A Not So Short Introduction To Statistical Machine Learning and Neural Networks

SOSC 2023

Prof. Luca Scrucca

Università degli Studi di Perugia
luca.scrucca@unipg.it
https://luca-scr.github.io
luca-scr
luca_scr

25 October 2023

STATISTICS

- **Statistics** is the discipline that studies the collection, analysis, interpretation, presentation, and organization of data.

- Classical statistical methods:
  - Point estimation
  - Confidence intervals
  - Hypothesis testing
  - Statistical models for regression and classification
  - Likelihood-based inference
  - Bayesian inference
  - Multivariate methods
  - ...

- **Leo Breiman** ("Statistical Modeling: The Two Cultures", Statistical Science, 2001) described "two cultures":

    1. "generative" modeling culture which seeks to develop stochastic models that fit the data, and then make inferences about the data-generating mechanism based on the structure of those models. Implicit is the notion that there is a true model generating the data, and often a "best" way to analyze the data.

    2. "predictive" modeling culture which focuses on predictions, ignoring the underlying data generating mechanism, and discuss only accuracy of predictions made by different algorithms.

- According to Breiman "Statistics starts with data. Think of the data as being generated by a black box [...]"

  Two main goals can be pursued when analyzing data:

    - Prediction, i.e to be able to predict what the responses are going to be to future input variables;
    - Inference, i.e to infer how nature is associating the response variables to the input variables.

Machine/Statistical Learning

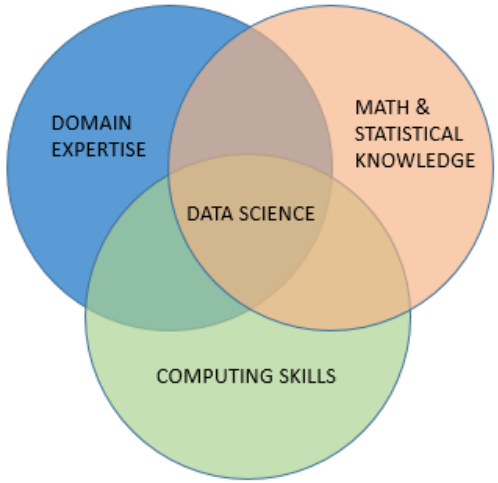
- **Machine learning** is the study of how computer algorithms can improve automatically through experience and by the use of data.

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E. – Tom Mitchell (1997)

- In the context of predictive modelling, the difference between machine learning and statistical learning is blurred.
  - Machine learning (ML) tends to be focused more on developing efficient algorithms that scale to large data in order to optimize a predictive model.
  - Statistical learning (SL) generally pays more attention to the probabilistic structure of the model in order to provide an assessment of the uncertainty.

- **Data Science** is a vaguely defined, constantly changing, cross-disciplinary field.
  From a statistician point of view, data science can be seen as a broader view of statistics.

  When physicists do mathematics, they don't say they're doing "number science". They're doing math. If you're analyzing data, you're doing statistics. You can call it data science or informatics or analytics or whatever, but it's still statistics.
  — Karl Broman (U of Wisconsin)

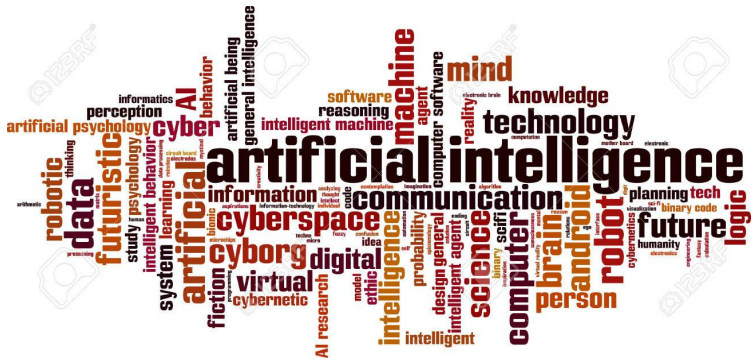


- **Big Data** refers to data sets that are too large or complex to be dealt with by traditional data analysis software.
  Big data are usually described in terms of three key concepts: volume, variety, and velocity.

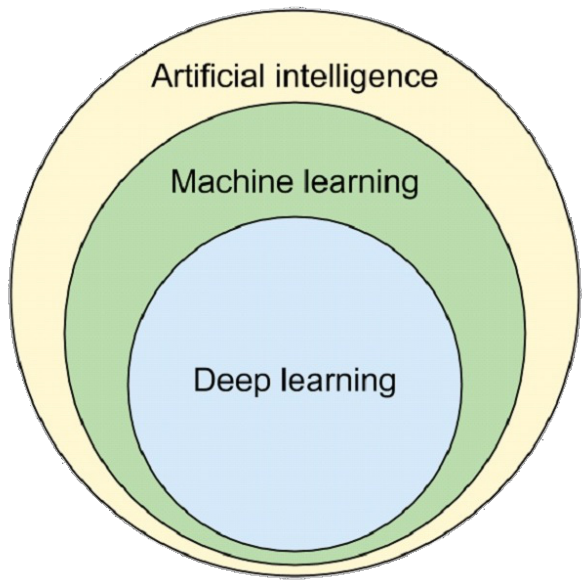
Artificial Intelligence

- **Artificial Intelligence** (AI) refers to the ability of digital machines to perform tasks that typically require human intelligence.

  Can machines think?
  – Alan Turing (1950)



- AI can be seen as a branch of CS, and in the years since its introduction in 1950s has experienced several waves of optimism followed by disappointment and the loss of funding (aka “AI winter”), followed by new approaches, success and renewed funding.
- There is a large debate and no single definition of the AI field that is universally accepted.
- In a broad sense, AI is an interdisciplinary science with multiple approaches, where advancements in ML and Deep Learning (DL) play a central role.
- AI encompasses various subfields, including ML, SL, and DL.



- Suppose we collected data for a sample of n observations. The **training set** is made of pairs of input and output variables:

$$\mathcal{D}_{\text{train}} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n}$$

- Assume there exists a dependency between them, so the output y_i can be expressed as a function of the input variables $\boldsymbol{x}_i$ and some other unobservable (latent) variables $\boldsymbol{z}_i$:

$$y_i = f(\boldsymbol{x}_i, \boldsymbol{z}_i)$$

- The aim of **supervised learning** is to fit a model to learn the mapping from the observable input to the output

$$\widehat{y}_i = g(\boldsymbol{x}_i \mid \boldsymbol{\theta})$$

where $g(.)$ is a statistical model and $\boldsymbol{\theta}$ the unknown parameters.

- The learning task corresponds to finding the parameters that minimize a loss function measuring the deviation of our prediction $\widehat{y}_i$ from the observed output y_i:

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} L(y_i, \widehat{y}_i) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} L(y_i, g(\boldsymbol{x}_i \mid \boldsymbol{\theta}))$$

- Different supervised learning algorithms differ in the models or the loss functions they assume, or the procedures they use in optimization.

- In **regression** problems
  - the output y_i is a numerical value (quantitative response)
  - $g(.)$ is a regressor function
  - loss is often the squared error, so the aim is to find the best $\boldsymbol{\theta}$ that minimize the fitting error.

- In **classification** problems
  - the output y_i is a discrete label (qualitative response)
  - $g(.)$ is a discriminant/classification function
  - if the loss function is the Zero-One loss the aim is to minimize the total number of misclassifications.

- Popular supervised learning models are:
  - Linear regression
  - Logistic regression
  - Generalized Linear Models (GLM)
  - Generalized Additive Models (GAM)
  - Linear Discriminant Analysis (LDA) and
    Quadratic Discriminant Analysis (QDA)
  - Naive Bayes methods
  - Mixture models (e.g. Gaussian mixtures)
  - Decision Trees (Regression and Classification Trees)
  - Ensemble methos (Bagging, Random Forests, Boosting)
  - Support Vector Machines (SVM)
  - Neural Networks (NN) and
    Deep Learning (DL)

- Suppose we collected a dataset $\mathcal{D}_{\text{train}} = \{\boldsymbol{x}_i\}_{i=1}^{n}$ composed of only a set of variables drawn from some unknown probability/density function

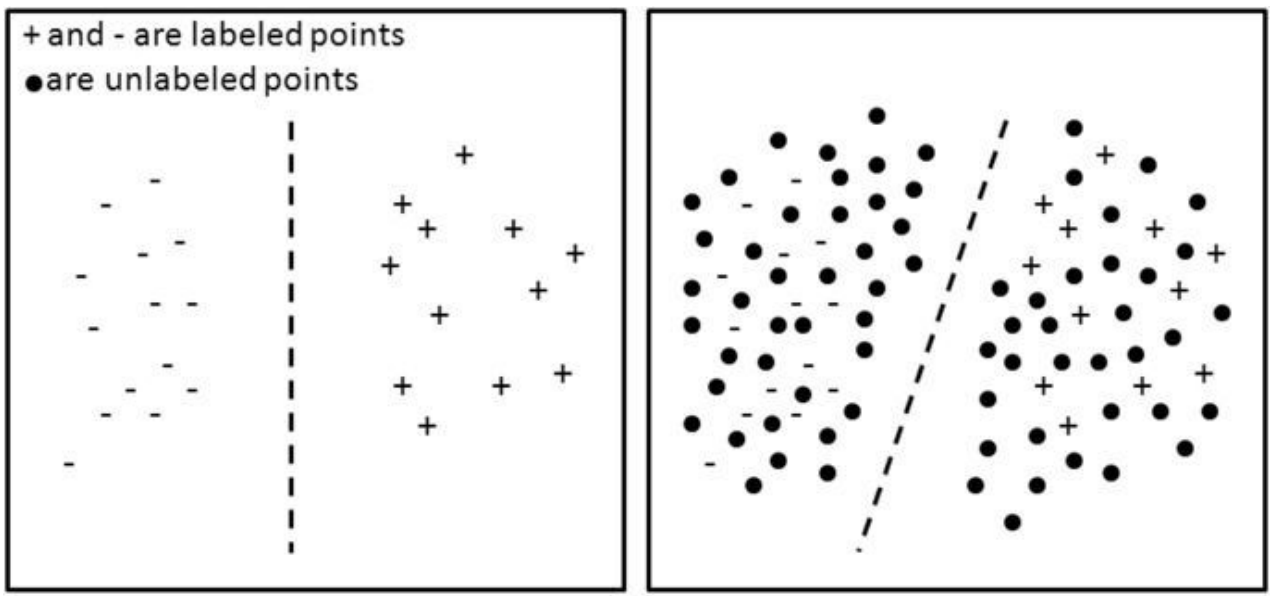
$$\boldsymbol{x}_i \sim p(\boldsymbol{x})$$

- In **unsupervised learning** for each case only the predictors vector $\boldsymbol{x}_i$ is observed, but there is no response y_i ($i = 1, \ldots, n$). Thus, we lack a response variable that can supervise our analysis.
- The aim is to estimate a model with parameters $\boldsymbol{\theta}$

$$\boldsymbol{x}_i \approx q(\boldsymbol{x} \mid \boldsymbol{\theta})$$

  where $q(.)$ is some working distribution depending on parameters $\boldsymbol{\theta}$.
- The learning task corresponds to finding the parameters that makes $q(.)$ as close as possible to the unknown $p(.)$ and from that understand the relationships between the variables or between the observations.
- Cluster analysis is a typical unsupervised learning task: look for the presence of one or more distinct groups of observations with no explicit assessment criterion because truth is not known (e.g. market segmentation to detect groups of customers).

- Many problems fall naturally into the supervised or unsupervised learning paradigms.
  However… sometimes the question is less clear-cut.
- There can be situations where for a subset of $m < n$ observations we have information on both the predictors and the response variable, and for the remaining $n - m$ observations we have only predictor measurements but no response measurement.



- Such a scenario is referred to as **semi-supervised learning**.

- Reinforcement learning focuses on training agents to make sequential decisions in an environment to maximize a cumulative reward.
- In reinforcement learning, an agent interacts with an environment, learns from its actions, and adjusts its behavior to achieve a specific goal or objective.

- Application domains:
  - recommendation systems (e-commerce websites, streaming services, ...)
  - robotics
  - game playing
  - autonomous vehicles

- In general, suppose that we have observed
  - a quantitative response (aka dependent variable, output, target, ...) denoted as Y, and
  - a set of p predictors (aka independent variables, covariates, features, ...) collected in the input vector $X = (X_1, X_2, \ldots, X_p)^\top$.
- Further, assume there exists some relationship between Y and X, i.e.

$$Y = f(X) + \epsilon$$

  where
  - $f()$ is unknown and represents the systematic information that X provides about Y;
  - ϵ is a random error term, which captures measurement errors and other discrepancies, independent of X and with zero mean.
- There are two main reasons to estimate $f()$:
  1. inference
  2. prediction

Inference

- In descriptive or explanatory modelling we want to understand how Y changes as a function of $(X_1, \ldots, X_p)$.
- Interesting questions:
  - Which predictors are associated with the response?
  - What is the relationship between the response and each predictor?
  - What is the functional form of the relationship between Y and each predictor?
  - There exists any cause-and-effect relationship? (causal inference)

Prediction

- In predictive modelling the goal is to predict the response variable based on the observed values of the predictors:

$$\widehat{Y} = \widehat{f}(X)$$

- $\widehat{f}()$ is often treated as a **black box**: we are not interested in knowing the exact form of $f()$, provided that it yields accurate predictions for Y.
- The prediction error of estimating Y using $\widehat{Y}$ can be decomposed as

$$Y - \widehat{Y} = f(X) + \epsilon - \widehat{f}(X) = (f(X) - \widehat{f}(X)) + (Y - f(X))$$

- Suppose that both $f()$ and X are fixed, then recalling that $\mathrm{E}[\epsilon] = 0$, the expected prediction error (under squared error loss) is given by

$$\begin{aligned}
\mathrm{E}[(Y - \widehat{Y})^2] &= \mathrm{E}[(f(X) + \epsilon - \widehat{f}(X))^2] \\
&= \mathrm{E}[(f(X) - \widehat{f}(X))^2] + \mathrm{E}[\epsilon^2] + 2\,\mathrm{E}[\epsilon(f(X) - \widehat{f}(X))] \\
&= \underbrace{(f(X) - \widehat{f}(X))^2}_{\text{reducible error}} + \underbrace{\mathrm{V}[\epsilon]}_{\text{irreducible error}}
\end{aligned}$$

Estimating $f()$

- Estimation (or learning in ML) is the process of applying a statistical/machine learning method to the training data to estimate the unknown function $f()$.
- Main goal: estimate $f()$ with the aim of minimizing the reducible error.
- The irreducible error provides a lower bound on the accuracy of our prediction for Y, and it is almost always unknown in practice.
- Several approaches are available, both parametric and non-parametric.

