Task 1 - Classification with Custom Decision Trees

This task utilises the Nursery Data Set and implements a decision tree classifier from scratch to preduct nursery application ranking

In [93]:
```
# Import necessary libraries
import csv
from collections import Counter
import random
import math
```

In [94]:
```
# Load and read dataset
def nursery_data(filepath='datasets/nursery.data'):
    X, y = [], []
    with open(filepath, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            if row:
                X.append(row[:-1])
                y.append(row[-1])
    return X, y
```

In [95]:
```
# Encode categorical data to integers
def encode_data(X, y):
    encoders = [{} for _ in range(len(X[0]))]
    for col in range(len(X[0])):
        unique_vals = sorted(set(row[col] for row in X))
        encoders[col] = {val: i for i, val in enumerate(unique_vals)}
        for row in X:
            row[col] = encoders[col][row[col]]
    label_map = {val: i for i, val in enumerate(sorted(set(y)))}
    y = [label_map[label] for label in y]
    return X, y, encoders, label_map
```

In [96]:
```
# Split dataset
def train_test_split(X, y, train_ratio=0.65):
    combined = list(zip(X, y))
    random.shuffle(combined)
    X[:], y[:] = zip(*combined)
    split_idx = int(len(X) * train_ratio)
    return X[:split_idx], y[:split_idx], X[split_idx:], y[split_idx:]
```

2. Create the Decision Tree Components

In [97]:
```
# Define the Decision Tree Classifier
class TreeNode:
    def __init__(self, feature=None, children=None, prediction=None):
        self.feature = feature
        self.children = children or {}
        self.prediction = prediction
```

```python
    def is_leaf(self):
        return self.prediction is not None
```

In [98]:
```python
# Entropy & Info Gain
def entropy(labels):
    total = len(labels)
    counts = Counter(labels)
    return -sum((c/total) * math.log2(c/total) for c in counts.values())

def info_gain_tree(parent, branches):
    total = len(parent)
    weighted = sum(len(b)/total * entropy(b) for b in branches)
    return entropy(parent) - weighted
```

In [99]:
```python
# Gini Index
def gini_index(labels):
    total = len(labels)
    counts = Counter(labels)
    return 1 - sum((c/total)**2 for c in counts.values())

def gini_gain(parent, branches):
    total = len(parent)
    return -sum(len(b)/total * gini_index(b) for b in branches)
```

In [100…
```python
# Dataset Split
def split_by_feature(X, y, feature):
    result = {}
    for xi, label in zip(X, y):
        key = xi[feature]
        if key not in result:
            result[key] = ([], [])
        result[key][0].append(xi)
        result[key][1].append(label)
    return result
```

In [101…
```python
# Tree Builder
def build_tree(X, y, criterion='info_gain'):
    if len(set(y)) == 1:
        return TreeNode(prediction=y[0])

    if not X[0]:
        return TreeNode(prediction=Counter(y).most_common(1)[0][0])

    best_gain = -float('inf')
    best_feature = None
    best_splits = None

    for i in range(len(X[0])):
        splits = split_by_feature(X, y, i)
        branches = [labels for _, labels in splits.values()]
        gain = info_gain_tree(y, branches) if criterion == 'info_gain' else gini
        if gain > best_gain:
            best_gain = gain
            best_feature = i
            best_splits = splits

    if best_feature is None or best_gain <= 0:
        return TreeNode(prediction=Counter(y).most_common(1)[0][0])
```

```python
    children = {}
    for val, (x_subset, y_subset) in best_splits.items():
        children[val] = build_tree(x_subset, y_subset, criterion)

    return TreeNode(feature=best_feature, children=children)
```

