Introduction to Machine Learning with R and mlr3

Bernd Bischl & Marvin N. Wright

DAGStat, March 2025

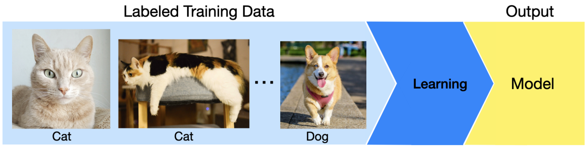
PART 1

WHAT IS ML?

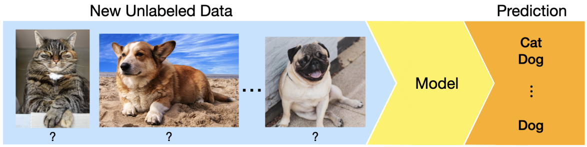
"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."
Tom Mitchell, Carnegie Mellon University, 1998

$\Rightarrow$ 99 % of this lecture is about **supervised learning**:

Training



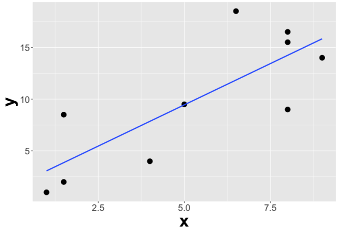
Prediction



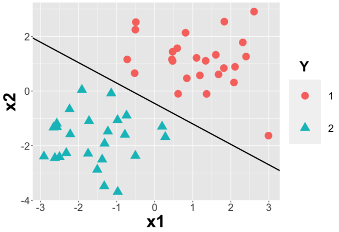
TASKS

- Supervised tasks are labeled data situations where the goal is to learn the functional relationship between inputs (features) and output (target)
- We distinguish between **regression** and **classification** tasks, depending on whether the target is **numerical** or **categorical**

Regression: Target is **numerical**, e.g., predict days a patient has to stay in hospital

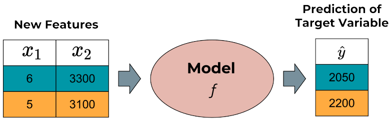


Classification: Target is **categorical**, e.g., predict one of two risk categories for a life insurance customer



MODELS AND PARAMETERS

- A model is a function that maps features to predicted targets

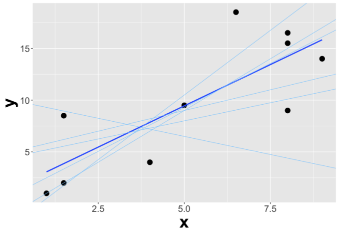


- For finding the model that describes the relation between features and target best, one needs to restrict the set of all possible functions
- This restricted set of functions is called **hypothesis space**. E.g., one could consider only simple linear functions as hypothesis space
- Functions are fully determined by parameters. E.g., in the case of linear functions, $y = \theta_0 + \theta_1 x$, the parameters θ_0 (intercept) and θ_1 (slope) determine the relationship between y and x
- Finding the optimal model means finding the optimal set of parameters

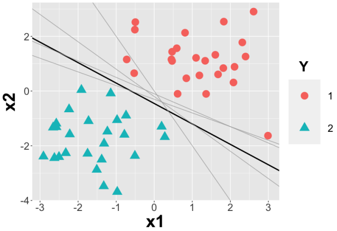
LEARNER

- Learns automatically the relation between features and target – given a set of training data
- Learner picks the best element of the **hypothesis space**, i.e., the function that fits the training data best

Regression:

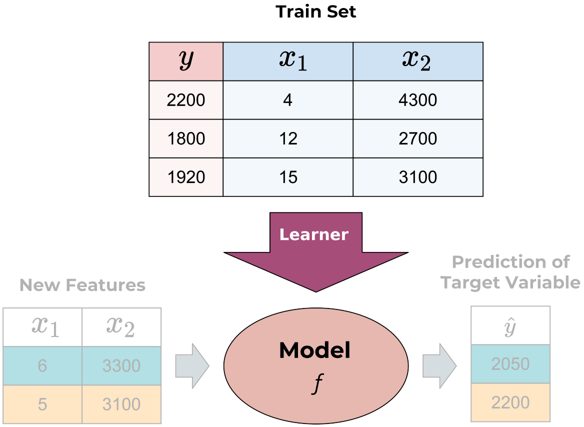


Classification:



LEARNER

- Learner uses labeled training data to learn a model f. This model is applied to new data for predicting the target variable



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

x_1	x_2
6	3300
5	3100

$\hat{y}$
2050
2200

LOSS AND RISK MINIMIZATION

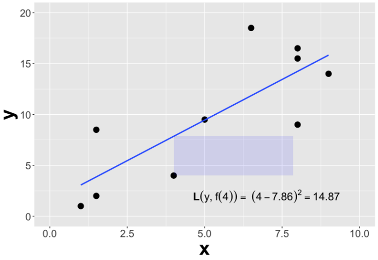
- Loss: Measured pointwise for each observation, e.g., L_2-loss

$$L\left(y, f(\mathbf{x})\right) = \left(y - f(\mathbf{x})\right)^2$$

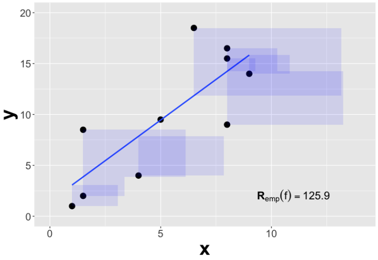
- Risk: Measured for entire model. Sums up pointwise losses.

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

Squared **loss** of one **observation**.



