Data Science & Artificial Intelligence

Summer Term 2022

IFM Institut für Fahrerassistenz und vernetzte Mobilität

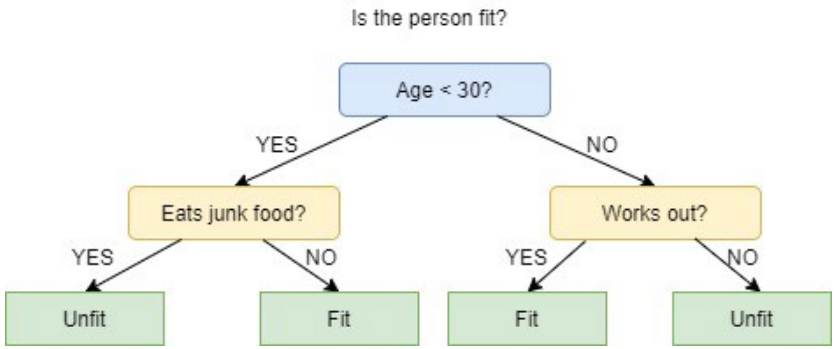
L05 Trees

J. Haselberger, D. Schneider, B. Stuhr,

L05.1 Introduction

Decision Trees (DTs)

- non-parametric supervised learning method for classification and regression
- predicts the value of a target variable by learning simple decision rules



- Classification trees are essentially a series of questions designed to assign a class
- good interpretability

L05.2 Definitions

Root Node

- topmost decision node

Leaf Nodes

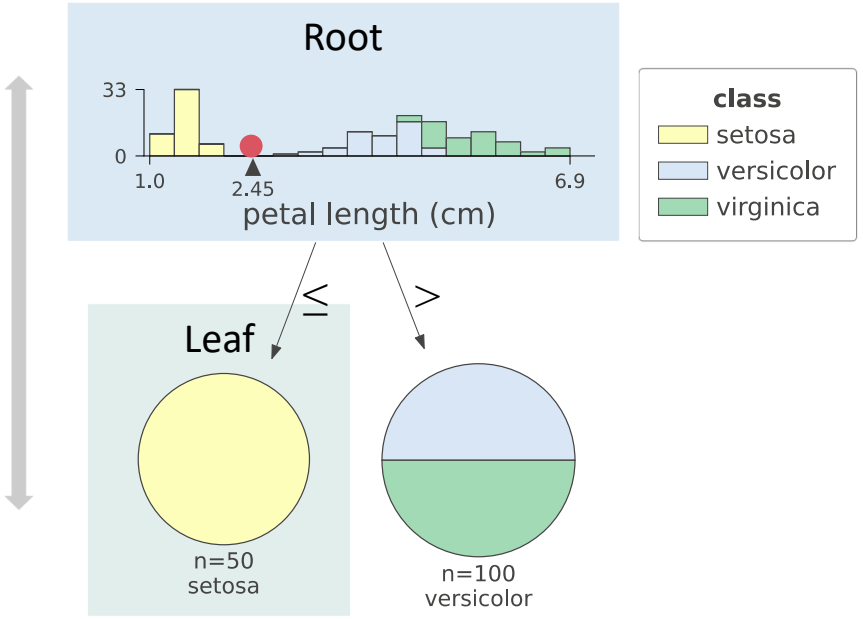
- also called terminal nodes
- nodes that don’t split into more nodes

split point

- decicion where to split between the nodes

Tree depth

- how many splits a tree can make



L05.2 Definitions

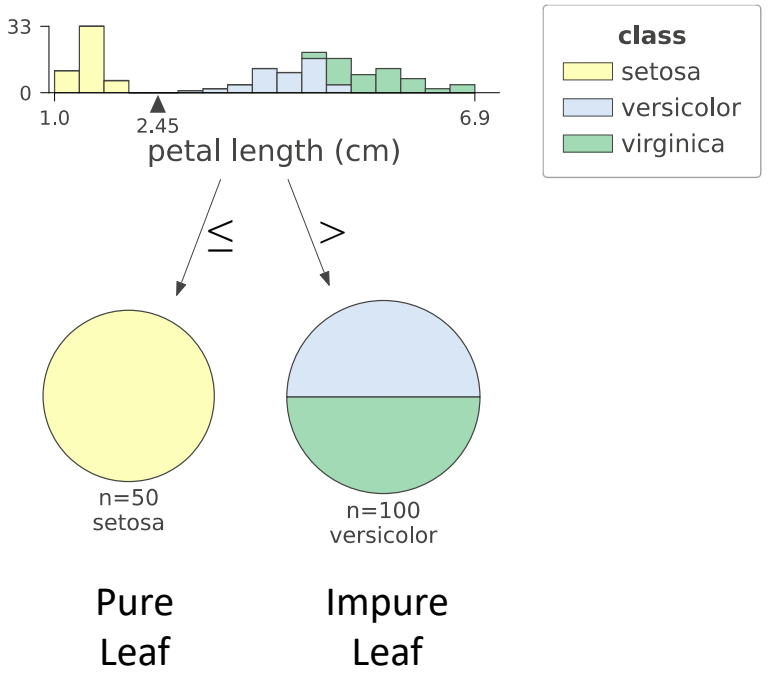
Output of a Decision Tree

- classes are assigned by majority vote

Greedy Algorithm

- continue to split until it has a pure node
- classification trees don’t split on pure nodes

? How do we find a good split point?



L05.3 Information Gain

Find a good split point

- A good value for a split point is one that separates one class well from the others
- this results in a large Information Gain

Information Gain (IG):

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N} I(D_j)$$

f: featrue split on
I: Impurity criterion
D_p: Dataset of the parent node
N: total number of samples
D_j: Dataset of the jth child node
N_j: number of samples in jth child node

L05.3 Information Gain

- Binary Splits
- For binary splits (left and right) we can simply the formula to:

Information Gain Binary Split (IG):

$$IG(D_p, f) = I(D_p) - \underbrace{\left(\frac{N_{left}}{N} I(D_{left}) + \frac{N_{right}}{N} I(D_{right}) \right)}_{\text{weighted impurity}}$$

f: featrue split on
I: Impurity criterion
D_p: Dataset of the parent node
N: total number of samples
D_j: Dataset of the jth child node
N_j: number of samples in jth child node

L05.4 Impurity

Impurity Function

- measures how impure a leaf is
- two alternatives: Gini Index and Entropy

Gini Index:

$$I_G = \sum_{k=1}^{K} p_k(1 - p_k)$$

- p_k is the probability to pick a certain label out of the leaf: $p_k = \frac{N_k}{N}$
- $1 - p_k$ is the probability not to pick that class
- K is the number of classes

L05.4 Impurity

Entropy

- Based on the following assumption:
  - worst case scenario in a leaf is to have $p_1 = p_2 = \cdots = p_k = \frac{1}{K}$
  - This means that all classes occur the same number of times

- The goal is to find a distribution p that is as far as possible from this case (we set q for the worst case)
- Use Kullback–Leibler Divergence to measure how different two distributions are

Kullback-Leibler Divergence:

$$KL(p||q) = \sum_{k=1}^{K} p_k \log\left(\frac{p_k}{q_k}\right)$$

L05.4 Impurity

From Kullback–Leibler Divergence to Entropy

$$KL(p||q) = \sum_{k=1}^{K} p_k \log\left(\frac{p_k}{q_k}\right) \qquad \text{with } q_k = \frac{1}{K}$$

$$= \sum_{k=1}^{K} p_k \log\left(\frac{p_k}{\frac{1}{K}}\right) \quad = \sum_{k=1}^{K} p_k \log\left(p_k\right) + \sum_{k=1}^{K} p_k \log\left(K\right) \quad = \sum_{k=1}^{K} p_k \log\left(p_k\right) + \log\left(K\right) \underbrace{\sum_{k=1}^{K} p_k}_{=1}$$

$$= \sum_{k=1}^{K} p_k \log\left(p_k\right) + \log\left(K\right)$$

- The goal is to maximize the distance, therefore the only relevant term is $\sum_{k=1}^{K} p_k \log\left(p_k\right)$

