Week_9 Descision Trees

October 16, 2024

Q1,2,3. Write a python function program to demonstrate the working of the decision tree based C4.5 and CART algorithms without using scikit-learn library. Use following data set for building the decision tree and apply this knowledge to classify a new sample.

[3]:
```
import pandas as pd
import numpy as np

class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None,␣
 ↪prediction=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.prediction = prediction

class C4_5_algorithm:
    def __init__(self):
        self.root = None

    def fit(self, dataset):
        self.root = self.create_tree(dataset)

    def calculate_entropy(self, labels):
        value_counts = labels.value_counts(normalize=True)
        return -np.sum(value_counts * np.log2(value_counts + 1e-9))

    def compute_gain(self, dataset, split_feature, target):
        initial_entropy = self.calculate_entropy(dataset[target])
        value_counts = dataset[split_feature].value_counts(normalize=True)

        weighted_entropy = sum(value_counts[v] * self.
 ↪calculate_entropy(dataset[dataset[split_feature] == v][target])
                               for v in value_counts.index)

        return initial_entropy - weighted_entropy

    def find_best_split(self, dataset, target):
```

```
        max_gain = -1
        best_feature = None

        for feature in dataset.columns[:-1]:
            gain = self.compute_gain(dataset, feature, target)
            if gain > max_gain:
                max_gain = gain
                best_feature = feature

        return best_feature

    def create_tree(self, dataset):
        target = dataset.columns[-1]
        labels = dataset[target]

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

        if len(dataset.columns) == 1:
            return Node(prediction=labels.mode()[0])

        best_feature = self.find_best_split(dataset, target)

        tree_node = Node(feature=best_feature)

        for threshold in dataset[best_feature].unique():
            subset = dataset[dataset[best_feature] == threshold]
            child_node = self.create_tree(subset.drop(columns=[best_feature]))
            if tree_node.left is None:
                tree_node.left = child_node
                tree_node.threshold = threshold
            else:
                tree_node.right = child_node

        return tree_node

    def classify_instance(self, node, instance):
        if node.prediction is not None:
            return node.prediction

        feature_value = instance[node.feature]

        if feature_value == node.threshold:
            return self.classify_instance(node.left, instance) if node.left␣
↪else node.prediction
        else:
```

```
            return self.classify_instance(node.right, instance) if node.right␣
 ↪else node.prediction

    def predict(self, instance):
        return self.classify_instance(self.root, instance)

class CART_algorithm:
    def __init__(self):
        self.root = None

    def fit(self, dataset):
        self.root = self.construct_tree(dataset)

    def calculate_gini(self, outcomes):
        total_count = len(outcomes)
        if total_count == 0:
            return 0
        proportions = outcomes.value_counts(normalize=True)
        return 1 - sum(proportions ** 2)

    def compute_gain(self, dataset, split_feature, outcome_col):
        initial_gini = self.calculate_gini(dataset[outcome_col])
        weighted_gini = 0

        for threshold in dataset[split_feature].unique():
            subset = dataset[dataset[split_feature] == threshold]
            weighted_gini += (len(subset) / len(dataset)) * self.
 ↪calculate_gini(subset[outcome_col])

        return initial_gini - weighted_gini

    def find_best_split(self, dataset, outcome_col):
        max_gain = -1
        best_feature = None

        for feature in dataset.columns[:-1]:
            gain = self.compute_gain(dataset, feature, outcome_col)
            if gain > max_gain:
                max_gain = gain
                best_feature = feature

        return best_feature

    def construct_tree(self, dataset):
        outcome_col = dataset.columns[-1]
        outcomes = dataset[outcome_col]
```

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

        if len(dataset.columns) == 1:
            return Node(prediction=outcomes.mode()[0])

        best_feature = self.find_best_split(dataset, outcome_col)
        node = Node(feature=best_feature)

        for threshold in dataset[best_feature].unique():
            subset = dataset[dataset[best_feature] == threshold]
            child_node = self.construct_tree(subset.
 ↪drop(columns=[best_feature]))
            if node.left is None:
                node.left = child_node
                node.threshold = threshold
            else:
                node.right = child_node

        return node

    def classify_sample(self, node, sample):
        if node.prediction is not None:
            return node.prediction

        feature_value = sample[node.feature]

        if feature_value == node.threshold:
            return self.classify_sample(node.left, sample) if node.left else␣
 ↪node.prediction
        else:
            return self.classify_sample(node.right, sample) if node.right else␣
 ↪node.prediction

    def predict(self, sample):
        return self.classify_sample(self.root, sample)

df = pd.read_csv("weather.csv")

C4_5_tree = C4_5_algorithm()
C4_5_tree.fit(df)

CART_tree = CART_algorithm()
CART_tree.fit(df)

new_instance = {
    'Weather': 'Sunny',
```

```
    'Temperature': 75,
    'Humidity': 70,
    'Breeze': 'Weak'
}
print("For the C4.5 algorithm: ")
new_instance_df = pd.DataFrame([new_instance])
print(f"The new data is:- {new_instance_df} ")

prediction = C4_5_tree.predict(new_instance_df.iloc[0])
print(f"The predicted decision for the new instance is: {prediction}")

print("\n")
print("For the CART algorithm: ")
new_instance_df = pd.DataFrame([new_instance])
print(f"The new data is:- {new_instance_df} ")

prediction = CART_tree.predict(new_instance_df.iloc[0])
print(f"The predicted decision for the new instance is: {prediction}")

print("**************************************************")


df = pd.read_csv("loan.csv")

C4_5_tree = C4_5_algorithm()
C4_5_tree.fit(df)

CART_tree = CART_algorithm()
CART_tree.fit(df)

new_instance = {
    'Income': 100,
    'Credit': 1000000
}
print("For the C4.5 algorithm: ")
new_instance_df = pd.DataFrame([new_instance])
print(f"The new data is:- {new_instance_df} ")

prediction = C4_5_tree.predict(new_instance_df.iloc[0])
print(f"The predicted decision for the new instance is: {prediction}")

print("\n")
print("For the CART algorithm: ")
new_instance_df = pd.DataFrame([new_instance])
print(f"The new data is:- {new_instance_df} ")

prediction = CART_tree.predict(new_instance_df.iloc[0])
```

```
print(f"The predicted decision for the new instance is: {prediction}")
```

```
For the C4.5 algorithm:
The new data is:-   Weather  Temperature  Humidity Breeze
0   Sunny           75         70   Weak
The predicted decision for the new instance is: No


For the CART algorithm:
The new data is:-   Weather  Temperature  Humidity Breeze
0   Sunny           75         70   Weak
The predicted decision for the new instance is: No
For the C4.5 algorithm:
The new data is:-    Income   Credit
0     100  1000000
The predicted decision for the new instance is: Yes


For the CART algorithm:
The new data is:-    Income   Credit
0     100  1000000
The predicted decision for the new instance is: Yes
```

[8]:

```
The predicted decision for the new sample is: No
```

[ ]: