TREES, FORESTS AND BOOSTING

KNOWLEDGE SHARING SESSION

Rostislav Stoyanov

May 28, 2025

Part I

TREES AND FORESTS

Trees and forests

Decision trees

- Structure consisting of (root, internal and left) nodes connected with branches.
- Each node represents a test on a data feature and each branch from the node represent an outcome from the test.
- Nodes without children are called leafs. Each leaf represents a prediction outcome of the model.

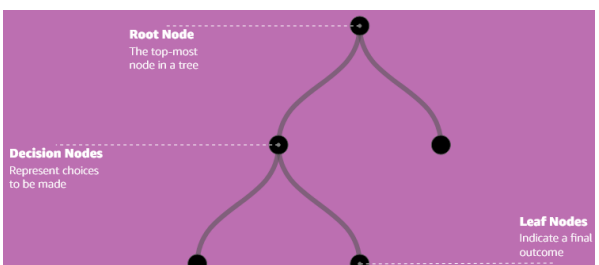


Figure. Tree structure parts[1]

[1] Picture taken from: `https://mlu-explain.github.io/decision-tree/`

- Trees are build in a top-down fashion by performing a greedy search. Multiple algorithms, which follow similar approach.
- Starting from the root node, a recursive algorithm is used to find the most optimal current split (test) at a node.
- The selection of a split at a node is based on a metric. Using the samples reaching the node, this metric is calculated in order to find which of all possible splits best optimizes the algorithm's objective.
- This process is repeated until some end condition is met (e.g. not enough data to split, maximum depth is reached).

Definition 1.1 (Shannon entropy)

Given a discrete random variable X, which takes values in the alphabet $\mathcal{X}$ and is distributed according to $p : \mathcal{X} \to [0, 1]$:

$$\mathrm{H}(X) := -\sum_{x \in \mathcal{X}} p(x) \log p(x) = \mathbb{E}[-\log p(X)] \tag{1}$$

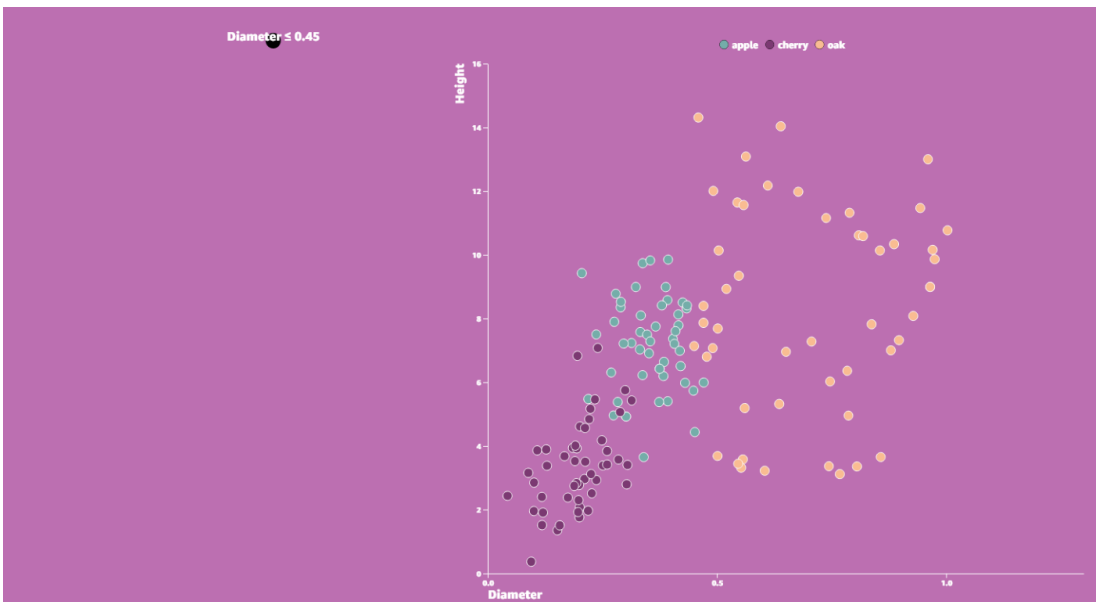
Definition 1.2 (Information gain)

