UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Practical Work 2:
Combining Multiple Classifiers

Supervised and Experiential Learning

João Valério

joao.agostinho@estudiantat.upc.edu

04/05/2023

Index

1. INTRODUCTION

The primary objective of this study is to implement, compare, and validate two combinations of multiple classifiers, specifically a random forest and a decision forest, utilizing the CART algorithm as the base-learner for tree induction.

In accordance with this approach, the descriptions outlined in reference [1] serve as the foundation for the development and analysis of the CART algorithm. Additionally, the implementation of random and decision forests adheres to the methodologies presented in references [2] and [3], respectively, in the order specified. The entire project is coded in Python programming language, with the aim of inducing and applying a set of trees to ascertain the overall classification accuracy of the model.

Consequently, the UC Irvine Machine Learning[1] and Kagle[2] repositories were utilized to carefully select three datasets that fulfil the necessary specifications. In order to broaden the scope of the algorithm's evaluation, diverse datasets were chosen, namely Hepatitis and Tic-Tac-Toe Endgame from footnote [1], and Cure The Princess from footnote [2]. These datasets exhibit the following characteristics: Hepatitis is a small dataset containing predominantly categorical data with numerical data, Tic-Tac-Toe Endgame is a medium-sized dataset with only categorical data, and Cure The Princess is a large dataset with solely numerical data. The selection process took into account not only the size limitations of the datasets, but also other characteristics such as class distribution deviation, percentage of instances in majority and minority classes, percentage of missing values, number of numerical and categorical features, and number of classes.

Furthermore, the three aforementioned datasets are subjected to thorough preprocessing to address missing values, varying feature types, and normalization, as per the specified order. Specifically, to prevent ordinal assumptions and potential biases towards numerical values, the data is transformed to exclusively numerical features, even though the algorithms are capable of handling both numerical and categorical data. Additionally, the datasets are divided into training and testing sets, with an 80% and 20% split, respectively. The training set is utilized to obtain the set of trees, while the testing set is used to evaluate the performance of the algorithm.

In order to conduct a comprehensive evaluation of the outcomes achieved by both algorithms across three databases, a table containing accuracy results and an ordered list of features by relevance was generated. The relevance of features was determined based on various hyperparameter combinations tested in this study.

Conclusively, detailed instructions for executing the code and supplementary material sources are furnished, enabling the replication of the present paper.

2. DATA

a. Characteristics

According to both repositories, the datasets selected and the respective characteristics are:

[1] The datasets Hepatitis and Tic-Tac-Toe Endgame acquired from this repository can be found in the following link: https://archive-beta.ics.uci.edu/

[2] The dataset Cure The Princess acquired from this repository can be found in the following link: https://www.kaggle.com/

- **Hepatitis**: small dataset with numerical and, predominantly, categorical data.
- **Tic-Tac-Toe Endgame**: medium dataset only with categorical data.
- **Cure The Princess**: large dataset only with numerical data.

Furthermore, besides the size restrictions, it was intended to obtain diversity between the data sets on the number of features and their properties. The exception is the number of classes, since all of the datasets concern binary classification problems, due to computational resource availability. Finally, the imbalance between class representations on the first 2 datasets pretends to test the generalization capacity of the Forest algorithms.

The attributes mentioned in each dataset are presented in Table 2.a.1.

Table 2.a.1 – Characteristics of the datasets.

	Characteristics of the Datasets							
	Number of Instances	**Numerical Features**	**Nominal Features**	**Number of Classes**	**Deviation of Class Distribution**	**Majority Class**	**Minority Class**	**Missing Values**
Hepatitis	155	6	13	2	29.35%	79.35%	20.65%	6.01% (42.23%*)
Tic-Tac-Toe Endgame	958	-	9	2	16.65%	66.34%	34.66%	-
Cure The Princess	2338	13	-	2	0.50%	50.34%	49.66%	-

* The value in parenthesis indicates that the feature with the highest rate of missing values has a portion of 43 % missing values. This is considered to make a large impact on the result, especially since the data is relatively small.

b. Preprocessing

In order to obtain a proper dataset to feed a supervised machine learning algorithm, it is important to preprocess the data. This domain is separated into five central categories: Data Upload (i.), Missing Values (ii.), Different Types (iii.), Different Ranges (iv.) and Data Splitting (v.), in which the sequential order referred to agrees with the respective code.

i. Data Upload

Regarding the data upload, the preprocessing code is able to read either .csv or .arff files, for code generalisation purposes. Thus, in the folder ‘Data’, Hepatitis is a .arff file, while the remaining are .csv ones.

ii. Missing Values

In the first subject, it is crucial to handle the problem of missing data, where two approaches are possible: deletion or imputation. The only dataset with missing values is Hepatitis, with a rate value of 6.01%. Since the number of cases is only 155, the deletion of observations results

in a considerable loss of information. Thus, it is chosen to implement imputation. Through that, it is expected to keep the information regarding the observations without inserting possible bias into the data.

