Decision Trees

Impurity Measures, Overfitting, Feature Importance, and Decision Boundaries

1. Introduction

Decision trees are one of the most intuitive and interpretable machine-learning algorithms, widely used for both classification and regression tasks. A decision tree models the data using a sequence of human-readable rules, such as "if feature $X \geq 12.7$ then go left, else go right." These simple logical rules break the input space into smaller and smaller regions until a prediction can be made.

The popularity of decision trees stems from several factors:

- **Interpretability:** Unlike neural networks or SVMs, trees are transparent. You can visualize them, print their rules, and explain their decisions.
- **Flexibility:** They can handle numerical features, categorical features, missing values, and nonlinear relationships.
- **Low preprocessing requirements:** No need for feature scaling or normalization.
- **Foundation for powerful ensemble methods:** Random Forests, Gradient Boosted Trees, and XGBoost all rely on decision trees as their base models.

Despite their simplicity, decision trees involve several subtle ideas: impurity measures, greedy splitting, overfitting due to depth, feature importance scoring, and the shape of decision boundaries. This tutorial explores all of these topics in a structured and intuitive way.

2. Impurity Measures: Gini, Entropy, and Misclassification Error

At every node, a decision tree must choose a feature and a threshold that best separates the data. This "best split" is chosen using an **impurity measure**, which quantifies how mixed the classes are in a node. A node with all points belonging to a single class is considered "pure," while a node containing a 50–50 mix of two classes is maximally impure.

There are three impurity measures commonly used:

2.1 Gini Impurity

Gini impurity is the default splitting criterion in many libraries such as scikit-learn. For a node with class probabilities p1,p2,...p_1, p_2, ...p1,p2,..., Gini is computed as:

$$\text{Gini} = 1 - \sum p_k^2$$

Interpretation:
Gini represents the probability that a randomly selected instance would be misclassified if it were labeled according to the node's class proportions.

2.2 Entropy (Information Gain)

Entropy measures the disorder or unpredictability:

$$H = -\sum pklog(pk)$$

A split is chosen to maximize **information gain**, which is:

$$Information\ Gain = \ Hparent - (wLHL + wRHR)$$

Entropy tends to produce slightly more balanced splits than Gini and is more sensitive to small changes in class distribution.

2.3 Misclassification Error

Defined as: $1 - max(pk)$

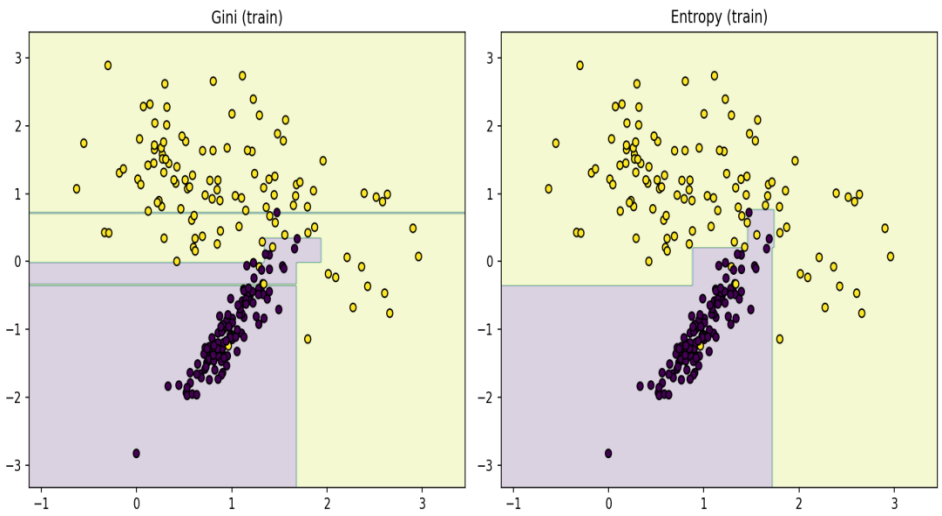
This is the simplest impurity measure but the least sensitive. It is not commonly used for splitting, but it is useful for pruning or validation.