Let T denote a set of training examples and α be an attribute describing the training examples. Furthermore, let $vals(\alpha)$ contains all possible values of a and $S_a(v) := \{x \in T | x_a = v\}$ be the set of training inputs for which the attribute $\alpha = v$.

$$\mathrm{IG}(T, \alpha) := \mathrm{H}(T) - \sum_{v \in vals(a)} \frac{|S_\alpha(v)|}{|T|} H(S_\alpha(v)) = \mathrm{H}(T) - \mathrm{H}(T \mid \alpha) \tag{2}$$

Decision trees

Training visualized



Source: `https://mlu-explain.github.io/decision-tree/`

Decision trees

Advantages of decision trees

- Can solve non-linear problems.
- Non-parametric models.
- Can handle various both categorical and numerical data. Furthermore, missing values are also supported.
- Easy to interpret.

Decision trees

Issues with decision trees

- Have to re-calculate the tree for new examples.
- "Vanilla" decision trees can't extrapolate.
- Overfitting. Complex trees fit the training data too well and do not generalize to new data.
  - We can use pre-pruning or post-pruning to mitigate this problem.
- Unstable. Small changes in the training samples can lead to very different decision trees.

Decision trees

High variance visualized

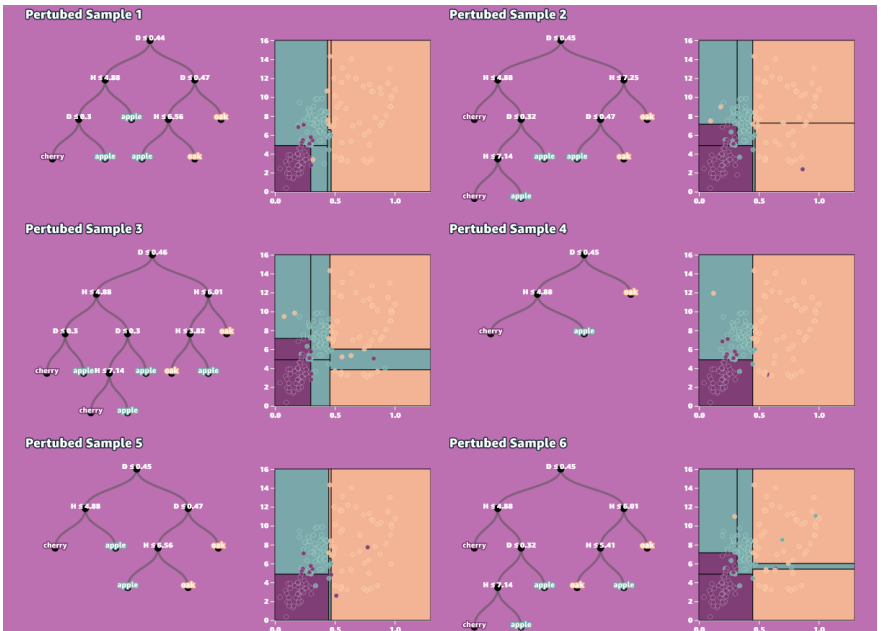


Figure. Example of trees build when changing the class of 5% of the training samples [2]

[2] Source: `https://mlu-explain.github.io/decision-tree/`

Ensemble learning

- Use ensemble learning to mitigate the weaknesses of decision trees.
- Ensemble learning is a paradigm in which multiple "weak learners" are trained on the same problem to achieve better performance.
- We will explore two methods:
  - Bagging (random forest)
  - Boosting (AdaBoost, gradient boosting)

BAGGING
BOOTSTRAPING

- Generate new samples of size B from a dataset with N samples using uniform sampling with replacement.
- Boostrapping needs two properties of the original dataset to work correctly (the samples are representative and independent samples of the true data distribution)[1]:
  - Representativity - The dataset is big enough to capture the underlying distributin.
  - Independence - The dataset is big enough so samples aren't correlated much.

Figure. Bootstrap [1]

[1] Source: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Bagging

Bagging algorithm

- Given a training set $D, |D| = n$, bootstrap m new training sets $D_i, i = 1, \ldots, m, |D_i| = n'$.
- Using each of the training sets D_i train a machine learning model M_i.
- The output of the ensemble is the average of all model outputs (regression) or voting (classification).
- The final model has less variance than a single trained model.

