Decision Tree - Lab

September 9, 2021

Modify the Decision Tree scratch code in our lecture such that: - Modify the scratch code so it can accept an hyperparameter max_depth, in which it will continue create the tree until max_depth is reached.

- Put everything into a class DecisionTree. It should have at least two methods, fit(), and predict()
- Load the iris data and try with your class

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
class Node:
    def __init__(self, gini, num_samples, num_samples_per_class,␣
 ↪predicted_class):
        self.gini = gini
        self.num_samples = num_samples
        self.num_samples_per_class = num_samples_per_class
        self.predicted_class = predicted_class
        self.feature_index = 0
        self.threshold = 0
        self.left = None
        self.right = None

class DecisionTree:
    def __init__(self, max_depth):
        self.max_depth = max_depth

    def find_split(self, X, y):
        """ Find split where children has lowest impurity possible
        in condition where the purity should also be less than the parent,
        if not, stop.
        """
        n_samples, n_features = X.shape
        if n_samples <= 1:
            return None, None

        #so it will not have any warning about "referenced before assignments"
        feature_ix, threshold = None, None
```

```
        # Count of each class in the current node.
        sample_per_class_parent = [np.sum(y == c) for c in range(self.
↪n_classes)] #[2, 2]

        # Gini of parent node.
        best_gini = 1.0 - sum((n / n_samples) ** 2 for n in␣
↪sample_per_class_parent)

        # Loop through all features.
        for feature in range(n_features):

            # Sort data along selected feature.
            sample_sorted = sorted(X[:, feature]) #[2, 3, 10, 19]
            sort_idx = np.argsort(X[:, feature])
            y_sorted = y[sort_idx] #[0, 0, 1, 1]

            sample_per_class_left = [0] * self.n_classes   #[0, 0]

            sample_per_class_right = sample_per_class_parent.copy() #[2, 2]

            #loop through each threshold, 2.5, 6.5, 14.5
            #1st iter: [-] [-++]
            #2nd iter: [--] [++]
            #3rd iter: [--+] [+]
            for i in range(1, n_samples): #1 to 3 (excluding 4)
                #the class of that sample
                c = y_sorted[i - 1]  #[0]

                #put the sample to the left
                sample_per_class_left[c] += 1  #[1, 0]

                #take the sample out from the right  [1, 2]
                sample_per_class_right[c] -= 1

                gini_left = 1.0 - sum(
                    (sample_per_class_left[x] / i) ** 2 for x in range(self.
↪n_classes)
                )

                #we divided by n_samples - i since we know that the left amount␣
↪of samples
                #since left side has already i samples
                gini_right = 1.0 - sum(
                    (sample_per_class_right[x] / (n_samples - i)) ** 2 for x in␣
↪range(self.n_classes)
                )
```

```
                #weighted gini
                weighted_gini = ((i / n_samples) * gini_left) + ( (n_samples -␣
↪i) /n_samples) * gini_right

                # in case the value are the same, we do not split
                # (both have to end up on the same side of a split).
                if sample_sorted[i] == sample_sorted[i - 1]:
                    continue

                if weighted_gini < best_gini:
                    best_gini = weighted_gini
                    feature_ix = feature
                    threshold = (sample_sorted[i] + sample_sorted[i - 1]) / 2 ␣
↪# midpoint

        #return the feature number and threshold
        #used to find best split
        return feature_ix, threshold

    def fit(self, X, y):
        self.n_classes = len(set(y))
        self.tree = self._fit(X,y)

    def _fit(self, X, y, depth=0):
        n_samples, n_features = X.shape

        num_samples_per_class = [np.sum(y == i) for i in range(self.n_classes)]
        #predicted class using the majority of sample class
        predicted_class = np.argmax(num_samples_per_class)

        #define the parent node
        node = Node(
            gini = 1 - sum((np.sum(y == c) / n_samples) ** 2 for c in␣
↪range(self.n_classes)),
            predicted_class=predicted_class,
            num_samples = y.size,
            num_samples_per_class = num_samples_per_class,
        )

        # task 1
        if depth < self.max_depth:
            #perform recursion
            feature, threshold = self.find_split(X, y)
            if feature is not None:
                #take all the indices that is less than threshold
                indices_left = X[:, feature] < threshold
```

```
                X_left, y_left = X[indices_left], y[indices_left]

                #tilde for negation
                X_right, y_right = X[~indices_left], y[~indices_left]

                #take note for later decision
                node.feature_index = feature
                node.threshold = threshold
                node.left = self._fit(X_left, y_left, depth + 1)
                node.right = self._fit(X_right, y_right, depth + 1)
        return node

    def predict(self, X_test):
        return [self._predict(x) for x in X_test]

    def _predict(self, sample):
        tree = self.tree
        while tree.left:
            if sample[tree.feature_index] < tree.threshold:
                tree = tree.left
            else:
                tree = tree.right
        return tree.predicted_class
```

[3]:
```
#fit starting with tree depth = 0
Xtrain = np.array([[2, 5],[3, 5],[10, 5],[19, 5]])
ytrain = np.array([0, 0, 1, 1])
Xtest = np.array(([[4, 6],[6, 9],[9, 2],[12, 8]]))
ytest = np.array([0, 0, 1, 1])

model = DecisionTree(max_depth=10)
model.fit(Xtrain, ytrain)
pred = model.predict(Xtest)

print("Tree feature ind: ", model.tree.feature_index)
print("Tree threshold: ", model.tree.threshold)
print("Pred: ", np.array(pred))
print("ytest: ", ytest)
```

```
Tree feature ind:  0
Tree threshold:  6.5
Pred:  [0 0 1 1]
ytest:  [0 0 1 1]
```

[4]:
```
from sklearn.datasets import load_iris
dataset = load_iris()
X, y = dataset.data, dataset.target
```

```
model = DecisionTree(max_depth=10)
model.fit(X, y)
pred = model.predict([[2, 2.9, 6, 1.3]])

print("Tree feature ind: ", model.tree.feature_index)
print("Tree threshold: ", model.tree.threshold)
print("Pred: ", np.array(pred))
```

```
Tree feature ind:  2
Tree threshold:  2.45
Pred:  [2]
```

[ ]: