS LINEAR CLASSIFIER



$$
\begin{aligned}
& \pi_k \cdot p(\mathbf{x}|y=k) \\
\propto \quad & \pi_k \exp\left(-\tfrac{1}{2}\mathbf{x}^T\Sigma^{-1}\mathbf{x} - \tfrac{1}{2}\boldsymbol{\mu_k}^T\Sigma^{-1}\boldsymbol{\mu_k} + \mathbf{x}^T\Sigma^{-1}\boldsymbol{\mu_k}\right) \\
= \quad & \exp\left(\log \pi_k - \tfrac{1}{2}\boldsymbol{\mu_k}^T\Sigma^{-1}\boldsymbol{\mu_k} + \mathbf{x}^T\Sigma^{-1}\boldsymbol{\mu_k}\right) \exp\left(-\tfrac{1}{2}\mathbf{x}^T\Sigma^{-1}\mathbf{x}\right) \\
= \quad & \exp\left(w_{0k} + \mathbf{x}^T\boldsymbol{w_k}\right) \exp\left(-\tfrac{1}{2}\mathbf{x}^T\Sigma^{-1}\mathbf{x}\right) \\
\propto \quad & \exp\left(w_{0k} + \mathbf{x}^T\boldsymbol{w_k}\right)
\end{aligned}
$$

by defining $w_{0k} := \log \pi_k - \frac{1}{2}\boldsymbol{\mu_k}^T\Sigma^{-1}\boldsymbol{\mu_k}$ and $\boldsymbol{w_k} := \Sigma^{-1}\boldsymbol{\mu_k}$.

By finally taking the log, we can write our transformed scores as linear:

$$f_k(\mathbf{x}) = w_{0k} + \mathbf{x}^T\boldsymbol{w_k}$$

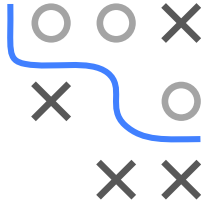
- The above is a little bit “lax” so lets carefully check
- We left out several (pos) multiplicative constants
- $\exp\left(-\frac{1}{2}\mathbf{x}^T\Sigma^{-1}\mathbf{x}\right)$ contains $\mathbf{x}$ but is the same for all classes
- $\log(at + b)$ is still isotonic for $a > 0$

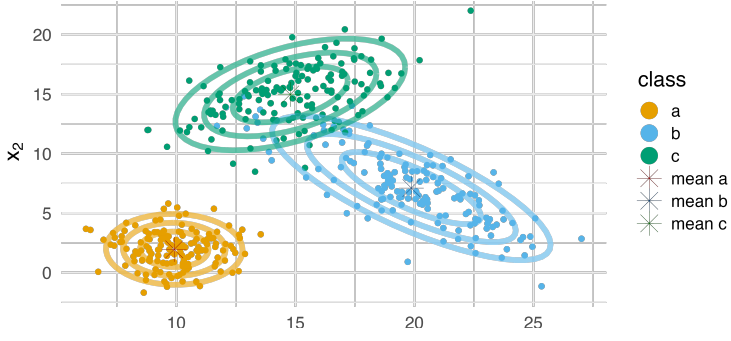
QUADRATIC DISCRIMINANT ANALYSIS

Doesn't assume equal covariances Σ_k per class, so generalizes LDA:

$$p(\mathbf{x}|y=k) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu_k})^T\Sigma_k^{-1}(\mathbf{x}-\boldsymbol{\mu_k})\right)$$

$\Rightarrow$ Better data fit but **requires estimation of more parameters** (Σ_k)!

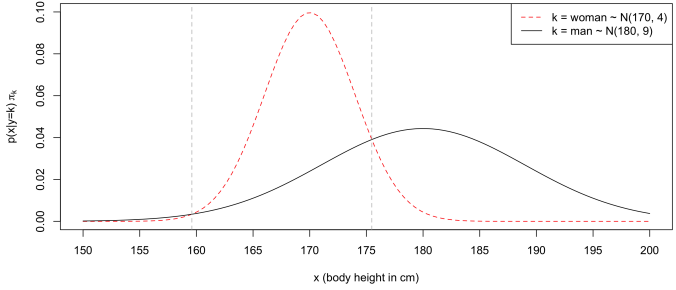




UNIVARIATE EXAMPLE WITH QDA

Different covariance matrices lead to multiple classification rules:

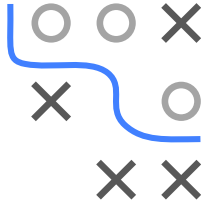
- $x < 159.6$ is being assigned to class *man*.
- $159.6 < x < 175.5$ is being assigned to class *woman*.
- $x > 175.5$ is being assigned to class *man*.



$\Rightarrow$ The separation function is quadratic, we learn a curved decision boundary (in 1D a little bit weird, as we learn an interval)

QDA DECISION BOUNDARIES

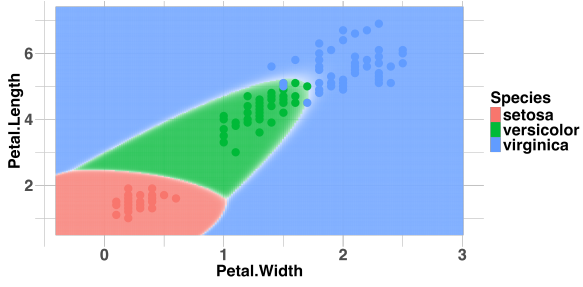
$$\begin{aligned}\pi_k(\mathbf{x}) &\propto \pi_k \cdot p(\mathbf{x}|y=k) \\ &\propto \pi_k |\Sigma_k|^{-\frac{1}{2}} \exp(-\frac{1}{2}\mathbf{x}^T\Sigma_k^{-1}\mathbf{x} - \frac{1}{2}\boldsymbol{\mu_k}^T\Sigma_k^{-1}\boldsymbol{\mu_k} + \mathbf{x}^T\Sigma_k^{-1}\boldsymbol{\mu_k})\end{aligned}$$



Taking log, we get a quadratic discriminant function in x:

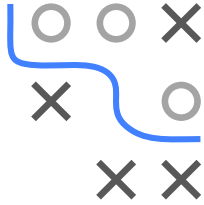
$$\log \pi_k - \frac{1}{2}\log|\Sigma_k| - \frac{1}{2}\boldsymbol{\mu_k}^T\Sigma_k^{-1}\boldsymbol{\mu_k} + \mathbf{x}^T\Sigma_k^{-1}\boldsymbol{\mu_k} - \frac{1}{2}\mathbf{x}^T\Sigma_k^{-1}\mathbf{x}$$

Allowing for curved decision boundaries:



PARAMETER ESTIMATION

Parameters $\boldsymbol{\theta}$ are estimated in a straightforward manner by:



$$\hat{\pi}_k = \frac{n_k}{n}, \text{ where } n_k \text{ is the number of class-}k \text{ observations}$$

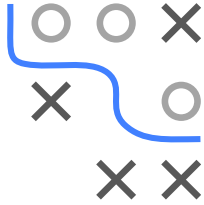
$$\hat{\boldsymbol{\mu}}_{\boldsymbol{k}} = \frac{1}{n_k} \sum_{i:y^{(i)}=k} \mathbf{x}^{(i)}$$

$$\hat{\Sigma}_k = \frac{1}{n_k - 1} \sum_{i:y^{(i)}=k} (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_{\boldsymbol{k}})(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_{\boldsymbol{k}})^T \qquad \text{(QDA)}$$

$$\hat{\Sigma} = \frac{1}{n - g} \sum_{k=1}^{g} \sum_{i:y^{(i)}=k} (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_{\boldsymbol{k}})(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_{\boldsymbol{k}})^T \quad \text{(LDA)}$$

As $\hat{\Sigma}_k, \hat{\Sigma}$ are $p \times p$ matrices (for p features), estimating all $\hat{\Sigma}_k$ involves $\frac{p(p+1)}{2} \cdot g$ parameters across g classes (vs. just $\frac{p(p+1)}{2}$ for LDA's $\hat{\Sigma}$)
(in addition to estimating priors and class means)

QDA PARAMETER ESTIMATION EXAMPLE



E.g., for a simple two-class, 2-dimensional dataset:

Class 1: $\mathbf{x}_1 = \begin{pmatrix}1\\2\end{pmatrix}, \mathbf{x}_2 = \begin{pmatrix}2\\3\end{pmatrix}$, Class 2: $\mathbf{x}_3 = \begin{pmatrix}6\\8\end{pmatrix}, \mathbf{x}_4 = \begin{pmatrix}7\\9\end{pmatrix}, \mathbf{x}_5 = \begin{pmatrix}8\\10\end{pmatrix}$

Class priors: $\hat{\pi}_1 = \frac{n_1}{n} = \frac{2}{5} = 0.4, \quad \hat{\pi}_2 = \frac{n_2}{n} = \frac{3}{5} = 0.6$

Class means: $\hat{\boldsymbol{\mu}}_1 = \frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_2) = \begin{pmatrix}1.5\\2.5\end{pmatrix}, \quad \hat{\boldsymbol{\mu}}_2 = \frac{1}{3}(\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5) = \begin{pmatrix}7\\9\end{pmatrix}$

Class covariances:

$$(\mathbf{x}_1 - \hat{\boldsymbol{\mu}}_1)(\mathbf{x}_1 - \hat{\boldsymbol{\mu}}_1)^\top = \begin{pmatrix}0.25 & 0.25\\0.25 & 0.25\end{pmatrix} = (\mathbf{x}_2 - \hat{\boldsymbol{\mu}}_1)(\mathbf{x}_2 - \hat{\boldsymbol{\mu}}_1)^\top$$

$$\Rightarrow \hat{\Sigma}_1 = \frac{1}{1}\left(\begin{pmatrix}0.25 & 0.25\\0.25 & 0.25\end{pmatrix} + \begin{pmatrix}0.25 & 0.25\\0.25 & 0.25\end{pmatrix}\right) = \begin{pmatrix}0.5 & 0.5\\0.5 & 0.5\end{pmatrix}$$

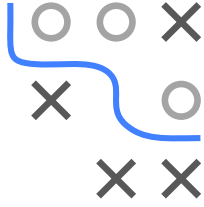
$$(\mathbf{x}_3 - \hat{\boldsymbol{\mu}}_2)(\mathbf{x}_3 - \hat{\boldsymbol{\mu}}_2)^\top = \begin{pmatrix}1 & 1\\1 & 1\end{pmatrix} = (\mathbf{x}_5 - \hat{\boldsymbol{\mu}}_2)(\mathbf{x}_5 - \hat{\boldsymbol{\mu}}_2)^\top,$$

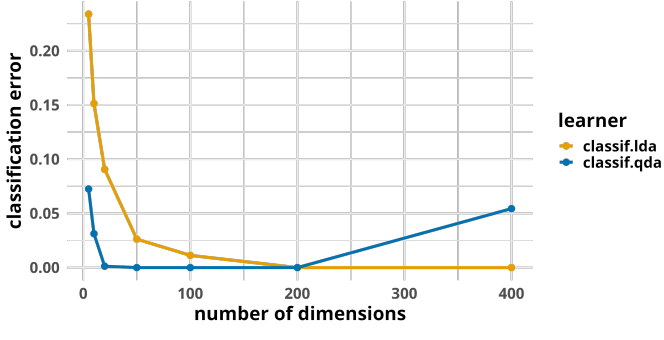
$$(\mathbf{x}_4 - \hat{\boldsymbol{\mu}}_2)(\mathbf{x}_4 - \hat{\boldsymbol{\mu}}_2)^\top = \begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$$

$$\Rightarrow \hat{\Sigma}_2 = \frac{1}{2}\left(\begin{pmatrix}1 & 1\\1 & 1\end{pmatrix} + \begin{pmatrix}0 & 0\\0 & 0\end{pmatrix} + \begin{pmatrix}1 & 1\\1 & 1\end{pmatrix}\right) = \begin{pmatrix}1 & 1\\1 & 1\end{pmatrix}$$

DISCRIMINANT ANALYSIS COMPARISON

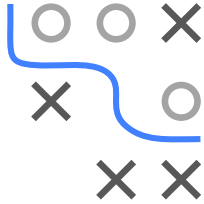
- We benchmark on simple toy data set(s)
- Normally distributed data per class, but unequal cov matrices
- And then increase dimensionality
- We might assume that QDA always wins here ...



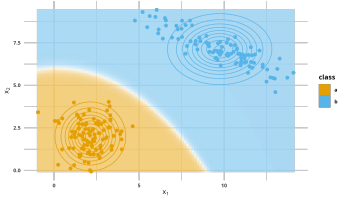


$\Rightarrow$ LDA might be preferable over QDA in higher dimensions!

Introduction to Machine Learning



Classification
Naive Bayes

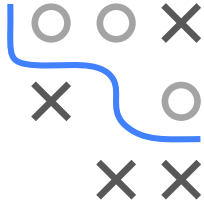


Learning goals

- Construction principle of NB
- Conditional independence assumption
- Numerical and categorical features
- Similarity to QDA, quadratic decision boundaries
- Laplace smoothing

NAIVE BAYES CLASSIFIER

Generative multiclass technique. Remember: We use Bayes' theorem and only need $p(\mathbf{x}|y = k)$ to compute the posterior as:



$$\pi_k(\mathbf{x}) \approx \mathbb{P}(y = k \mid \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y = k)\mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})} = \frac{p(\mathbf{x}|y = k)\pi_k}{\sum_{j=1}^{g} p(\mathbf{x}|y = j)\pi_j}$$

NB is based on a simple **conditional independence assumption**: the features are conditionally independent given class y.

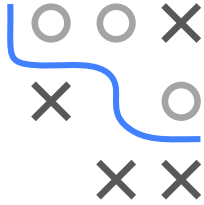
$$p(\mathbf{x}|y = k) = p((x_1, x_2, ..., x_p)|y = k) = \prod_{j=1}^{p} p(x_j|y = k).$$

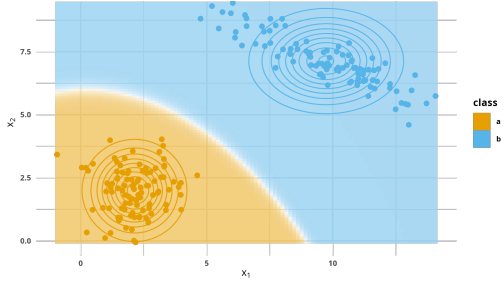
So we only need to specify and estimate the distributions $p(x_j|y = k)$, which is considerably simpler as these are univariate.

NUMERICAL FEATURES

Use univariate Gaussians for $p(x_j|y=k)$, and estimate $(\mu_{kj}, \sigma^2_{kj})$.

Because of $p(\mathbf{x}|y=k) = \prod_{j=1}^{p} p(x_j|y=k)$, joint conditional density is Gaussian with diagonal, non-isotropic covariances, and different across classes, so **QDA with diagonal covariances**.

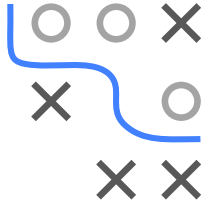




Note: In the above plot the data violates the NB assumption.

NB: CATEGORICAL FEATURES

We use a categorical distribution for $p(x_j|y=k)$ and estimate the probabilities p_{kjm} that, in class k, our j-th feature has value m, $x_j = m$, simply by counting frequencies.



$$p(x_j|y=k) = \prod_m p_{kjm}^{[x_j=m]}$$

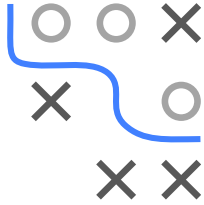
Because of the simple conditional independence structure, it is also very easy to deal with mixed numerical / categorical feature spaces.

ID	Class	Sex	Survived the Titanic
1	2nd	male	no
2	1st	male	yes
3	3rd	female	yes
4	1st	female	yes
5	2nd	female	yes
6	3rd	female	no

$$p(x_{\text{sex}}|y=\text{yes}) = p_{yes,sex,female}^{[x_{\text{sex}}=\text{female}]} \cdot p_{yes,sex,male}^{[x_{\text{sex}}=\text{male}]}$$
$$= \frac{3}{4}^{[x_{\text{sex}}=\text{female}]} \cdot \frac{1}{4}^{[x_{\text{sex}}=\text{male}]}$$

LAPLACE SMOOTHING

If a given class and feature value never occur together in the training data, then the frequency-based probability estimate will be zero, e.g.:
$p_{\text{no, class, 1st}}^{[x_{class}=\text{1st}]} = 0$ (everyone from 1st class survived in the previous table)

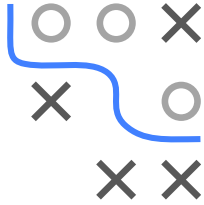


This is problematic because it will wipe out all information in the other probabilities when they are multiplied!

$$\pi_{no}(\text{class = 1st, sex = male}) = \frac{\hat{p}(x_{class}|y = no) \cdot \hat{p}(x_{sex}|y = no) \cdot \hat{\pi}_{no}}{\sum_{j=1}^{g} \hat{p}(\text{class = 1st, sex = male}|y = j)\hat{\pi}_j} = 0$$

LAPLACE SMOOTHING

A simple numerical correction is to set these zero probabilities to a small value to regularize against this case.



- Add constant $\alpha > 0$ (e.g., $\alpha = 1$).
- For a categorical feature x_j with M_j possible values:

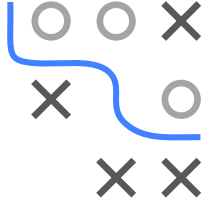
$$p_{kjm}^{[x_j=m]} = \frac{n_{kjm} + \alpha}{n_k + \alpha M_j} \quad \left(\text{instead of } p_{kjm}^{[x_j=m]} = \frac{n_{kjm}}{n_k}\right)$$

where:

- n_{kjm}: count of $x_j = m$ in class k,
- n_k: total counts in class k,
- M_j: number of possible distinct values of x_j.

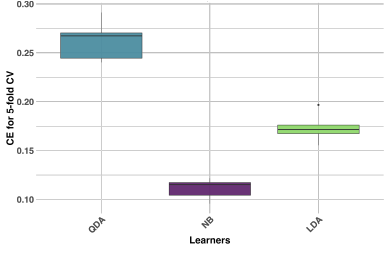
This ensures that our posterior probabilities are non-zero due to such effects, preserving the influence of all features in the model.

