

Model Combination, Decision Trees, Random Forests

Jindřich Libovický (reusing materials by Milan Straka)

 November 24, 2025

After this lecture you should be able to

- Ensemble models with uncorrelated predictions.
- Distill ensembles into smaller models.
- Implement Decision Trees and Random Forests for classification and regression
- Explain how the splitting criterion depend on optimized loss function

Model Combination

The goal of **ensembling** is to combine several models in order to reach higher performance.

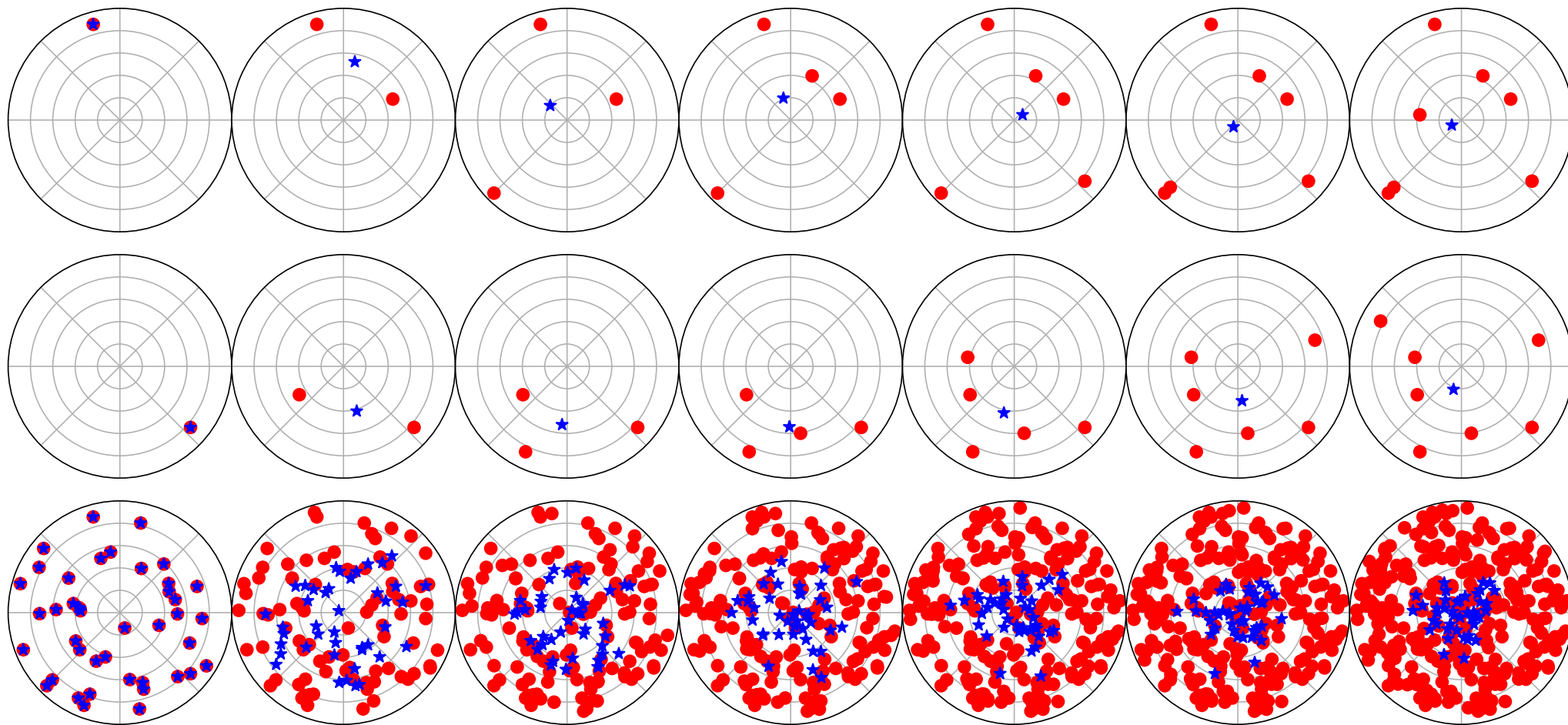
The simplest approach is to train several independent models and then combine their predictions by averaging or voting.

The terminology varies, but for classification:

- voting (or hard voting) usually means predicting the class predicted most often by the individual models,
- averaging (or soft voting) denotes averaging the returned model distributions and predicting the class with the highest probability.

