MATH5836: Data and Machine Learning

Week 7: Decision Trees and Random Forests

Sarat Moka *UNSW, Sydney*

Key Topics

Book:

(A) Data Science and Machine Learning: Mathematical and Statistical Methods, by Kroese, Botev, Taimre, and Vaisman. Click here to download a pdf copy.

7.1 Decision Trees for Classification and Regression

Decision trees are versatile machine learning models used for both classification and regression tasks. They work by recursively partitioning the data space and fitting a simple prediction model within each partition.

Example 7.1.1

The left panel of Figure 7.1 shows a training set of 15 two-dimensional points (two features) falling into two classes (red and blue). How should the new feature vector (black point) be classified?

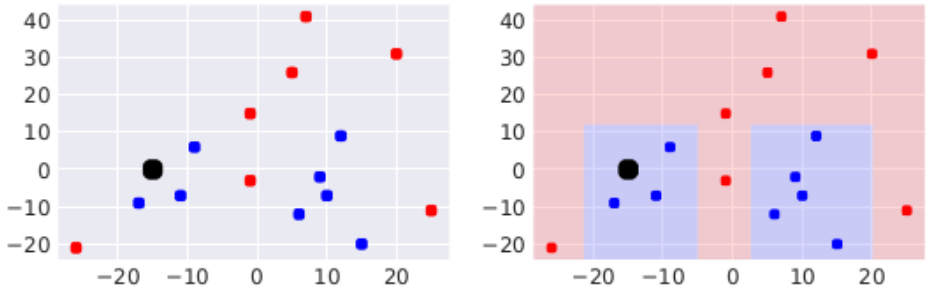

Figure 7.1: Left: training data and a new feature. Right: a partition of the feature space.

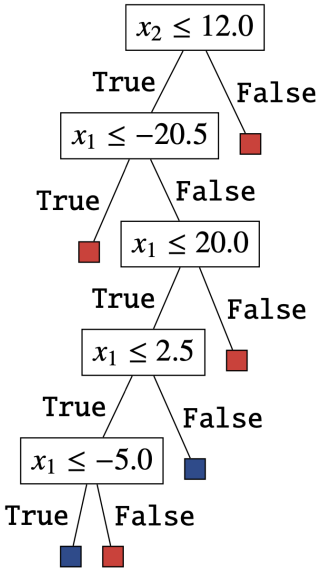

Figure 7.2: The decision- tree that corresponds to the partition in Figure 7.1.

Decision Tree Construction

Construction of a decision tree involves three components: splitting criterion, recursive partitioning, and stopping criteria. These are explained below.

1. **Splitting Criterion**:
   - A decision tree splits the data at each node based on a *criterion* that maximizes the separation of the data.
   - For classification, the common criteria are Gini impurity and information gain, while for regression, it is typically the sum of squared error.
   - *Optimal split:* Select the feature j^* and threshold t^* that yield the best split according to the chosen criterion. This involves finding the feature-threshold pair (j^*, t^*) that optimises the splitting criterion.
2. **Recursive Partitioning**:
   - Starting with the full training dataset at the root node, the dataset at each node is recursively partitioned by selecting the feature and threshold that maximizes the chosen splitting criterion.
   - For a feature j and a threshold t, split the dataset $D = \{(x_i, y_i) : i = 1, \ldots, n\}$ into D_{left} and D_{right}:
   $$D_{\text{left}} = \{(x_i, y_i) \in D \mid x_{i,j} \leq t\} \tag{7.1}$$
   $$D_{\text{right}} = \{(x_i, y_i) \in D \mid x_{i,j} > t\} \tag{7.2}$$
   where $x_{i,j}$ is the value of feature j for the data point x_i.
   - The partitioning process is repeated for each child node until a *stopping criterion* is met.
3. **Stopping Criteria**: The recursive partitioning stops when one of the following criteria is met:
   - Maximum tree depth is reached.
   - Minimum number of samples in a node is less than a specified threshold.
   - No further information gain or variance reduction is possible.

