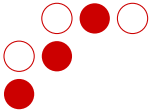
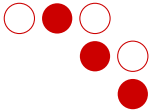
Module B: Tree-based models

Abdulmalek Al-Gahmi, PhD

September 9, 2023

Decision trees

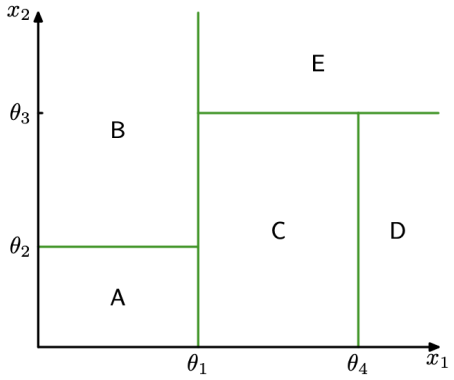
Descision Trees

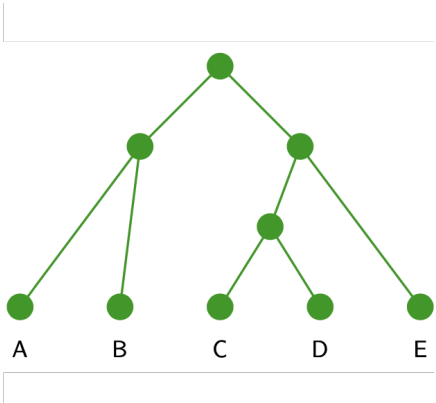
- One of the most widely used ML models in practice
- Based on the very common CS **tree** data structure.
- Simple yet powerful: Can be thought of as a disjunction of conjunctions.
- Can be used for both classification and regression (See for example CARTs: Classification and Regression Trees)
- Partitions the input space into rectangular (cuboid) areas each with a common label (classification) or value (regression)
- Used ensemble learning: a key learning method.

Classification decision trees

An example decision tree [1]

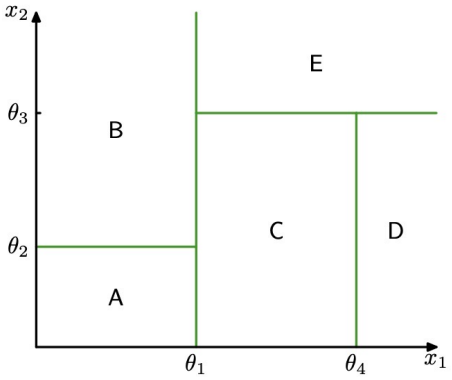
Label this tree to the right with features and decision boundaries:

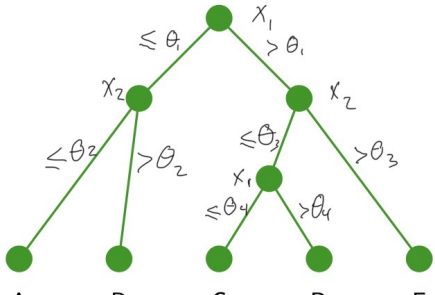




An example decision tree

Label this tree to the right with features and decision boundaries:





Another example [4]: The dataset

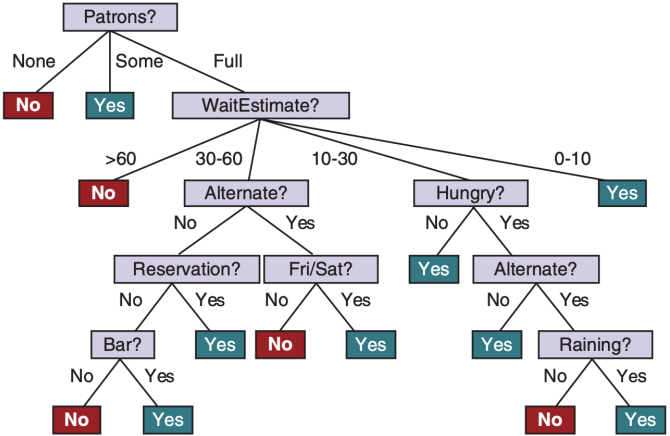
Will you wait to be seated?

Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Alt: An alternative? **Bar**: Has a bar wait in. **Fri**: Fri/Sat? **Hun**: Hungry? **Pat**: How many people? **Price**: The price range. **Rain**: Raining outside? **Res**: Made a reservation? **Type**: The restaurant kind. **Est**: The wait estimate.