NAIVE BAYES: APPLICATION AS SPAM FILTER

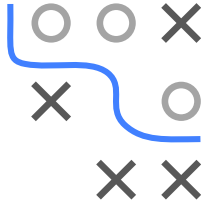


- In the late 90s, NB became popular for e-mail spam detection
- Word counts were used as features to detect spam mails
- Independence assumption implies: occurrence of two words in mail is not correlated, this is often wrong;
  "viagra" more likely to occur in context with "buy"...
- In practice: often still good performance

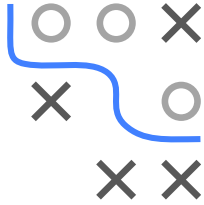
Benchmarking QDA, NB and LDA on `spam`:



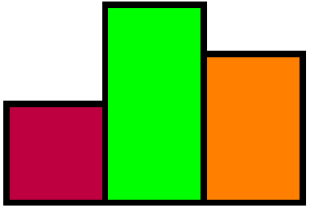
INTRODUCTION TO MACHINE LEARNING



Introduction to Machine Learning



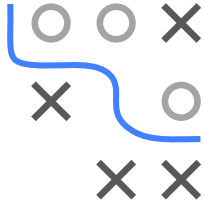
Evaluation
Generalization Error



Learning goals

- Understand the goal of performance estimation
- Know the formal definition of generalization error as a statistical estimator of future performance
- Understand the difference between GE for a model and GE for a learner.
- Understand the difference between outer and inner loss

PERFORMANCE ESTIMATION



- For a trained model, we want to know its future **performance**.
- Training works by ERM on $\mathcal{D}_{\text{train}}$ (inducer, loss, risk minimization):

$$\mathcal{I} : \mathbb{D} \times \mathbf{\Lambda} \to \mathcal{H}, \quad (\mathcal{D}, \boldsymbol{\lambda}) \mapsto \hat{f}_{\mathcal{D},\boldsymbol{\lambda}}.$$

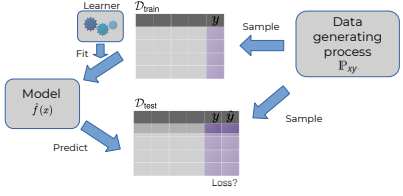
$$\min_{\theta \in \Theta} \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

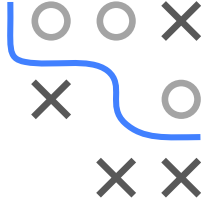
- Due to effects like overfitting, we cannot simply use this **training error** to gauge our model, this is likely optimistically biased. (more on this later!)
- We want: the true expected loss, a.k.a. **generalization error**.
- To reliably estimate it, we need independent, unseen **test data**.
- This simply simulates the application of the model in reality.

GE FOR A FIXED MODEL

- GE for a fixed model: $\mathrm{GE}\left(\hat{f}, L\right) := \mathbb{E}\left[L\left(y, \hat{f}(\mathbf{x})\right)\right]$
  Expectation over a single, random test point $(\mathbf{x}, y) \sim \mathbb{P}_{xy}$.
- Estimator, **if a dedicated test set is available** (size m)

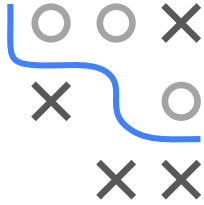
$$\widehat{\mathbf{GE}}(\hat{f}, L) := \frac{1}{m} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{test}}} \left[L\left(y, \hat{f}(\mathbf{x})\right)\right]$$



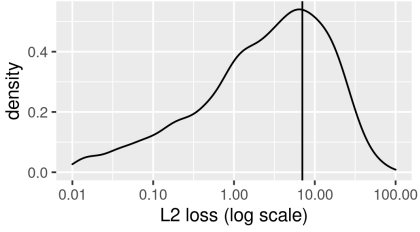


NB: Very often, no dedicated test-set is available, and what we describe here is not same as hold-out splitting (see later).

EXAMPLE: TEST LOSS AS RANDOM VARIABLE

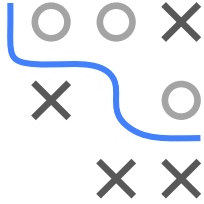


- For a fixed model and dedicated i.i.d. test set, we can easily approximate the complete test loss distribution $L(y, \hat{f}(\mathbf{x}))$.
- LM on `mlbench::friedman1` test problem
- With $n_{\text{train}} = 500$ we create a fixed model
- We feed 5000 fresh test points to model
- And plot the pointwise *L2* loss.



- The result is a unimodal distribution with long tails.
- Mean and one standard deviation to either side are highlighted in grey.

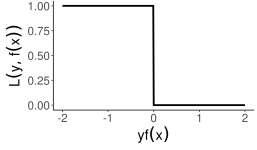
INNER VS OUTER LOSS



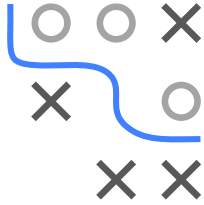
- Sometimes, we would like to evaluate our learner with a different loss L or metric ρ.
- Nomenclature: ERM and **inner loss**; evaluation and **outer loss**.
- Different losses, if computationally advantageous to deviate from outer loss of application; e.g., optimization faster with inner L2 or maybe no implementation for outer loss exists.

Example: Linear binary classifier / Logistic regression.

- Outside: We often want to eval with "nr of misclassifed examples", so 0-1 loss.
- Problem: 0-1 neither differentiable nor continuous. Hence: Inner loss = binomial. (0-1 actually NP hard).
- For evaluation, differentiability is not required.



SET-BASED PERFORMANCE METRICS



- Metric ρ measures quality of predictions as scalar on one test set.

$$\rho : \bigcup_{m \in \mathbb{N}} \left(\mathcal{Y}^m \times \mathbb{R}^{m \times g} \right) \to \mathbb{R}, \quad (\mathbf{y}, \boldsymbol{F}) \mapsto \rho(\mathbf{y}, \boldsymbol{F}).$$

- Needed as some metrics are not observation-based losses but defined on sets, e.g. AUC or metrics in survival analysis.
- For test data of size m, $\boldsymbol{F}$ is prediction matrix

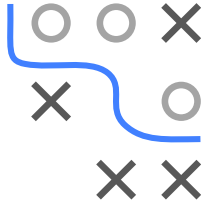
$$\boldsymbol{F} = \begin{bmatrix} \hat{f}(\mathbf{x}^{(1)}) \\ \dots \\ \hat{f}(\mathbf{x}^{(m)}) \end{bmatrix} \in \mathbb{R}^{m \times g}$$

- Point-wise loss L can easily be extended to a ρ_L:

$$\rho_L(\mathbf{y}, \boldsymbol{F}) = \frac{1}{m} \sum_{i=1}^{m} L(y^{(i)}, \boldsymbol{F}^{(i)}) \quad \left(= \frac{1}{m} \sum_{i=1}^{m} L(y^{(i)}, \hat{f}(\mathbf{x}^{(i)}) \right).$$

MODEL GE VS. LEARNER GE

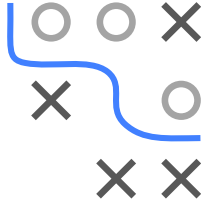
To clear up a major point of confusion (or totally confuse you):



- In ML we frequently face a weird situation.
- We are usually given a single data set, and at the end of our model fitting (and evaluation and selection) process, we will likely fit one model on exactly that complete data set.
- We only trust in unseen-test-error estimation – but have no data left for that final model.
- So in the construction of any practical estimator we cannot really use that final model!
- Hence, we will now evaluate the next best thing: The inducer, and the quality of a model produced when fitted on (nearly) the same number of points!

GENERALIZATION ERROR FOR INDUCER

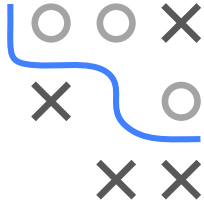
$$\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho) := \lim_{n_{\text{test}} \to \infty} \mathbb{E}\left[\rho\left(\mathbf{y}, \boldsymbol{F}_{\mathcal{D}_{\text{test}}, \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})}\right)\right]$$



- Quality of models when fitted with $\mathcal{I}_{\boldsymbol{\lambda}}$ on n_{train} points from $\mathbb{P}_{xy}$.
- Expectation **both** over $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, sampled independently.
- This is estimated by all following **resampling** procedures.
- NB: All of the models produced during that phase of evaluation are only intermediate results.

GENERALIZATION ERROR FOR INDUCER

$$\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho) := \lim_{n_{\text{test}} \to \infty} \mathbb{E}\left[\rho\left(\mathbf{y}, \boldsymbol{F}_{\mathcal{D}_{\text{test}}, (\mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda}))}\right)\right]$$



- We can already see a potential source of pessimistic bias in our estimator: While we would like to estimate a GE with $n_{\text{train}} = |\mathcal{D}|$, the size of the complete data set, in practice we can only do this for strictly smaller values, so that test data is left to work with.
- For pointwise losses ρ_L:

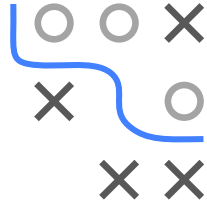
$$\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho_L) := \mathbb{E}\left[L(y, \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})(\mathbf{x}))\right]$$

  Expectation **both** over $\mathcal{D}_{\text{train}}$ and $(\mathbf{x}, y)$ independently from $\mathbb{P}_{xy}$.
- Retcon for GE of model: GE of learner, conditional on $\mathcal{D}_{\text{train}}$

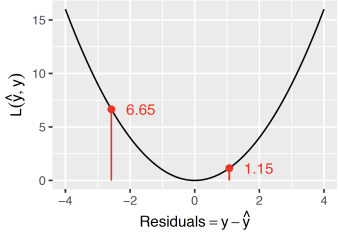
$$\mathrm{GE}\left(\hat{f}, L\right) := \mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho_L | \mathcal{D}_{\text{train}})$$

  if $\hat{f} = \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})$ and $n_{\text{train}} = |\mathcal{D}_{\text{train}}|$.

Introduction to Machine Learning



Evaluation
Measures for Regression

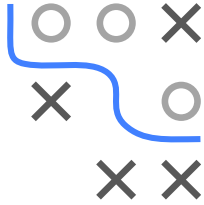


Learning goals

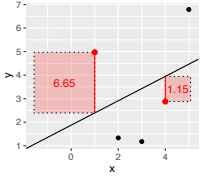
- Know the definitions of mean squared error (MSE) and mean absolute error (MAE)
- Understand the connections of MSE and MAE to L2 and L1 loss
- Know the definition of Spearman's ρ
- Know the definitions of R^2 and generalized R^2

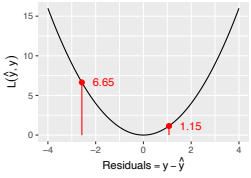
MEAN SQUARED ERROR (MSE)

$$\rho_{MSE}(\mathbf{y}, \boldsymbol{F}) = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 \in [0; \infty) \qquad \rightarrow L2 \text{ loss.}$$



Outliers with large prediction error heavily influence the MSE, as they enter quadratically.



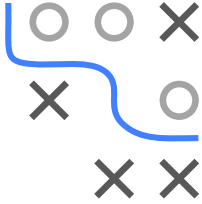


Similar measures:

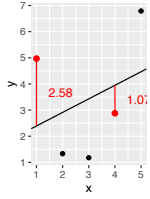
- Sum of squared errors: $\rho_{SSE}(\mathbf{y}, \boldsymbol{F}) = \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2$
- Root MSE (orig. scale): $\rho_{RMSE}(\mathbf{y}, \boldsymbol{F}) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2}$

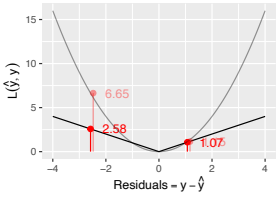
MEAN ABSOLUTE ERROR

$$\rho_{MAE}(\mathbf{y}, \boldsymbol{F}) = \frac{1}{m} \sum_{i=1}^{m} |y^{(i)} - \hat{y}^{(i)}| \in [0; \infty) \qquad \rightarrow L1 \text{ loss.}$$



More robust, less influenced by large residuals, more intuitive than MSE.



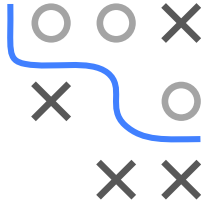


Similar measures:

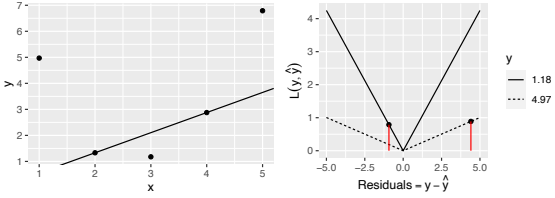
- Median absolute error (for even more robustness)

MEAN ABSOLUTE PERCENTAGE ERROR

$$\rho_{MAPE}(\mathbf{y}, \boldsymbol{F}) = \frac{1}{m} \sum_{i=1}^{m} \left| \frac{y^{(i)} - \hat{y}^{(i)}}{y^{(i)}} \right| \in [0; \infty)$$



Small $|y|$ influence more strongly. Cannot handle $y = 0$.

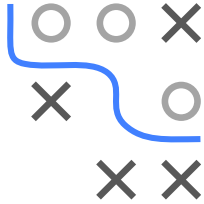


Similar measures:

- Mean Absolute Scaled Error (MASE)
- Symmetric Mean Absolute Percentage Error (sMAPE)

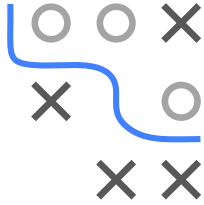
R^2

$$\rho_{R^2}(\mathbf{y}, \boldsymbol{F}) = 1 - \frac{\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{m}(y^{(i)} - \bar{y})^2} = 1 - \frac{SSE_{LinMod}}{SSE_{Intercept}}.$$

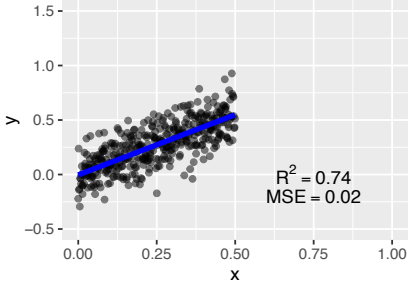


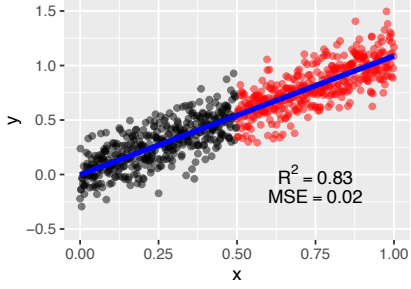
- Well-known classical measure for LMs – on train data.
- "Fraction of variance explained" by the model.
- How much SSE of constant baseline is reduced when we use more complex model?
- $\rho_{R^2} = 1$: all residuals are 0, we predict perfectly,
- $\rho_{R^2} = 0.9$: LM reduces SSE by factor of 10.
  $\rho_{R^2} = 0$: we predict as badly as the constant model.
- Is $\in [0, 1]$ on train data; as LM is always better than intercept.

R^2 VS MSE



- Better R^2 does not necessarily imply better fit.
- Data: $y = 1.1x + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.15)$.
- Fit half (black) and full data (black and red) with LM.

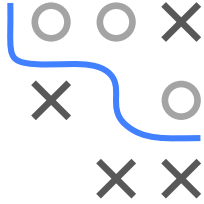




- Fit does not improve, but R^2 goes up.
- But: Invariant w.r.t. to linear scaling of y, MSE is not.

GENERALIZED R^2 FOR ML

$$1 - \frac{Loss_{ComplexModel}}{Loss_{SimplerModel}}.$$

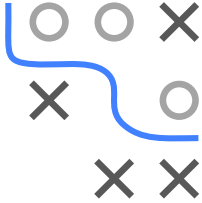


- E.g., model vs constant, LM vs non-linear model, tree vs forest, model with fewer features vs model with more, ...
- We could use arbitrary measures.
- In ML we would rather evaluate on test set.
- Can then become negative, e.g., for SSE and constant baseline, if our model fairs worse on the test set than a simple constant.

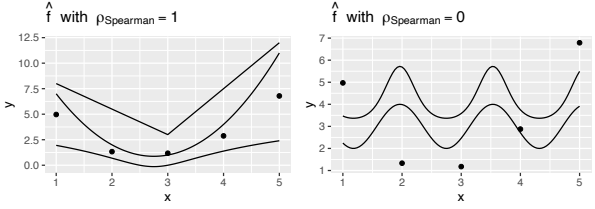
SPEARMAN'S ρ

Can be used if we care about the relative ranks of predictions:

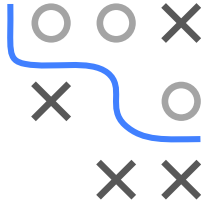
$$\rho_{\text{Spearman}}(\mathbf{y}, \boldsymbol{F}) = \frac{\text{Cov}(\text{rg}(\mathbf{y}), \text{rg}(\hat{\mathbf{y}}))}{\sqrt{\text{Var}(\text{rg}(\mathbf{y}))} \cdot \sqrt{\text{Var}(\text{rg}(\hat{\mathbf{y}}))}} \in [-1, 1],$$



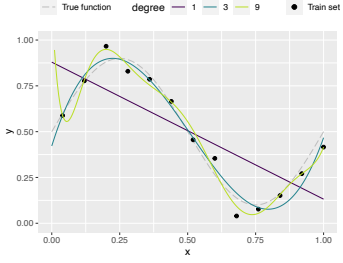
- Very robust against outliers
- A value of 1 or -1 means that $\hat{\mathbf{y}}$ and $\mathbf{y}$ have a perfect monotonic relationship.
- Invariant under monotone transformations of $\hat{\mathbf{y}}$



Introduction to Machine Learning



Evaluation
Training Error

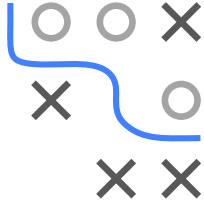


Learning goals

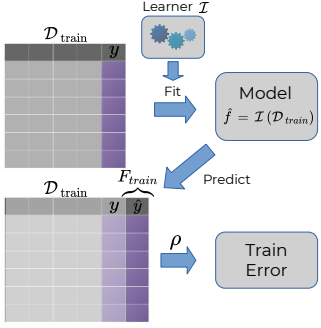
- Understand the definition of training error
- Understand why train error is unreliable for models of higher complexity when overfitting can occur

TRAINING ERROR

Simply plugin predictions for data that model has been trained on:



$$\rho(\mathbf{y}_{\text{train}}, \boldsymbol{F}_{\text{train}}) \text{ where } \boldsymbol{F}_{\text{train}} = \begin{bmatrix} \hat{f}_{\mathcal{D}_{\text{train}}}(\mathbf{x}^{(1)}_{\text{train}}) \\ \dots \\ \hat{f}_{\mathcal{D}_{\text{train}}}(\mathbf{x}^{(m)}_{\text{train}}) \end{bmatrix}$$

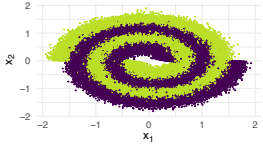


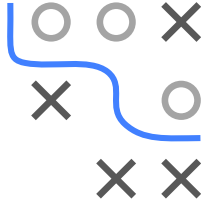
A.k.a. apparent error or resubstitution error.

