COMSS30035, Machine learning: Regresssion and Classification Trees

James Cussens

School of Computer Science
University of Bristol

5th October 2025

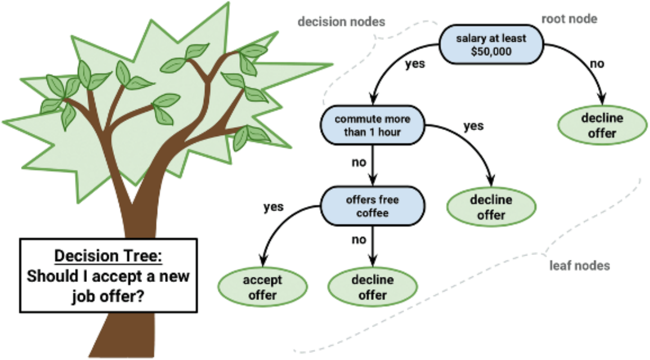
Acknowledgement

- These slides are adapted from ones originally created by Edwin Simpson.

Decision Trees

A Decision Tree is a tree-structured predictive model where:

- Each internal node represents a decision test on a feature (e.g., Is salary $> 50k$?).
- Each branch represents the outcome of the test (e.g., yes/no).
- Each leaf node represents a predicted class label (for classification) or a numerical value (for regression).



Decision Trees as Partitioning Input Space

- One model is responsible for assigning a decision for each region of input space;
- The correct model for an input $\boldsymbol{x}$ is chosen by traversing the binary decision tree, following the path from the top to a leaf.
- Leaf node is responsible for assigning a decision, such as a:
  - Class label;
  - Probability distribution over class labels;
  - Scalar value (for regression tasks).

Learning the Tree Structure

- Which input variable to use at each node?
- Which attributes will be used to split at each node? How to determine the threshold?
- Classification and Regression Trees (CART): one of many possible learning algorithms
- Objective: greedily minimise the error
  - Regression: sum-of-squares
  - Classification: cross-entropy as used in neural networks or Gini impurity

Learning the Tree Structure

- Which input variable to use at each node?
- Which attributes will be used to split at each node? How to determine the threshold?
- Classification and Regression Trees (CART): one of many possible learning algorithms
- Objective: greedily minimise the error
  - Regression: sum-of-squares
  - Classification: cross-entropy as used in neural networks or Gini impurity

Learning the Tree Structure

- Which input variable to use at each node?
- Which attributes will be used to split at each node? How to determine the threshold?
- Classification and Regression Trees (CART): one of many possible learning algorithms
- Objective: greedily minimise the error
  - Regression: sum-of-squares
  - Classification: cross-entropy as used in neural networks or Gini impurity

Learning the Tree Structure

- Which input variable to use at each node?
- Which attributes will be used to split at each node? How to determine the threshold?
- Classification and Regression Trees (CART): one of many possible learning algorithms
- Objective: greedily minimise the error
  - Regression: sum-of-squares
  - Classification: cross-entropy as used in neural networks or Gini impurity

Learning the Tree Structure

- Number of possible solutions grows combinatorially with the number of input variables
- Greedy algorithm: add nodes one-at-a-time, choosing the best split at each point

1. Start from the root node
2. Run *exhaustive search* over each possible variable and threshold for a new node. For each variable and threshold:
    - Compute average of the target variable for each leaf of the proposed node
    - Compute the error if we stop adding nodes here
3. Choose the variable & threshold that minimise the error
4. Add a new node for the chosen variable and threshold.
5. Repeat step 2 until there are only n data points associated with each leaf node.
6. Prune back the tree to remove branches that do not reduce error by more than a small tolerance value, ϵ.

Learning the Tree Structure

- Number of possible solutions grows combinatorially with the number of input variables
- Greedy algorithm: add nodes one-at-a-time, choosing the best split at each point
  1. Start from the root node
  2. Run *exhaustive search* over each possible variable and threshold for a new node. For each variable and threshold:
     - Compute average of the target variable for each leaf of the proposed node
     - Compute the error if we stop adding nodes here
  3. Choose the variable & threshold that minimise the error
  4. Add a new node for the chosen variable and threshold.
  5. Repeat step 2 until there are only n data points associated with each leaf node.
  6. Prune back the tree to remove branches that do not reduce error by more than a small tolerance value, ϵ.

Learning the Tree Structure

- Number of possible solutions grows combinatorially with the number of input variables
- Greedy algorithm: add nodes one-at-a-time, choosing the best split at each point
  1. Start from the root node
  2. Run *exhaustive search* over each possible variable and threshold for a new node. For each variable and threshold:
     - Compute average of the target variable for each leaf of the proposed node
     - Compute the error if we stop adding nodes here
  3. Choose the variable & threshold that minimise the error
  4. Add a new node for the chosen variable and threshold.
  5. Repeat step 2 until there are only n data points associated with each leaf node.
  6. Prune back the tree to remove branches that do not reduce error by more than a small tolerance value, ϵ.

Learning the Tree Structure

- Number of possible solutions grows combinatorially with the number of input variables
- Greedy algorithm: add nodes one-at-a-time, choosing the best split at each point
  1. Start from the root node
  2. Run *exhaustive search* over each possible variable and threshold for a new node. For each variable and threshold:
     - Compute average of the target variable for each leaf of the proposed node
     - Compute the error if we stop adding nodes here
  3. Choose the variable & threshold that minimise the error
  4. Add a new node for the chosen variable and threshold.
  5. Repeat step 2 until there are only n data points associated with each leaf node.
  6. Prune back the tree to remove branches that do not reduce error by more than a small tolerance value, ϵ.

Learning the Tree Structure

- Number of possible solutions grows combinatorially with the number of input variables
- Greedy algorithm: add nodes one-at-a-time, choosing the best split at each point
  1. Start from the root node
  2. Run *exhaustive search* over each possible variable and threshold for a new node. For each variable and threshold:
     - Compute average of the target variable for each leaf of the proposed node
     - Compute the error if we stop adding nodes here
  3. Choose the variable & threshold that minimise the error
  4. Add a new node for the chosen variable and threshold.
  5. Repeat step 2 until there are only n data points associated with each leaf node.
  6. Prune back the tree to remove branches that do not reduce error by more than a small tolerance value, ϵ.

Learning the Tree Structure

- Number of possible solutions grows combinatorially with the number of input variables
- Greedy algorithm: add nodes one-at-a-time, choosing the best split at each point
  1. Start from the root node
  2. Run *exhaustive search* over each possible variable and threshold for a new node. For each variable and threshold:
     - Compute average of the target variable for each leaf of the proposed node
     - Compute the error if we stop adding nodes here
  3. Choose the variable & threshold that minimise the error
  4. Add a new node for the chosen variable and threshold.
  5. Repeat step 2 until there are only n data points associated with each leaf node.
  6. Prune back the tree to remove branches that do not reduce error by more than a small tolerance value, ϵ.

Learning the Tree Structure

- Number of possible solutions grows combinatorially with the number of input variables
- Greedy algorithm: add nodes one-at-a-time, choosing the best split at each point
  1. Start from the root node
  2. Run *exhaustive search* over each possible variable and threshold for a new node. For each variable and threshold:
     - Compute average of the target variable for each leaf of the proposed node
     - Compute the error if we stop adding nodes here
  3. Choose the variable & threshold that minimise the error
  4. Add a new node for the chosen variable and threshold.
  5. Repeat step 2 until there are only n data points associated with each leaf node.
  6. Prune back the tree to remove branches that do not reduce error by more than a small tolerance value, ϵ.

