# code

September 3, 2023

[139]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

[140]:
```
path = '../../dataset/decision-tree.csv'
df = pd.read_csv(path)

features = [feat for feat in df]
features.remove("Outcome")
```

[141]:
```
temp, test_df = train_test_split(df, test_size = 0.2, random_state = 10)
train_df, val_df = train_test_split(temp, test_size = 0.25, random_state = 10)
```

[142]:
```
def calc_entropy(data):
    label_column = data[:, -1]
    _, counts = np.unique(label_column, return_counts=True)

    probabilities = counts / counts.sum()
    entropy = sum(probabilities * -np.log2(probabilities))

    return entropy

def calc_info_gain(data,column_no,value):
    parent_entropy = calc_entropy(data)

    left_data = data[data[:,column_no]<=value]
    right_data = data[data[:,column_no]>value]

    n = len(left_data) + len(right_data)
    p_left_data = len(left_data) / n
    p_right_data = len(right_data) / n

    child_entropy =  (p_left_data * calc_entropy(left_data) + p_right_data *␣
 ↪calc_entropy(right_data))

    information_gain = parent_entropy - child_entropy
```

```
    return information_gain

def splits(data,col_idx):
    potential_splits = []
    values = data[:, col_idx]
    unique_values = np.unique(values)
    for index in range(len(unique_values)):
        if index != 0:
            current_value = unique_values[index]
            previous_value = unique_values[index - 1]
            potential_split = (current_value + previous_value) / 2
            potential_splits.append(potential_split)
    return potential_splits

def calc_best_gain(data,column_no):
    best_gain = -1
    split_threshold = None
    potential_splits = splits(data,column_no)
    for value in potential_splits:
        # calculate the information gain
        gain = calc_info_gain(data, column_no, value)
        if gain > best_gain:
            best_gain = gain
            split_threshold = value
    return best_gain,split_threshold

def find_most_info_feat(data):
    max_info_gain = -1
    max_info_feat = None
    selected_threshold = None
    for col in range (0,data.shape[1]-1):
        gain,threshold = calc_best_gain(data,col)
        if max_info_gain < gain:
            max_info_gain = gain
            max_info_feat = features[col]
            selected_threshold = threshold
    return max_info_feat,selected_threshold
```

[143]:
```
def get_most_common_label(data):
    labels = data[:, -1]
    unique_labels, counts = np.unique(labels, return_counts=True)
    most_common_label = unique_labels[np.argmax(counts)]
    return most_common_label

def all_same_class(data):
    labels = data[:, -1]
    return len(np.unique(labels)) == 1
```

[144]:
```
class Node:
    def __init__(self, label, data, depth):
        self.data = data
        self.depth = depth
        self.children = []
        self.feature_name = None
        self.threshold = None
        self.label = label
    def copy(self):
        new_node = Node(self.label, self.data, self.depth)
        new_node.children = [child.copy() for child in self.children]
        new_node.feature_name = self.feature_name
        new_node.threshold = self.threshold
        new_node.label = self.label
        return new_node

def print_decision_tree(node, indent=""):
    if node is None:
        return

    if node.label is not None:
        print(indent + "Leaf: Class", node.label)
    else:
        print(indent + "Node: Split on", node.feature_name, "<=", node.
 ↪threshold)
        for child in node.children:
            print_decision_tree(child, indent + "  ")

def predict(tree, sample):
    if tree.label is not None:
        return tree.label

    # Check if the children list is empty
    if not tree.children:
        return None  # Indicate that the prediction couldn't be made

    if sample[features.index(tree.feature_name)] <= tree.threshold:
        return predict(tree.children[0], sample)
    else:
        return predict(tree.children[1], sample)

def evaluate_decision_tree(tree, test_data):
    true_positives = 0
    true_negatives = 0
    false_positives = 0
    false_negatives = 0
```

```
    for sample in test_data:
        prediction = predict(tree, sample)
        actual = sample[-1]  # Assuming the last column contains the actual␣
↪class labels

        if prediction == actual:
            if actual == 1:
                true_positives += 1
            else:
                true_negatives += 1
        else:
            if actual == 1:
                false_negatives += 1
            else:
                false_positives += 1

    accuracy = (true_positives + true_negatives) / len(test_data)
    precision = true_positives / (true_positives + false_positives) if␣
↪(true_positives + false_positives) != 0 else 0
    recall = true_positives / (true_positives + false_negatives) if␣
↪(true_positives + false_negatives) != 0 else 0

    return accuracy, precision, recall

def calculate_macro_metrics(tree, test_data):
    num_classes = len(np.unique(test_data[:, -1]))  # Assuming class labels are␣
↪in the last column
    class_accuracies = np.zeros(num_classes)
    class_precisions = np.zeros(num_classes)
    class_recalls = np.zeros(num_classes)

    for class_label in range(num_classes):
        # Filter test data for the current class
        class_data = test_data[test_data[:, -1] == class_label]

        if len(class_data) > 0:
            accuracy, precision, recall = evaluate_decision_tree(tree,␣
↪class_data)
            class_accuracies[class_label] = accuracy
            class_precisions[class_label] = precision
            class_recalls[class_label] = recall

    macro_accuracy = np.mean(class_accuracies)
    macro_precision = np.mean(class_precisions)
    macro_recall = np.mean(class_recalls)

    return macro_accuracy, macro_precision, macro_recall
```