Numerical Data:

For the numerical data, the considered metric is the K-Nearest Neighbours Imputer, in which each sample's missing values are imputed according to the mean of the k-nearest neighbours considered. As the function considered from the sckit-learn library is optimised to the general cases of numerical imputation, the best parameters among the tested are the default ones. According to that, k assumes a value of 5, with uniform weight distribution between the neighbours.

Categorical Data:

In the case of categorical features, the missing nominal value, '?', is considered as a new feature named 'Unknown', since a wrong categorical attribution could result in poor clustering, due to the transversal effect of the missing values.

iii. Different Types

Thirdly, it is essential to consider the base-learner algorithm (CART) to induce the trees. In this study, the model is able to handle numerical and categorical data. However, in order to avoid ordinal assumptions and bias towards numerical values, it is decided to transform the categorical features into numerical ones.

So, as two (Hepatitis and Tic-Tac-Toe Endgame) of the three datasets are not exclusively numeric, it is necessary to implement a conversion between the categorical features into numerical ones. Thus, the three datasets evolve into solely numerical ones.

The approach considered is One hot encoding, where the nominal variables are converted into a binary vector. A prior disadvantage is the increasing dimensionality of the data, worsening the time and memory complexity. However, by attributing the same weight to all the categorical features, it is avoided the insertion of bias. Therefore, it is expected an increment in the quality of the trees conceived.

Table 2.b.iii.1 exposes the number of features priorly to and after the One hot encoding execution.

Table 2.b.iii.1 – Variation in the number of features.

	Total Number of Features	
	Previously	**After**
Hepatitis	19	42
Tic-Tac-Toe Endgame	9	28

According to the table, it is registered an increment of 23 and 19 features on Hepatitis and Tic-Tac-Toe datasets, respectively.

iv. Different Ranges

Then, the normalisation of all the numerical data is executed. Since different features have distinct numerical ranges, the weights between them are non-uniformly distributed, inserting biased pieces of information in the model. As there is no relevant information pointing out that certain features should have more weight than others, it is implemented a uniform weight distribution along the attributes.

The method considered is Min-Max Scaling, in which each instance has a linear value attribution between 0 (minimum) and 1 (maximum).

v. Data Splitting

Finally, the dataset is split into training and testing, where the set of rules are conceived and assessed, respectively. Thus, the training data constituted 80% of the original dataset, while the testing represented the remaining 20%.

3. ALGORITHM

The CART method is a popular decision tree-based algorithm used for supervised machine learning tasks. It can handle both categorical and continuous input features, recursively partitioning the input space and fitting simple models in each region. CART's flexibility and interpretability make it a powerful base learner for ensemble methods, where multiple trees are combined to improve accuracy. In this work, CART will serve as the foundation for inducing trees, providing a solid base for subsequent ensemble learning.

The impurity measures are conducted according to the Gini index of impurity, in which the smaller the value the greater the separation. Furthermore, the quality of the results obtained by the algorithm will depend on the number of features considered (F), the number of trees (NT) and the depth of the tree (D).

In order to develop the algorithm, the descriptions presented in [1] are considered as the base of its development. Therefore, its correspondent pseudocode for tree acquisition, with the correct order of execution, can be described as follows:

CART:

Input: train_data: Training data.