Parametric methods

A two-step model-based approach:

1. Select the functional form, or shape, of $f()$.
   For example, the **linear model** assumes that $f()$ is linear in X:
   $$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$
2. Select a procedure that uses the training data to fit or train the model.
   For example, in the linear model case we only need to **estimate the parameters** $(\beta_0, \beta_1, \beta_2, \ldots, \beta_p)$. A popular approach is (ordinary) least squares (OLS), but many other exists (maximum likelihood, regularized ML, Bayesian estimation, ...).

- This model-based approach is called parametric because it reduces the problem of estimating $f()$ down to one of estimating a set of parameters (the coefficients of the model).

Pros: generally is much easier to estimate a set of parameters than it is to fit an entirely arbitrary function $f()$.

Cons: the selected model can be a poor approximation of true unknown form of $f()$.

Non-parametric methods

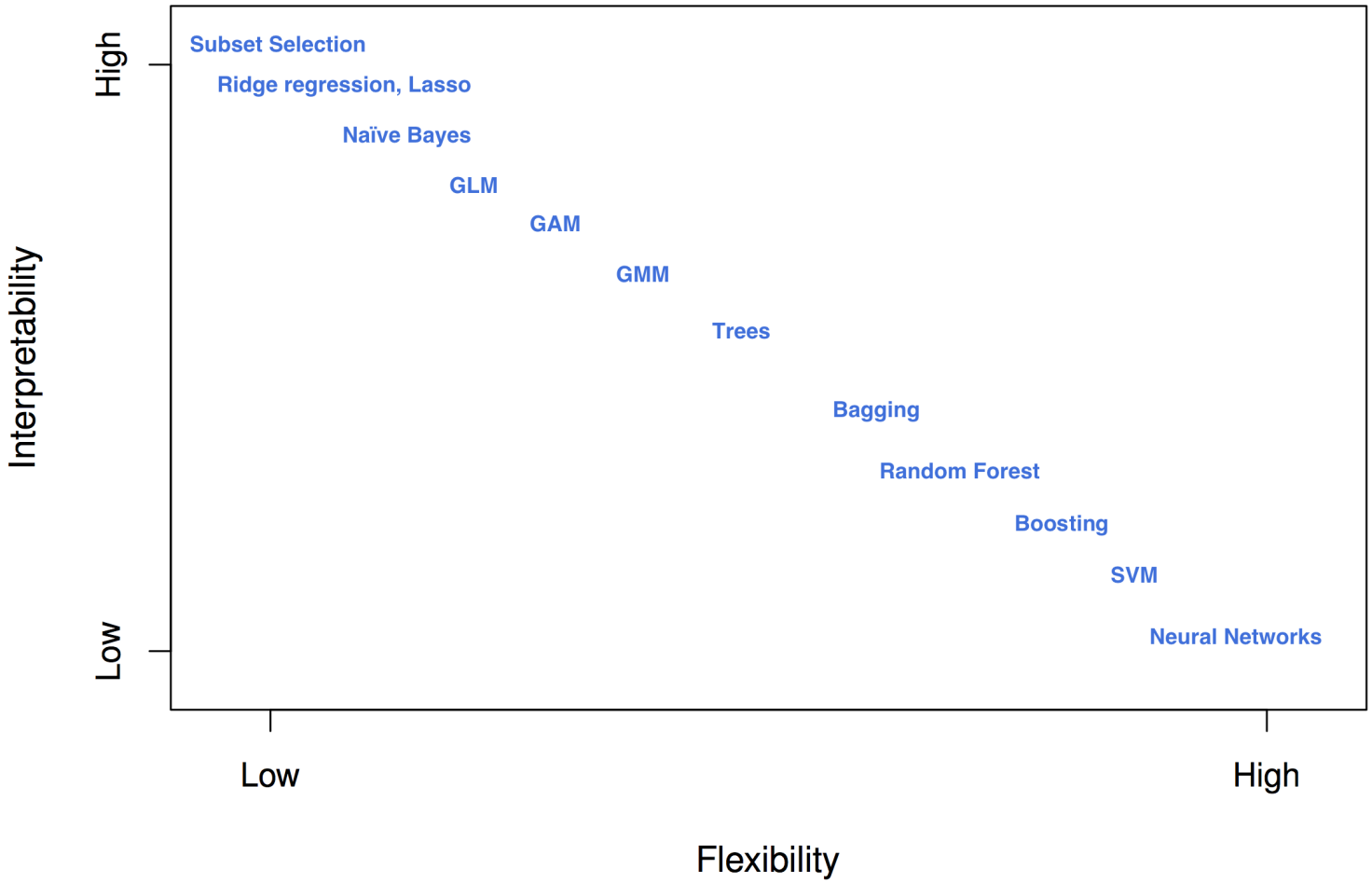
- Non-parametric methods do not make explicit assumptions about the functional form of $f()$.
- They try to estimate $f()$ getting as close to the data points as possible without being too rough or wiggly.

Pros: avoid the assumption of a particular functional form for $f()$, so they have the potential to accurately fit a wider range of possible shapes for $f()$.

Cons: a large number of observations is required to accurately estimate $f()$.

Trade-off between model interpretability and flexibility

- If we are mainly interested in explanatory inference, then simple models (e.g. Linear Models, Logistic Regression) are much more interpretable than black-box models (e.g. Random Forest, SVM, Neural Networks).
- Flexible models allow to fit many different possible functional forms for $f()$, but usually require estimating a larger number of parameters.
- In general, as the flexibility of a model/algorithm increases, its interpretability decreases.
- **Overfitting** is the main risk, i.e. to follow the observed data (including the error/noise component) too closely.
- If we are only interested in prediction, then the interpretability of the predictive model may be simply not of interest.
- Flexible models may provide good fit but there is the risk of overfitting.
- Models involving fewer variables are often preferred over more complicated models involving several variables or features.



Interpretability vs flexibility using different statistical/machine learning methods

- Suppose we have fit a model $f(x)$ to some training data $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$, and we wish to assess its performance.
- Compute the **average squared prediction error** over $\mathcal{D}_{\text{train}}$:

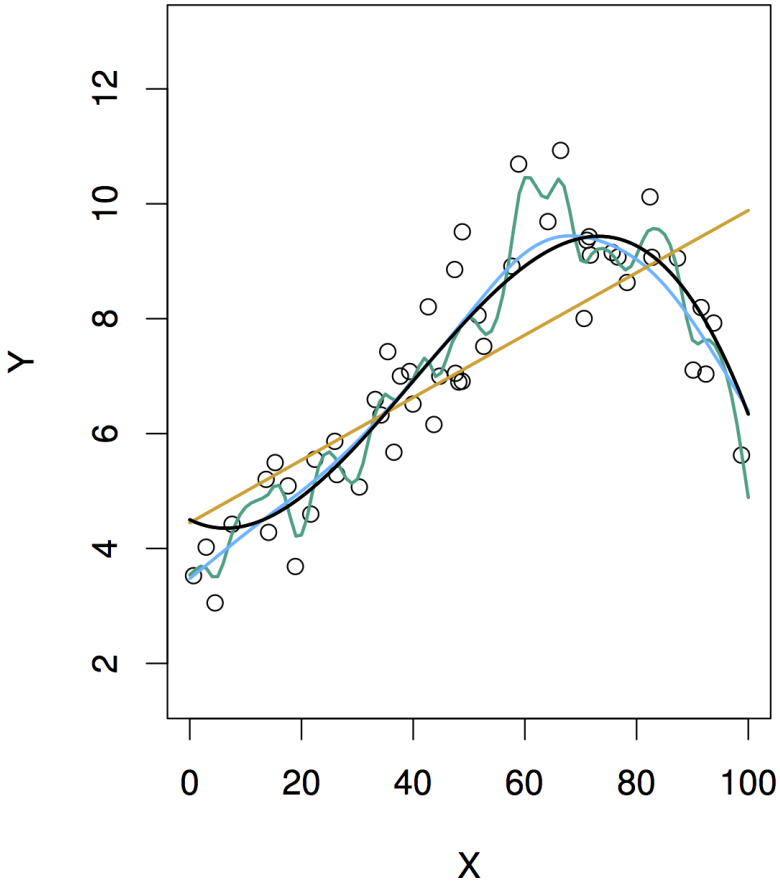
$$\text{MSE}_{\text{train}} = \frac{1}{n} \sum_{i \in \mathcal{D}_{\text{train}}} [y_i - \widehat{f}(\boldsymbol{x}_i)]^2$$

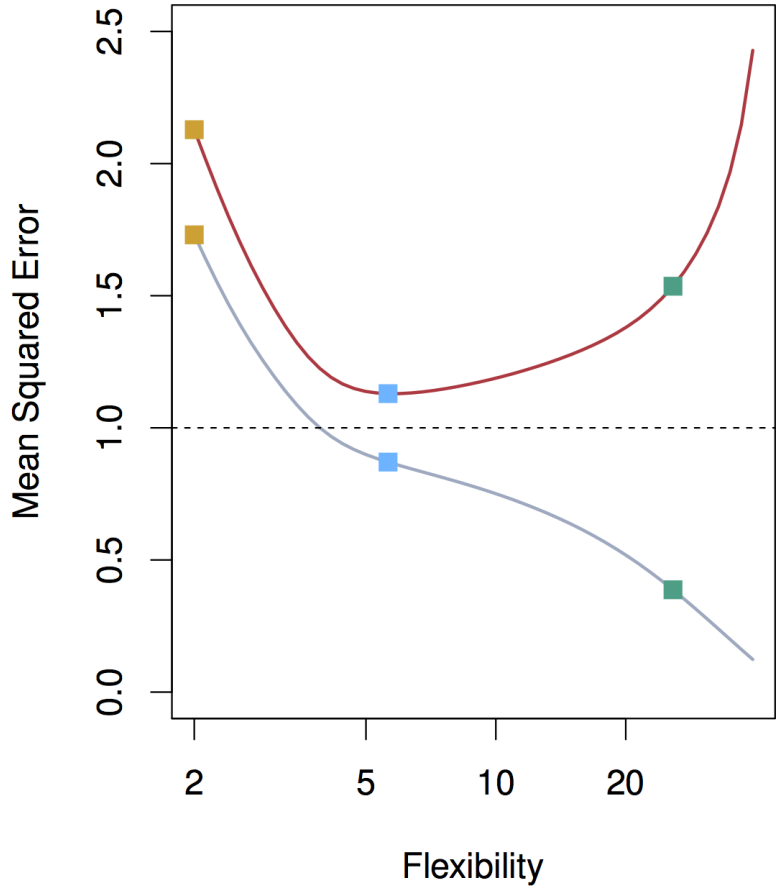
  - Since the same data is used both for "learning" and for "evaluating" the fit of a model, this gives an optimistic evaluation of model accuracy.
  - If used for selecting the complexity of a statistical model, it is biased toward overfitting models.

- Compute the MSE on a **test set** $\mathcal{D}_{\text{test}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m}$, i.e. a fresh dataset not used for parameters estimation:

$$\text{MSE}_{\text{test}} = \frac{1}{m} \sum_{i \in \mathcal{D}_{\text{test}}} [y_i - \widehat{f}(\boldsymbol{x}_i)]^2$$

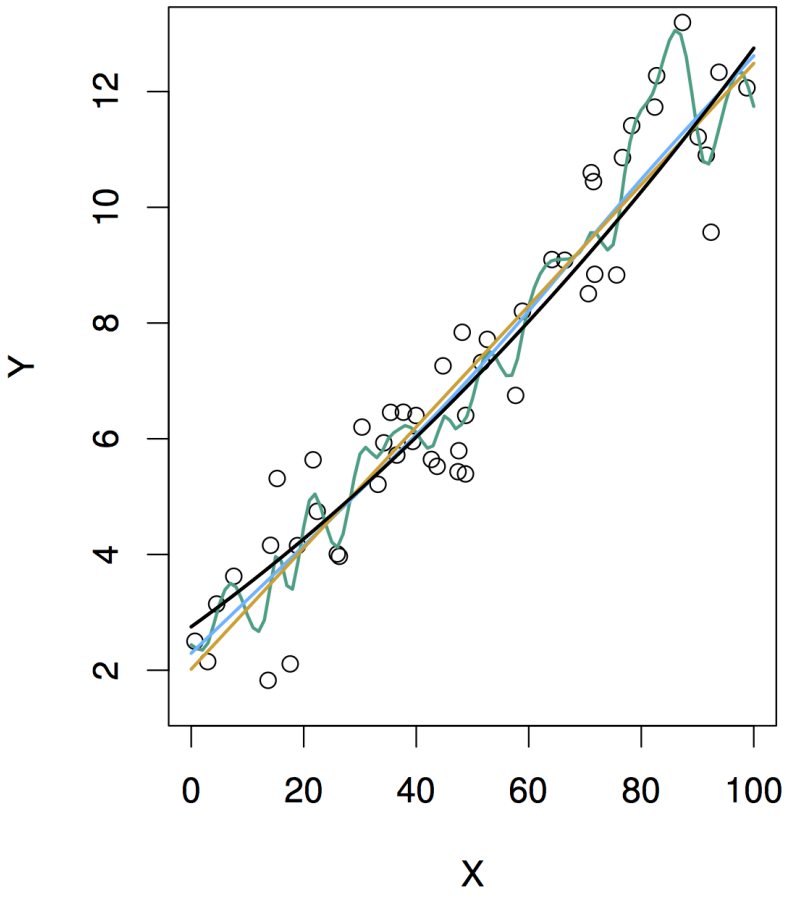
  - This is a more realistic measure of how accurately an algorithm is able to predict outcome values for previously unseen data.