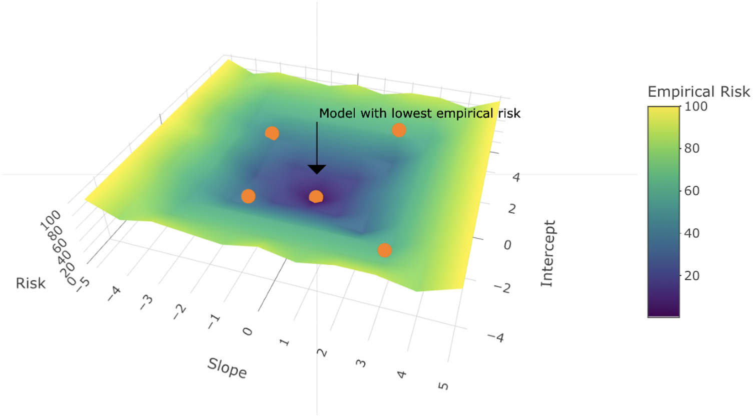
Empirical **risk** of entire **model**



EMPIRICAL RISK MINIMIZATION

- The risk surface visualizes the empirical risk for all possible parameter values of the parameter vector $\boldsymbol{\theta}$
- Minimizing the empirical risk is usually done by numerical optimization

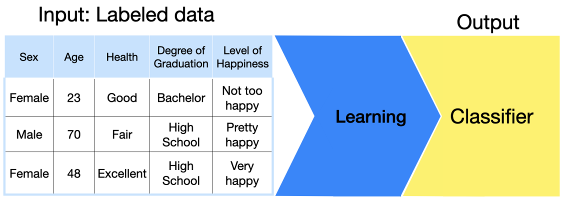
$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}).$$



CLASSIFICATION TASKS

- Learn function that assigns categorical class labels to observations
- Each observation belongs to exactly one class
- The task can contain two (binary) or multiple (multi-class) classes

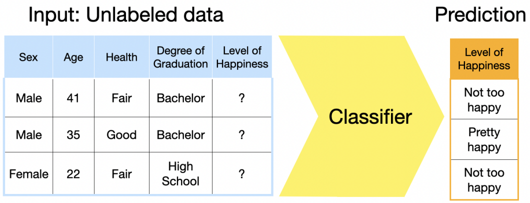
Training



Input: Labeled data

Sex	Age	Health	Degree of Graduation	Level of Happiness
Female	23	Good	Bachelor	Not too happy
Male	70	Fair	High School	Pretty happy
Female	48	Excellent	High School	Very happy

Prediction



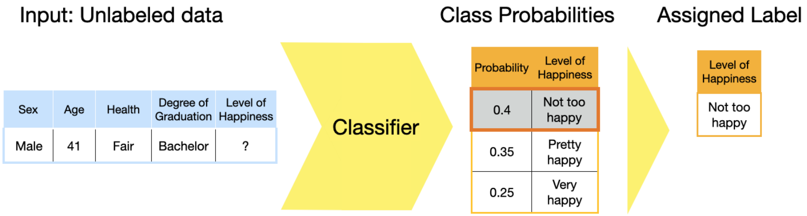
Input: Unlabeled data

Sex	Age	Health	Degree of Graduation	Level of Happiness
Male	41	Fair	Bachelor	?
Male	35	Good	Bachelor	?
Female	22	Fair	High School	?

Level of Happiness
Not too happy
Pretty happy
Not too happy

BASIC DEFINITIONS

- For every observation a model outputs the probability (probabilistic classifier) or score (scoring classifier) of each class
- In the multi-class case, the class label is usually assigned by choosing the class with the maximum score or probability
- In the binary case, a class label is assigned by choosing the class whose probability or score exceeds a threshold value c



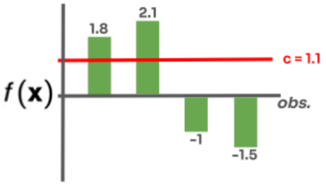
Sex	Age	Health	Degree of Graduation	Level of Happiness
Male	41	Fair	Bachelor	?

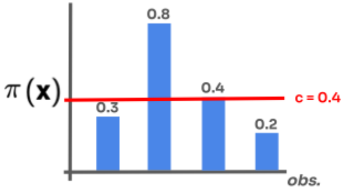
Probability	Level of Happiness
0.4	Not too happy
0.35	Pretty happy
0.25	Very happy

Level of Happiness
Not too happy

THRESHOLDING

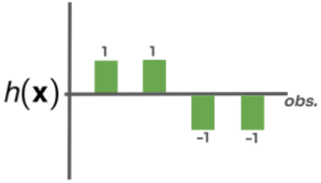
- For imbalanced cases or class with costs, we might want to deviate from the standard conversion of scores to classes
- Introduce basic concept (for binary case) and add details later
- Convert scores or probabilities to class outputs by thresholding: $h(\mathbf{x}) := [\pi(\mathbf{x}) \geq c]$ or $h(\mathbf{x})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

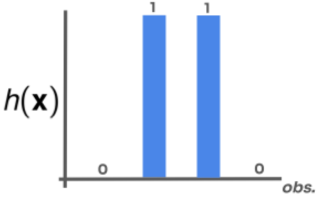




THRESHOLDING

- 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





PART 1

ML Basics: Data, Model, Learner, ERM

Learner Overview

Performance Estimation

Performance Measures

K-NN – METHOD SUMMARY

REGRESSION CLASSIFICATION NONPARAMETRIC WHITE-BOX

General idea

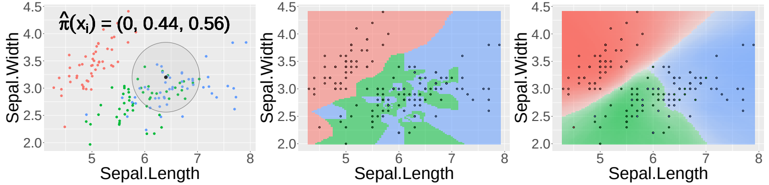
- **similarity** in feature space (w.r.t. certain **distance metric** $d(\mathbf{x}^{(i)}, \mathbf{x})$) $\rightsquigarrow$ similarity in target space
- **Prediction** for $\mathbf{x}$: construct k**-neighborhood** $N_k(\mathbf{x})$ from k points closest to $\mathbf{x}$ in $\mathcal{X}$, then predict
  - (weighted) mean target for **regression**: $\hat{y} = \frac{1}{\sum\limits_{i:\mathbf{x}^{(i)} \in N_k(\mathbf{x})} w_i} \sum\limits_{i:\mathbf{x}^{(i)} \in N_k(\mathbf{x})} w_i y^{(i)}$ with $w_i = \frac{1}{d(\mathbf{x}^{(i)},\mathbf{x})}$

