Comprehensive Review: Decision Trees

Master's Level Data Science

Contents

1 Introduction

This review synthesizes material from the lecture slides (`dtree-1.pdf`) and audio transcript (`DecisionTreeBasics.`
We cover the case study, the learning algorithm, uncertainty measures, worked examples, and practical considerations for decision trees.

2 Mathematical Formulations

2.1 Uncertainty Measures

Let a node contain a dataset S with K classes. Denote by p_i the fraction of points in class i, so $\sum_{i=1}^{K} p_i = 1$. We define:

1. *Misclassification Rate*:

$$u_{\text{mis}}(S) = 1 - \max_i p_i = \min_i \bigl(1 - p_i\bigr)$$

2. *Gini Index*:

$$u_{\text{gini}}(S) = \sum_{i=1}^{K} p_i(1 - p_i) = 1 - \sum_{i=1}^{K} p_i^2$$

3. *Entropy*:

$$u_{\text{ent}}(S) = -\sum_{i=1}^{K} p_i \, \log(p_i)$$

(All logs are natural logarithms.)

2.2 Benefit of a Split

Consider splitting S into S_L and S_R, with fractions $p_L = |S_L|/|S|$ and $p_R = |S_R|/|S|$. Let $u(\cdot)$ be any uncertainty measure. Then the *reduction in uncertainty* is

$$\Delta u = u(S) \; - \; \big[p_L \, u(S_L) + p_R \, u(S_R) \big].$$

Often we weight by $|S|$ when comparing across nodes, but the greedy algorithm simply picks the split maximizing Δu.

3 Geometric Illustrations

3.1 Binary Splits in $\mathbb{R}^2$

Below is a TikZ illustration of two successive splits on features x_1 and x_2.

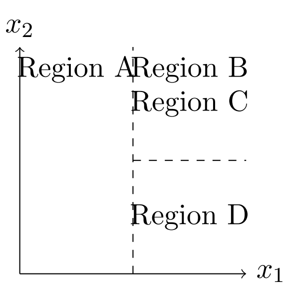

Figure 1: Illustration of two-level axis-aligned splits in $\mathbb{R}^2$.

4 Worked Example

We demonstrate with Python and scikit-learn on a synthetic two-dimensional dataset.

4.1 Data Acquisition and Preprocessing

We generate a toy dataset of two classes separable by decision tree.

```
import numpy as np
from sklearn.datasets import make_classification
X, y = make_classification(
    n_samples=200, n_features=2, n_informative=2,
    n_redundant=0, n_clusters_per_class=1, random_state=42
)
```

4.2 Feature Representation

We standardize features for numerical stability.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

4.3 Model Training

Train a CART decision tree classifier using Gini index.

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(
    criterion='gini',
    max_depth=3,
    random_state=42
)
clf.fit(X_scaled, y)
```

4.4 Model Evaluation

Split data, compute accuracy and classification report.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

X_tr, X_te, y_tr, y_te = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42
)
clf.fit(X_tr, y_tr)
y_pred = clf.predict(X_te)
acc = accuracy_score(y_te, y_pred)
print(f'Accuracy: {acc:.2f}')
print(classification_report(y_te, y_pred))
```

5 Algorithm Description

The greedy top-down tree-building algorithm (CART) proceeds:

1. **Initialize**: Start with root node containing all data.
2. **Evaluate Splits**: For each leaf node, examine all features and all candidate thresholds (midpoints between sorted unique values).

3. **Compute Uncertainty Reduction**: For each candidate split, compute $\Delta u = u(S) - [p_L u(S_L) + p_R u(S_R)]$.

4. **Select Best Split**: Choose the leaf and split yielding maximum Δu.

5. **Partition**: Split the chosen leaf into two child nodes.

6. **Repeat**: Continue until stopping criteria (max depth, min samples, or zero uncertainty) are met.

6 Empirical Results

We study the effect of tree depth on test accuracy.

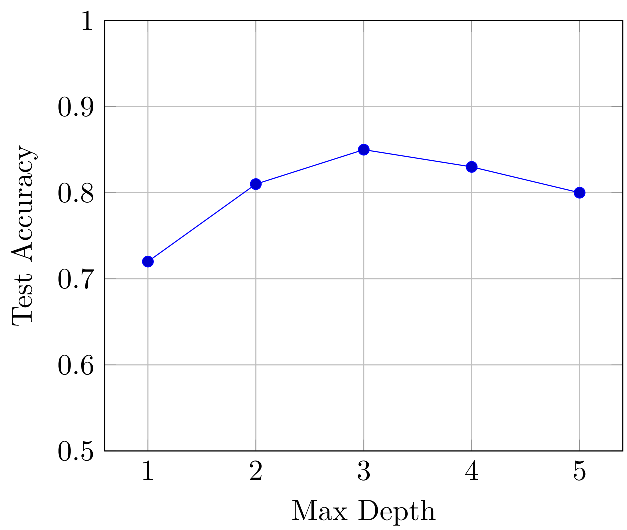

Figure 2: Accuracy vs. maximum tree depth on test set.

7 Interpretation & Guidelines

- **Bias-Variance Tradeoff**: Shallow trees underfit (high bias), deep trees overfit (high variance).
- **Stopping Criteria**: Limit depth, require minimum samples per leaf, or prune post hoc to avoid overfitting.
- **Feature Engineering**: Categorical features may be one-hot encoded; ordinal splits retain order.
- **Interpretability**: Trees provide clear question–answer rules favored in domains requiring transparency.

8 Future Directions / Extensions

- **Ensembles**: Random Forests and Gradient Boosted Trees improve accuracy and robustness.
- **Oblique Splits**: Allow linear combinations of features at splits for more flexibility.
- **Cost-Sensitive Trees**: Incorporate asymmetric misclassification costs.
- **Online Trees**: Incremental updates for streaming data.