L05.4 Impurity

From Kullback–Leibler Divergence to Entropy

- maximize the distance (= to maximize the order):

$$\max_p \left(\sum_{k=1}^{K} p_k \log\left(p_k\right) \right)$$

- is the same as minimize the Disorder (= Entropy)

$$\min_p \underbrace{\left(-\sum_{k=1}^{K} p_k \log\left(p_k\right) \right)}_{\text{Entropy}}$$

Entropy:

$$I_H = -\sum_{k=1}^{K} p_k log_2(p_k)$$

L05.6 Working Principle

Find the best split point

- For every possible split point:
  1. calculate the Impurity I (Gini or Entropy)
  2. calculate the Information Gain IG
- Pick the Split point with the highest resulting Information Gain

Information Gain Binary Split (IG):

$$IG(D_p, f) = I(D_p) - \underbrace{\left(\frac{N_{left}}{N} I(D_{left}) + \frac{N_{right}}{N} I(D_{right})\right)}_{\text{weighted impurity}}$$

Gini Index:

$$I_G = \sum_{k=1}^{K} p_k(1 - p_k)$$

Entropy:

$$I_H = -\sum_{k=1}^{K} p_k log_2(p_k)$$

L05.7 Regression

Transfer to regression problems:

- Instead of Gini or Entropy, the Mean Squared Loss is used as Impurity Function

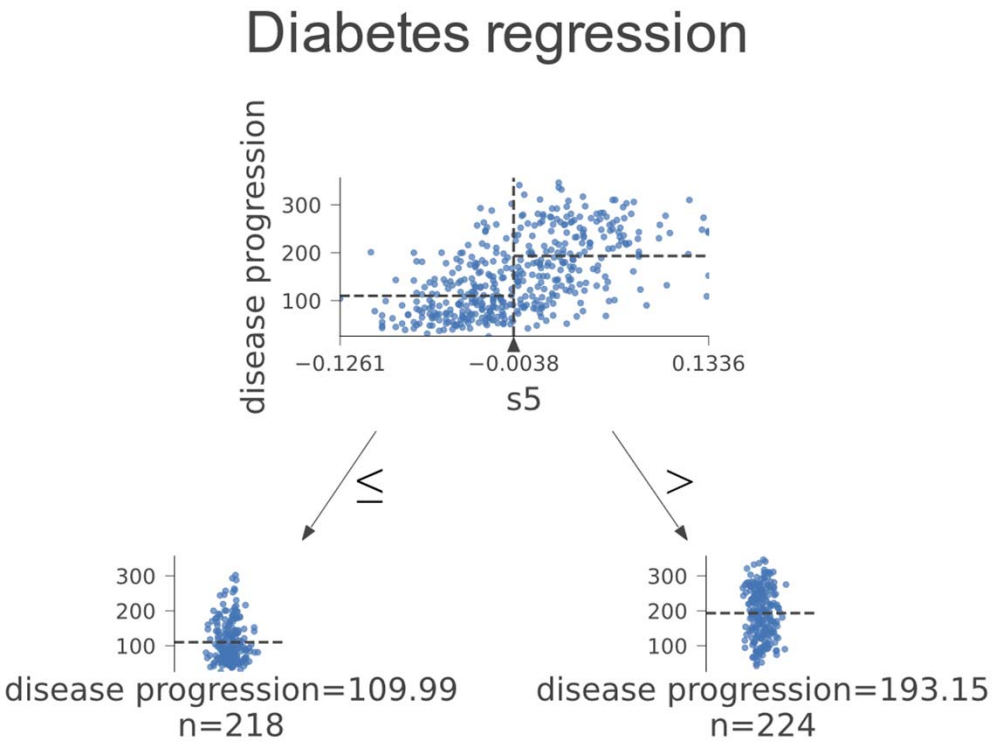
Mean Squared Loss:

$$I_M = \frac{1}{N_j} \sum_{y \in D_j} (y - \bar{y})^2$$

- The output of the *j*th Leaf is the mean value of its data:

Mean Leaf value:

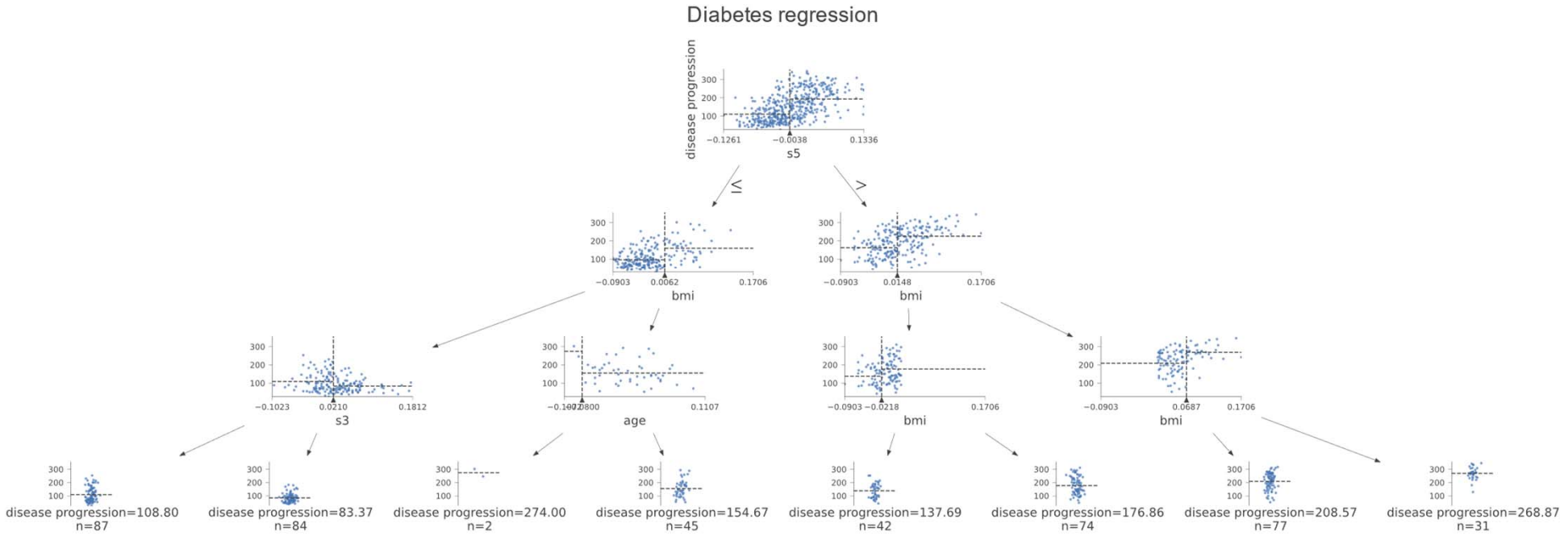
$$\bar{y}_j = \frac{1}{N_j} \sum_{y \in D_j} y$$



L05.7 Regression

Transfer to regression problems:

- Problem: Regression Trees can grow infinitely
- Tradeoff between regression error and max tree depth



