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. 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.

It should be noted here, that this paper is rather a Proof of Concept of a very early version of the PCF and has several shortcomings in research and empirical tests, due to a lack of time and no complete, fast implementation. I will discuss these shortcomings further in Chapter VI, but the most important ones I will list right away:

- Tests with a more sofisticated γ (cmp. II-A)
- The PCF's performance on high dimensional data
- A Tree's growing behaviour
- Benchmarks

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 Leaves. 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[1] 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 (Algorithm 1, line 7). [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]

In order to chose a proper splitting value β_X is passed as another parameter to FIT. β_X represents the boundries for every dimension of the feature space based on X. For each dimension β_X contains a tuple with the minimum and maximum value in the dimension of all observations in X.

β_X[dimension] is passed to a pseudo-random number generator generating a random value so that $lower(\beta_X[\text{dimension}]) \leq$ random number $\leq upper(\beta_X[\text{dimension}])$ (Algorithm 1, line 8).

Afterwards X, y, β_X are splitted into two new disjoint partitions and FIT is recursively applied to the two new partitions (Algorithm 1, lines 10ff).

Since τ_h is defined, the maximum amount of nodes a Tree can have is $2^{\tau_h+1} - 1$ if the Tree would be perfectly balanced. [3, chapter 16.1] For each node FIT is called, so building a Tree has a worst case time complexity of $\mathcal{O}((2^{\tau_h+1} - 1) * \mathcal{O}(\text{FIT}))$. $\mathcal{O}(\text{FIT})$ is determined by the size of X, since X has to be splitted and by $\mathcal{O}(\gamma)$. That said, a single FIT operation would have a worst case time complexity of $\mathcal{O}(|X| + \mathcal{O}(\gamma))$,

[1] There could be another Interface for the predictor, which, as of right now is not yet specified.

which would mean the time complexity of the whole fitting process would be $\mathcal{O}((2^{\tau_h+1}-1) * (|X| + \mathcal{O}(\gamma)))$.

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

A Tree's FIT operation.

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 and its loss,
τ_l	–	loss threshold,
$\tau_{\|X\|}$	–	threshold for the size of X,
τ_h	–	height limit of the Tree

Output: void

predictor, loss $\leftarrow \gamma(X, y)$
if loss $\leq \tau_l$ then
  $\Theta \leftarrow$ LEAF(true, predictor, X, y)
else if $h > \tau_h$ or $|X| < \tau_{|X|}$ or loss $> \tau_l$ then
  $\Theta \leftarrow$ LEAF(false, predictor, X, y)
else
  dimension $\leftarrow h$ mod $|X[0]|$
  split $\leftarrow$ RANDOM(β_X[dimension])
  $\Theta \leftarrow$ NODE(split, NIL, NIL)
  split X, y and β_X into X', X'', y', y'', β'_X, β''_X
  FIT(Θ.left, X', y', $h+1$, β'_X, ...)
  FIT(Θ.right, X'', y'', $h+1$, β''_X, ...)
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 (Algorithm 2).

Λ must not be an element of the label set.

PREDICT is fairly similar to the search operation of a binary tree, except for the type distinction and the prediction.[2, chapter 12.2] Therefore, PREDICT has a worst case time complexity of $\mathcal{O}(\tau_h + \mathcal{O}(\text{predictor}))$.

Algorithm 2 : PREDICT(Θ, x, h)

A Tree's PREDICT operation.