7.1.1 Splitting using Gini Index

- One commonly used criterion for classification is the Gini Index for selecting the optimal feature j^* and threshold t^*. The Gini Index measures the impurity of a node, and our goal is to find the feature-threshold pair (j^*, t^*) that minimizes the Gini Index after the split, leading to purer child nodes.
- To find the best split, we follow these steps:

1. **Compute Gini Index for the parent node**: Calculate the Gini Index (or, Gini impurity) for the parent node before the split. At any node with dataset D, the Gini impurity is defined as:
$$\text{Gini}(D) = 1 - \sum_{i=1}^{C} p_i^2 \tag{7.3}$$
where p_i is the proportion of samples belonging to class i in dataset D, and C is the number of classes.
2. **Evaluate all possible splits**: For each feature j and each possible threshold t (which can be a midpoint between any two consecutive values of the feature), split the data into two subsets:
$$D_{\text{left}} = \{(x_i, y_i) \mid x_i[j] \leq t\}$$
$$D_{\text{right}} = \{(x_i, y_i) \mid x_i[j] > t\}$$
3. **Compute Gini Index for the Child Nodes**: Calculate the Gini Index for each of the child nodes resulting from the split.
4. **Calculate Weighted Gini Index for the Split**: Compute the weighted Gini Index for the split as follows:
$$G_{\text{split}}(j, t) = \frac{N_{\text{left}}}{N} G(D_{\text{left}}) + \frac{N_{\text{right}}}{N} G(D_{\text{right}})$$
where N is the total number of instances in the parent node, N_{left} and N_{right} are the number of instances in the left and right child nodes respectively, and $G(D_{\text{left}})$ and $G(D_{\text{right}})$ are the Gini Indices for the left and right child nodes.
5. **Select the Best Split**: Find the feature-threshold pair (j^*, t^*) that minimizes the weighted Gini Index:
$$(j^*, t^*) = \arg\min_{(j,t)} G_{\text{split}}(j, t)$$

Remark 7.1.1

By selecting the feature j^* and threshold t^* that minimize the weighted Gini Index, we ensure that the resulting child nodes are as pure as possible, which leads to more effective splits and, ultimately, a more accurate decision tree.

Remark 7.1.2

For classification, instead of Gini Index, we can use *entropy impurity* defined by
$$\text{Entropy}(D) = -\sum_{i=1}^{C} p_i \log_2 p_i. \tag{7.4}$$

Example 7.1.2

Finding the best Gini-split on a 1D toy dataset: Consider the dataset

$$X = [1, 2, 3, 4, 5, 6]^\top, \quad y = [0, 0, 1, 1, 0, 1]^\top.$$

We evaluate splits at mid-points $1.5, 2.5, \ldots, 5.5$. For instance, at $t = 2.5$,

$$D_{\text{left}} = \{1, 2\}, \quad y_{\text{left}} = [0, 0], \quad G(D_{\text{left}}) = 0,$$

$$D_{\text{right}} = \{3, 4, 5, 6\}, \quad y_{\text{right}} = [1, 1, 0, 1], \quad G(D_{\text{right}}) = 1 - \left((3/4)^2 + (1/4)^2\right) = 0.375,$$

so the weighted Gini is

$$G_{\text{split}}(t = 2.5) = \frac{2}{6} \cdot 0 + \frac{4}{6} \cdot 0.375 = 0.25.$$

Repeat for $t = 1.5, 3.5, 4.5, 5.5$ and one finds the minimal $G_{\text{split}} = 0.25$ at $t = 2.5$. Thus the optimal split is

$$(j^* = 1,\ t^* = 2.5).$$

7.1.2 Splitting Criterion using Sum of Squared Errors (SSE)

- For regression tasks, a commonly used criterion is the Sum of Squared Errors (SSE) to select the optimal feature j^* and threshold t^*. The SSE measures the variance within a node, and our goal is to find the feature-threshold pair (j^*, t^*) that minimizes the SSE after the split, leading to child nodes with lower variance.
- To find the best split, we follow these steps:
  1. **Compute SSE for the parent node**: Calculate the SSE for the parent node before the split. At any node with dataset D, the SSE is defined as:

$$\text{SSE}(D) = \sum_{i=1}^{N} (y_i - \bar{y})^2 \tag{7.5}$$

     where y_i is the target value for instance i, $\bar{y}$ is the mean target value of the dataset D, and N is the number of instances in D.
  2. **Evaluate all possible splits**: For each feature j and each possible threshold t (which can be a midpoint between any two consecutive values of the feature), split the data into two subsets:

$$D_{\text{left}} = \{(x_i, y_i) \mid x_i[j] \leq t\}$$
$$D_{\text{right}} = \{(x_i, y_i) \mid x_i[j] > t\}$$

  3. **Compute SSE for the Child Nodes**: Calculate the SSE for each of the child nodes resulting from the split:

$$\text{SSE}(D_{\text{left}}) = \sum_{i \in D_{\text{left}}} (y_i - \bar{y}_{\text{left}})^2$$

$$\text{SSE}(D_{\text{right}}) = \sum_{i \in D_{\text{right}}} (y_i - \bar{y}_{\text{right}})^2$$

where $\bar{y}_{\text{left}}$ and $\bar{y}_{\text{right}}$ are the mean target values of the left and right child nodes, respectively.

4. **Calculate Weighted SSE for the Split**: Compute the weighted SSE for the split as follows:
$$\text{SSE}_{\text{split}}(j, t) = \frac{N_{\text{left}}}{N}\text{SSE}(D_{\text{left}}) + \frac{N_{\text{right}}}{N}\text{SSE}(D_{\text{right}})$$
where N is the total number of instances in the parent node, N_{left} and N_{right} are the number of instances in the left and right child nodes respectively, and $\text{SSE}(D_{\text{left}})$ and $\text{SSE}(D_{\text{right}})$ are the SSE for the left and right child nodes.

- By evaluating all possible splits and selecting the one that minimises the weighted SSE, the decision tree is able to create splits that reduce the overall variance in the target variable, leading to more accurate predictions.

Example 7.1.3

Regression split on a 1D toy dataset: Let

$$X = [1, 2, 3, 4], \quad y = [2, 4, 5, 8].$$

Consider a split at $t = 2.5$:

$$D_{\text{left}} = \{1, 2\},\ y_{\text{left}} = [2, 4],\ \bar{y}_{\text{left}} = 3,\ \text{SSE}(D_{\text{left}}) = (2-3)^2 + (4-3)^2 = 2.$$

$$D_{\text{right}} = \{3, 4\},\ y_{\text{right}} = [5, 8],\ \bar{y}_{\text{right}} = 6.5,\ \text{SSE}(D_{\text{right}}) = (5-6.5)^2 + (8-6.5)^2 = 4.5.$$

Weighted SSE:

$$\text{SSE}_{\text{split}}(t = 2.5) = \frac{2}{4} \cdot 2 + \frac{2}{4} \cdot 4.5 = 3.25.$$

Similarly evaluate at $t = 1.5, 3.5$ and find the minimal weighted SSE at the best threshold.

7.1.3 Prediction

Construction of a decision tree as described above is equivalent to training a model f_θ. Once construction is complete, we can use the decision tree for prediction on a new data point.

- **Classification**: For a given input x, traverse the tree from the root to a leaf node. The predicted class $\hat{y}$ is the majority class in the leaf node:
$$\hat{y} = \arg\max_{c \in C} \left(\frac{1}{|D_{\text{leaf}}|} \sum_{(x_i, y_i) \in D_{\text{leaf}}} \mathbb{I}(y_i = c) \right) \tag{7.6}$$
where D_{leaf} is the dataset in the leaf node and $\mathbb{I}$ is the indicator function.

- **Regression**: For a given input x, traverse the tree from the root to a leaf node. The predicted value $\hat{y}$ is the mean of the target values in the leaf node:

$$\hat{y} = \frac{1}{|D_{\text{leaf}}|} \sum_{(x_i, y_i) \in D_{\text{leaf}}} y_i \tag{7.7}$$

Example 7.1.4

Prediction walk-through: Suppose our trained tree (for classification) is

$$\text{if } x_1 \leq 3.0: \ \hat{y} = 0, \quad \text{else if } x_2 \leq 1.5: \ \hat{y} = 1, \quad \text{else}: \ \hat{y} = 0.$$

Classify

$$x^{(a)} = (2.5,\, 2.0), \quad x^{(b)} = (4.0,\, 1.0), \quad x^{(c)} = (4.0,\, 2.0).$$

- For $x^{(a)}$: $x_1 = 2.5 \leq 3.0 \implies \hat{y}^{(a)} = 0$.
- For $x^{(b)}$: $x_1 = 4.0 > 3.0$, then $x_2 = 1.0 \leq 1.5 \implies \hat{y}^{(b)} = 1$.
- For $x^{(c)}$: $x_1 = 4.0 > 3.0$, then $x_2 = 2.0 > 1.5 \implies \hat{y}^{(c)} = 0$.

7.1.4 Evaluation

- **Performance Metrics**: Evaluate the performance of the decision tree using appropriate metrics. For classification, common metrics include accuracy, precision, recall, and the F1 score. For regression, common metrics include Mean Squared Error (MSE) and Mean Absolute Error (MAE).

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(\hat{y}_i = y_i) \tag{7.8}$$

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \tag{7.9}$$

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i| \tag{7.10}$$

- **Visualization**: Visualize the decision tree to understand the decision rules and the structure of the model. Tools such as tree diagrams can be used to illustrate the splits and the criteria used at each node.

7.2 Model Complexity, Overfitting and the Bias–Variance Trade-Off

Decision trees can suffer from high variance if grown too deep, or from high bias if pruned too aggressively. A plot of training error vs. validation error as a function of $|T_\ell|$ or $D_{\max}$ can help us in

understanding bias vs variance trade-off: A large gap between these errors indicates high variance; both high indicates high bias. Recall that in regression,

$$\mathbb{E}_{X,Y}\left[(\hat{f}(x)-y)^2\right] = \underbrace{\left(\mathbb{E}[\hat{f}(x)]-f(x)\right)^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}\left[(\hat{f}(x)-\mathbb{E}[\hat{f}(x)])^2\right]}_{\text{Variance}} + \sigma_\epsilon^2.$$

- Deeper trees lower bias but elevate variance term.
- Pruning trades variance for bias, balancing overall MSE.

There are two broad approaches used for reducing overfitting.

- **Pre-pruning (early stopping)**: We have discussed in Section 7.1 that one way to halt splitting is when
  - tree depth reaches a maximum $D_{\max}$, or
  - node has fewer than $m_{\min}$ samples, or
  - impurity reduction falls below a threshold.

  Remark 7.2.1

  Depth, Leaves, and Minimum Samples:

  - Increasing *maximum depth* $D_{\max}$ or decreasing *min-samples-per-leaf* $m_{\min}$ reduces bias but increases variance.
  - The number of leaves $|T_\ell|$ controls model capacity; very large $|T_\ell| \to$ overfitting.

- **Post-pruning (cost-complexity pruning)**: First grow a full tree T_0, then select a subtree $T \subseteq T_0$ to trade off fit vs. size. Define the cost-complexity objective

  $$R_\alpha(T) \;=\; R(T) \;+\; \alpha\,|T_\ell|,$$

  where
  - $R(T)$ is the empirical risk (misclassification rate or SSE) of T,
  - $|T_\ell|$ is the number of leaves in T,
  - $\alpha \geq 0$ is the complexity parameter.

  For each α one finds the subtree T_α minimising $R_\alpha(T)$. Use K-fold cross-validation over α to pick the optimal complexity.

Exercise 7.2.1

Bias–Variance Trade-Off in a 1-D Regression Tree: We illustrate how tree depth trades bias vs. variance on a simple dataset.

- Generate $n = 50$ points

$$x_i \overset{iid}{\sim} \texttt{Unif}(0,1), \quad y_i = x_i^2 + \varepsilon_i, \quad \varepsilon_i \sim N(0, 0.05^2).$$

- Split into a training set of 30 points and a validation set of 20 points.
- Fit $\hat{f}_d$ = `DecisionTreeRegressor` from `sklearn` with maximum depth $d \in \{1, 2, 3, 5, 10\}$.
- Compute and plot

$$\text{MSE}_{\text{train}}(d) = \frac{1}{30} \sum_{i \in \text{train}} \big(\hat{f}_d(x_i) - y_i\big)^2, \quad \text{MSE}_{\text{val}}(d) = \frac{1}{20} \sum_{i \in \text{val}} \big(\hat{f}_d(x_i) - y_i\big)^2.$$

- Experiment with different noise levels σ_ε and sample sizes n.
- Observe how the optimal depth shifts as variance increases.

7.3 Random Forests

Random Forests build on the single-tree learner by training a whole "forest" of decision trees and then combining their outputs:

- For classification, each tree votes for a class and the forest predicts the majority class.
- For regression, the forest's prediction is the average of all tree-predicted values.

Each tree in the forest is grown on a different bootstrap sample of the data, and (optionally) at each split only a random subset of features is considered. These randomisations help decorrelate the trees and typically reduce over-fitting compared to one large tree.

Remark 7.3.1

Random forests can be viewed as an *ensemble* method, which we will cover in full details next week. Consider this a teaser: next week we'll study bagging, boosting, and see exactly how and why combining many trees often yields much better performance and stability than any single tree alone.

Construction of Random Forests

The construction of a Random Forest involves the following steps:

1. **Bootstrap Sampling**: Given a training dataset D with N instances, create B bootstrap samples D_b (where $b = 1, 2, \ldots, B$). Each bootstrap sample is generated by randomly sampling N instances from D with replacement.
2. **Training Decision Trees**: For each bootstrap sample D_b, train an unpruned decision tree T_b. During the training of each tree, at each node, a random subset of features is selected from the total set of features. The best feature-threshold pair is chosen only from this subset to perform the split.
3. **Aggregating Predictions**: For a classification problem, the final prediction of the Random Forest is obtained by majority voting among the B decision trees. For a regression problem, the final prediction is obtained by averaging the predictions of the B decision trees.