Another example [4]: A decision tree

Will you wait to be seated?



Representation

- Trees are rooted graphs without cycles. Among other representations, we can use graph representations such as **adjacency lists** or **adjacency matrix** to represent DTs.
- The topmost node is called the **root**.
- Non-leaf nodes (the root or internal nodes) represent features.
- **Leaf nodes** represent labels. Leaf nodes can have more that one label.
- Nodes are connected with links labeled with feature values.
- A path is the set of links connecting the root node to a leaf.
- Predicting the label (or value) of an unseen data example $\boldsymbol{x}$ given a trained DT involves using its feature values to traverse the tree starting at the root until a leaf node is reached.

Expressiveness

- We can build a consistent DT for any training set such that there is a path from the root to a leaf node for every example. Is that good?
- DTs can express any function of discrete input features.
- DTs can approximate any function with continuous input features.
- As trees, DTs lend themselves well to recursion.

How many splits per node?

- Called the branching factor of the tree.
- Could be two or more.
- Some implementations such as CART use branching factor of 2. In other words, the resulting tree is binary.

Which feature to start with?

The answer determines how the DT is constructed.

- A node is **pure**, if all the examples under it belong to the same class.
- We use a **greedy heuristic** based on a measure of node impurity. That is:
  - 0 if the node is **pure**.
  - largest if all labels are equally represented.
- This usually results in simpler trees but not necessarily optimal.
- Two popular ways to measure impurity: Entropy, and Gini index.

Using Entropy to construct DTs

Given a set of examples D with labels y_i where $i \in \{1, 2, \ldots, L\}$ and L is the number of unique labels, the entropy is defined as:

$$\mathsf{H}(\mathsf{D}) = -\sum_{i=1}^{L} p_i \log p_i$$

We use the entropy to calculate the information gain of a feature F as follows:

$$\mathsf{Gain}(\mathsf{F}, \mathsf{D}) = H(D) - \sum_{f \in values(F)} \frac{|D_f|}{|D|} H(D_f)$$

where D_f is the set of all examples in D where feature F has the value f. Let's create function to calculate the information gain.

More about entroy

- High entropy means:
  - a uniform-like distribution
  - a flat(ish) histogram
  - less predictable
- Low entropy means:
  - many ups and downs
  - more predictable

Using the Gini index to construct DTs

To compute Gini impurity for a set of examples with labels y_i where $i \in \{1, 2, \ldots, L\}$ and L is the number of unique labels, let p_i be the fraction of examples with the label y_i

$$\mathsf{G}(\mathsf{D}) = \sum_{i=1}^{L} \sum_{k \neq i} p_i p_k = \sum_{i=1}^{L} \left(p_i \sum_{k \neq i} p_k \right) = \sum_{i=1}^{L} p_i (1 - p_i) = \sum_{i=1}^{L} p_i - \sum_{i=1}^{L} {p_i}^2 = 1 - \sum_{i=1}^{L} {p_i}^2$$

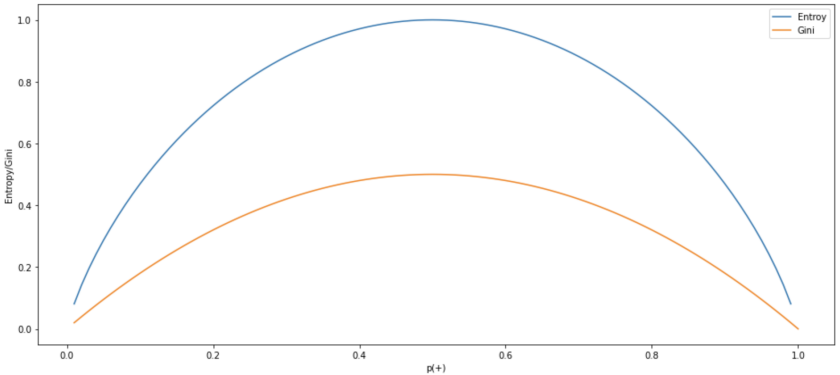
We use the calculate the weighted gini index for a feature F as follows:

$$\mathsf{Gini}(\mathsf{F}, \mathsf{D}) = \sum_{f \in values(F)} \frac{|D_f|}{|D|} G(D_f)$$

where D_f is the set of all examples in D where feature F has the value f.

