

Combining Multiple Classifiers: Forests

Eric Cañas Tarrasón



Course: 2019/2020

Contents

1	Introduction	2
2	Implementation	2
2.1	C4.5	2
2.1.1	Prediction Implementation	3
2.2	Forest Classifiers	4
2.2.1	Random Forest	4
2.2.2	Decision Forest	5
2.3	<i>Forest</i> Prediction	5
2.4	Scalability of the Implementation	5
3	Results	6
3.1	Short Dataset - Votes	7
3.1.1	Previous <i>Rules</i> Based Method Results	7
3.1.2	Random and Decision Forest Results	7
3.2	Medium Dataset - Cars	9
3.2.1	Previous <i>Rules</i> Based Method Results	9
3.2.2	Random and Decision Forest Results	10
3.3	Large Dataset - Nursery	11
3.3.1	Previous <i>Rules</i> Based Method Results	11
3.3.2	Random and Decision Forest Results	11
4	How to execute the code	13
4.1	Code Structure	13
5	Conclusions	14

1 Introduction

The following report will present the implementation and the analysis of two **Ensemble of Classifiers** methods: **Random Forest**[1] and **Decision Forest**[2]. Both methods using as base classifiers **C4.5 Trees**[3].

Ensemble Classifiers uses set of classifiers, which should be trained with **independent data** (ideally), for classifying through a **voting system**. This method aims to reduce the **discrimination error** of each one of the weak classifiers which composes them. It will be possible as long as they have an accuracy **higher than random**, being **independent** among them. Moreover these concrete methods allows to measure the **relevance** which each feature have when predicting.

2 Implementation

This work is composed by three basic algorithms: The *Base Learner (C4.5)*, *Random Forest* and *Decision Forest*. The source code of these three algorithm, as well as the complementary files for reading data and experimentation, can be found attached to this document.

2.1 C4.5

C4.5 has been the algorithm selected for being used as *Base Learner*, it is described as follows:

Algorithm 1 C4.5

Data: Set of categorical examples **X** with **I** attributes. Set of his expected outputs **Y**. Optional: **F** amount of X_i attributes for being analyzed at each node.

Result: *Root node* of the classification tree. Node contains: **is leaf flag**, **prediction** which gives the predicted Class, **attribute** used for select the new son if node is not a leaf, **Sons** depending on the value of attribute.

```
1  $H_y \leftarrow \text{Entropy}(\mathbf{Y})$ 
2 node.prediction  $\leftarrow \text{Most Common}(\mathbf{Y})$  // If ask to this node it will predict it
3 if ( $H_y$  is 0.) or (there are no more attributes in X) then
4   | node.isLeafFlag  $\leftarrow \text{True}$  // This node is a leaf of the Tree
5 else
6   | if F (is not required) or (is higher than I) then
7     | Gains Ratio  $\leftarrow [\text{GainRatio}(X_i, \mathbf{Y}) \text{ for each } X_i \in \mathbf{X}]$ 
8   | else
9     |  $X_F \leftarrow F$  random attributes from X
10    | Gains Ratio  $\leftarrow [\text{GainRatio}(X_i, \mathbf{Y}) \text{ for each } X_i \in X_F]$ 
11  | end
12  | node.attribute  $\leftarrow$  Attribute with higher Gain Ratio // Attribute used for predict
    | // For each possible value of the  $X_i$  attribute
13  | foreach  $X_{i_v} \in X_i$  do
14    |  $(X_V, Y_V) \leftarrow (\mathbf{X}, \mathbf{Y})$  pairs where the value of  $X_i$  column is  $X_{i_v}$ . Deleting  $X_i$  column from X.
15    | node.Child_v = C45( $X_V, Y_V$ ) // Child_v Saves the value  $X_{i_v}$ 
16  | end
17 end
18 return node
```

This pseudo-code could be summarized on the following key points:

1. **Calculate the entropy of Y attribute:** If it is 0. it will mean that there is only one value in Y, thus this node will be directly a **leaf**. If it is greater, this node is not able to classify with **100% of confidence** and the algorithm will continue generating the tree while there were remaining attributes.
2. **Saving the Most Common Y as prediction:** No matter if the node is a leaf or not, the *Most Common Y* value among the remaining ones is saved as prediction. If it is a leaf, this prediction will be always used, else, this prediction will be only used when there will be no children with the searched *Value*. This case is common when, on the validation set, appears a value of X_i that there was **not present** in the training set in that current node.
3. **If Node is a leaf:** Mark it as a leaf and do nothing.
4. **If Node is not a leaf:**
 - (a) **Compute the Gain Ratio of its attribute:** If **F** parameter was passed (used when constructing through *Random Forest* method), compute only the gain Ratio of **F** random attributes. *Gain Ratio* formula is presented at equation 2

$$GainRatio(X, A) = \frac{Gain(X, A)}{SplitInfo_A(X)} \quad (1)$$

Where *Gain* is:

$$Gain(X, A) = H_x - Info(X, A) \quad (2)$$

Being H_x the general *entropy* of the objective attribute (calculated at step 1) and $Info(X, A)$, the sum of the objective attribute entropy of each subset X_v divided by the frequency of V .

Being $SplitInfo_A(X)$ the potential information of X_v .

Being X_v a subset which represents the training data set X when splitting it into an **individual partition** for each possible values of the attribute A .

