Partial Classification Forest

Jonas Faßbender

jonas.fassbender@fc-web.de

Abstract

This paper is concerned with a paradigm I call Partial Classification. It is based on the premise that for a given classification problem a solution must fulfill a certain quality criteria not always achievable by non-partial classification approaches. For some of these classification problems a partial solution achieving the quality criteria is more useful than a non-partial solution not fulfilling the criteria. A partial solution is a classifier that either predicts an observation or returns nothing if the corresponding label is not certain enough.

The whole idea behind Partial Classification opposes the classical, non-partial classification approach: rather than increasing the quality of a classifier, the quality — which must be achieved — is set beforehand and the space in which the classifier can predict with the given quality needs to be increased instead.

This paper proposes a method realizing Partial Classification called Partial Classification Forest (PCF). The PCF is a Monte Carlo based ensemble method. It is a Meta Classifier generating instances of a k-d tree like structure in order to partition the feature space of a given dataset. The PCF predicts on those partitions in which it achieves the given quality threshold.

Index Terms—Supervised Machine Learning, Partial Classification, Monte Carlo Method, Meta Classifier

I. Introduction

Some datasets do not allow a classifier to generate a decision surface good enough to be able to predict unseen observations well. A dataset contains tuples of observations and their labels. An observation is a point inside the feature space whereas the label is an element from a finite set called label set. A classification problem is the goal to find a function that maps the feature space to the label space based on the observations and their respective labels provided by the dataset.[6, chapter 18] This function is called a classifier.

Sometimes a classifier needs to fulfill a certain quality criteria in order to consider it a solution to the respective classification problem. For example: think of any classification problem that needs a classifier to have an accuracy of 99 percent — in this case impossible to find for the given dataset.

For some of those classification problems it may still be valuable to predict only on partitions of the feature space if the problem is partially solvable. A classification problem is partially solvable if a classifier can return the label of an observation or nothing if it is not certain enough it can predict the correct label. Non-partial classifier always return a label.

For the example above: if there exists at least one partition of the feature space in which a classifier can be found that has an accuracy of 99 percent the problem still can be partially solved if this is desired.

More generally accuracy would be the quality metric, the classifier's accuracy is the quality measurement and 99 percent is the quality threshold which must be met by a classifier's quality measurement in order to consider the classifier a (partial) solution to the classification problem. Any metric — for example a loss metric – can be used to determine a classifier's quality.

I call this concept Partial Classification. The idea opposes the classical, non-partial approach.

The non-partial approach would be trying to increase a classifier's quality until it equals or exceeds the quality threshold. Partial Classification takes the other way around. A partial classifier takes the quality threshold as its input and tries to find partitions of the feature space in which it can reach the quality. So rather than increasing the quality of the classifier the goal in Partial Classification is to increase the (sub-)space in which a partial classifier can predict.

This paper proposes a Monte Carlo based ensemble method realizing Partial Classification called Partial Classification Forest (PCF). The PCF builds an ensemble of trees having a structure similar to k-d trees. An instance of this tree structure partitions the feature space of a given dataset in order to find partitions, making Partial Classification possible.

It should be noted here: this paper is rather a Proof of Concept describing the structures and algorithms of a very early version of the PCF. It has several shortcomings in research and empirical tests, due to a lack of time and no complete, fast implementation. These shortcomings are:

- Tests with a more sophisticated γ (see Section II-A)
- The PCF's performance on high dimensional data
- Untested possible optimizations and features (see Section V)
- A Tree's growing behavior
- Benchmarks

In Section II I will lay out the structure and the operations of the k-d tree like structure the PCF uses.

In this paper such 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 III I will describe how the PCF works and utilizes Tree instances. After that I will continue displaying the application of the PCF and compare it to other, non-partial classifiers.

Because this paper is concerned with an early version of the PCF it will also contain a discussion about possible additional features which could further increase the PCF's performance in Section V.

Last I will finish with my conclusions and a road map.