1. Initialize self.root as None.

2. Create a new Node object and assign it to self.root with the following parameters: data: train_data; y_col: "Class"; depth: 0; F: self.F (number of randomly selected features)

3. Call the make_split() method on self.root with parameter D as self.D to start the recursive splitting process.

3.1. If self.opt_feature is not None, which means the current node is not a leaf node:

a. If self.depth < D, continue with the splitting process, otherwise stop.

b. Get the optimal feature and split point by calling get_opt_split() method on the current node.

c. If optimal feature and split point are not None:

i. Extract the data points that are below the split point and assign it to below.

ii. Extract the data points that are above the split point and assign it to above.

iii. Create a left child node by initializing a new Node object with the following parameters: data: below; y_col: "Class"; depth: self.depth + 1; F: self.F

iv. Create a right child node by initializing a new Node object with the following parameters: data: above; y_col: "Class"; depth: self.depth + 1; F: self.F

v. Call the make_split() method recursively on the left child node with parameter D as D.

vi. Call the make_split() method recursively on the right child node with parameter D as D.

3.2. If self.opt_feature is None, which means the current node is a leaf node:

a. Assign the predicted label (class with maximum counts) to self.predicted_label.

b. Stop the splitting process.

Additionally, Decision and Random Forests are ensemble learning methods that build upon the CART algorithm as their base learner. Decision Forests consist of multiple decision trees that are combined to make predictions, while Random Forests further enhance this approach by adding randomness to the tree-building process. These ensemble methods leverage the strengths of the CART algorithm, which provides the foundation for inducing individual trees, to improve the prediction accuracy and robustness of the resulting model. By combining the outputs of multiple trees, Decision and Random Forests are able to capture complex interactions among input features, handle noisy data, and reduce the risk of overfitting. In this work, we will utilise the power of Decision and Random Forests, building upon the CART base learner, to create a highly accurate and robust prediction model for our task.

The algorithms in this work are developed based on the formal descriptions presented in [2] for Random Forest and [3] for Decision Forest. The pseudocodes, outlining the correct order of execution, are constructed to ensure a rigorous and reproducible approach to algorithm implementation.

Random Forest:

Input: df_train: the training dataset

Output: trees: Trained decision trees

Steps:

1. Initialize an empty list to store the CART models

2. Initialize an empty list to store the trees

3. Loop from 0 to NT:

a. Create a CART model with D and F parameters

b. Append the CART model to CART_models list

c. Perform bootstrapped sampling of the original training set with bootstrap parameter

d. Reset the index of the bootstrapped dataset

e. Train the CART model on the bootstrapped dataset

f. Get the tree from the trained CART model

g. Append the tree to trees list

4. Set the trees list to self.trees attribute

5. Return the trees list: return self.get_trees()

Decision Forest:

Input: df_train - Training data

Output: trees - List of trained decision trees

1. Initialize an empty list to store the trained decision trees

2. For each tree in the number of trees to be built (NT):

a. If F == -1, generate a random integer between 1 and the number of features in the training data and store it as F_aux

b. Else, use the value of F provided during initialization as F_aux

c. Create a CART model with maximum depth D and F_aux as the number of features to consider at each split

d. Choose F_aux number of features randomly from the training data without replacement, including the label column

e. Merge the randomly chosen features with the label column to create a new DataFrame, df_F

f. Train the CART model with df_F to obtain a decision tree

g. Append the trained decision tree to the list of trees

End for

The interpreter, which belongs to the CART class, corresponds to the following pseudocode.

Predict:

Input: df_test: the test dataset

Output: y_pred: the predicted labels for the test dataset

1. Initialize an empty array to store the predicted labels

2. Loop over each sample in the test dataset:

a. Initialize an empty dictionary to store the class votes:

b. Loop over each tree in the forest:

i. Get the prediction from the current tree for the current sample

ii. If the tree prediction is not already in class_votes, add it with a value of 1, otherwise, increment its value by 1

c. Get the class with the highest number of votes

d. Append the majority vote to the y_pred array

3. Return the predicted labels for the entire test dataset

Finally, the implementation is conceived in Python's programming language, through which it is expected to apply a set of trees to the validation/test datasets and obtain the global classification accuracy of each model, as well as the features' importance ranking.