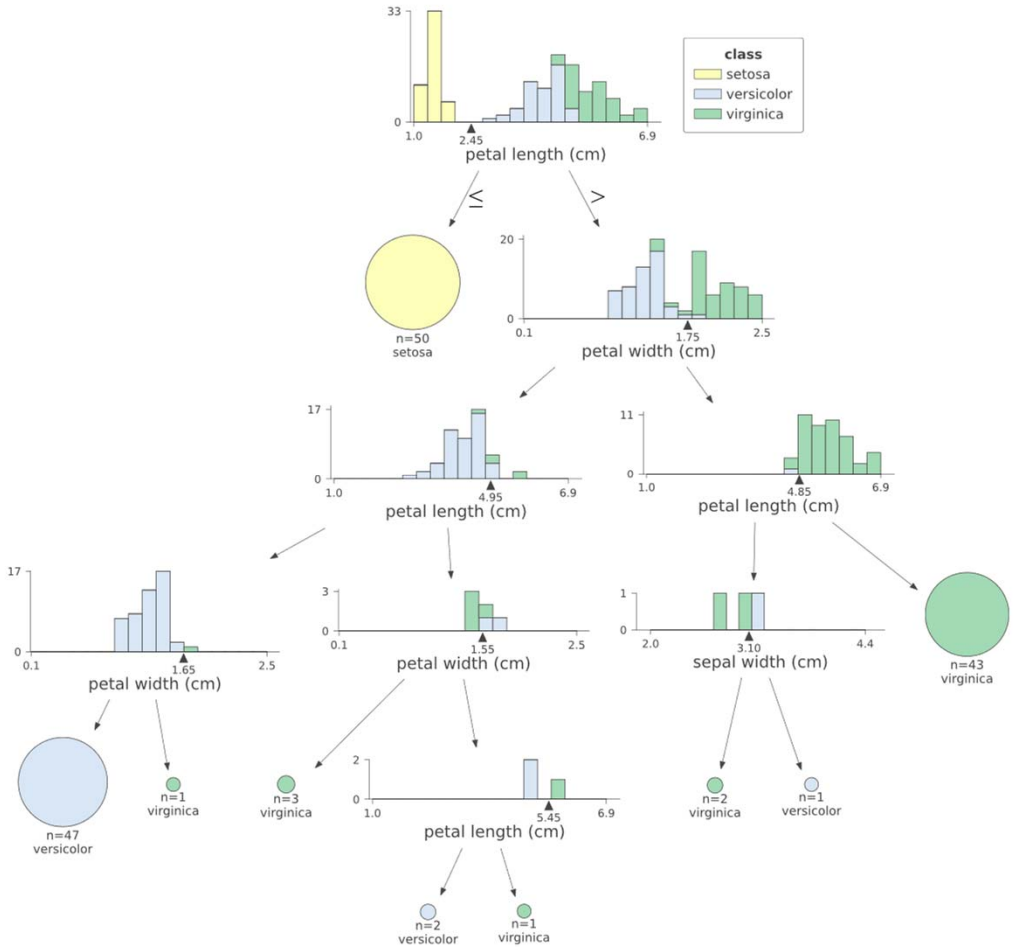
L05.8 Advantages and Disadvantages

Advantages:

- Simple to understand and to interpret
- Requires no data preparation
- handle both numerical and categorical data
- white box model

Disadvantages:

- can lead to a very deep trees (over-complex trees)
- often leads to overfitting on the training dataset
- can be unstable because small variations in the data might result in a completely different tree



L05.9 Prevent Overfitting

Simple Approaches:

Restrict tree size

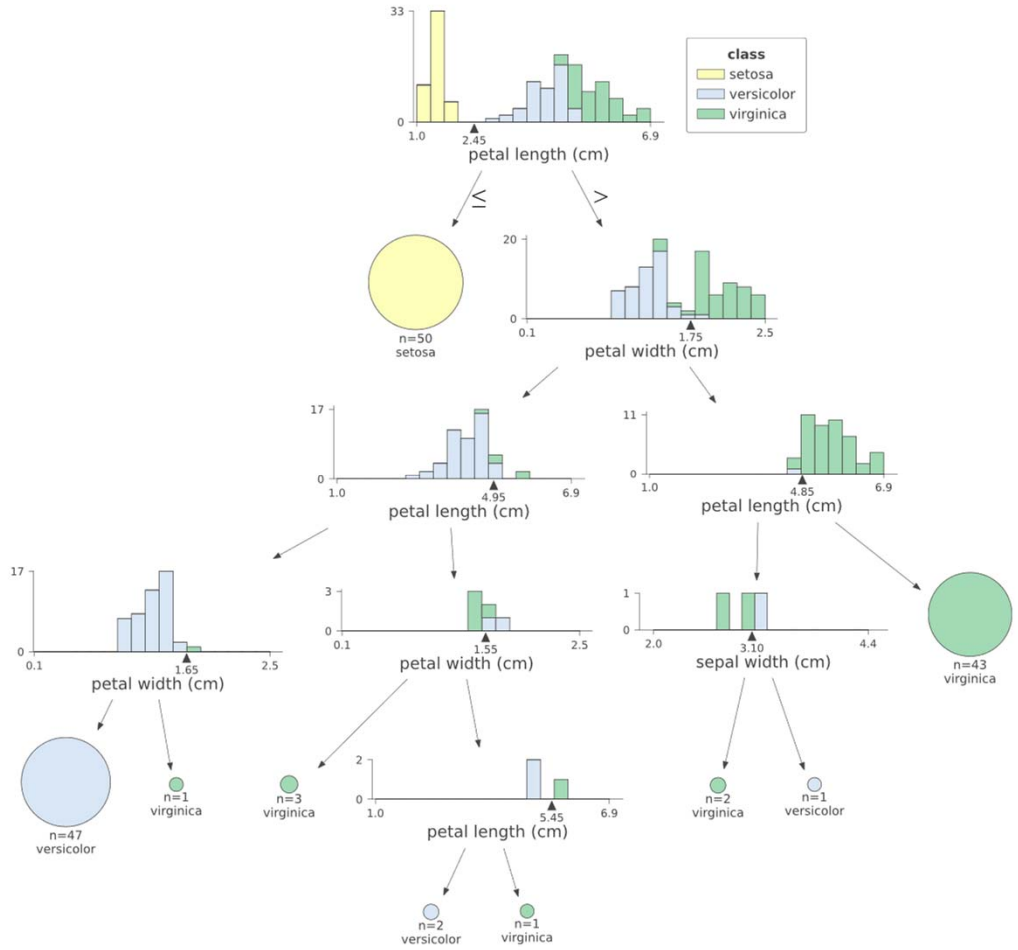
- limit how many splits a tree can make

Required number of data points per node

- Only split if a minimum number of observations per node is available

Impurity Gain threshold

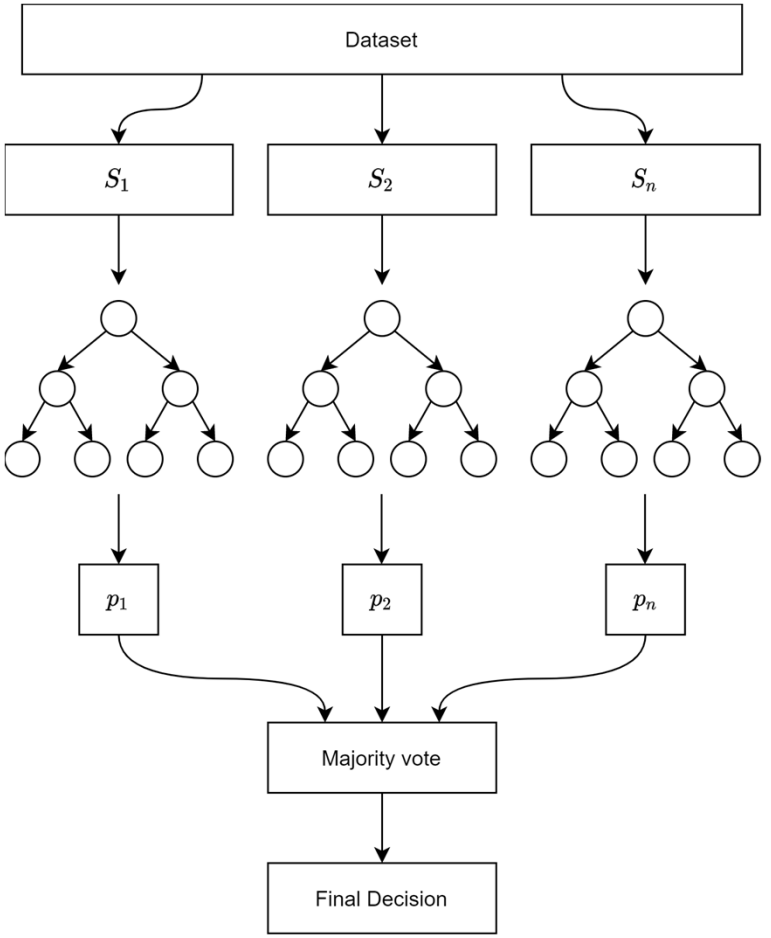
- Split only if a threshold value for the Impurity Gain has been reached