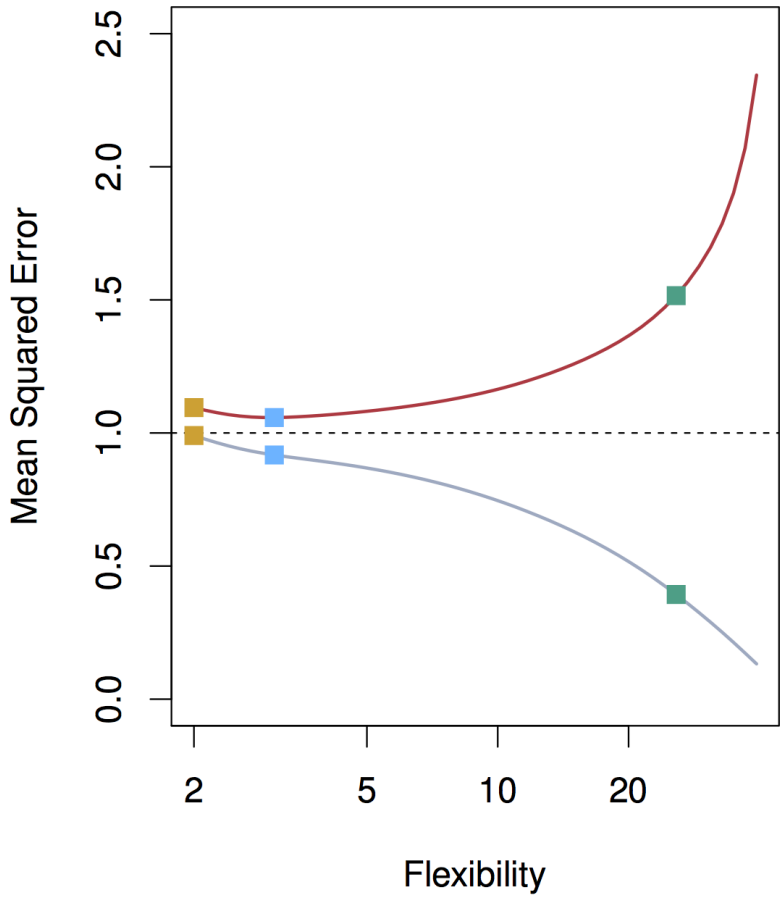




Models: 1) true model —— 2) linear regression fit —■— 3) smoothing spline fit with medium flexibility —■— 4) smoothing spline fit with high flexibility —■—

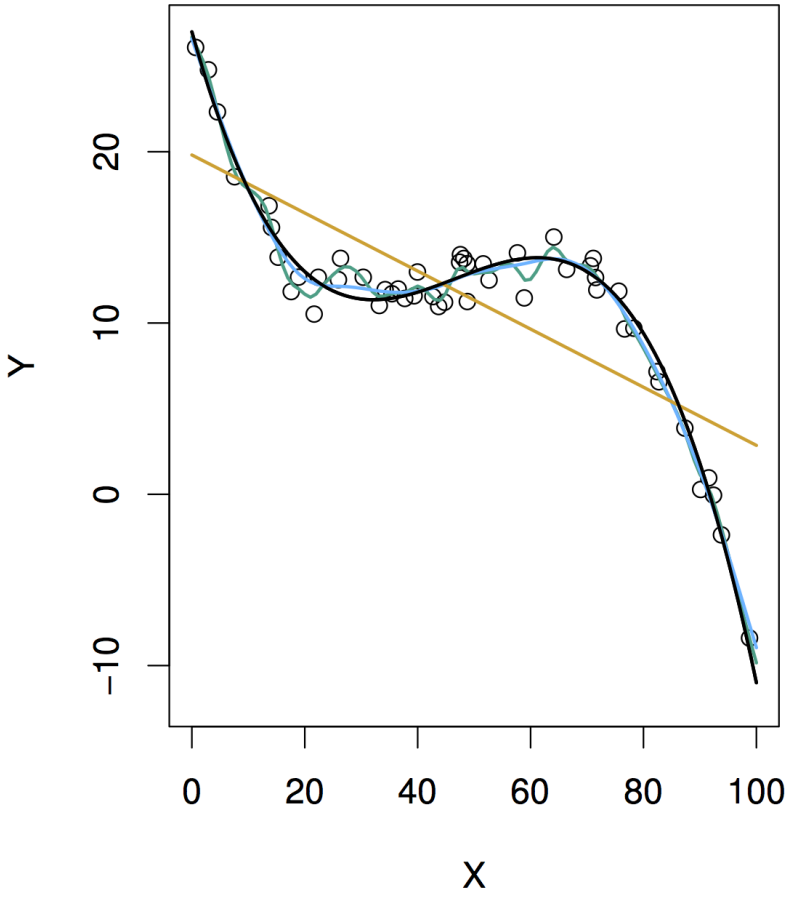
Accuracy measure: (i) training MSE —— (ii) test MSE —— (iii) irreducible MSE – – –

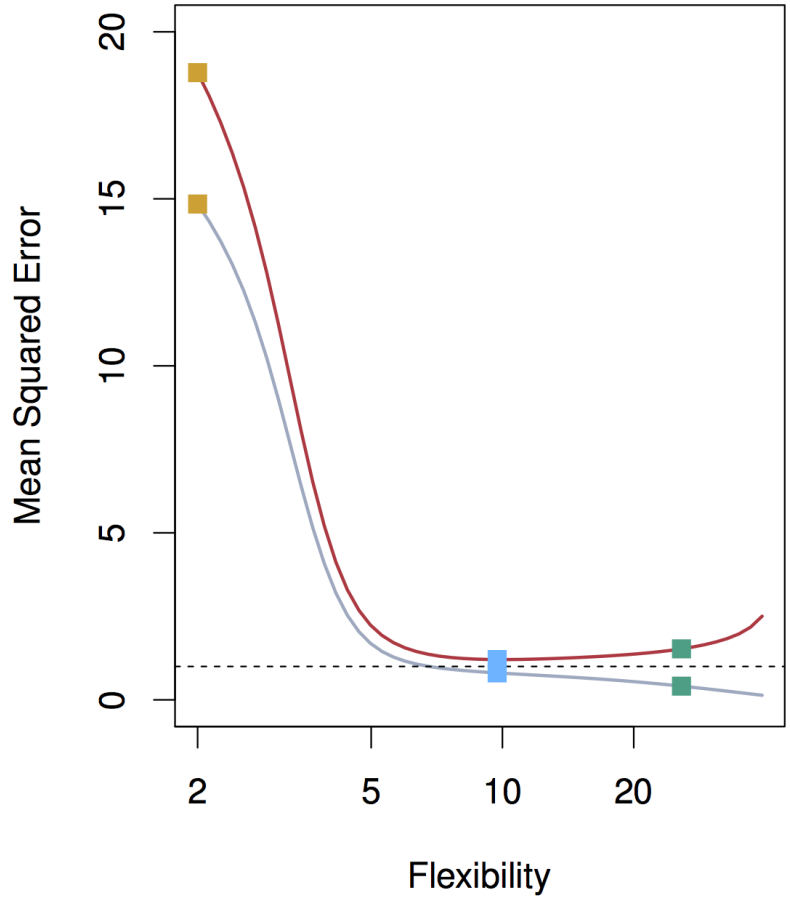




Models: 1) true model —— 2) linear regression fit —■— 3) smoothing spline fit with medium flexibility —■— 4) smoothing spline fit with high flexibility —■—

Accuracy measure: (i) training MSE —— (ii) test MSE —— (iii) irreducible MSE – – –





Models: 1) true model —— 2) linear regression fit —■— 3) smoothing spline fit with medium flexibility —■— 4) smoothing spline fit with high flexibility —■—

Accuracy measure: (i) training MSE —— (ii) test MSE —— (iii) irreducible MSE – – –

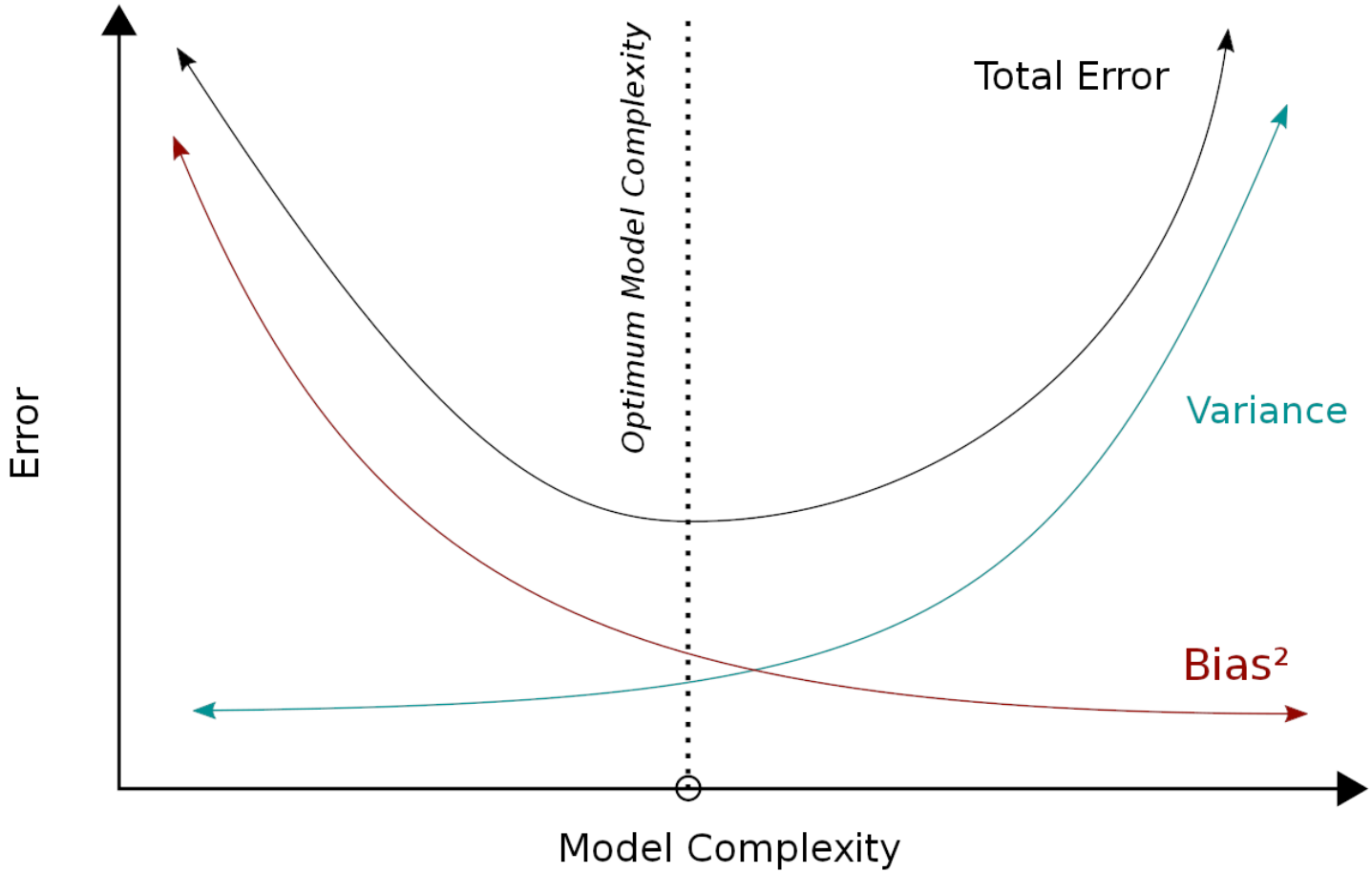
- The expected test error for a new observation value $\boldsymbol{x}_0$ can always be decomposed as