And being the entropy:

$$H_x = - \sum_{v=1}^V p(x_v) \log_2(p(x_v)) \quad (3)$$

- (b) **Select the attribute with higher Gain Ratio:** Since this attribute will be the one which will be able to split the dataset in subsets with the **higher amount of information**, this attribute will be the one which this node will use to predict.
- (c) **Recursively generate a child for each possible value of that attribute:** For each possible value of that attribute **generate a child**. This child will be a complete tree trained with X_v , a subset composed by those instances where the column X_i had this value v , and **without** this column X_i .

This recursive algorithm generates the tree through the **DFS**[4] method. The implementation could introduce the frequency of the of the Y value selected as node.prediction into the current Y_v subset of the node as **confidence**, in order to estimate how trustworthy is this prediction. Moreover, this implementation could introduce a maximum allowed depth for the trees as regularization method.

2.1.1 Prediction Implementation

Predicting in **C4.5** trees only needs to descend through the tree depending on the values that each attribute takes. Algorithm 2 describes this process:

Algorithm 2 C4.5-Prediction

Data: Set of categorical examples **X**. **Node** of the Tree (Starting by root).

Result: **Y** prediction

```
1  $X_v \leftarrow$  Value of  $X_i$  with  $i = \mathbf{node.attribute}$ 
2 if ( $\mathbf{node.isLeafFlag}$ ) or (There is no  $Child_v$  for  $X_v$  value) then
3   | prediction  $\leftarrow$   $\mathbf{node.prediction}$ 
4 else
5   |  $X_{new} \leftarrow$  X without the attribute  $X_i$  with  $i = \mathbf{node.attribute}$ 
6   | prediction  $\leftarrow$  C4.5-Prediction( $X_{new}$ ,  $\mathbf{node.Child_v}$ )
7 end
8 return prediction
```

This prediction method takes the final decision in two cases:

1. When arrives to a **leaf**.
2. When arrives to a point where the next attribute to predict has a value X_v for which there are **no associated children**.

The algorithm will still deciding a new node of the tree and continue descending until one of this cases is satisfied.

2.2 Forest Classifiers

Two *Ensemble Classifiers* based on the previously presented **C4.5** tree have been implemented: **Random Forest** and **Decision Forest**. For the sake of simplicity, both will be described sequentially. However, in order to ensure the horizontal scalability of the algorithm, they have been **parallelized** into its implementation, generating each tree in a different *CPU core*. It allows the implementation to achieve greater Speed Ups as more *CPU Cores* can be offered.

2.2.1 Random Forest

Random Forest is the first version of *Forest* implemented. The differences that it presents with **Decision Forest** can be summarized as: The dataset which each tree receives and the use of parameter **F**, in the way that was described on pseudo-code 1.

Algorithm 3 Random Forest

Data: Set of categorical examples **X** with **I** attributes. Set of his expected outputs **Y**. **F** amount of X_i attributes for being analyzed at each node of the trees. **NT** amount of tree to use into the forest.

Result: A **Forest** represented as a list of **C4.5** classification trees.

```
1 Forest  $\leftarrow \emptyset$ 
2 foreach  $n \in [0..NT]$  do
3   |  $(X_n, Y_n) \leftarrow$  Bootstrap(X, Y)
4   | Add to Forest  $\rightarrow$  C4.5( $X_n, Y_n, \mathbf{F}$ )
5 end
6 return Forest
```

As seen, in order to build the **Random Forest** algorithm only needs to generate **NT C4.5 Trees**, requiring for **F**. Each of this trees will receive a different **bootstrapped** version of the dataset.

2.2.2 Decision Forest

Decision Forest is based on the same principle than previously presented **Random Forest** differing from it in the training data that each tree receive:

Algorithm 4 Decision Forest

Data: Set of categorical examples **X** with **I** attributes. Set of his expected outputs **Y**. **F** amount of X_i attributes for being analyzed by each tree. **NT** amount of tree to use into the forest.

Result: A **Forest** represented as a list of **C4.5** classification trees.

```

1 Forest  $\leftarrow \emptyset$ 
2 foreach  $n \in [0..NT]$  do
3   |  $X_n \leftarrow$  Matrix taking F random columns from X.
4   | Add to Forest  $\rightarrow$  C4.5( $X_n, Y$ )
5 end
6 return Forest
```

Is important to remark that the maximum amount of different trees that this algorithm can produce will be always $\binom{I}{F}$ where **I** are the total amount of attributes of **X**. Being:

$$\binom{I}{F} = \frac{I!}{F!(I-F)!} \quad (4)$$

It is important to have it in account for deciding **NT** when using datasets with a small amount of attributes. For example: In datasets with 5 attributes, the maximum number of different trees that can be generated is only 10, when **I** is 2 or 3.

2.3 Forest Prediction

Both *Forest* predict by a simple non weighted **voting method**. This method could be summarized as follows:

Algorithm 5 Forest Prediction

Data: An **x** instance, from which we want to predict its **Y** value. A **Forest**.

Result: **Y** prediction

```

1 Votes  $\leftarrow$  [C4.5-Prediction(x, Tree) for each  $Tree \in \mathbf{Forest}$ ]
2 Y Prediction  $\leftarrow$  Mode of Votes
3 return Y Prediction
```

2.4 Scalability of the Implementation

One of the purposes of the developed work was to generate an efficient an scalable implementation. Besides all the reuse of calculations and efficient implementations of the tree construction, *Forest* generation has been developed as a natural **parallel process**, in order to ensure its horizontal scalability. As illustrative example, figure 1 shows how the addition of *CPU cores* affects to the efficiency of *Forest* generation.

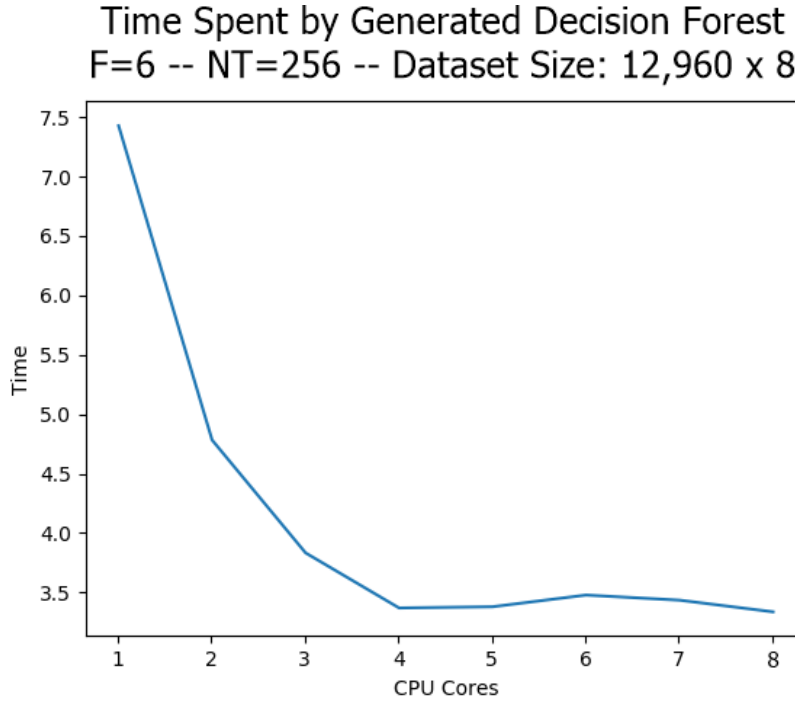


Figure 1: Ten executions mean time spent for constructing a **Decision Forest** with 256 trees with $\mathbf{F} = 6$. Executions have been performed in a system with 4 *CPU Hardware cores* and 8 *CPU Virtual cores* for Hyper-threading. X axis represents the amount of cores where the program has been allowed to execute a thread. Spent time is given in seconds

As seen it is able to reduce the computational time required for each *Forest* as it have more hardware resources available. As will be seen in the following section, it has allowed to make a more extensive analysis of results, adding more demanding parameters and evaluating the models through a **50-fold** cross-validation, in order to make the results more accurate and stable.

3 Results

In this section we will **analyze** the result of the implemented algorithm over **3 different datasets**. All experiments will be analyzed by ***k-fold methodology***, with a previous **shuffling** of the dataset in order to avoid implicit pre-orders.

For the sake of simplicity, original datasets has been modified, including a row with the **names of columns** at first line of the *CSV*. This modification allows to extract the *Feature Relevance* from each *Forest* without making any modification at code. Three used datasets, have been selected due its human **understandability**.

Moreover, in order to make this analysis comparable with the one performed at first *Practical Work*, these datasets will be the same used there. It will allow to compare the performance of **both methodologies** in terms of *Validation Accuracy* and the extracted *Feature Relevance*. Finally, For each dataset the following **hyper-parameterization** will be tested:

- **NT**: The *Number of Trees* within the *Forest*. Firstly, the following list of parameters was proposed: **1, 10, 25, 50, 75 & 100**. However, in order to verify if the performance could increase even more

when the depth of the *Forest* increases considerably, **two additional** large values have been added in the superior limit: **250 & 500**.

- **F**: The number of random features that the **Random Forest** will use when splitting nodes, or which the **Decision Forest** will receive as training set. For **Random Forest** the parameters experimented will be: **1**, **3**, $\log_2(\mathbf{I} + 1)$ & $\sqrt{\mathbf{I}}$ where **I** are the amount of features of the dataset. For **Decision Forest** the parameters experimented will be: $\lfloor \frac{\mathbf{I}}{4} \rfloor$, $\lfloor \frac{\mathbf{I}}{2} \rfloor$, $\lfloor \frac{\mathbf{I}+3}{4} \rfloor$ & *Random*(1, **I**) where *Random*() means a random integer number between 1 and **I** for each tree, taking it from an **uniform distribution**.
- **Feature Relevance**: In order to be representative of the method, *Feature Relevance* will be extracted from the *Forest* with **NT** = 250 and **F** = *Best*(**F**), being *best*(**F**) the **F** which gave the best *Accuracy* results over the dataset.

3.1 Short Dataset - Votes

Congressional Voting Records Dataset includes the votes of each one of the **U.S. House of Representatives Congressmen** on the **16 key votes** identified by the *CQA*, and its objective variable records if they are from the **democrat or republican** party.

This dataset is the shortest one analyzed including a total of **435 instances** with **16 binary attributes**. The fact of the attributes are binary compensates the high dimensionality of the dataset in terms of required **computational resources**, producing always **Binary Trees**.

Classes Distribution:

- **democrat** → 61.38%
- **republican** → 38.62%

3.1.1 Previous *Rules* Based Method Results

Rules based method obtained a **93.8%** of *Validation Accuracy* over this dataset. Pointing out as the most important features: *Physician Fee Freeze*, *Synfuels Corporation Cutback* and *Anti Satellite Test Ban*, in this order.

3.1.2 Random and Decision Forest Results

Tables 1 and 2 show the **accuracy** measures obtained by the **Random Forest** and **Decision Forest** for the full combinatory of proposed hyper-parameters. In order to be as accurate as possible with results, these results has been extracted from a **50-fold cross-validation** analysis.

