Random Forests – A Long-Short Strategy for Japanese Stocks

Wu Zhuoyuan, Wu Tingan, Qiu Zixin

WHU Fintech Stu

December 7, 2022

Decision trees – learning rules from data

- How trees learn and apply decision rules
- Decision trees in practice
- Overfitting and regularization
- Hyperparameter tuning

Random forests – making trees more reliable

- Why ensemble models perform better
- Bootstrap aggregation
- Random forests

What is a decision tree

- Decision trees learn and sequentially apply a set of rules that split data points into subsets and then make one prediction for each subset. The predictions are based on the outcome values for the subset of training samples that result from the application of a given sequence of rules.
- **Classification trees** predict a probability estimated from the relative class frequencies or the value of the majority class directly.
- **Regression trees** compute prediction from the mean of the outcome values for the available data points.

What is a decision tree

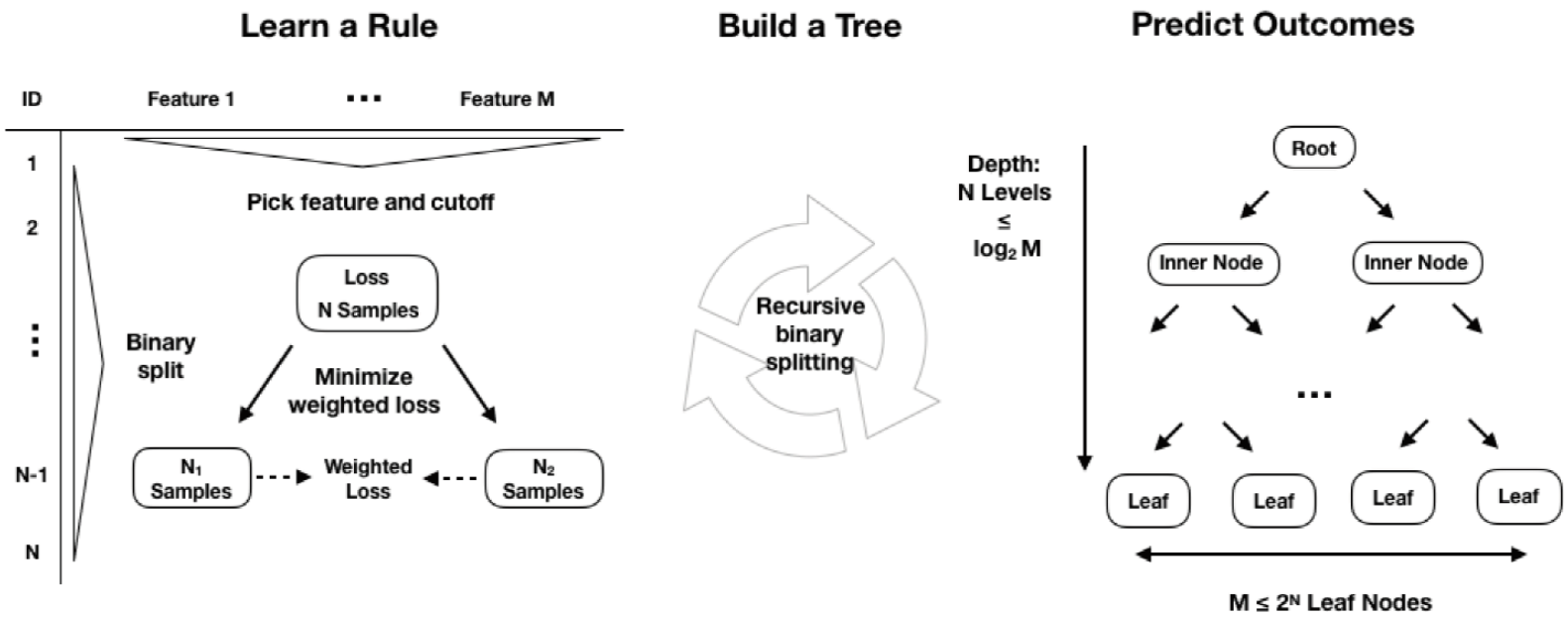
- Each of these rules relies on one particular feature and uses a threshold to split the samples into two groups, with values either below or above the threshold for this feature.
- A binary tree naturally represents the logic of the model: the root is the starting point for all samples, nodes represent the application of the decision rules, and the data moves along the edges as it is split into smaller subsets until it arrives at a leaf node, where the model makes a prediction.