EXAMPLE 1: KNN

For large data, and some models, train error **can maybe** yield a good approximation of the GE:

- Use k-NN ($k = 15$).
- Up to 30K points from `spirals` to train.
- Use very large extra set for testing (to measure "true GE").

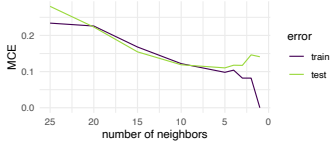


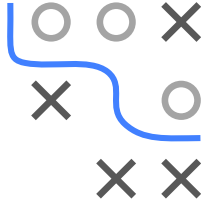


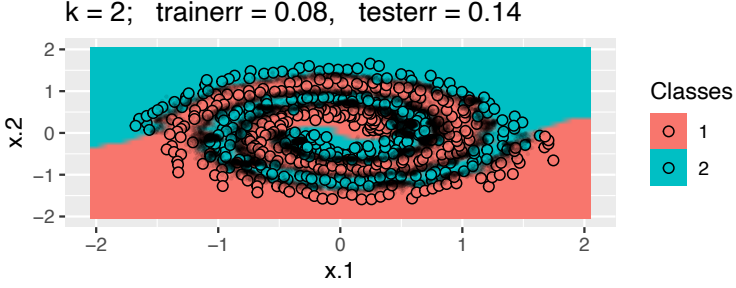
We increase train size, and see how gap between train error and GE closes.

EXAMPLE 1: KNN

- Fix train size to 500 and vary k.
- Low train error for small k is deceptive. Model is very local and overfits.



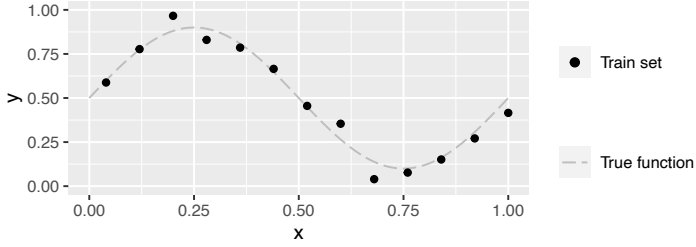


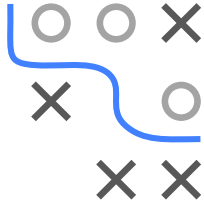


Black region are misclassifications from large test test.

EXAMPLE 2: POLYNOMIAL REGRESSION

Sample data from $0.5 + 0.4 \cdot \sin(2\pi x) + \epsilon$



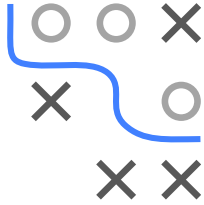


We fit a d^{th}-degree polynomial:

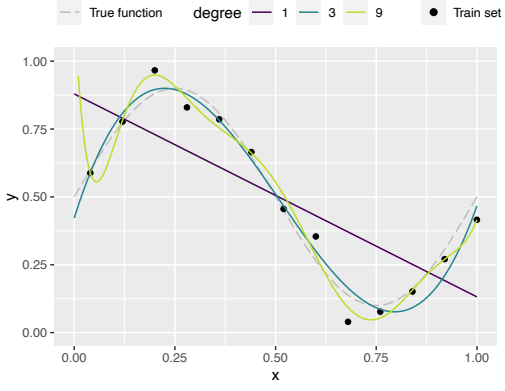
$$f(\mathbf{x} \mid \boldsymbol{\theta}) = \theta_0 + \theta_1 \mathbf{x} + \cdots + \theta_d \mathbf{x}^d = \sum_{j=0}^{d} \theta_j \mathbf{x}^j.$$

EXAMPLE 2: POLYNOMIAL REGRESSION

Simple model selection problem: Which d?



Visual inspection vs quantitative MSE on training set:

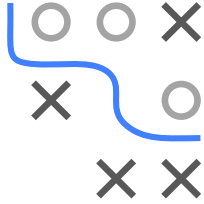


- $d = 1$: MSE = 0.036: clearly underfitting
- $d = 3$: MSE = 0.003: pretty OK
- $d = 9$: MSE = 0.001: clearly overfitting

Using the train error chooses overfitting model of maximal complexity.

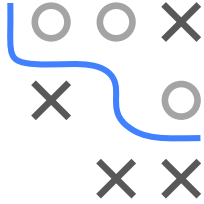
TRAIN ERROR CAN EASILY BECOME 0

- For 1-NN it is always 0 as each $\mathbf{x}^{(i)}$ is its own NN at test time.
- Extend any ML training in the following way: After normal fitting, we also store the training data. During prediction, we first check whether x is already stored in this set. If so, we replicate its label. The train error of such an (unreasonable) procedure will be 0.
- There are so called interpolators - interpolating splines, interpolating Gaussian processes - whose predictions can always perfectly match the regression targets, they are not necessarily good as they will interpolate noise, too.



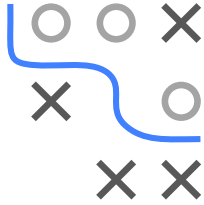
CLASSICAL STATISTICAL GOF MEASURES

- **Goodness-of-fit** measures like R^2, likelihood, AIC, BIC, deviance are all based on the training error.
- For models of restricted capacity, and enough data, and non-violated distributional assumptions: they might work.
- Hard to gauge when that breaks, for high-dim, more complex data.
- How do you compare to non-param ML-like models?

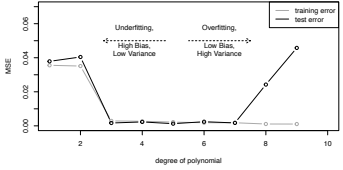


Out-of-sample testing is probably always a good idea!

Introduction to Machine Learning



Evaluation
Test Error



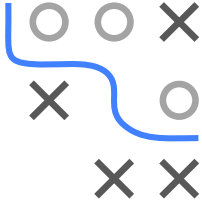
Learning goals

- Understand the definition of test error
- Understand that test error is more reliable than train error
- Bias-Variance analysis of holdout splitting

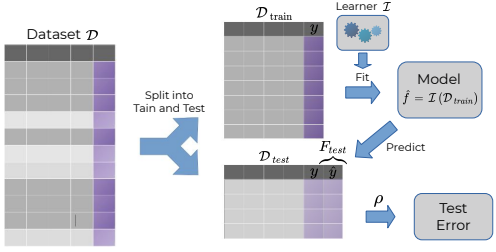
TEST ERROR AND HOLD-OUT SPLITTING

- Simulate prediction on unseen data, to avoid optimistic bias:

$$\rho(\mathbf{y}_{\text{test}}, \boldsymbol{F}_{\text{test}}) \text{ where } \boldsymbol{F}_{\text{test}} = \begin{bmatrix} \hat{f}_{\mathcal{D}_{\text{train}}}(\mathbf{x}^{(1)}_{\text{test}}) \\ \dots \\ \hat{f}_{\mathcal{D}_{\text{train}}}(\mathbf{x}^{(m)}_{\text{test}}) \end{bmatrix}$$



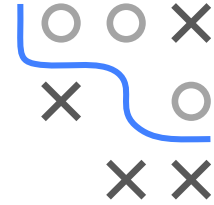
- Partition data, e.g., 2/3 for train and 1/3 for test.

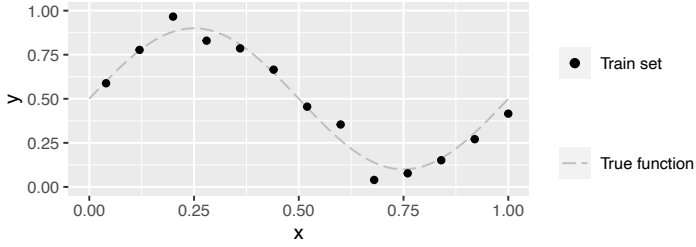


A.k.a. holdout splitting.

EXAMPLE: POLYNOMIAL REGRESSION

Previous example:

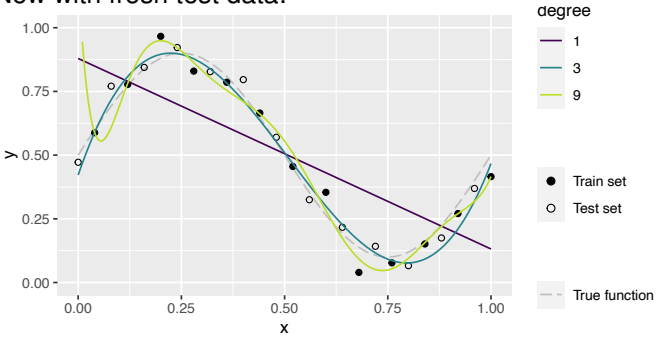


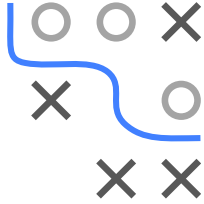


$$f(\mathbf{x} \mid \boldsymbol{\theta}) = \theta_0 + \theta_1 \mathbf{x} + \cdots + \theta_d \mathbf{x}^d = \sum_{j=0}^{d} \theta_j \mathbf{x}^j.$$

EXAMPLE: POLYNOMIAL REGRESSION

Now with fresh test data:



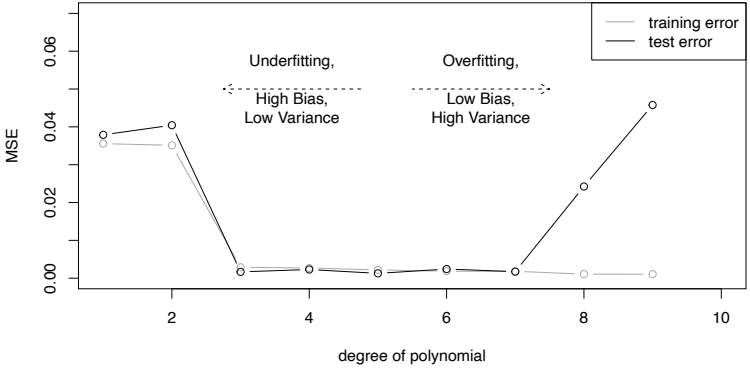


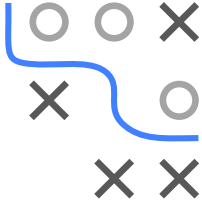
- $d = 1$: MSE = 0.038: clearly underfitting
- $d = 3$: MSE = 0.002: pretty OK
- $d = 9$: MSE = 0.046: clearly overfitting

While train error monotonically decreases in d, test error shows that high-d polynomials overfit.

TEST ERROR

Let's plot train and test MSE for all d:

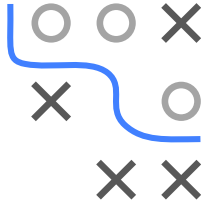




Increasing model complexity tends to cause

- a decrease in training error, and
- a U-shape in test error
  (first underfit, then overfit, sweet-spot in the middle).

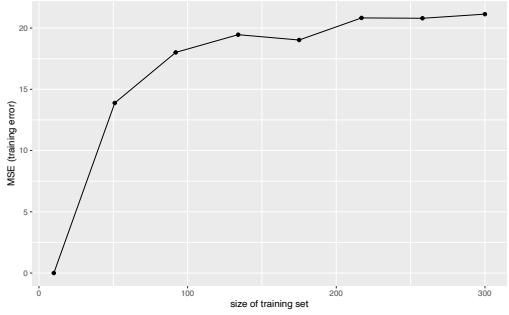
TRAINING VS. TEST ERROR



- Boston Housing data
- Polynomial regression (without interactions)

The training error...

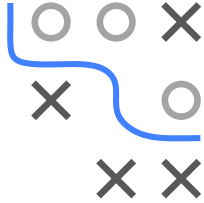
- decreases with smaller training set size as it becomes easier for the model to learn all observed patterns perfectly.

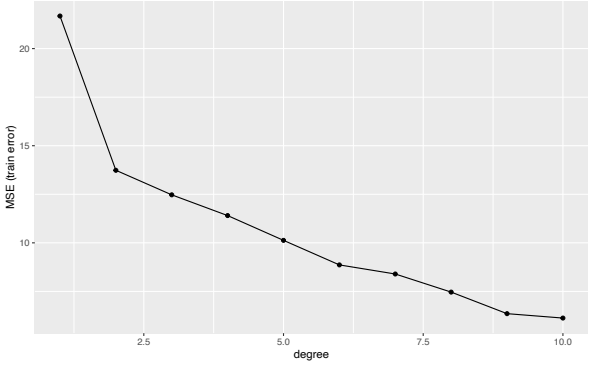


TRAINING VS. TEST ERROR

The training error...

- decreases with increasing model complexity as the model gets better at learning more complex structures.

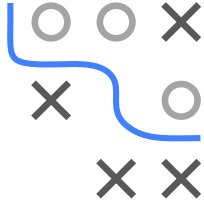


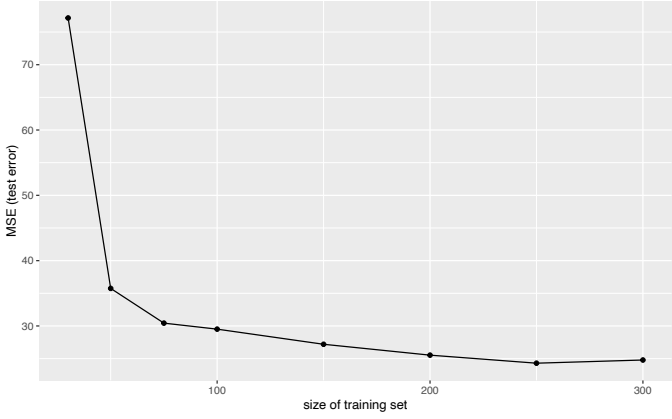


TRAINING VS. TEST ERROR

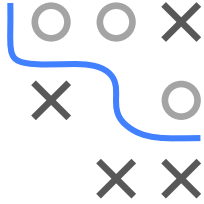
The test error...

- will typically decrease with larger training set size as the model generalizes better with more data to learn from.



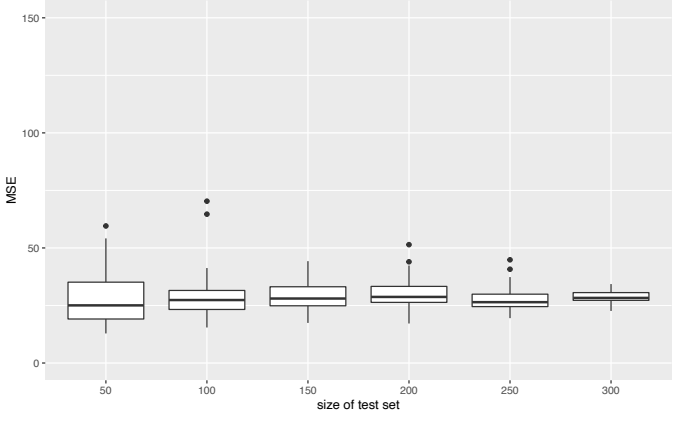


TRAINING VS. TEST ERROR

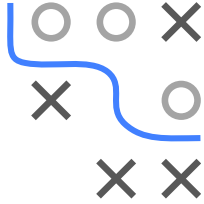


The test error...

- will have higher variance with smaller test set size.

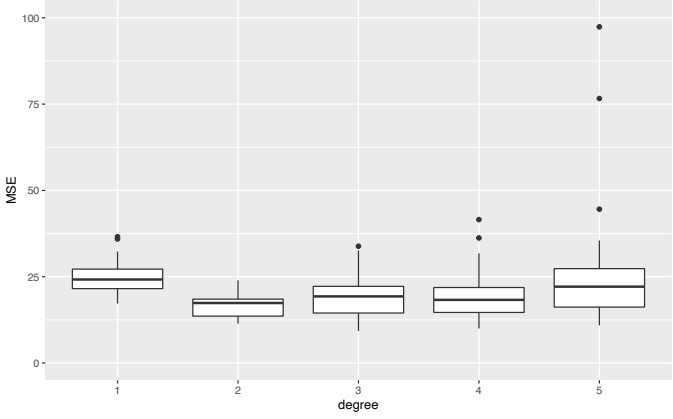


TRAINING VS. TEST ERROR

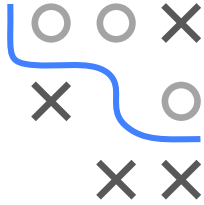


The test error...

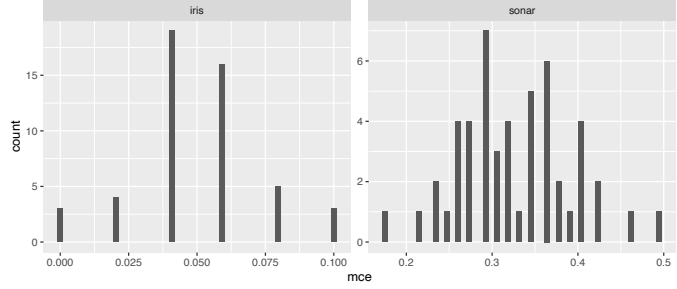
- will have higher variance with increasing model complexity.



BIAS AND VARIANCE



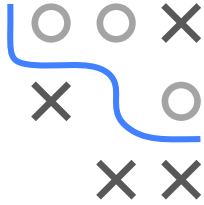
- Test error is a good estimator of GE, given a) we have enough data b) test data is representative i.i.d.
- Estimates for smaller test sets can fluctuate considerably – this is why we use resampling in such situations.
  Repeated $\frac{2}{3}$ / $\frac{1}{3}$ holdout splits:
  `iris` ($n = 150$) and `sonar` ($n = 208$).



BIAS-VARIANCE OF HOLD-OUT – EXPERIMENT

Hold-out sampling produces a trade-off between **bias** and **variance** that is controlled by split ratio.

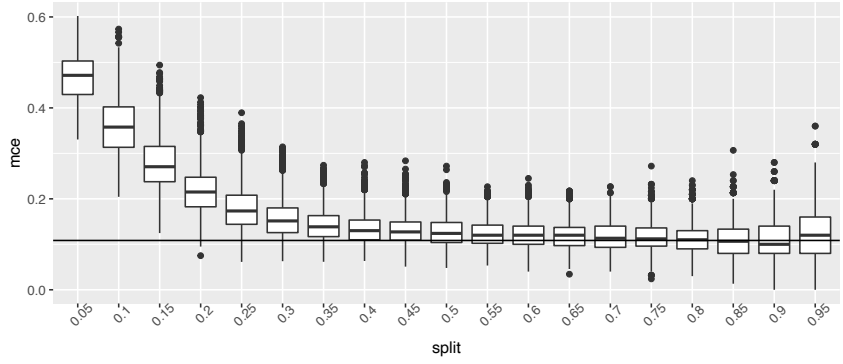
- Smaller training set $\rightarrow$ poor fit, pessimistic bias in $\widehat{\mathrm{GE}}$.
- Smaller test set $\rightarrow$ high variance.

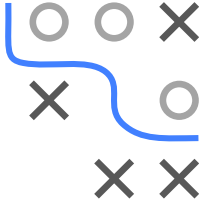


Experiment:

- `spirals` data ($sd = 0.1$), with CART tree.
- Goal: estimate real performance of a model with $|\mathcal{D}_{\text{train}}| = 500$.
- Split rates $s \in \{0.05, 0.10, ..., 0.95\}$ with $|\mathcal{D}_{\text{train}}| = s \cdot 500$.
- Estimate error on $\mathcal{D}_{\text{test}}$ with $|\mathcal{D}_{\text{test}}| = (1 - s) \cdot 500$.
- 50 repeats for each split rate.
- Get "true" performance by often sampling 500 points, fit learner, then eval on 10^5 fresh points.

BIAS-VARIANCE OF HOLD-OUT – EXPERIMENT

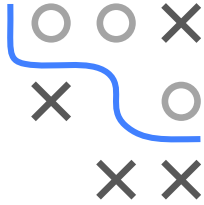


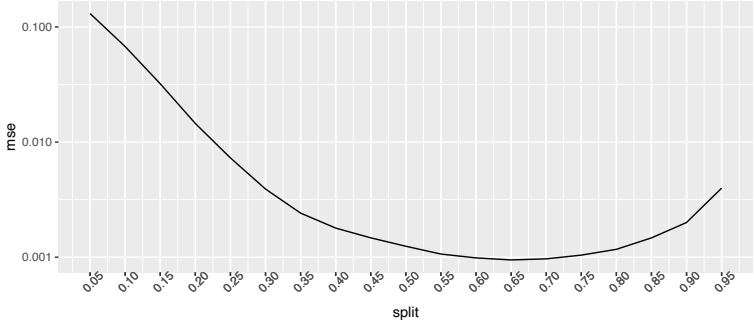


- Clear pessimistic bias for small training sets – we learn a much worse model than with 500 observations.
- But increase in variance when test sets become smaller.

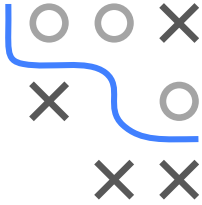
BIAS-VARIANCE OF HOLD-OUT – EXPERIMENT

- Let's now plot the MSE of the holdout estimator.
- NB: Not MSE of model, but squared difference between estimated holdout values and true performance (horiz. line in prev. plot).
- Best estimator is ca. train set ratio of 2/3.
- NB: This is a single experiment and not a scientific study, but this rule-of-thumb has also been validated in larger studies.

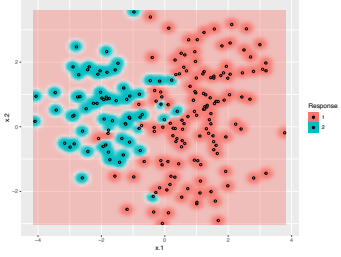




Introduction to Machine Learning



Evaluation
Overfitting and Underfitting

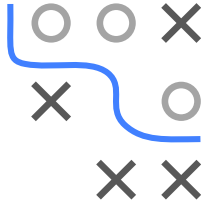


Learning goals

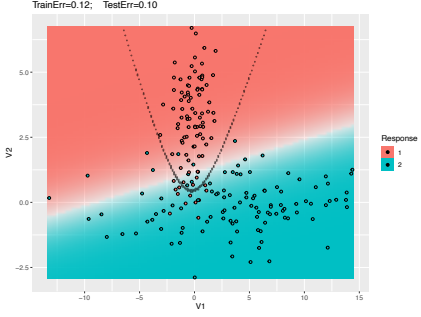
- Understand definitions of overfitting and underfitting

UNDERFITTING

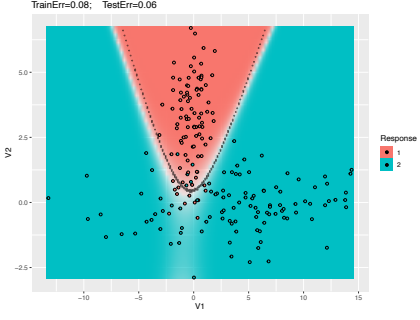
- Occurs if model does not reflect true shape of underlying function
- Hence, predictions will be less good as they could be
- High train error and high test error
- Hard to detect, as we don't know what the Bayes error is for a task



Underfitted model

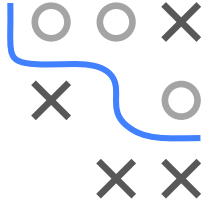


Appropriate model

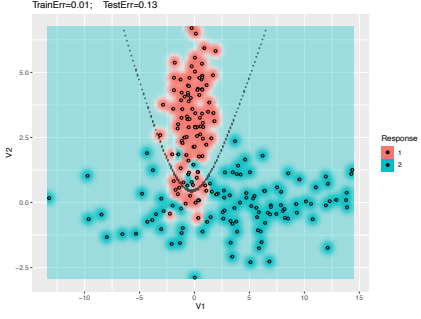


OVERFITTING

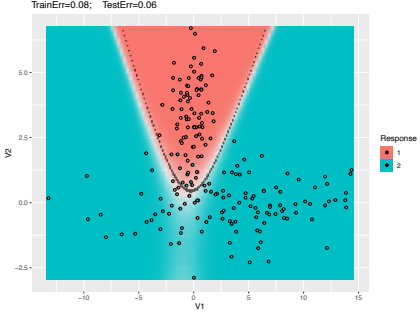
- Overfitting occurs when the model reflects noise or artifacts in training data, which do not generalize
- Small train error, at cost of test high error
- Hence, predictions of overfitting models cannot be trusted - but proper ML evaluation workflows should make it visible



Overfitted model

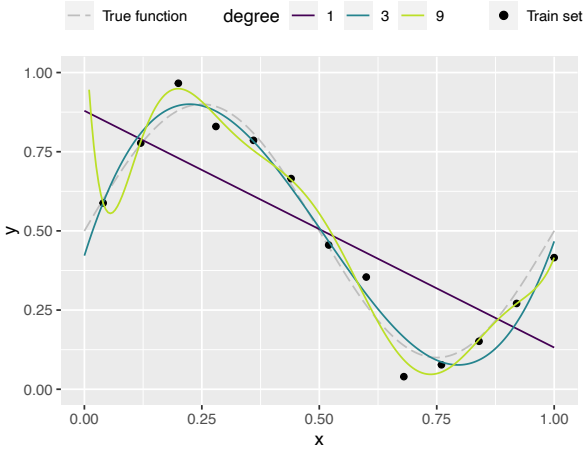


Appropriate model

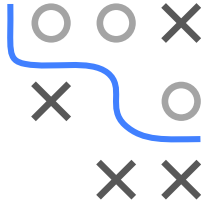


UNDER- AND OVERFITTING IN REGRESSION

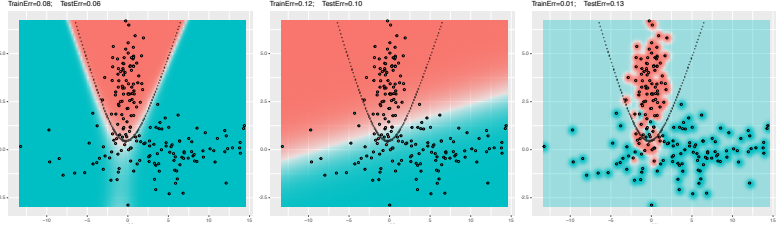
- Poly-Regression, on data from sinusoidal function
- LM underfits, high-d overfits



MATHEMATICAL DEFINITIONS



- Nearly no reference does that, here is one approach
- Underfitting $UF(\hat{f}, L) := \mathrm{GE}\left(\hat{f}, L\right) - \mathrm{GE}(f^*, L)$
  Diff in GE between $\hat{f}$ and the Bayes optimal model
- Overfitting $OF(\hat{f}, L) := \mathrm{GE}\left(\hat{f}, L\right) - \mathcal{R}_{\text{emp}}(\hat{f}, L)$
  Diff between (theoretical) GE and training error

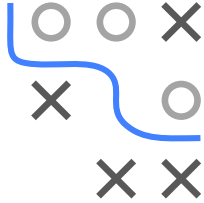


NB: Now, RHS is both UF and OF, let's say OF has "prio".

OVERFITTING TRADE-OFFS

The potential for overfitting is influenced by:

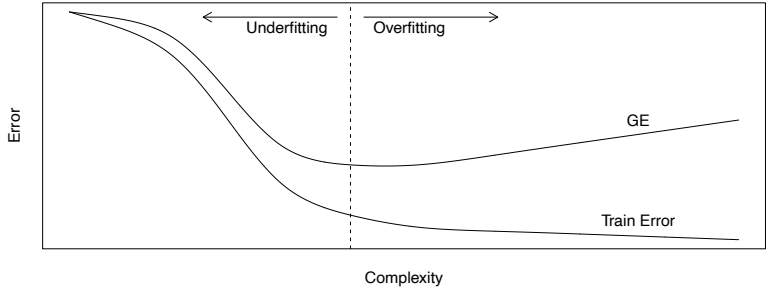
- Complexity of hypothesis space
- Amount of training data
- Dimensionality of feature space
- Irreducible noise

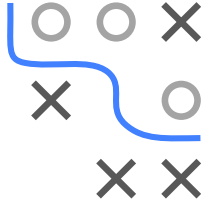


Implications:

- The larger / more complex is $\mathcal{H}$, the more data we need to tell candidate models apart
- The less data we have, the more we need to stick with "constrained" $\mathcal{H}$
- OF can happen for LMs too: If feature space is very high-dim
- Tightly connected to the bias-var-noise decomposition of GE of a learner ($\rightarrow$ which we study elsewhere).

COMPLEXITY VS GE

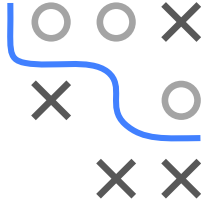




- Common U-shape of GE if complexity or train-rounds go up.
- Optimal level of complexity:
  Simplest model for which GE is not significantly outperformed
- We could also call "Point of OF" the point where GE goes up.

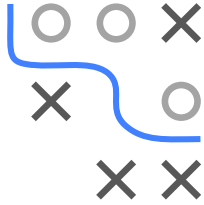
AVOIDING OVERFITTING

- Use more or better data – not always possible, but maybe can augment data, e.g., for images
- Constrain $\mathcal{H}$ directly by using less complex model classes
- Many learners come with HPs that can constrain complexity
- Use "early-stopping"
- Occam's razor in model selection: If GE not strongly reduced for more complex class, use the simpler model.

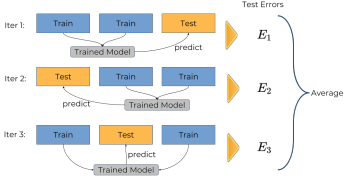


All of the above are methods of regularization, which we study in a dedicated chapter.

Introduction to Machine Learning



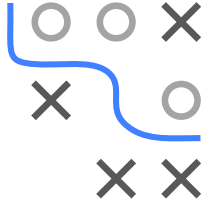
Evaluation
Resampling 1



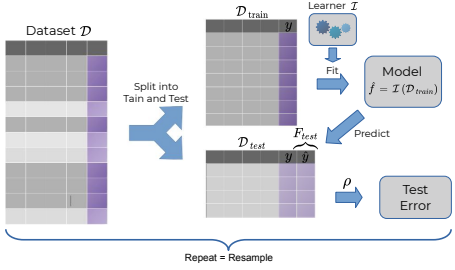
Learning goals

- Understand how resampling techniques extend the idea of simple train-test splits
- Understand the ideas of cross-validation, bootstrap and subsampling

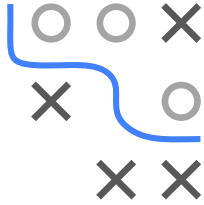
RESAMPLING



- **Goal**: estimate $\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n, \rho_L) = \mathbf{E}\left[L(y, \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})(\mathbf{x}))\right]$.
- Holdout: Small trainset = high pessimistic bias; small testset = high var.
- Resampling: Repeatedly split in train and test, then average results.
- Allows to have large trainsets large (low pessimistic bias) since we use $\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho)$ as a proxy for $\mathrm{GE}(\mathcal{I}, \boldsymbol{\lambda}, n, \rho)$)
- And reduce var from small testsets via averaging over repetitions.



RESAMPLING STRATEGIES



- Represent train and test sets by index vectors::
  $J_{\text{train}} \in \{1, \ldots, n\}^{n_{\text{train}}}$ and $J_{\text{test}} \in \{1, \ldots, n\}^{n_{\text{test}}}$
- Resampling strategy = collection of splits:

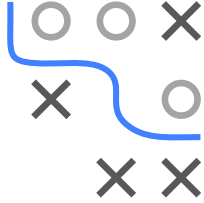
$$\mathcal{J} = \left((J_{\text{train},1}, J_{\text{test},1}), \ldots, (J_{\text{train},B}, J_{\text{test},B}) \right).$$

- Resampling estimator:

$$\widehat{\text{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda}) = \text{agr}\Big(\rho\Big(\mathbf{y}_{J_{\text{test},1}}, \boldsymbol{F}_{J_{\text{test},1}, \mathcal{I}(\mathcal{D}_{\text{train},1}, \boldsymbol{\lambda})} \Big), \\ \vdots \\ \rho\Big(\mathbf{y}_{J_{\text{test},B}}, \boldsymbol{F}_{J_{\text{test},B}, \mathcal{I}(\mathcal{D}_{\text{train},B}, \boldsymbol{\lambda})} \Big) \Big),$$

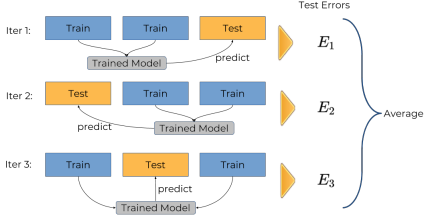
- Aggregation agr is typically "mean" and $n_{\text{train}} \approx n_{\text{train},1} \approx \cdots \approx n_{\text{train},B}$.

CROSS-VALIDATION



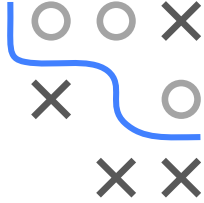
- Split the data into k roughly equally-sized partitions.
- Each part is test set once, join $k-1$ parts for training.
- Obtain k test errors and average.
- Fraction $(k-1)/k$ is used for training, so 90% for 10CV
- Each observation is tested exactly once.

Example: 3-fold CV

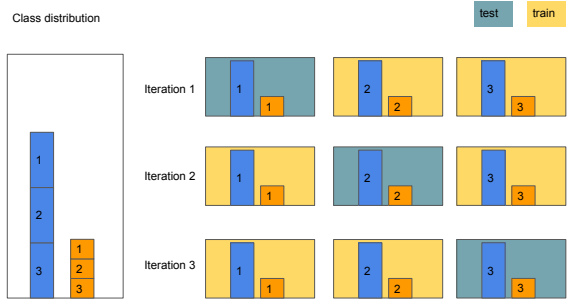


CROSS-VALIDATION - STRATIFICATION

- Used when target classes are very imbalanced
- Then small classes can randomly get very small in samples
- Preserve distrib of target (or any feature) in each fold
- For classes: simply CV-split the class data, then join

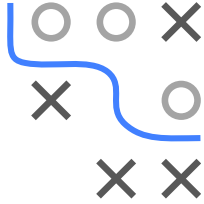


Example: stratified 3-fold cross-validation

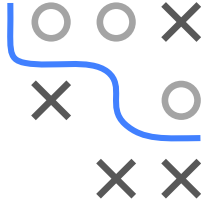


CROSS-VALIDATION

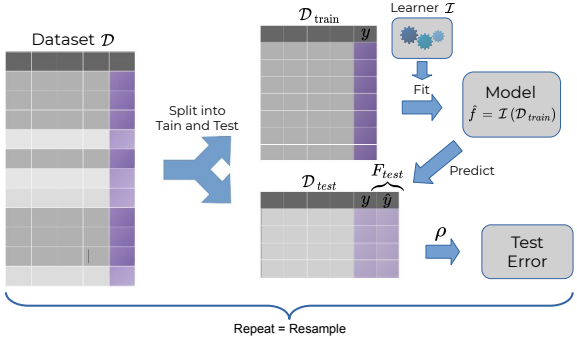
- 5 or 10 folds are common.
- $k = n$ is known as "leave-one-out" CV (LOO-CV)
- Bias of $\widehat{\mathrm{GE}}$: The more folds, the smaller. LOO nearly unbiased.
- LOO has high var, better many folds for small data but not LOO
- Repeated CV (avg over high-fold CVs) good for for small data.



SUBSAMPLING

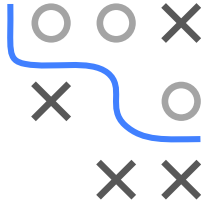


- Repeated hold-out with averaging, a.k.a. Monte Carlo CV.
- Typical choices for splitting: $\frac{4}{5}$ or $\frac{9}{10}$ for training.