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

if TYPE(Θ) is Node then
  dimension $\leftarrow h$ mod $|x|$
  if $x[dimension] \leq \Theta$.split then
    PREDICT(Θ.left, x, $h+1$)
  else
    PREDICT(Θ.right, x, $h+1$)
  end if
else if Θ.active then
  return Θ.predictor(x)
else
  return Λ
end if

III. Partial Classification Forest

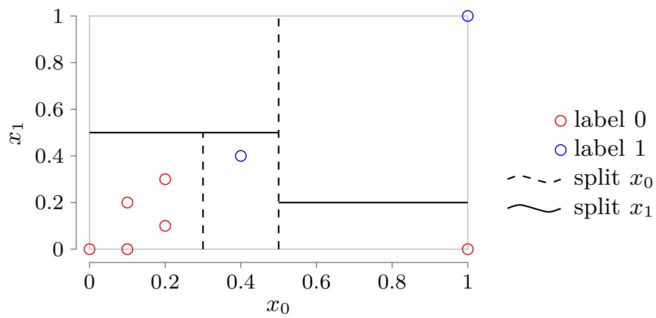
The PCF has two parameters, N and an array with N pointers. N is the amount of Trees the PCF maintains. Initially the pointers inside the array are references to Nil nodes.

The PCF offers the same two operations a Tree has, FIT (Algorithm 3) and PREDICT (Algorithm 4), both abstractions to the equivalent Tree operations.

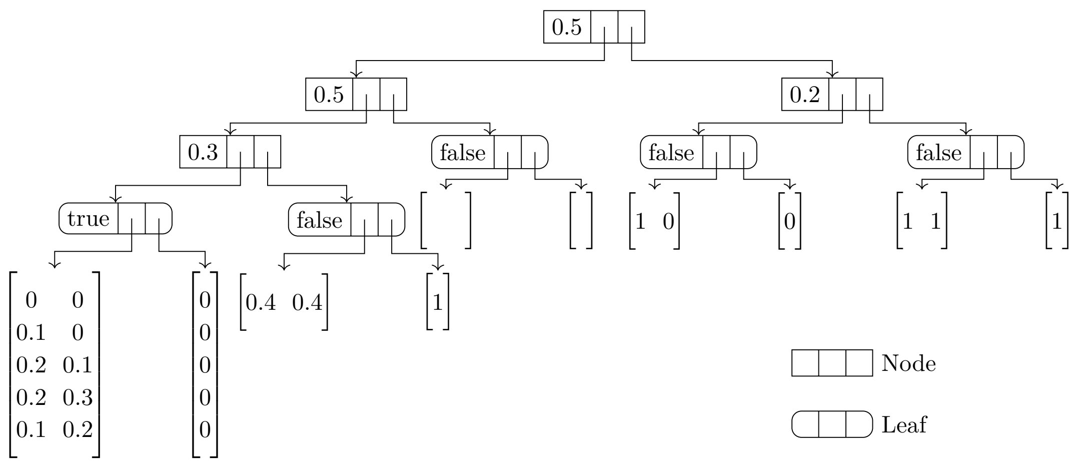
Once FIT is executed, the pointers are references to the root of fitted Tree instances.

Both FIT and PREDICT can be implemented as multithreaded operations as long as γ is threadsafe, since the Tree instances are independent of each other and the shared parameters X, y (FIT) and x (PREDICT) are read only, making synchronization unnecessary.

FIT first computes β_X which has a time complexity of $\mathcal{O}(|X[0]| * |X|)$, $|X[0]|$ denoting the amount of dimensions the feature space has.



(a) Scatterplot showing the observations and the splits done by the FIT operation.



(b) The structure of the Tree generated by FIT.

Fig. 1: Example of FIT on a dataset seen in Figure 1a. γ simply computes the probability of each label in y and returns a function returning the label with the maximum probability and as loss 1 - the maximum probability. The thresholds are: $\tau_l = 1$, $\tau_{|X|} = 2$. τ_h can be any integer above 2.

After that the Tree's FIT operation is called N times, which means the PCF's FIT operation has a worst case time complexity of $\mathcal{O}(N * (2^{\tau_h+1} - 1) * (|X| + \mathcal{O}(\gamma)) + |X[0]| * |X|)$ (cmp. II-A).

The PCF's PREDICT operation first initializes an array with N elements (Algorithm 4, line 1). Each Tree instance fills one element of the array with its prediction. After that the PCF's PREDICT operation takes the label predicted most and returns it as its prediction for the observation x (Algorithm 4, lines 5, 6).