4. ANALYSIS

a. Evaluation Metrics

Firstly, previously to the analysis of the results, it is necessary to clarify the evaluation metrics employed. Regarding the overall algorithm performance, the metrics utilised, in the training and test phases, are the following:

- **Accuracy [%]**: The primary evaluation metric for assessing the performance of the algorithm is accuracy, which measures the proportion of correct predictions. It is worth noting that the number of predictions is equivalent to the number of trees in the algorithm. As such, the final prediction for each row is determined by counting the most frequently predicted class. In the event of a tie, the code's implemented function will always return the index of the first element in the list. Moreover, the accuracy per class is also indicated.
- **Feature Ranking**: Feature ranking is an ordered list of features, sorted by their relevance, which is determined through various combinations of hyper-parameters. The relevance metric is defined based on the following numerical criteria:
    1. **Highest frequency in the trees**: Features that appear more frequently in the decision trees are considered more relevant and are given higher rankings in the feature ranking list.
    2. **Lowest average split point**: Features that result in a lower average split point in the decision trees, indicating better discriminatory power, are considered more relevant and are given higher rankings in the feature ranking list.
    3. **Lowest average depth**: Features that result in a lower average depth of decision tree nodes, indicating simpler and more interpretable trees, are considered more relevant and are given higher rankings in the feature ranking list.

This meticulous process ensures that the feature ranking list accurately reflects the relevance of each feature in the context of the specific hyper-parameter combinations used in the analysis.

Ultimately, the results obtained for accuracy are ultimately documented in the accuracy.txt file, whereas the feature ranking of each combination, along with the corresponding accuracies achieved, are documented in the features_relevance.txt file. Both of these files can be found within the designated Data directory.

b. Hyper-Parameters

Each of the aforementioned metrics is computed for every possible combination of hyper-parameters, which includes:

- **Number of Trees (NT)**: the number of desired trees constructed by the CART algorithm. Each tree in the ensemble is constructed independently and represents a distinct

representation of the underlying data due to the inherent variability in the algorithm's construction process.

- **Number of Features (F)**: refers to the count of randomly selected features that are considered during the node splitting process in a Random Forest or in each individual tree within a Decision Forest. These features are used as candidate split points to partition the data at each node in the tree construction process.
- **Maximum Depth (D)**: denotes the maximum depth or level that a tree can reach during its construction process. It serves as a pre-pruning technique that can be used to control the issue of overfitting. Limiting the depth of the tree can prevent it from becoming overly complex and capturing noise or irrelevant patterns in the data, thereby promoting generalization and mitigating the risk of overfitting.
- **Bootstrap (BS)**: refers to the practice of randomly sampling the training set, a technique that is exclusively applicable to the Random Forest algorithm. This technique can be advantageous in mitigating the risk of overfitting and facilitating improved generalization performance.

Table 4.b.1 indicates the values considered for the combinations on the analysis conducted on topic 4.c.

Table 4.b.1 – Hyper-Parameters.

	Random Forest						**Decision Forest**					
NT	1	10	25	50	75	100	1	10	25	50	75	100
F	1	2	int($\log_2$M+1)	int($\sqrt{M}$)			int(M/4)	int(M/2)	int(3M/4)	Runif(1,M)		
D	2	10	30	100*			2	10	30	100*		
BS	0.3*											

Notes:
M: indicates the number of features in the dataset.
Runif(1,M): a function generating a pseudorandom integer value, ru, such that 1 ≤ ru ≤ M with a uniform distribution probability.
With D=100 no pruning is conducted in any dataset.
* The bootstrap value is fixed due to resource limitations.

In summary, each Forest's algorithm presents a total of 96 distinct potential combinations per dataset, encompassing the various permutations of hyper-parameter settings.

c. Accuracy Analysis

Through the execution of the code provided on the datasets selected, the accuracies were obtained. In that perspective, the analysis of the outcomes regarding each dataset will be conducted individually in (i.), Hepatitis, (ii.), Tic-Tac-Toe Endgame, and (iii.), Cure the Princess. In the event of a tie in terms of accuracy, the model characterized by simplicity is considered the superior option, while the model characterized by complexity is considered the inferior option.

As a final remark, it should be noted that the datasets provided do not differentiate between training and testing data. However, in order to ensure a consistent analysis, the train and test data are evenly distributed with respect to the classes prior to conducting the data split. This approach ensures that the model's performance can be reliably assessed, as it avoids any potential biases in the training and testing data distribution that may affect the correct evaluation of the model.

i. Hepatitis

The accuracies achieved for both Forest's algorithms related to the Hepatitis dataset are presented in table 4.b.i.1.

Table 4.b.i.1 – Accuracy in the Hepatitis dataset.