Information gain vs gini index

- Information gain: used by the ID3 and C4.5 algorithms. Best feature $\rightarrow$ maximum information gain.
- Gini index: used by CART (Classification And Regression Trees). Best feature $\rightarrow$ minimum weighted gini index.



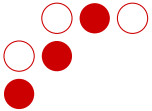
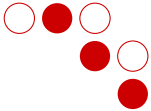
The ID3 (Iterative Dichotomiser 3) algorithm

function ID3(D)
  if all examples in D have same label **then**
    return a leaf with that label
  else if there are no features left to test **then**
    return a leaf with the most common label
  else
    Choose the best feature to split the data with $\bar{F}$
    Use the best feature $\bar{F}$ to create the next node
    Add a branch from the node above for every possible value f of the best feature $\bar{F}$
    for each branch **do**
      Calculate D_f (the data where $\bar{F}$ has the value f)
      remove $\bar{F}$ from the set of features
      ID3(D_f)
    end for
  end if
end function

An example [3]: Construct the decision tree

Construct a decision tree based on the following table? What activity would this tree predict for $(Near, No, Yes)$? Use the information gain.

Deadline?	Party?	Lazy?	Activity
Urgent	Yes	Yes	Go out
Urgent	No	Yes	Stay home
Near	Yes	Yes	Go out
None	Yes	No	Go out
None	No	Yes	Go out
None	Yes	No	Go out
Near	No	No	Stay home
Near	No	Yes	Stay home
Near	Yes	Yes	Go out
Urgent	No	No	Stay home

Regression decision trees

Continuous features and regression

- Numeric and continuous features present a challenge:
  - Categorize or discretize them.
  - Find the best thresholds to split them by based on the appropriate impurity measure.
- Continuous output means regression:
  - report the mean of the output of all the examples under the leaf node.
  - use linear regression on the examples under the leaf node.

Regression trees

- Must know how to handle continuous features and report continuous output.
- Must be able to find out the "best" feature and threshold values to split the data at.
- Use variance (from statistics) as a measure of impurity. Given a set of examples D with a continuous output y, we define the variance of y as:

$$\mathsf{V}(\mathsf{D}) = var(y) = \frac{1}{|D|}\sum_{i}^{|D|}(y_i - \bar{y})^2$$

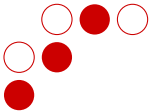
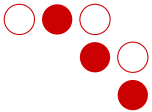
- Interested in calculating the reduction in variance caused by splitting the data based a continuous feature F at a threshold value t. Such reduction is defined as :

$$\mathsf{Reduction}(\mathsf{D},\mathsf{F},\mathsf{t}) = V(D) - [\frac{|D_{F\leq t}|}{|D|}V(D_{F\leq t}) + \frac{|D_{F>t}|}{|D|}V(D_{F>t}]$$

Here the best split threshold corresponds to the least variance reduction.

Advantages and disadvantages

- Advantages:
  - Simple and powerful
  - More interpretable
  - Efficient
  - Unaffected by scales of values
  - Work well when with missing values
  - Work well with discrete features
  - Work well when there are a lot of features but only some of them are important
  - Lend themselves well to ensemble methods
- Disadvantages
  - Difficult to work with continuous features
  - Unstable
  - Suffer from high-variance
  - Tend to overfit especially in large trees

Ensemble methods

overview

- An ensemble of predictors is a set of predictors whose individual decisions are combined in some way such as averaging, voting, etc.
- The the predictors must be different somehow:
  - Different algorithms
  - Same algorithms but different hyperparameters.
  - Different data
- An individual predictor is called a **base model** while a collection of predictors is called an **ensemble model**.
- There are many ways to create ensembles:
  - Voting
  - Stacking
  - Bagging
    - Random forests
  - Boosting

Voting and stacking

- Voting involves training multiple models on the same data and using voting strategy (majority vote, plurality, vote, or the mean in case of regression, etc) to obtain the final he final prediction.
- Stacking involves:
  - training multiple individual models on the same data. These are called first-level models.
  - training a model to combine the results of the individual models. This is called the second-level model, or meta-learner.