2.4 Comparing Gini and Entropy

In practice:

- Both produce very similar trees.
- Gini is faster to compute (no logarithms).
- Entropy may create deeper trees with more balanced splits.
- Differences in accuracy are usually negligible.

```
# Plot trees and decision boundaries
fig, axs = plt.subplots(1, 2, figsize=(12,5))
plot_decision_boundary(clf_gini, X_train, y_train, ax=axs[0], title='Gini (train)')
plot_decision_boundary(clf_entropy, X_train, y_train, ax=axs[1], title='Entropy (train)')
plt.tight_layout()
plt.savefig('gini_vs_entropy_boundaries.png', dpi=150)
plt.show()
```



However, the impurity choice can still affect the tree's structure, especially when several splits have nearly identical usefulness.

3. How Splits Are Chosen: The Greedy Algorithm

Decision trees use a **greedy** strategy:

1. Look at all possible splits for all features.
2. Compute impurity reduction for each split.
3. Choose the split that maximizes immediate gain.
4. Recur for the left child and right child.

This process continues until a stopping condition is met (e.g., max depth is reached, all leaves pure, or the node contains too few samples).

Key consequences of greediness:

- Trees do **not** find the globally optimal structure.
- Early split decisions strongly influence later structure.
- They may become large and overfit noisy data.

Nevertheless, the greedy approach is computationally efficient and works well for most real-world problems.

4. Overfitting and Tree Depth

One of the biggest weaknesses of decision trees is their tendency to **overfit**.

If unconstrained, a tree will:

- Continue splitting until every leaf is pure.
- Memorize the training data.
- Create extremely irregular decision boundaries.
- Perform poorly on unseen data.

This is the classical **bias–variance trade-off**:

4.1 Shallow Trees (Low Depth)

- High bias
- Underfit the data
- Produce smooth boundaries
- Miss important patterns

4.2 Deep Trees (High Depth)

- Low bias
- High variance
- Overfit noise
- Create jagged, complex regions

```python
fig, ax = plt.subplots(figsize=(7,5))
ax.plot([str(d) for d in depths], train_scores, marker='o', label='train')
ax.plot([str(d) for d in depths], test_scores, marker='o', label='test')
ax.set_xlabel('max_depth')
ax.set_ylabel('Accuracy')
ax.set_title('Effect of tree depth on accuracy (moons dataset)')
```

4.3 Controlling Overfitting

Key hyperparameters include:

- **max_depth** – Most effective control on complexity
- **min_samples_split** – Minimum samples needed before splitting
- **min_samples_leaf** – Minimum samples allowed in a leaf
- **max_leaf_nodes** – Upper bound on leaf count
- **ccp_alpha** – Cost-complexity pruning parameter

4.4 Pruning

Pruning grows a full tree and then removes branches that don't improve validation performance. This produces smoother, simpler, better-generalizing trees.

5. Feature Importance

Decision trees naturally provide **feature importance scores**, indicating which features were most useful for splitting.

Two main types exist:

5.1 Impurity-Based Importance (Default)

Each time a feature is used to split a node, the drop in impurity is recorded. Summing these impurity drops across all nodes gives the importance score.