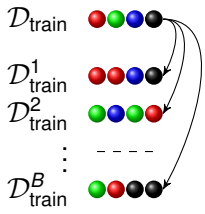


- Smaller subsampling rate = larger pessimistic bias
- More reps = smaller var

BOOTSTRAP

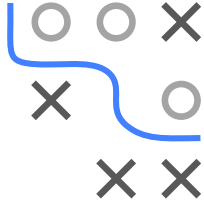


- Draw B trainsets of size n with replacement from orig $\mathcal{D}$
- Testsets = Out-Of-Bag points: $\mathcal{D}^b_{\text{test}} = \mathcal{D} \setminus \mathcal{D}^b_{\text{train}}$

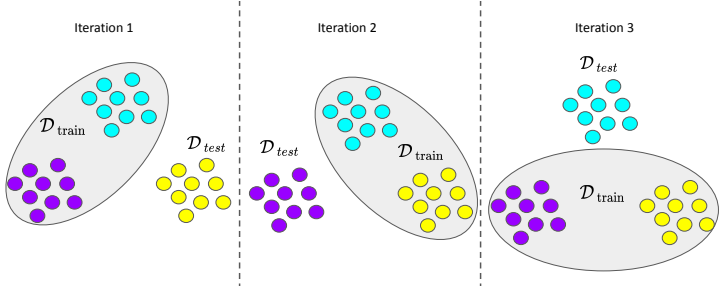


- Similar analysis as for subsampling
- Trainsets contain about 2/3 unique points:
  $1 - \mathbb{P}((\mathbf{x}, y) \notin \mathcal{D}_{\text{train}}) = 1 - \left(1 - \frac{1}{n}\right)^n \overset{n \to \infty}{\longrightarrow} 1 - \frac{1}{e} \approx 63.2\%$
- Replicated train points can lead to problems and artifacts
- Extensions B632 and B632+ also use trainerr for better estimate when data very small

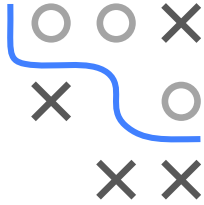
LEAVE-ONE-OBJECT-OUT



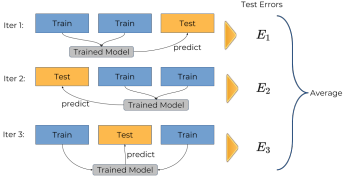
- Used when we have multiple obs from same objects, e.g., persons or hospitals or base images
- Data not i.i.d. any more
- Data from same object should **either** be in train **or** testset
- Otherwise we likely bias $\widehat{\mathrm{GE}}$
- CV on objects, or leave-one-object-out



Introduction to Machine Learning



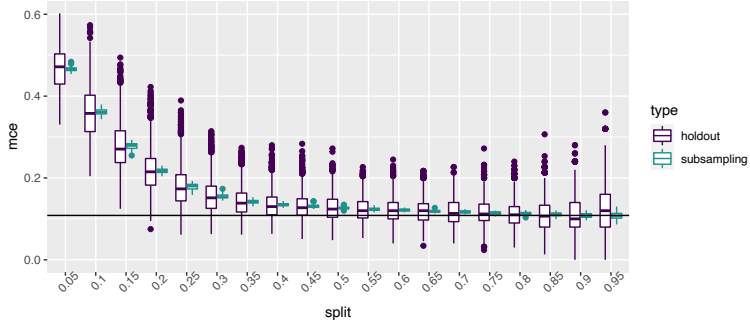
Evaluation
Resampling 2

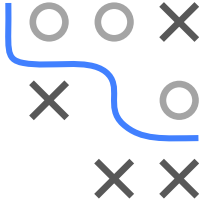


Learning goals

- Understand why resampling is better estimator than hold-out
- In-depth bias-var analysis of resampling estimator
- Understand that CV does not produce independent samples
- Short guideline for practical use

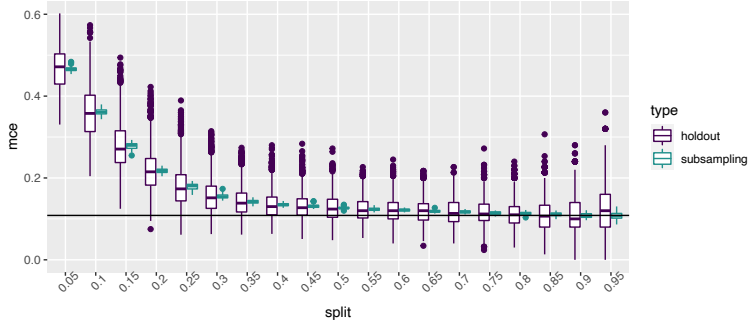
BIAS-VARIANCE ANALYSIS FOR SUBSAMPLING

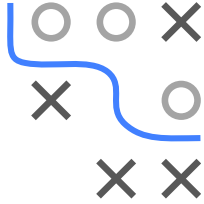




- Reconsider bias-var experiment for holdout (maybe re-read)
- Split rates $s \in \{0.05, 0.1, ..., 0.95\}$ with $|\mathcal{D}_{\text{train}}| = s \cdot 500$.
- Holdout vs. subsampling with 50 iters
- 50 replications

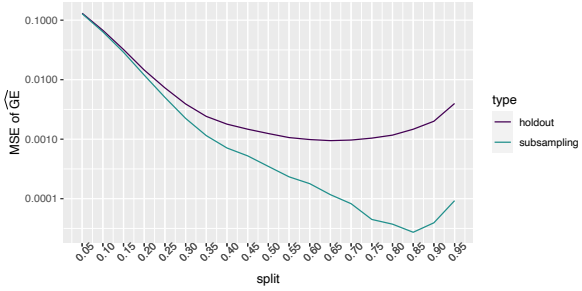
BIAS-VARIANCE ANALYSIS FOR SUBSAMPLING

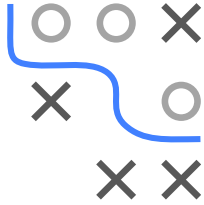




- Both estimators are compared to "real" MCE (black line)
- SS same pessimistic bias as holdout for given s, but much less var

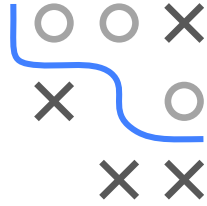
BIAS-VARIANCE ANALYSIS FOR SUBSAMPLING





- MSE of $\widehat{\mathrm{GE}}$ strictly better for SS
- Smaller var of SS enables to use larger s for optimal choice
- The optimal split rate now is a higher $s \approx 0.8$.
- Beyond $s = 0.8$: MSE goes up because var doesn't go down as much as we want due to increasing overlap in trainsets (see later)

DEDICATED TESTSET SCENARIO - ANALYSIS

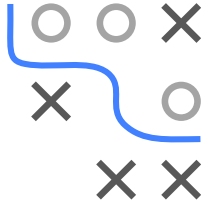


- Goal: estimate $\mathrm{GE}\left(\hat{f}\right) = \mathbb{E}\left[L\left(y, \hat{f}(\mathbf{x})\right)\right]$ via

$$\widehat{\mathrm{GE}}\left(\hat{f}\right) = \frac{1}{m} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{test}}} L\left(y, \hat{f}(\mathbf{x})\right)$$

  Here, only $(\mathbf{x}, y)$ are random, they are m i.i.d. fresh test samples
- This is: average over i.i.d $L(y, \hat{f}(\mathbf{x}))$, so directly know $\mathbb{E}$ and var. And can use CLT to approx distrib of $\widehat{\mathrm{GE}}\left(\hat{f}\right)$ with Gaussian.
- $\mathbb{E}[\widehat{\mathrm{GE}}\left(\hat{f}\right)] = \mathbb{E}[L\left(y, \hat{f}(\mathbf{x})\right)] = \mathrm{GE}\left(\hat{f}\right)$
- $\mathbb{V}[\widehat{\mathrm{GE}}\left(\hat{f}\right)] = \frac{1}{m}\mathbb{V}[L\left(y, \hat{f}(\mathbf{x})\right)]$
- So $\widehat{\mathrm{GE}}\left(\hat{f}\right)$ is unbiased estimator of $\mathrm{GE}\left(\hat{f}\right)$, var decreases linearly in testset size, have an approx of full distrib (can do NHST, CIs, etc.)
- NB: Gaussian may work less well for e.g. 0-1 loss, with $\mathbb{E}$ close to 0, can use binomial or other special approaches for other losses

PESSIMISTIC BIAS IN RESAMPLING



- Estim $\mathrm{GE}(\mathcal{I}, n)$ (surrogate for $\mathrm{GE}\left(\hat{f}\right)$ when $\hat{f}$ is fit on full $\mathcal{D}$, with $|\mathcal{D}| = n$) via resampling based estim $\widehat{\mathrm{GE}}(\mathcal{I}, n_{\text{train}})$

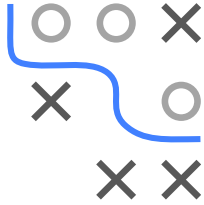
$$\widehat{\mathrm{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda}) = \mathrm{agr}\Big(\rho\Big(\mathbf{y}_{J_{\text{test},1}}, \boldsymbol{F}_{J_{\text{test},1}, \mathcal{I}(\mathcal{D}_{\text{train},1}, \boldsymbol{\lambda})}\Big), \\ \vdots \\ \rho\Big(\mathbf{y}_{J_{\text{test},B}}, \boldsymbol{F}_{J_{\text{test},B}, \mathcal{I}(\mathcal{D}_{\text{train},B}, \boldsymbol{\lambda})}\Big)\Big),$$

- Let's assume agr is avg and ρ is loss-based, so ρ_L
- The ρ are simple holdout estims. So:

$$\mathbf{E}[\widehat{\mathrm{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda})] \approx \mathbf{E}[\rho\Big(\mathbf{y}_{J_{\text{test}}}, \boldsymbol{F}_{J_{\text{test}}, \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})}\Big)]$$

- NB1: In above, as always for $\mathrm{GE}(\mathcal{I})$, both $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ (and so $\mathbf{x} \in \mathcal{D}_{\text{test}}$) are random vars, and we take E over them
- NB2: Need $\approx$ as maybe not all train/test sets in resampling of exactly same size

PESSIMISTIC BIAS IN RESAMPLING



$$\mathbf{E}[\widehat{\mathrm{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda})] \approx \mathbf{E}[\rho\Big(\mathbf{y}_{J_{\text{test}}}, \boldsymbol{F}_{J_{\text{test}}, \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})}\Big)] =$$

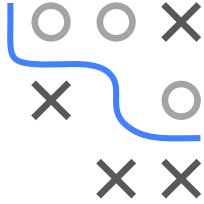
$$\mathbf{E}\left[\frac{1}{m} \sum_{(\mathbf{x},y) \in \mathcal{D}_{\text{test}}} L(y, \mathcal{I}(\mathcal{D}_{\text{train}})(\mathbf{x}))\right] = \mathrm{GE}(\mathcal{I}, n_{\text{train}})$$

$\Rightarrow$

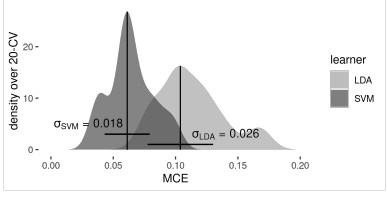
- So when we use $\widehat{\mathrm{GE}}(\mathcal{I}, \mathcal{J}, \rho, \boldsymbol{\lambda})$ to to estimate $\mathrm{GE}(\mathcal{I}, n)$, our expected value is nearly correct, it's $\mathrm{GE}(\mathcal{I}, n_{\text{train}})$
- But fitting $\mathcal{I}$ on less data (n_{train} vs full n) usually results in model with worse perf, hence estimator is pessimistically biased
- Bias the stronger, the smaller our training splits in resampling.

NO INDEPENDENCE OF CV RESULTS

- Similar analysis as before holds for CV
- Might be tempted to report distribution or SD of individual CV split perf values, e.g. to test if perf of 2 learners is significantly different
- But k CV splits are not independent



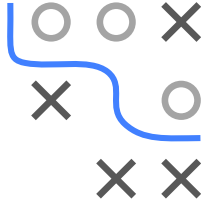
A t-test on the difference of the mean GE estimators yields a highly significant p-value of $\approx 7.9 \cdot 10^{-5}$ on the 95% level.



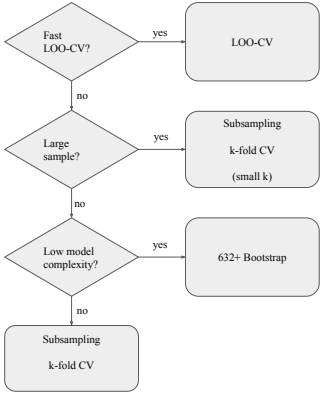
LDA vs SVM on `spam` classification problem, performance estimation via 20-CV w.r.t. MCE.

NO INDEPENDENCE OF CV RESULTS

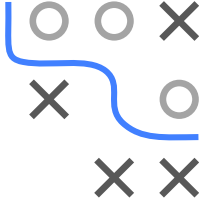
- $\mathbb{V}[\widehat{\mathrm{GE}}]$ of CV is a difficult combination of
  - average variance as we estim on finite trainsets
  - covar from test errors, as models result from overlapping trainsets
  - covar due to the dependence of trainsets and test obs appear in trainsets
- Naively using the empirical var of k individual $\widehat{\mathrm{GE}}$s (as on slide before) yields biased estimator of $\mathbb{V}[\widehat{\mathrm{GE}}]$. Usually this underestimates the true var!
- Worse: there is no unbiased estimator of $\mathbb{V}[\widehat{\mathrm{GE}}]$ [Bengio, 2004]
- Take into account when comparing learners by NHST
- Somewhat difficult topic, we leave it with the warning here



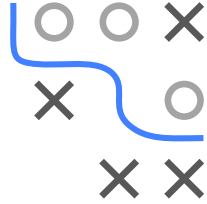
SHORT GUIDELINE



- 5-CV or 10-CV have become standard.
- Do not use hold-out, CV with few folds, or SS with small split rate for small n. Can bias estim and have large var.
- For small n, e.g. $n < 200$, use LOO or, probably better, repeated CV.
- For some models, fast tricks for LOO exist
- With $n = 100.000$, can have "hidden" small-sample size, e.g. one class very small
- SS usually better than bootstrapping. Repeated obs can cause problems in training, especially in nested setups where the “training” set is split up again.



Introduction to Machine Learning



Evaluation
Simple Measures for Classification

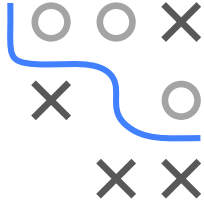
		True Class y	
		$+$	$-$
Pred. $\hat{y}$	$+$	True Positive (TP)	False Positive (FP)
	$-$	False Negative (FN)	True Negative (TN)

Learning goals

- Know the definitions of misclassification error rate (MCE) and accuracy (ACC)
- Understand the entries of a confusion matrix
- Understand the idea of costs
- Know definitions of Brier score and log loss

LABELS VS PROBABILITIES

In classification we predict:



1. Class labels:

$$\boldsymbol{F} = \left(\hat{o}_k^{(i)}\right)_{i \in \{1,\ldots,m\}, k \in \{1,\ldots,g\}} \in \mathbb{R}^{m \times g},$$

   where $\hat{o}_k^{(i)} = [\hat{y}^{(i)} = k]$, $k = 1, \ldots, g$ is the one-hot-encoded class label prediction.

2. Class probabilities:

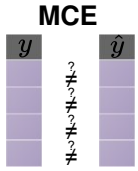
$$\boldsymbol{F} = \left(\hat{\pi}_k^{(i)}\right)_{i \in \{1,\ldots,m\}, k \in \{1,\ldots,g\}} \in [0,1]^{m \times g}$$

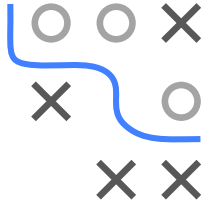
$\rightarrow$ These form the basis for evaluation.

LABELS: MCE & ACC

The **misclassification error rate (MCE)** counts the number of incorrect predictions and presents them as a rate:

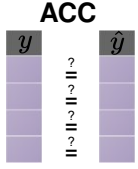
$$\rho_{MCE} = \frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \neq \hat{y}^{(i)}] \in [0, 1].$$





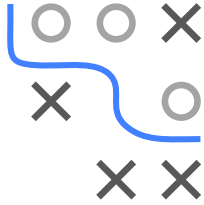
Accuracy (ACC) is defined in a similar fashion for correct classifications:

$$\rho_{ACC} = \frac{1}{m} \sum_{i=1}^{m} [y^{(i)} = \hat{y}^{(i)}] \in [0, 1].$$



- If the data set is small this can be brittle.
- MCE says nothing about how good/skewed predicted probabilities are.
- Errors on all classes are weighted equally, which is often inappropriate.

LABELS: CONFUSION MATRIX



Much better than reducing prediction errors to a simple number is tabulating them in a **confusion matrix**:

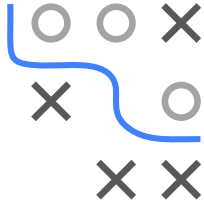
- true classes in columns,
- predicted classes in rows.

We can nicely see class sizes (predicted/true) and where errors occur.

True classes

Predicted classes		**setosa**	**versicolor**	**virginica**	**error**	*n*
	setosa	50	0	0	0	50
	versicolor	0	46	4	4	50
	virginica	0	4	46	4	50
	error	0	4	4	8	-
	n	50	50	50	-	150

LABELS: CONFUSION MATRIX



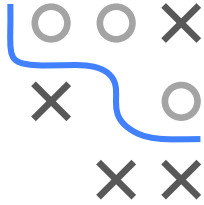
- In binary classification, we typically call one class "positive" and the other "negative".
- The positive class is the more important, often smaller one.

		True Class y	
		$+$	$-$
Pred.	$+$	True Positive (TP)	False Positive (FP)
$\hat{y}$	$-$	False Negative (FN)	True Negative (TN)

e.g.,

- **True Positive** (TP) means that an instance is classified as positive that is really positive (correct prediction).
- **False Negative** (FN) means that an instance is classified as negative that is actually positive (incorrect prediction).

LABELS: COSTS



We can also assign different costs to different errors via a **cost matrix**.

$$Costs = \frac{1}{n} \sum_{i=1}^{n} C[y^{(i)}, \hat{y}^{(i)}]$$

Example: Depending on certain features (age, income, profession, ...) a bank wants to decide whether to grant a 10,000 EUR loan.

Predict if a person is solvent (yes / no).
Should the bank lend them the money?

Examplary costs:
Loss in event of default: 10,000 EUR
Income through interest paid: 100 EUR

		True classes	
		solvent	**not solvent**
Predicted classes	**solvent**	0	10,000
	not solvent	100	0

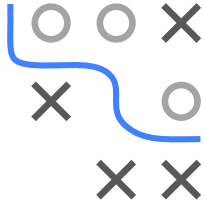
LABELS: COSTS

Cost matrix

		True classes	
		solvent	**not solvent**
Predicted classes	**solvent**	0	10,000
	not solvent	100	0

Confusion matrix

		True classes	
		solvent	**not solvent**
Predicted classes	**solvent**	70	3
	not solvent	7	20



- If the bank gives everyone a credit, who was predicted as *solvent*, the costs are at:

$$\begin{aligned} \textit{Costs} &= \frac{1}{n} \sum_{i=1}^{n} C[y^{(i)}, \hat{y}^{(i)}] \\ &= \frac{1}{100} \left(100 \cdot 7 + 0 \cdot 70 + 10.000 \cdot 3 + 0 \cdot 20\right) = 307 \end{aligned}$$

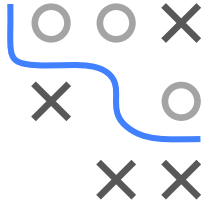
- If the bank gives everyone a credit, the costs are at:

$$\textit{Costs} = \frac{1}{100} \left(100 \cdot 0 + 0 \cdot 77 + 10.000 \cdot 23 + 0 \cdot 0\right) = 2.300$$

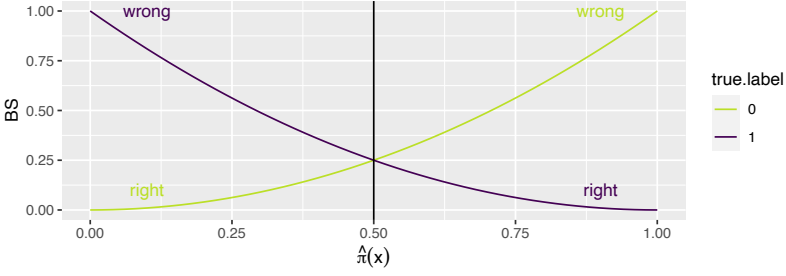
PROBABILITIES: BRIER SCORE

Measures squared distances of probabilities from the true class labels:

$$\rho_{BS} = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{\pi}^{(i)} - y^{(i)} \right)^2$$

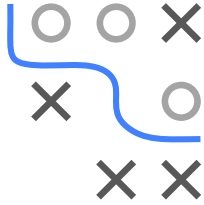


- Fancy name for MSE on probabilities.
- Usual definition for binary case; $y^{(i)}$ must be encoded as 0 and 1.



PROBABILITIES: BRIER SCORE

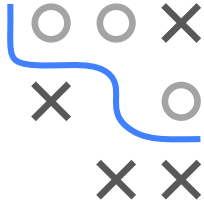
$$\rho_{BS,MC} = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{g} \left(\hat{\pi}_k^{(i)} - o_k^{(i)} \right)^2$$



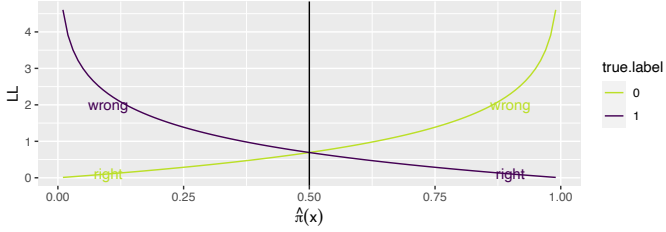
- Original by Brier, works also for multiple classes.
- $o_k^{(i)} = [y^{(i)} = k]$ marks the one-hot-encoded class label.
- For the binary case, $\rho_{BS,MC}$ is twice as large as ρ_{BS}: in $\rho_{BS,MC}$, we sum the squared difference for each observation regarding both class 0 **and** class 1, not only the true class.

PROBABILITIES: LOG-LOSS

Logistic regression loss function, a.k.a. Bernoulli or binomial loss, $y^{(i)}$ encoded as 0 and 1.

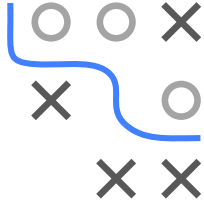


$$\rho_{LL} = \frac{1}{m} \sum_{i=1}^{m} \left(-y^{(i)} \log \left(\hat{\pi}^{(i)} \right) - \left(1 - y^{(i)} \right) \log \left(1 - \hat{\pi}^{(i)} \right) \right).$$



- Optimal value is 0, “confidently wrong” is penalized heavily.
- Multi-class version: $\rho_{LL,MC} = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{g} o_k^{(i)} \log \left(\hat{\pi}_k^{(i)} \right)$.

Introduction to Machine Learning



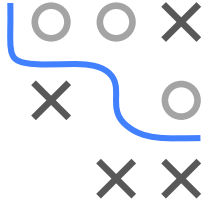
Evaluation
ROC Basics

		True Class y		
		$+$	$-$	
Pred. $\hat{y}$	$+$	TP	FP	PPV = $\frac{TP}{TP+FP}$
	$-$	FN	TN	NPV = $\frac{TN}{FN+TN}$
		TPR = $\frac{TP}{TP+FN}$	TNR = $\frac{TN}{FP+TN}$	Accuracy = $\frac{TP+TN}{TOTAL}$

Learning goals

- Understand why accuracy is not an optimal performance measure for imbalanced labels
- Understand the different measures computable from a confusion matrix
- Be aware that each of these measures has a variety of names

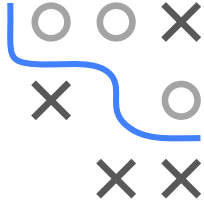
CLASS IMBALANCE

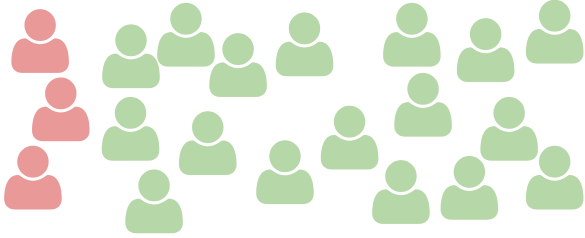


- Assume a binary classifier diagnoses a serious medical condition.
- Label distribution is often **imbalanced**, i.e, not many people have the disease.
- Evaluating on mce is often inappropriate for scenarios with imbalanced labels:
  - Assume that only 0.5 % have the disease.
  - Always predicting “no disease” has an mce of 0.5 %, corresponding to very high accuracy.
  - This sends all sick patients home $\rightarrow$ bad system
- This problem is known as the **accuracy paradox**.

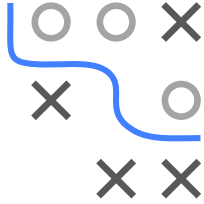
CLASS IMBALANCE

Classifying all observations as “no disease” (green) yields top accuracy simply because the “disease” occurs so rarely $\rightarrow$ accuracy paradox.





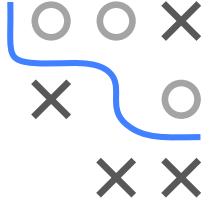
IMBALANCED COSTS



- Another point of view is **imbalanced costs**.
- In our example, classifying a sick patient as healthy should incur a much higher cost than classifying a healthy patient as sick.
- The costs depend a lot on what happens next: we can well assume that our system is some type of screening filter, and often the next step after labeling someone as sick might be a more invasive, expensive, but also more reliable test for the disease.
- Erroneously subjecting someone to this step is undesirable (psychological, economic, medical expense), but sending someone home to get worse or die seems much more so.
- Such situations not only arise under label imbalance, but also when costs differ (even though classes might be balanced).
- We could see this as imbalanced costs of misclassification, rather than imbalanced labels; both situations are tightly connected.

IMBALANCED COSTS

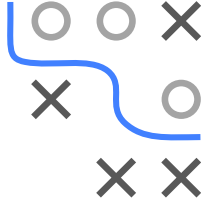
Imbalanced costs: classifying incorrectly as "no disease" incurs very high cost.



- Problem: if we were able to specify costs precisely, we could evaluate or even optimize on them.
- This important subfield of ML is called **cost-sensitive learning**, which we will not cover in this lecture unit.
- Unfortunately, users find it notoriously hard to come up with precise cost figures in imbalanced scenarios.
- Evaluating "from different perspectives", with multiple metrics, often helps to get a first impression of system quality.

ROC ANALYSIS

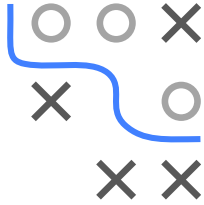
- **ROC analysis** is a subfield of ML which studies the evaluation of binary prediction systems.
- ROC stands for “receiver operating characteristics” and was initially developed by electrical engineers and radar engineers during World War II for detecting enemy objects in battlefields – still has the funny name.



http://media.iwm.org.uk/iwm/mediaLib//39/media-39665/large.jpg

LABELS: ROC METRICS

From the confusion matrix (binary case), we can calculate "ROC" metrics.

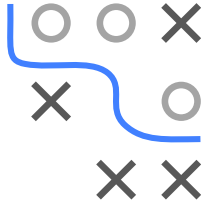


		True Class y		
		$+$	$-$	
Pred.	$+$	TP	FP	$\rho_{PPV} = \frac{\text{TP}}{\text{TP+FP}}$
$\hat{y}$	$-$	FN	TN	$\rho_{NPV} = \frac{\text{TN}}{\text{FN+TN}}$
		$\rho_{TPR} = \frac{\text{TP}}{\text{TP+FN}}$	$\rho_{TNR} = \frac{\text{TN}}{\text{FP+TN}}$	$\rho_{ACC} = \frac{\text{TP+TN}}{\text{TOTAL}}$

- True positive rate ρ_{TPR}: how many of the true 1s did we predict as 1?
- True Negative rate ρ_{TNR}: how many of the true 0s did we predict as 0?
- Positive predictive value ρ_{PPV}: if we predict 1, how likely is it a true 1?
- Negative predictive value ρ_{NPV}: if we predict 0, how likely is it a true 0?
- Accuracy ρ_{ACC}: how many instances did we predict correctly?

LABELS: ROC METRICS

Example:

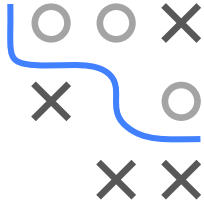


		Actual Class y		
		Positive	Negative	
$\hat{y}$ Pred.	Positive	**True Positive** (TP) = 20	**False Positive** (FP) = 180	Positive predictive value = TP / (TP + FP) = 20 / (20 + 180) = **10%**
	Negative	**False Negative** (FN) = 10	**True Negative** (TN) = 1820	Negative predictive value = TN / (FN + TN) = 1820 / (10 + 1820) ≈ **99.5%**
		True Positive Rate = TP / (TP + FN) = 20 / (20 + 10) ≈ **67%**	True Negative Rate = TN / (FP + TN) = 1820 / (180 + 1820) = **91%**	

https://en.wikipedia.org/wiki/Receiver_operating_characteristic

MORE METRICS AND ALTERNATIVE TERMINOLOGY

Unfortunately, for many concepts in ROC, 2-3 different terms exist.

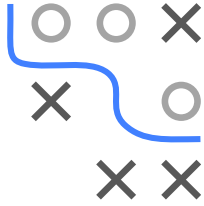


		True condition			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$	F_1 score = $\frac{1}{\frac{1}{2}\left(\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}\right)}$

▸ Clickable version/picture source

▸ Interactive diagram

LABELS: F_1 MEASURE



- It is difficult to achieve high **positive predictive value** and high **true positive rate** simultaneously.
- A classifier predicting more positive will be more sensitive (higher ρ_{TPR}), but it will also tend to give more *false* positives (lower ρ_{TNR}, lower ρ_{PPV}).
- A classifier that predicts more negatives will be more precise (higher ρ_{PPV}), but it will also produce more *false* negatives (lower ρ_{TPR}).

The F_1 **score** balances two conflicting goals:

1. Maximizing positive predictive value
2. Maximizing true positive rate

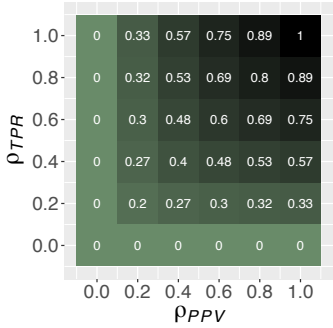
ρ_{F_1} is the harmonic mean of ρ_{PPV} and ρ_{TPR}:

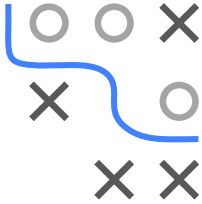
$$\rho_{F_1} = 2 \cdot \frac{\rho_{PPV} \cdot \rho_{TPR}}{\rho_{PPV} + \rho_{TPR}}$$

Note that this measure still does not account for the number of true negatives.

LABELS: F_1 MEASURE

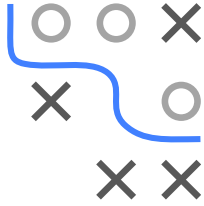
F_1 score for different combinations of ρ_{PPV} & ρ_{TPR}.
$\rightarrow$ Tends more towards the lower of the two combined values.



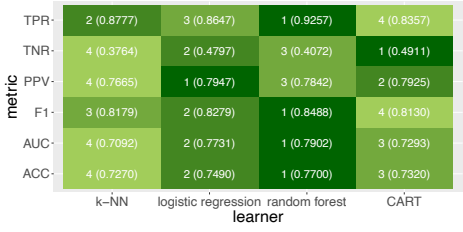


- A model with $\rho_{TPR} = 0$ (no positive instance predicted as positive) or $\rho_{PPV} = 0$ (no true positives among the predicted) has $\rho_{F_1} = 0$.
- Always predicting "negative": $\rho_{F_1} = 0$.
- Always predicting "positive":
  $\rho_{F_1} = 2 \cdot \rho_{PPV} / (\rho_{PPV} + 1) = 2 \cdot n_+ / (n_+ + n)$,
  which will be small when the size of the positive class n_+ is small.

WHICH METRIC TO USE?

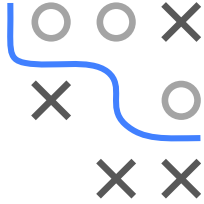


- As we have seen, there is a plethora of methods.
  $\rightarrow$ This leaves practitioners with the question of which to use.
- Consider a small benchmark study.
  - We let k-NN, logistic regression, a classification tree, and a random forest compete on classifying the `credit risk` data.
  - The data consist of 1000 observations of borrowers' financial situation and their creditworthiness (good/bad) as target.
  - Predicted probabilities are thresholded at 0.5 for the positive class.
  - Depending on the metric we use, learners are ranked differently according to performance (value of respective performance measure in parentheses):

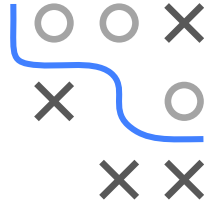


WHICH METRIC TO USE?

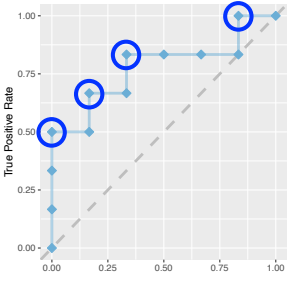
- We need not expect overly large discrepancies in general, but neither will we always see an unambiguous picture.
- Different metrics emphasize different aspects of performance.
  $\rightarrow$ The choice should be made in the domain context.
- For practitioners it is vital to understand what should be evaluated exactly, and which measure is appropriate.
  - Regarding credit risk, for instance, defaults are to be avoided, but not at all cost.
  - The bank must undertake a certain risk to remain profitable, so a more balanced measure such as the F_1 score might be in order.
  - On the other hand, a system detecting weapons at an airport should be able to achieve very high true positive rates, even if this comes at the expense of some false alarms.



Introduction to Machine Learning



Evaluation
Measures for Binary Classification: ROC Visualization

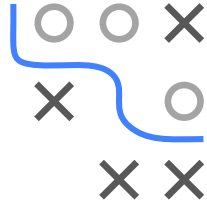


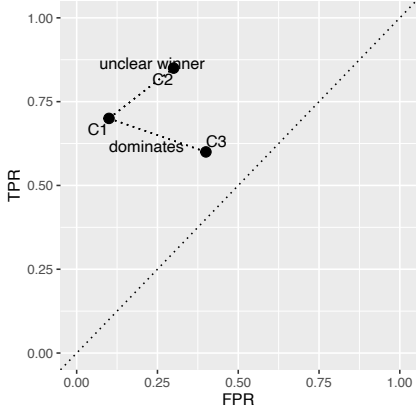
Learning goals

- Understand ROC curve
- Be able to compute a ROC curve manually
- Understand that ROC curve is invariant to class priors at test-time
- Discuss threshold selection
- Understand AUC

LABELS: ROC SPACE

- For comparing classifiers, we characterize them by their TPR and FPR values and plot them in a coordinate system.
- We could also use two different ROC metrics which define a trade-off, for instance, TPR and PPV.





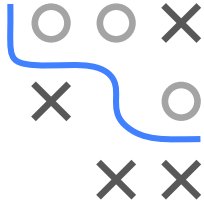
		True Class y	
		$+$	$-$
Pred. $\hat{y}$	$+$	TP	FP
	$-$	FN	TN

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

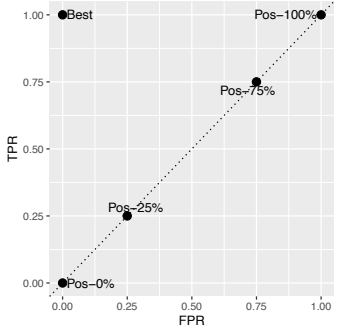
$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

LABELS: ROC SPACE

- The best classifier lies on the top-left corner, where FPR equals 0 and TPR is maximal.
- The diagonal is worst as it corresponds to a classifier producing random labels (with different proportions).

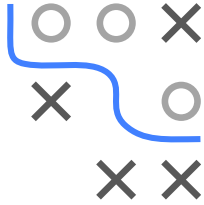


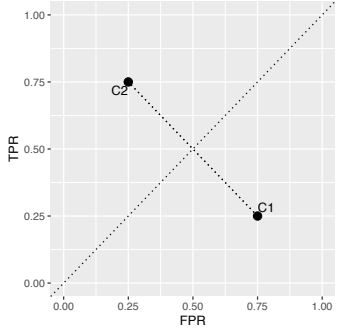
- If each positive x will be randomly classified with 25% as "pos", TPR $= 0.25$.
- If we assign each negative x randomly to "pos", FPR $= 0.25$.