L05.10 Bagging

Working Principle

- the training dataset S is split into N smaller subsets
- a separate tree is trained for each of these subsets ➔ Ensemble
- The final output is determined by majority vote (for classification) or mean value (for regression) of the individual tree outputs



For classification:

$$P = \arg\max\{p_1, p_2 \ldots, p_n\}$$

For regression:

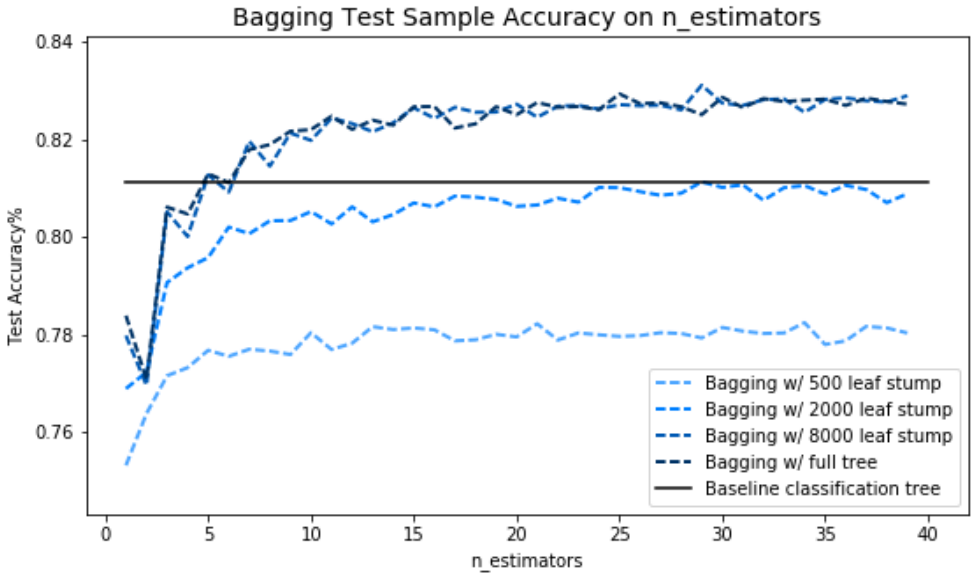
$$P = \frac{1}{N}\sum_{n=1}^{N} p_n$$

Face Overfitting (high variance)
Bagging is an effective way to minimize the variance problem

L05.10 Bagging

Hyperparameter

- N determines how often the data is split and how many individual trees are learned
- the tree depth controls how many splits a tree within the ensemble can make
- The performance of the bagging ensemble exceeds the CART benchmark only when the number of estimators exceeds a certain limit



Bagging tree depth
Limiting the tree depth is quite superfluous in a bagging ensemble

L05.10 Bagging

Disadvantages

- Loss of interpretability
  - the final bagged classifier is not a tree
  - we lose the clear interpretability of a classification / regression tree
- Computational complexity
  - multiplying the work of growing a single tree by N
  - especially if we use the more elaborate implementation that prunes and validates the data with the original training data

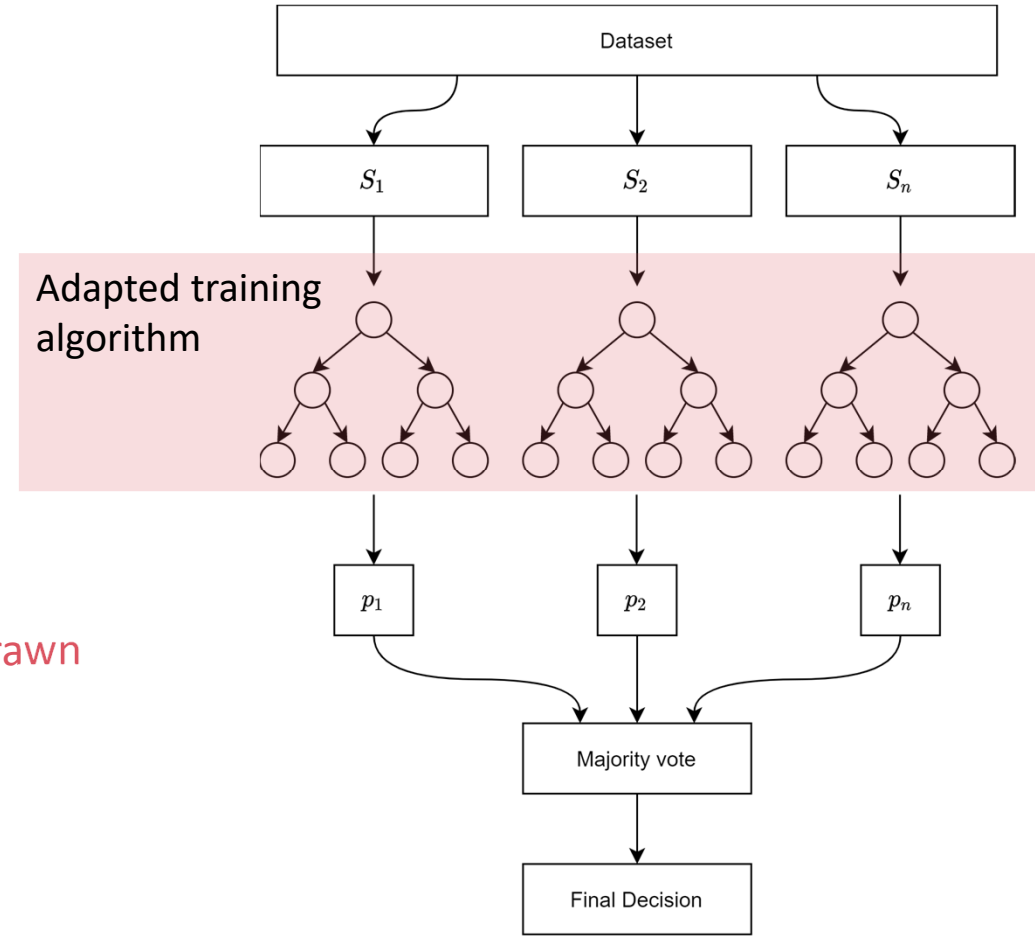
L05.11 Random Forests

Working Principle

- based on Bagging
- the training dataset S is split into N smaller subsets
- a separate tree is trained for each of these subsets

However, the training algorithm for the individual trees is different:

- Bagging: split are allowed in every dimension d
- Random Forests: splits are only allowed in randomly drawn subsets τ of the dimensions: $\tau \ll d$
- Rule of thumb: $\tau \approx \sqrt{d}$



L05.11 Random Forests

Intuition

Given that there is a very strong predictor in the dataset, along with a number of other moderately strong predictors:

Using Bagging:

- in the collection of the bagged trees, most or all of the trees will use this strong predictor in the top split
- All of the bagged trees are quite similar to each other
- predictions from the bagged trees will be highly correlated

Random Forests:

- weaker predictors will have also the chance to be a split candidate as τ is randomly selected

L05.12 Boosting

Intuition

- Bagging and Random Forest ensembles consists of large trees (strong learners)
- What if we use only very small trees (weak learners)?

Strong Learners	**VS**	**Weak Learners**
capable to reduce the error to zero		a bit better than guessing

L05.12 Boosting

Adaptive Boosting: AdaBoost

- tries to reduce the Bias problem
- uses only Decision Stumps

Random Forests

- each tree has the same influence on the model output

VS

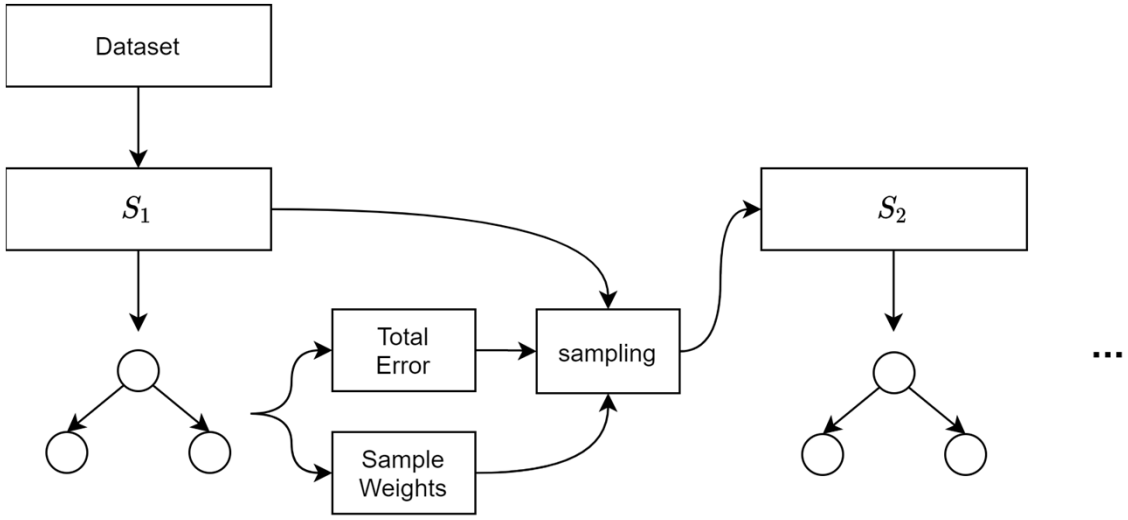
AdaBoost

- some stumps have higher influence than others
- error of the first stump makes influence how second stump is trained

L05.12 AdaBoost

Working Principle

1. each sample is assigned a weight that describes how important the correct classification of the sample is
2. based on S_n a Decision Stump h_n is trained
3. calculate the Total Error ϵ
4. calculate new sample weights
5. sample a new dataset S_{n+1}
6. reset the sample weights for S_{n+1}



$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

L05.12 AdaBoost

Working Principle

- For N samples, the sample weights are initialized to $\frac{1}{N}$

Total Error ϵ of a stump:

- Sum of weights associated with misclassified samples

$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

Amount of say α

- defines how influential the stump h_t is for the resulting model output

Amount of say:

$$\alpha_t = \frac{1}{2} \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

L05.12 AdaBoost

Working Principle

Sample weights update rule

- different updates for correctly and misclassified samples:

misclassified samples:

$$w_i \leftarrow w_i \, e^{\alpha}$$

correctly classified samples:

$$w_i \leftarrow w_i \, e^{-\alpha}$$

- after calculation of the new sample weights, the weights need to be normalized so that $\sum_{i=1}^{N} w_i = 1$

L05.12 AdaBoost

Use Case: Heart Disease

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

initial dataset

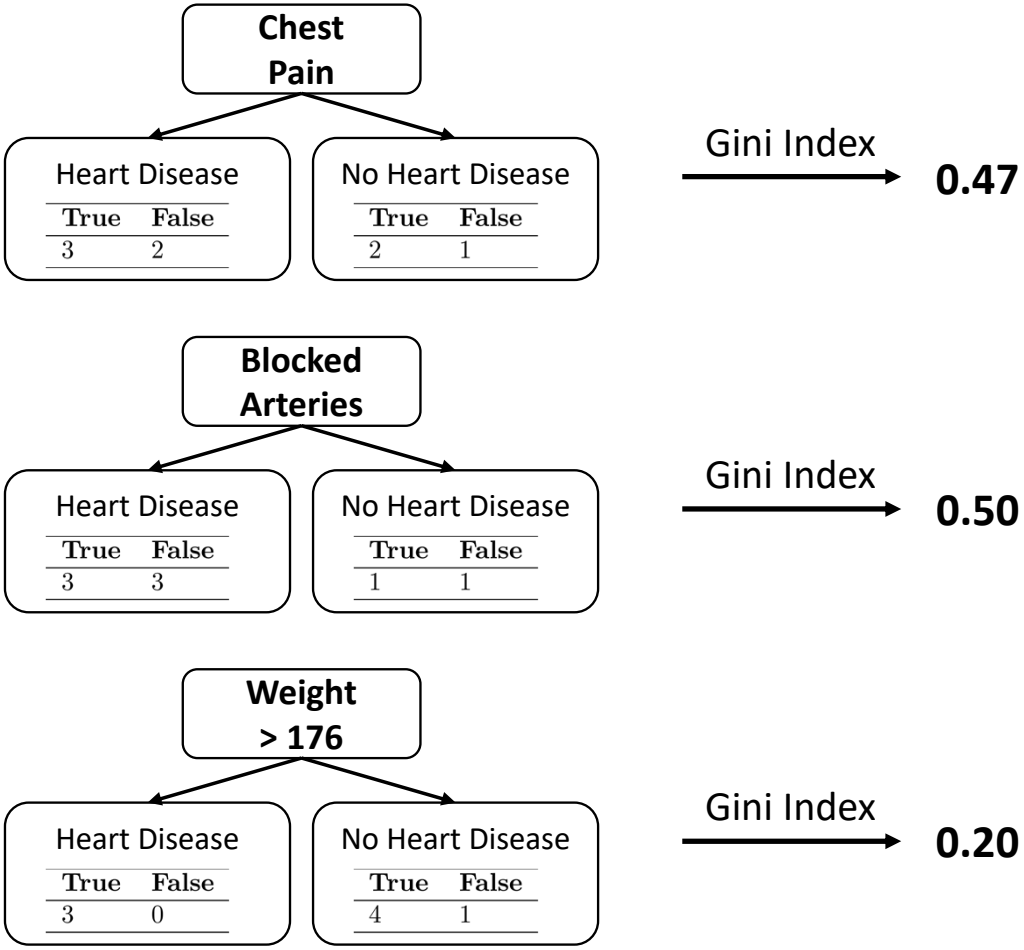
For the first iteration init all weights to $\frac{1}{N}$