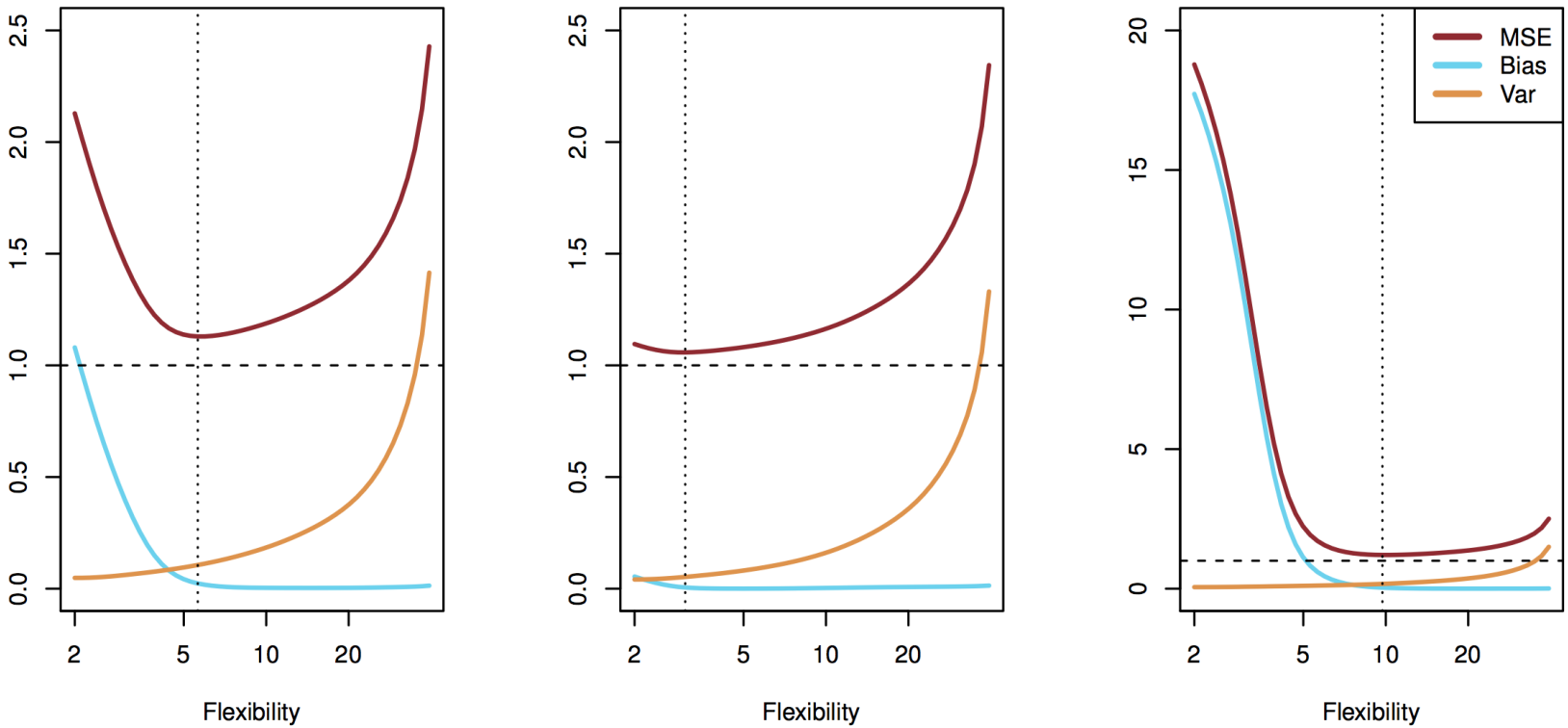
$$\mathrm{E}[y_0 - \widehat{f}(\boldsymbol{x}_0)]^2 = \mathrm{V}[\widehat{f}(\boldsymbol{x}_0)] + \mathrm{B}[\widehat{f}(\boldsymbol{x}_0)]^2 + \mathrm{V}[\epsilon]$$

  where

  - $\mathrm{V}[\widehat{f}(\boldsymbol{x}_0)]$ is the variance expressing the amount by which $\widehat{f}()$ would change if we estimated it using a different training dataset;
  - $\mathrm{B}[\widehat{f}(\boldsymbol{x}_0)]$ is the bias expressing the error that is introduced by approximating the data distribution by a statistical model;
  - $\mathrm{V}[\epsilon]$ is the irreducible error.

- The expected test error can never be smaller than the irreducible error.
- In general, more flexible statistical methods have higher variance and smaller bias. On the contrary, simpler models have smaller variance but higher bias.
- To minimize the expected test error, we need to select a statistical/machine learning method that simultaneously achieves low variance and low bias.


Error
Optimum Model Complexity
Total Error
Variance
Bias²
Model Complexity



MSE and the bias-variance trade-off

- Suppose that we seek to estimate $f()$ on the basis of the training observations $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$, where in this case $y_i \in \{C_1, C_2, \ldots, C_K\}$ is the class or label associated with the ith observation.

- **Training classification error rate** is the proportion of misclassified observations, i.e.

$$\text{CE}_{\text{train}} = \frac{1}{n} \sum_{i \in \mathcal{D}_{\text{train}}} \mathbb{1}(y_i \neq \widehat{y}_i)$$

  where
  - $\widehat{y}_i$ is the predicted class label for the ith observation using $\widehat{f}()$;
  - $\mathbb{1}(y_i \neq \widehat{y}_i)$ is the indicator function that returns 1 if $y_i \neq \widehat{y}_i$ and 0 otherwise.

- The **test classification error rate** associated with a set of test observations $\mathcal{D}_{\text{test}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m}$ is given by

$$\text{CE}_{\text{test}} = \frac{1}{m} \sum_{i \in \mathcal{D}_{\text{test}}} \mathbb{1}(y_i \neq \widehat{y}_i)$$

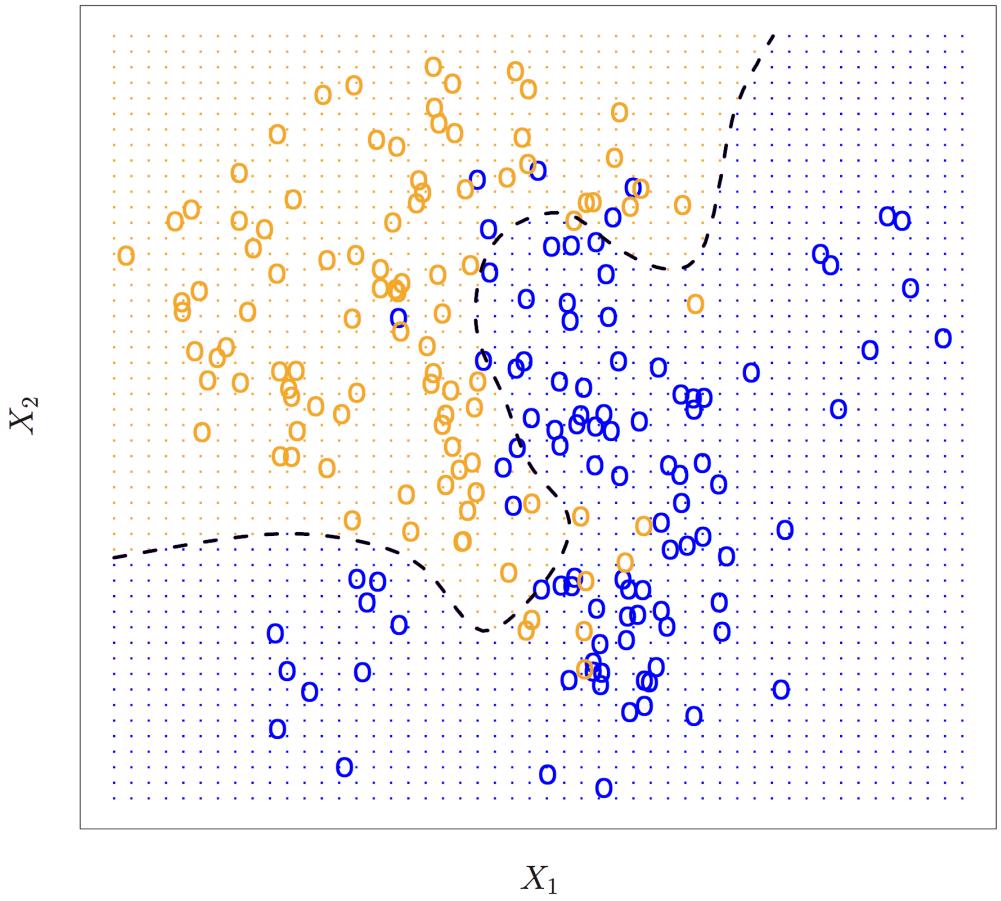
The Bayes Classifier

- The test error rate CE_{test} is minimized, on average, by a very simple classifier that assigns each observation to the most likely class, given its predictor values.
- According to the **Bayes classifier**, a test observation with predictor vector $\boldsymbol{x}_0$ should be assigned to the class C_k (with $k = 1, \ldots, K$) for which

$$\Pr[Y = C_k \mid \boldsymbol{x}_0] \qquad \text{is maximum}$$

- The Bayes error rate is the lowest possible test error rate produced by the Bayes classifier:
  - error rate at $\boldsymbol{x}_0$ is $1 - \max_k \Pr[Y = C_k \mid \boldsymbol{x}_0]$.
  - overall Bayes error rate is $1 - \text{E}\left[\max_k \Pr[Y = C_k \mid X]\right]$

  The Bayes error rate is analogous to the irreducible error for classification tasks.
- The Bayes decision boundary defines the regions in which a test observation will be assigned to one of the K classes.



Two-class simulated dataset. The dashed line represents the Bayes decision boundary with Bayes error rate $\approx$ 13%

- For real data, we do not know the conditional distribution of Y given X, and so computing the Bayes classifier is impossible.

Loss functions

- In supervised machine learning problems, a training set of n data points is available, $\mathcal{D}_{\text{train}} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n}$, where $\boldsymbol{x}_i$ represents the p features on the ith observation and y_i represents the value of the response variable.
- The main goal is to build a model whose predictions $\widehat{y}_i$ are as close as possible to the true response values y_i.
- A **loss function** aims at measuring model's prediction error:

$$\mathcal{L} = \sum_{i=1}^{n} \mathcal{L}(y_i, \widehat{y}_i)$$

- Properties of a loss function:
  - continuous and differentiable (everywhere);
  - convex, i.e. only one global minimum point exists, so optimization methods like gradient descent are guaranteed to return the globally optimal solution. In practice, this is hard to achieve, and most loss functions are non-convex (i.e. they have multiple local minima);
  - symmetric, i.e. the error above the target should cause the same loss as the same error below the target;
  - computationally efficient, i.e. fast and scalable.

Loss Functions and Maximum Likelihood

- Many of the loss functions used in ML are derived from the maximum likelihood principle.
- In maximum likelihood estimation (MLE) we are trying to fit a model with parameters $\boldsymbol{\theta}$ that maximizes the probability of the observed data given the model: $\Pr(\mathcal{D}|\boldsymbol{\theta})$.
- MLEs are computed as

$$\widehat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \log \Pr(\mathcal{D}|\boldsymbol{\theta})$$

- Thus, the loss function for a random sample $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$ can be defined as

$$\mathcal{L} = -\log \Pr(\mathcal{D}|\boldsymbol{\theta}) = -\sum_{i=1}^{n} \log \Pr(\boldsymbol{x}_i|\boldsymbol{\theta})$$

  so $\mathcal{L}(y_i, \widehat{y}_i) = -\log \Pr(\boldsymbol{x}_i|\boldsymbol{\theta})$.
- Because negative logarithm is a monotonically decreasing function, maximizing the likelihood is equivalent to minimizing the loss.

Squared loss

- The squared loss is defined as

$$\mathcal{L}_{\text{sq}}(y_i, \widehat{y}_i) = (y_i - \widehat{y}_i)^2$$

- This is the loss function used in ordinary least squares (OLS), the most common method for solving linear regression problems.
- Pros:
  - continuous and differentiable everywhere;
  - convex (has only one global minimum);
  - easy to compute;
  - obtained assuming a Gaussian distribution for the errors.

  Cons:
  - sensitive to outliers.

Absolute loss

- The absolute loss is defined as

$$\mathcal{L}_{\text{abs}}(y_i, \widehat{y}_i) = |y_i - \widehat{y}_i|$$

- Pros:
  - not overly affected by outliers;
  - easy to compute;
  - obtained assuming a Laplace distribution for the errors.

  Cons:
  - non-differentiable at 0, which makes it hard to be used by derivative-based optimization methods, such as gradient descent.

Huber loss

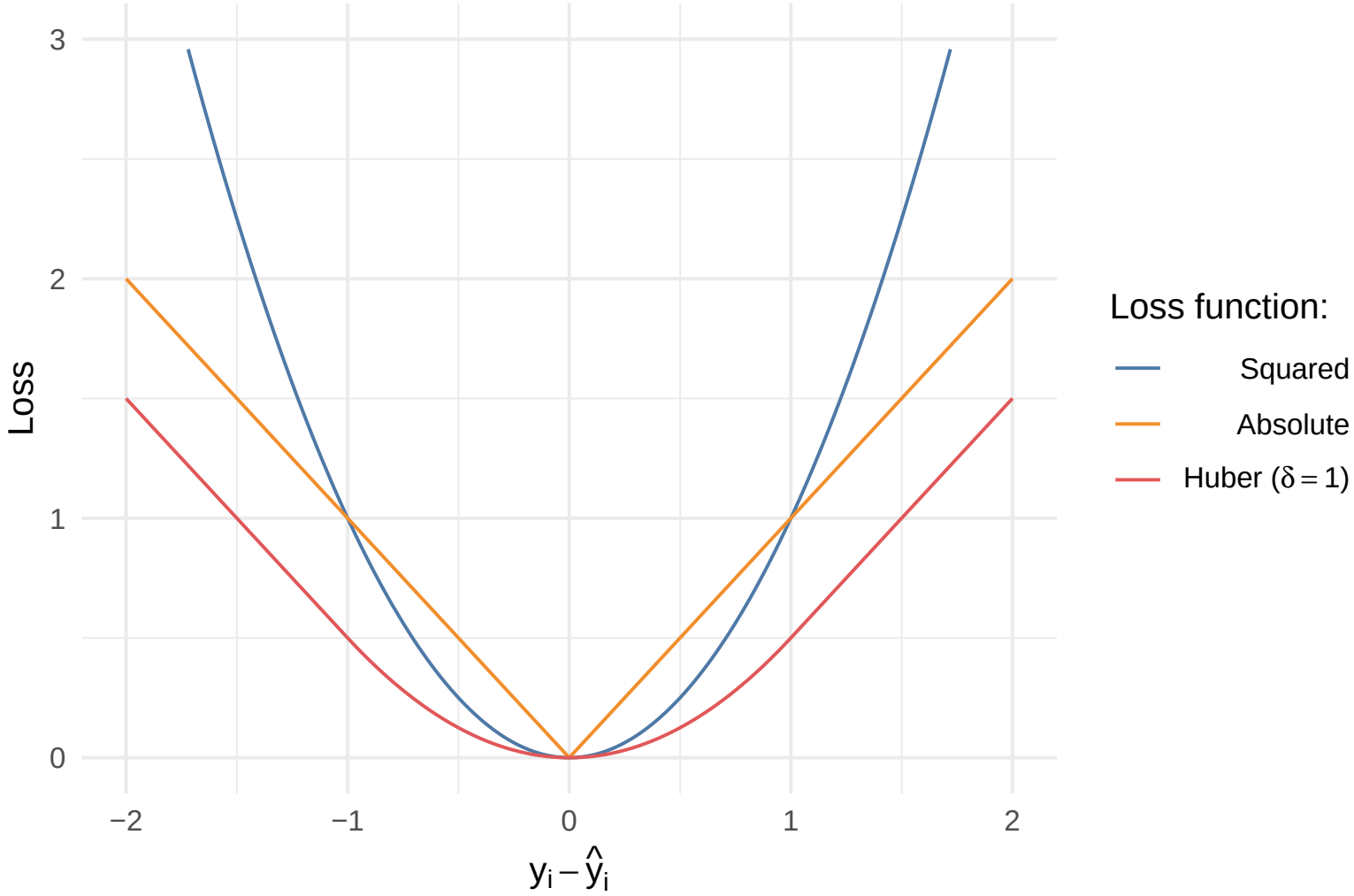
- The Huber loss is a combination of squared loss and absolute loss and it is defined as

