ayman-boufarhi-cart

November 26, 2023

1 Regression

1.1 *Approximation of the "tip" column in the Tips example dataset using the CART decision tree model.*

Loading libraries

[1]:
```
import pandas as pd
import numpy as np
```

Loading data

[2]:
```
import seaborn as sns
dataSet = sns.load_dataset("tips")
print("Examples from the dataset:")
print(dataSet.head())  # Display the first few rows of the dataset
```

```
Examples from the dataset:
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

Definition of a class to represent a tree node

[3]:
```
class Node:
    def __init__(self, feature=None, threshold=None, value=None, left=None,␣
 ↪right=None):
        self.feature = feature          # Splitting feature
        self.threshold = threshold      # Threshold value for splitting
        self.value = value              # Prediction value for a leaf
        self.left = left                # Left subtree
        self.right = right              # Right subtree
```

Function to calculate the Gini index

[4]:
```
def calculate_gini(y):
    classes = np.unique(y)
```

```
    gini = 1.0
    for cls in classes:
        p = np.sum(y == cls) / len(y)
        gini -= p ** 2
    return gini
```

Function to split data based on a feature and a threshold value

```
[5]: def split_data(X, y, feature, threshold):
    left_mask = X.iloc[:, feature] <= threshold
    right_mask = ~left_mask
    return X[left_mask], y[left_mask], X[right_mask], y[right_mask]
```

Function to find the best split

```
[6]: def find_best_split(X, y):
    num_features = X.shape[1]
    best_gini = float('inf')
    best_feature = None
    best_threshold = None

    for feature in range(num_features):
        values = np.unique(X.iloc[:, feature])
        for threshold in values:
            X_left, y_left, X_right, y_right = split_data(X, y, feature,␣
 ↪threshold)
            gini_left = calculate_gini(y_left)
            gini_right = calculate_gini(y_right)
            gini = (len(y_left) * gini_left + len(y_right) * gini_right) /␣
 ↪len(y)

            if gini < best_gini:
                best_gini = gini
                best_feature = feature
                best_threshold = threshold

    return best_feature, best_threshold
```

Recursive function to build the tree

```
[7]: def build_tree(X, y, depth=0, max_depth=None):
    if depth == max_depth or len(np.unique(y)) == 1:
        # Create a leaf
        return Node(value=np.mean(y))

    feature, threshold = find_best_split(X, y)
    if feature is not None:
        # Split the data based on the best feature and threshold
```

```
        X_left, y_left, X_right, y_right = split_data(X, y, feature, threshold)

        # Build the subtrees recursively
        left_subtree = build_tree(X_left, y_left, depth + 1, max_depth)
        right_subtree = build_tree(X_right, y_right, depth + 1, max_depth)

        # Return the current node
        return Node(feature=feature, threshold=threshold, left=left_subtree,␣
↪right=right_subtree)
    else:
        # No split possible, create a leaf
        return Node(value=np.mean(y))
```

Function to predict a single observation

[8]:
```
def predict_one(tree, x):
    if tree.value is not None:
        return tree.value
    elif x.iloc[tree.feature] <= tree.threshold:
        return predict_one(tree.left, x)
    else:
        return predict_one(tree.right, x)
```

Function to predict a set of observations

[9]:
```
def predict(tree, X):
    return [predict_one(tree, x) for _, x in X.iterrows()]
```

Check the type of each column and convert categorical columns

[10]:
```
for col in dataSet.columns:
    if dataSet[col].dtype.name == 'category':
        dataSet[col] = dataSet[col].cat.codes
```

Split the data into features (X) and target variable (y)

[11]:
```
X = dataSet.drop("tip", axis=1)
y = dataSet["tip"]

print("Examples of features (X) and target variable (y):")
print("Features (X):")
print(X.head())
print("Target variable (y):")
print(y.head())
```

```
Examples of features (X) and target variable (y):
Features (X):
   total_bill  sex  smoker  day  time  size
```

```
0       16.99    1       1    3     1     2
1       10.34    0       1    3     1     3
2       21.01    0       1    3     1     3
3       23.68    0       1    3     1     2
4       24.59    1       1    3     1     4
Target variable (y):
0    1.01
1    1.66
2    3.50
3    3.31
4    3.61
Name: tip, dtype: float64
```

Build the decision tree

[12]:
```python
tree = build_tree(X, y, max_depth=3)
print("Decision tree built successfully.")
```

```
Decision tree built successfully.
```

Predict values

[13]:
```python
predictions = predict(tree, X)
print("Predictions for examples from the dataset:")
print(predictions)
```