The worst case time complexity of the PCF's PREDICT operation is $\mathcal{O}(N * (\tau_h + \mathcal{O}(\text{predictor})) + N)$, since a Tree's PREDICT operation is executed N times, plus the most predicted label must be determined, which is $\mathcal{O}(N)$.

IV. Tests

This chapter will present two conducted tests. The first one will visualize what the descision surface of the PCF looks like, the second

Algorithm 3 : FIT($\Pi, X, y, \gamma, \tau_l, \tau_{|X|}, \tau_h$)

The PCF's FIT operation.
Inputs:

Π	–	a PCF instance,
X	–	input data,
y	–	labels of X,
γ	–	function returning a predictor and its loss,
τ_l	–	loss threshold,
$\tau_{\lvert X\rvert}$	–	threshold for the size of X,
τ_h	–	height limit of the Tree

Output: void

compute β_X
for all $\Theta \in \Pi$.trees do
  FIT(Θ, X, y, 0, β_X, ...)
end for

Algorithm 4 : PREDICT(Π, x)

The PCF's PREDICT operation.
Inputs:

Π	–	a PCF instance,
x	–	an observation,

Output: the predicted label or Λ

predictions $\leftarrow [\Lambda; N]$
for $i = 1$ to N do
  predictions[i] = PREDICT(Π.trees[i], x, 0)
end for
determine l_{max}, the label predicted most
return l_{max}

will show how the PCF behaves with different amounts of Trees.

Both tests are performed on a randomly generated, normalized, two dimensional dataset. The dataset contains five thousand observations.

V. Further optimizations and additional features

VI. Conclusion

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.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. Introduction to Algorithms, Third Edition, 3rd ed. The MIT Press, 2009.

[3] Teschl, G., and Teschl, S. Mathematik für Informatiker, Band 1, 4th ed. Springer Spektrum, 2013.

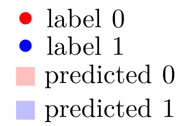


Fig. 2: The descision surfaces of differently configured PCFs. The threshold $\tau_{|X|}$ and the amount of Trees of the PCF are different for each Figure, ?? - ??.

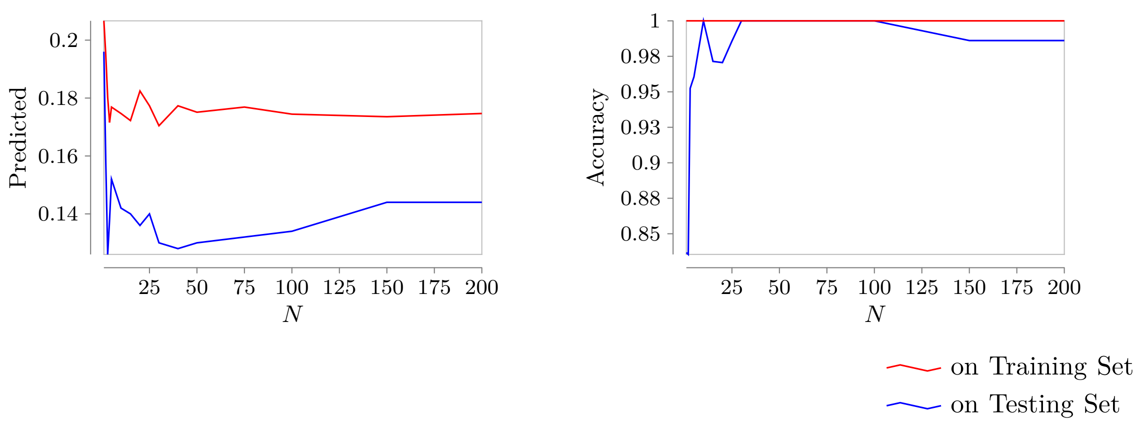


Fig. 3: The amount of Trees of the PCF N, in relation to the Predicted and the Accuracy value.