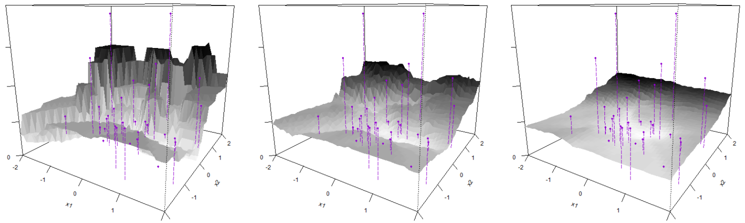
    $\rightarrow$ optional: higher weights w_i for close neighbors
  - most frequent class for **classification**: $\hat{y} = \underset{\ell \in \{1,\ldots,g\}}{\arg\max} \sum\limits_{i:\mathbf{x}^{(i)} \in N_k(\mathbf{x})} \mathbb{I}(y^{(i)} = \ell)$

    $\Rightarrow$ Estimating posterior probabilities as $\hat{\pi}_\ell(\mathbf{x}^{(i)}) = \frac{1}{k} \sum\limits_{i:\mathbf{x}^{(i)} \in N_k(\mathbf{x})} \mathbb{I}(y^{(i)} = \ell)$
- **Nonparametric** behavior: parameters = training data; no compression of information
- Not immediately interpretable, but inspection of neighborhoods can be revealing

K-NN – METHOD SUMMARY

Hyperparameters Neighborhood **size** k (locality), **distance** metric (next page)





Classification
Left: Neighborhood for exemplary observation in `iris`, $k = 50$
Middle: Prediction surface for $k = 1$
Right: Prediction surface for $k = 50$

Regression
Left: Prediction surface for $k = 3$
Middle: Prediction surface for $k = 7$
Right: Prediction surface for $k = 15$

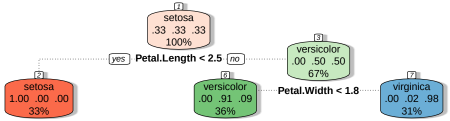
- Small $k \Rightarrow$ very local, "wiggly" decision boundaries
- Large $k \Rightarrow$ rather global, smooth decision boundaries

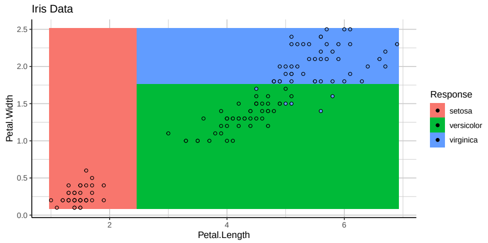
CART – METHOD SUMMARY

REGRESSION | CLASSIFICATION | NONPARAMETRIC | WHITE-BOX | FEATURE SELECTION

General idea (CART – Classification and Regression Trees)

- Start at root node containing all data
- Perform repeated **axis-parallel binary splits** in feature space to obtain **rectangular partitions** at terminal nodes $Q_1, \dots, Q_M$
- Splits based on reduction of node **impurity** $\rightarrow$ empirical risk minimization (**ERM**)
- In each step:
  - Find **optimal split** (feature-threshold combination) $\rightarrow$ greedy search
  - Assign constant prediction c_m to all obs. in Q_m
    $\rightarrow$ Regression: c_m is average of y
    $\rightarrow$ Classif.: c_m is majority class (or class proportions)
  - Stop when a pre-defined criterion is reached
    $\rightarrow$ See Complexity control

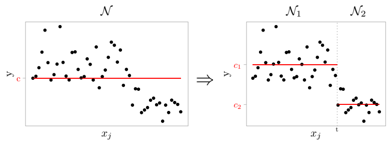




Hypothesis space $$\mathcal{H} = \left\{ f(\mathbf{x}) : f(\mathbf{x}) = \sum_{m=1}^{M} c_m \mathbb{I}(\mathbf{x} \in Q_m) \right\}$$

CART – METHOD SUMMARY

Empirical risk



- Splitting **feature** x_j **at split point** t divides a parent node $\mathcal{N}$ into two child nodes:

  $$\mathcal{N}_1 = \{(\mathbf{x}, y) \in \mathcal{N} : x_j \leq t\} \text{ and } \mathcal{N}_2 = \{(\mathbf{x}, y) \in \mathcal{N} : x_j > t\}$$

- Compute empirical risks in child nodes and minimize their sum to find best split (impurity reduction):

  $$\arg\min_{j,t} \mathcal{R}(\mathcal{N}, j, t) = \arg\min_{j,t} \mathcal{R}(\mathcal{N}_1) + \mathcal{R}(\mathcal{N}_2)$$

  Note: If $\mathcal{R}$ is the average instead of the sum of loss functions, we need to reweight: $\frac{|\mathcal{N}_t|}{|\mathcal{N}|}\mathcal{R}(\mathcal{N}_t)$
- In general, compatible with arbitrary losses – typical choices:
  - *g*-way classification:

Brier score $\rightarrow$ **Gini** impurity	**Bernoulli** loss $\rightarrow$ **entropy** impurity
$\mathcal{R}(\mathcal{N}) = \sum_{(\mathbf{x},y)\in\mathcal{N}} \sum_{k=1}^{g} \hat{\pi}_k^{(\mathcal{N})}(1 - \hat{\pi}_k^{(\mathcal{N})})$	$\mathcal{R}(\mathcal{N}) = -\sum_{(\mathbf{x},y)\in\mathcal{N}} \sum_{k=1}^{g} \hat{\pi}_k^{(\mathcal{N})} \log \hat{\pi}_k^{(\mathcal{N})}$

  - Regression (**quadratic** loss): $\mathcal{R}(\mathcal{N}) = \sum_{(\mathbf{x},y)\in\mathcal{N}} (y - c)^2$ with $c = \frac{1}{|\mathcal{N}|} \sum_{(\mathbf{x},y)\in\mathcal{N}} y$

