DELIVERABLE : Hybrid Random Forest

Archambeau Thomas, Fratacci Emeline, Montese Sara,
Rullier Antoine, Traoré Mohamed

March 30, 2022

Abstract

This document is a technical report of our one year cap projet advised by Pr.Prevost, in collaboration with the LDR (Learning, Data & Robotics Lab) of ESIEA.
In this paper, we study a new kind of Random Forest, which uses Gaussian Mixture Models and Near Class Mean Classifier inside the nodes. The final goal is to make this algorithm incremental both in data and in classes, which we do not cover here. We study the impact of different parameter of this algorithm, and its performance on different well-known datasets.

Contents

1 Introduction **2**

2 Random selection **2**
2.1 Sub-features . . . 2
2.1.1 Results as a function of the depth . . . 2
2.1.2 Global Results . . . 3
2.2 Feature quality . . . 4
2.2.1 Results . . . 4

3 Random Selection **5**
3.1 Experiments and results . . . 5

4 Generative Nodes **5**
4.1 Threshold . . . 5
4.2 Accuracy . . . 5

5 Test on Cifar10 and Fashion-MNIST **5**
5.1 Cifar10 . . . 6
5.2 Fashion-MNIST . . . 6

6 Perspective **7**
6.1 Code improvements . . . 7
6.2 Co-variance matrices . . . 7
6.2.1 Discriminating Nodes . . . 7
6.3 Generative Nodes . . . 7
6.4 Divide by 2 . . . 7

1 Introduction

Random forests are well-known algorithms in machine learning. They are based on decision trees and use ensemble learning to make them work together. In a decision tree, each node is a binary classifier that split data into two sub-groups. Each sub-group is the data for a new node.
We propose here a new kind of random forest, which use Gaussian Mixture Model on top (node with generative model) and Near Class Mean Classifier model for the other nodes (discriminating ones).

2 Random selection

In a decision tree, each node uses different features available to find the best way to split the data. However, computationally speaking it would be very fastidious to look at all the features in each node of the tree. Moreover, the forest may contain hundreds of trees, which contain hundreds of nodes. Thus we must choose a limited number of features to consider in each node: this is what we call *random selection*. In this process, since only one subset of features is chosen, we must define a way to select them.

2.1 Sub-features

The parameter *sub-features* refers to the number of features selected in each node. Today, the state of the art for this parameter is $sqrt(D)$ with D the total number of features. This formula has been experimentally discovered with standard random forests. In this report we propose a new formula for this parameter. Note that this new formula is used with our NCMForest only.
With the tree going deeper, the number of samples per node is decreasing (is divided by 2 at each depth on average). So, in order for the algorithm to correctly estimate co-variance parameters (that we need for NCMNodes), we need to decrease the dimension of data. Meaning that instead of choosing a constant number of sub-features for random selection, we will decrease it with the depth of the tree. We propose in this report the following formula:

$$\frac{D}{2^{depth}}$$

We will name it *divide by two*.
The goal of the following experiments is to compare the two formulas for random selection: *sqrt* and *divide by two*. Since this parameter has an impact on the computation of the co-variance in discriminating nodes, we compute the experiment with Mahalanobis and Euclidean distance for discriminating nodes.
We tested the algorithm on Digits data set, combining the parameters:

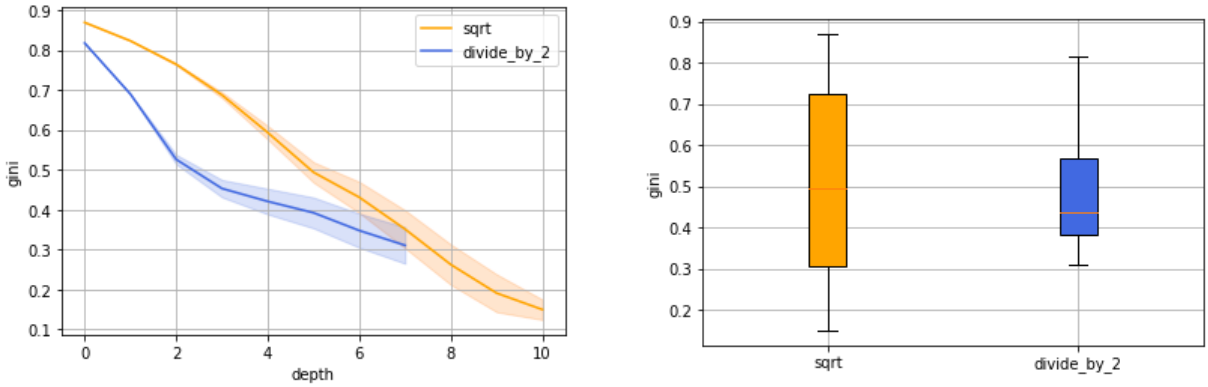
- distance: euclidean, Mahalanobis
- method_max_features: divide_by_2, sqrt

2.1.1 Results as a function of the depth

As the co-variance matrix should be estimated with more precision, we can expect an impact of nodes classification consequently on nodes purity. Figure 1 shows nodes purity for each depth of the forest with the state of the art formula *sqrt* and with our formula *divide by two*.

In the (a) graph is shown, for each depth, the mean of Gini indices of nodes at the same depth. We observed that Gini decreased faster with *divide by two*.
The (b) graph is a boxplot of Gini of all nodes. The purity median is slightly lower for our parameter, which indicates an improvement, as expected. But more importantly, Gini's variance is reduced a lot.

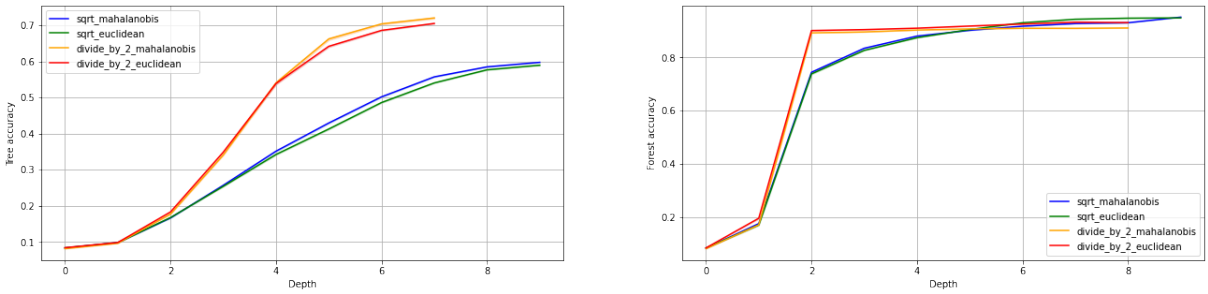
The figure 2 shows the accuracy of the NCMForest at each depth with *sqrt* and with *divide by two* for the numbers of selected features. In the (a) graph it is shown the mean of trees accuracy. We observed that trees reach a better accuracy and those faster with *divide by two* than with *sqrt*. In the (b) graph, it is shown the accuracy of the whole forest for each depth. We notice that, accordingly with trees accuracy, the forest learns faster with *divide by two* but does not reach higher accuracy.



(a) Gini at each depth

(b) Gini per node

Figure 1: Purity on DIGITS, number of features



(a) Mean of trees accuracy at each depth

(b) Forest accuracy at each depth

Figure 2: Accuracy on Digits, number of features

2.1.2 Global Results

The figure 3 shows the accuracy means and standard deviations of 20 trainings, considering the default parameters for the NCMForest, in particular considering no generative layers.

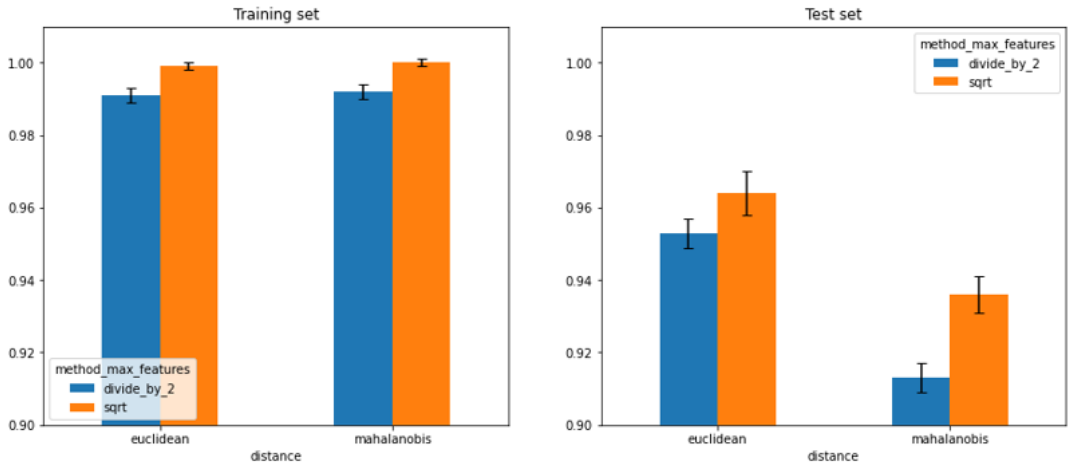


Figure 3: Accuracy on Digits (global results)

Comparing accuracy scores and training time, we can observe that the decreasing function *divide by two* is slightly less accurate than the state of the art. On the other hand, the table 5 shows that our proposal is faster, taking on average 15-20% less time for training.

	TRAINING SET				TESTING SET			
	mean		std		mean		std	
method_max_features	*divide_by_2*	*sqrt*	*divide_by_2*	*sqrt*	*divide_by_2*	*sqrt*	*divide_by_2*	*sqrt*
distance								
mahalanobis	0.992	1	0.002	0.001	0.913	0.936	0.004	0.005
euclidean	0.991	0.999	0.002	0.001	0.953	0.964	0.004	0.006

Figure 4: Accuracy scores

	mean		std	
method_max_features	*divide_by_2*	*sqrt*	*divide_by_2*	*sqrt*
distance				
mahalanobis	34.61	41.36	1.07	2.57
euclidean	40.97	47.04	2.70	2.95

Figure 5: Training time (in seconds)

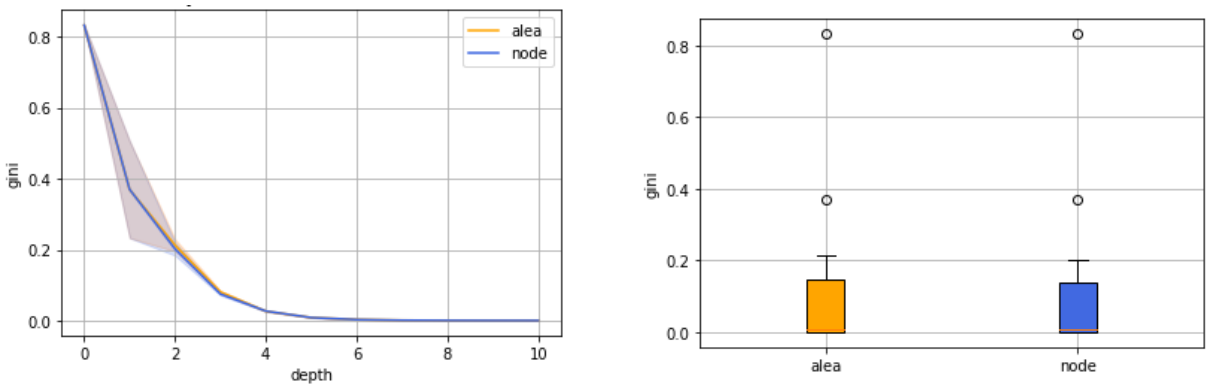
A bit after we compute this test, we realized that *divide by two* had a flaw in the way it is implemented. For this test, the algorithm decreases the number of features when the tree deepens as it should. When the formula outputs one feature, though, the node becomes a leaf and the tree doesn't continue to grow. We have corrected this behavior and now, the number of features has a minimum of two and the tree can use two features for all its next nodes. In this way it will continue to grow until another condition stops it. Perhaps these results could be modified if tests are recomputed.

2.2 Feature quality

In a standard Random Forest, D features are selected randomly at first, and then each one is tested with purity indices on a split to find the features corresponding to the best separation. In our NCMForest, each discriminating node uses all of D features at the same time since they use a Nearest Class Mean Classifier, which is able to split multidimensional data. Since all selected features are used, we thought it can be worth it to select them better than with randomness.
The idea is to compute the ratio of inter-variance on within variance of data for each feature, sort these ratios and pass them into a Soft-max function. The result is a probability for each feature: the higher the probability, the more useful the corresponding feature is. After this computation, we can select D sub-features accordingly to their probabilities.

2.2.1 Results

As we can see on figure 6, the gini index decreases along the tree, as expected. However, it does not show any significant improvement by selecting the features over random selection (*alea*).



(a) Gini at each depth

(b) Gini per node

Figure 6: Purity on MNIST, quality of features

3 Random Selection

To build the tree, we must decide which class will go into the left or right child of each discriminating node. There are several ways to achieve this, in this section we will discuss two of them: random and majority class selection.

In random selection, we simply put randomly each class inside one of the children of the current node.
In majority class selection, firstly we compute the class that has most elements in the current node. Secondly, we put this class into the left child of the tree, and all other classes in the right child.

3.1 Experiments and results

We tested this parameter on Iris and Digits data sets. As we can see in figure 7, the accuracy does improve when we choose the majority class in Digits. The results on Iris are not very significant, since the data set is really small. For

Selection	Iris	Digits
Random	98.0%	94.9%
Majority class	96.0%	95.4%

Figure 7: Accuracy depending on random selection

larger data sets, we expect the majority class selection to perform significantly better than the random selection.

4 Generative Nodes

4.1 Threshold

The threshold is used to select the proportion of samples that are selected in the left child of the generative node, by comparing their distance to the one defined by the Centroid in each node. The goal of this experiment is to study the influence of this parameter on the model performance. We tested values for the threshold ranging from 50% to 100% of the samples. The tests were performed on Digits and MNIST data sets.
Results obtained in figure 8 show a strong positive correlation between the value of the parameter *alpha* and the accuracy of the generative node. The 100% value being better for MNIST is a bit strange though, and 95% seems to be the better choice overall.

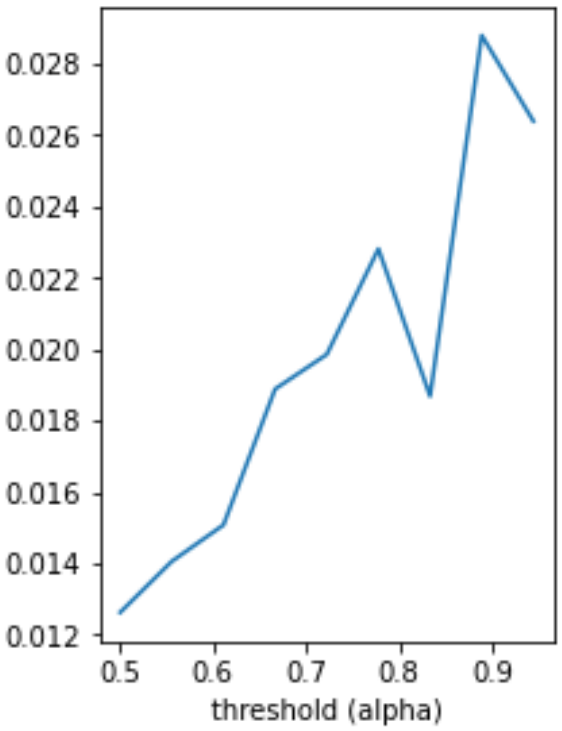
4.2 Accuracy

To estimate how good performs the first generative layer of the NCMForest, we can't compute a standard score. In fact, the standard score needs that each tree outputs the label of a class. But since generative nodes have only two outputs, unless the classification problem has only two classes, generative layer cannot provide to the function score what it needs. So we created a metric to evaluate these generative nodes. If X_i is the data of class i and $T_i(x)$ the proportions of x recognized by trees responsible for modes of class i, then our metric is $mean(T_i(X_i) \forall\ i \in \mathrm{N}^*$
On Cifar10, extracted features from a pre-trained Densenet121 on Image-Net, the NCMForest obtains a value for our metric equal to 83% with a variance close to zero. It means that 83% of samples are well classified by their respective tree.

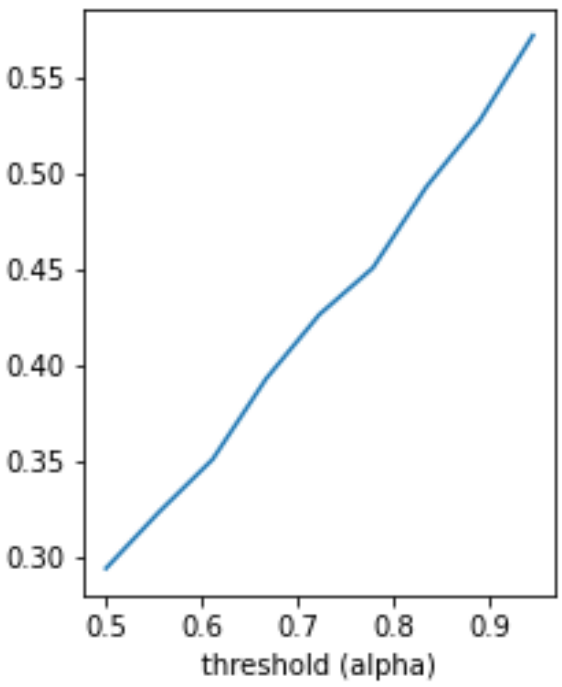
We realize that the metric we use here naturally increase when the threshold increase. So, maybe this test can re-computed with another metric, see 6.3.

5 Test on Cifar10 and Fashion-MNIST

Finally we computer some results on benchmark data sets to compare the NCMForest with the state of the art.
In these tests, data are first passed through a ConvNet and then given to the NCMForest. Indeed, random forests are only a small part of the pipeline and in order to train them on images datasets, they need pre-computed features. To avoid any bias in the result, we use pre-trained ConvNet, pre-trained on others datasets such as Image-Net.
Each time we compute accuracy and a score of the first layer of generative nodes only. This score are computed the following way : formalization of pred1 and pred2
On the following results, you will see a column named *Features*. It is the number of features we keep from the output of ConvNets and that we give as input to the forest. We don't use all features because the training take too much time to compute and reducing the number of features has a big impact on the training time. This process of features selection takes bests features only regarding the ratio between/within variance.



(a) Digits



(b) MNIST

Figure 8: Accuracy depending on the threshold

5.1 Cifar10

The following table shows the results using two different approaches: feature selection with SelectKBest and feature extraction with PCA.

Model	Features	Tree Depth	Test Accuracy	Training time
Densenet121 + Feature Selection + NCMForest	662	1	23%	40’
		6	41%	1h
		10	44%	1h20’
Densenet121 + PCA + NCMForest	330	1	16%	10’
		6	33%	20
		10	29%	25’

The following table provide some information about inference time :

ConvNet(features output)	Features	Test Accuracy	Training time	Testing time
Densenet121(1024)	662	44%	1h20’	7’
	1024	46%	2h07’	18’

5.2 Fashion-MNIST

Model	Features	Tree Depth	Test Accuracy	Training time
PCA + NCMForest	74	1	70%	19’
		6	78%	20’
		10	80%	21’
AlexNet + Feature Selection + NCMForest	74	1	67%	10’
		6	73%	17’
		10	75%	18’

6 Perspective

6.1 Code improvements

During the project, we thought at some points that could have been improved. It is not about changing the algorithm but just modify the way it is implemented to gain speed and make it sturdier.

6.2 Co-variance matrices

There is an improvement to be made in the way the co-variance matrices are actually computed. See https://scikit-learn.org/stable/modules/covariance.html Actually, some error like *"SVD did not converge"* occurs sometimes during inversions of co-variance matrices. We think that calculating the matrices with sklearn's covariance tools such as reduced covariance, sparse inverse covariance or robust covariance could solve these problems, we could even think of using a tolerance value that will be added to the values before the passage to the inverse matrix.

6.2.1 Discriminating Nodes

In the current implementation, NCM nodes used in the random forest are written entirely by us in python. Using an implementation that is pre-existing should lead to better performance, for example replacing them with a GMM with one component from scikit-learn.

6.3 Generative Nodes

There is some work to do about the way of evaluate these nodes. The problem is that we need mode label to compute interesting metrics and these doesn't exist for prediction.

To reduce inference time maybe use the predict function of GMMs instead of the one of OneCentroid could be an idea.

6.4 Divide by 2

An idea could be to start decreasing the number of features when it is needed and not from the beginning. This would allow for better accuracy, since the number of feature is not decreased until actually needed for the computation.