Semester Project

Part 1 - Decision Tree

Author
Andrey Nosov

April 24, 2024

1 Introduction

This report aims to briefly describe a decision tree implementation in python that was created as part of the "Exploring Gradient Boosting" semester project and compare its performance to an existing scikit decision tree model.

2 Model architecture

In its current form the `CustomDecisionTreeClassifier` can be used only for classification problems. It can be initialized and used with the following integer type parameters:

- **max_depth** - the maximum depth of the tree
- **min_samples_split** - the minimum number of samples required to split an internal node.
- **min_samples_leaf** - the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches.

To measure the quality of each split the model uses weighted Gini impurity. First Gini impurity is calculated for the left and right nodes that were created after the split using the formula

$$G = 1 - \sum_{i=1}^{C} p(i)^2 \tag{1}$$

where C is the number of classes present in the data and $p(i)$ is the probability of picking an element of class i in the current node. After obtaining the impurity measurement for both nodes a weighted sum is performed to compute the final quality measurement of the split.

$$G_{split} = p_L * G_L + p_R * G_R \tag{2}$$

The weights p_L and p_R represent a fraction of elements that were split into the left and right nodes respectively.

To stop the splitting process from going infinitely the model uses the following stopping criteria:

- All the samples in the node belong to the same class.
- The current depth of the tree is equal to the maximum depth.
- The number of samples is less than the minimum number of samples required to split an internal node.
- No split yielded a new node with more samples than the minimum required.

3 Model comparison

To compare the performance of the `CustomDecisionTreeClassifier` and the scikit `DecisionTreeClassifier` the following operations were performed:

1. Load dataset
2. Split the data into train and test datasets with a 80:20 ratio
3. Perform parameter hyper-tuning of the three hyper-parameters (`max_depth`, `min_samples_leaf`, `min_samples_split`) for each model using K-Fold cross validation using the train dataset
4. Train both classifiers on the whole train dataset using found best hyper-parameters
5. Check the models performances on the test dataset

For a fair comparison the default Gini impurity split quality measurement was used in the scikit model and all parameters not present in the `CustomDecisionTreeClassifier` model were left at their default values.

3.1 Iris dataset

At first the two models were compared on the Iris dataset that is part of the scikit library. This dataset contains 150 samples with 4 features per sample and 3 classes in total. The structure of the dataset can be seen in figure 1.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Figure 1: Structure of the Iris dataset

After performing steps described above, we obtained the following results:

1. In both cases the same best hyper parameters were found - `max_depth = 3`, `min_samples_leaf = 1` and `min_samples_split = 2`
2. In both cases the same accuracy of `1.0` was obtained during the prediction of the test labels (corresponding confusion matrices in figure 2)

From these results we can say that the two models behaved identically both during the hyper-tuning and the test phases.

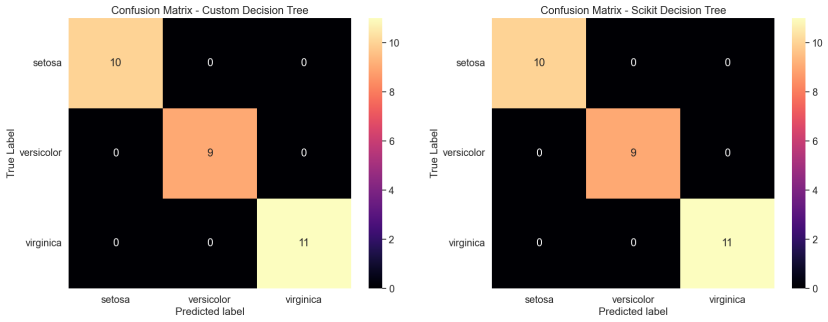

Figure 2: Comparison of confusion matrices for Iris dataset

3.2 Wine Quality Dataset

To see how the models perform on more complex data the Wine Quality dataset from the UCI Machine Learning repository was chosen. It contains 1599 samples with 11 features and 6 classes. The structure of the dataset can be seen in figure 3.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

Figure 3: Structure of the Wine Quality dataset

After again performing the same steps we obtained the following results:

1. Slightly different hyper-parameters were chosen for the two models. While `min_samples_leaf = 1` and `min_samples_split = 2` were the same for both, the scikit model showed better performance with `max_depth = 19` and the custom model performed better with `max_depth = 20`.
2. A very similar accuracy was obtained for the prediction of the test dataset - `0.56` for the scikit model and `0.59` for the custom model. Confusion matrices for both models can be seen in figure 4.

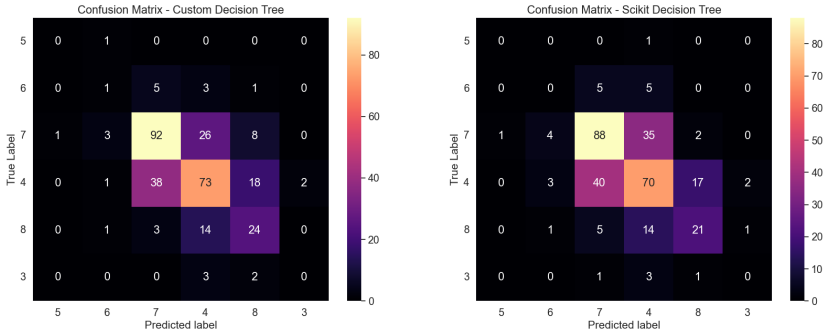

Figure 4: Comparison of confusion matrices for Wine Quality dataset

From these results we can tell that the two models behaved very similarly, but not identically. While performing the parameter hyper-tuning, the scikit model was much faster when creating high depth trees than the custom model, which suggests that some optimization methods are in place that can affect the result. This could explain the different behaviour. In the end the custom model performed even slightly better on the test dataset showing better accuracy.

4 Conclusion

In this part of the semester project a custom decision tree was successfully implemented from scratch in python and compared with the scikit model. It showed very similar and even slightly better performance in terms of accuracy, but lacks the speed of the scikit tree.