[145]:
```
def build_decision_tree(data, depth=0, max_depth=10, min_samples_split=10):
    # Stopping criteria
    if depth >= max_depth or len(data) < min_samples_split:
        label = get_most_common_label(data)
        return Node(label,None, depth)

    if all_same_class(data):
        label = data[0, -1]
        return Node(label, None, depth)

    # Find the best feature to split on
    best_feature, threshold = find_most_info_feat(data)
    node = Node(None, data, depth)
    node.feature_name = best_feature
    node.threshold = threshold

    # Split the data and build child nodes
    left_data = data[data[:, features.index(best_feature)] <= threshold]
    right_data = data[data[:, features.index(best_feature)] > threshold]

    node.children.append(build_decision_tree(left_data, depth + 1, max_depth,␣
 ↪min_samples_split))
    node.children.append(build_decision_tree(right_data, depth + 1, max_depth,␣
 ↪min_samples_split))

    return node

def prune_decision_tree(tree, validation_data):
    if tree is None:
        return None

    # Prune children first
    for i, child in enumerate(tree.children):
        tree.children[i] = prune_decision_tree(child, validation_data)

    # Calculate accuracy on the validation data without this subtree
    tree_accuracy, tree_precision, tree_recall = evaluate_decision_tree(tree,␣
 ↪validation_data)

    # Calculate accuracy on the validation data with this subtree
    pruned_tree = tree.copy()  # Create a copy of the tree
    pruned_tree.children = []  # Remove children to prune the subtree

    pruned_tree_accuracy, pruned_tree_precision, pruned_tree_recall =␣
 ↪evaluate_decision_tree(pruned_tree, validation_data)
```

```
    # If pruning improves accuracy or doesn't degrade it significantly, prune␣
↪the subtree
    if pruned_tree_accuracy >= tree_accuracy:
        return pruned_tree
    else:
        return tree
```