```
Predictions for examples from the dataset:
[2.8560919540229883, 1.8607999999999998, 2.8560919540229883, 4.035070422535212,
4.035070422535212, 4.035070422535212, 1.8607999999999998, 4.035070422535212,
2.8560919540229883, 2.8560919540229883, 1.8607999999999998, 4.035070422535212,
2.8560919540229883, 2.8560919540229883, 2.8560919540229883, 4.035070422535212,
1.8607999999999998, 2.8560919540229883, 2.8560919540229883, 2.8560919540229883,
2.8560919540229883, 2.8560919540229883, 2.8560919540229883, 4.035070422535212,
2.8560919540229883, 2.8560919540229883, 2.364347826086956, 1.8607999999999998,
4.035070422535212, 2.8560919540229883, 1.8607999999999998, 2.8560919540229883,
2.8560919540229883, 2.8560919540229883, 2.8560919540229883, 4.035070422535212,
2.8560919540229883, 2.8560919540229883, 2.8560919540229883, 4.035070422535212,
2.8560919540229883, 2.8560919540229883, 2.364347826086956, 1.8607999999999998,
4.035070422535212, 2.8560919540229883, 4.035070422535212, 4.035070422535212,
4.035070422535212, 2.8560919540229883, 1.8607999999999998, 1.8607999999999998,
4.035070422535212, 1.8607999999999998, 4.035070422535212, 2.8560919540229883,
4.035070422535212, 4.035070422535212, 1.8607999999999998, 4.035070422535212,
2.8560919540229883, 2.364347826086956, 1.8607999999999998, 2.8560919540229883,
2.8560919540229883, 2.8560919540229883, 2.8560919540229883, 1.0,
2.8560919540229883, 2.8560919540229883, 1.8607999999999998, 2.8560919540229883,
4.035070422535212, 4.035070422535212, 2.8560919540229883, 1.8607999999999998,
2.8560919540229883, 4.035070422535212, 4.035070422535212, 2.8560919540229883,
2.8560919540229883, 2.8560919540229883, 1.8607999999999998, 4.035070422535212,
2.8560919540229883, 4.035070422535212, 2.364347826086956, 2.8560919540229883,
```

```
4.035070422535212, 2.8560919540229883, 4.035070422535212, 4.035070422535212,
1.0, 2.8560919540229883, 4.035070422535212, 4.035070422535212,
4.035070422535212, 1.8607999999999998, 2.8560919540229883, 1.8607999999999998,
1.8607999999999998, 2.8560919540229883, 4.035070422535212, 4.035070422535212,
2.8560919540229883, 2.8560919540229883, 2.8560919540229883, 4.035070422535212,
2.8560919540229883, 2.364347826086956, 2.364347826086956, 1.0,
4.035070422535212, 4.035070422535212, 4.035070422535212, 2.8560919540229883,
4.035070422535212, 1.8607999999999998, 1.8607999999999998, 4.035070422535212,
1.8607999999999998, 2.364347826086956, 2.364347826086956, 2.8560919540229883,
1.8607999999999998, 5.148571428571429, 1.8607999999999998, 2.364347826086956,
1.8607999999999998, 4.035070422535212, 2.8560919540229883, 2.8560919540229883,
1.8607999999999998, 1.8607999999999998, 2.8560919540229883, 1.8607999999999998,
1.8607999999999998, 2.364347826086956, 2.8560919540229883, 2.364347826086956,
2.8560919540229883, 5.148571428571429, 5.148571428571429, 5.148571428571429,
2.8560919540229883, 1.8607999999999998, 2.8560919540229883, 1.8607999999999998,
1.8607999999999998, 1.8607999999999998, 2.364347826086956, 2.364347826086956,
2.8560919540229883, 4.035070422535212, 2.8560919540229883, 5.148571428571429,
5.148571428571429, 4.035070422535212, 2.364347826086956, 2.8560919540229883,
2.8560919540229883, 1.8607999999999998, 2.8560919540229883, 2.364347826086956,
2.8560919540229883, 4.035070422535212, 2.8560919540229883, 4.035070422535212,
1.8607999999999998, 1.8607999999999998, 4.035070422535212, 2.8560919540229883,
5.15, 4.035070422535212, 2.8560919540229883, 4.035070422535212,
2.8560919540229883, 2.364347826086956, 1.8607999999999998, 4.035070422535212,
4.035070422535212, 4.035070422535212, 4.035070422535212, 4.035070422535212,
4.035070422535212, 5.148571428571429, 2.8560919540229883, 2.5,
2.8560919540229883, 4.035070422535212, 2.8560919540229883, 2.8560919540229883,
4.035070422535212, 2.8560919540229883, 2.8560919540229883, 1.8607999999999998,
1.8607999999999998, 4.035070422535212, 2.364347826086956, 2.364347826086956,
2.8560919540229883, 1.8607999999999998, 2.364347826086956, 2.8560919540229883,
2.8560919540229883, 2.8560919540229883, 4.035070422535212, 4.035070422535212,
4.035070422535212, 1.8607999999999998, 4.035070422535212, 4.035070422535212,
4.035070422535212, 2.364347826086956, 4.035070422535212, 1.8607999999999998,
2.5, 1.8607999999999998, 1.8607999999999998, 4.035070422535212,
1.8607999999999998, 2.364347826086956, 1.8607999999999998, 2.8560919540229883,
2.364347826086956, 2.8560919540229883, 1.8607999999999998, 2.8560919540229883,
2.364347826086956, 4.035070422535212, 4.035070422535212, 2.8560919540229883,
1.8607999999999998, 1.8607999999999998, 2.8560919540229883, 1.8607999999999998,
1.8607999999999998, 4.035070422535212, 4.035070422535212, 4.035070422535212,
4.035070422535212, 4.035070422535212, 2.8560919540229883, 2.8560919540229883]
```

Create a new observation and make a prediction

[14]:
```
print("Example of a new observation:")
new_observation = pd.DataFrame({
    "total_bill": 38.07,
    "sex": "Male",
    "smoker": "No",
    "day": "Sun",
```

```
    "time": "Dinner",
    "size": 3
}, index=[0])
print(new_observation)
```

```
Example of a new observation:
   total_bill   sex smoker  day    time  size
0       38.07  Male     No  Sun  Dinner     3
```

Convert categorical values to numeric

[15]:
```
for col in new_observation.columns:
    if new_observation[col].dtype.name == 'category':
        new_observation[col] = new_observation[col].cat.codes
```

Make the prediction

[16]:
```
prediction = predict_one(tree, new_observation.iloc[0])
print("Predicted Tip:", prediction)
```

```
Predicted Tip: 4.035070422535212
```