Differences between Linear Models and Decision Trees

- For a linear model, the parameter values allow an interpretation of the impact of the input variables on the output and the model's prediction.
- For a decision tree, the various possible paths from the root to the leaves determine how the features and their values lead to specific decisions by the model.
- As a consequence, decision trees are capable of capturing interdependence among features that linear models cannot capture.

Differences between Linear Models and Decision Trees

- The following diagram highlights how the model learns a rule:



How decision trees grow?

- To build an entire tree during training, the learning algorithm repeats this process of dividing the feature space, that is, the set of possible values for the p input variables, X1, X2, . . . , Xp, into mutually-exclusive and collectively exhaustive regions, each represented by a leaf node.
- Tree-based learning takes a **top-down**, **greedy approach**, known as **recursive binary splitting**, to overcome this computational limitation.
- The splitting logic of regression and classification trees are different.

The data – monthly stock returns and features

- We will select a subset of the Quandl US equity dataset covering the period 2006-2017. And we will compute monthly returns and 25 (hopefully) predictive features for the 500 most-traded stocks based on the 5-year moving average of their dollar volume, yielding 56756 observations. The features include:
  - **Historical returns** for the past 1, 3, 6, and 12 months.
  - **Momentum indicators** that relate the most recent 1- or 3-month returns to those for longer horizons.
  - **Technical indicators** designed to capture volatility like the (normalized) average true range (NATR and ATR) and momentum like the **relative strength index** (RSI).
  - **Factor loadings** for the five Fama-French factors based on rolling OLS regressions.
  - **Categorical variables** for year and month, as well as sector.

Building a regression tree with time-series data

- Regression trees make predictions based on the mean outcome value for the training samples assigned to a given node, and typically rely on the mean-squared error to select optimal rules during recursive binary splitting.

$$D = \{\mathbf{x}_i, y_i\}_{i=1}^{n} \qquad \mathbf{x}_i = \{x_i^1, x_i^2, \cdots, x_i^d\}$$

$$R_1(j,s) = \{\mathbf{x}_i | x_i^j \leq s\} \quad R_2(j,s) = \{\mathbf{x}_i | x_i^j > s\}$$

$$\hat{c}_1 = \frac{1}{|R_1(j,s)|} \sum_{\mathbf{x}_i \in R_1(j,s)} y_i \qquad \hat{c}_2 = \frac{1}{|R_2(j,s)|} \sum_{\mathbf{x}_i \in R_2(j,s)} y_i$$

$$\min_{j,s} \left[\sum_{\mathbf{x}_i \in R_1(j,s)} (y_i - \hat{c}_1)^2 + \sum_{\mathbf{x}_i \in R_2(j,s)} (y_i - \hat{c}_2)^2 \right]$$

Building a regression tree with time-series data

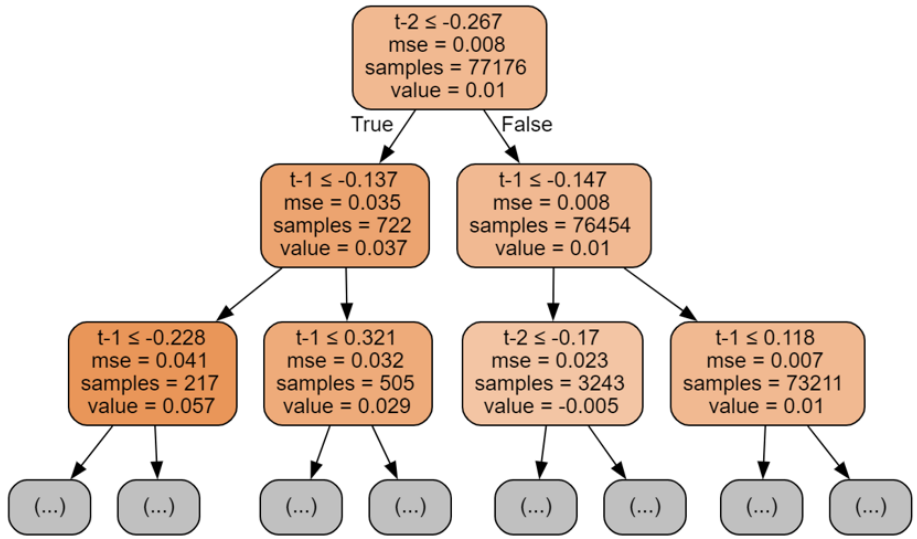
- We will only use 2 months of lagged returns to predict the following month, in the vein of an AR(2) model from the previous chapter.