Random Forest Accuracy				
		F		
		1	3	4
NT	1	85.0%	93.2%	91.2%
	10	91.5%	95.8%	95.2%
	25	94.5%	95.5%	95.8%
	50	93.8%	95.5%	95.8%
	75	94.2%	95.5%	95.8%
	100	93.5%	95.8%	95.5%
	250	94.0%	95.5%	95.8%
	500	94.0%	95.8%	95.8%

Table 1: *Validation Accuracy* of Random Forest for all the hyper-parameter combinatory over *Votes* dataset. Since $F = \lfloor \log_2(16 + 1) \rfloor = \lfloor \sqrt{16} \rfloor = 4$ this parameter has been not duplicated.

Decision Forest Accuracy					
		F			
		4	8	12	Random(1,16)
NT	1	86.4%	89.8%	92.5%	88.5%
	10	90.8%	94.8%	95.5%	94.0%
	25	92.5%	94.8%	95.8%	95.8%
	50	92.2%	94.5%	95.8%	95.2%
	75	92.0%	95.0%	95.5%	95.2%
	100	91.5%	95.2%	95.5%	95.2%
	250	91.8%	94.8%	95.5%	95.2%
	500	91.5%	95.0%	95.5%	95.2%

Table 2: *Validation Accuracy* of Decision Forest for all the hyper-parameter combinatory over *Votes* dataset.

As shown in these tables, the final accuracy of the method tends to improve for both cases as **NT** increases, especially for **Random Forest**. **Decision Forest** shows its maximum around 25 and 50 trees. For **F**, however there are different scenarios:

- **Random Forest:** In **Random Forest**, each tree will really have access to the full set of features, but only to random **F** of them at each time. It have a remarkable implication: While its possible to note slight differences when using **F** = 1 (Random attribute selection at each node), there are **no relevant differences** between using **3** or **4**.
- **Decision Forest:** In **Decision Forest**, parameter **F** have more relevant implications. There are differences between use **4**, **8** or **12** features that still present while **NT** increases. So it is possible to fix an optimal **F** among the experimented ones in **F** = 12.

F parameter presents in both cases a regularization method, in order to make the tree learners inside less likely to overfit.

If comparing these results with the ones obtained from *Rules* methods is possible to conclude that, in general, using an **embedded classifier** (when **NT** > 1) improves the *Validation Accuracy* of the resultant model. Both *Forests* have enhanced the results in its best case from the previous **93.8%** to **95.8%**.

Figure 2 summarizes the relevance of each feature extracted from both methods.

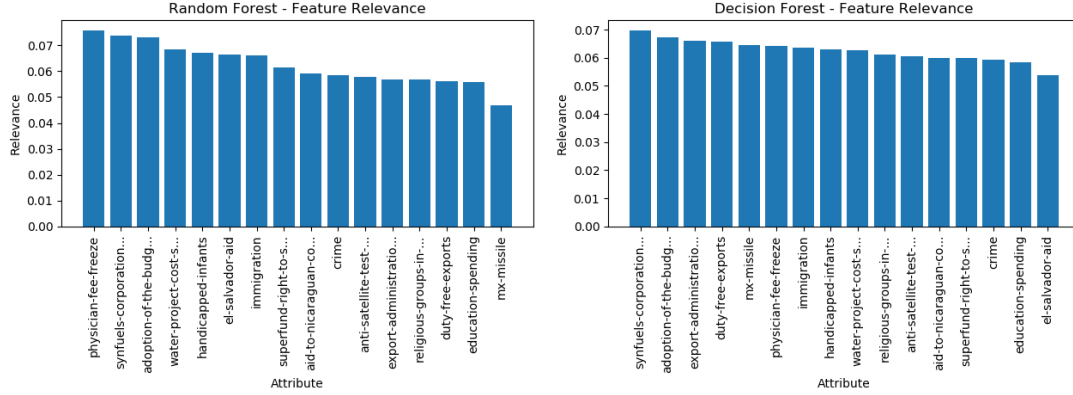


Figure 2: Feature relevance extracted from **Random Forest** and **Decision Forest** methods. These plots have been extracted from examples using **NT** = 250 trees in both cases, with **F** = 4 for **Random Forest** and **F** = 12 for **Decision Forest**. It is important to remark that while the plots generated for **Random Forest** were similar for all the cases, results from **Decision Forest** had a high variability

When comparing these results with the ones obtained by *Rules* algorithm, it is remarkable that first two most relevant features (*Physician Fee Freeze*, *Synfuels Corporation Cutback*) still stable. In the meantime, the third one (*Anti Satellite Test Ban*), decreases its relevance until last positions, in spite of it was possible to directly predict with it the 10% of the dataset. **Decision Forest** relevance results can not be taken into account since there was a high variability among them in each execution.

3.2 Medium Dataset - Cars

Car Evaluation Dataset is one of the **most commonly used** when testing classification algorithms, especially for constructive induction and structure discovery methods. It is because it was derived from a simple **hierarchical decision model** originally developed for the demonstration of *DEX* [6], where some parts of the structure have been hidden.

It has a total of **1,728 instances**. The purpose of the dataset is to predict how much a car is **acceptable or not** in function of **6 different properties**: *buying price*, *maintaining price*, *doors*, *persons capacity*, *luggage boot size* and *safety*. In this case, it is important to remark that with 6 different properties the maximum amount of different trees that **Decision Forest** can generate is 20, so we will expect no remarkable *Accuracy* improvements when using a larger parameter **NT**.

Classes distribution:

- **unacceptable** → 70.02%
- **acceptable** → 22.22%
- **good** → 4%
- **very good** → 3.76%

3.2.1 Previous *Rules* Based Method Results

Rules based method obtained a **95.0%** of *Validation Accuracy* over its dataset. Pointing out the following relevance order: *Buying price*, *maintaining price*, *quantity of doors*, *safety*, *luggage boot* and *number of persons*.

3.2.2 Random and Decision Forest Results

Tables 3 and 4 shows one more time the **Validation Accuracy** measures obtained by **Random Forest** and **Decision Forest** for the full combinatory of proposed hyper-parameters. These results have been extracted from a **50-fold cross-validation** analysis.

Random Forest Accuracy				
NT	F			
		1	2	3
	1	75.4%	91.4%	91.4%
	10	87.2%	94.4%	94.4%
	25	91.7%	94.5%	94.4%
	50	91.9%	94.5%	94.5%
	75	92.5%	94.5%	94.4%
	100	91.9%	94.5%	94.3%
	250	92.6%	94.5%	94.5%
	500	92.6%	94.6%	94.5%

Table 3: *Validation Accuracy* of **Random Forest** for all the hyper-parameter combinatory over *Cars* dataset. Since $F = \lfloor \log_2(6 + 1) \rfloor = \lfloor \sqrt{6} \rfloor = 2$ this parameter has been not duplicated.

Decision Forest Accuracy					
NT	F				
		1	3	4	Random(1,6)
	1	69.9%	68.5%	70.1%	71.2%
	10	69.9%	73.1%	79.2%	73.4%
	25	69.9%	71.2%	81.4%	71.5%
	50	69.9%	70.9%	82.2%	70.9%
	75	69.9%	70.9%	83.0%	70.5%
	100	69.9%	70.9%	82.8%	70.2%
	250	69.9%	70.9%	83.4%	70.4%
	500	69.9%	70.8%	83.4%	70.2%

Table 4: *Validation Accuracy* of **Decision Forest** for all the hyper-parameter combinatory over *Cars* dataset.

As its shown, there is a clear difference now among both cases. While *Random Forest* obtains a great performance since lower NTs, arriving up to **94.5%** of *Validation Accuracy*, **Decision Forest** is not able to achieve results higher to **83.4%**. However, the most important conclusion which can be extracted from here is: Why it occurs? It will be deeply explained in following analysis, when seeing at feature relevance plots. However, there is a clear clue onto **Decision Forest** table: The performance stills increasing as **F** increases.

For a better explanation lets see at figure 3, which summarizes the relevance of each feature extracted from both methods.

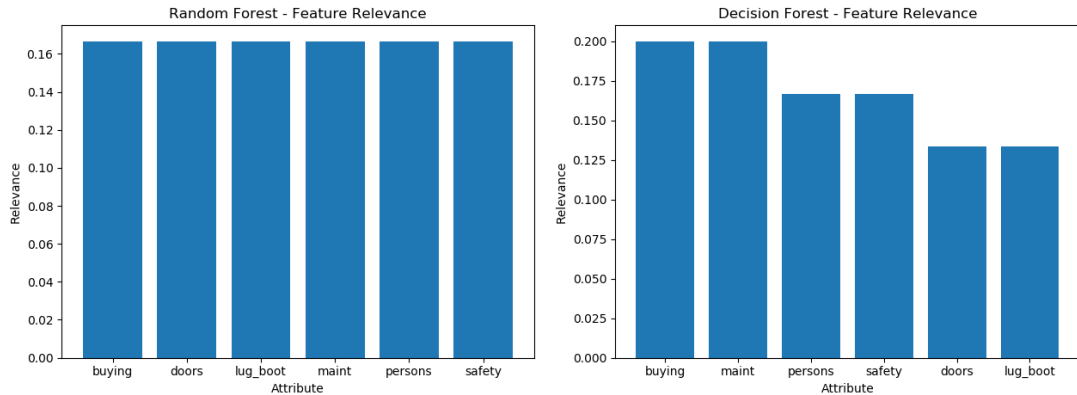


Figure 3: Feature relevance extracted from **Random Forest** and **Decision Forest** methods. These plots have been extracted from examples using **NT** = 250 trees in both cases, with **F** = 2 for **Random Forest** and **F** = 4 for **Decision Forest**. In **Random Forest** there were no cases where a different relevance among features was found. **Decision Forest** different experiments presented as variability as before.

This case is so different from last, in **Random Forest** there were no cases where any feature won among another, while **Decision Forest** continued presenting as variability as previous case. Comparing it with *Accuracy Results* obtained from tables 3 and 4 the conclusions are clear: Every tree **needs the total amount of Features for classifying** correctly. As **Random Forest** allowed it to see the total amount of features, there was no *Forests* that used less than **I** for its complete classification. In the **Decision Forest** case, it also needed to use the full of features that arrived, but as they are selected randomly, its feature relevance results were affected uniquely by randomness. It occurs due to the nature of the dataset, which came from a **hierarchical decision model**, which prepared it for work with all the proposed features.

As conclusions, it is no possible to compare the *Feature Relevance* extracted from these algorithm with the one extracted from the *Rules* method. Moreover, in terms of accuracy, this method has obtained worse results, falling from **95.0%** of *Accuracy* to **94.6%** in best case.

3.3 Large Dataset - Nursery

As its **official description** relates: *Nursery Database* is derived from a **hierarchical decision model** developed to **rank applications** for nursery schools. It was used during 1980's to give explanations about the rejected applications. Final decision of the model depended on three sub-problems: *occupation of parents and child's nursery, family structure and financial standing, and social and health picture of the family*.

It have a total of **12,960 instances** with **8 attributes** and **between 2 and 4 possible values** for each attribute. It is the **largest dataset** analyzed.

Classes distribution:

- not recommended → 33.33%
- recommended → 0.02%
- very recommended → 2.53%
- priority → 32.91%
- special priority → 31.20%

3.3.1 Previous *Rules* Based Method Results

Rules based method obtained a **99.5%** of *Validation Accuracy* over its dataset. Pointing out the following attributes as the most important ones: *Health, Parents, Children*.

3.3.2 Random and Decision Forest Results

As previous analysis, tables 5 and 6 show the **Validation Accuracy** measures obtained by the **Random Forest** for the full combinatory of hyper-parameters, extracting these results from a **50-fold cross-validation** analysis.

Random Forest Accuracy				
		F		
		1	2	3
NT	1	76.4%	94.2%	96.8%
	10	93.5%	98.8%	98.9%
	25	95.8%	99.2%	99.1%
	50	96.8%	99.3%	99.2%
	75	97.0%	99.4%	99.1%
	100	97.0%	99.4%	99.2%
	250	97.4%	99.4%	99.2%
	500	97.3%	99.4%	99.2%

Table 5: *Validation Accuracy* of **Random Forest** for all the hyper-parameter combinatory over *Nursery* dataset. Since $\lfloor \log_2 8 + 1 \rfloor = 3$ these parameters have been not duplicated.

Decision Forest Accuracy					
		F			
		2	4	6	Random(1,8)
NT	1	45.4%	61.2%	73.2%	59.2%
	10	56.6%	75.6%	92.1%	80.3%
	25	58.9%	83.3%	94.3%	86.0%
	50	59.2%	84.2%	95.5%	91.0%
	75	58.6%	87.0%	96.0%	91.5%
	100	58.2%	88.1%	95.9%	92.8%
	250	58.2%	90.5%	96.4%	93.7%
	500	58.2%	91.7%	96.5%	94.2%

Table 6: *Validation Accuracy* of **Decision Forest** for all the hyper-parameter combinatory over *Nursery* dataset.

As both tables shows, the scenario is, one more time, so similar to the one presented by *Cars* dataset (tables 3 and 4). As dataset comes from a **hierarchical decision model** it is prepared for using all its features. These are cases which **Random Forests** are able to cover, but not **Decision Forests**. Therefore, **Decision Forest** table shows how it is impossible for the *Forest* to solve the problem using two Features per tree, and it needs a lot of trees when using four. In fact, an **unique tree** (NT = 1) using **F** = 3 in **Random Forest** approach is able to beat the performance of the best **Decision Forest** configuration.

In terms of *Feature Relevance*, figure 4 summarizes the results extracted from both methods.

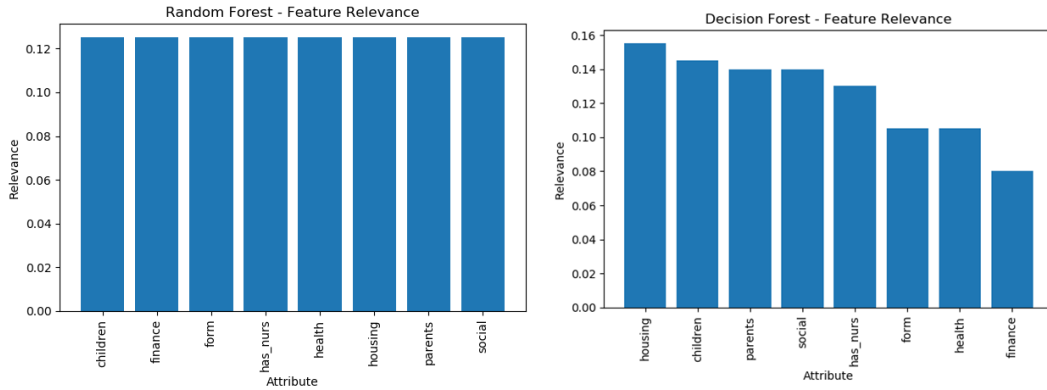


Figure 4: Feature relevance extracted from **Random Forest** and **Decision Forest** methods. These plots have been extracted from examples using NT = 250 in both cases, with **F** = 2 for **Random Forest** and **F** = 6 for **Decision Forest**. In **Random Forest** there was not any case where a different relevance among features was found. **Decision Forest** different experiments presented as variability as before.

As this figure shows, one more time it is impossible for **Random Forest** to find any specially relevant features, while **Decision Forest** presented a high variability, decreasing a lot the confidence that we can have in its results and making them not conclusive.

These results make evident one of the principal differences between the way how *Rules* algorithms and *Forest* extract its *Feature Relevance*. In *Rules* algorithm we could see how, for example, it was possible to deduce the class *Not Recommended* (which have a 33.33% of coverage) only by the attribute *Health*.

