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, refers 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), which builds 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 and the operations of a Tree generated by the PCF before, in Section III, describing how PCF utilizes Tree instances. 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 the following properties: (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) optionally a predictor which is used to classify oberservations during the PREDICT operation. Only if a Leaf's active property is true, a predictor must be provided. A Leaf also has two vectors with arbitrary length as properties: (iii) a vector 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. A Nil node does not have any properties.

A. The FIT operation

The FIT operation constructs a Tree, based on a dataset split in observations (X) and their labels (y). Algorithm 1 shows how FIT recursively builds a Tree, which is at the beginning a pointer to a Nil node.

The most important parameter passed to FIT is γ. γ is a function returning (i) a predictor and (ii) the loss of it. Otherwise γ is treated as a black box by the PCF, so what the predictor is and how its loss is calculated are not relevant to the PCF, as long as the predictor is callable and returns an element from the label set when called (Algorithm 2, line 9). The loss returned by γ gets compared to the quality threshold τ_l. Is the loss $\leq \tau_l$ the predictor is good enough and Θ is transformed to an active Leaf (Algorithm 1, lines 2, 3).

There are two other thresholds besides τ_l, $\tau_{|X|}$, τ_h. Both regulate the behaviour of a Tree's growth. $\tau_{|X|}$ defines a minimum amount of observations a Leaf must contain. One can easily imagine, without $\tau_{|X|}$ or $\tau_{|X|} = 0$ a Tree would never stop growing, since FIT would continue to split empty partitions, trying to find a smaller partition which would be predictable, even though no predictor could be generated without observations to train it on.

τ_h further regulates the maximum path length of a Tree. It is necessary besides $\tau_{|X|}$, because of the following scenario: be $\tau_{|X|} = 2$ and there are two equal observations in the dataset, but both having a different label than the other one. Now γ, passed X containing only those two identical observations, returns a predictor with a loss $> \tau_l$. Since $|X|$ is still not smaller than $\tau_{|X|}$ FIT would continue trying to separate the two inseperable observations. To prevent such a szenario τ_h tells FIT to stop before the Tree's height, the amount of edges of the longest path, would exceed τ_h. The path length of the Tree's root to Θ is passed as a parameter h to FIT.

Now, if neither τ_l is exceeded nor $\tau_{|X|}$ or τ_h is violated, FIT performs a split and transforms Θ to a Node (Algorithm 1, lines 7ff). The dimension the split is performed on is chosen in a cyclic manner, a practise also applied to k-d trees. [1] But rather than chosing the splitting value at the median of the observations in the dimension, which is done in order to construct balanced k-d trees, the splitting value is random.[1]

Afterwards FIT is recursively applied to the two new partitions (Algorithm 1, lines 11, 12).

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

Inputs:

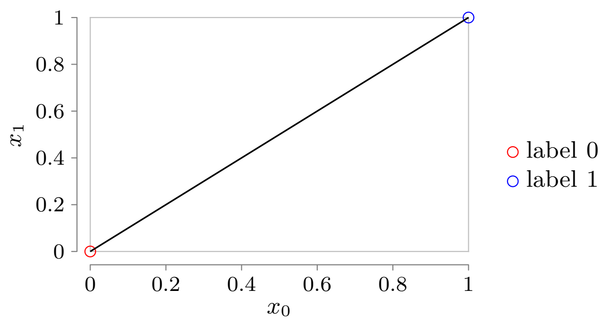
Θ	–	a pointer to a Nil node; initially pointing to the root node of an empty Tree,
X	–	input data,
y	–	labels of X,
h	–	height of the Tree; initially $h = 0$,
β_X	–	lower and upper boundries of every dimension of X,
γ	–	function returning a predictor an its quality,
τ_l	–	quality threshold,
$\tau_{\|X\|}$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

Output: void

```
1: predictor, loss ← $\gamma(X, y)$
2: if loss $\leq \tau_l$ then
3:     $\Theta$ ← LEAF(true, predictor, $X$, $y$)
4: else if $h > \tau_h$ or $|X| < \tau_{|X|}$ or loss $> \tau_l$ then
5:     $\Theta$ ← LEAF(false, predictor, $X$, $y$)
6: else
7:     dimension ← $h$ mod $|X[0]|$
8:     split ← RANDOM($\beta_X$[dimension ])
9:     $\Theta$ ← NODE(split, NIL, NIL)
10:    split $X$, $y$ and $\beta_X$ into $X'$, $X''$, $y'$, $y''$, $\beta'_X$, $\beta''_X$
11:    FIT($\Theta$.left, $X'$, $y'$, $h+1$, $\beta'_X$, ...)
12:    FIT($\Theta$.right, $X''$, $y''$, $h+1$, $\beta''_X$, ...)
13: end if
```

B. The PREDICT operation

The PREDICT operation traverses a Tree until it encounters a Leaf. If the Leaf is active a label to a provided observation x is returned by the predictor property of the Leaf, otherwise Λ is returned.



Λ must not be an element of the label set.

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($\Theta$) is Node then
2:     dimension $\leftarrow h \bmod |x|$
3:     if $x[\mathit{dimension}] \leq \Theta$.split then
4:         PREDICT($\Theta$.left, $x$, $h+1$)
5:     else
6:         PREDICT($\Theta$.right, $x$, $h+1$)
7:     end if
8: else if $\Theta$.active then
9:     return $\Theta$.predictor($x$)
10: else
11:     return $\Lambda$
12: end if
```

References

[1] Brown, R. A. Building a balanced k-d tree in $o(kn \log n)$ time. Journal of Computer Graphics Techniques (JCGT) 4, 1 (March 2015), 50–68.