LABELS: ROC SPACE

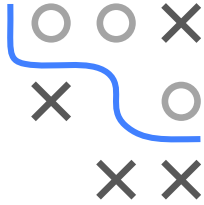
- In practice, we should never obtain a classifier below the diagonal.
- Inverting the predicted labels ($0 \mapsto 1$ and $1 \mapsto 0$) will result in a reflection at the diagonal.
  $\Rightarrow \text{TPR}_{\text{new}} = 1 - \text{TPR}$ and $\text{FPR}_{\text{new}} = 1 - \text{FPR}$.





LABEL DISTRIBUTION IN TPR AND FPR

TPR and FPR (ROC curves) are insensitive to the class distribution in the sense that they are not affected by changes in the ratio n_+/n_- (at prediction).



Example 1:
Proportion $n_+/n_- = 1$

	Actual Positive	Actual Negative
Pred. Positive	40	25
Pred. Negative	10	25

$\text{MCE} = 35/100 = 0.35$
$\text{TPR} = 0.8$
$\text{FPR} = 0.5$

Example 2:
Proportion $n_+/n_- = 2$

	Actual Positive	Actual Negative
Pred. Positive	80	25
Pred. Negative	20	25

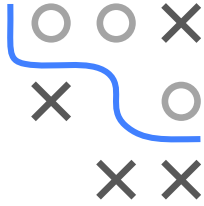
$\text{MCE} = 45/150 = 0.3$
$\text{TPR} = 0.8$
$\text{FPR} = 0.5$

Note: If class proportions differ during training, the above is not true. Estimated posterior probabilities can change!

FROM PROBABILITIES TO LABELS: ROC CURVE

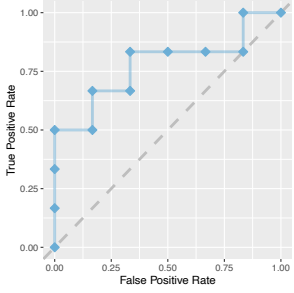
Remember: Both probabilistic and scoring classifiers can output classes by thresholding:

$$h(\mathbf{x}) = [\pi(\mathbf{x}) \geq c] \quad \text{or} \quad h(\mathbf{x}) = [f(\mathbf{x}) \geq c_f].$$



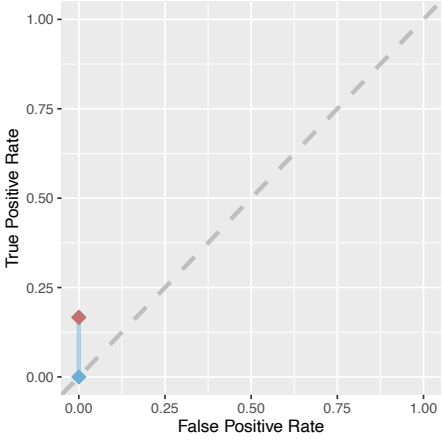
To draw a ROC curve:

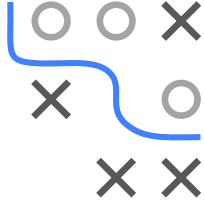
1. Rank test observations on decreasing score.
2. Start with $c = 1$, so we start in $(0, 0)$; we predict everything as negative.
3. Iterate through all possible thresholds c and proceed for each observation x as follows:
   - If x is positive, move TPR $1/n_+$ up, as we have one TP more.
   - If x is negative, move FPR $1/n_-$ right, as we have one FP more.



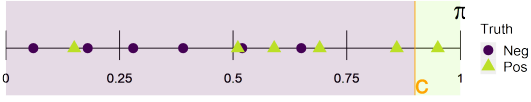
DRAWING ROC CURVES

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



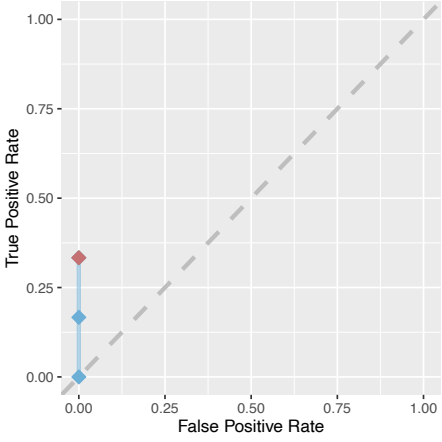


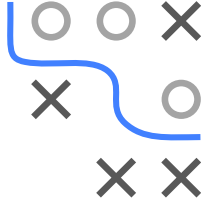
$c = 0.9$
$\rightarrow$ TPR = 0.167
$\rightarrow$ FPR = 0



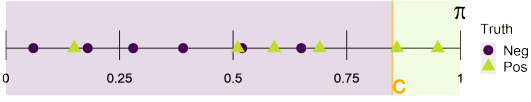
DRAWING ROC CURVES

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



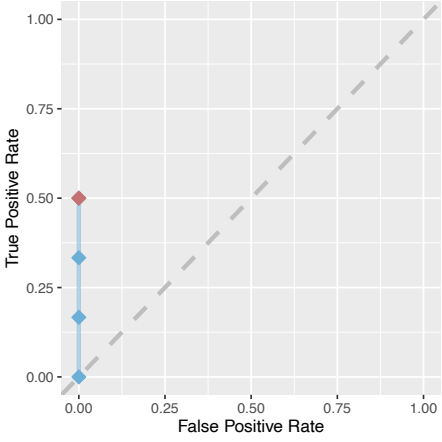


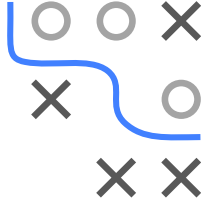
$c = 0.85$
$\rightarrow$ TPR = 0.333
$\rightarrow$ FPR = 0



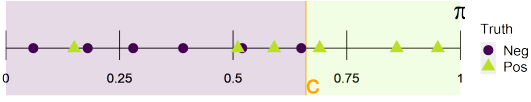
DRAWING ROC CURVES

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



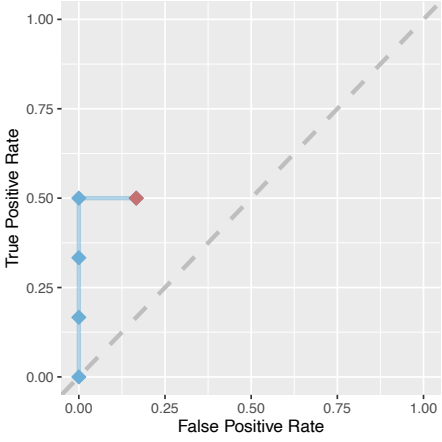


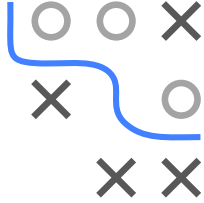
$c = 0.66$
$\rightarrow$ TPR = 0.5
$\rightarrow$ FPR = 0



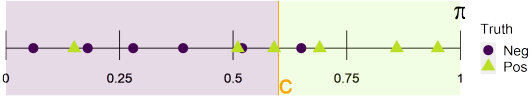
DRAWING ROC CURVES

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



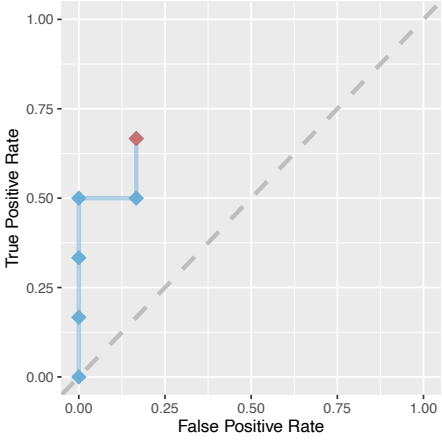


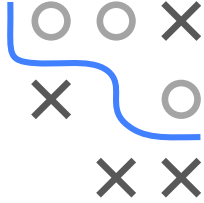
$c = 0.6$
$\rightarrow$ TPR = 0.5
$\rightarrow$ FPR = 0.167



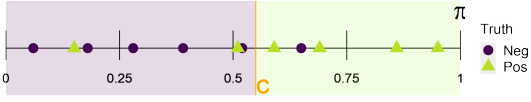
DRAWING ROC CURVES

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



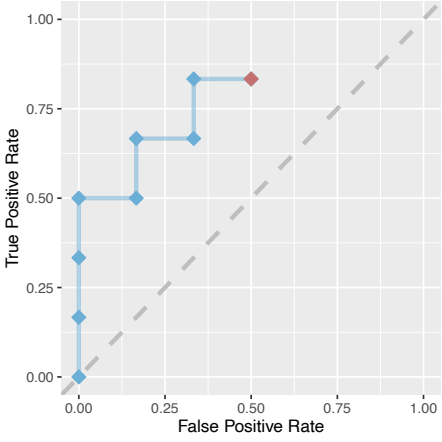


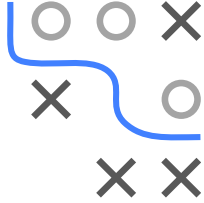
$c = 0.55$
$\rightarrow$ TPR = 0.667
$\rightarrow$ FPR = 0.167



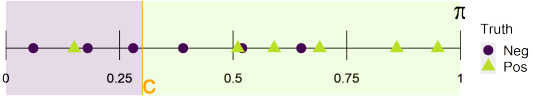
DRAWING ROC CURVES

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



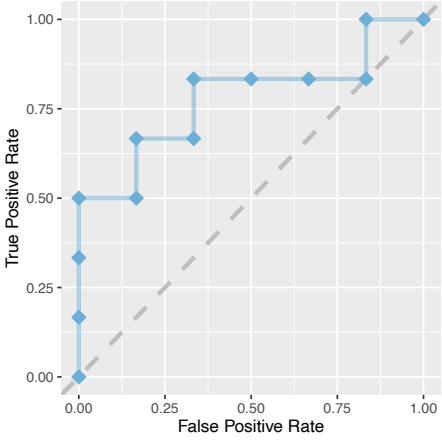


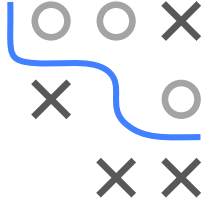
$c = 0.3$
$\rightarrow$ TPR = 0.833
$\rightarrow$ FPR = 0.5



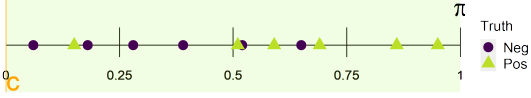
DRAWING ROC CURVES

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06



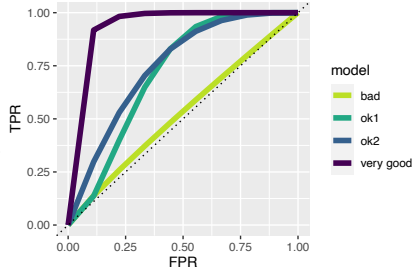


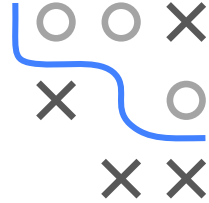
$c = 0$
$\rightarrow$ TPR = 1
$\rightarrow$ FPR = 1



ROC CURVE PROPERTIES

- The closer the curve to the top-left corner, the better.
- If ROC curves cross, a different model might be better in different parts of the ROC space.

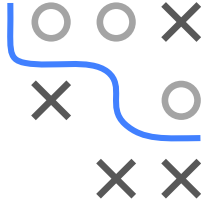




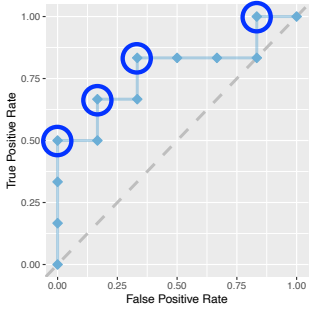
- Small thresholds will very liberally predict the positive class, and result in a potentially higher FPR, but also higher TPR.
- High thresholds will very conservatively predict the positive class, and result in a lower FPR and TPR.
- As we have not defined the trade-off between false positive and false negative costs, we cannot easily select the "best" threshold.
  $\rightarrow$ Visual inspection of all possible results seems useful.

CHOOSING THRESHOLD / OPERATING POINT

Often done visually and post-hoc, as class imbalances or costs are unknown a-priori.

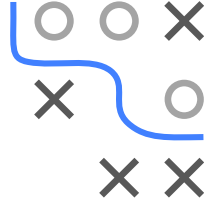


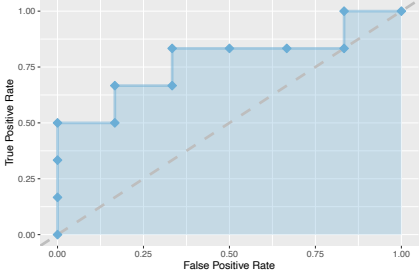
- Identify non-dominated points
- Assess TPR / FPR
- Decide which combo is best for task
- Pick associated threshold



AUC: AREA UNDER ROC CURVE

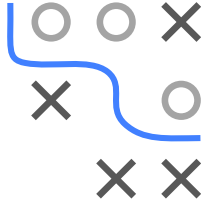
- AUC $\in [0, 1]$ is a single metric to evaluate scoring classifiers – independent of the chosen threshold.
  - AUC = 1: perfect classifier
  - AUC = 0.5: random, non-discriminant classifier
  - AUC = 0: perfect, with inverted labels





AUC AS A RANK-BASED METRIC

- We can also interpret the AUC as the probability of our classifier ranking a random positive observation higher than a random negative one.
- A perfect classifier will rank all positive above all negative observations, achieving AUC = 1.



Truth	Score
1	0.9
1	0.76
1	0.7
0	0.5
1	0.45
0	0.3
0	0.1

Choose a random positive

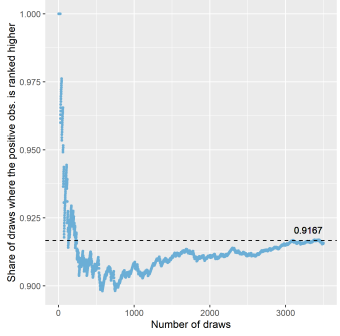
1	0.76

Choose a random negative

0	0.3

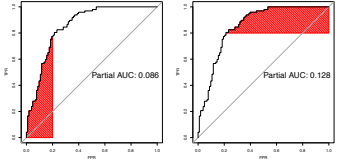
AUC = 0.9167

- Classifier ranks the positive higher than the negative
- This happens with a mean probability of 0.9167



Introduction to Machine Learning

Evaluation
Partial AUC

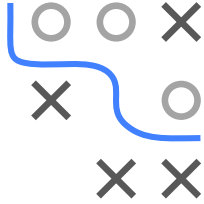


Learning goals

- Understand that entire AUC is not always relevant
- Learn about partial AUC

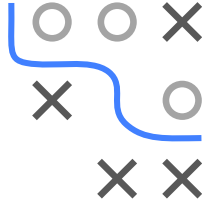
PARTIAL AUC

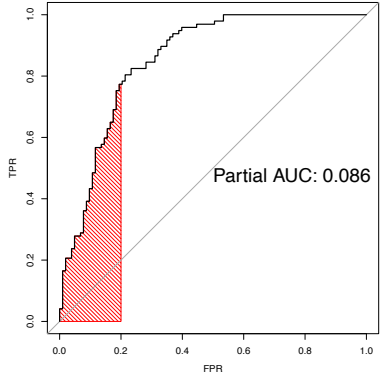
- TPR and FPR often treated asymmetrically in biomed contexts
- TPR = disease detection, is crucial
- But low FPR needed to avoid unnecessary treatments
- Common solution: Fix either TPR or FPR to a required value and optimize the other, but not easy to select exact point

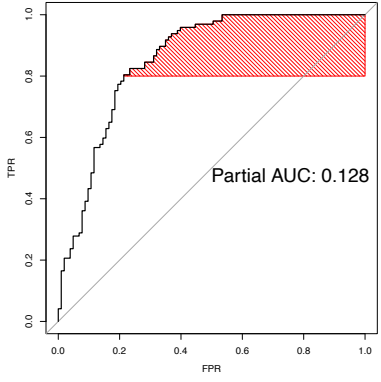


PARTIAL AUC

- Can be useful to limit region under ROC curve
- E.g. FPR > 0.2 or TPR < 0.8 might not be acceptable for task, then we don't want to integrate over that region



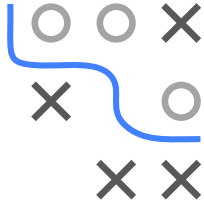




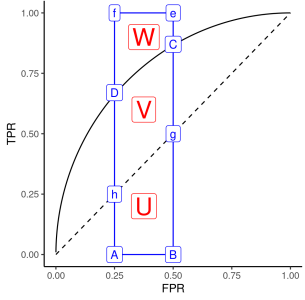
CORRECTED PARTIAL AUC

- Range of pAUC depends on cut-off values
- Normalize to $[0, 1]$:

$$\text{pAUC}_{\text{corrected}} = \frac{1}{2}\left(1 + \frac{\text{pAUC} - \text{pAUC}_{\text{min}}}{\text{pAUC}_{\text{max}} - \text{pAUC}_{\text{min}}}\right),$$

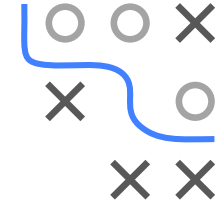


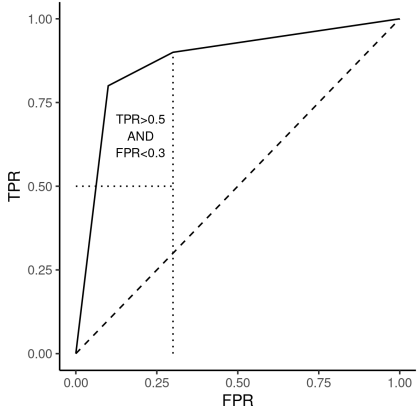
- pAUC is V+U = "A-B-C-D"
- pAUC_{min} is pAUC of random classifier, so U = "A-B-g-h"
- pAUC_{max} is U+V+W = "A-B-e-f"
- Compute percentage of V in V+W
- Rescale so random=0.5; optimal=1



2WAY PARTIAL AUC

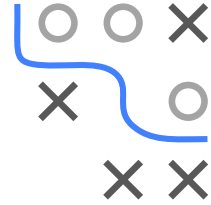
- Can also limit both TPR and FPR
- 2way pAUC = compute area under 2way limited segment





Introduction to Machine Learning

Evaluation
Multi-Class AUC



AUC(pos|neg) = AUC(1|3)

	Y	$\hat{\pi}_1$	$\hat{\pi}_2$	$\hat{\pi}_3$
pos	1	0.7	0.2	0.1
pos	1	0.5	0.3	0.2
~~2~~	~~0.3~~	~~0.5~~	~~0.2~~	
~~2~~	~~0.4~~	~~0.5~~	~~0.1~~	
neg	3	0.6	0.1	0.3
neg	3	0.1	0.1	0.8

AUC(pos|neg) = AUC(3|1)

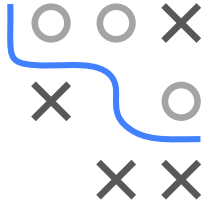
	Y	$\hat{\pi}_1$	$\hat{\pi}_2$	$\hat{\pi}_3$
neg	1	0.7	0.2	0.1
neg	1	0.5	0.3	0.2
~~2~~	~~0.3~~	~~0.5~~	~~0.2~~	
~~2~~	~~0.4~~	~~0.5~~	~~0.1~~	
pos	3	0.6	0.1	0.3
pos	3	0.1	0.1	0.8

Learning goals

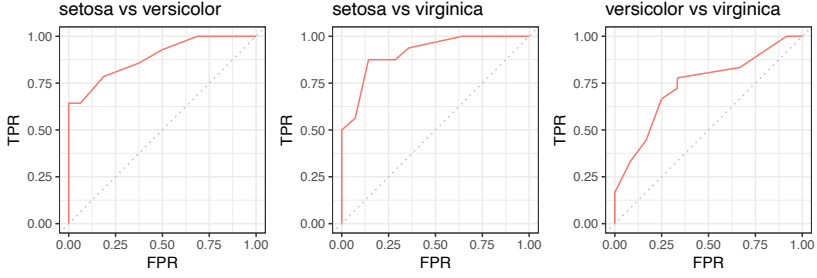
- Understand that generalizing AUC to multi-class is not trivial
- Learn how multi-class AUC can be derived

MULTI-CLASS AUC

- AUC and other ROC metrics for binary classification
- Different ways to estimate **multi-class AUC**
- Often based on aggregated binary AUCs:
  e.g. 1-vs-1 or 1-vs-rest

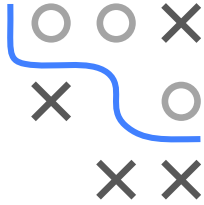


Example: 1-vs-1 on `iris`



MULTI-CLASS AUC

- Def AUC($k \mid \ell$) for classes k (pos) and ℓ (neg)
- Compute AUC: Subset preds to rows of true k and ℓ, use $\hat{\pi}_k$
- Interprete: Prob that random member of ℓ has a lower prob to belong to class k than random member of class k.