	Random Forest					
	Depth	**Number of Trees**	**Number of Features**	**Accuracy [%]**	**Accuracy Class 0 [%]**	**Accuracy Class 1 [%]**
Best	2	1	1	90.32	33.33	96.43
Worst	10	1	2	35.48	66.67	32.14
	Decision Forest					
Best	30	50	10	93.55	33.33	100.00
Worst	30	1	10	54.84	33.33	57.14

Upon careful evaluation of the achieved results, it is evident that both of Forest's algorithms have demonstrated an impressive ability to attain sound outcomes. When comparing these outcomes with the results obtained by RULES on paper [4], it becomes apparent that Forest's algorithms possess a notable capacity for extracting crucial patterns from the dataset. Even the most optimal performance of the RULES algorithm fell short of surpassing the interpretability achieved by Forest's algorithms.

The Random Forest algorithm's highest achievement was precisely 90.32% accuracy (33.33% - Class 0; 96.43% - Class 1) using an exceptionally simple tree configuration (2 - Depth; 1 - Number of Trees; 1 - Number of Trees). Remarkably, this result was obtained by 73% of the conceived combinations, equivalent to 70 out of 96 models, indicating the algorithm's ability to quickly learn the underlying data patterns. However, given the extremely small size of the dataset, it may be more plausible to select a more complex set of hyperparameters, from 70 referred models, to allow for increased complexity in making predictions on new data, which is not adequately reflected in the performance of the test set.

On the other hand, the Decision Forest algorithm achieved a slightly higher accuracy of 93.55% (33.33% - Class 0; 100.0% - Class 1) with a significantly more complex set of parameters (30 - Depth; 50 - Number of Trees; 10 - Number of Trees), further supporting the aforementioned

idea. Despite the marginal difference in accuracy and substantial difference in complexity, the Decision Forest model appears to be a more reliable choice for prediction purposes, in which the number of trees holds a significant impact.

Both models, however, exhibit significant challenges in accurately classifying Class 0 instances due to the pronounced class imbalance in the dataset, as demonstrated in Table 2.a.1. In fact, it appears to be challenging for both models to improve the classification of Class 0 without adversely impacting the accuracy of Class 1. Consequently, the models that perform well in classifying Class 0 instances tend to have poorer overall performance.

Therefore, when dealing with unbalanced datasets, special attention may be required to address issues such as overfitting and bias towards the majority class, as observed in this dataset, by directly intervening on the dataset during preprocessing or model training.

ii. Tic-Tac-Toe Endgame

In the Tic-Tac-Toe Endgame dataset the values obtained are in Table 4.b.ii.1.

Table 4.b.ii.1 – Accuracy in the Tic-Tac-Toe Endgame dataset.

	Random Forest					
	Depth	**Number of Trees**	**Number of Features**	**Accuracy [%]**	**Accuracy Class 0 [%]**	**Accuracy Class 1 [%]**
Best	2	75	5	78.13	53.19	86.21
Worst	100	10	4	62.50	36.17	71.03
	Decision Forest					
Best	10	100	20	98.44	93.62	100.00
Worst	10	1	20	67.71	57.45	71.03

When examining the accuracy results for the Tic-Tac-Toe Endgame dataset, it is evident that the difference between the Random Forest and Decision Forest models is more pronounced compared to the results in section 4.b.i. Notably, when comparing these results with those reported in [4], the Decision Forest outperforms the RULE's algorithm, while the Random Forest performs less effectively than the other two. Intriguingly, it is worth mentioning that, across datasets ranging from Hepatitis to Tic-Tac-Toe Endgame, the RULE's algorithm consistently reduces the complexity of the number of rules, whereas the Forest models require a more complex structure to achieve accurate predictions.

In the context of the Decision Forest, it is evident that the number of trees in the ensemble has a significant impact on the final accuracy achieved. Specifically, an increase of 99 trees resulted in a notable improvement of 30.73% in accuracy, with 36.17% for Class 0 and 28.97% for Class 1. On the other hand, in the Random Forest, the discrepancy between the best and worst approaches is 15.63%, with 17.02% for Class 0 and 15.18% for Class 1, which is half of the

latter. This discrepancy is not linearly related to the complexity of the model, due to the inherent randomness of the Random Forest algorithm. Although the inherent randomness of the Random Forest algorithm has the potential to yield favourable results with simple models, it also introduces challenges in the training process.