Bagging

Random forest

- Random forest uses bagging to train multiple decision trees. We are using bagging to minimize variance, so use deep trees.
- Additionally, for each decision tree trained keep only a part of the features. This has two advantages:
  - Reduce correlation of the trees – if a feature is a very strong predictor it will be used in all trees making them correlated.
  - Increase robustness to missing data. Some of the trees will only have features without missing data.

Bagging

Advantages of random forest

- Similar to decision trees can handle different types of features and solve non-linear problems.
- Better performance than a single decision tree.
- Low risk of overfitting.

Bagging

Disadvantages of random forest

- Loss of interpretability (compared to decision trees).
- Can't extrapolate.
- Typically higher bias than a single tree [Has+09].
- Very dependent on the bootstrapped datasets. Recreation of experiment would require the same sampling (keeping track of the seeds).
- Needs modifications to handle class imbalance.

Part II

BOOSTING

- Similarly to bagging uses many "weak" models to produce a strong ensemble model ("committee"). The two approaches, however, are fundamentally different.
- Models are fit iteratively (sequentially) so that they complement each other.
- Each new model attempts to improve prediction on samples for which the previous models have been wrong.

Algorithm AdaBoost [Has+09]

1: Initialize the observation weights $w_i = 1/N, i = 1, 2, \ldots, N$.
2: **for** $m = 1, \ldots, M$ **do**
3: Fit a classifier $G_m(x)$ to the training data using weights w_i.
4: Compute

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I\left(y_i \neq G_m\left(x_i\right)\right)}{\sum_{i=1}^{N} w_i}.$$

5: Compute $\alpha_m = \log\left(\left(1 - \text{err}_m\right) / \text{err}_m\right)$.
6: Set $w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot I\left(y_i \neq G_m\left(x_i\right)\right)\right], i = 1, 2, \ldots, N$.
7: **end for**
8: Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

- On the right, AdaBoost fit for a toy example.
- Fit for 10 iterations.
- Weak learners are decision stumps with depth 1 and 2 leaf nodes.

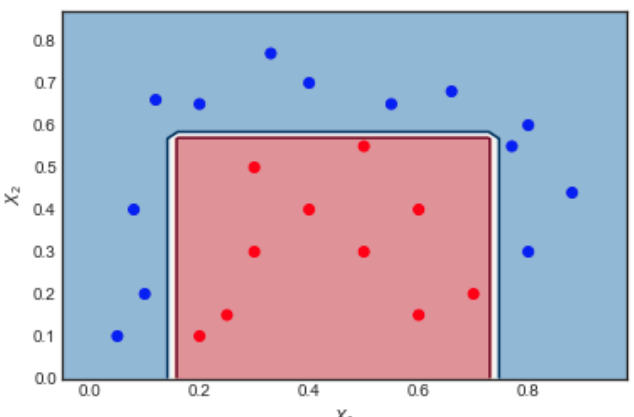


Figure. AdaBoost on a toy example[3]

[1] Picture taken from: `https://xavierbourretsicotte.github.io/AdaBoost.html`

Boosting

AdaBoost

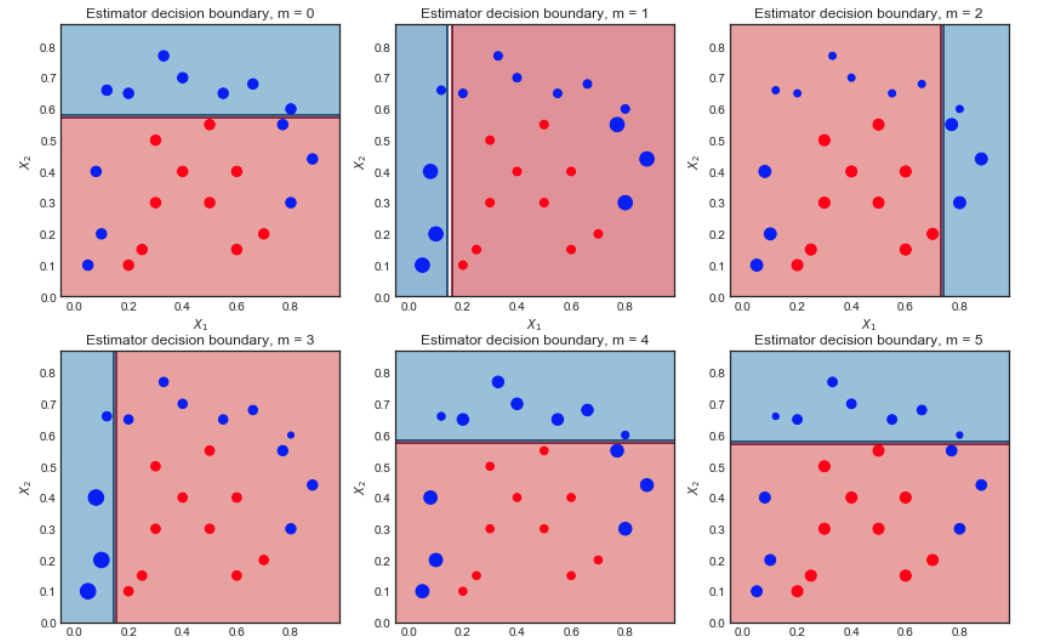


Figure. Visualization of the sequence of weak learners and the sample weights

- Advantages:
  - Improves performance of weak classifiers fast.
  - Easy to use and tune, not a lot of parameters than a single classifier.
  - Less prone to overfitting.
- Drawbacks:
  - Sensitive to noisy data and outliers.
  - Hard to extend to other tasks.

Additive modeling

- Boosting success comes from fitting an additive expansion:

$$f(x) = \sum_{m=1}^{M} \beta_m b\left(x; \gamma_m\right) \tag{3}$$

  In the case of AdaBoost:
  - $b\left(x; \gamma\right) \in \mathbb{R}$ are functions of multivariate argument x, with parameters γ. AdaBoost uses trees as basis functions, with γ being the split variables and points as well as the leaf predictions.
  - The expansion coefficients $\beta_m; m = 1, 2, \ldots, M$ are the model weights α_m.
- Fitting is done by minimizing a loss over all training examples:

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b\left(x_i; \gamma_m\right)\right). \tag{4}$$

- Fitting 4 is very expensive, so instead we can approximate by adding new functions without modifiying the previous ones:

Algorithm Forward Stagewise Additive Modeling [Has+09]

1: Initialize $f_0(x) = 0$.
2: **for** $m = 1, \ldots, M$ **do**
3: Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(x_i\right) + \beta b\left(x_i; \gamma\right)\right).$$

4: Set $f_m(x) = f_{m-1}(x) + \beta_m b\left(x; \gamma_m\right)$.
5: **end for**

- It can be proven that AdaBoost is a forward stagewise additive modeling with loss function: $L(y, f(x)) = exp(-y\, f(x))$
- We need to be able to solve step 3 for general loss criteria.

FUNCTIONAL GRADIENT DESCENT

- Explore numerical optimization approaches.
- Want to minimize:

$$J(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$$

- However $J(f)$ depends on f only on N training points (vector of model predictions on training points):

$$\mathbf{f} = \{f(x_1), f(x_2), \ldots, f(x_N)\}^T.$$

- So we can rewrite $J(f)$:

$$J(f) = \sum_{i=1}^{N} L(y_i, \mathbf{f})$$

Functional gradient descent

Steepest descent

- Perform gradient descent on:

$$J(f) = \sum_{i=1}^{N} L(y_i, \mathbf{f}) \ ,$$

  where $\mathbf{f}_0 = \mathbf{h}_0$ is an initial guess, and each successive $\mathbf{f}_m$ is induced based on the current parameter vector $\mathbf{f}_{m-1}$, which is the sum of the previously induced updates.
- The negative gradient direction at $\mathbf{f} = \mathbf{f}_{m-1}$ is $\mathbf{g}_m$:

$$g_{im} = \left[\frac{\partial L\left(y_i, f\left(x_i\right)\right)}{\partial f\left(x_i\right)} \right]_{f(x_i)=f_{m-1}(x_i)}$$

- We also need a weight ρ_m to perform an update. Two options:
  - $\rho_m = \arg\min_\rho L\left(\mathbf{f}_{m-1} - \rho\, \mathbf{g}_m\right)$.
  - Some constant value (e.g. 0.01).
- $\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m$

Functional gradient descent

Steepest descent

- $-g_m \in \mathbb{R}$ is the direction in which we want to update our training data predictions.
- We need more than f, which is predictions on training data.
  - Might not be defined for test points.
  - Might not exist a hypothesis (tree) that generates this outcome on the training data.