Example: AUC(3|1) with $g = 3$ classes

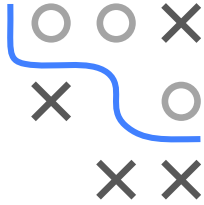
AUC(pos|neg) = AUC(3|1)

	Y	$\hat{\pi}_1$	$\hat{\pi}_2$	$\hat{\pi}_3$
neg	1	0.7	0.2	0.1
neg	1	0.5	0.3	0.2
	2	0.3	0.5	0.2
	2	0.4	0.5	0.1
pos	3	0.6	0.1	0.3
pos	3	0.1	0.1	0.8

1. Subset pred rows to true classes 1 and 3
2. Use $k = 3$ as pos and $\ell = 1$ as neg class
3. Compute standard AUC with $\hat{\pi}_3$ as scores
4. AUC(3|1) = 1:
   all pos have higher $\hat{\pi}_3$ than negs

MULTI-CLASS AUC

- For binary classes: always $\text{AUC}(1|0) = \text{AUC}(0|1)$
- For multi-class usually: $\text{AUC}(k \mid \ell) \neq \text{AUC}(\ell \mid k)$
- **Example** with $g = 3$ where $\text{AUC}(1|3) \neq \text{AUC}(3|1)$:
  - $\text{AUC}(3|1) = 1$ (RHS) as before
  - $\text{AUC}(1|3) \neq 1$ (LHS)



AUC(pos|neg) = AUC(1|3)

	Y	$\hat{\pi}_1$	$\hat{\pi}_2$	$\hat{\pi}_3$
pos	1	0.7	0.2	0.1
pos	1	0.5	0.3	0.2
	2	0.3	0.5	0.2
	2	0.4	0.5	0.1
neg	3	0.6	0.1	0.3
neg	3	0.1	0.1	0.8

AUC(pos|neg) = AUC(3|1)

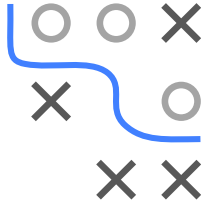
	Y	$\hat{\pi}_1$	$\hat{\pi}_2$	$\hat{\pi}_3$
neg	1	0.7	0.2	0.1
neg	1	0.5	0.3	0.2
	2	0.3	0.5	0.2
	2	0.4	0.5	0.1
pos	3	0.6	0.1	0.3
pos	3	0.1	0.1	0.8

MULTI-CLASS AUC

Hand and Till (2001) proposed to avg AUC via **1-vs-1**:

- For all class pairs, compute AUC($k \mid \ell$).

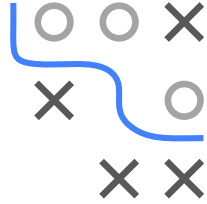
$$\text{AUC}_{MC} = \frac{1}{g(g-1)} \sum_{k \neq \ell} \text{AUC}(k|\ell) \in [0, 1].$$



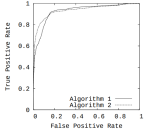
Comments:

- Other defs use **1-vs-rest** and need to avg only g AUC values
- 1-vs-rest creates imbal classes even if orig classes are balanced
- Imbalanced classes can be considered by weighting individual AUC values with class priors [Ferri et al. (2003)]

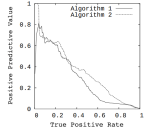
Introduction to Machine Learning



Evaluation
Precision-Recall Curves



(a) Comparison in ROC space

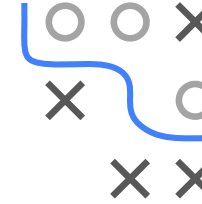


(b) Comparison in PR space

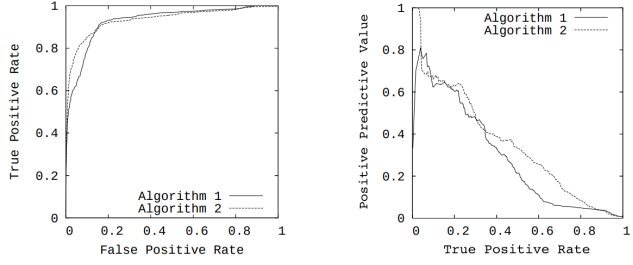
Learning goals

- Understand PR curves
- Same as PPV-TPR curve
- Compare to standard TPR-FPR ROC curve

PRECISION-RECALL CURVES



- Slightly changed ROC plot
- Simply plot precision and recall, instead of TPR-FPR
- Precision = $\rho_{PPV} = \frac{TP}{TP+FP}$, recall = $\rho_{TPR} = \frac{TP}{TP+FN}$
- Might call them TPR-PPV curve
- NB: Both metrics don't depend on TNs

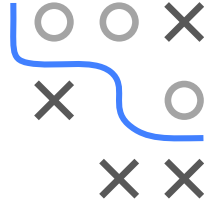


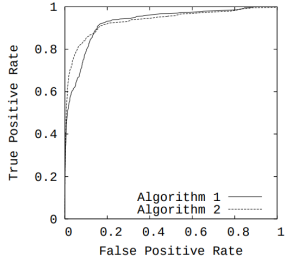
(a) Comparison in ROC space (b) Comparison in PR space

Davis and Goadrich (2006): The Relationship Between Precision-Recall and ROC Curves (URL).

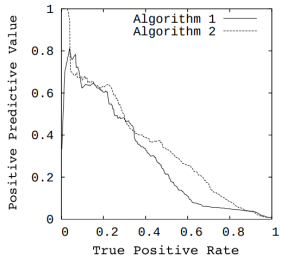
PRECISION-RECALL CURVES

- Might be better for highly imbal data ($n_- \gg n_+$) than TPR-FPR
- Figure (a): ROC; both learners seem to perform well
- Figure (b): PR; visible room for improvement (top-right=best)
- PR reveals better that algo 2 has advantage over 1





(a) Comparison in ROC space

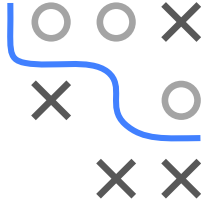


(b) Comparison in PR space

Davis and Goadrich (2006): The Relationship Between Precision-Recall and ROC Curves (URL).

IMBALANCED DATA

- Assume imbalanced classes with $n_- \gg n_+$
- If neg class large, typically less interested in high TNR = low FPR, but more in PPV
- Large (abs) change in FP yields small change in FPR
- PPV likely more informative



FP=10:

	True +1	True -1
Pred. Pos	100	10
Pred. Neg	10	9990
Total	110	10000

$\text{TPR} = 10/11$
$\text{FPR} = 0.001$
$\text{PPV} = 10/11$

FP=100:

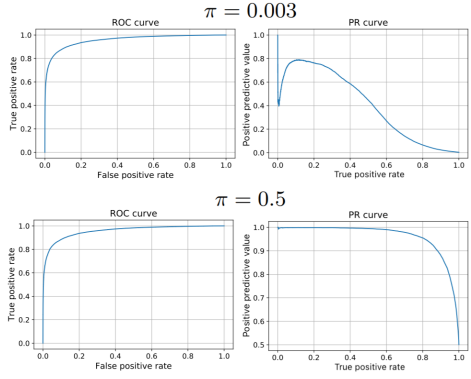
	True +1	True -1
Pred. +1	100	100
Pred. -1	10	9900
Total	110	10000

$\text{TPR} = 10/11$
$\text{FPR} = 0.01$
$\text{PPV} = 1/2$

RHS: Given test says +1, it's now a coin flip that this is correct.

IMBALANCED DATA

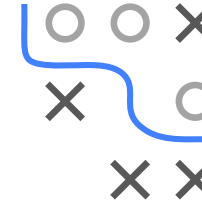
- Top row: Imbal classes with $\pi = 0.003$
- Bottom: balanced with $\pi = 0.5$
- ROC curves (LHS) are similar
- PR curve (RHS) changes strongly from imbal to bal classes

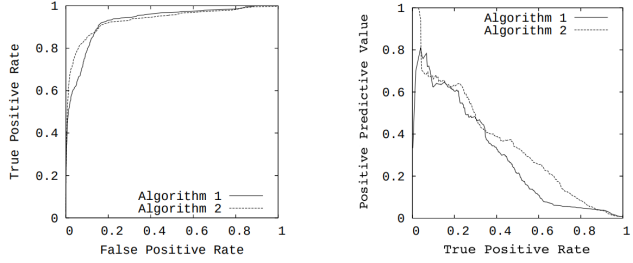


Wissam Siblini et. al. (2004): Master your Metrics with Calibration (URL).

CONCLUSIONS

- Curve fully dominates in ROC space iff dominates in PR-space
- In imbalanced situations rather use PR than standard TPR-FPR
- If comparing few models on a single task, probably plot both. Then observe and think.
- For tuning: can also use PR-AUC (or partial versions)

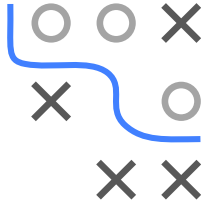




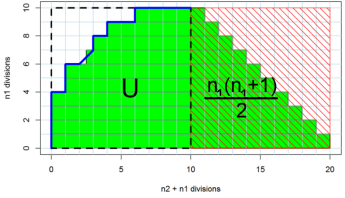
(a) Comparison in ROC space (b) Comparison in PR space

Davis and Goadrich (2006): The Relationship Between Precision-Recall and ROC Curves (URL).

Introduction to Machine Learning



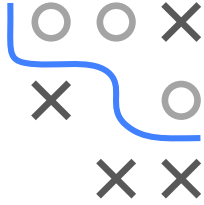
Evaluation
AUC & Mann-Whitney-U Test



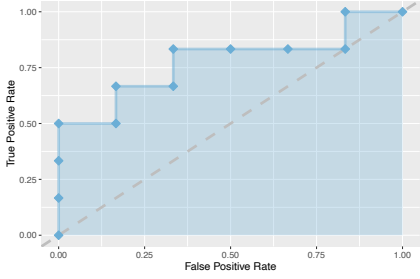
Learning goals

- Understand the rank-based nature of AUC
- See the connection between AUC and Mann-Whitney-U statistic

AUC AS A RANK-BASED METRIC



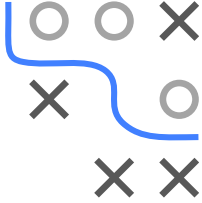
- The AUC metric is intimately related to the **Mann-Whitney-U test**, also known as **Wilcoxon rank-sum test**.
- This connection is best understood viewing the AUC from a slightly different angle: it is, in effect, a **rank-based** metric.
- Recall that, constructing the ROC curve, we count TP and FP.



- The AUC abstracts from the actual classification scores and considers only their rank.

AUC AS A RANK-BASED METRIC

- We can interpret the AUC as the probability of our classifier ranking a random positive observation higher than a random negative one.
- A perfect classifier will rank all positive above all negative observations, achieving AUC = 1.



Truth	Score
1	0.9
1	0.76
1	0.7
0	0.5
1	0.45
0	0.3
0	0.1

Choose a random positive

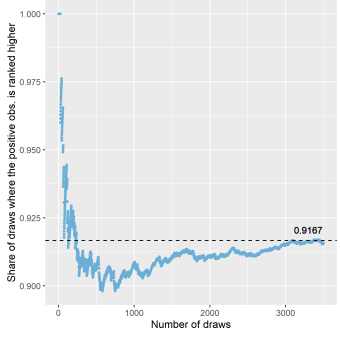
1	0.76

Choose a random negative

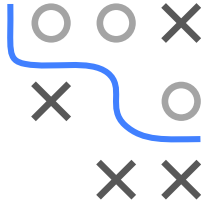
0	0.3

AUC = 0.9167

- Classifier ranks the positive higher than the negative
- This happens with a mean probability of 0.9167



MANN-WHITNEY-U TEST

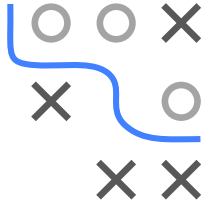


- The Mann-Whitney-U test is a **non-parametric hypothesis test** on the difference in location between two samples X_1, X_2 of sizes n_1 and n_2, respectively.
- Under the null, X_1 and X_2 follow the same (unknown) distribution $\mathbb{P}$, and for any pair of observations $x_{1,1} \in X_1$, $x_{2,1} \in X_2$ drawn at random from $\mathbb{P}$, the following statement holds: $\mathbb{P}(x_{1,1} \in X_1) > \mathbb{P}(x_{2,1} \in X_2) = \mathbb{P}(x_{1,1} \in X_1) < \mathbb{P}(x_{2,1} \in X_2) = 0.5$.
- The test statistic estimates the probability of a random sample from X_1 ranking higher than one from X_2 (R_1 denoting the sum of ranks of the $x_{1,i}$):

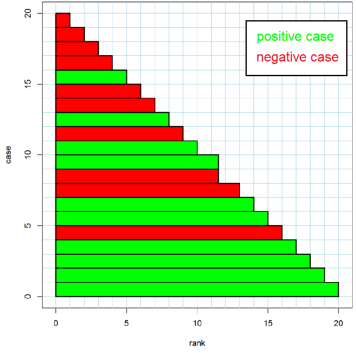
$$U = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \mathbb{I}[x_{1,i} > x_{2,j}] = R_1 - \frac{n_1(n_1+1)}{2}$$

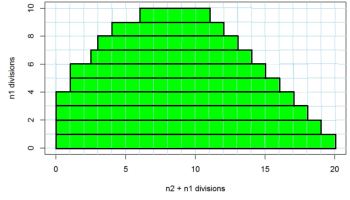
- For large samples, U is approximately normally distributed.

AUC & MANN-WHITNEY-U TEST

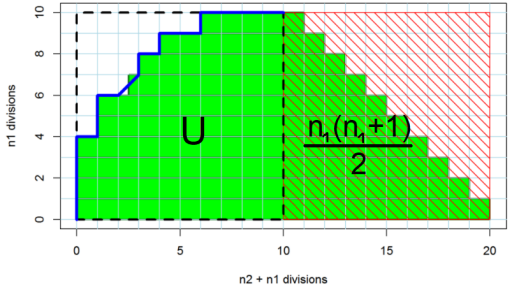


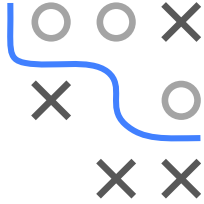
- We can directly interpret the AUC in the light of the U statistic.
- In order to see this, plot the ranks of all the scores as a stack of horizontal bars, and color them by label.
- Next, keep only the green ones, and slide them horizontally to get a nice even stairstep on the right edge:





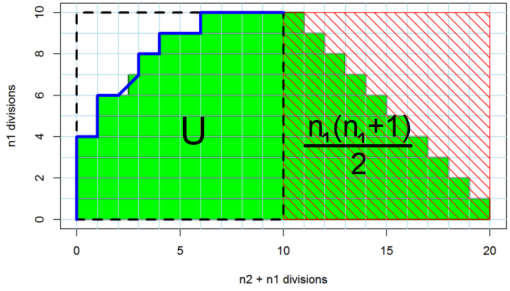
AUC & MANN-WHITNEY-U TEST

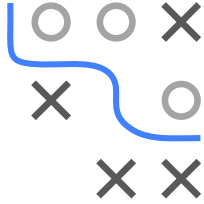




- Denoting the respective numbers of cases as n_+ and n_-, we have:
  $U = R_+ - \frac{n_+(n_+ + 1)}{2}$.
- The area of the green bars on the right is equal to $\frac{n_+(n_+ + 1)}{2}$.

AUC & MANN-WHITNEY-U TEST





- U: area of the green bars on the left.
- $n_+ \cdot n_-$: area of the dashed rectangle.

$\Rightarrow$ AUC is U normalized to the unit square:

$$\text{AUC} = \frac{U}{n_+ \cdot n_-}.$$

INTRODUCTION TO MACHINE LEARNING