Classification trees and random forests implementation

Ivan Nikolov[1]

[1] *in7357@student.uni-lj.si, 63190378*

Classification trees

Algorithm implementation

Classification trees assume that we can interpret and partition the data generating process into homogenous regions where there is little or no uncertainty left about the target variable. To achieve this, we use the CART algorithm, which utilizes a greedy approach for binary partitioning of the data. At each step it chooses the best possible feature and threshold for splitting the data, based on some predefined criterion (in our case the Gini impurity measure).

Our main function for building the tree (`build`) takes the feature matrix `X` and a vector with the corresponding class variables `y` and builds the tree using recursion. First, we check if the data is already clean and no further partitioning can be done (only consists of target variables from one class) or if we are below the minimum sample size when a node is still split. If one of these conditions are met, we stop the tree building.

Otherwise, we continue and in the method `find_best_treshold_feature`, we iterate through the features and thresholds in order to find the combination that minimizes the Gini impurity measure. For a selected feature, we sort the threshold values and iterate through them, one by one. In each step, the number of instances in each class is updated. In case, there are multiple instances with the same threshold value, we skip the Gini calculation util all of them are updated in the counters. Then we calculate a Gini measure weighted with the proportion of elements in the sibling nodes, using the following formula

$$Gini(node) = \frac{|D_L|}{|D|} gini(D_L) + \frac{|D_R|}{|D|} gini(D_R)$$

. Where D denotes the training dataset and D_L and D_R the resulting left and right leaves.

To increase the model robustness to new data, the thresholds are set as the average values between two consecutive thresholds from the training dataset.

Results

For testing the performance of our implementation, we used the `tki-resistance.csv` dataset, where we used the first 130 rows for training and the remainder for evaluation.

From table 1 we can see that our model has a 0.0 misclassification rate on the training set, which suggests that it captures the complexity of the data well. On the testing set, the performance is similar to the sklearn implementation.

	Misclass. rate	Std. error
Our model (training set)	0.0	0.0
Our model (testing set)	0.19	0.05
Sklearn (training set)	0.0	0.0
Sklearn (testing set)	0.21	0.05

Table 1. Classification tree performance comparison

Estimation of standard errors was done using bootstrap. In a series of 1000 iterations, we sampled the model predictions and corresponding class values with replacement. In each iteration, we calculated the misclassification rate. At last, based on the attained distribution, we calculated the standard deviation and used that as an estimate for the standard error.

Random forest

Algorithm implementation

Random forests extend the idea of bagging and aim to build more decorrelated trees. The training part is composed of a for loop where we select a random bootstrap sample from the data to build the tree from. We also keep track of the out-of-bag indices because they will be needed later on for calculating feature importance. It is important to note that randomness is included at feature level too. For each node split, a random subset of $\sqrt{n}$ number of features is considered. Prediction is made by combining all votes from all trees and taking the most frequent one.

Feature importance is calculated by iterating through all the trees, and for each tree we iterate through all the features used for building that tree. First we calculate the misclassification rate (M) on the corresponding out-of-bag samples for that tree, then we randomly permute the values of the feature and recalculate the misclassification rate again (M_p). The feature importance is calculated as the $M_p - M$. These values are summed together for each feature and divided by the number of trees that have that feature.

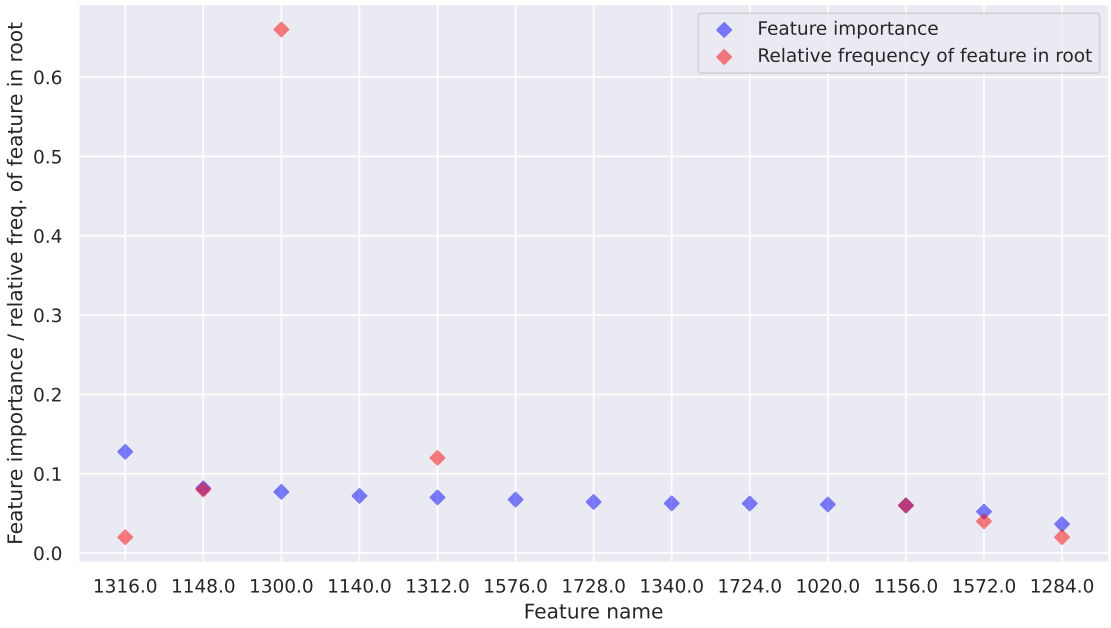

Figure 1. Variable importance and relative frequency of features in root node. Shown in blue are the top 10 features scores (first 10) together with the feature scores for the features in the root nodes. Shown in red are the relative frequencies for the features in the root nodes for 100 non-random trees.

Results

The evaluation procedure for random forests was the same as the one described in the classification trees section. On table 2, we can see that our implementation performs equally well as the one from sklearn.

	Misclass. rate	Std. error
Our model (training set)	0.0	0.0
Our model (testing set)	0.02	0.02
Sklearn (training set)	0.0	0.0
Sklearn (testing set)	0.02	0.02

Table 2. Random forest performance comparison

On Figure 2 we can see that the misclassification rate substantially drops until we reach $n = 50$. After that, it keeps switching between the values 0.02 and 0.03. This suggests, that a forest with $n = 50$ is complex enough to describe the data.

We also did an analysis on the feature importance and compared the feature importance values with the relative frequency of the features in the root nodes from 100 non-random trees. The 100 trees were build by bootstrapping the original dataset, however all trees used all the features and no additional randomness was present in the feature selection. The results are depicted in Figure 1, revealing that only 7 distinct features appear in the root nodes, with four of them ranking in the top 10 by importance score. Feature `1300.0` is most often used in the root nodes, and it's ranked third by the feature importance algorithm. The different order in feature importance versus features in root nodes can be attributed to the fact that random forest trees only use a subset of features at each node split. Feature importance scores also rely on out-of-bag samples, and their values may exhibit slight variations based on the selected random seed.

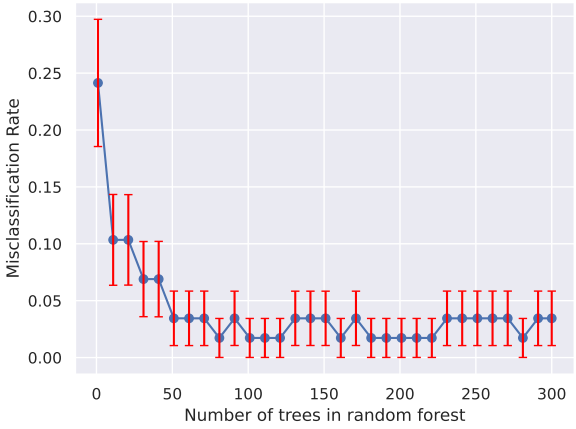

Figure 2. Misclassification rate versus the number of trees. Shown in red are the standard errors calculated using bootstap.