Optimization

- **Exhaustive** search over all split candidates, choice of risk-minimal split
- In practice: reduce number of split candidates (e.g., using quantiles instead of all observed values)

RANDOM FORESTS – METHOD SUMMARY

REGRESSION | CLASSIFICATION | NONPARAMETRIC | BLACK-BOX | FEATURE SELECTION

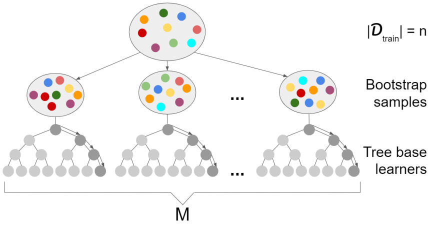
General idea

- **Bagging ensemble** of M tree **base learners** fitted on **bootstrap** data samples

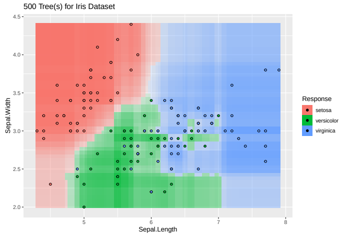
  $\Rightarrow$ Reduce **variance** by ensembling while slightly increasing **bias** by bootstrapping
  - Use unstable, **high-variance** base learners by letting trees grow to full size
  - Promoting **decorrelation** by random subset of candidate features for each split
- **Predict** via averaging (regression) or majority vote (classification) of base learners

Hypothesis space

$$\mathcal{H} = \left\{ f(\mathbf{x}) : f(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{T^{[m]}} c_t^{[m]} \mathbb{I}(\mathbf{x} \in Q_t^{[m]}) \right\}$$



Schematic depiction of bagging process



Prediction surface for `iris` data with 500-tree ensemble

RANDOM FORESTS – METHOD SUMMARY

Empirical risk & Optimization Just like tree base learners

Out-of-bag (OOB) error

- Ensemble prediction for obs. outside individual trees' bootstrap training sample $\Rightarrow$ unseen test sample
- Use resulting loss as unbiased estimate of **generalization error**
- Mainly useful for tuning and less for model comparison as we usually compare all models uniformly by CV

Feature importance

- Based on **improvement in split criterion:** aggregate improvements by all splits using j-th feature
- Based on **permutation:** permute j-th feature in OOB observations and compute impact on OOB error

Hyperparameters

- **Ensemble size**, i.e., number of trees
- **Complexity** of base learners, e.g., tree depth, min-split, min-leaf-size
- **Number of split candidates**, i.e., number of features to be considered at each split
  $\Rightarrow$ frequently used heuristics with total of p features: $\lfloor \sqrt{p} \rfloor$ for classification, $\lfloor p/3 \rfloor$ for regression

GRADIENT BOOSTING – METHOD SUMMARY

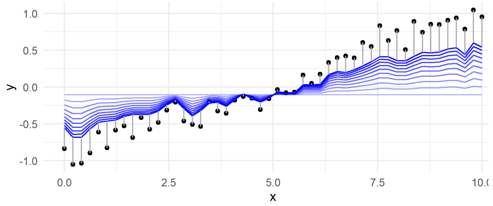
REGRESSION | CLASSIFICATION | (NON)PARAMETRIC | BLACK-BOX | FEATURE SELECTION

General idea

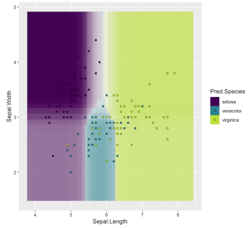
- **Sequential ensemble** of *M* **base learners** by greedy forward stagewise additive modeling
  - In each iteration a base learner is fitted to current **pseudo residuals** $\Rightarrow$ one boosting iteration is one approximate **gradient step in function space**
  - Base learners are typically **trees**, **linear regressions** or **splines**
- **Predict** via (weighted) sum of base learners

Hypothesis space

$$\mathcal{H} = \left\{ f(\mathbf{x}) : f(\mathbf{x}) = \sum_{m=1}^{M} \beta^{[m]} b(\mathbf{x}, \boldsymbol{\theta}^{[m]}) \right\}$$



Boosting prediction function with GAM base learners for univariate regression problem after 10 iterations



Boosting prediction surface with tree base learners for `iris` data after 100 iterations (*right:* contour lines of discriminant functions)

GRADIENT BOOSTING – METHOD SUMMARY

Empirical risk

- In general, compatible with any **differentiable** loss
- Base learner in iteration *m* is fitted on **Pseudo residuals**:
  $\tilde{r}^{(i)} = -\frac{\partial L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)}{\partial f\left(\mathbf{x}^{(i)}\right)}$ by minimizing the **L2-loss**: $\sum_{i=1}^{n} (\tilde{r}^{(i)} - b(\mathbf{x}^{(i)}, \boldsymbol{\theta}))^2$

Optimization

- Same optimization procedure as base learner, while keeping the current ensemble $\hat{f}^{[m-1]}$ fixed
  $\Rightarrow$ Efficient and generally applicable since *inner* loss is always L2
- $\beta^{[m]}$ is found via **line search** or fixed to a **small constant value** and combined with the leaf values $c_t^{[m]}$ for tree base learners: $\tilde{c}_t^{[m]} = \beta^{[m]} \cdot c_t^{[m]}$

Hyperparameters

- **Ensemble size**, i.e., number of base learners
- **Complexity** of base learners (depending on type used)
- **Learning rate** β, i.e., impact of next base learner

GRADIENT BOOSTING – PRACTICAL HINTS

Scalable Gradient Boosting

- **Feature and data subsampling** for each base learner fit
- **Parallelization** and **approximate split finding** for tree base learners
- GPU accelaration

Explainable / Componentwise Gradient Boosting

- Base learners of **simple linear regression** models or **splines**, selecting a single feature in each iteration
- Allows **feature selection** and creates an **interpretable** model since uni- and bivariate effects can be visualized directly.
- Feature interactions can be learned via ranking techniques (e.g., GA^{2}M FAST)

