own_decision_tree_classifier

February 14, 2024

1 Decision Tree classifier

[52]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[53]:
```
# scikit-learn package
from sklearn.datasets import load_iris, load_breast_cancer, load_wine
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split
```

1.0.1 Own DecisionTreeClassifier implementation

[54]:
```
class Node:

    def __init__(self, X, y, gini):
        self.X = X
        self.y = y
        self.gini = gini
        self.feature_index = 0
        self.threshold = 0
        self.left = None
        self.right = None
```

[55]:
```
# Implement a decision tree classifier
class MyDecisionTreeClassifier:

    def __init__(self, max_depth):
        self.__classes = None
        self.__features = None
        self.__root_node = None
        self.max_depth = max_depth

    @staticmethod
    def gini(groups, classes):
        '''
```

```
        A Gini score gives an idea of how good a split is by how mixed the
        classes are in the two groups created by the split.

        A perfect separation results in a Gini score of 0,
        whereas the worst case split that results in 50/50
        classes in each group result in a Gini score of 0.5
        (for a 2 class problem).
        '''
        total_size = sum(groups)
        return 1 - sum(
                    ( (groups[x] / total_size) ** 2 for x in classes )
                )


    def split_data(self, X, y) -> tuple[int, int]:

        # test all the possible splits in O(N*F) where N in number of samples
        # and F is number of features

        # return index and threshold value

        output_size = y.size
        if output_size <= 1:
            return None, None

        class_counts = [np.sum(y == c) for c in self.__classes]

        best_gini = 1 - sum( (n / output_size) ** 2  for n in class_counts )
        best_feature, best_threshold = None, None

        for feature in self.__features:

            thresholds, classes = zip(*sorted(zip(X[:, feature], y)))

            left_classes = [0] * len(self.__classes)
            right_classes = class_counts.copy()

            for i in range(output_size):
                c = classes[i-1]

                left_classes[c] += 1
                right_classes[c] -= 1

                left_gini = self.gini(left_classes, self.__classes)
                right_gini = self.gini(right_classes, self.__classes)
```

```
                gini = (i * left_gini + (output_size - i) * right_gini) / output_size

                if thresholds[i] == thresholds[i - 1]:
                    continue

                if gini < best_gini:
                    best_gini = gini
                    best_feature = feature
                    best_threshold = (thresholds[i] + thresholds[i-1]) / 2

        return best_feature, best_threshold

    def build_tree(self, X, y, depth = 0):
        number_of_samples_per_class = [np.sum(y == c) for c in self.__classes]
        predicted_class = np.argmax(number_of_samples_per_class)
        node = Node(
            gini=self.gini(number_of_samples_per_class, self.__classes),
            X=number_of_samples_per_class,
            y=predicted_class
        )

        if depth < self.max_depth:
            feature, threshold = self.split_data(X, y)
            if feature is not None:
                samples_idx_left = X[:,feature] < threshold
                X_left, y_left = X[samples_idx_left], y[samples_idx_left]
                X_right, y_right = X[~samples_idx_left], y[~samples_idx_left]
                node.feature_index = feature
                node.threshold = threshold
                node.left = self.build_tree(X_left, y_left, depth+1)
                node.right = self.build_tree(X_right, y_right, depth+1)

        return node

    def fit(self, X, y):
        self.__classes = set(y)
        self.__features = range(X.shape[1])
        self.__root_node = self.build_tree(X, y)


    def predict(self, X_test):
        node = self.__root_node
        while node.left:
            if X_test[node.feature_index] < node.threshold:
                node = node.left
            else:
```

```
            node = node.right

        return node.y

    def evaluate(self, X_test, y_test):
        correct_answers = sum( self.predict(x) == y_test[i] for i, x in␣
↪enumerate(X_test) )
        return correct_answers / len(y_test)
```

1.1 Testing

To test our implementation we will load **iris**, **breast_cancer** and **wine** datasets from sklearn and then compare performance of sklearn decision tree with ours.

[56]:
```
# Loading dataset
def load_dataset(load_function):
    dataset = load_function()
    X, y = dataset.data, dataset.target
    return train_test_split(X, y, test_size= 0.20)

def fit_sklearn_model(X, X_test, y, y_test):
    sklearn_classifier = DecisionTreeClassifier()
    sklearn_classifier = sklearn_classifier.fit(X, y)
    predictions = sklearn_classifier.predict(X_test)
    return sum(predictions == y_test) / len(y_test)

def fit_own_implementation(X, X_test, y, y_test):
    tree_classifier = MyDecisionTreeClassifier(6)
    tree_classifier.fit(X, y)
    return tree_classifier.evaluate(X_test, y_test)
```

1.1.1 Iris dataset

[57]:
```
dataset = load_dataset(load_iris)

print("Sklearn:", fit_sklearn_model(*dataset), "Own:",␣
↪fit_own_implementation(*dataset))
```

```
Sklearn: 0.9 Own: 0.9

/tmp/ipykernel_34716/2771890370.py:23: RuntimeWarning: invalid value encountered
in scalar divide
  ( (groups[x] / total_size) ** 2 for x in classes )
```

1.1.2 Breast Cancer dataset

[58]:
```
dataset = load_dataset(load_breast_cancer)

print("Sklearn:", fit_sklearn_model(*dataset), "Own:",␣
 ↪fit_own_implementation(*dataset))
```

```
/tmp/ipykernel_34716/2771890370.py:23: RuntimeWarning: invalid value encountered
in scalar divide
  ( (groups[x] / total_size) ** 2 for x in classes )
```

```
Sklearn: 0.9298245614035088 Own: 0.9385964912280702
```

1.1.3 Wine dataset

[59]:
```
dataset = load_dataset(load_wine)

print("Sklearn:", fit_sklearn_model(*dataset), "Own:",␣
 ↪fit_own_implementation(*dataset))
```

```
Sklearn: 0.9166666666666666 Own: 0.9166666666666666
```

```
/tmp/ipykernel_34716/2771890370.py:23: RuntimeWarning: invalid value encountered
in scalar divide
  ( (groups[x] / total_size) ** 2 for x in classes )
```

1.2 Conclusions

As we can see, our gives approximately the same performance as sklearn one.