$$\mathcal{L}_{\text{Huber}}(y_i, \widehat{y}_i) = \begin{cases} \frac{1}{2}(y_i - \widehat{y}_i)^2 & \text{if } |y_i - \widehat{y}_i| \leq \delta \\ \delta(|y_i - \widehat{y}_i| - \frac{1}{2}\delta) & \text{if } |y_i - \widehat{y}_i| > \delta \end{cases}$$

  for some hyperparameter $\delta > 0$.

- Pros:
  - continuous and differentiable everywhere;
  - less sensitive to outliers than squared loss.

  Cons:
  - slower to compute;
  - requires tuning of the hyperparameter δ;
  - does not have a maximum likelihood interpretation.


3
2
1
0
Loss
−2
−1
0
1
2
$y_i - \hat{y}_i$
Loss function:
Squared
Absolute
Huber ($\delta = 1$)

Zero-One loss

- The simplest loss function is the zero-one loss function defined as

$$\mathcal{L}_{01}(y_i, \widehat{y}_i) = \mathbb{1}(y_i \neq \widehat{y}_i)$$

  where $y_i \in \{0,1\}$ is the observed class, $\widehat{y}_i$ the corresponding predicted class, and $\mathbb{1}()$ the indicator function that returns 1 if its argument is true, and 0 otherwise.

- By encoding $y_i \in \{-1, +1\}$, the zero-one loss can also be defined as

$$\mathcal{L}_{01}(y_i, s_i) = \mathbb{1}(y_i s_i < 0)$$

  where $s_i \in \mathbb{R}$ is the linear score s.t. $\Pr(y_i = +1) = 1/(1 + \exp(-s_i))$.

- This loss function counts the number of prediction errors made by the classifier (misclassification error).

- Pros:
  - easy to compute.

  Cons:
  - non-differentiable and non-continuous.

Log loss or Cross-entropy loss

- Denote the binary response as $y_i \in \{0,1\}$ and the probability of positive case as $\Pr(y_i = 1) = p_i$, then the log loss is defines as

$$\mathcal{L}_{\log}(y_i, p_i) = -y_i \log(p_i) - (1 - y_i) \log(1 - p_i)$$

- Equivalently, denoting the binary response as $y_i \in \{-1, +1\}$ and $s_i \in \mathbb{R}$ the linear score, the log loss can also be defined as

$$\mathcal{L}_{\log}(y_i, s_i) = \log(1 + \exp(-y_i s_i))$$

- Pros:
  - continuous and differentiable everywhere;
  - convex (has only one global minimum);
  - obtained assuming a Bernoulli distribution for the response variable;
  - loss function used in logistic regression;
  - easily extended to multi-class classification problems.

  Cons:
  - symmetric.

Hinge loss

- For the binary response $y_i \in \{-1, +1\}$, the hinge loss is defined as

$$\mathcal{L}_{\text{hinge}}(y_i, s_i) = \max(0, 1 - y_i s_i)$$

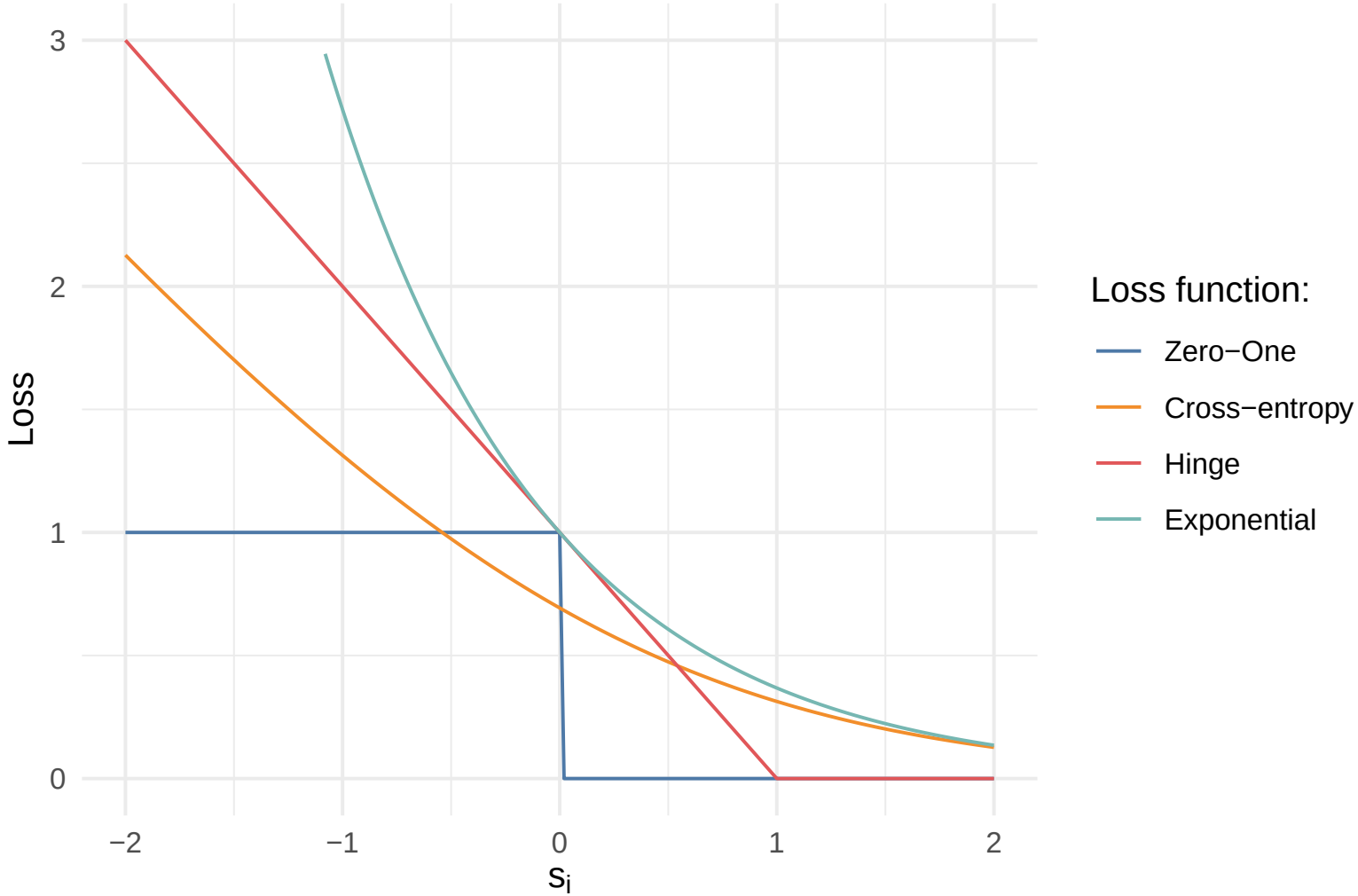
- Hinge loss is employed by support vector machines (SVM) to obtain a classifier with "maximal margin":
  - when y_i and s_i have the same sign (a correct prediction) and $s_i \geq 1$ the loss is 0;
  - when y_i and s_i have opposite signs, the loss increases linearly with s_i, and similarly if $s_i < 1$, even if it has the same sign (a correct prediction, but not by enough margin).

Exponential loss

- For the binary response $y_i \in \{-1, +1\}$, the exponential loss is defined as

$$\mathcal{L}_{\text{exp}}(y_i, s_i) = \exp(-y_i s_i)$$

- A more aggressive loss function which grows exponentially for negative values and is thus very sensitive to wrong predictions.
- Exponential loss is employed by AdaBoost classifier.


3
2
1
0
Loss
−2
−1
0
1
2
s_i
Loss function:
Zero−One
Cross−entropy
Hinge
Exponential

- Often models require the tuning of hyper-parameters (k in KNN, number of components in GMM, smoothing parameter, lasso and/or ridge parameters, number of hidden layers and number of nodes in ANNs, etc.).
- Sometimes we have no test data available for estimating MSE, classification error, etc.
- In all these cases, a separated validation dataset $\mathcal{D}_{\text{val}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{V}$ should be used.
- However, instead of setting aside a validation set, it is preferable to use resampling methods.
- No free lunch theorem: *no method/algorithm/model dominates all others over all possible datasets.*
- Realistically, we should decide for any given set of data which method produces the best results.
- This is the most challenging part of statistical/machine learning in practice.

Resampling

- Resampling methods are a fundamental tool in modern statistics.
- They involve repeatedly drawing samples from a training set and refitting a model of interest on each sample to obtain additional information about the fitted model.
- They can be computationally expensive, because the same statistical model must be fitted multiple times using different subsets of the training data.
- Goals
  - Model assessment (evaluating model's performance)
  - Model selection (selecting the level of flexibility of a model, i.e. hyperparameters tuning)
  - Model inference (provide a measure of accuracy of a parameter estimate or of a given statistical/machine learning method)

- Several possible **performance metrics** can be adopted.
- For regression problems, the error is usually measured by the **root mean square error**:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_i (y_i - \widehat{f}(x_i))^2}$$

  or directly using the MSE.
- For classification problems, the error can be measured by the **classification error**:

$$\text{CE} = \frac{1}{n}\sum_i \mathbb{1}(y_i \neq \widehat{y}_i)$$

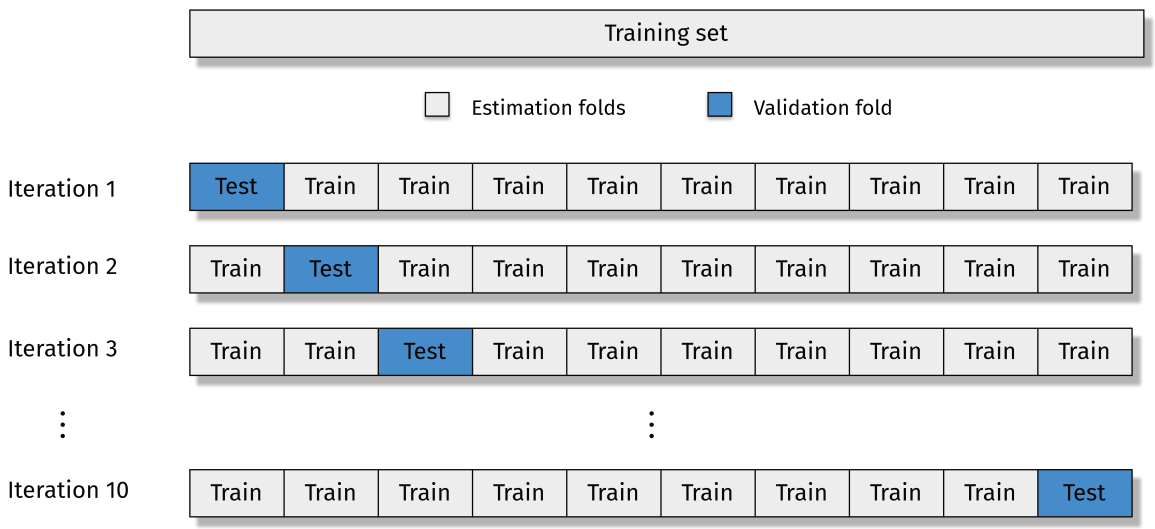
  Many other measures are available: sensitivity/specificity, ROC-AUC, precision/recal, F-score, log-loss or cross-entropy, Brier score, etc.
- If a validation set is not available, an estimate of the true error must be obtained by resampling methods.

- Cross-validation is a widely used resampling approach for estimating the performance of a statistical/machine learning model/algorithm.

V-fold cross-validation

The set of training observations is randomly splitted into V parts or folds. The model is trained using all but the vth fold, then the remaining vth fold is used as validation set. This is done in turn for each fold $v = 1, \ldots, V$, and then the results are combined.

10-fold cross-validation scheme



- When $V = n$, the procedure is called **leave-one-out cross-validation** (LOOCV), because we leave out one data point at a time.

Bootstrap

- The **bootstrap** is a flexible and powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator (e.g. standard errors or confidence intervals for regression coefficients) or the predictions provided by a statistical/machine learning method.
- Bootstrap takes random samples with replacement of the same size as the original data set.
- Since sampling is made with replacement, some observations may be selected more than once and each observation has a 63.2% chance of showing up at least once.

  The probability for an observation of not being selected in any of n draws from n samples with replacement is $(1 - 1/n)^n$.
  Then $\lim_{n\to\infty}(1-1/n)^n = e^{-1} \approx 0.368$, *and the probability of being selected at least once is* $1 - e^{-1} \approx 0.632$.
- The observations not selected (approximately 1/3 of the sample) are usually referred to as the out-of-bag observations.

Bootstrap algorithm for a classification task

1. Create a bootstrap sample by random sampling with replacement;
2. Fit a classifier using the bootstrap sample as training set;
3. Predict out-of-bag observations to get bootstrap classification error;
4. Repeated steps 1-3 multiple times (usually 30 – 100) and then combine the results.

Bias-variance trade-off for bootstrap

- The bootstrap estimates of error rate have less variability than V-fold CV, but larger bias (similar to 2-fold CV).
  If the training set size is small, this bias may be problematic, but will decrease as the training set sample size becomes larger.
- The "632" bootstrap method tries to reduce the bias by creating a performance estimate that is a combination of the simple bootstrap estimate and the estimate from predicting the training set:

$$(0.632 \times \text{bootstrap error rate}) + (0.368 \times \text{training error rate})$$

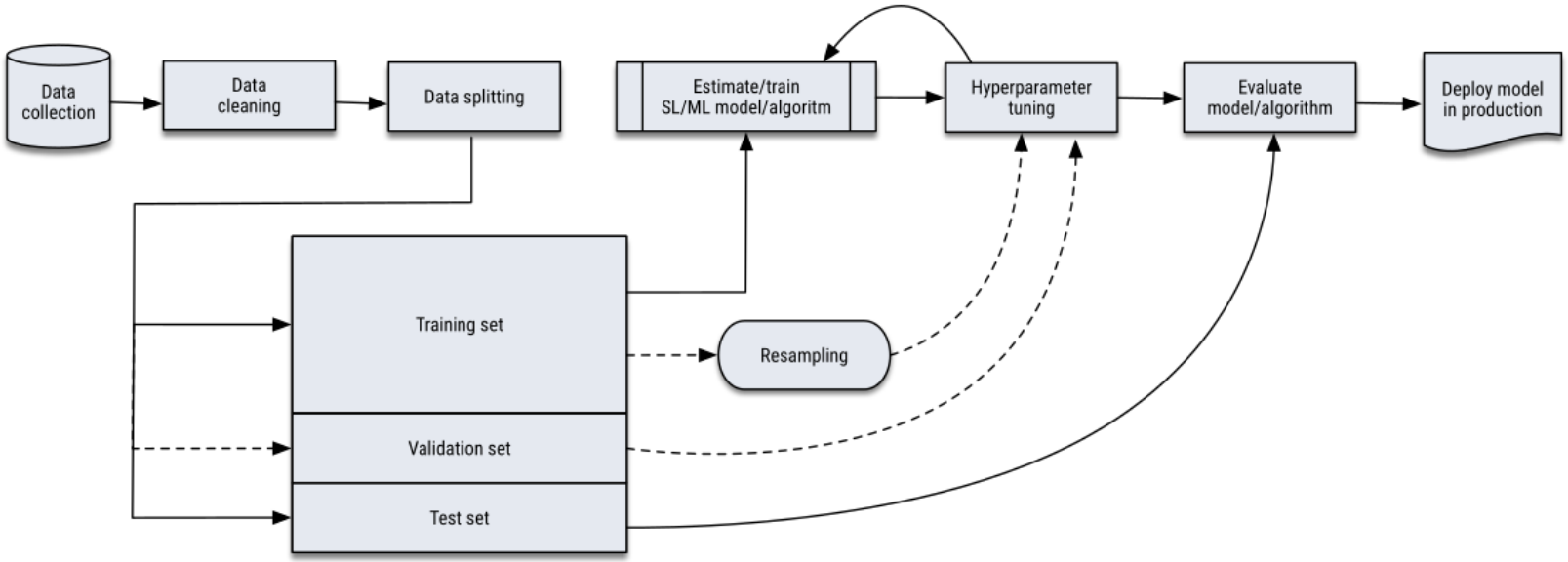
One standard error rule

- Instead of selecting the model with the "best" tuning parameter value, other schemes for selecting a single model can be used.
- A popular choice is the so-called "one standard error rule":

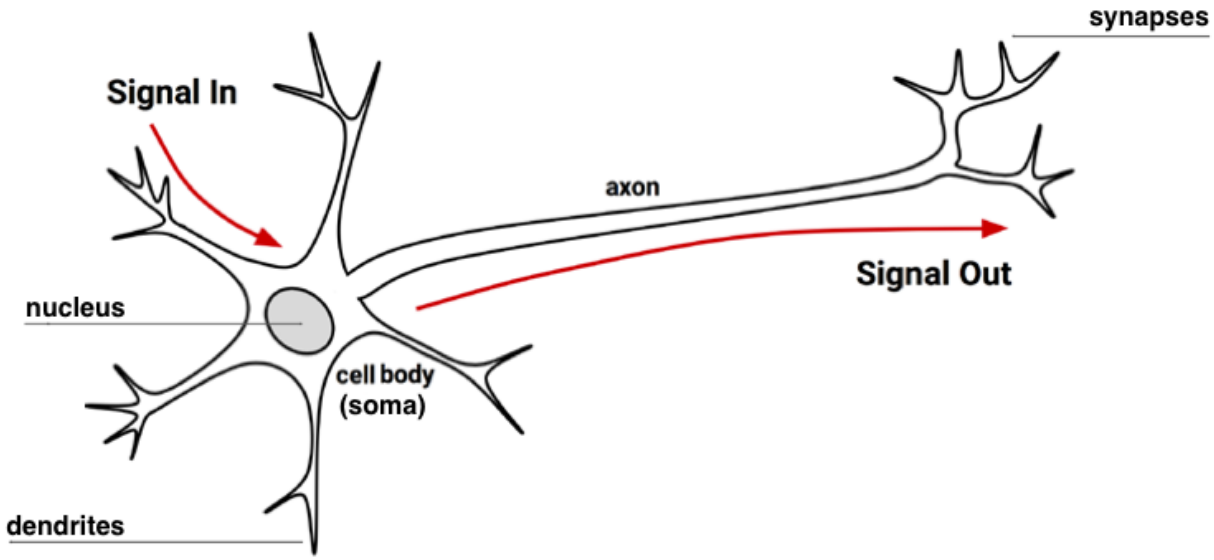
  "all else equal (up to one standard error), go for the simpler (more regularized/parsimonious) model"

- In practice:
  - the model with the best performance value is identified;
  - an estimate of the standard error of performance is computed by a resampling method;
  - the final model is the simplest model whose estimated performance is within one standard error from the best model performance.

Statistical/Machine Learning Pipeline



- **Artificial Neural Networks** (ANNs) are a broad class of biologically inspired nonlinear algorithms for solving both regression and classification problems.
- A brief history of ANNs:
  - 1940s: McCulloch-Pitts (1943) introduced the neuron as a simplified model of a biological neuron. This laid the theoretical foundation for ANNs.
  - 1950s: Rosenblatt (1957) developed the perceptron, a type of artificial neuron capable of binary classification.
  - 1960s - 1970s: limits of single-layer perceptrons were identified for solving complex problems that were not linearly separable (1st AI Winter).
  - 1980s: back-propagation algorithm was developed independently by multiple researchers (Werbos, Hinton, ...) which allowed for efficient training of multi-layer ANNs.
  - 1990s: limited progress and high expectations lead to 2nd AI Winter with decreasing funding.
  - 2010s - Present: advances in computing power, availability of large datasets, and introduction of deep learning techniques, led to a resurgence of interest and breakthroughs in neural networks.



A pictorial description of biological brain activity

1. Neuroelectric signals are received by the cell's dendrites through a biochemical process.
2. If a sufficient number of (cumulative) input signals enter the neuron through the dendrites, the cell body generates a response signal and transmits it down the axon to the synaptic terminals.
3. The specific number of input signals required for a response signal is dependent on the individual neuron.
4. When the generated signal reaches the synaptic terminals it flow out and interact with dendrites of neighboring neurons.

Several variants of ANNs exist, but all can be defined in terms of the following characteristics:

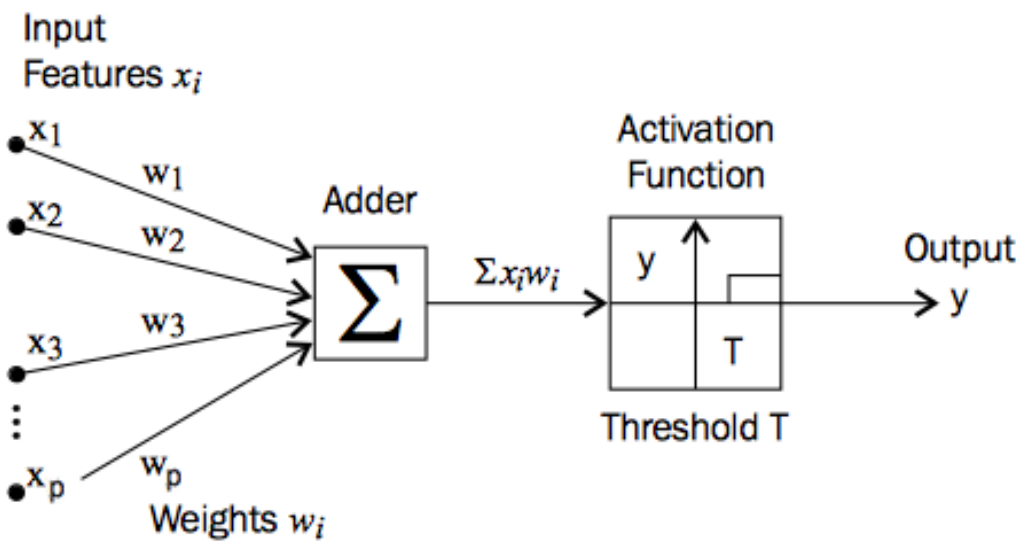
- An activation function, which transforms a neuron's combined input signals into a single output signal to be broadcasted further in the network.
- A network topology (or architecture), which describes the number of neurons in the model, the number of layers and their connections.
- The training algorithm that specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal.

- The specification of a simple ANN model is the following:

$$y \leftarrow f(x) = f\left(\alpha + \sum_{j=1}^{p} w_j x_j\right)$$

where

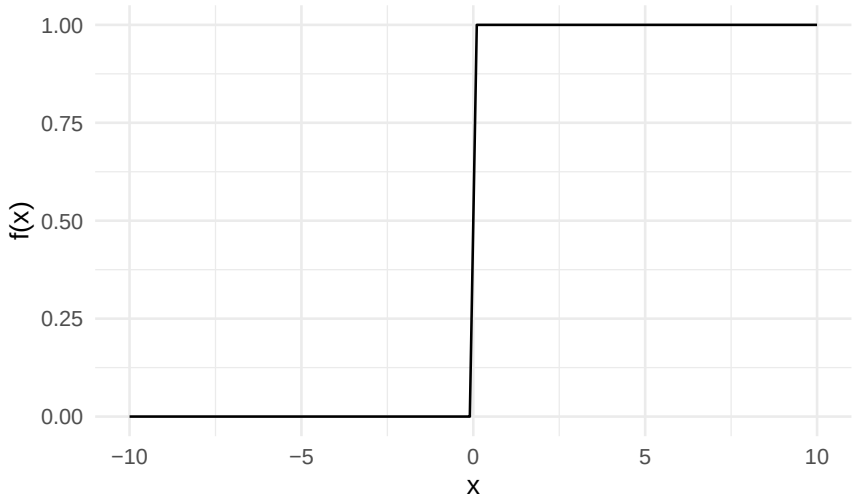
- y is the output signal (response variable)
- x_j are the input signals (features or predictors)
- w_j are the weights associated to input signals (coefficients)
- α is the "bias" (intercept)
- $f()$ is the activation function



Threshold/Unit step activation function

- A simple activation function that outputs a binary result based on whether the input is above or below a specified threshold:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

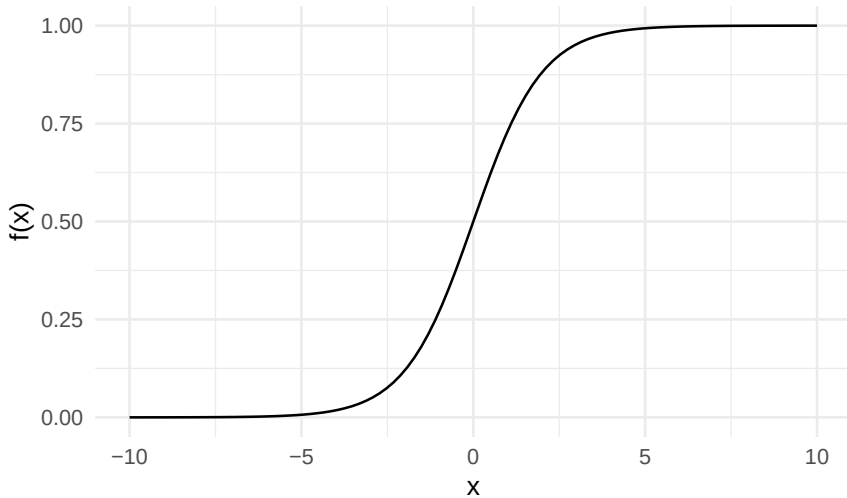


- Also called “unit step activation function”, it is rarely used in practice because the discontinuity at the threshold point makes the activation function non-differentiable. This impedes gradient availability for training using the back-propagation algorithm.

Sigmoid activation function

- An activation function that maps any real-valued number to the range $(0,1)$, and it is defined as

$$f(x) = \frac{1}{1+e^{-x}}$$

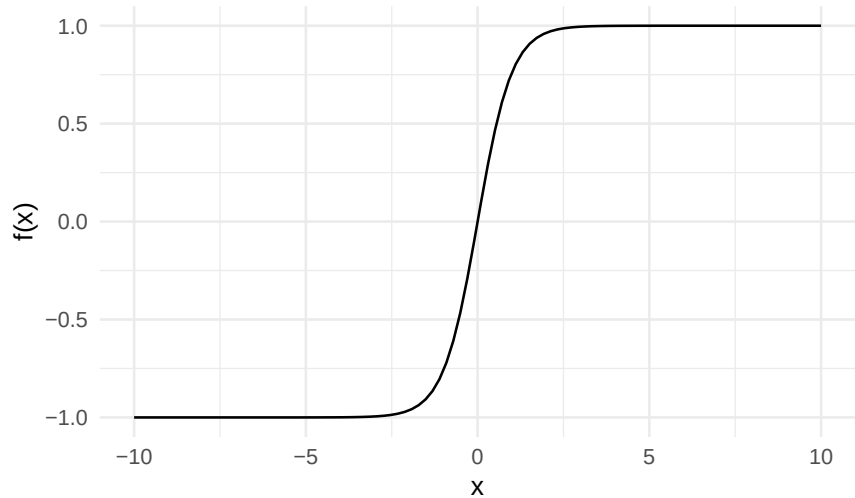


- In ANN the sigmoid function is commonly used in the hidden layers.
- It is particularly useful in situations where the output of a neuron needs to be constrained in $(0,1)$, e.g. as the output layer in tasks involving binary classification.
- The sigmoid function has limitations, especially in the context of deep learning due to so-called “vanishing gradient problem” which can lead to slower training and convergence.

Hyperbolic tangent (tanh) activation function

- An activation function maps any real-valued number to the range $(-1,1)$, and it is defined as

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



- The hyperbolic tangent function is often used as an activation function in the hidden layers of a ANN.
- It is particularly useful in zero-centered data scenarios.
- As for the sigmoid function, it suffers from the vanishing gradient problem.

Identity activation function

$f(x) = x$

Often used in regression contexts to link the last hidden layer with the output layer.

Rectified linear unit (ReLU)

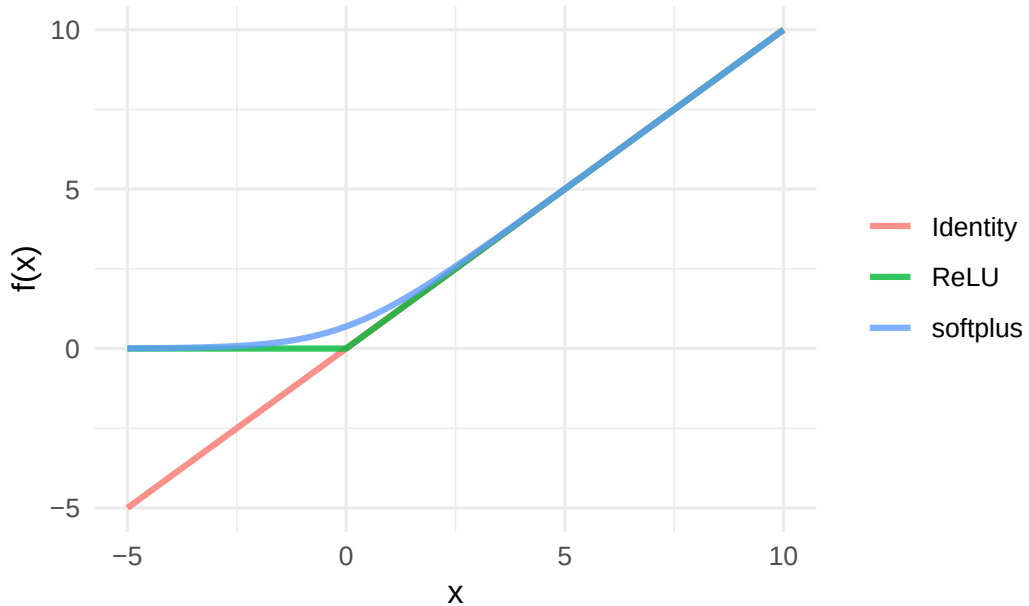
$f(x) = \max(0, x)$

This is a popular activation function for deep neural networks.

Softplus

$f(x) = \log(1 + e^x)$

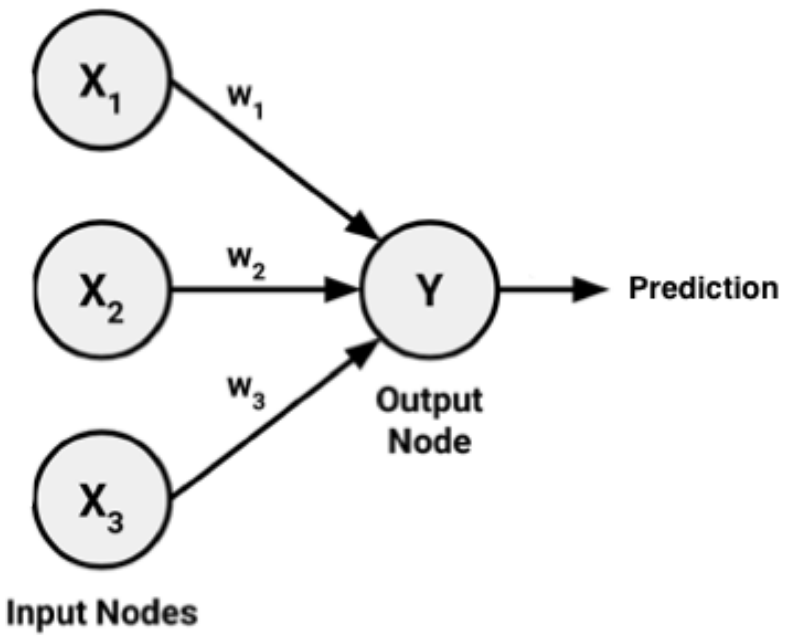
It is a smooth approximation to the ReLU activation function.



The input and output nodes are arranged in groups of nodes known as layers.

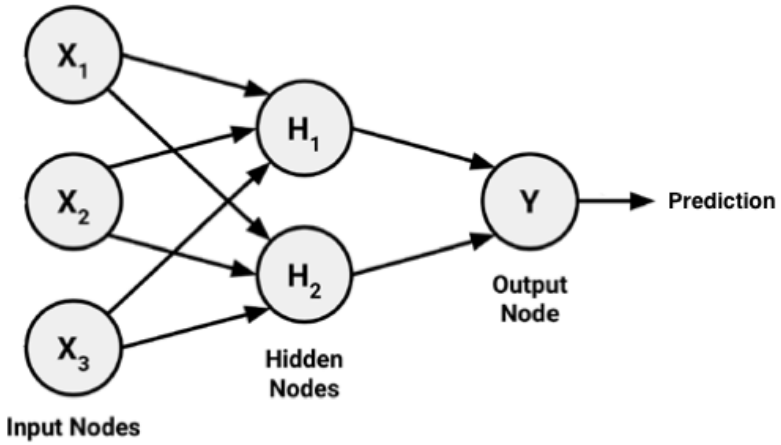
SINGLE-LAYER ANN

The input nodes process the incoming data exactly as it is received, so the network has only one set of connection weights.

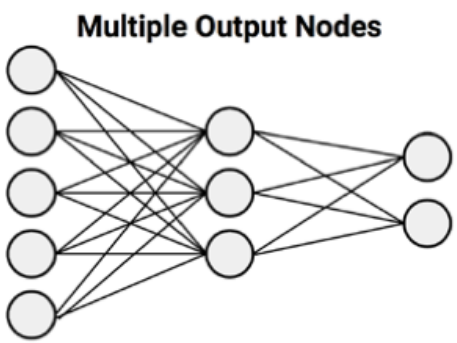


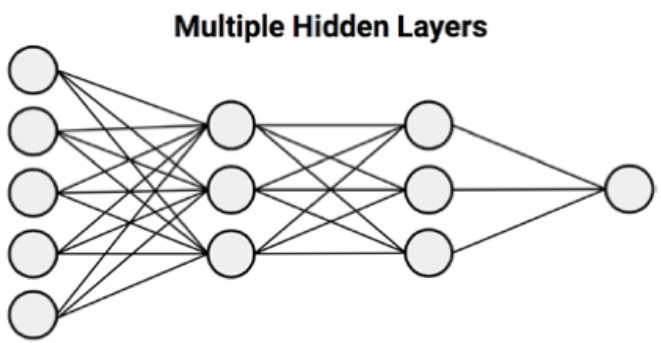
Multiple-layer ANN

- Often additional hidden layers are included to process the signals from the input nodes prior to reaching the output node.



- An ANNs with multiple hidden layers is called a **Deep Neural Network** (DNN) and the training of such network is referred to as deep learning.





Feedforward ANNs

- Networks in which the input signal is fed continuously in one direction from connection to connection until it reaches the output layer.
- Feedforward ANNs have been extensively applied to real-world problems.
- The **feedforward neural network** (FFNN) or **multilayer perceptron** (MLP) is the de facto standard ANN topology.

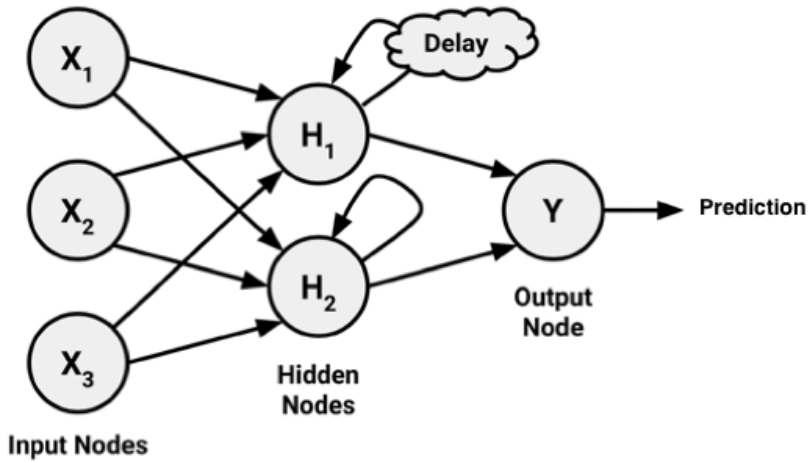
$$y \leftarrow f(x) = f_0\left(\alpha_0 + \sum_{h=1}^{H} w_h f_h\left(\alpha_h + \sum_{j=1}^{p} w_{jh} x_j\right)\right)$$

where

  - H is the number of hidden units
  - f_0 is the activation function for the output layer
  - f_h are the activation functions for the hidden layers
  - w_h and w_{jh} are the layers' weights
  - α_0 and α_h are the layers' bias coefficients

Recurrent (or feedback) ANNs

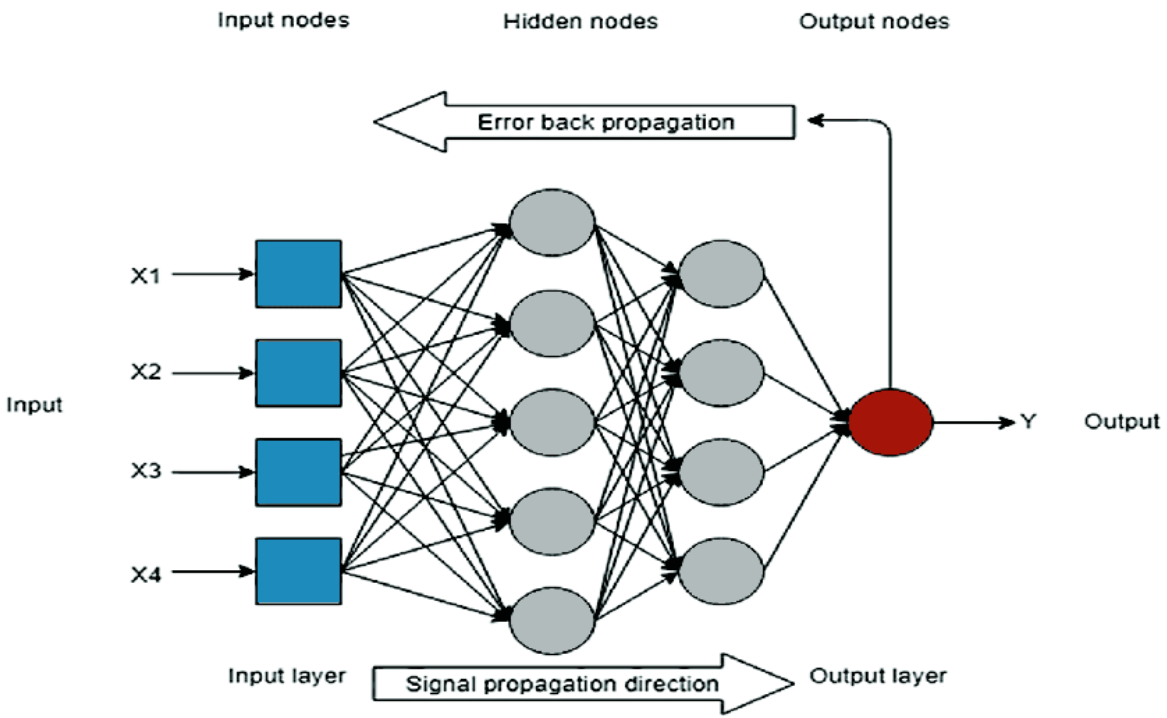
- Networks in which the signals travel in both directions using loops.
- This property allows extremely complex patterns to be learned.
- In spite of their potential, recurrent networks are still largely theoretical and are rarely used in practice.