Mathematical Formulation

Let $D = \{(x_i, y_i)\}_{i=1}^{N}$ be the training dataset, where x_i represents the feature vector and y_i represents the target variable. The steps can be mathematically formulated as follows:

1. **Bootstrap Sampling**:
$$D_b = \{(x_i, y_i)\}_{i=1}^{N}, \quad \text{where } D_b \text{ is sampled with replacement from } D$$
2. **Training Decision Trees**: For each bootstrap sample D_b, a decision tree T_b is trained. At each node, select a random subset of features $F \subseteq \{1, 2, \ldots, p\}$ (where p is the total number of features), and determine the best split:
$$(j^*, t^*) = \underset{(j,t):j \in F}{\arg\min} \ \text{Impurity}(D_{\text{left}}, D_{\text{right}})$$
where Impurity could be the Gini Index, Entropy, or SSE, depending on the problem type.
3. **Aggregating Predictions**: For a classification problem, the final prediction $\hat{y}$ for a new instance x is:
$$\hat{y} = \text{mode}\{T_b(x)\}_{b=1}^{B},$$
where $T_b(x)$ is the prediction by the b-th tree. For a regression problem, the final prediction $\hat{y}$ is:
$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} T_b(x)$$

Out-of-Bag Error Estimation

An important property of Random Forests is the ability to estimate the generalization error using Out-of-Bag (OOB) samples. For each bootstrap sample D_b, approximately one-third of the original instances are not included (out-of-bag samples).

Mathematically, for a dataset $D = \{(x_i, y_i)\}_{i=1}^N$:

1. Generate B bootstrap samples $D_1, D_2, \ldots, D_B$ by sampling N instances with replacement from D.
2. For each bootstrap sample D_b, identify the out-of-bag samples D_b^{OOB} which are the data points not included in D_b.
3. Construct the b-th tree T_b on the bootstrap sample D_b.

The OOB error estimation involves the following steps:

1. For each instance (x_i, y_i) in the original dataset, collect the predictions from all trees that did not use (x_i, y_i) in their bootstrap sample.
2. Aggregate these predictions (majority vote for classification and mean for regression) to obtain the OOB prediction $\hat{y}_{\text{OOB},i}$.
3. Compute the OOB error as follows:

$$\text{OOB Error} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(\hat{y}_{\text{OOB},i} \neq y_i) \quad \text{(for classification)}$$

$$\text{OOB Error} = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_{\text{OOB},i})^2 \quad \text{(for regression)}$$

By aggregating the predictions from multiple decision trees and leveraging bootstrap sampling, Random Forests achieve better generalization performance and robustness compared to individual decision trees.

7.4 Random Forests vs Neural Networks

In practice, the choice between a random forest and a neural network depends on the nature of the problem and the size and structure of the available data. Below we summarize situations where one model family tends to outperform the other.

Random forests excel when:

1. **Ease of training and tuning of hyperparameters.** Random forests require only a few intuitive hyperparameters (e.g., number of trees, maximum tree depth, minimum samples per leaf). Default settings often yield strong performance, making model development fast and straightforward.

2. **Higher interpretability.** Individual decision trees are inherently transparent, and random forests provide reliable feature-importance scores. This makes it easier to explain model behavior and extract actionable insights.

3. **Robustness to overfitting.** By averaging the predictions of many decorrelated trees, random forests reduce variance and generalize well even when individual trees overfit the training data.

4. **Ability to handle missing data.** Many implementations support surrogate splits or built-in imputation strategies, so you can train directly on datasets with missing values without extensive preprocessing.

Neural networks excel when:

1. **Handling unstructured data.** Convolutional and recurrent architectures can ingest raw images, audio waveforms, text sequences, and graphs—automatically learning hierarchical feature representations from pixels, samples, or tokens.

2. **Modeling extremely large datasets with complex patterns.** Deep networks can approximate highly nonlinear functions and benefit from millions of training examples, capturing intricate relationships that shallow models may miss.

3. **Transfer learning.** Pretrained networks (e.g., on ImageNet or large language corpora) can be fine-tuned on new tasks with comparatively little data, drastically reducing training time and improving performance in domains with limited labeled samples.