Module 7: KNN, Decision Trees, and Ensembles

Overview

This module introduces non-parametric and tree-based models—approaches that differ fundamentally from the neural networks of Module 6. K-Nearest Neighbors makes predictions based on similar training examples. Decision trees partition the feature space using interpretable rules. Ensemble methods like Random Forests and Gradient Boosting combine multiple models to achieve state-of-the-art performance on tabular data.

---

1. K-Nearest Neighbors

The Algorithm

KNN is beautifully simple: 1. Store all training data 2. For a new example, find the k closest training examples 3. Predict by majority vote (classification) or average (regression)

Distance Metrics

Euclidean Distance:

$$d(x, y) = \sqrt{(\Sigma(x - y)^2)}$$

Manhattan Distance:

$$d(x, y) = \Sigma|x - y|$$

The k Hyperparameter

- **k=1**: Predict same as nearest neighbor (high variance)
- **Small k**: Complex boundaries, risk of overfitting
- **Large k**: Smoother boundaries, risk of underfitting
- **k = $\sqrt{n}$**: Common starting point

Strengths and Weaknesses

Strengths: - Simple to understand and implement - No training phase - Naturally handles multi-class - Non-linear decision boundaries

Weaknesses: - Slow prediction (O(n) per query) - Sensitive to irrelevant features - Requires feature scaling - Curse of dimensionality

---

2. Decision Trees

The Idea

Decision trees recursively partition the feature space using axis-aligned splits:

```
if feature   threshold :
    if feature   threshold :
        predict class A
    else:
        predict class B
else:
    predict class C
```

Splitting Criteria

How do we choose which feature and threshold to split on?

Entropy:

$$H(S) = -\Sigma\ p\ \log(p)$$

Measures disorder. Low entropy = pure (one class dominates).

Information Gain:

$$IG(S, A) = H(S) - \Sigma\ (|S|/|S|) \times H(S)$$

Reduction in entropy after splitting on attribute A.

Gini Impurity:

$$Gini(S) = 1 - \Sigma\ p^2$$

Alternative to entropy, often used in practice (CART).

Building the Tree

1. Start with all data at root
2. Find the split that maximizes information gain
3. Create child nodes for each partition
4. Recurse on each child
5. Stop when: pure node, max depth, or min samples

---

3. Preventing Overfitting in Trees

The Problem

Deep trees can perfectly fit training data but fail on new data.

Regularization Strategies

Pre-pruning (stop early): - Maximum depth - Minimum samples per leaf - Minimum information gain

Post-pruning (grow then trim): - Remove nodes that don't improve validation - Cost-complexity pruning

Hyperparameters

Parameter	Effect when increased
max_depth	More complex, more overfitting
min_samples_split	Simpler, less overfitting
min_samples_leaf	Simpler, smoother predictions

---

4. Ensemble Methods

Why Ensembles?

Individual models have errors. If errors are independent, combining models reduces total error.

Variance reduction: Average noisy predictions smoother result **Bias reduction:** Combine complementary models

Bagging (Bootstrap Aggregating)

1. Create multiple bootstrap samples (random with replacement)
2. Train a model on each sample
3. Combine by voting/averaging

Reduces variance without increasing bias.

Random Forests

Bagging + feature randomization: - Each tree trained on bootstrap sample - At each split, consider only random subset of features - Decorrelates trees more variance reduction

Random forests are often the best “out-of-box” method for tabular data.

---

5. Gradient Boosting

The Idea

Build models sequentially, each correcting errors of the previous:

1. Train $model_1$ on data
2. Compute residuals (errors)
3. Train $model_2$ to predict residuals
4. Add $model_2$’s predictions (scaled) to $model_1$’s
5. Repeat

Key Hyperparameters

- **n_estimators**: Number of trees/rounds
- **learning_rate**: Shrinkage per round (smaller = more trees needed)
- **max_depth**: Usually shallow (3-8) for boosting

Implementations

- **XGBoost**: Fast, regularized, widely used
- **LightGBM**: Even faster, histogram-based
- **CatBoost**: Handles categorical features well

---

6. Comparing Methods

Method	Interpretable?	Handles Non-linearity	Training Speed	Prediction Speed
KNN	Medium	Yes	None	Slow
Decision Tree	High	Yes	Fast	Fast
Random Forest	Low	Yes	Medium	Medium
Gradient Boosting	Low	Yes	Slow	Fast

When to Use What

- **KNN**: Small data, need simplicity
- **Single Tree**: Need interpretability
- **Random Forest**: Default for tabular data, robust
- **Gradient Boosting**: Maximum performance, can tune

---

7. Feature Importance

Why It Matters

Understanding which features matter helps: - Interpret the model - Remove irrelevant features - Debug unexpected predictions

Tree-based Importance

For each feature, sum the improvement (information gain) across all splits using that feature, weighted by samples.

Random Forest Importance

Average importance across all trees. More stable than single tree.

Permutation Importance

1. Evaluate model on validation set
2. Shuffle one feature
3. Measure performance drop
4. Large drop = important feature

---

8. Practical Considerations

Scaling

- **KNN**: Requires scaling (distances affected)
- **Trees**: Do NOT require scaling (splits are invariant)

Missing Values

- **KNN**: Usually requires imputation
- **Trees**: Can handle natively (XGBoost, LightGBM)

Categorical Features

- **KNN**: Requires encoding
- **Trees**: Can split on categories directly (some implementations)

Key Takeaways

1. **KNN** predicts by similarity—simple but slow
2. **Decision trees** partition space with interpretable rules
3. **Entropy and information gain** guide tree construction
4. **Overfitting** is controlled by depth limits and pruning
5. **Bagging** reduces variance by averaging
6. **Random Forests** add feature randomization for decorrelation
7. **Gradient Boosting** builds sequential error-correcting models
8. **Ensembles** often outperform single models

Connections to Future Modules

- **Module 11**: Trees in pipelines
- **Module 12**: Fairness considerations with tree models

Further Reading

- [Links to be added]
- Introduction to Statistical Learning, Chapters 8
- XGBoost documentation