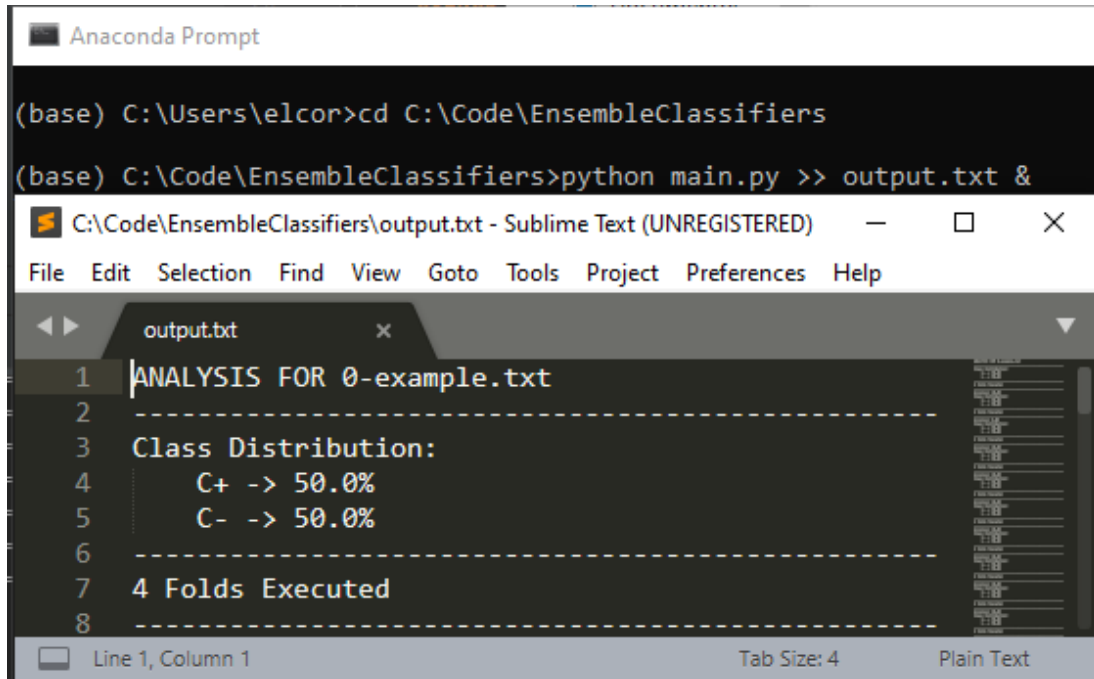
However, we are not able to extract that kind of special *Relevance* neither in **Random Forest** nor **Decision Forest** by the frequency method. The reason is the following one: *Forests* for sure will produce the *Health* node as soon as they can, but then, for the rest of classes they will produce other nodes. As we are measuring the unique frequency, and not its point within the tree hierarchy, all nodes in the tree will have the same weight, so this information will be lost.

In terms of performance, *Rules* algorithm was able to obtain a 99.5% of *Accuracy* while **Random Forest** only arrived to 99.4%. This difference of 0.1% have a very low relevance, however, in absolute terms, *Forests* were **not able to beat** the results of the previous *Rules Classifier*.

4 How to execute the code

Project has been developed using **Python**, and in order to execute it you must have a **Python 3** version installed in your computer as well as **Numpy** library. This implementation will use by default all the *CPU cores* available at your computer for parallelizing the process of generate *Forests*. If you want to reduce the amount of *CPU cores* which it will use, the variable `DEFAULT_THREADS` must be modified from the first lines of file "Source/*Forest.py*".

It is executable through the *main* file, executing `python ./main.py >> output.txt &` on your command prompt **from the project folder**, as figure 5 shows.



The image shows a screenshot of a computer screen. At the top, there is an 'Anaconda Prompt' window. The command prompt shows the following commands and output:

```
(base) C:\Users\elcor>cd C:\Code\EnsembleClassifiers
(base) C:\Code\EnsembleClassifiers>python main.py >> output.txt &
```

Below the command prompt, there is a 'Sublime Text (UNREGISTERED)' window. The file 'output.txt' is open, and its content is displayed as follows:

```
1 ANALYSIS FOR 0-example.txt
2 -----
3 Class Distribution:
4     C+ -> 50.0%
5     C- -> 50.0%
6 -----
7 4 Folds Executed
8 -----
```

The Sublime Text window also shows a menu bar with 'File', 'Edit', 'Selection', 'Find', 'View', 'Goto', 'Tools', 'Project', 'Preferences', and 'Help'. The status bar at the bottom indicates 'Line 1, Column 1', 'Tab Size: 4', and 'Plain Text'.

Figure 5: Example of code execution and its expected output.

4.1 Code Structure

Project is divided in **two** sections:

- **Data folder:** It contains all the datasets that we want to analyze, with *CSV* format. In order to make new visualizations simpler **without modifying** any line of code, those *CSV* should contain

on first row the **names** of each column, being the **objective attribute** named "*class*". All of them will be executed in **alphabetical order** when main is executed. By default it contains **4 datasets**:

- **0-example.csv**: It is the **toy dataset** used as example on the class slides. Can be used as a **test** for compare with the expected results.
- **1-short-votes.csv**, **2-medium-cars.csv** and **3-large-nursery.csv**: The three **analyzed datasets** presented at sections 3.1, 3.2 and 3.3
- **Code**: is divided in the following 5 files:
 - **C45.py**: It is the implementation of the **C4.5** base learner. Contains a class named *C45* which implements the algorithm. It is implemented following the *scikit-learn* standards, with a **fit and predict** as its principal functions. This class is also in charge of defining the **tree structure**, as well as extract the attributes used by it for predicting.
 - **Forest.py**: It is the implementation of the **Random Forest** and **Decision Forest** algorithms. Contains a class named *Forest* which implements both algorithms, selecting by the parameter *forest_type* one or another. It is also implemented following the *scikit-learn* standards. This class is in charge of plotting the *Feature Relevance*, measuring the frequency of each feature in the forest.
 - **Reader.py**: It only contains an **iterable function**. It is in charge of reading a *CSV* file, **dividing** it dynamically in **k training/validation folds**, and yielding at each step one of these **folds**. It also analyzes the **class distribution** over the read dataset.
 - **Analyze.py**: It executes the **complete analysis** of a dataset, showing its **class distribution**, and for each one of the implemented *Forest* algorithm saving at results folder: the table with the **Validation Accuracy** for full combinatorial of attributes and a plot of the **Feature Relevance** extracted from it.
 - **main.py**: It executes the analysis for **all the datasets** in the *Data* folder, and is the one which must be executed when **running** the project.