The main idea behind ensembling is that if models have uncorrelated errors, then by averaging model predictions the errors will cancel out.

Consider ensembling predictions generated uniformly on a planar disc:



If we denote the prediction of a model y_i on a training example (\mathbf{x}, t) as $y_i(\mathbf{x}) = t + \varepsilon_i(\mathbf{x})$, so that $\varepsilon_i(\mathbf{x})$ is the model error on example \mathbf{x} , the mean square error of the model is

$$\mathbb{E}[(y_i(\mathbf{x}) - t)^2] = \mathbb{E}[\varepsilon_i^2(\mathbf{x})].$$

Considering M models, we analogously get that the mean square error of the ensemble is

$$\mathbb{E}\left[\left(\frac{1}{M} \sum_i \varepsilon_i(\mathbf{x})\right)^2\right].$$

Finally, assuming that the individual errors ε_i have zero mean and are *uncorrelated*, we get that $\mathbb{E}[\varepsilon_i(\mathbf{x})\varepsilon_j(\mathbf{x})] = 0$ for $i \neq j$, and therefore,

$$\mathbb{E}\left[\left(\frac{1}{M} \sum_i \varepsilon_i(\mathbf{x})\right)^2\right] = \mathbb{E}\left[\frac{1}{M^2} \sum_{i,j} \varepsilon_i(\mathbf{x})\varepsilon_j(\mathbf{x})\right] = \frac{1}{M} \mathbb{E}\left[\frac{1}{M} \sum_i \varepsilon_i^2(\mathbf{x})\right],$$

so the average error of the ensemble is $\frac{1}{M}$ times the average error of the individual models.

Bagging – Bootstrap Aggregation

For neural network models, training models with independent random initialization is usually enough, given that the loss has many local minima, so the models tend to be quite independent just when using different random initialization.

However, algorithms with convex loss functions usually converge to the same optimum independent of randomization.

In these cases, we can use **bagging**, which stands for **bootstrap aggregation**.

In bagging, we construct a different dataset for every model we train. We construct it using **bootstrapping** – we sample as many training instances as the original dataset has, but **with replacement**.

Such dataset is sampled using the same empirical data distribution and has the same size, but is not identical.

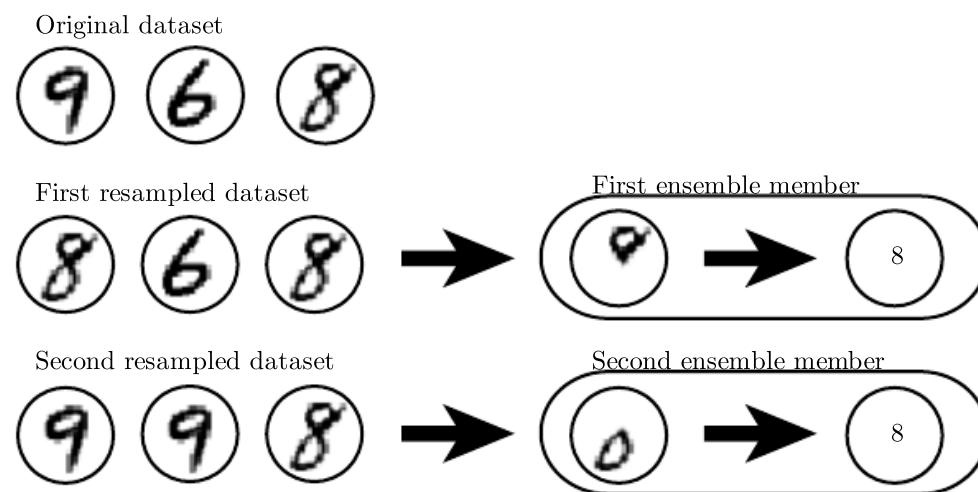


Figure 7.5 of "Deep Learning" book, <https://www.deeplearningbook.org>

- Model ensemble might be too slow or too big to use.
- Knowledge distillation = training a **student** model that mimics behaviour of a **teacher model** (a bigger one or model ensemble).

Algorithm:

1. Process training data (or additional unlabelled data) with the best current model and get the output distribution $p_{\text{teacher}}(\mathbf{y}|\mathbf{x}; \mathbf{w})$ (sometimes called *pseudolikelihood*)
2. Train a model with $H(p_{\text{teacher}}(\mathbf{y}|\mathbf{x}; \mathbf{w}), p_{\text{student}}(\mathbf{y}|\mathbf{x}; \mathbf{w}))$ as a training objective.

