Decision Trees, Entropy, Routing Networks & Mixture of Experts
Structured Study Notes (Compiled from Interactive Discussion)

1. Why Study Decision Trees Today

Decision Trees are studied not because they are the most powerful modern models, but because they reveal the fundamental structure of learned decision-making. They represent the smallest complete model of intelligence: features, loss, learning, generalization, and reasoning.

Decision Trees are the conceptual ancestors of routing networks and Mixture of Experts models used in modern large language models and agentic systems.

2. Core Intuition of Decision Trees

A decision tree repeatedly asks questions that reduce uncertainty in the data. Each internal node is a decision rule, and each leaf node is an expert that predicts an output.

3. Gini Impurity

Gini impurity measures how mixed the class labels are at a node.

```
def gini(labels):
    values, counts = np.unique(labels, return_counts=True)
    probabilities = counts / counts.sum()
    return 1 - np.sum(probabilities ** 2)
```

4. Entropy

Entropy comes from information theory and measures uncertainty. It is maximized when classes are equally likely and zero when the node is pure.

```
def entropy(labels):
    values, counts = np.unique(labels, return_counts=True)
    probabilities = counts / counts.sum()
    return -np.sum(probabilities * np.log2(probabilities))
```

5. Information Gain

Information Gain measures how much entropy is reduced by a split. Decision trees choose splits that maximize information gain.

```
def information_gain(parent, left, right, target):
    parent_entropy = entropy(parent[target])
    n = len(left) + len(right)
```

```
    weighted_entropy = (
        len(left)/n * entropy(left[target]) +
        len(right)/n * entropy(right[target])
    )
    return parent_entropy - weighted_entropy
```

6. Finding the Best Split

The algorithm evaluates all possible feature-threshold pairs and selects the one that minimizes impurity or maximizes information gain.

```
def best_split(df, target):
    best_feature = None
    best_threshold = None
    best_score = float("inf")

    for feature in df.columns:
        if feature == target:
            continue

        thresholds = df[feature].unique()
        for t in thresholds:
            left = df[df[feature] <= t]
            right = df[df[feature] > t]
            if len(left) == 0 or len(right) == 0:
                continue
            score = weighted_gini(left, right, target)
            if score < best_score:
                best_score = score
                best_feature = feature
                best_threshold = t

    return best_feature, best_threshold, best_score
```

7. Majority Class at Leaf Nodes

When the tree can no longer split, it predicts the majority class among the samples that reached the node.

```
def majority_class(labels):
    return labels.value_counts().idxmax()
```

This counts occurrences of each class and returns the class with the highest count. It minimizes misclassification error at the leaf.

8. Building the Tree Recursively

```
class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None, value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value


def build_tree(df, target, depth=0, max_depth=3):
    labels = df[target]
```

```
    if len(labels.unique()) == 1:
        return Node(value=labels.iloc[0])

    if depth >= max_depth:
        return Node(value=majority_class(labels))

    feature, threshold, score = best_split(df, target)
    if feature is None:
        return Node(value=majority_class(labels))

    left_df = df[df[feature] <= threshold]
    right_df = df[df[feature] > threshold]

    left_child = build_tree(left_df, target, depth+1, max_depth)
    right_child = build_tree(right_df, target, depth+1, max_depth)

    return Node(feature, threshold, left_child, right_child)
```

9. Prediction

```
def predict(node, sample):
    if node.value is not None:
        return node.value
    if sample[node.feature] <= node.threshold:
        return predict(node.left, sample)
    else:
        return predict(node.right, sample)
```

10. Decision Trees → Routing Networks → Mixture of Experts

Decision Trees perform hard routing: each input follows exactly one path. Routing networks soften this idea by assigning probabilities to multiple paths. Mixture of Experts scales this concept by routing inputs to a small subset of neural experts.

Modern MoE-based Transformers use the same core idea as decision trees: conditional computation through routing.

11. Key Takeaway

Decision Trees are not obsolete. They are the conceptual foundation of modern sparse, routed, and agentic AI systems.