UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Practical Work 2: Decision Forests

Victor Badenas

Supervised and Experiene Learning
Master in Artificial Intelligence (FIB-MAI)

May 2021

Contents

Chapter 1

Introduction

1.1 Introduction

In this work, whe have implemented and tested a version of *RandomForestClassifier* and *DecisionForestClassifier*. Both approaches are quite similar and have a lot of common concepts, that is why they have been developed in an OOP style following some SOLID principles. The algorithms have been tested for the required configurations which as a reminder are:

1. Random Forest Classifier
   (a) $NT = 1, 10, 25, 50, 75, 100$
   (b) $F = 1, 3, log2(M + 1), \sqrt{M}$
2. Decision Forest Classifier
   (a) $NT = 1, 10, 25, 50, 75, 100$
   (b) $F = int(M/4), int(M/2), int(3 * M/4), RU(1, M)$

Where RU stands for a random uniform value between 1 and M. The previous combinations have been tested on UCI Datasets. The datasets chosen for this experiments are:

1. iris (< 500)
2. car ($500 < n\_instances < 2000$)
3. kr-vs-kp (> 2000)

The datasets will be explained later on in the report. Finally we will comment the results extracted from the models, the times for each task, accuracies and the most relevant features.

Chapter 2

Classifiers

In this chapter we will discuss the base classes for all Classifiers as well as the particularities of each of the two classifiers implemented for this work. The implementation of the Classifiers is segmented in several parts:

1. BaseClassifier (*./src/base_classifier.py*)
2. ForestInterpreter (*./src/forest_interpreter.py*)
3. Node, Tree, Leaf: all of them inherit from BaseClassifier (*./src/tree_units.py*)
4. BaseForest: inherits from BaseClassifier and ForestInterpreter (*./src/base_forest.py*)
5. RandomForestClassifier: inherits from BaseForest (*./src/random_forest.py*)
6. DecisionForestClassifier: inherits from BaseForest (*./src/decision_forest.py*)

The *BaseClassifier* is an abstract class implementing the fit, predict and fit_predict methods for the classes that inherit from it. *ForestInterpreter* implements the loading and inference methods to create a Forest from a model previously saved as json. The *Node, Tree and Leaf* classes are the main units of the building of a tree structure and they all inherit from *BaseClassifier* as they need fit and predict methods. Then the *BaseForest* class inherits from *BaseClassifier* and *ForestInterpreter*. The first for the basic train methods and the later for inference. Finally the two *ForestClassifiers* inherit from *BaseForest* and will each implement their distinguishable features.

2.1 Leaf

The class *Leaf* implements the basic termination of a Tree structure. The main concept behind it is that once a branch of the tree has been terminated, will imply that a prediction has to be made for that instance and so, when in training we reach a leaf, we need to compute the probability for the instance to be of a class. For that we store the probability ccomputed as the count of class instances for the instances that reach the Leaf in the node divided by the number of instances in the leaf. When predicting, a dictionary containing the class names and the probability will be returned.

2.2 Node/Tree

The *Node* and *Tree* class are equivalent. However, the tree class is instantiated by the Classifiers while the Node is only instantiated by the Tree or another Node. The class is responsible for multiple actions as defined by the following methods:

2.2.1 fit

The goal of this method is to determine the best split of the node and determine the two branches if is not feasible to split the data anymore, a return statement forces the parent node to terminate that Node attempt with a Leaf. The main process for the training of the node is as follows:

Algorithm 1: Node/Tree fit method

```
Result: self
X := data for the node;
F := number of random features;
attributes := dataset attributes;
node_gini := gini_index(X);
best_gain := 0;
best_feature, best_value := None, None;
if $F < 0$ or $len(attributes) < F$ then
    features := attributes
else
    features := K random attributes
end
for feature in features do
    for unique value in feature column in X do
        true_split, false_split = try to split by the condition. $feature == value$ for
          categorical and $feature >= value$ for numerical;
        if $len(true\_split) == 0$ or $len(false\_split) == 0$ then
            skip to next value as it does not split the data;
        end
        gain := node_gini - split_gain(true_split, false_split);
        if $gain > best\_gain$ then
            best_feature := feature;
            best_value := value;
            best_gain := gain;
        end
    end
end
true_split, false_split := split(X, best_value, best_feature);
// initialize both branches from the node. If the split only contains one item, create leaf
  instead and finally call recursively the nodefit method. If the method returns a None,
  replace the newly created node with a Leaf instead to terminate the process.;
branches[True] := initialize_branch(true_split);
branches[False] := initialize_branch(false_split);
```

The algorithm above is implemented on the Node/Tree. First, the data for the node X, the number of random features to F and dataset attributes *attributes*. Then we compute the gini_index for the instances in X. The gini_index is computed efficiently using pandas following the following expression:

$$Gini(X) = 1 - \sum_{i=0}^{N} (\frac{X_{x \in C_i}}{len(X)})^2$$

after computing the gini_index of the data in the node, a set of F features are chosen from the dataset attributes. If there are not enough attributes, all attributes will be used. Also, if $F < 0$, all attributes will be considered. For each feature considered, all (feature, value) pairs in the dataset are tested and the gini index gain is computed for all and the split condition with the highest gain is chosen to be the condition for this node.

Once the condition is set, the branches are then initialized. There are 3 scenarios:

1. the number of instances is 1 for a given split. Then the branch is initialized to a Leaf.
2. the branch is initialized as a Node but no gain is found in further splitting the data. Then the branch is initialized to a Leaf.
3. the branch is initialized to a Node and it fits approppiately. We then continue creating nodes recursively.

2.2.2 predict

The predict method in the Node will evaluate the condition and then return whatever the branch returns recursively. This way we can return the dictionary containing the probabilities provided by the Leaf of the tree.

2.3 ForestInterpreter

The forest interpreter class contains the main loading and prediction methods for any BaseForest type of Classifier. The most interesting method for the class is the *predict* method. The predict method will ask the prediction for each instance to each tree and then average the probabilities for each of the class in order to get the most probable class for each instance of the dataset. It will call the predict method for each tree in the classifier. The class was optimized using the python package *multiprocessing* which allows us to generate a pool of threads that run a function's code for each item in a list. This way we can paralellize the computation of the predictions for each three among the desired number of jobs using:

```python
import multiprocessing as mp
from functools import partial
with mp.Pool(self.n_jobs) as p:
    predictions = list(p.map(partial(self._predict_tree, X=X), self.trees))
```

2.4 BaseForest

The main class for the ForestClassifiers. Implements the main fit methods for generating the trees from the train dataset. Also implements the methods to save the Classifiers as json objects. The fit method initializes the dataset characteristics and then creates NT trees and calls the fit method for each one of them. The same optimization that was done in the predict function in the ForestInterpreter class was done here, where each tree is fitted in a different thread to be able to paralellize the computation.

2.5 DecisionForestClassifier

The first final object is the DecisionForestClassifier, which inherits from BaseForest all the base methods for inference and training. The functions that the class implements is the *_fit_tree* method. In the case of the DecisionForestClassifier, the method first generates a dataset with F columns chosen at random from the dataset attributes. Once this dataset is generated, a tree is fitted with the newly constructed dataset where the per-node selection parameter F is set to -1 to exhaustively search for all (feature, value) pairs of the instances at each node.

2.6 RandomForestClassifier

Finally the RandomForestClassifier, which also inherits from BaseForest for the same reasons as DecisionForestClassifier, is implemented. Similarly to the DecisionForestClassifier, it only implements the *_fit_tree* method, which generates a bootstrapped dataset with the same number of instances as the original dataset. Then initializes the tree where the F value is set to the F value of the RandomForestClassifier to explore only F features at each node.