OLS Regression Results

Dep. Variable:	y	R-squared:	0.001
Model:	OLS	Adj. R-squared:	0.001
Method:	Least Squares	F-statistic:	31.83
Date:	Sun, 04 Dec 2022	Prob (F-statistic):	1.53e-14
Time:	12:22:50	Log-Likelihood:	75823.
No. Observations:	77176	AIC:	-1.516e+05
Df Residuals:	77173	BIC:	-1.516e+05
Df Model:	2		
Covariance Type:	nonrobust		

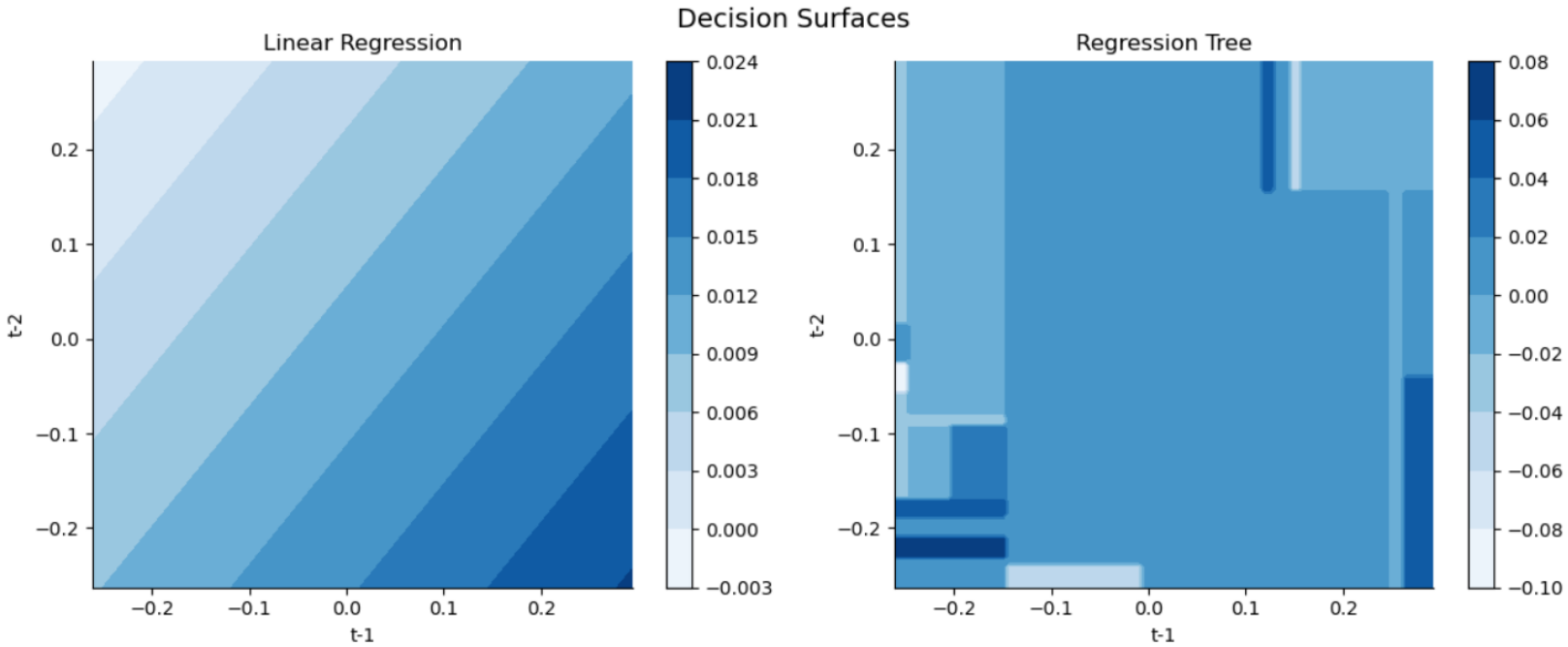
	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0100	0.000	30.232	0.000	0.009	0.011
t-1	0.0227	0.004	6.322	0.000	0.016	0.030
t-2	-0.0179	0.004	-5.003	0.000	-0.025	-0.011