Bagging: Overview

- Two reasons to create ensembles:
  - Reduce bias
  - Reduce variance
- Bagging is common with DTs:
  - Smooths them out
  - Helps with their instability

Bagging: Bootstrap aggregation

- Draw T bootstrap samples.
- Train T base models (decision trees are commonly used) one for each sample. Base models of the same type are used.
- Aggregate the predictions of these T trees (somehow) to predict the output of an unseen example.
  - For classification problems, return the plurality vote (majority vote for binary classification) of all the output classes returned by the T decision trees.
  - For regression problems, return the average output:

$$\hat{y} = \frac{1}{T}\sum_{t=1}^{T}\hat{y}_t$$

    where $\hat{y}_t$ is the output from the decision tree trained on sample t.
- Although samples are different, they are correlated and the trained trees are similar.

Bootstrap samples

- Given a dataset *D* with *N* examples, a bootstrap sample is a sample of size *N* drawn randomly **with replacement** from the dataset *D*.
- Some examples might show multiple times in the sample.
- Some examples might not show at all in the sample. We call these examples **out of bag** examples.

Bootstrap sample: An example

Given the dataset to right, Here are 10 bootstrap samples each of size 10.

```
S0: 1,9,6,6,8,5,2,4,5,7
S1: 4,5,3,9,2,2,3,9,5,1
S2: 4,2,2,3,9,0,4,5,4,9
S3: 1,9,6,3,8,5,6,9,4,7
S4: 1,6,6,2,6,1,7,8,6,1
S5: 6,0,2,4,2,2,8,6,3,1
S6: 7,5,4,3,1,9,5,1,5,1
S7: 8,5,0,0,8,4,0,7,0,0
S8: 1,4,3,0,5,8,3,0,4,0
S9: 8,0,0,6,3,0,8,1,0,6
```

i	Deadline?	Party?	Lazy?	Activity
0	Urgent	Yes	Yes	Go out
1	Urgent	No	Yes	Stay home
2	Near	Yes	Yes	Go out
3	None	Yes	No	Go out
4	None	No	Yes	Go out
5	None	Yes	No	Go out
6	Near	No	No	Stay home
7	Near	No	Yes	Stay home
8	Near	Yes	Yes	Go out
9	Urgent	No	No	Stay home

Random forests

- One of the most popular ensemble methods.
- Based on bagging applied exclusively to decision trees.
- In addition to the bootstrap samples, RF requires that only a random subset of features is considered at each node during the construction of the tree. This is called **feature bagging**.
- The idea is to make the base decision trees as different as possible.

Out of bag score/error

- Used to measure the prediction error/accuracy of random forests.
- It is the mean error on each example x, using only the trees that did not have x in their bootstrap sample.
- For classification:
  - Find the predicted classes for example x using the trees that were trained on samples not including x.
  - Find the plurality vote of these classes and report it as the final predicted class.
  - Use the confusion matrix to compare the actual classes to the predicted classes of all the dataset.

An example

Given the following 5 bootstrap samples, calculate the out of bag error?

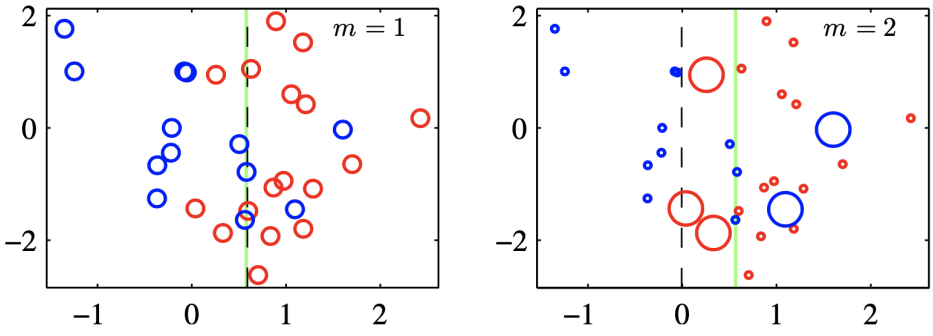
```
S0: 1,9,6,6,8,5,2,4,5,7
S1: 7,5,4,3,1,9,5,1,5,1
S2: 8,5,0,0,8,4,0,7,0,0
S3: 1,4,3,0,5,8,3,0,4,0
S4: 8,0,0,6,3,0,8,1,0,6
```

i	Deadline?	Party?	Lazy?	Activity	out of bag
0	Urgent	Yes	Yes	Go out	
1	Urgent	No	Yes	Stay home	
2	Near	Yes	Yes	Go out	
3	None	Yes	No	Go out	
4	None	No	Yes	Go out	
5	None	Yes	No	Go out	
6	Near	No	No	Stay home	
7	Near	No	Yes	Stay home	
8	Near	Yes	Yes	Go out	
9	Urgent	No	No	Stay home	