Tuning

- Use **early-stopping** to determine ensemble size
- Various **regularization parameters**, e.g., L1/L2, number of leaves, ... that need to be carefully tuned
- Tune learning rate and base learner complexity hyperparameters on **log-scale**

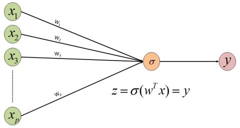
NEURAL NETWORKS – METHOD SUMMARY

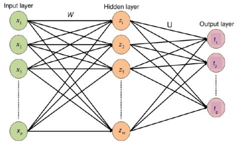
REGRESSION CLASSIFICATION (NON)PARAMETRIC BLACK-BOX

General idea

- Learn **composite function** through series of nonlinear feature transformations, represented as **neurons**, organized hierarchically in **layers**
  - Basic neuron operation: 1) affine **transformation** ϕ (weighted sum of inputs), 2) nonlinear **activation** σ
  - Combinations of simple building blocks to create a complex model
- Optimize via **mini-batch stochastic gradient descent (SGD)** variants:
  - Gradient of each weight can be infered from the **computational graph** of the network $\rightarrow$ **Automatic Differentiation** (AutoDiff)
  - Algorithm to compute weight updates based on the loss is called **Backpropagation**

Hypothesis space $\mathcal{H} = \left\{ f(\mathbf{x}) : f(\mathbf{x}) = \tau \circ \phi \circ \sigma^{(h)} \circ \phi^{(h)} \circ \sigma^{(h-1)} \circ \phi^{(h-1)} \circ \ldots \circ \sigma^{(1)} \circ \phi^{(1)}(\mathbf{x}) \right\}$





NEURAL NETWORKS – METHOD SUMMARY

Architecture

- Input layer: original features $\mathbf{x}$
- Hidden layers: nonlinear transformation of previous layer $\phi^{(h)} = \sigma^{(h-1)}(\phi^{(h-1)})$
- Output layer: number of output neurons and activation depends on problem $\tau(\phi)$
  - Regression: one output neuron, $\tau =$ identity
  - Binary classification: one output neuron, $\tau = \frac{1}{1+\exp(-\boldsymbol{\theta}^\top \mathbf{x})}$ (logistic sigmoid)
  - Multiclass Classification: g output neurons, $\tau_j = \frac{\exp(f_j)}{\sum_{j=1}^{g} \exp(f_j)}$ (softmax)

Empirical risk In general, compatible with any differentiable loss

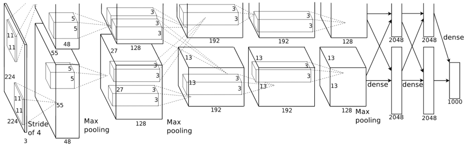
Optimization

- Variety of different optimizers, mostly based on some form of **stochastic gradient descent (SGD)**
- Improvements:
  (1) Accumulation of previous gradients $\rightarrow$ **Momentum**
  (2) Weight specific scaling based on previous squared gradients $\rightarrow$ **RMSProb**
  $\Rightarrow$ **ADAM** combines (1) and (2)
  (3) Learning rate schedules, e.g., decaying or cyclical learning rates
- Training progress is measured in full passes over the full training data, called **epochs**
- **Batch size** is a hyperparameter and limited by input data dimension

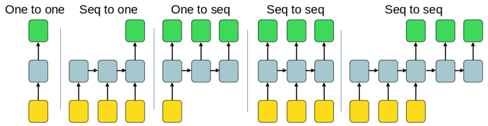
NEURAL NETWORKS – METHOD SUMMARY

Network types Large variety of architectures for different data modelities

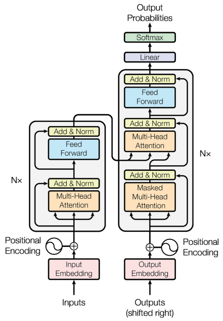
- **Feedforward NNs / multi-layer perceptrons (MLPs):** sequence of **fully-connected** layers ⇒ tabular data
- **Convolutional NNs (CNNs):** sequence of feature map extractors with spatial awareness ⇒ images, time series
- **Recurrent NNs (RNNs):** handling of sequential, variable-length information ⇒ times series, text, audio
- **Transformers:** Learning invariances from data, handling multiple/any data modalities



Convolutional network architecture



Recurrent network architecture



Transformer network architecture

NEURAL NETWORKS – METHOD SUMMARY

Hyperparameters

- **Architecture**:
  - Lots of design choices $\Rightarrow$ tuning problem of its own.
  - Typically: hierachical optimization of components (cells) and macro structure of network $\rightarrow$ **Neural Architecture Search (NAS)**
  - Many predifined (well working) architectures exist for standard tasks
- **Training**:
  - Initial learning rate and various regularization parameters
  - Number of epochs is determined by **early-stopping**
  - **Data-augmentation**, e.g., applying random rotations to input images

Foundation models

- **Enormous** models trained on vast amounts of (general) data, e.g., all of wikipedia, in **self-supervised** fashion
- Used as starting point (**pre-trained**) and fine-tuned via **transfer** or **few-shot** learning for other tasks requiring little data
- Examples: GPT-3 for language, CLIP for vision-language, . . .

PART 1

ML Basics: Data, Model, Learner, ERM

Learner Overview

Performance Estimation

Performance Measures

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):

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

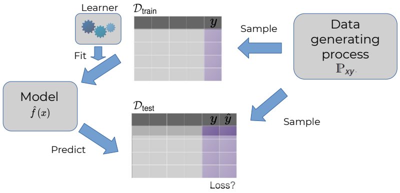
$$\min_{\theta \in \Theta} \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

- 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)

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



Learner
$\mathcal{D}_{\text{train}}$
y
Sample
Data generating process $\mathbb{P}_{xy}$.
Fit
Model $\hat{f}(x)$
$\mathcal{D}_{\text{test}}$
y $\hat{y}$
Sample
Predict
Loss?

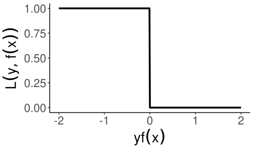
NB: Very often, no dedicated test-set is available, and what we describe here is not same as hold-out splitting (see later).