[146]:
```
tree = build_decision_tree(train_df.values)
print_decision_tree(tree)
```

```
Node: Split on Glucose <= 144.5
  Node: Split on BMI <= 26.35
    Node: Split on Age <= 51.0
      Leaf: Class 0.0
      Leaf: Class 0.0
    Node: Split on Pregnancies <= 5.5
      Node: Split on Age <= 22.5
        Node: Split on SkinThickness <= 36.0
          Leaf: Class 0.0
          Leaf: Class 0.0
        Node: Split on Glucose <= 124.5
          Node: Split on Age <= 39.5
            Node: Split on BMI <= 32.2
              Leaf: Class 0.0
              Node: Split on DiabetesPedigreeFunction <= 0.5105
                Node: Split on BMI <= 45.4
                  Node: Split on Pregnancies <= 0.5
                    Leaf: Class 0.0
                    Leaf: Class 0.0
                  Leaf: Class 1.0
                Node: Split on Insulin <= 68.0
                  Leaf: Class 0.0
                  Leaf: Class 1.0
            Node: Split on SkinThickness <= 25.5
              Node: Split on BloodPressure <= 89.0
                Leaf: Class 1.0
                Leaf: Class 0.0
              Leaf: Class 0.0
          Node: Split on Insulin <= 199.0
            Node: Split on DiabetesPedigreeFunction <= 0.3115
              Node: Split on BloodPressure <= 66.0
                Leaf: Class 0.0
                Leaf: Class 0.0
              Node: Split on BMI <= 28.75
                Leaf: Class 0.0
                Node: Split on BloodPressure <= 74.0
                  Leaf: Class 1.0
                  Node: Split on BMI <= 31.4
```

```
                Leaf: Class 0.0
                Leaf: Class 1.0
          Leaf: Class 0.0
    Node: Split on DiabetesPedigreeFunction <= 0.759
      Node: Split on Glucose <= 99.0
        Node: Split on Pregnancies <= 11.0
          Node: Split on Glucose <= 28.5
            Leaf: Class 1.0
            Leaf: Class 0.0
          Leaf: Class 1.0
        Node: Split on BloodPressure <= 87.0
          Node: Split on Pregnancies <= 9.5
            Node: Split on Age <= 33.0
              Leaf: Class 1.0
              Node: Split on Glucose <= 104.5
                Leaf: Class 1.0
                Node: Split on Glucose <= 111.0
                  Leaf: Class 0.0
                  Leaf: Class 1.0
            Node: Split on Insulin <= 126.0
              Node: Split on Age <= 41.5
                Leaf: Class 0.0
                Leaf: Class 0.0
              Leaf: Class 1.0
          Leaf: Class 0.0
      Leaf: Class 1.0
Node: Split on Glucose <= 159.5
  Node: Split on BloodPressure <= 85.5
    Node: Split on Age <= 24.5
      Leaf: Class 0.0
      Node: Split on BMI <= 39.05
        Node: Split on Pregnancies <= 4.5
          Node: Split on BMI <= 23.8
            Leaf: Class 0.0
            Node: Split on BMI <= 33.9
              Leaf: Class 1.0
              Leaf: Class 1.0
          Node: Split on DiabetesPedigreeFunction <= 0.866
            Node: Split on Glucose <= 149.0
              Leaf: Class 0.0
              Leaf: Class 0.0
            Leaf: Class 1.0
        Leaf: Class 0.0
    Node: Split on BMI <= 30.0
      Leaf: Class 1.0
      Leaf: Class 1.0
  Node: Split on Age <= 57.5
    Node: Split on Pregnancies <= 6.5
```

```
        Node: Split on Pregnancies <= 5.5
          Node: Split on BMI <= 37.2
            Node: Split on DiabetesPedigreeFunction <= 0.2695
              Leaf: Class 1.0
              Node: Split on SkinThickness <= 37.5
                Node: Split on Age <= 27.5
                  Leaf: Class 0.0
                  Leaf: Class 1.0
                Leaf: Class 0.0
            Leaf: Class 1.0
          Leaf: Class 0.0
        Leaf: Class 1.0
      Leaf: Class 0.0
```

[147]:
```
test_accuracy, test_precision, test_recall =␣
 ↪calculate_macro_metrics(tree,test_df.values)
print("\nBefore Pruning:")
print(f'Mean Macro Accuracy: {test_accuracy}')
print(f'Mean Macro Precision: {test_precision}')
print(f'Mean Macro Recall: {test_recall}\n\n')
```

```
Before Pruning:
Mean Macro Accuracy: 0.6288135593220339
Mean Macro Precision: 0.5
Mean Macro Recall: 0.2288135593220339
```