In terms of the overall results presented in the accuracy.txt file, it is noteworthy that the model exhibits higher ease in classifying instances of class 1. Once again, this discrepancy can be attributed to the class imbalance in the dataset, although the impact is relatively smaller in the Decision Forest compared to the Random Forest. However, it is important to note that the bootstrapping performed in the Random Forest is 30%, which may contribute relevantly to the larger discrepancy in performance.

In conclusion, similar to the previous analysis, the Decision Forest model emerges as the most reliable method, despite its higher complexity compared to the other algorithms.

iii. Cure The Princess

Lastly, in the Cure the Princess dataset, the largest one, the following values were reached.

Table 4.b.iii.1 – Accuracy in the Cure the Princess dataset.

	Random Forest					
	Depth	**Number of Trees**	**Number of Features**	**Accuracy [%]**	**Accuracy Class 0 [%]**	**Accuracy Class 1 [%]**
Best	30	25	3	57.27	88.48	23.56
Worst	10	75	3	45.94	4.53	90.67
	Decision Forest					
Best	100	100	6	74.36	69.14	80.00
Worst	2	100	3	48.08	0.00	100.00

Based on the attained results, the Decision Forest surpasses the Random Forest by a margin of 17% in terms of performance on the best models. However, the worst models exhibit similar values, indicating that the randomness and strong bootstrapping employed to restrict the learning process of the model may be influencing the outcomes. This suggests that the instances in the dataset may be complex and play a crucial role in model performance.

Further analysis reveals that the Decision Forest achieves the best results with the third most complex model, while the Random Forest model is less complex due to the limited perspective of the dataset caused by bootstrapping, which prevents the development of a complex model that can capture the dataset's entire complexity.

Moreover, the accuracy of classifying class 1 is observed to be largely influenced by the number of trees in both Decision and Random Forests, whereas class 0 does not show any clear

correlation with hyperparameters. Notably, the Random Forest demonstrates superior performance in classifying class 0, surpassing the Decision Forest by 19%.

d. Feature Relevance Analysis

The current chapter focuses on the study of feature relevance ranking obtained through the execution of the models. The features_relevance.txt file provides access to the complete set of feature relevance for each combination. However, in this document, only the set of features from the best models concluded in 4.c is illustrated in Appendix A.

i. Hepatitis

Based on the previous analysis, it is evident that the Random Forest model that yielded the best result is characterized by extreme simplicity. Therefore, it only considers the SGOT feature in the nodes, with an average split point of 0.0, indicating the complete purity of the nodes. Moreover, the number of leaves in the tree is only 2.

On the other hand, the Decision Forest model has a much more complex tree with a total of 2006 leaves, divided between Class 0 (626 leaves) and Class 1 (1380 leaves). Among the features, AGE appears to be the most frequently used, with a count of 463. The average split point and average depth values for AGE are 0.4 and 15.0, respectively. Most features have an average split point value of 0.5, but ALK_PHOSPHATE and SGOT have the lowest values of 0.1. Additionally, SPIDERS_Unknown has the lowest average depth value of 4.0, although with a low frequency of 3. Finally, even though many features are considered, AGE, ALK_PHOSPHATE, SGOT, PROTIME, ALBUMIN, and BILIRUBIN are the most frequent and, therefore, the most relevant ones in the prediction process.

ii. Tic-Tac-Toe Endgame

In the analysis of Random Forest feature relevance, Table A.2 reveals that all features have the same average split point, indicating equal importance, as previously noted in [4]. However, middle-middle-square_o stands out with the lowest average depth of 0.5, signifying its central importance in the game. Furthermore, top-left-square_o, top-left-square_x, top-right-square_x, and middle-middle-square_o are the most relevant features, contributing to a total of 93 leaves (Class 0 - 26; Class 1 - 67).

Similar to the Random Forest, the Decision Forest also exhibits an average split point of 0.5 for all features, and middle-middle-square_o has the lowest average depth of 0.0, indicating its significant influence in the game. Notably, bottom-middle-square_x, top-left-square_x, bottom-right-square_x, middle-left-square_x, and middle-right-square_x are identified as the most relevant features following the game's convention, where X is the first to play and has the highest influence on the game's development. The final complexity of the model is weighty since it has 16746 leaves, comprising 7756 leaves for class 0 and 8990 leaves for class 1.

iii. Cure the Princess

In conclusion, two notable patterns are observed in both Forest algorithms. Firstly, the average split point of all features is found to be equal to or lower than 0.3, indicating a high level of feature purity. Secondly, the average depth of features in the trees is generally around half of

the maximum depth defined for the tree, indicating a distributed influence of the features in the decision-making process.