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.

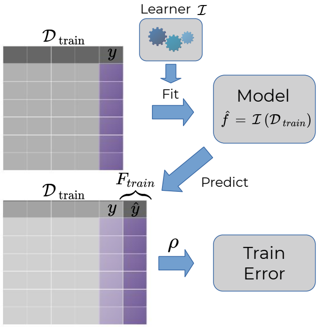
- Outside: We often want to eval with "nr of misclassifed examples", so 0-1 loss.
- Problem: 0-1 neither differentiable nor continuous. Hence: Inner loss = binomial. (0-1 actually NP hard).
- For evaluation, differentiability is not required.



TRAINING ERROR

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

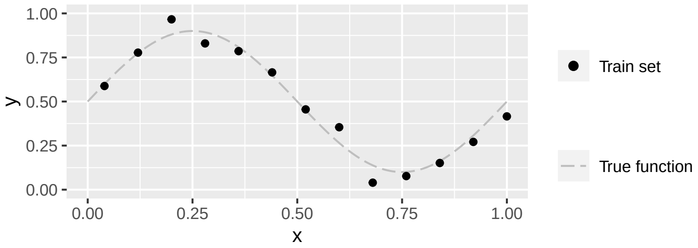
$$\rho(\mathbf{y}_{\text{train}}, \boldsymbol{F}_{\text{train}}) \text{ where } \boldsymbol{F}_{\text{train}} = \begin{bmatrix} \hat{f}_{\mathcal{D}_{\text{train}}}(\mathbf{x}^{(1)}_{\text{train}}) \\ \dots \\ \hat{f}_{\mathcal{D}_{\text{train}}}(\mathbf{x}^{(m)}_{\text{train}}) \end{bmatrix}$$


Learner $\mathcal{I}$
$\mathcal{D}_{\text{train}}$
y
Fit
Model
$\hat{f} = \mathcal{I}(\mathcal{D}_{train})$
Predict
F_{train}
$\mathcal{D}_{\text{train}}$
y
$\hat{y}$
ρ
Train Error

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

EXAMPLE 2: POLYNOMIAL REGRESSION

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



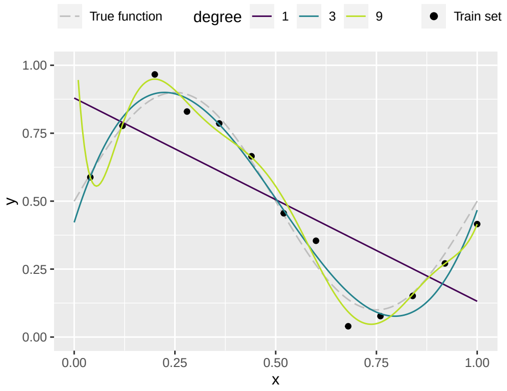
We fit a d^{th}-degree polynomial:

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

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

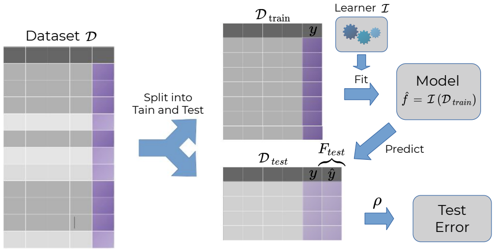
Using the train error chooses overfitting model of maximal complexity.

TEST ERROR AND HOLD-OUT SPLITTING

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

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

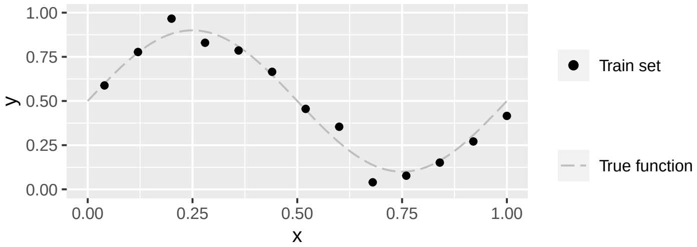
- Partition data, e.g., 2/3 for train and 1/3 for test.



A.k.a. holdout splitting.

EXAMPLE: POLYNOMIAL REGRESSION

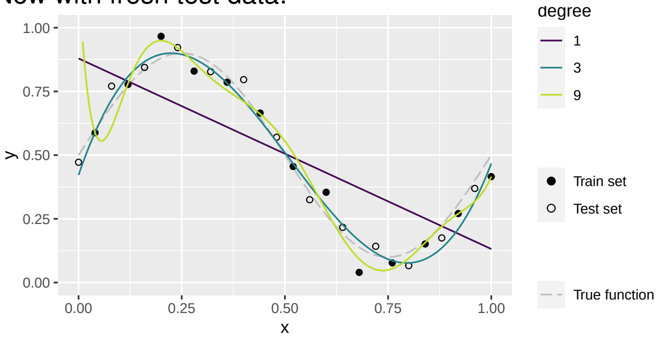
Previous example:



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

EXAMPLE: POLYNOMIAL REGRESSION

Now with fresh test data:

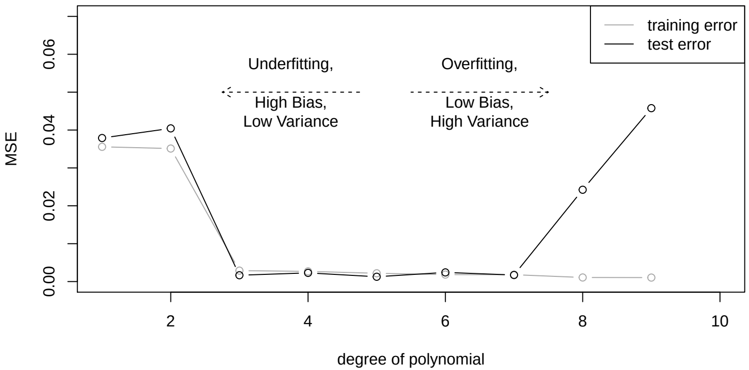


- $d = 1$: MSE = 0.038: clearly underfitting
- $d = 3$: MSE = 0.002: pretty OK
- $d = 9$: MSE = 0.046: clearly overfitting