Intuition: Complete distribution provides stronger supervision than just one-hot target, so it is easier for the smaller model to learn from such synthetic data.

Historical note: Term knowledge distillation comes from a 2015 paper by Geoffrey Hinton et al. Before, a similar approach was called model compression.

Decision Trees

Decision Trees

The idea of decision trees is to partition the input space into regions and solving each region with a simpler model.

We focus on **Classification and Regression Trees** (CART; Breiman et al., 1984), but there are additional variants like ID3, C4.5, ...

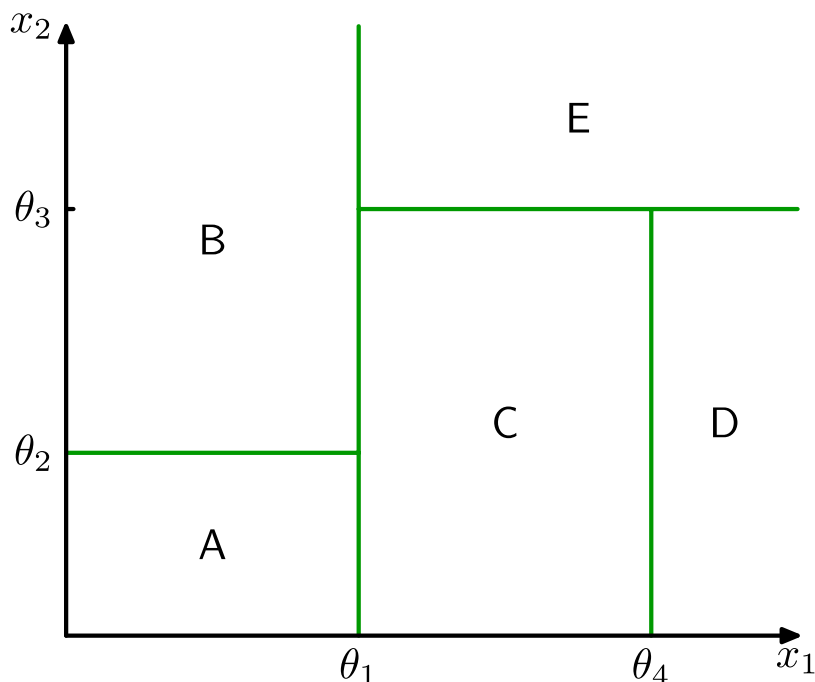


Figure 14.6 of Pattern Recognition and Machine Learning.

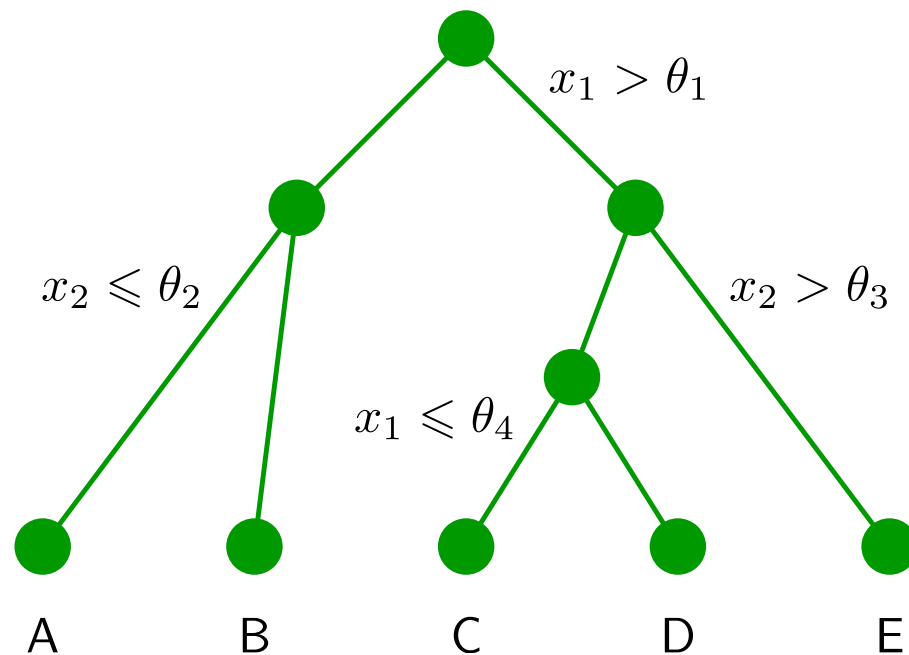


Figure 14.5 of Pattern Recognition and Machine Learning.