Omnibus:	3351.598	Durbin-Watson:	1.995
Prob(Omnibus):	0.000	Jarque-Bera (JB):	10867.041
Skew:	0.102	Prob(JB):	0.00
Kurtosis:	4.827	Cond. No.	11.1



Building a regression tree with time-series data

- To further illustrate the difference, we can visualize the current return predictions as a function of the feature space, that is, as a function of the range of values for the lagged returns.



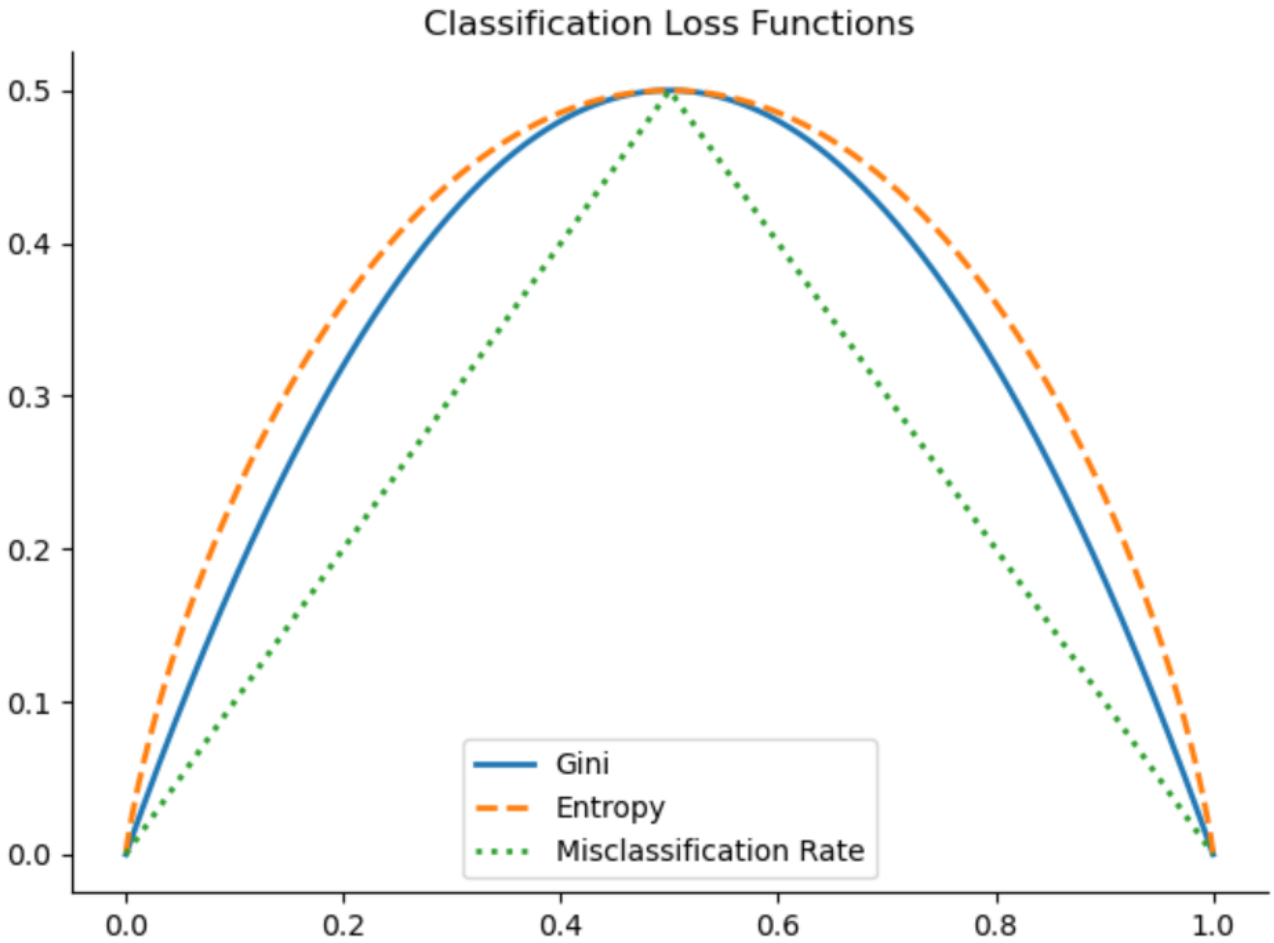
Building a classification tree

- When growing a classification tree, we also use recursive binary splitting, but instead of evaluating the quality of a decision rule using the reduction of the mean-squared error, we can use **Gini impurity** or **cross-entropy**, which are more sensitive to **node purity** that refers to the extent of the preponderance of a single class in a node.

