Machine Learning For Data Science I, Homework 01

Maj Gaberšček, 27212075

March 2023

1 Problem

This homework's task was implementation of decision trees and random forests in programming language `Python`.

Decision tree and Random forest needed to be implemented as classes. Classes were supposed to have methods for building decision trees or random forests on numerical data (atrributes) with binary target variables. They were also supposed to contain methods, that provide predicting with already trained decision tree or random forest. Random forest should also have a method importance, which calculates each attribute's importance, as described in [1].

2 Implementation

2.1 Overview

All code is provided in file `hw_tree.py`. Decision trees and Random forests are provided as classes with different methods. There are also some auxiliary functions, that solve different subproblems. Each function is well documented and has describtion of what it does. Towards the end of the script, we import the *Fourier-transform infrared spectroscopy* dataset and apply it to Decision trees and Random forests.

2.2 Decision tree

Decision tree skeleton is implemented as a class `Tree`. We can initialize it, by specifying different parameters, such as random generator or minimal number of samples (to continue with tree expansion). `Tree` also consists of method `build`, which fits a tree to training array of attribute values, `X` and target array `y`. This method then returns an object of type `TreeNode`, which represents a node in a tree (either a decision node or a leaf node). Tree is being fitted by an auxiliarry function `recursively_build`. We are building the tree recursively, by firstly choosing the attribute and value, over which we are going to breach the tree.

This is selected by calculating the weighted Gini impurity of targets in potential subtrees (selecting is done inside auxiliarry function `find_best_split`). We always split by the attribute and value combination, that provides smallest weighted Gini impurity.

The `TreeNode` class represents a node of a tree. The node can be of two types: a decision node or a leaf (also called terminal node). A **decision node** breaches the tree into two subtrees, left subtree and right subtree, which are also of type `TreeNode`. Decision node class also has two other attributes: `attribute`, which tells us, by which attribute we are splitting the tree and `value`, which is a value, by which we split. For example, if we are splitting by attribute 10 and value 25, left subtree in fitted on subdata, which has attribute 10 less or equal to 25, and the right subtree is fitted on subdata, which has attribute 10 greater then 25. If `TreeNode` represents a **leaf** of a tree, it only has one attribute, which is the prediction class (0 or 1 in our case). `TreeNode` also has a method `predict`, which is recursivelly written as well. It checks if `TreeNode` is a decision node or a leaf. If it is a leaf it returns prediction of this leaf. If it is a decision node, it checks the input attributes (values, which we want to predict the target on) and predicts on suitable subtree (depends on value and attribute of the tree).

2.3 Random forest

The skeleton for Random forests is in class `RandomForest`. This class contains a tree skeleton as an attribute, along with attributes for number of trees in the random forest and random generator attribute. It also has a `build` method, which fits random trees on data and creates a `RFModel` object, which represents a model forest, fitted on `X` and `y`.

`RFModel` represents a trained model and it has the following attributes: `trees`, which is a list of trees in the forest; `X`, `y` and `rand`, a random generator. There is a method `predict`, which iterates over a trees and predict the target in each tree. Then each tree votes, and the class with majority of votes is returned by the random tree. Second method in random forest model is `importance`. It calculates importance of each attribute of the dataset `X` and then returns array with importances. Importance is being calculated by iterating over all attributes. Then we permute values of this attribute and use the original training set (with permuted i-th attribute) to test tree predictions. The idea is, that if the attribute is very important and it contributes a lot towards the prediction, then the predictions with permuted attribute will be very different to the original ones. So we check missclassification percentage and if it is high, we conclude, that the parameter is important.

3 Results

First we built a full decision tree (minimum samples to still continue breaching is 2) on the *Fourier-transform infrared spectroscopy* dataset. Firstly we test our model on training data. As expected, model returned a 100% accuracy

(misclassification rate was 0, as well as the standard error). Then we tested our model on testing dataset (which was not used when fitting the tree). We got missclassification rate of 25,8% and standard error 0,06.

Random forest was built and fitted on the same dataset (*Fourier-transform infrared spectroscopy*). Forest consisted of 100 trees, which used random square root number of attributes for breaching. Again, as already expected: when testing on training data, we got misclassification rate and standard error 0. When testing on testing data, missclassification rate was 1,72%, which is a lot better then one full decision tree. Standard error was 0,02.

Lastly, we calculated parameter importance for random forest, built as discussed previously. Then we plotted the returned data, which can be seen in Figure 1. Higher importance means, that attribute contributes more to the final prediction. Obviously, attributes with higher importance, are more often in tree roots (and also more often choosen to split the tree).

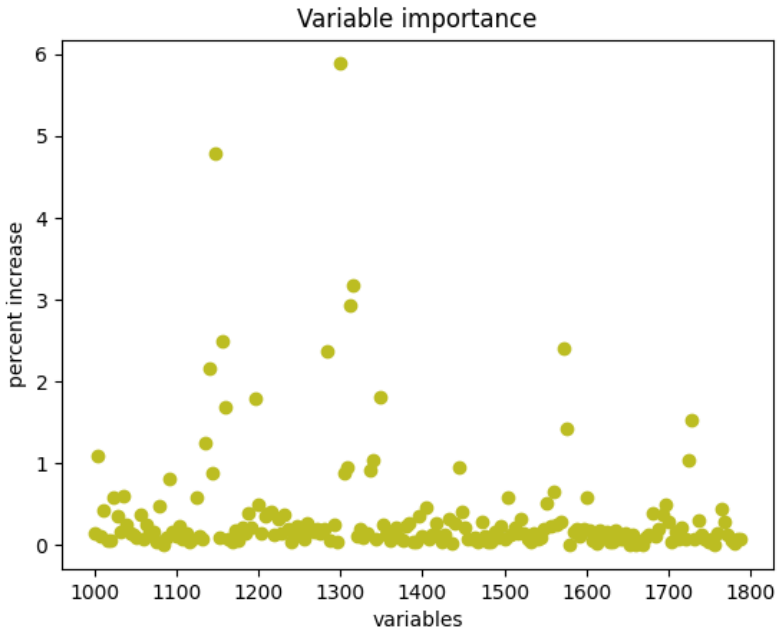

Figure 1: Importance of attributes in a random tree on Fourier-transform infrared spectroscopy data

References

[1] L Breiman. "Random Forests". In: *Machine Learning* 45 (Oct. 2001), pp. 5–32. DOI: 10.1023/A:1010950718922.