Number of nodes in each layer

- The complexity of an ANN depends on the number of nodes in each layer.
- The number of input nodes is predetermined by the number of features (or predictors) in the input data.
- The number of output nodes is predetermined by the number of outcomes to be modelled (single response or multiple response variables) or the number of classes (if the response is qualitative).
- The number of hidden nodes is selected by the user before training the model.
- In general, more complex network topologies with a greater number of network connections allow the learning of more complex problems. But there exists the risk of overfitting (i.e. it may generalize poorly to future data).
- Large neural networks can be computationally expensive and slow to train.
- Best practice: use the fewest nodes that result in adequate performance in a validation dataset.

- Training of ANNs and DNNs by adjusting connection weights is very computationally intensive.
- **Back-propagation** is the most common algorithm used for training.
- Back-propagation employs gradient descent to minimize the loss function with respect to the weights by recursively applying the chain rule of calculus from the output layer back to the input layer.



Back-propagation algorithm

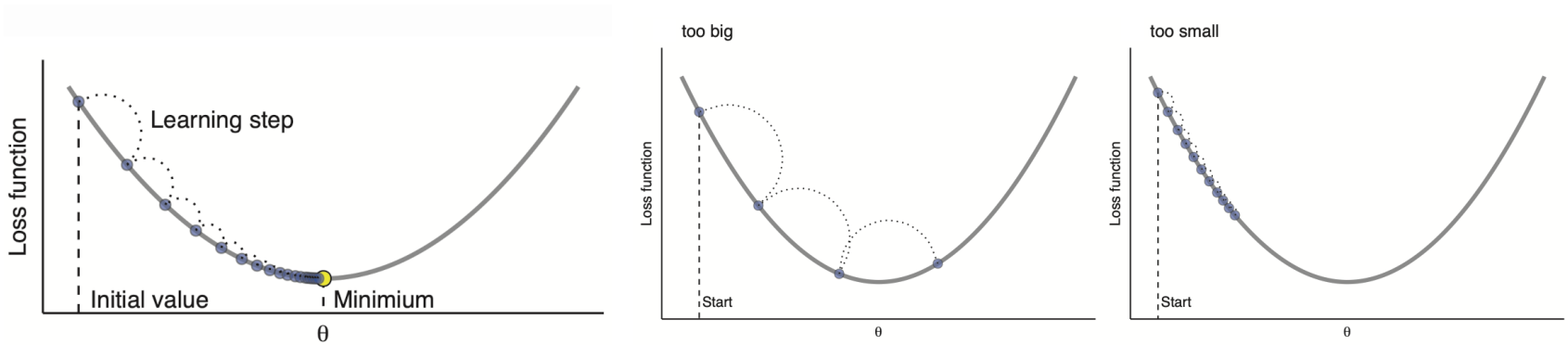
1. Weights are initialized at random.
2. The algorithm iterates through many cycles (epochs) consisting of a forward and a backward phase:
    - In the forward phase the neurons are activated in sequence from the input layer to the output layer, applying each neuron's weights and activation function along the way. Upon reaching the final layer, an output signal is produced.
    - In the backward phase the network's output signal resulting from the forward phase is compared to the true target value in the training data. The difference between the network's output signal and the true value results in an error that is propagated backwards in the network to modify the connection weights between neurons and reduce future errors.
3. The process is iterated until a stopping criterion is satisfied.

Gradient Descent (GD)

- GD is a generic iterative algorithm that aims at minimizing a loss function.
- GD measures the local gradient of the loss function for a given set of parameters and takes steps in the direction of the descending gradient.
- Given a loss function $\mathcal{L}(\boldsymbol{w})$, start with an arbitrarily chosen solution $\boldsymbol{w}_0$ and move it a new solution in the direction of the negative gradient $\nabla \mathcal{L}(\boldsymbol{w})$.
  At iteration t update the weights using

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \times \frac{1}{n} \nabla \mathcal{L}(\boldsymbol{w})$$

  where η is the **learning rate** controlling how large the updating step should be:
  - if too small, then the algorithm might converge very slowly;
  - if too large, then the algorithm can have issues with convergence or make the algorithm divergent.



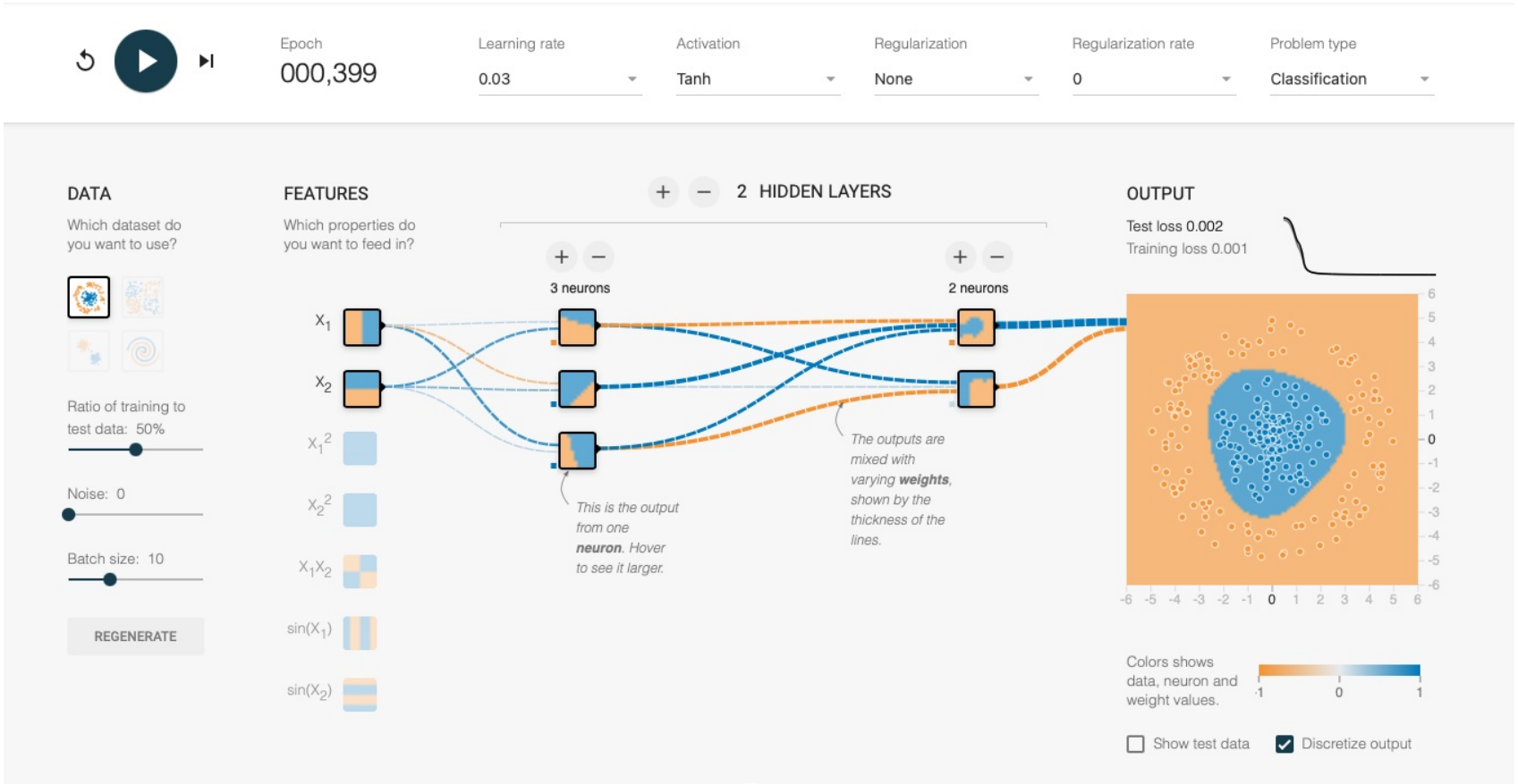
Stochastic Gradient Descent (SGD)

- For non convex loss functions there may be several local minima, plateaus, etc., that makes finding the global minimum difficult.
- SGD algorithms are a modification of GD where the gradient is calculated using just a random small part of the observations instead of all of them.
- **Batch SGD** computes the gradient of the loss function for a subset of observations (minibatches) and update the solution using

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \times \frac{1}{|\mathcal{B}|} \nabla \sum_{i \in \mathcal{B}} \mathcal{L}_i(\boldsymbol{w})$$

  - Online SGD is a special case of batch SGD in which each minibatch has only one observation.
  - Ordinary GD is a special case of SGD in which there is only one batch containing all observations.
- The introduction of randomness can help to jump out of local minima and plateaus to get sufficiently near the global minimum.
- Often, SGD leads to a reduction in computation time.

http://playground.tensorflow.org



Penalization

- Complex ANNs tend to overfit, thus to avoid overfitting problems more elaborate versions of this objective function can be obtained by including a penalty term. For example

$$\mathcal{L}(\boldsymbol{w}; \lambda) = \mathcal{L}(\boldsymbol{w}) + \lambda \times \text{Penalty}(\boldsymbol{w})$$

  where
  - $\lambda \geq 0$ is a tuning hyperparameter (decay) which controls the level of penalization;
  - $\text{Penalty}(w)$ is a penalty function which discourage large weights (regularization effect).

- Note that both the $\mathcal{L}(\boldsymbol{w})$ and the $\text{Penalty}(\boldsymbol{w})$ functions make sense if the variables are measured on the same scale. Thus, as a preliminary step all the features need to be rescaled (e.g. by standardization).

Pros and cons

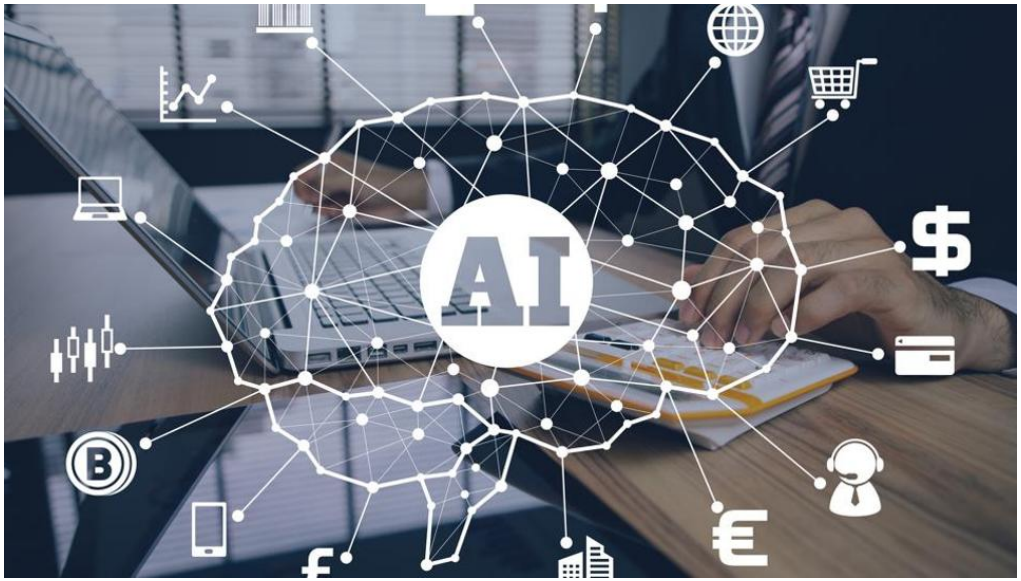
Advantages:

- Flexibility: ANNs allow for good approximation of practically any regression function. A neural network with at least one hidden layer of sufficient neurons is a universal function approximator.
- Compactness of representation: the estimated regression function is identified by a limited number of components.
- Can be adapted to classification or numeric prediction problems.
- Makes few assumptions about the data's underlying distribution and relationships.

Drawbacks:

- Computationally intensive and slow to train, particularly if the network topology is complex.
- Arbitrariness: there are no strong criteria with which to choose the number of hidden nodes.
- Instability of estimates: the nature of objective function implies that its properties are difficult to identify, especially the existence of a single minimum point. Instead, there is empirical evidence of the frequent presence of local minima, and different results may be obtained if the optimization algorithm is started from different points.
- Overfitting: very prone to overfitting training data.
- Inference: there are no standard errors associated with the coefficients or other inferential procedures.
- Interpretation: results in a complex black box model that is difficult, if not impossible, to interpret

Conclusions



- Machine Learning and Statistical Learning play a vital role in the broader field of Artificial Intelligence.
- They enable machines to learn from data, make predictions, and solve complex tasks without being explicitly programmed for a specific task.
- ML/SL techniques have widespread applications across various domains, revolutionizing industries and improving decision-making processes.

References (for starters...)

- James G., Witten D., Hastie T. and Tibshirani R. (2021) An Introduction to Statistical Learning: with Applications in R, 2nd edition, Springer-Verlag https://www.statlearning.com
- Murphy K. P. (2022) Probabilistic Machine Learning: An Introduction, MIT Press https://probml.github.io/pml-book/book1.html
- Murphy K. P. (2023) Probabilistic Machine Learning: Advanced Topics, MIT Press https://probml.github.io/pml-book/book2.html
- Goodfellow I., Bengio Y. and Courville A. (2016) Deep Learning, MIT Press http://www.deeplearningbook.org
- Nielsen M. A. (2015) Neural Networks and Deep Learning, Determination Press http://neuralnetworksanddeeplearning.com


THANK YOU
ANY QUESTION?