Inference

- Just follow the branching rules until you reach a leaf.
- Output a prediction (real value/distribution/predicted class) based on the leaf.

Training

- Training data is stored in tree leaves -- the leaf prediction is based on what is data items are in the leaf.
- At the beginning the tree is a single leaf node.
- Adding a node = leaf \rightarrow decision node + 2 leaves
- The goal of training = finding the most consistent leaves for the prediction

Later, we will show that the consistency measures follow from the loss function, we are optimizing.



Assume we have an input dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$, $\mathbf{t} \in \mathbb{R}^N$. At the beginning, the decision tree is just a single node and all input examples belong to this node. We denote $I_{\mathcal{T}}$ the set of training example indices belonging to a node \mathcal{T} .

For each leaf (a node without children), our model predicts the average of the training examples belonging to that leaf, $\hat{t}_{\mathcal{T}} = \frac{1}{|I_{\mathcal{T}}|} \sum_{i \in I_{\mathcal{T}}} t_i$.

We use a **criterion** $c_{\mathcal{T}}$ telling us how *uniform* or *homogeneous* the training examples of a node \mathcal{T} are – for regression, we employ the sum of squares error between the examples belonging to the node and the predicted value in that node; this is proportional to the variance of the training examples belonging to the node \mathcal{T} , multiplied by the number of the examples. Note that even if it is not *mean* squared error, it is sometimes denoted as MSE.

$$c_{\text{SE}}(\mathcal{T}) \stackrel{\text{def}}{=} \sum_{i \in I_{\mathcal{T}}} (t_i - \hat{t}_{\mathcal{T}})^2, \text{ where } \hat{t}_{\mathcal{T}} = \frac{1}{|I_{\mathcal{T}}|} \sum_{i \in I_{\mathcal{T}}} t_i.$$

To split a node, the goal is to find

1. A feature and (i.e., a for loop over all features)
2. Its value (i.e., a for loop over all unique feature values)

such that when splitting a node \mathcal{T} into \mathcal{T}_L and \mathcal{T}_R , the resulting regions decrease the overall criterion value the most, i.e., the difference $c_{\mathcal{T}_L} + c_{\mathcal{T}_R} - c_{\mathcal{T}}$ is the lowest.

We usually employ several constraints, the most common ones are:

- **maximum tree depth**: we do not split nodes with this depth;
- **minimum examples to split**: we only split nodes with this many training examples;
- **maximum number of leaf nodes**: we split until we reach the given number of leaves.

The tree is usually built in one of two ways:

- if the number of leaf nodes is unlimited, we usually build the tree in a depth-first manner, recursively splitting every leaf until one of the above constraints is invalidated;
- if the maximum number of leaf nodes is given, we usually split such leaf \mathcal{T} where the criterion difference $c_{\mathcal{T}_L} + c_{\mathcal{T}_R} - c_{\mathcal{T}}$ is the lowest.

Terminological note: Decision tree with unlimited size can be considered a non-parametric model: it is a way of building an index. With a limited size, it has a fixed number of parameters to be learned and it can be considered a parametric model.

For multi-class classification, we predict the class which is the most frequent in the training examples belonging to a leaf \mathcal{T} .

To define the criteria, let us denote the average probability for class k in a region \mathcal{T} as $p_{\mathcal{T}}(k)$.

For classification trees, one of the following two criteria is usually used:

- **Gini index**, also called **Gini impurity**, measuring how often a randomly chosen element would be incorrectly labeled if it was randomly labeled according to $\mathbf{p}_{\mathcal{T}}$:

$$c_{\text{Gini}}(\mathcal{T}) \stackrel{\text{def}}{=} |I_{\mathcal{T}}| \sum_k p_{\mathcal{T}}(k)(1 - p_{\mathcal{T}}(k)),$$

- **Entropy Criterion**

$$c_{\text{entropy}}(\mathcal{T}) \stackrel{\text{def}}{=} |I_{\mathcal{T}}| \cdot H(\mathbf{p}_{\mathcal{T}}) = -|I_{\mathcal{T}}| \sum_{\substack{k \\ p_{\mathcal{T}}(k) \neq 0}} p_{\mathcal{T}}(k) \log p_{\mathcal{T}}(k).$$

- Training GLMs and MLPs is formulated as optimizing a loss function.
- For an already constructed decision tree, we can do it the same way. For each leaf, do the optimization and find the best parameter.
- So far, we were always interested in $\arg \min$, i.e., parameters that minimize the loss.
- If we plug the $\arg \min$ value in the loss function, we get the minimum reachable loss for the given tree structure.
- By splitting a leaf, we want to decrease the minimum reachable loss \Rightarrow the **minimum node loss is the splitting criterion**.