[148]:
```
pruned_tree = prune_decision_tree(tree,val_df.values)
print_decision_tree(pruned_tree)
```

```
Node: Split on Glucose <= 144.5
  Node: Split on BMI <= 26.35
    Node: Split on Age <= 51.0
      Leaf: Class 0.0
      Leaf: Class 0.0
    Node: Split on Pregnancies <= 5.5
      Node: Split on Age <= 22.5
        Node: Split on SkinThickness <= 36.0
          Leaf: Class 0.0
          Leaf: Class 0.0
        Node: Split on Glucose <= 124.5
          Node: Split on Age <= 39.5
            Node: Split on BMI <= 32.2
              Leaf: Class 0.0
              Node: Split on DiabetesPedigreeFunction <= 0.5105
                Node: Split on BMI <= 45.4
                  Node: Split on Pregnancies <= 0.5
```

```
                    Leaf: Class 0.0
                    Leaf: Class 0.0
                  Leaf: Class 1.0
                Node: Split on Insulin <= 68.0
                  Leaf: Class 0.0
                  Leaf: Class 1.0
            Node: Split on SkinThickness <= 25.5
              Node: Split on BloodPressure <= 89.0
                Leaf: Class 1.0
                Leaf: Class 0.0
              Leaf: Class 0.0
          Node: Split on Insulin <= 199.0
            Node: Split on DiabetesPedigreeFunction <= 0.3115
              Node: Split on BloodPressure <= 66.0
                Leaf: Class 0.0
                Leaf: Class 0.0
              Node: Split on BMI <= 28.75
                Leaf: Class 0.0
                Node: Split on BloodPressure <= 74.0
                  Leaf: Class 1.0
                  Node: Split on BMI <= 31.4
                    Leaf: Class 0.0
                    Leaf: Class 1.0
            Leaf: Class 0.0
      Node: Split on DiabetesPedigreeFunction <= 0.759
        Node: Split on Glucose <= 99.0
          Node: Split on Pregnancies <= 11.0
            Node: Split on Glucose <= 28.5
              Leaf: Class 1.0
              Leaf: Class 0.0
            Leaf: Class 1.0
          Node: Split on BloodPressure <= 87.0
            Node: Split on Pregnancies <= 9.5
              Node: Split on Age <= 33.0
                Leaf: Class 1.0
                Node: Split on Glucose <= 104.5
                  Leaf: Class 1.0
                  Node: Split on Glucose <= 111.0
                    Leaf: Class 0.0
                    Leaf: Class 1.0
              Node: Split on Insulin <= 126.0
                Node: Split on Age <= 41.5
                  Leaf: Class 0.0
                  Leaf: Class 0.0
                Leaf: Class 1.0
            Leaf: Class 0.0
        Leaf: Class 1.0
  Node: Split on Glucose <= 159.5
```

```
Node: Split on BloodPressure <= 85.5
  Node: Split on Age <= 24.5
    Leaf: Class 0.0
    Node: Split on BMI <= 39.05
      Node: Split on Pregnancies <= 4.5
        Node: Split on BMI <= 23.8
          Leaf: Class 0.0
          Node: Split on BMI <= 33.9
            Leaf: Class 1.0
            Leaf: Class 1.0
        Node: Split on DiabetesPedigreeFunction <= 0.866
          Node: Split on Glucose <= 149.0
            Leaf: Class 0.0
            Leaf: Class 0.0
          Leaf: Class 1.0
      Leaf: Class 0.0
  Node: Split on BMI <= 30.0
    Leaf: Class 1.0
    Leaf: Class 1.0
Node: Split on Age <= 57.5
  Node: Split on Pregnancies <= 6.5
    Node: Split on Pregnancies <= 5.5
      Node: Split on BMI <= 37.2
        Node: Split on DiabetesPedigreeFunction <= 0.2695
          Leaf: Class 1.0
          Node: Split on SkinThickness <= 37.5
            Node: Split on Age <= 27.5
              Leaf: Class 0.0
              Leaf: Class 1.0
            Leaf: Class 0.0
        Leaf: Class 1.0
      Leaf: Class 0.0
    Leaf: Class 1.0
  Leaf: Class 0.0
```

[149]:
```
test_accuracy, test_precision, test_recall =␣
 ↪calculate_macro_metrics(pruned_tree,test_df.values)
print("\nAfter Pruning:")
print(f'Mean Macro Accuracy: {test_accuracy}')
print(f'Mean Macro Precision: {test_precision}')
print(f'Mean Macro Recall: {test_recall}')
```

```
After Pruning:
Mean Macro Accuracy: 0.6288135593220339
Mean Macro Precision: 0.5
Mean Macro Recall: 0.2288135593220339
```