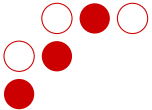
Boosting

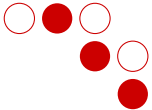
- Can a set of “weak learners” be used to create a strong learner?
- A weak learner is one that is doing slightly better than randomly selecting a class.
- In boosting, new models are added to correct the errors made by existing models. In other words, it gives misclassified examples more say.

Boosting



[Source: Pattern Recognition and Machine Learning]





AdaBoost

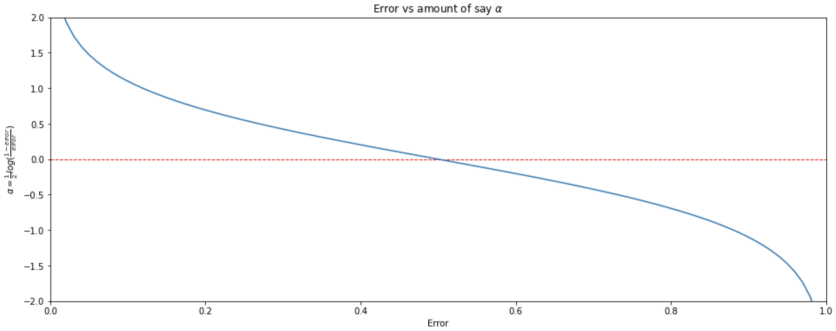
AdaBoost

- Works for binary classification
- Three main ideas:
  - Use of weak learners
  - Base models are not independent; they help make other base models.
  - Not all models have the same say.
- Data preprocessing: code your classes as -1 and 1

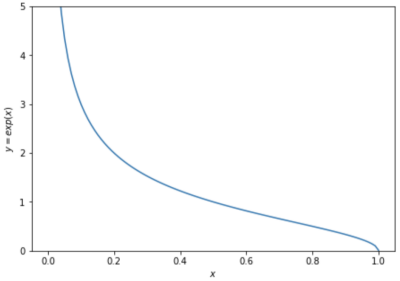
AdaBoost

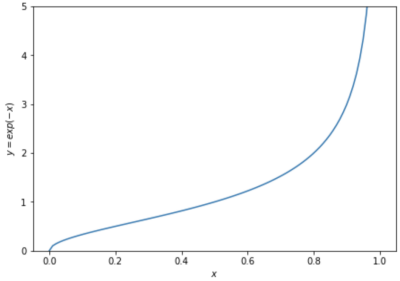
1. Given a number T and dataset D of size N
2. Initialize sample weights uniformly as $w_i = \frac{1}{N}$ for each example i in D
3. For $t = 1$ to T
   - 3.1 Get a dataset D_t that is distributed according to w
   - 3.2 Train a weak learner h_t on D_t
   - 3.3 Compute training error $\epsilon_t = \sum_{i=1}^{N} \mathbf{I}[h_t(x_i) \neq y_i] \, w_i^{(t)}$
   - 3.4 Compute amount of say $\alpha_t = \frac{1}{2} \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
   - 3.5 Update weights for each example i in D: $w_i = w_i \cdot e^{-\alpha_t y_i h_t(x_i)}$
   - 3.6 Re-normalize weights such that $\sum_{i=1}^{N} w_i = 1$
4. Final prediction of an unseen example x: $h(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

AdaBoost: α (amount of say)



AdaBoost: How weights are updated

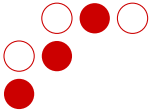
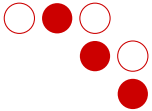




An example [2]

Assuming that the best feature given the data below is **Party?**. Draw out the corresponding decision stump.