$$\text{Gini impurity} = \sum_k p_{mk}(1 - p_{mk})$$

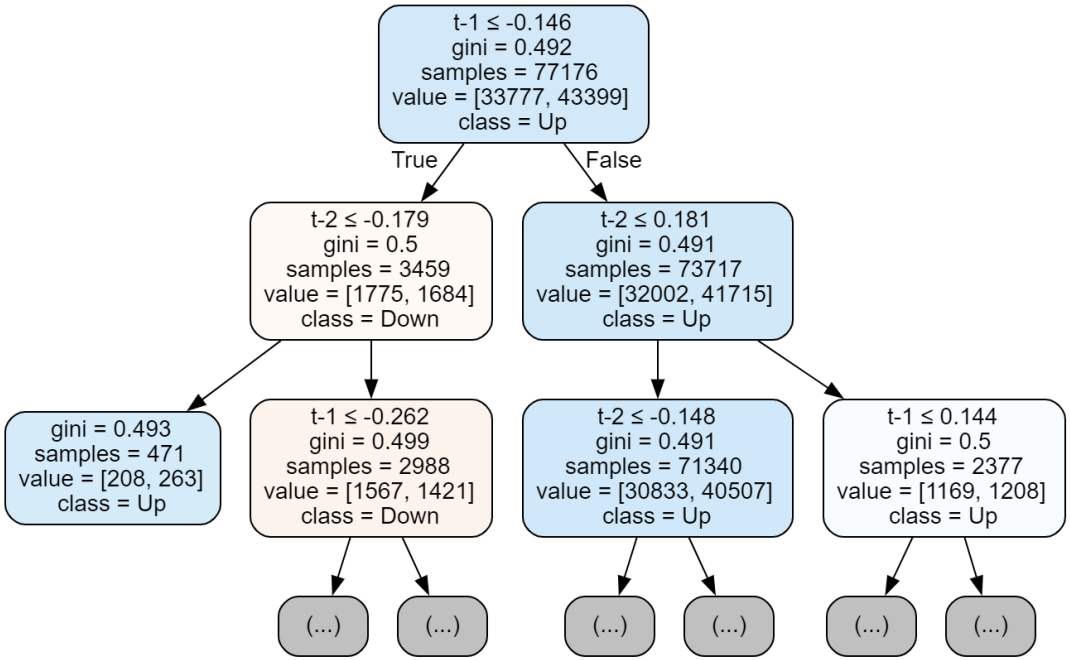
$$\text{cross entropy} = -\sum_k p_{mk}\, log(p_{mk})$$

Building a classification tree



Building a classification tree

- The following diagram shows how the model uses different features and indicates the split rules for both continuous and categorical (dummy) variables:



Overfitting

- Decision trees have a strong tendency to overfit, especially when a dataset has a large number of features relative to the number of samples.
- There are multiple ways to address the risk of overfitting, including:
  - Dimensionality reduction improves the feature-to-sample ratio by representing the existing features with fewer, more informative, and less noisy features.
  - Ensemble models, such as random forests, combine multiple trees while randomizing the tree construction
- Decision trees provide several **regularization hyperparameters** to limit the growth of a tree and the associated complexity. And **tree pruning** is an more powerful tool to reduce the complexity of a tree.