In [102…

```python
# Main function to run the classifier
def predict(tree, sample):
    while not tree.is_leaf():
        val = sample[tree.feature]
        if val in tree.children:
            tree = tree.children[val]
        else:
            return None
    return tree.prediction

def accuracy(y_true, y_pred):
    return sum(1 for yt, yp in zip(y_true, y_pred) if yt == yp) / len(y_true)

def confusion_matrix(y_true, y_pred, labels):
    matrix = [[0]*len(labels) for _ in labels]
    for yt, yp in zip(y_true, y_pred):
        if yp is not None:
            matrix[yt][yp] += 1
    return matrix
```

In [103…

```python
# Ensemble Voting
def ensemble_predict(sample, trees, weights):
    votes = {}
    for tree, weight in zip(trees, weights):
        pred = predict(tree, sample)
        if pred is not None:
            votes[pred] = votes.get(pred, 0) + weight
    return max(votes.items(), key=lambda x: x[1])[0]
```

In [104…

```python
#Tree Printing
def print_tree(node, depth=0):
    indent = "   " * depth
    if node.is_leaf():
        print(f"{indent}Predict: {node.prediction}")
    else:
        for val, child in node.children.items():
            print(f"{indent}If feature[{node.feature}] == {val}:")
            print_tree(child, depth+1)
```

In [105…

```python
# Load and prepare data
X, y = nursery_data()
X, y, encoders, label_map = encode_data(X, y)

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

# Train both trees
tree_info = build_tree(X_train, y_train, criterion='info_gain')
tree_gini = build_tree(X_train, y_train, criterion='gini')

# Predict & evaluate
y_pred_info = [predict(tree_info, x) for x in X_test]
y_pred_gini = [predict(tree_gini, x) for x in X_test]
```

```python
acc_info = accuracy(y_test, y_pred_info)
acc_gini = accuracy(y_test, y_pred_gini)

print("Information Gain Tree Accuracy:", acc_info)
print("Gini Index Accuracy:", acc_gini)

# Ensemble voting
ensemble_preds = [ensemble_predict(x, [tree_info, tree_gini], weights=[0.5, 0.5]
ensemble_acc = accuracy(y_test, ensemble_preds)
print("Ensemble Accuracy:", ensemble_acc)

# Confusion Matrix
labels = sorted(set(y))
matrix_info = confusion_matrix(y_test, y_pred_info, labels)
matrix_gini = confusion_matrix(y_test, y_pred_gini, labels)
matrix_ensemble = confusion_matrix(y_test, ensemble_preds, labels)

# Simple print
print("\nConfusion Matrix (Information Gain):")
for row in matrix_info:
    print(row)

print("\nConfusion Matrix (Gini Index):")
for row in matrix_gini:
    print(row)

print("\nConfusion Matrix (Ensemble):")
for row in matrix_ensemble:
    print(row)
```

```
Information Gain Tree Accuracy: 0.9717813051146384
Gini Index Accuracy: 0.32429453262786595
Ensemble Accuracy: 0.9832451499118166

Confusion Matrix (Information Gain):
[1526, 0, 0, 0, 0]
[0, 1402, 0, 11, 6]
[0, 0, 0, 0, 1]
[0, 7, 0, 1398, 0]
[0, 5, 3, 0, 82]

Confusion Matrix (Gini Index):
[0, 1526, 0, 0, 0]
[0, 1471, 0, 0, 0]
[0, 1, 0, 0, 0]
[0, 1438, 0, 0, 0]
[0, 100, 0, 0, 0]

Confusion Matrix (Ensemble):
[1526, 0, 0, 0, 0]
[0, 1454, 0, 11, 6]
[0, 0, 0, 0, 1]
[0, 40, 0, 1398, 0]
[0, 15, 3, 0, 82]
```

Conclusion

- Information Gain Tree (around 97%) and Ensemble (around 98%) were much more accurate than the Gini Index Tree (32%)
- Information Gain showed high accuracy for most classes except priority and spec_prior
- Gini Index Tree showed well enough accuracy for classes other than the spec_prior
- Ensemble Tree improved in priority and spec_prior classes but showed some struggle