L05.12 AdaBoost

Use Case: Heart Disease

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

initial dataset

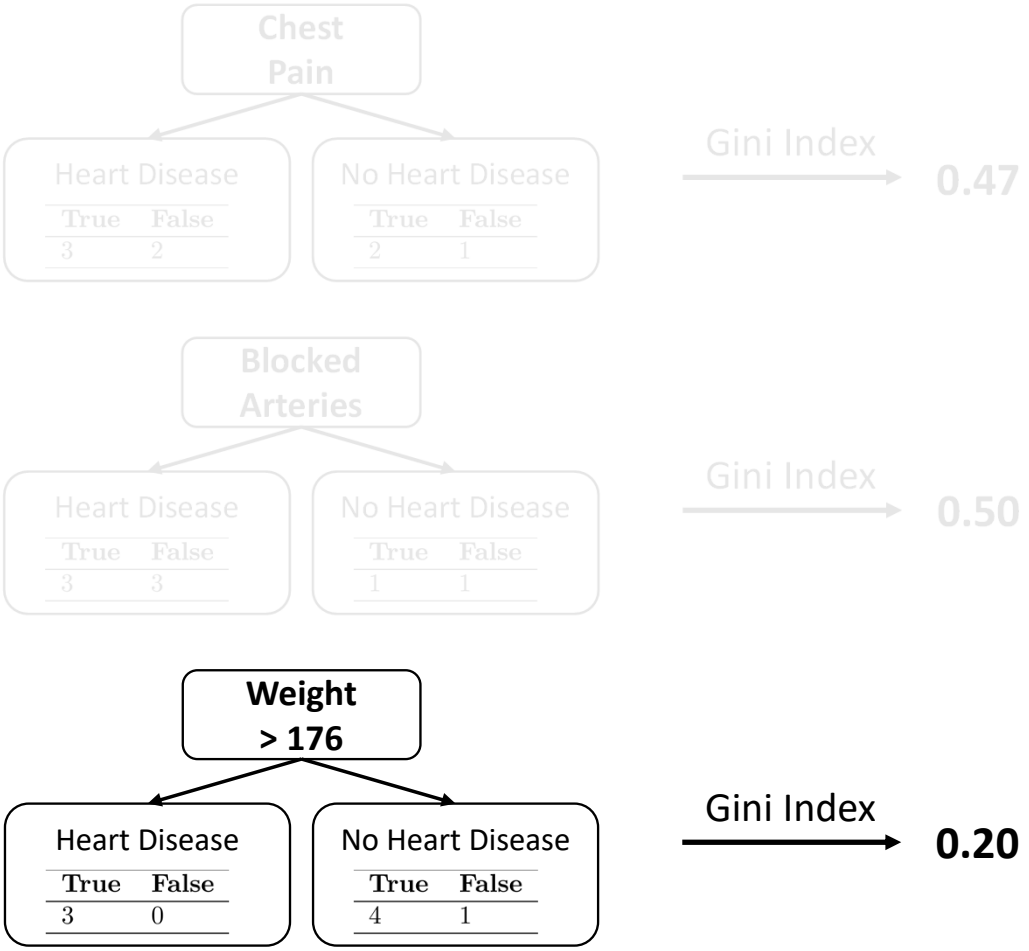


L05.12 AdaBoost

Use Case: Heart Disease

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

initial dataset



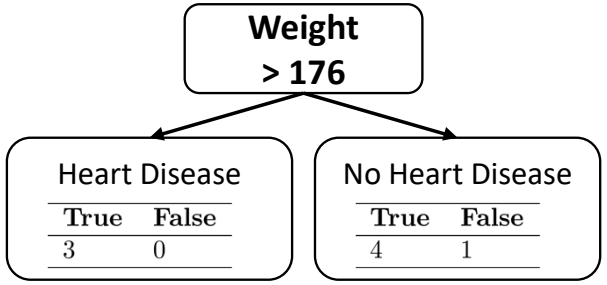
L05.12 AdaBoost

Use Case: Heart Disease

Update the misclassified sample weights

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

initial dataset



$$\epsilon = \frac{1}{8}$$

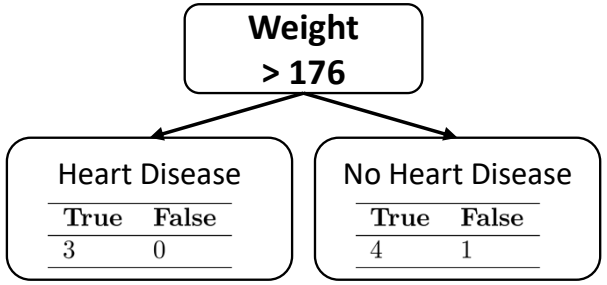
$$\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right) = \frac{1}{2}\log\left(\frac{1-\frac{1}{8}}{\frac{1}{8}}\right) = 0.97$$

$$w_i \leftarrow w_i\, e^{\alpha} = \frac{1}{8}e^{0.97} = 0.33$$

L05.12 AdaBoost

Use Case: Heart Disease

Update the misclassified sample weights



Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	
No	Yes	180	Yes	1/8	
Yes	No	210	Yes	1/8	
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	
No	Yes	125	No	1/8	
Yes	No	168	No	1/8	
Yes	Yes	172	No	1/8	

initial dataset

$$\epsilon = \frac{1}{8}$$

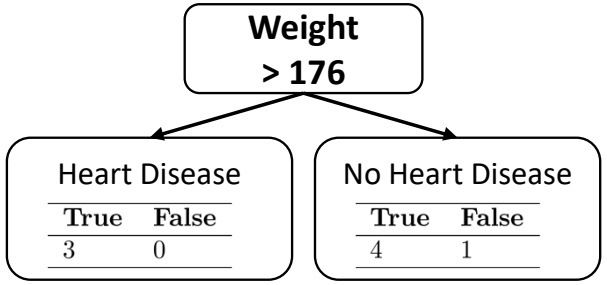
$$\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right) = \frac{1}{2}\log\left(\frac{1-\frac{1}{8}}{\frac{1}{8}}\right) = 0.97$$

$$w_i \leftarrow w_i\, e^{\alpha} = \frac{1}{8}e^{0.97} = 0.33$$

L05.12 AdaBoost

Use Case: Heart Disease

Update the correct sample weights



Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	
No	Yes	180	Yes	1/8	
Yes	No	210	Yes	1/8	
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	
No	Yes	125	No	1/8	
Yes	No	168	No	1/8	
Yes	Yes	172	No	1/8	

initial dataset

$$\epsilon = \frac{1}{8}$$

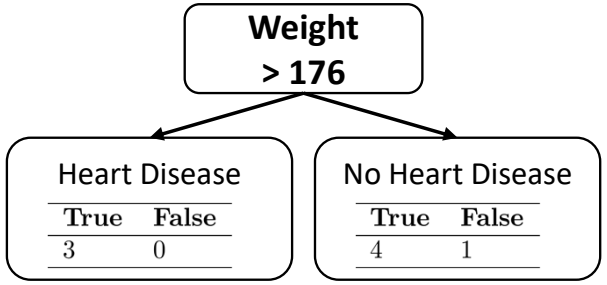
$$\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right) = \frac{1}{2}\log\left(\frac{1-\frac{1}{8}}{\frac{1}{8}}\right) = 0.97$$

$$w_i \leftarrow w_i\, e^{-\alpha} = \frac{1}{8}e^{-0.97} = 0.05$$

L05.12 AdaBoost

Use Case: Heart Disease

Update the correct sample weights



Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	0.05
No	Yes	180	Yes	1/8	0.05
Yes	No	210	Yes	1/8	0.05
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	0.05
No	Yes	125	No	1/8	0.05
Yes	No	168	No	1/8	0.05
Yes	Yes	172	No	1/8	0.05

initial dataset

$$\epsilon = \frac{1}{8}$$

$$\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right) = \frac{1}{2}\log\left(\frac{1-\frac{1}{8}}{\frac{1}{8}}\right) = 0.97$$

$$w_i \leftarrow w_i\, e^{-\alpha} = \frac{1}{8}e^{-0.97} = 0.05$$

L05.12 AdaBoost

Use Case: Heart Disease

- Create a new data set with the same size, init the sample weights to $\frac{1}{N}$
- sample from the previous dataset based on the normalized sample weights
- Samples with a high weight are very likely to appear in the new dataset
- Duplicates are allowed

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight	Norm. Weight
Yes	Yes	205	Yes	1/8	0.05	0.07
No	Yes	180	Yes	1/8	0.05	0.07
Yes	No	210	Yes	1/8	0.05	0.07
Yes	Yes	167	Yes	1/8	0.33	0.49
No	Yes	156	No	1/8	0.05	0.07
No	Yes	125	No	1/8	0.05	0.07
Yes	No	168	No	1/8	0.05	0.07
Yes	Yes	172	No	1/8	0.05	0.07