II. The Tree structure

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

A Tree has two types of nodes: non-leaf nodes — here denoted as (i) Nodes — and leaf nodes denoted as (ii) Leaves. It provides two operations: (i) FIT, initializing the Tree and (ii) PREDICT, returning a label for an observation or Λ if uncertain.

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' quality — determined during the FIT operation — equals or exceeds the defined quality threshold; (ii) optionally a classifier which is used to classify observations during the PREDICT operation. Only if a Leaf's *active* property is true, a classifier 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 into observations (X) and their labels (y). Algorithm 1 shows how FIT recursively builds a Tree from a pointer to a Nil node.

The most important parameter passed to FIT is γ. γ is a function returning (i) a classifier and (ii) the loss of it. Otherwise γ is treated as a black box by the PCF, so what the classifier is and how its loss is calculated are not relevant to the PCF, as long as the classifier is callable[1] and returns an element from the label set when being 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 classifier returned by γ is considered good enough and Θ is transformed into an active Leaf (Algorithm 1, lines 2, 3).

There are two other thresholds besides τ_l: $\tau_{|X|}$ and τ_h. Both regulate the behavior of a Tree's growth. $\tau_{|X|}$ defines a minimum amount of observations a Leaf must contain. 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 could be predictable, even though no classifier can be generated without observations to train it on.

τ_h further regulates the maximum path length of a Tree. It is necessary besides $\tau_{|X|}$: be $\tau_{|X|} = 2$ and there are two equal observations in the dataset, both having a different label than the other one. γ — passed X containing only those two identical observations — returns a classifier with a loss $> \tau_l$. Since $|X|$ is still not smaller than $\tau_{|X|}$ FIT would continue trying to separate the two inseparable observations. To prevent such a scenario τ_h regulates 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. τ_h also provides a way to further regulate the time complexity of FIT and PREDICT.

[1]There could be another interface for the classifier, for example a predict method similar to the scikit-learn library.[2]

FIT performs a split and transforms Θ to a Node (Algorithm 1, lines 7ff), if neither the classifier's loss matches τ_l nor $\tau|X|$ or τ_h is violated. The dimension the split is performed on is chosen in a cyclic manner, a practice also applied to k-d trees (Algorithm 1, line 7). [1] But rather than choosing the splitting value at the median of the observations in the dimension — done in order to construct balanced k-d trees — the splitting value is determined randomly.[1]

In order to chose a proper splitting value β_X is passed as another parameter to FIT. β_X represents the boundaries 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[2] 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 split into two new disjoint partitions and FIT is recursively applied to both (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 is perfectly balanced.[7, chapter 16.1] For each node FIT is called, so building a Tree has a worst case time complexity of:

$$\mathcal{O}((2^{\tau_h+1} - 1) * \mathcal{O}(\text{FIT})). \tag{1}$$

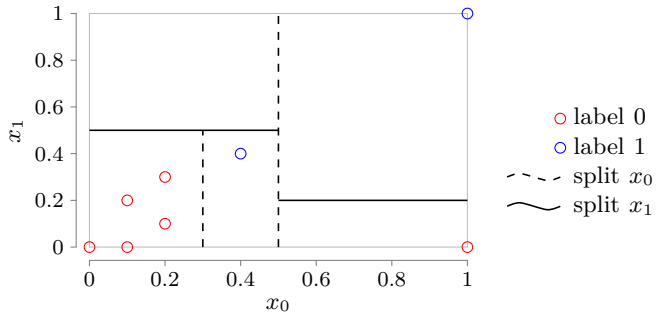
$\mathcal{O}$(FIT) is determined by the size of X — since X has to be iterated in order to split it — and by $\mathcal{O}(\gamma)$. That said, a single FIT operation would have a worst case time complexity of:

$$\mathcal{O}(|X| + \mathcal{O}(\gamma)), \tag{2}$$

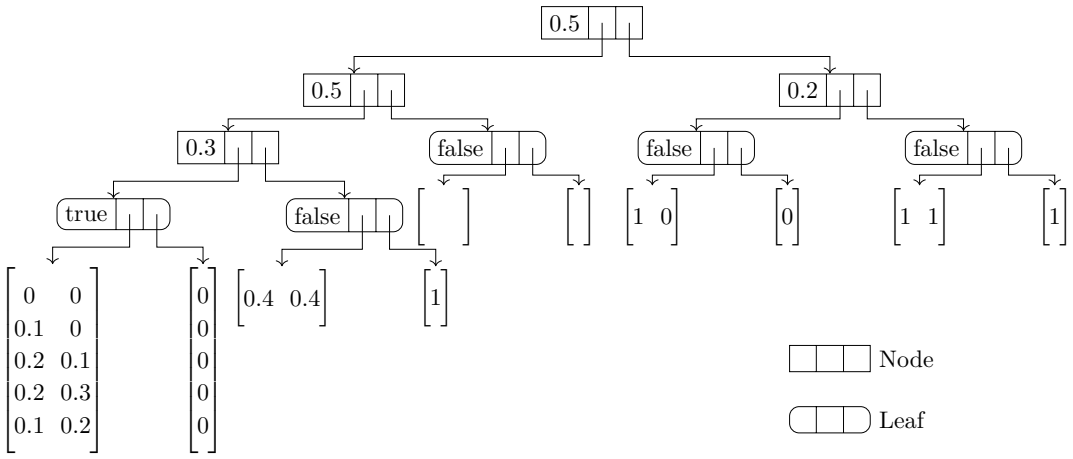
which means the time complexity of the whole fitting process is:

$$\mathcal{O}((2^{\tau_h+1} - 1) * (|X| + \mathcal{O}(\gamma))) \tag{3}$$

[2]The implementation used in Section IV utilizes Python 3.6's random library.[8, chapter 9.6]



(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 applied to the 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.

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 classifier property of the Leaf, otherwise nothing (here denoted with Λ) is returned (Algorithm 2).

Λ must not be an element of the label set.

Since the PCF does not predict on its own but instead uses other classifier instances for the actual prediction the PCF is a Meta Classifier.[4, chapter 4.6]

PREDICT is fairly similar to the search operation of a binary tree, except for the type distinction and the prediction.[3, chapter 12.2] Therefore, PREDICT has a worst case time complexity of:

$$\mathcal{O}(\tau_h + \mathcal{O}(\text{classifier})). \tag{4}$$

III. Partial Classification Forest

The PCF has two parameters, (i) N and (ii) an array with N pointers. N is the amount of

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 boundaries of every dimension of X,
- γ – function returning a classifier and its loss,
- τ_l – loss threshold,
- $\tau_{|X|}$ – threshold for the size of X,
- τ_h – height limit of the Tree

Output: void

```
1: classifier, loss ← γ(X, y)
2: if loss ≤ τ_l then
3:     Θ ← LEAF(true, classifier, X, y)
4: else if h > τ_h or |X| < τ_|X| or loss > τ_l then
5:     Θ ← LEAF(false, classifier, 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)

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

```
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.classifier(x)
10: else
11:    return Λ
12: end if
```

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 roots of fitted Tree instances.

Both FIT and PREDICT can be implemented as multi-threaded operations as long as γ is thread-safe, 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:

$$\mathcal{O}(|\text{dimensions}(X)| * |X|). \tag{5}$$

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:

$$\mathcal{O}(N * \text{FIT} + |\text{dimensions}(X)| * |X|) \tag{6}$$

(see Equation 3 for $\mathcal{O}$(FIT)).

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:

$$\mathcal{O}(N * (\tau_h + \mathcal{O}(\text{classifier})) + N), \tag{7}$$

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

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

since a Tree's PREDICT operation is executed N times, plus the most predicted label must be determined, which is $\mathcal{O}(N)$.

Algorithm 4 : PREDICT(Π, x)

The PCF's PREDICT operation.
Inputs:
Π – a PCF instance,
x – an observation,
Output: the predicted label or Λ

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

IV. Application

This Section will further outline the use case of Partial Classification and the PCF, displaying results of a test on an artificial dataset and comparing the PCF to other, non-partial classifiers. Furthermore a test of the PCF's behavior with different amounts of Trees will be shown.

Both tests are performed on a randomly generated, normalized, two dimensional and binary labeled dataset. The dataset contains five thousand observations and was designed to be unpredictable, when predicted as a whole. The plane from which the observations are generated contains five partitions in which the observations all have the same label. Observations from those five partitions make up twenty percent of all observations and are the only ones which should be predicted, because all points not inside those partitions are labeled randomly. The optimal decision surface would be equal to the area of the five partitions.

All observations are generated by a pseudo-random number generator[8, chapter 9.6], therefore: every point on the plane has the same probability to be chosen as an observation for the dataset.

The dataset is designed this way in order to be able to show the application of the PCF in the domain of Partial Classification. Only twenty percent of the whole dataset is predictable which makes it impossible for other, non-partial classifiers to be able to find an adequate decision surface (see Figure 3). The following tests all take $\tau_l = 1$ as quality threshold which is impossible to achieve predicting on the whole dataset.

It is common practice in machine learning to split a dataset into a training and a test set in order to find the best model.[6, chapter 18] This approach is also used for the PCF. The training set is used as the parameters X and y of FIT (Algorithm 3) while PREDICT (Algorithm 4) is used on every observation of the test set.

Two metrics are used to describe the behavior of the PCF, (i) *predicted* and (ii) *accuracy*. Both metrics are derived from comparing the label of an observation returned by PREDICT with its actual label from the dataset.

Predicted is the percentage of predicted observations, while *accuracy* is the percentage of correctly classified observations from the test set. *Accuracy* is used as the quality measurement, which means in the context of this tests,

every classifier not able to achieve $accuracy = 1$ when predicting the test set does not solve the classification problem.

For both tests γ, τ_l and τ_h are the same. γ and τ_l are equal to the values used in Figure 1 while τ_h equals 32. The dataset was split into a training and a test set such that ten percent of the observations were used as the test set.[3] The observations for the test set were chosen randomly.

The first test will show the decision surfaces of PCF instances with different N and $\tau_{|X|}$. For the test three different values, two, five and ten were used for both N and $\tau_{|X|}$ to show how those two parameters change the decision surface of the PCF instances. The test shows that for $\tau_{|X|} = 2$ the PCF instances over-fitted the data, *predicted* exceeding twenty percent — the amount of accurate observations — while *accuracy* fails to equal τ_l. This results in the chaotic decision surfaces shown in Figures 2a - 2c. On the other hand for $\tau|X| = 10$ the decision surface is very small which means the PCF instances' *predicted* is far less than twenty percent (Figures 2g - 2i).

How non-partial classifiers perform on the dataset can be seen in Figure 3. Three common classifiers are tested, all from the scikit-learn library (version 0.20.1).[2] Their parameters are the standard parameters given by scikit-learn. Tested were the Support Vector Machine, Random Forest and K Nearest Neighbors, where K in this case equals five.[2] Figure 3 also shows the *accuracy* of these classifiers. None are close to $\tau_l = 1$.

The second test shows the influence the amount of Trees N has on *predicted* and *accuracy*. In this test $\tau_{|X|}$ equals four.

Figure 4 shows: for this dataset with the chosen thresholds the PCF instances with $N < 30$ variate, both their *predicted* and *accuracy* values. Is $30 \leq N \leq 100$ *accuracy* is constant and equals τ_l. *Predicted* on the other hand still rises in this interval. Is $N \geq 100$ *predicted* is higher than *predicted* with $N = 100$ and constant, but *accuracy* fails to meet τ_l, which means the PCF instances are over-fitting.

Furthermore the second test relates *predicted* and *accuracy* of the observations from the test set to the values for the training set, showing that more training observations are predictable than test observations (see Figure 4).

[3]For splitting scikit-learn's model_selection.train_test_split was used during the tests.[2]

V. Possible additional features

There are some possible features and optimizations to the PCF which were not discussed in the previous Sections. Again it should be noted that these features and optimizations are yet untested (see Section I).

The proposed features are:

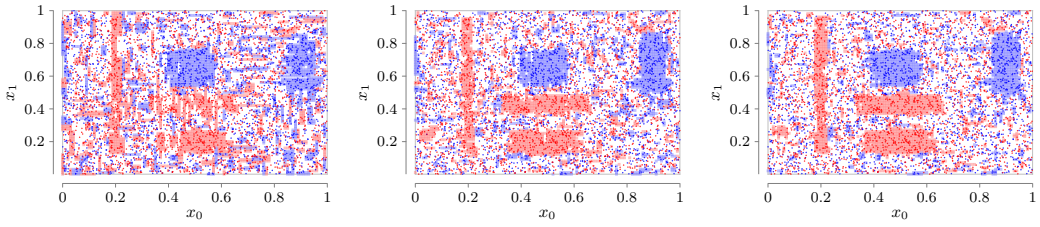
1) Weighing partitions.
2) Compressing a Tree after FIT.
3) Select k best or reducing N in another way.
4) Choosing the dimension a Node has randomly rather than cyclic (see Section II-A).
5) Add a rotation matrix to every Tree.
6) A second, more light-weight variant of the PCF reducing the memory usage, depending on γ.

The first feature would be to weigh partitions. As of right now, the PCF's PREDICT operation determines l_{max} as the label predicted most (Algorithm 4, line 5).

This could be further refined with weighing each partition based on two properties: (i) the amount of observations a partition contains and (ii) its volume. This would result in a weight determined as:

$$\text{weight}(\text{partition}) = \frac{|\text{partition}.X|}{V(\text{partition})}. \quad (8)$$

A partition with a lot of observations and a small volume would have a higher weight than one with a small amount of observations and a high volume, further increasing the probability that the label predicted by the partition with the higher weight is determined as l_{max}. So instead of just counting each label predicted and returning the one predicted most, the weight of each prediction is summed and the label with



(a)

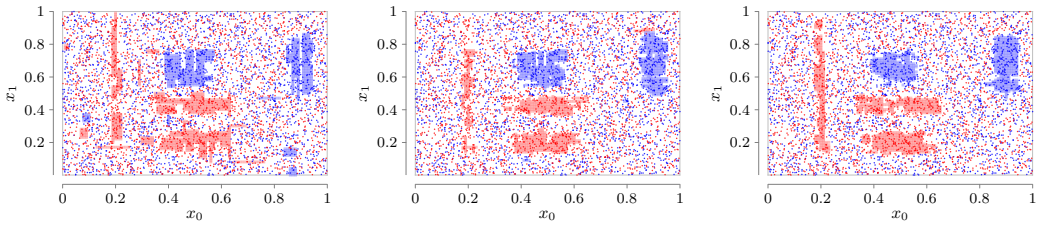
$\tau_{|X|} = 2, N = 2,$
$predicted = 0.37,$
$accuracy = 0.670$

(b)

$\tau_{|X|} = 2, N = 5,$
$predicted = 0.308,$
$accuracy = 0.721$

(c)

$\tau_{|X|} = 2, N = 10,$
$predicted = 0.298,$
$accuracy = 0.792$



(d)

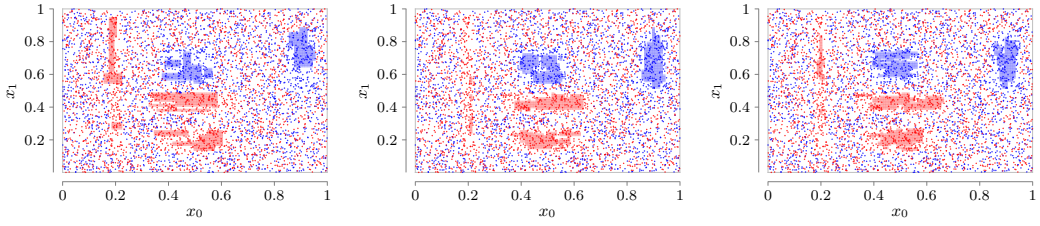
$\tau_{|X|} = 5, N = 2,$
$predicted = 0.144,$
$accuracy = 0.903$

(e)

$\tau_{|X|} = 5, N = 5,$
$predicted = 0.106,$
$accuracy = 0.962$

(f)

$\tau_{|X|} = 5, N = 10,$
$predicted = 0.138,$
$accuracy = 1$



(g)

$\tau_{|X|} = 10, N = 2,$
$predicted = 0.076,$
$accuracy = 1$

(h)

$\tau_{|X|} = 10, N = 5,$
$predicted = 0.074,$
$accuracy = 1$

(i)

$\tau_{|X|} = 10, N = 10,$
$predicted = 0.068,$
$accuracy = 1$

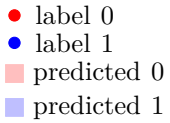
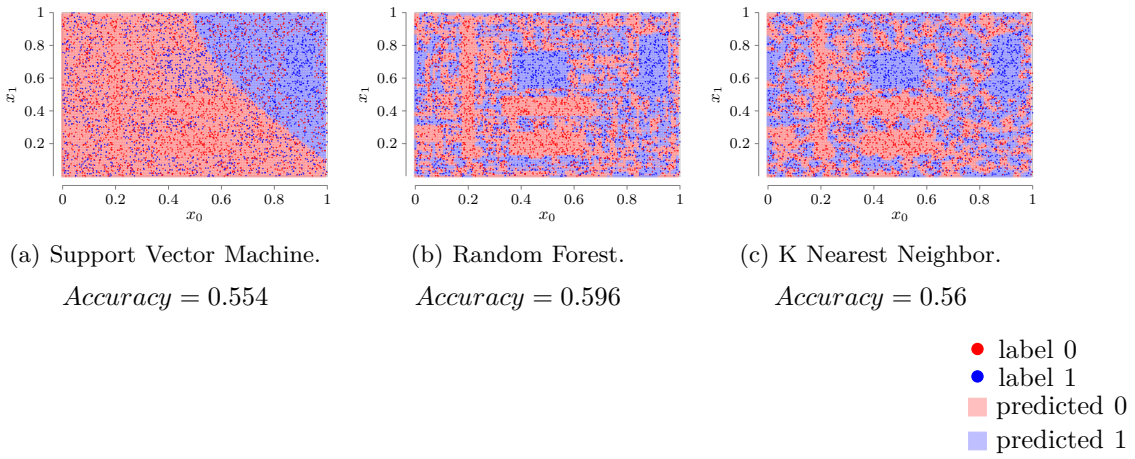


Fig. 2: The decision surfaces of differently configured PCF instances. The threshold $\tau_{|X|}$ and the amount of Trees N are different for each Figure, 2a - 2i.



(a) Support Vector Machine. $Accuracy = 0.554$

(b) Random Forest. $Accuracy = 0.596$

(c) K Nearest Neighbor. $Accuracy = 0.56$

Fig. 3: The decision surfaces of non-partial classifiers.

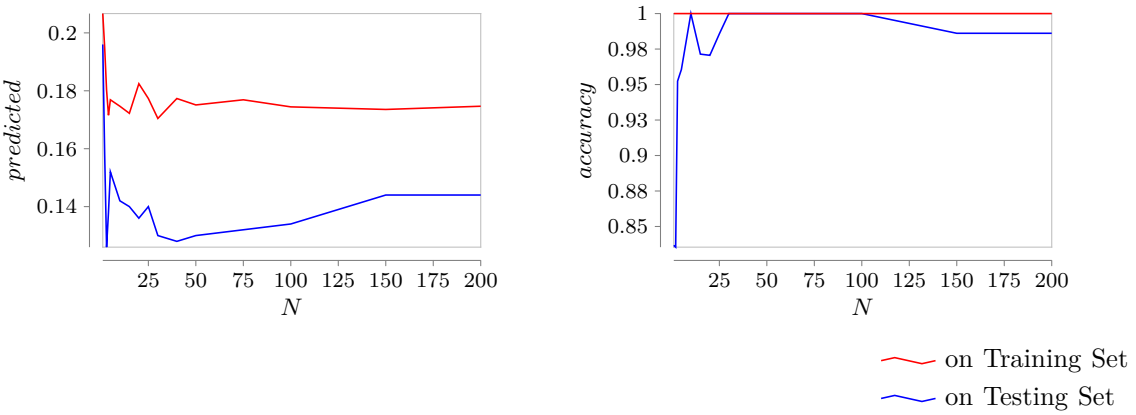


Fig. 4: The amount of Trees N, in relation to *predicted* and *accuracy*.

the highest sum will be returned by PREDICT as l_{max}.

The second optimization is compressing a Tree after FIT. If a Node has two inactive Leaves as children the Node is unnecessary since either way Λ is returned. The Node could be transformed into an inactive Leaf, reducing the path length of the branch by one and decreasing the amount of nodes by two, which decreases the size of the Tree and therefore the time complexity of PREDICT.

Another feature would be to reduce the amount of Trees N after FIT, removing Trees with little active partitions from the PCF instance, decreasing the time complexity of PREDICT.

Two features inspired by methods used in the Approximate Nearest Neighbor Search are (i) choosing the splitting dimension — like the splitting value — at random rather than cyclic (see Section II-A) and (ii) also rotating the data for each Tree instance.[5, pages 17 - 27] This further increases the possibility of having Trees with different structures and therefore

partitions.[5, page 24]

In Section IV and in Figure 1 a very simple classifier is returned by γ. It only computes the probability for each label and returns the label with the highest probability (see Figure 1).

