Slide 1: Multi-Level Classification

Multi-level classification, also known as hierarchical classification, is a machine learning approach where labels are organized in a hierarchical structure, often resembling a tree or a directed acyclic graph (DAG). In contrast to flat classification, where classes are independent of each other, multi-level classification captures relationships between labels, such as parent-child or sibling relationships, allowing for more complex decision-making processes.

The goal of multi-level classification is to predict labels at multiple levels of the hierarchy. For example, in an e-commerce product categorization system, a product like a smartphone may belong to categories like "Electronics" (parent) and "Mobile Phones" (child).

Types of Multi-Level Classification

1. Local Classifiers per Parent Node: A separate classifier is trained for each parent node. For example, once a parent category like "Electronics" is predicted, a new classifier will determine which child class (like "Mobile Phones" or "Laptops") to choose.
2. Local Classifiers per Level: A classifier is trained for each level of the hierarchy. For example, the first classifier will determine the high-level category (e.g., "Electronics" or "Home Appliances"), and subsequent classifiers predict more granular categories (like "Mobile Phones" or "Washing Machines").
3. Global Classifier: A single model is used to predict the entire path from the root to the leaf node in the hierarchy.

Please give pictorial representation as well

Slide 2: Decision Trees Algorithm

A Decision Tree is a tree-like structure where each internal node represents a decision on a feature, each branch represents the outcome of the decision, and each leaf node represents a class label. The model recursively splits the data based on the features that provide the highest information gain (or lowest Gini impurity).

simple representation of a Decision Tree for the "Play Tennis" example based on the features: Outlook, Temperature, Humidity, and Wind.

Dataset

Outlook	Temperature	Humidity	Wind	Play Tennis?
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes

Decision Tree Representation

For simplicity, let’s assume that Outlook is the feature with the highest information gain for our first split. The tree could look something like this:

```
               Outlook
              /   |    \
        Sunny   Overcast  Rainy
         /        |        \
 Humidity      Play        Wind
    /          Tennis     /   \
 High        (Yes)     Weak    Strong
  /                     /         \
No                    Yes         No
```

Slide 3: Gini Impurity

Before Split:

Probability of "Yes" $p_{\text{Yes}} = \frac{3}{5} = 0.6$
Probability of "No" $p_{\text{No}} = \frac{2}{5} = 0.4$

Using the Gini formula:

$$\text{Gini Impurity (before split)} = 1 - (p_{\text{Yes}}^2 + p_{\text{No}}^2)$$

$$= 1 - (0.6^2 + 0.4^2) = 1 - (0.36 + 0.16) = 1 - 0.52 = 0.48$$

After Split:

Split Details

1. **Outlook = Sunny**: 2 samples (both are "No").
2. **Outlook = Overcast**: 1 sample (label is "Yes").
3. **Outlook = Rainy**: 2 samples (both are "Yes").

Let’s calculate the Gini Impurity for each child node:

- **Node 1 (Sunny)**:
  - Both samples are "No".
  - Gini Impurity for this node:
    $$= 1 - (1^2 + 0^2) = 1 - 1 = 0$$
- **Node 2 (Overcast)**:
  - The sample is "Yes".
  - Gini Impurity for this node:
    $$= 1 - (1^2 + 0^2) = 1 - 1 = 0$$
- **Node 3 (Rainy)**:
  - Both samples are "Yes".
  - Gini Impurity for this node:
    $$= 1 - (1^2 + 0^2) = 1 - 1 = 0$$

1. **Sunny**: $\frac{2}{5} \times 0 = 0$
2. **Overcast**: $\frac{1}{5} \times 0 = 0$
3. **Rainy**: $\frac{2}{5} \times 0 = 0$

The total Gini Impurity after the split:

$$\text{Gini Impurity (after split)} = 0 + 0 + 0 = 0$$

Slide 4: Entropy

Before split:

The formula for entropy is:

$$\text{Entropy} = -\sum_{i=1}^{K} p_i \log_2(p_i)$$

where p_i is the probability of each class i.

Probability of "Yes" $p_{\text{Yes}} = \frac{3}{5} = 0.6$
Probability of "No" $p_{\text{No}} = \frac{2}{5} = 0.4$

Now we substitute these values into the formula:

$$\text{Entropy (before split)} = -(0.6 \cdot \log_2(0.6) + 0.4 \cdot \log_2(0.4))$$

Calculating each term:

1. $0.6 \cdot \log_2(0.6) \approx -0.442$
2. $0.4 \cdot \log_2(0.4) \approx -0.528$

So,

$$\text{Entropy (before split)} = 0.442 + 0.528 = 0.970$$

After split:

Split Details

1. **Outlook = Sunny**: 2 samples, both labeled "No."
2. **Outlook = Overcast**: 1 sample, labeled "Yes."
3. **Outlook = Rainy**: 2 samples, both labeled "Yes."

Let's calculate the Entropy for each of these child nodes.