5 Conclusions

Through this work, **Random Forest** and **Decision Forest** algorithms have been **implemented** and **analyzed**, using **C4.5** trees as Base Learner.

Implementation part has drew the following results:

- **Tree Classifiers** presents usually an easy recursive implementation. Despite that coding sequential implementations could produce an slightly more efficient algorithms, the recursive implementation has been selected, because it is more natural, understandable and readable.
- Since **Forest Classifiers** are composed by a set of independently trained trees, their constructions can be seen as a natural **parallel process**. It allows to parallelize the process computing each tree in a different *CPU* thread, reducing the computational time required.

However, the most relevant apprenticeships comes from the **analysis part**:

- **Random Forest vs Decision Forest**: In **Validation Accuracy** metrics, **Random Forest** tends to always obtain **better performance**, moreover, it allows a wider range of parameters to obtain its **optimal performance**. This is due to its tree generation system which allows to generate stronger trees, since they have access to **all the features** of the set. This base learner strength can be seen in all generated tables, on the row **NT = 1**, where **Random Forest** always obtained better

performance than **Decision Forest**. It is even more remarkable in these cases where the dataset really hides a **hierarchical structure**, like *Cars* or *Nursery* datasets, where it was too difficult for **Decision Forest** to obtain good metrics using only a subset of features at each tree.

- **Regularization:** While **Decision Forest** presents an unique and strong vertical regularization method, based on the features which it uses at each tree, **Random Forest** presents a **double regularization method**. **Random Forest** also takes profit of a weak vertical regularization, allowing to use all features but not always in the same order, but in addition, it also presents a horizontal regularization method through *bootstrap*, never training two *Forests* with the same full training set.
- **Parameterization:** The parameterization of **NT** is clear: The higher the better. However, there is a point where there is no sense in continue increasing **NT**. This point usually depends on the amount of features of the dataset and, in the **Random Forest** case, also on the size of it. Looking on *Accuracy* tables (specially tables 3, 4 and 6) it can be seen how the new added parameters (**NT** = 250 and **NT** = 500) were able to continue improving the performance of the model. This improving of performance as more classifiers are included, is the principle which sustains the theory of the *Ensemble Classifiers* methods.

Parameterization of **F** is a more difficult case. In **Random Forest** case this parameterization is less relevant, since it can obtain pretty similar results using: **3**, $\lfloor \log_2(\mathbf{I} + 1) \rfloor$ or $\lfloor \sqrt{\mathbf{I}} \rfloor$ (Note that **F** = 1 means a totally random selection of attributes by each node). This is due that, unlike **Decision Forest**, it will still being able to generate trees using all features **if needed**. However, In **Decision Forest** the parameterization of **F** should be more accurate, being good values those which are higher, like: $\frac{3*\mathbf{I}}{4}$ or even $\text{Random}(1, \mathbf{I})$ in some cases. When deciding **F** in **Decision Forest** it is important to take a look at $\binom{\mathbf{F}}{\mathbf{I}}$ since it will indicate the maximum **NT** parameter which will make sense.

- **Feature Relevance:** Forest methods are able to extract relevance indexes for each feature, based on how **frequently** they are used among its trees. It is possible in cases where there is a big amount of instances which do not need all the features for being classified, as for example the case shown at *Votes* dataset (figure 2). However, in cases like the one presented by the *Cars* (figure 3) or *Nursery* (figure 4) datasets, where there are no clear winners due that all the features should be used for classifying some instances of the training set, no conclusions can be drawn. In these cases, all the trees within a *Forest* will use all the available features. It will generate the same frequency in **Random Forest**, and practically random frequencies in **Decision Forest**. This *Feature Relevance* extraction could be improved using more aggressive regularization methods like pruning or defining a max depth for trees. However, for real cases where *Feature Relevance* research is the main purpose of the work, **other methodologies** should be used.

References

- [1] Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001).
Available at: <https://link.springer.com/article/10.1023/A:1010933404324>
- [2] Kam Ho, T. The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 8, pp. 832-844, (1998).
Available at: <https://ieeexplore.ieee.org/document/709601>
- [3] Quinlan, J. C4.5: Programs for machine learning. Morgan Kaufmann Publishers, San Mateo (1993)
Available at: <https://link.springer.com/article/10.1007/BF00993309>
- [4] Pierre Trémaux, C. École Polytechnique of Paris. in Public conference, December 2, 2010
- [5] Efronm, B. Bootstrap Methods: Another Look at the Jackknife. The Annals of Statistics, Vol. 7, No. 1 (1979), pp. 1-26.
Available at: https://link.springer.com/chapter/10.1007/978-1-4612-4380-9_40
- [6] Bohanec, M. and Rajkovic, V. Expert system for decision making. Sistemica 1(1), pp. 145-157, 1990.)
Available at: <http://kt.ijs.si/MarkoBohanec/pub/Sistemica90.pdf>