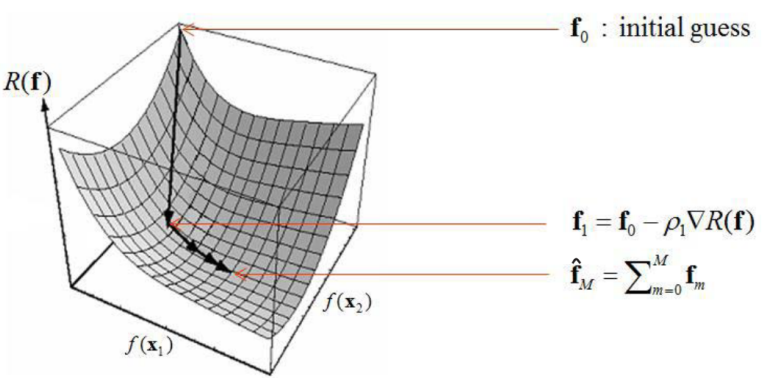


Figure. Steepest descent illustration. The "risk" criterion R(f), is plotted as a function of f evaluated at two training data points x1 and x2. [SE10]

- The gradient is defined only on the N training points and we want to generalize $\mathbf{f}_m$.
- A solution is to find the hypothesis $h \in \mathcal{H}$ that best approximates -g as out step direction.
- Approach this a least squares regression problem over $\mathcal{H}$:

$$\min_{h \in \mathcal{H}} \sum_{i=1}^{N} (-g - h(x_i))^2$$

In other words, we are fitting our model over the pseudo-residuals.

Functional gradient descent

Gradient descent

- $T(x; p) \in \mathcal{H}$ is the actual step direction (model) we are using.
- Can think of it as projection of -g onto $\mathcal{H}$.

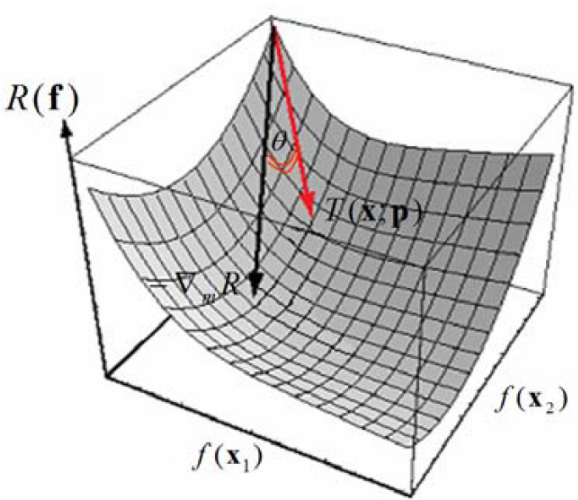


Figure. Surrogate-loss illustration. The base-learner most parallel to the negative gradient vector is chosen at every step of the gradient-descent algorithm [SE10]

Algorithm Gradient boosting [Fri01]

1: $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^N L(y_i, \rho)$
2: **for** $m = 1, \ldots, M$ **do**
3: $\quad \tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, i = 1, \ldots, N$
4: $\quad \mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^N [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
5: $\quad \rho_m = \arg\min_\rho \sum_{i=1}^N L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$
6: $\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
7: **end for**

Next time

- XGBoost and friends.
- Handling categorical features.

USEFUL RESOURCES

- For trees [MM97]
- Random forest + boosting `https://www.youtube.com/watch?v=fz1H03ZKvLM`
- Gradient boosting `https://www.youtube.com/watch?v=wPqtzj5VZus`

References I

[Fri01] Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232.

[Has+09] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[MM97] Tom M Mitchell and Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997.

[SE10] Giovanni Seni and John F Elder. "Ensemble methods in data mining: improving accuracy through combining predictions". In: *Synthesis lectures on data mining and knowledge discovery* 2.1 (2010), pp. 1–126.