Deadline?	Party?	Lazy?	Activity	$w^{(t)}$	$w^{(t+1)}$
Urgent	Yes	Yes	Party		
Urgent	No	Yes	Study		
Near	Yes	Yes	Party		
None	Yes	No	Party		
None	No	Yes	Pub		
None	Yes	No	Party		
Near	No	No	Study		
Near	No	Yes	TV		
Near	Yes	Yes	Party		
Urgent	No	No	Study		

Gradient boosting

Gradient boosting

- is one of the best known predictive models of structured data.
- is popular among industrial applications and in machine learning competitions.
- has many implementations: XGBoost, NGBoost, LightGBM, and the gradient boosting machine (GBM).
- trains an ensemble of models based on loss gradients.
- adds new boosting models (usually trees with 8 to 32 leaves), which pay attention not to specific examples, but to the gradient between the right answers and the answers given by the previous models.
- requires a differentiable loss function $L(y, \hat{y})$; usually the squared error for regression and logarithmic loss for classification.

The general gradient boosting algorithm

Input: A training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value: $F_0(x) = \arg\min_{\hat{y}} \sum_{i=1}^n L(y_i, \hat{y})$
2. For $m = 1$ to M:
   - 2.1 Compute the so-called pseudo-residuals:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n$$

   - 2.2 Fit a regression tree (a weak learner) to pseudo-residuals r_{im} and train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$. Use R_{jm} to denote the $j = 1, \cdots J_m$ leaf nodes of the tree of iteration m
   - 2.3 For each leaf node R_{jm}, compute the following output value:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L\left(y_i, F_{m-1}(x_i) + \gamma\right)$$

   - 2.4 Update the model: $F_m(x) = F_{m-1}(x) + \eta \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}(x \in R_{jm})$ where η is the learning rate.
3. Output $F_M(x)$

Gradient boosting for regression

1. $L = (y - \hat{y})^2$ and $F_0(x) = \underset{\hat{y}}{\arg\min} \sum_{i=1}^{n} L(y_i, \hat{y}) = \bar{y}$
2. For $m = 1$ to M:
   2.1 Compute the so-called pseudo-residuals, which leads to

$$r_{im} = -\frac{\partial(y_i, F(x_{m-1}))}{\partial F(x_{m-1})} = 2(y_i - F(x_{m-1}))$$

   2.2 Fit a regression tree (a weak learner) to pseudo-residuals r_{im} and train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   2.3 Compute multiplier γ_{jm}, which leads to: $\gamma_{jm} = \frac{1}{n} \sum_{i=1}^{n} r_{im}$
   2.4 Update the model: $F_m(x) = F_{m-1}(x) + \eta \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}(x \in R_{jm})$ where η is the learning rate.
3. Output $F_M(x)$

Gradient boosting for classification

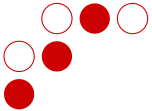
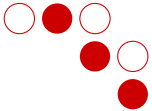
1. $L_i = -y_i \log p_i + (1 - y_i) \log(1 - p_i) = \log(1 + e^{\hat{y}_i}) - y_i \hat{y}_i$ where $\hat{y}_i = \log(odds) = \log(\frac{p_i}{1-p_i})$ and $F_0(x) = \arg\min_{\hat{y}} \sum_{i=1}^{n} L(y_i, \hat{y}) = \log(\frac{\bar{y}}{1-\bar{y}})$
2. For $m = 1$ to M:
   2.1 Compute the so-called pseudo-residuals, which leads to

$$r_{im} = -\frac{\partial(y_i, F(x_{m-1}))}{\partial F(x_{m-1})} = y_i - p_i$$

   2.2 Fit a regression tree (a weak learner) to pseudo-residuals r_{im} and train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   2.3 Compute multiplier γ_{jm}, which leads to:

$$\gamma_{jm} = \frac{\sum_{x_i \in R_{jm}} y_i - p_i}{\sum_{x_i \in R_{jm}} p_i(1 - p_i)}$$

   2.4 Update the model: $F_m(x) = F_{m-1}(x) + \eta \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}(x \in R_{jm})$ where η is the learning rate.
3. Output $F_M(x)$

Questions?

References I

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. isbn: 0387310738.

[2] S. Marsland. *Machine Learning: An Algorithmic Perspective, Second Edition*. Chapman & Hall. CRC Press, 2014. isbn: 9781466583337. url: `https://books.google.com/books?id=6GvSBQAAQBAJ`.

[3] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. url: `probml.ai`.

[4] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson series in artificial intelligence. Pearson, 2021. isbn: 9781292401133.