Pros:

- Fast, built-in

- Useful for initial exploration

Cons:

- Biased toward numerical features with many unique values
- Biased when features are correlated

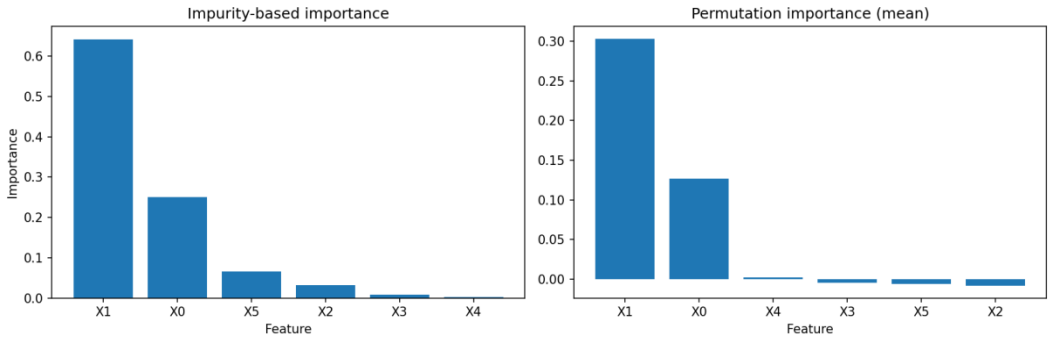
5.2 Permutation Importance

A model-agnostic and more reliable method:

1. Measure baseline performance on validation data.
2. Randomly shuffle values of feature XjX_jXj.
3. Performance drops by Δ = importance.

```
# Permutation importance
perm = permutation_importance(clf_full, X3_test, y3_test, n_repeats=30, random_state=0)
perm_df = pd.DataFrame({'feature': feature_names, 'perm_importance_mean': perm.importances_mean}).sort_values('perm_importance_mean', ascending=False)
print('\nPermutation importances:')
print(perm_df)

# Plot importances
fig, axs = plt.subplots(1,2, figsize=(12,4))
axs[0].bar(imp_df['feature'], imp_df['importance'])
axs[0].set_title('Impurity-based importance')
axs[0].set_xlabel('Feature')
axs[0].set_ylabel('Importance')
axs[1].bar(perm_df['feature'], perm_df['perm_importance_mean'])
axs[1].set_title('Permutation importance (mean)')
axs[1].set_xlabel('Feature')
plt.tight_layout()
plt.savefig('feature_importances.png', dpi=150)
plt.show()
```



If shuffling a feature significantly reduces accuracy, it is important.

Pros:

- Avoids high-cardinality bias
- Works with any model

Cons:

- More computationally expensive

6. Decision Boundaries

Decision trees create **axis-aligned partitioning** of the input space.

Example:

- A split like $x \leq 2.5$ draws a vertical line.
- A split like $y > 1.7$ draws a horizontal line.

This means:

- Trees approximate nonlinear boundaries using many small rectangles.
- Deeper trees → more rectangles → more irregular shapes.
- Overfitting appears as extremely jagged decision boundaries.

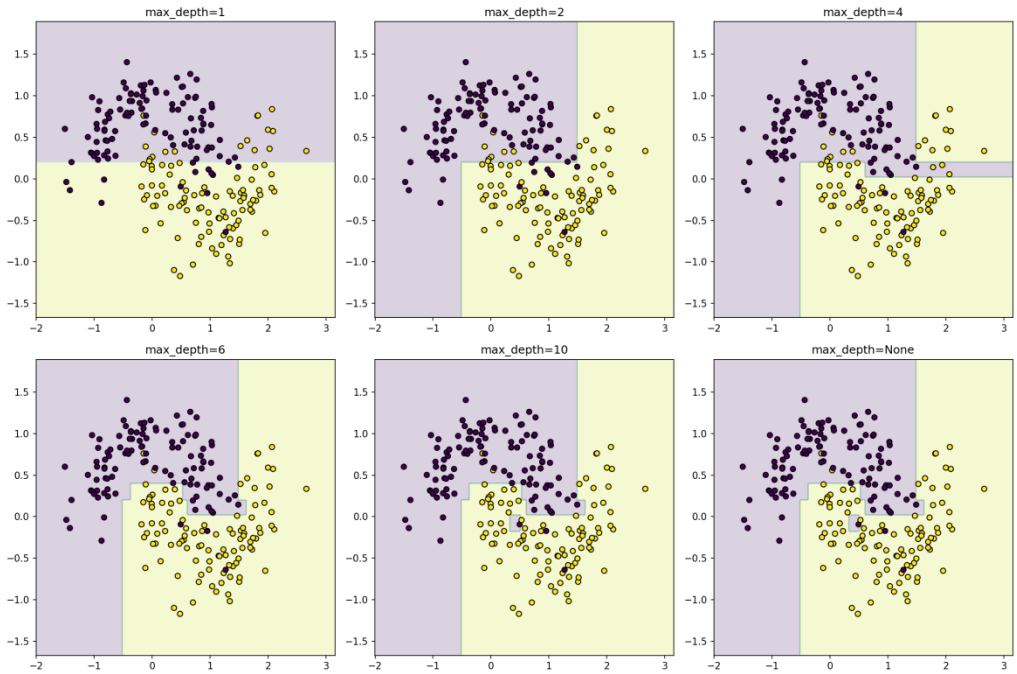
6.1 Shallow vs Deep Trees (Visual Intuition)

- **max_depth=2** → simple regions, coarse shapes
- **max_depth=4** → moderately refined boundaries
- **max_depth=10** → highly detailed boundary wrapping around noise

6.2 Example Image (Description)

If you were to plot a tree's decision boundary on a dataset like *make_moons*, you would see:

- Blue and orange background regions divided by vertical/horizontal slices
- Circular points (training data) grouped into rectangular zones
- X-shaped test points scattered across regions
- Shallower trees produce smoother, rectangular blocks
- Deeper trees carve very detailed shapes around individual points



```python
# Visualize decision boundaries for selected depths
fig, axs = plt.subplots(2, 3, figsize=(15,10))
selected = [1, 2, 4, 6, 10, None]
for ax, d in zip(axs.ravel(), selected):
    clf = DecisionTreeClassifier(max_depth=d, random_state=0)
    clf.fit(X2_train, y2_train)
    title = f'max_depth={d}'
    plot_decision_boundary(clf, X2_train, y2_train, ax=ax, title=title)
plt.tight_layout()
plt.savefig('depth_boundaries.png', dpi=150)
plt.show()
```

7. Practical Tips and Recommendations

To effectively use decision trees in real projects:

7.1 Choosing an impurity measure

- Use **Gini** by default → fast & stable
- Use **Entropy** if:
    - class balance is important
    - you want deeper, more information-theoretic splits

7.2 Preventing Overfitting

- Always tune **max_depth**
- Use **min_samples_leaf ≥ 2**
- Use **cross-validation** to choose hyperparameters
- Consider **pruning** (ccp_alpha)

7.3 Understanding feature importance

- Use permutation importance for trustworthy explanations
- Avoid relying solely on impurity-based scores when features are correlated

7.4 Handling class imbalance

- Use **class_weight='balanced'**
- Use metrics like F1, ROC-AUC, precision-recall

7.5 When to avoid a single decision tree

Use an ensemble (Random Forest or Gradient Boosted Trees) when:

- You need higher accuracy
- The dataset is large and noisy
- Stability is important

Single trees are excellent for interpretability but rarely achieve state-of-the-art accuracy on complex datasets.

8. Conclusion

Decision trees combine interpretability with practical flexibility, making them essential tools for both beginners and advanced practitioners. Despite their simple appearance, they rely on a combination of theoretical concepts (impurity measures, greedy optimization), structural choices (depth, pruning), and analytical tools (feature importance, decision boundary visualization).

Understanding these components enables:

- Better tuning for generalization
- Clearer model explanations
- More effective use of ensembles
- Insightful explorations of datasets

Mastering decision trees provides a foundation for many advanced methods that dominate modern machine learning, such as Random Forests, XGBoost, and Gradient Boosting.

GitHub Link : https://github.com/Valluru-Abhinaya/Machine-Learning---Tutorial