Gini and Entropy Losses

Binary Gini as (M)SE Loss

Recall that $I_{\mathcal{T}}$ denotes the set of training example indices belonging to a leaf node \mathcal{T} , let $n_{\mathcal{T}}(0)$ be the number of examples with target value 0, $n_{\mathcal{T}}(1)$ be the number of examples with target value 1, and let $p_{\mathcal{T}} = \frac{1}{|I_{\mathcal{T}}|} \sum_{i \in I_{\mathcal{T}}} t_i = \frac{n_{\mathcal{T}}(1)}{n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1)}$.

Consider sum of squares loss $L(p) = \sum_{i \in I_{\mathcal{T}}} (p - t_i)^2$.

By setting the derivative of the loss to zero, we get that the p minimizing the loss fulfills $|I_{\mathcal{T}}|p = \sum_{i \in I_{\mathcal{T}}} t_i$, i.e., $p = p_{\mathcal{T}}$.

The value of the loss is then

$$\begin{aligned} L(p_{\mathcal{T}}) &= \sum_{i \in I_{\mathcal{T}}} (p_{\mathcal{T}} - t_i)^2 = n_{\mathcal{T}}(0)(p_{\mathcal{T}} - 0)^2 + n_{\mathcal{T}}(1)(p_{\mathcal{T}} - 1)^2 \\ &= \frac{n_{\mathcal{T}}(0)p_{\mathcal{T}}^2}{(n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1))^2} + \frac{n_{\mathcal{T}}(1)(1 - p_{\mathcal{T}})^2}{(n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1))^2} = \frac{(n_{\mathcal{T}}(1) + n_{\mathcal{T}}(0))n_{\mathcal{T}}(0)n_{\mathcal{T}}(1)}{(n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1))(n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1))} \\ &= (n_{\mathcal{T}}(0) + n_{\mathcal{T}}(1))(1 - p_{\mathcal{T}})p_{\mathcal{T}} = |I_{\mathcal{T}}| \cdot p_{\mathcal{T}}(1 - p_{\mathcal{T}}). \end{aligned}$$

Again let $I_{\mathcal{T}}$ denote the set of training example indices belonging to a leaf node \mathcal{T} , let $n_{\mathcal{T}}(k)$ be the number of examples with target value k , and let $p_{\mathcal{T}}(k) = \frac{1}{|I_{\mathcal{T}}|} \sum_{i \in I_{\mathcal{T}}} [t_i = k] = \frac{n_{\mathcal{T}}(k)}{|I_{\mathcal{T}}|}$.

Consider a distribution \mathbf{p} on K classes and non-averaged NLL loss $L(\mathbf{p}) = \sum_{i \in I_{\mathcal{T}}} -\log p_{t_i}$.

By setting the derivative of the loss with respect to p_k to zero (using a Lagrangian with constraint $\sum_k p_k = 1$), we get that the \mathbf{p} minimizing the loss fulfills $p_k = p_{\mathcal{T}}(k)$.

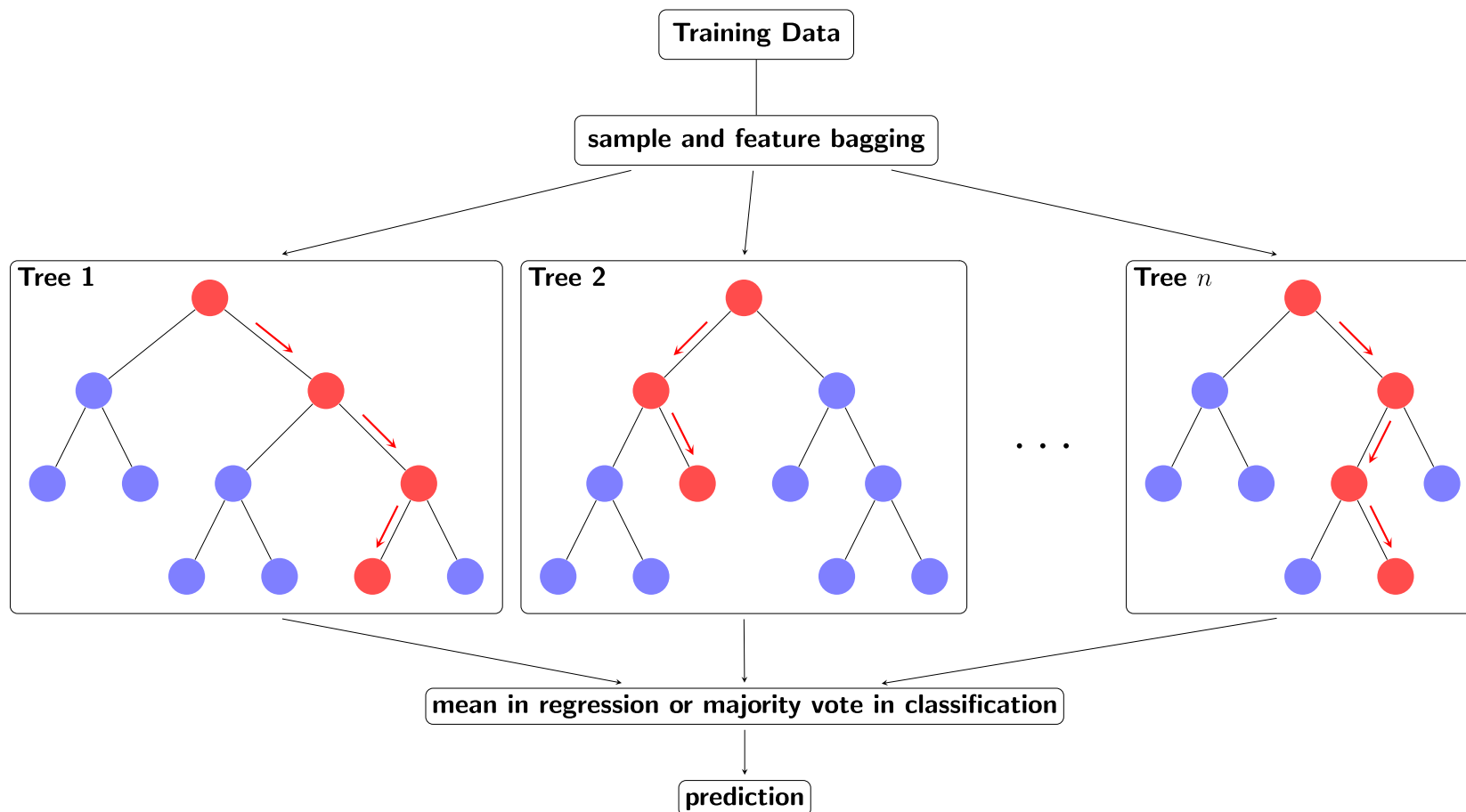
The value of the loss with respect to $\mathbf{p}_{\mathcal{T}}$ is then

$$\begin{aligned} L(\mathbf{p}_{\mathcal{T}}) &= \sum_{i \in I_{\mathcal{T}}} -\log p_{t_i} \\ &= - \sum_{\substack{k \\ p_{\mathcal{T}}(k) \neq 0}} n_{\mathcal{T}}(k) \log p_{\mathcal{T}}(k) \\ &= -|I_{\mathcal{T}}| \sum_{\substack{k \\ p_{\mathcal{T}}(k) \neq 0}} p_{\mathcal{T}}(k) \log p_{\mathcal{T}}(k) = |I_{\mathcal{T}}| \cdot H(\mathbf{p}_{\mathcal{T}}). \end{aligned}$$

Random Forests

Random Forests

Bagging of data combined with a random subset of features (sometimes called *feature bagging*).



<https://tex.stackexchange.com/questions/503883/illustrating-the-random-forest-algorithm-in-tikz>

Bagging

Every decision tree is trained using bagging (on a bootstrapped dataset).

Random Subset of Features

During each node split, only a random subset of features is considered when finding the best split. A fresh random subset is used for every node.

Extra Trees

The so-called extra trees are even more randomized, not finding the best possible feature value when choosing a split, but considering uniformly random samples from a feature's empirical range (minimum and maximum in the training data).

Demo

<https://cs.stanford.edu/~karpathy/svmjs/demo/demoforest.html>

After this lecture you should be able to

- Ensemble models with uncorrelated predictions
- Distill ensembles into smaller models
- Implement Decision Trees and Random Forests for classification and regression
- Explain how the splitting criterion depend on optimized loss function