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*:

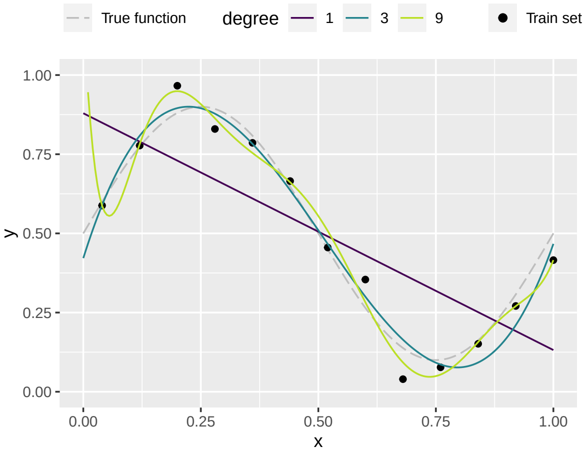


Increasing model complexity tends to cause

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

UNDER- AND OVERFITTING IN REGRESSION

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



PART 1

ML Basics: Data, Model, Learner, ERM

Learner Overview

Performance Estimation

Performance Measures

METRICS FOR REGRESSION

Commonly used evaluation metrics include:

- Sum of Squared Errors (SSE): $\rho_{SSE}(\mathbf{y}, \boldsymbol{F}) = \sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$
- Mean Squared Error (MSE): $\rho_{MSE}(\mathbf{y}, \boldsymbol{F}) = \frac{1}{m}\sum_{i=1}^{m} SSE$
- Root Mean Squared Error (RMSE): $\rho_{RMSE}(\mathbf{y}, \boldsymbol{F}) = \sqrt{MSE}$
- R-Squared: $\rho_{R^2}(\mathbf{y}, \boldsymbol{F}) = 1 - \frac{\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{m}(y^{(i)} - \bar{y})^2}$
- Mean Absolute Error (MAE):
  $\rho_{MAE}(\mathbf{y}, \boldsymbol{F}) = \frac{1}{m}\sum_{i=1}^{m}|y^{(i)} - \hat{y}^{(i)}| \in [0; \infty)$

METRICS FOR CLASSIFICATION

Commonly used evaluation metrics include:

- Accuracy:
  $\rho_{ACC} = \frac{1}{m}\sum_{i=1}^{m}[y^{(i)} = \hat{y}^{(i)}] \in [0,1]$.
- Misclassification error (MCE):
  $\rho_{MCE} = \frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \neq \hat{y}^{(i)}] \in [0,1]$.
- Brier Score:
  $\rho_{BS} = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{\pi}^{(i)} - y^{(i)}\right)^2$
- Log-loss:
  $\rho_{LL} = \frac{1}{m}\sum_{i=1}^{m}\left(-y^{(i)}\log\left(\hat{\pi}^{(i)}\right) - \left(1 - y^{(i)}\right)\log\left(1 - \hat{\pi}^{(i)}\right)\right)$.

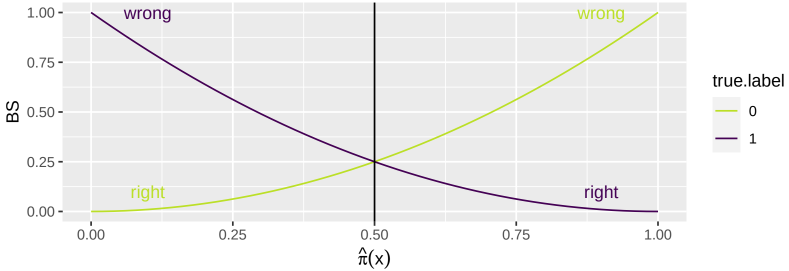
The probabalistic metrics, Brier Score and Log-Loss penalize false confidence, i.e. predicting the wrong label with high probability, heavily.

PROBABILITIES: BRIER SCORE

Measures squared distances of probabilities from the true class labels:

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

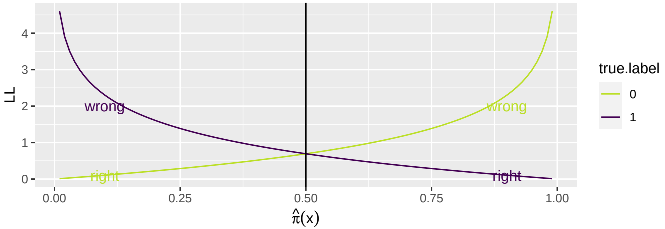
- Fancy name for MSE on probabilities.
- Usual definition for binary case; $y^{(i)}$ must be encoded as 0 and 1.



PROBABILITIES: LOG-LOSS

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

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

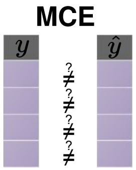


- Optimal value is 0, “confidently wrong” is penalized heavily.
- Multi-class version: $\rho_{LL,MC} = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{g} o_k^{(i)} \log \left(\hat{\pi}_k^{(i)} \right)$.

LABELS: MCE & ACC

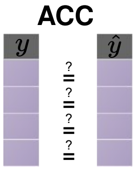
The **misclassification error rate (MCE)** counts the number of incorrect predictions and presents them as a rate:

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



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.

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**.

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.

LABELS: ROC METRICS

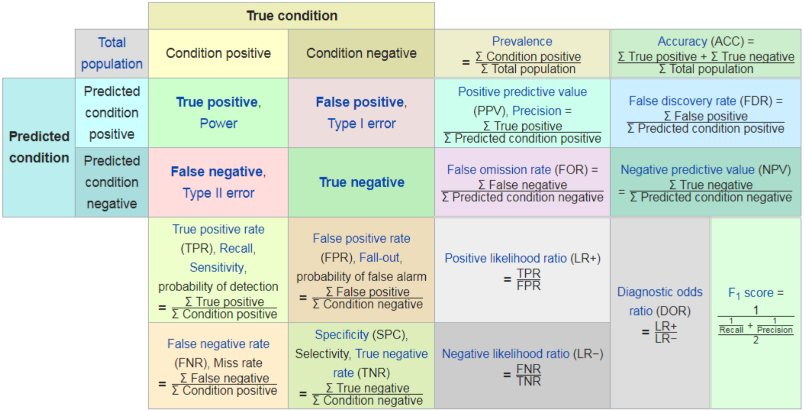
From the confusion matrix (binary case), we can calculate "ROC" metrics.

