Partial Classification Forest

Jonas Faßbender

jonas@fc-web.de

Abstract

I. Introduction

Some datasets do not allow a classifier to generate a descision surface good enough to be able to predict unseen observations well enough. Well, in this case, referrs to a context dependent threshold for any quality measurement of a classifier, for example the accuracy or an information loss metric.

But for some of those problems, it may still be valuable to predict only on partitions of the feature space, in which the dataset is 'clean' enough, meaning a classifier can be found within the subset of the dataset laying inside one of those partitions which equals or exceeds the threshold.

This paper proposes a Monte Carlo based ensemble method called Partial Classification Forest (PCF), builing an ensemble of trees having a structure similar to k-d trees to partition the feature space of the dataset in order to find 'clean' partitions. In the following a tree generated by the PCF is spelled Tree with a capital T, rather than tree, which is used to denote the tree data structure.

In Section II I will lay out the structure of a Tree generated by the PCF before, in Section III, describing how PCF uses the Trees and listing its parameters. After that I will continue displaying test results using PCF. In Section V I will discuss further optimizations and possible additional features before finishing with a conclusion.

II. The Tree structure

A Tree generated by the PCF is a binary search tree structure similar to k-d trees. Its purpose is to randomly generate disjoint partitions of a feature space.

A Tree has two types of nodes, non-leaf nodes, here denoted as Nodes and leaf nodes denoted as Leafs. It provides two operations: (i) FIT, initializing the Tree and (ii) PREDICT, returning a label for an oberservation.

The Node structure contains three properties: (i) a split value; (ii) a left and (iii) a right successor, both references to either another Node or a Leaf.

A Leaf on the other hand, is the structure representing a partition of the feature space, having (i) active, a boolean value deciding whether the partition's quality, determined during the FIT operation, is equal or better than the defined threshold or not; (ii) a predictor which is used to classify oberservation during the PREDICT operation and two vectors with arbitrary length (iii) containing the observations of the dataset used in FIT, which are laying inside the partition and (iv) their inherent labels.

During the FIT operation a Tree contains a third type of node, Nil. Nil is used to initialize Trees and the left and right successor of a Node. These nodes are transformed during FIT to either a Node or a Leaf, so after the FIT operation a Tree does not contain Nil nodes anymore.

Algorithm 1 : FIT($\Theta, X, y, h, \beta_X, \gamma, \tau_g, \tau_{|X|}, \tau_h$)

Inputs:

Θ	–	a pointer to a Nil node; initially pointing to the root node of an empty Tree,
X	–	input data,
y	–	labels of X,
h	–	height of the Tree; initially $h = 0$,
β_X	–	lower and upper boundries of every dimension of X,
γ	–	function returning a predictor an its quality,
τ_g	–	quality threshold,
$\tau_{\|X\|}$	–	threshold for the size of X,
τ_h	–	height limit of the Tree

Output: void

```
1: predictor, gain ← γ(X, y)
2: if h > τ_h or |X| < τ_|X| or gain < τ_g then
3:     Θ ← LEAF(false, predictor, X, y)
4: else if gain ≥ τ_g then
5:     Θ ← LEAF(true, predictor, X, y)
6: else
7:     dimension ← h mod |X[0]|
8:     split ← RANDOM(β_X[dimension ])
9:     Θ ← NODE(split, NIL, NIL)
10:    split X, y and β_X into X′, X″, y′, y″, β′_X, β″_X
11:    FIT(Θ.left, X′, y′, h + 1, β′_X, ...)
12:    FIT(Θ.right, X″, y″, h + 1, β″_X, ...)
13: end if
```

Algorithm 2 : PREDICT(Θ, x, h)

Inputs:

Θ	–	a Tree node; initially pointing to the root of the Tree,
x	–	an observation,
h	–	height of the Tree; initially $h = 0$

Output: the predicted label or Λ

```
1: if TYPE(Θ) is Node then
2:     dimension ← h mod |x|
3:     if x[dimension] ≤ Θ.split then
4:         PREDICT(Θ.left, x, h + 1)
5:     else
6:         PREDICT(Θ.right, x, h + 1)
7:     end if
8: else if Θ.active then
9:     return Θ.predictor(x)
10: else
11:    return Λ
12: end if
```