How to regularize a decision tree

Parameter	Description	Default	Options
`max_depth`	The maximum number of levels: split the nodes until `max_depth` has been reached. All leaves are pure or contain fewer samples than `min_samples_split`.	None	int
`max_features`	Number of features to consider for a split.	None	None: all features `int`: # features `float`: fraction `auto`, `sqrt`: sqrt(`n_features`) `log2`: log2(`n_features`)
`max_leaf_nodes`	Split nodes until creating this many leaves.	None	None: unlimited `int`
`min_impurity_decrease`	Split node if impurity decreases by at least this value.	0	`float`
`min_samples_leaf`	A split will only be considered if there are at least `min_samples_leaf` training samples in each of the left and right branches.	1	`int`; `float` (as a percent of N)

Decision tree pruning

- One approach to limit the number of leaf nodes is to avoid further splits unless they yield significant improvements in the objective metric. The downside of this strategy, however, is that sometimes, splits that result in small improvements enable more valuable splits later as the composition of the samples keeps changing.
- **Cost-complexity-pruning** generates a sequence of subtrees by adding a penalty for adding leaf nodes to the tree model and a regularization parameter. Applied to the large tree, an increasing penalty will automatically produce a sequence of subtrees. Cross-validation of the regularization parameter can be used to identify the optimal, pruned subtree.

Using GridsearchCV with a custom metric

- As highlighted in Chapter 6, scikit-learn provides a method to define ranges of values for multiple hyperparameters. It automates the process of cross-validating the various combinations of these parameter values to identify the optimal configuration.

Ensemble model

- An **ensemble** integrates the predictions of several **base estimators**, trained using one or more learning algorithms, to **reduce the generalization error** that these models produce on their own.
- For ensemble learning to achieve this goal, the **individual models** must be:
  - **Accurate**
  - **Independent**

Two groups of ensemble methods

- **Averaging methods** train several base estimators **independently** and then average their predictions.
  - Bagging
  - Random forest
- **Boosting methods** train base estimators **sequentially** with the specific goal of **reducing the bias** of the combined estimator. (*Chapter 12, Boosting Your Trading Strategy*)

Why we use bagging

- **How to reduce the variance**.
  - For a given a set of **independent** observations, each with a variance of σ^2, the standard error of the sample mean is given by $\sigma/\sqrt{n}$.
- **Bagging** refers to **the aggregation of bootstrap** samples.
  - **randomly** sample **with replacement**.
  - has **the same number of observations** as the original dataset
  - may contain **duplicates** due to replacement.

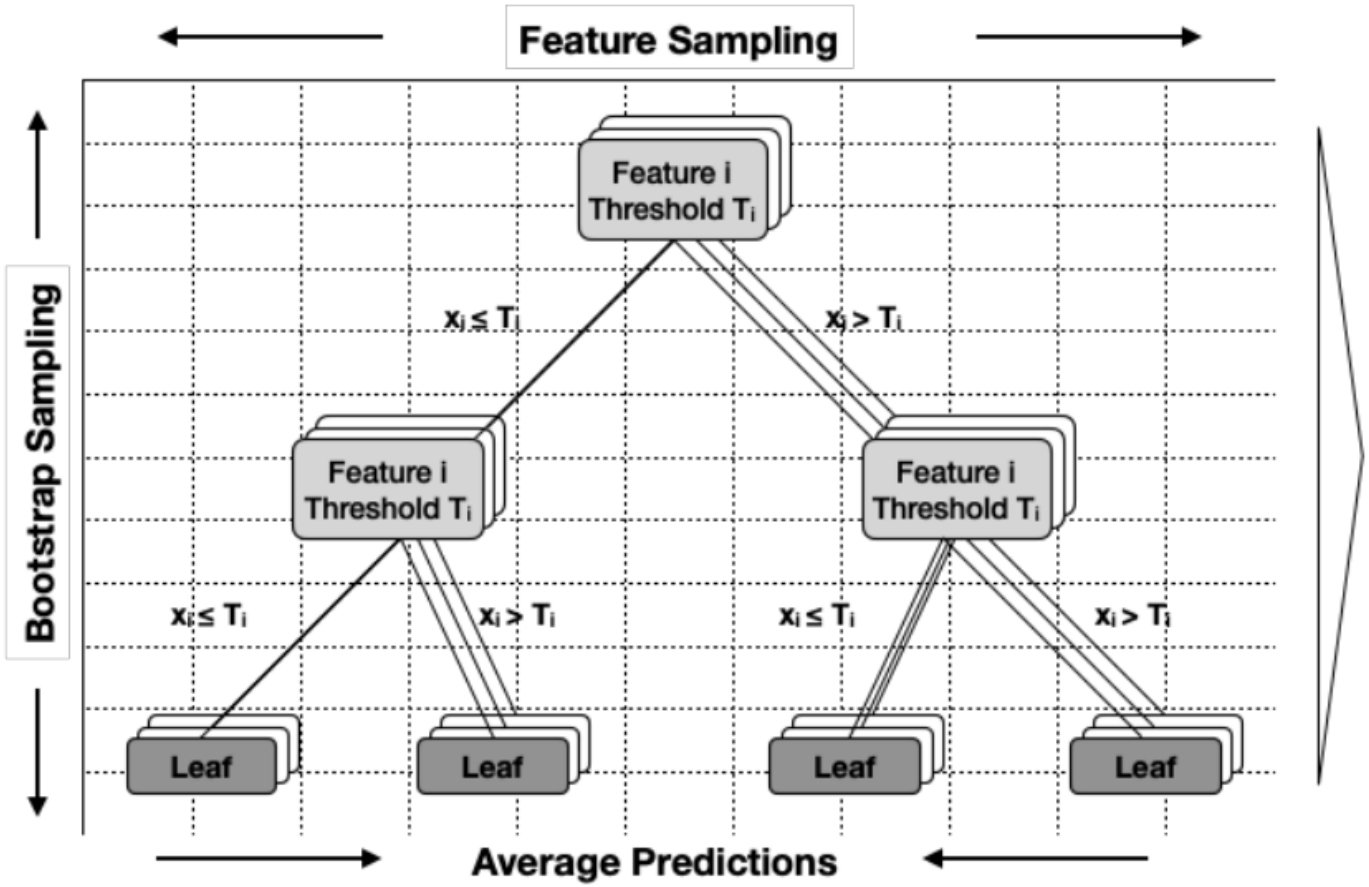
How to apply bagging to decision trees

- **create bootstrap samples** from our training data by repeatedly sampling with replacement.
- **train one decision tree** on each of these samples.
- create an ensemble prediction by **averaging** over the predictions of the different trees.

How to build a random forest

- **Random forests** would be:
  - To train each ensemble member on **bootstrapped** training data.
  - To **randomly** sample **from the features** used in the model **without replacement**.
- The **sample size** for the features differs:
  - For **classification**, the sample size is typically the **square root of the number of features**.
  - For **regression**, it can be anywhere **from one-third to all** features and should be selected **based on cross-validation**.

How to build a random forest



Feature importance for random forests

- For a given feature, the **importance score** is the total reduction in the objective function's value **due to splits on this feature** and is **averaged** over all trees.
- The score is measured in terms of **the mean-squared error** for regression trees and the **Gini impurity** or **entropy** for classification trees.

Out-of-bag testing

- **Out-of-bag (OOB) observation**
  - Consider that a bootstrap sample has the same size, n, as the original sample.
  - Hence, the probability of not entering a bootstrap sample at all is $(1 - 1/n)^n$, which converges (quickly) to 1/e, or roughly **one third**.

Out-of-bag testing

- **Predict** the response for an OOB sample **for each tree** built without this observation.
- **Average** the predicted responses (if regression is the goal) or **take a majority vote** for a single ensemble prediction **for each OOB sample**.
- **Average** the errors **of all OOB samples** to produce an unbiased generalization error.

Out-of-bag testing

	g_1	g_2	g_3	$\cdots$	g_T
$(\mathbf{x}_1, y_1)$	$\tilde{\mathcal{D}}_1$	$\star$	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_2, y_2)$	$\star$	$\star$	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_3, y_3)$	$\star$	$\tilde{\mathcal{D}}_2$	$\star$		$\tilde{\mathcal{D}}_T$
$\cdots$					
$(\mathbf{x}_N, y_N)$	$\tilde{\mathcal{D}}_1$	$\tilde{\mathcal{D}}_2$	$\star$		$\star$

- Take care to avoid a **lookahead bias** that would ensue if OOB observations could be selected **out-of-order**.

Pros and cons of random forests

- **Advantages**
  - **Perform on par with the best supervised learning algorithms**.
  - Provide a **reliable feature importance** estimate.
  - Offer efficient estimates of the test error **without incurring the cost** of repeated model training associated with cross-validation.
- **Disadvantages**
  - **Less interpretable**
  - **High computational costs**
  - **Slower**