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

- 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?

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) $= \frac{\text{LR+}}{\text{LR−}}$	F_1 score = $\frac{1}{\frac{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}{2}}$
		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−) $= \frac{\text{FNR}}{\text{TNR}}$		

▸ 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

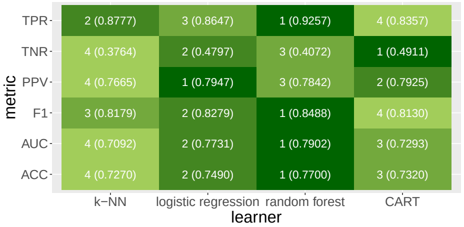
ρ_{F_1} is the harmonic mean of ρ_{PPV} and ρ_{TPR}:

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

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

WHICH METRIC TO USE?

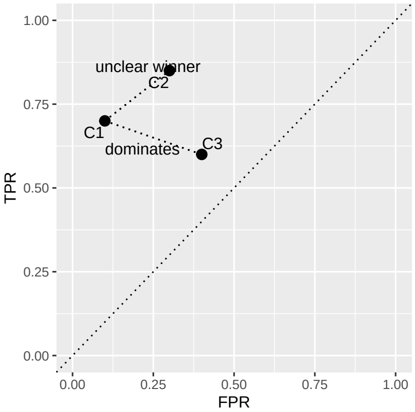
- As we have seen, there is a plethora of methods.
  $\rightarrow$ This leaves practitioners with the question of which to use.
- Consider a small benchmark study.
  - We let k-NN, logistic regression, a classification tree, and a random forest compete on classifying the `credit risk` data.
  - The data 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):



	k−NN	logistic regression	random forest	CART
TPR	2 (0.8777)	3 (0.8647)	1 (0.9257)	4 (0.8357)
TNR	4 (0.3764)	2 (0.4797)	3 (0.4072)	1 (0.4911)
PPV	4 (0.7665)	1 (0.7947)	3 (0.7842)	2 (0.7925)
F1	3 (0.8179)	2 (0.8279)	1 (0.8488)	4 (0.8130)
AUC	4 (0.7092)	2 (0.7731)	1 (0.7902)	3 (0.7293)
ACC	4 (0.7270)	2 (0.7490)	1 (0.7700)	3 (0.7320)

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

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

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

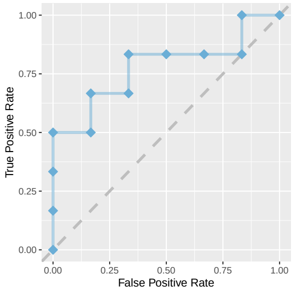
FROM PROBABILITIES TO LABELS: ROC CURVE

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

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

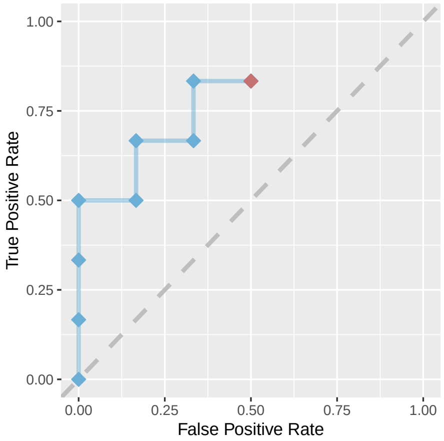
To draw a ROC curve:

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.

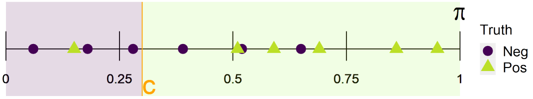


DRAWING ROC CURVES

#	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

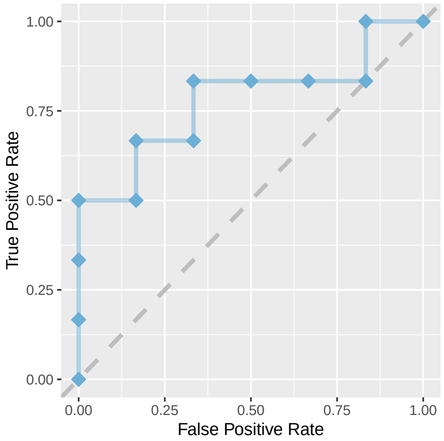


$c = 0.3$
$\rightarrow$ TPR = 0.833
$\rightarrow$ FPR = 0.5

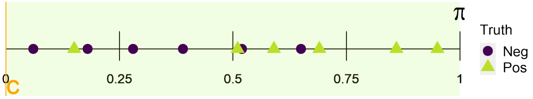


DRAWING ROC CURVES

#	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

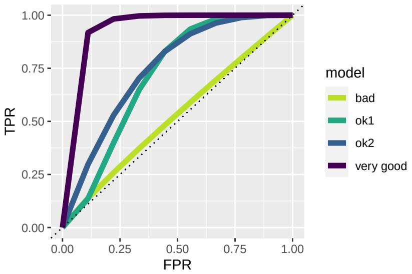


$c = 0$
$\rightarrow$ TPR = 1
$\rightarrow$ FPR = 1



ROC CURVE PROPERTIES

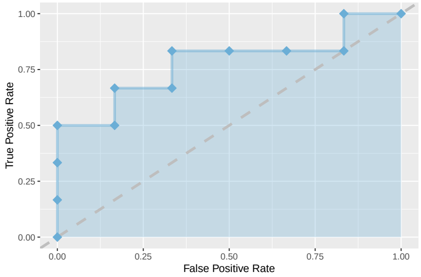
- The closer the curve to the top-left corner, the better.
- If ROC curves cross, a different model might be better in different parts of the ROC space.



- 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.

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


True Positive Rate
False Positive Rate