- **Node 1 (Sunny)**:
  - Both samples are "No" (pure node).
  - Entropy:
$$\text{Entropy} = -(1 \cdot \log_2(1) + 0 \cdot \log_2(0)) = 0$$
- **Node 2 (Overcast)**:
  - The single sample is "Yes" (pure node).
  - Entropy:
$$\text{Entropy} = -(1 \cdot \log_2(1) + 0 \cdot \log_2(0)) = 0$$
- **Node 3 (Rainy)**:
  - Both samples are "Yes" (pure node).
  - Entropy:
$$\text{Entropy} = -(1 \cdot \log_2(1) + 0 \cdot \log_2(0)) = 0$$

1. **Sunny**: $\frac{2}{5} \times 0 = 0$
2. **Overcast**: $\frac{1}{5} \times 0 = 0$
3. **Rainy**: $\frac{2}{5} \times 0 = 0$

The total Entropy after the split is:

$$\text{Entropy (after split)} = 0 + 0 + 0 = 0$$

Slide 5: Overfitting

Overfitting in a Decision Tree occurs when the tree learns not just the general patterns in the data but also the noise or outliers, leading to a model that performs very well on the training data but poorly on new, unseen data. Overfitting typically happens when the tree grows too deep, capturing every detail of the training data.

```
                          Outlook
                       /     |      \
                  Sunny   Overcast   Rainy
                 /     |      |       |
          Humidity    Yes    Wind    Temperature
           /      \            |        |
       High      Normal      Weak     Mild
       /   \         |        |        |
  Wind   Temperature Yes     Yes      Cool
   |          |                     /     \
Weak      Mild                   Yes     Strong
  |           |                            |
 No        Yes                           No
```

- Low Generalization: With this structure, the tree is very specific to this dataset. If we test it on new data, such as another combination of "Sunny" and "Humidity = High" but with different wind conditions, the model might misclassify due to the overly complex structure.

In practice, pruning (removing unnecessary branches) or limiting tree depth can help avoid overfitting, making the model simpler and better at generalizing to new data.

Slide 6: Random Forest Ensemble Averging

Random Forests reduce the likelihood of overfitting by creating an ensemble of multiple Decision Trees, each trained on different random subsets of the data and features. This randomness introduces diversity among the trees, so they capture general patterns rather than specific details or noise in the training data.

Tree 1:

```
                Outlook
               /       \
         Sunny          Overcast
          |                |
     Humidity             Yes
       /     \
    High     Normal
    /            \
   No             Yes
```

Tree 2:

```
             Humidity
             /       \
        High        Normal
       /                 \
    Wind                 Yes
     /    \
 Weak    Strong
  |         |
Yes         No
```

Tree 3:

```
           Outlook
          /       \
     Sunny         Rainy
      |              |
    Wind            Yes
   /     \
Weak      Strong
 |           |
No          Yes
```

Example Scenario:
To make a prediction for new data, say, `Outlook = Sunny`, `Humidity = High`, `Wind = Weak`:

1. Tree 1: Predicts No based on the path `Outlook = Sunny` → `Humidity = High`.
2. Tree 2: Predicts Yes based on the path `Humidity = High` → `Wind = Weak`.
3. Tree 3: Predicts No based on the path `Outlook = Sunny` → `Wind = Weak`.

With majority voting, the final prediction for this instance is No (since two out of three trees predict "No").

Slide 7: Evaluation Metrics

Accuracy

- **Definition**: The proportion of correctly classified instances out of the total instances.
- **Formula**:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Confusion Matrix:

Actual / Predicted	**Fraud (Positive)**	**Legitimate (Negative)**
Fraud	TP = 70	FN = 30
Legitimate	FP = 20	TN = 880

Step 1: Calculate Precision

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{70}{70 + 20} = \frac{70}{90} \approx 0.778$$

Precision of **0.778** means that about **77.8% of transactions predicted as fraud were actually fraud**.

Step 2: Calculate Recall

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{70}{70 + 30} = \frac{70}{100} = 0.7$$

Recall of **0.7** means that the model **correctly identified 70% of the actual fraudulent transactions**.

Step 3: Calculate F1 Score

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{0.778 \times 0.7}{0.778 + 0.7}$$

Calculating the numerator:

$$0.778 \times 0.7 = 0.5446$$

Calculating the denominator:

$$0.778 + 0.7 = 1.478$$

$$\text{F1 Score} = 2 \times \frac{0.5446}{1.478} \approx 2 \times 0.3685 = 0.737$$

Precision is important in situations where false positives are costly.
Recall is critical in situations where missing positives is costly.
F1 Score balances Precision and Recall, making it suitable when both false positives and false negatives carry significant costs.

Slide 8: Decision Tree

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
# Sample dataset
data = {
    'weight': [1.5, 2.0, 0.5, 0.3, 5.0, 0.7, 2.2],
    'dimensions': [15, 20, 5, 3, 50, 7, 25],
    'category': ['Electronics', 'Electronics', 'Accessories', 'Accessories', 'Furniture',
'Accessories', 'Electronics']
}

# Create DataFrame
df = pd.DataFrame(data)

# Feature matrix and target variable
X = df[['weight', 'dimensions']]
y = df['category']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier(max_depth=3, random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Slide 9: Random Forest

```
# Import necessary libraries
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

```
# Sample text dataset
text_data = {
    'review': [
        'This product is amazing!',
        'Terrible, not worth the money.',
        'Very satisfied with the quality.',
        'I would not recommend this product.',
        'Exceeded my expectations!',
        'Waste of money, very disappointing.',
        'I love it, great quality and value.'
    ],
    'sentiment': ['Positive', 'Negative', 'Positive', 'Negative', 'Positive', 'Negative', 'Positive']
}

# Create DataFrame
df = pd.DataFrame(text_data)

# Features and target
X = df['review']
y = df['sentiment']

# Convert text data to TF-IDF features
tfidf = TfidfVectorizer(max_features=100)
X_tfidf = tfidf.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.3,
random_state=42)

# Initialize the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf.fit(X_train, y_train)

# Make predictions
y_pred = rf.predict(X_test)

# Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```