A classifier like that does not need to know where the observations are in the partition which makes it unnecessary to keep them in a Leaf node as X and y (see Algorithm 1). Instead a dictionary with every label from the label space where each label is mapped to the amount of observations inside the partition having the particular label is enough. A possible feature would be to provide a second variant of the PCF which passes this dictionary instead of the observations to γ, decreasing the complexity of the PCF for classifiers which do not need to know where each observations lays inside the partition.

VI. Conclusions

In the Introduction I described what I mean by Partial Classification. The whole concept is based on the condition that a classifier needs to maintain a certain quality in its predictions for a given classification problem. If it can not do this, it is not regarded a solution.

Partial Classification is a way to still find classifiers that partially solve a classification problem not solvable as a whole. A partial classifier only predicts the label of an observation if it lays inside a partition it can classify. Otherwise it returns nothing, indicating that the classifier is not certain which label the observation has.

This changes the whole approach to increasing a classifier's quality. Since a partial classifier has its quality set beforehand, the goal to increasing the partial classifier further is to enlarge the (sub-)space it can predict on. On the other hand: non-partial classifier do not have the quality given as a threshold, which means their quality is variable rather than the space they predict on.

This paper provides a description of a Partial Classification method, the Partial Classification Forest (PCF). The PCF's biggest advantage is the fact it is a very flexible Meta-Classifier, since it treats the classifier instances it relies on (and their quality measurement) as a black box (see Section II-A - γ).

This paper describes the core structures and operations of the PCF and shows its use on an artificial problem designed to display the use of it over other, non-partial classifiers.

Also stated in the Introduction, this paper has some shortcomings in research and empirical tests regarding the PCF, due to a lack of time and no complete, fast implementation. A list of these shortcomings is also provided in the Introduction. The next step will be to make a fast and complete implementation before proceeding with the research and the empirical tests.

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] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning (2013).

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

[4] Hanke, M., Halchenko, Y. O., and Oosterhof, N. N. PyMVPA Manual, 2.6.1st ed. PyMVPA.org, 2017.

[5] Kalantidis, Y. Approximate Nearest Neighbor Search. image.ntua.gr/iva/files/ann.pdf, 2010.

[6] Russel, S., and Norvig, P. Künstliche Intelligenz: Ein moderner Ansatz, 3 ed. Pearson, Higher Education, München, 2012.

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

[8] van Rossum, G., and the Python development team. The Python Library Reference, 3.6th ed. Python Software Foundation, 2019.