initial dataset

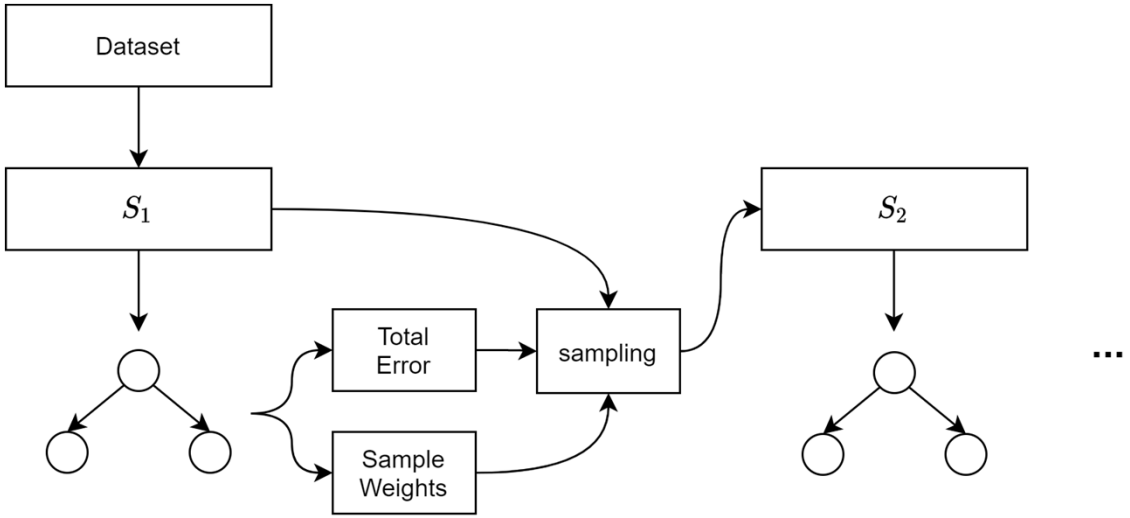
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
No	Yes	156	No	1/8
Yes	Yes	167	Yes	1/8
No	Yes	125	No	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	167	Yes	1/8
Yes	Yes	172	No	1/8
Yes	Yes	205	Yes	1/8
Yes	Yes	167	Yes	1/8

new dataset

L05.12 AdaBoost

Use Case: Heart Disease

- Now a new stump is learned based on the new data set



L05.12 AdaBoost

Intuition

- AdaBoost is not prone to overfitting though there is no concrete proof for this
- being extended beyond binary classification and has found use cases in text and image classification as well

- Boosting technique learns progressively, it is important to ensure that you have quality data.
- AdaBoost is also extremely sensitive to Noisy data and outliers

L05.13 Gradient Boosting

AdaBoost

- builds stumps
- the amount of say depends on how well the earlier error was compensated
- weighted data samples

VS

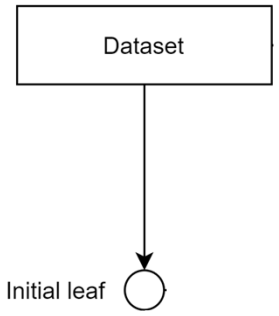
Gradient Boosting

- starts with a single leaf
- builds fixed sized trees based on the errors of the previous steps

L05.13 Gradient Boosting

Working Principle

- start by making a single initial leaf



regression:

$$F_0 = \frac{1}{N}\sum_{i=1}^{N} y_i$$

classification:

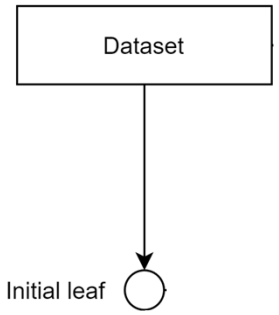
$$F_0 = \log(\text{odds}) = \log\left(\frac{p}{1-p}\right)$$

- $\log(odds)$ is the Logistic Regression equivalent of the average
  - Example: four samples are true, two false ➔ $\log(odds) = \log\left(\frac{4}{2}\right) = 0.7$

L05.13 Gradient Boosting

Working Principle

- we can convert $\log(odds)$ to probabilities:

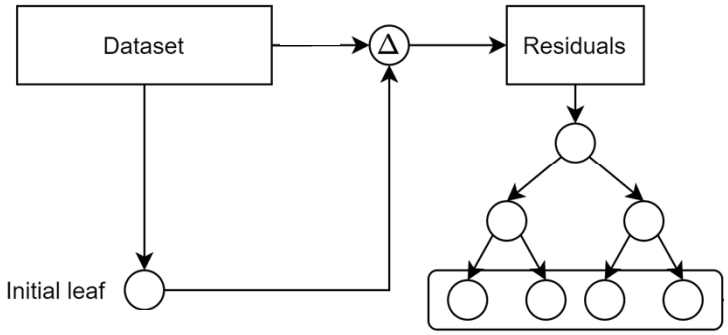


probabilities:

$$p = \frac{\exp(\log(\text{odds}))}{1 + \exp(\log(\text{odds}))}$$

L05.13 Gradient Boosting

Working Principle



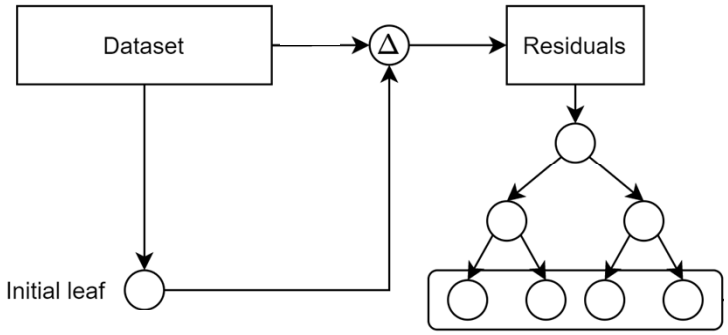
- now we build a tree based on the errors
- in comparison to AdaBoost we build trees that are significantly larger than mere stumps
- typically, the number of leaves is set to 8 up to 32
- the residuals are defined by the difference between the label and the predicted values
- $Error = Label - Prediction$

residuals:

$$r_{i,m} = \underbrace{-\left[\frac{\delta L(y_i, F(x_i))}{\delta F(x_i)}\right]}_{\text{Gradient}}; F(x) = F_{m-1}(x)$$

L05.13 Gradient Boosting

Working Principle



- for training a differentiable loss function is needed:

regression:

$$L(y_i, F(x)) = \frac{1}{2}\,(Label - Prediction)^2$$

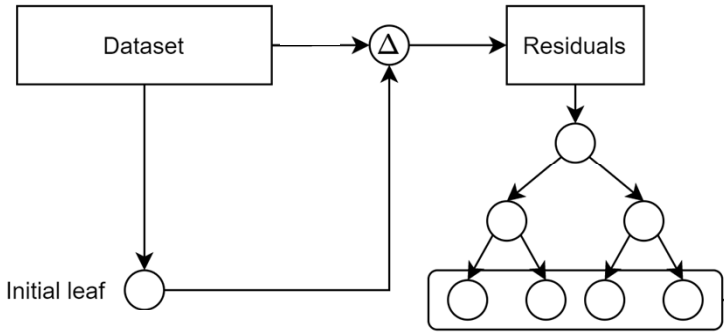
classification:

$$L(y_i, F(x)) = -\left[y_i \log(p) + (1 - y_i) \log(1 - p)\right]$$
$$= -y_i \log(\text{odds}) + \log\left(1 + \exp(\log(\text{odds}))\right)$$

- the loss function for classification is based on Log Likelihood

L05.13 Gradient Boosting

Working Principle



- get the output of a leaf:

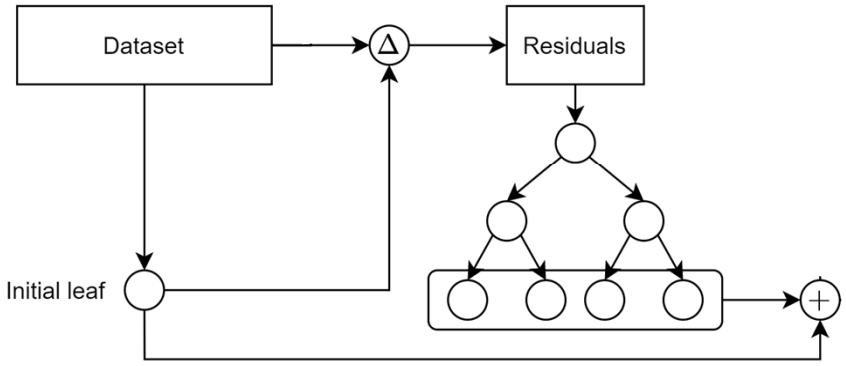
- in regression, the output is simply the average value of the elements of the leaf

classification:

$$\gamma_{j,m} = \frac{\sum \text{leaf residulas}}{\sum \text{prev. probability - (1-prev. probability)}}$$