Attribute Selection in CART

- For Classification Trees:
  Gini Impurity (for Classification):

$$Gini(D) = 1 - \sum_{i=1}^{C} p_i^2 \tag{1}$$

  Information Gain (optional):

$$IG(D, A) = Entropy(D) - \sum_{v \in Splits} \frac{|D_v|}{|D|} Entropy(D_v) \tag{2}$$

- For Regression Trees:

$$MSE(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} (y_i - \bar{y})^2 \tag{3}$$

Example of Information Gain Calculation

A	B	C	Class
yes	yes	yes	3
yes	yes	no	2
yes	no	yes	3
yes	no	no	1
no	yes	yes	3
no	yes	no	2
no	no	yes	3
no	no	no	1

Step 1: calculate start entropy:

$$E_{start}(S) = -\frac{2}{8}log_2(\frac{2}{8}) - \frac{2}{8}log_2(\frac{2}{8}) - \frac{4}{8}log_2(\frac{4}{8}) = 0.5 + 0.5 + 0.5 = 1.5\ bits$$

Step 2: entropy of the training data if S were divided into S1 and S2: i.e. for attribute A:

$$E_A(S) = \frac{4}{8}(-\frac{1}{4}log_2(\frac{1}{4}) - \frac{1}{4}log_2(\frac{1}{4}) - \frac{2}{4}log_2(\frac{2}{4})) + \frac{4}{8}$$
$$(-\frac{1}{4}log_2(\frac{1}{4}) - \frac{1}{4}log_2(\frac{1}{4}) - \frac{2}{4}log_2(\frac{2}{4})) = 1.5\ bits$$

Repeat for all attributes

Step 3: calculate information gain for all possible divisions: i.e. for attribute A:

$$Gain(A) = E(S) - E_A(S) = 1.5\ bits - 1.5\ bits = 0\ bits$$

Example of Information Gain Calculation

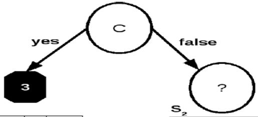
A	B	C	Class
yes	yes	yes	3
yes	yes	no	2
yes	no	yes	3
yes	no	no	1
no	yes	yes	3
no	yes	no	2
no	no	yes	3
no	no	no	1

The same calculation is performed also for the remaining attributes resulting in information gains of:

$$Gain(A) = E(S) - E_A(S) = 1.5\ bits - 1.5\ bits = 0\ bits$$

$$Gain(B) = E(S) - E_B(S) = 1.5\ bits - 1\ bit = 0.5\ bits$$

$$Gain(C) = E(S) - E_C(S) = 1.5\ bits - 0.5\ bits = 1\ bit$$



S_1

A	B	C	Class
yes	yes	yes	3
yes	no	yes	3
no	yes	yes	3
no	no	yes	3

S_2

A	B	C	Class
yes	yes	no	2
yes	no	no	1
no	yes	no	2
no	no	no	1

Stopping Criteria in Decision Trees

- Growing a tree without limits often leads to **overfitting**.
- We need **stopping criteria** to decide when to stop splitting:
  - Maximum tree depth reached;
  - Minimum number of samples in a leaf node;
  - Minimum number of samples required to split a node;
  - Impurity (e.g. Gini, entropy, MSE) below a threshold;
  - No further information gain from any split.
- These rules help balance **model complexity** and **generalization**.

Pruning

- Balance residual training-set error against model complexity
- Start with a tree T_0
- Consider pruning each node in T_0 by combining the branches to obtain tree T
- Compute a criterion $C(T) = \sum_{\tau=1}^{|T|} e_\tau(T) + \lambda|T|$
- If $C(T) \leq C(T_0)$ keep the pruned tree, else reinstate the pruned node.

Pruning

- Balance residual training-set error against model complexity
- Start with a tree T_0
- Consider pruning each node in T_0 by combining the branches to obtain tree T
- Compute a criterion $C(T) = \sum_{\tau=1}^{|T|} e_\tau(T) + \lambda|T|$
- If $C(T) \leq C(T_0)$ keep the pruned tree, else reinstate the pruned node.

Pruning

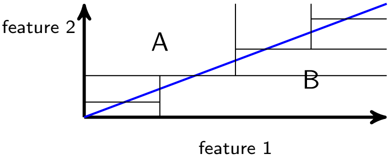
- Balance residual training-set error against model complexity
- Start with a tree T_0
- Consider pruning each node in T_0 by combining the branches to obtain tree T
- Compute a criterion $C(T) = \sum_{\tau=1}^{|T|} e_\tau(T) + \lambda|T|$
- If $C(T) \leq C(T_0)$ keep the pruned tree, else reinstate the pruned node.

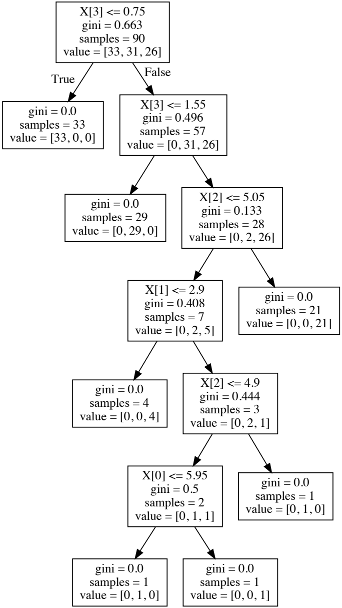
Pruning

- Balance residual training-set error against model complexity
- Start with a tree T_0
- Consider pruning each node in T_0 by combining the branches to obtain tree T
- Compute a criterion $C(T) = \sum_{\tau=1}^{|T|} e_\tau(T) + \lambda|T|$
- If $C(T) \leq C(T_0)$ keep the pruned tree, else reinstate the pruned node.

Interpretability

- The sequence of decisions is often easier to interpret than other methods (think of neural networks);
- However, sometimes small changes to the dataset cause big changes to the tree;
- If the optimal decision boundary is not aligned with the axes of an input variable, we need a lot of splits.





Pros and Cons of CART

Pros

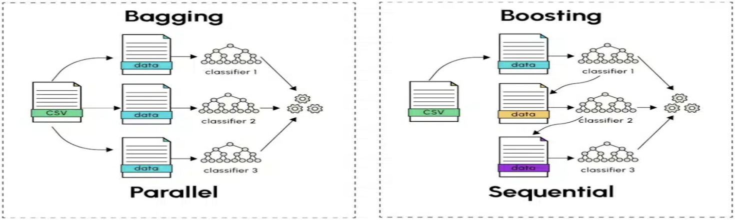
- Easy to interpret and visualize
- Handles both classification and regression
- Non-parametric (no strong assumptions)
- Works with numerical and categorical features
- Captures nonlinear relationships
- Robust to scaling of features

Cons

- Prone to overfitting (needs pruning)
- High variance (unstable to small data changes)
- Only axis-aligned splits
- Biased toward features with many split points
- Single tree is often less accurate than ensembles
- Regression outputs are piecewise constant

Ensemble Methods

- **Bagging (Bootstrap Aggregating)**
  Train multiple trees on bootstrapped samples; average or vote. *(Reduces variance).*
- **Random Forests**
  Extension of Bagging with random feature subsets at splits. *(Stable, accurate).*
- **Boosting (AdaBoost, Gradient Boosting, XGBoost)**
  Sequentially build trees, focusing on correcting errors (reduces bias). *(High accuracy).*



https://datascientest.com/en/bagging-vs-boosting

Reading

There is typo on p. 666 of Bishop where a minus sign has gone missing. Equation (14.32) should be:

$$Q_\tau(T) = -\sum_{k=1}^{K} p_{\tau k} \ln p_{\tau k}$$

- Bishop §14.4.
- Murphy §18.1

Problems and quizzes

- Bishop Exercise 14.10
- Bishop Exercise 14.11 (don't forget about the typo!)
- Quizzes:
  - Week 3: Trees