L05.13 Gradient Boosting

Working Principle



- get the output of the model

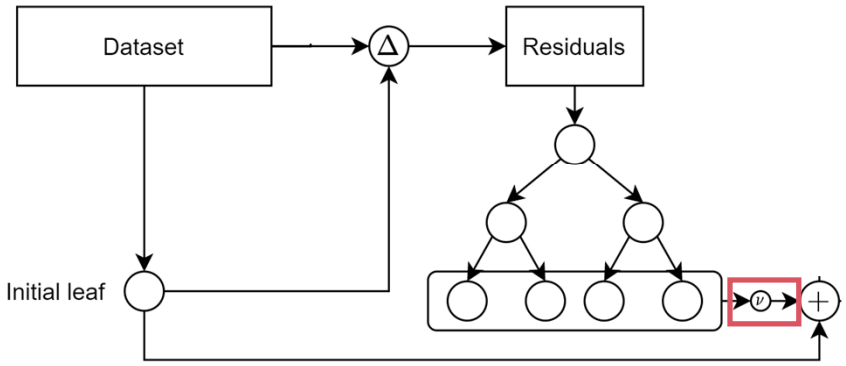
output:

$$F_m(x) = F_{m-1} + \sum_{j=1}^{J} \gamma_{j,m} I(x \in R_{j,m})$$

- Simplified: add the output values $\gamma_{j,m}$ for all leaves $R_{j,m}$ that a sample x can be found in

L05.13 Gradient Boosting

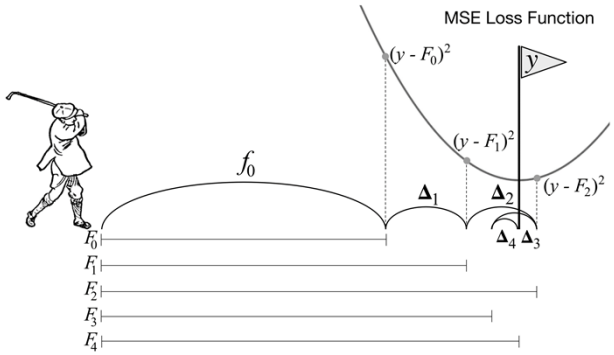
Working Principle



- **Problem**: model is prone to overfitting
- therefore, we are approaching the goal in several small steps using the learning rate ν

output:

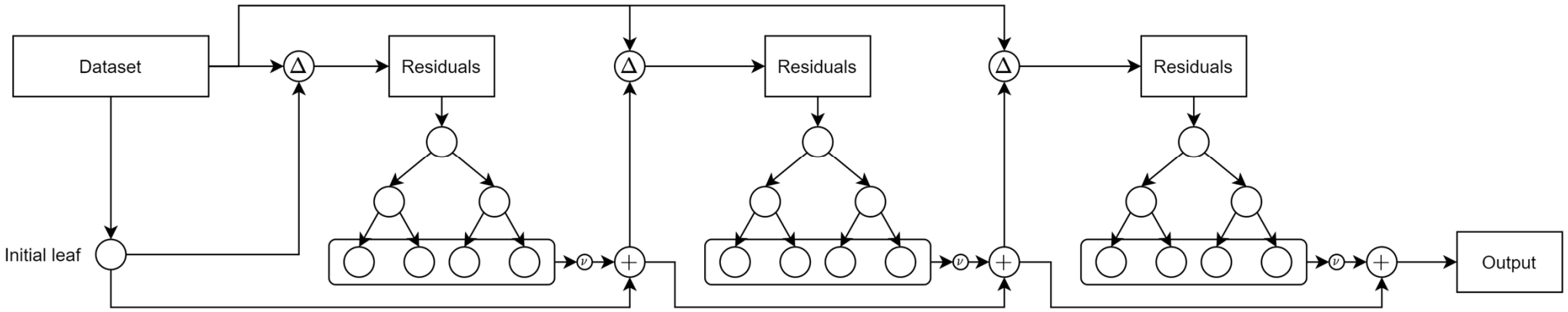
$$F_m(x) = F_{m-1} + \nu \sum_{j=1}^{J} \gamma_{j,m} I(x \in R_{j,m})$$



L05.13 Gradient Boosting

Working Principle

- typically about 100 trees are learned in practice



L05.13 Gradient Boosting

Comparison

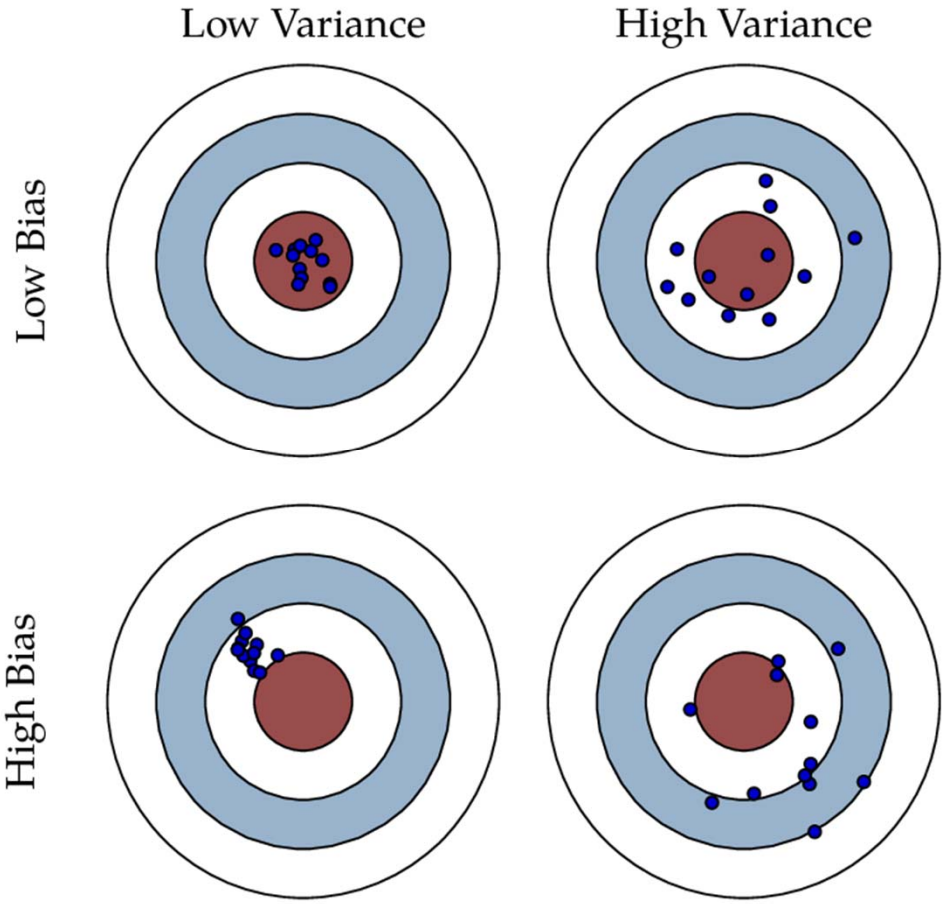
AdaBoost	**Gradient Boosting**
shortcomings identified by high-weight data points	shortcomings identified by the gradient
only decision stumps	larger trees with 8 up to 32 leaves
each tree has different weight based on its performance	all trees are weighed equally,
based on its performance	predictive capacity is restricted by ν
max. variance is captured by weigths for classifiers and observations	variance in data is captured by fitting the residuals

L05.14 Remarks

Bias VS. Variance

- **Boosting**
  - is based on **weak** learners (low variance, high bias)
  - make use of shallow trees (even as small as stumps)
  - reduces error by reducing the bias by aggregating outputs of many models

- **Bagging / Random Forest**
  - based on strong learners (low bias, high variance)
  - make use of fully grown trees
  - reduces error by reducing the model variance by averaging multiple models trained on different subsets of the dataset (Random Forests: random selection of possible splits)



www.hs-kempten.de/ifm