Furthermore, in the Random Forest, the features with higher importance are Phoenix Feather and Fairy Dust, whereas in the Decision Forest, Basilisk Scale, Chimera Fang, and Witch's Brew are the more influential features. This suggests that the lower accuracy of class 1 in the Random Forest could be attributed to the absence of Basilisk Scale, Chimera Fang, and Witch's Brew features. On the other hand, the lower accuracy of class 0 in the Decision Forest may be related to the relatively lower relevance of Phoenix Feather and Fairy Dust compared to the configurations of the Random Forest.

5. CODE

a. Organization

The code developed is organized into 6 main classes:

- **Preprocessing.py**: contains the PREPROCESS class, which consists of the preprocessing of the datasets characterised in Chapter 2.
- **NODE_CLASS.py** and **CART_CLASS.py**: contain the implementation of the CART algorithm as a base learner on tree-induction.
- **RANDOM_FOREST_CLASS.py** and **DECISION_FOREST_CLASS.py**: implementation of Random Forest and Decision Forest algorithms, respectively, considering CART as the base learner on tree-induction. Even though there are coincident methods, it opted to separate completely the implementations, for further distinguished development purposes.
- **main.py**: the main .py file, where the preprocessing, training and testing stages are executed.

The description of each function can be accessed through the .py files, as well as the detailed description of each step of the code.

b. Modules

The necessary modules for the code development and the respective versions are the following: **numpy** - version 1.24.2, **pandas** - version 1.5.3**, scikit_learn** - version 1.2.2, **scipy** - version 1.10.1 and **tabulate** - version 0.9.0. The Python version used is 3.8.8 through PyCharm CE software.

c. Execution

To execute the code through the terminal the following steps should be taken:

1. pip install numpy
2. pip install pandas
3. pip install scikit_learn
4. pip install scipy
5. pip install tabulate
6. python3 /PATH_WHERE_THE_FILE_main.py_IS_INSERTED

Ex: /Users/joaovalerio/Documents/"MAI UPC"/"2 Semester"/SEL/W1/source/main.py

7. /PATH_WHERE_THE_FOLDER_DATA_IS_INSERTED

   Ex: /Users/joaovalerio/Documents/MAI UPC/2 Semester/SEL/W1

From the execution of the code, the output is printed in the terminal. However, for management purposes, the same output can be found on the accuracy.txt and feature_relevance.txt inside the Data and Outputs folders, as stated previously.

6. CONCLUSION

All the initially proposed goals have been successfully achieved, as will be elaborated upon in the conclusion.

In general, it has been observed that higher depths, numbers of trees, and features in the model result in increased memory and time complexities. However, these trade-offs often lead to improved overall performance, with tuning allowing for finding a balance between underfitting and overfitting.

Upon evaluating both implemented forest algorithms, it was evident that Random Forest exhibited greater efficiency, primarily due to the employment of bootstrapping. However, Decision Forest showed more promising results in terms of performance. It is noted, however, that these conclusions cannot be extrapolated to other applications, as they heavily rely on the specific bootstrapping techniques employed and the inherent characteristics of the data.

Furthermore, it was observed that forest algorithms face challenges when dealing with unbalanced datasets, where improving classification accuracy for one class often compromises the performance of the remaining classes.

In conclusion, both Random Forest and Decision Forest algorithms have shown the potential to achieve suitable results for classification processes.

7. BIBLIOGRAPHY

[1] BREIMAN, L.; *et al.* (1984). *CART: Classification and Regression Trees*. USA: International Biometric Society.

[2] BREIMAN, L. (2001). *Random Forests*. Netherlands: Kluwer Academic Publishers.

[3] HO, T.K. (1998). *The Random Subspace Method for Constructing Decision Forests*. IEEE.

[4] VALÉRIO, J. (2023). *Rule-Based Classifier RULES*. Spain: UPC.

A. Appendix

Table A.1 – Accuracy in the Hepatitis dataset.

Random Forest - D=2; NT=1; F=1			
Feature	**Frequency**	**Average Split Point**	**Average Depth**
SGOT	3	0.0	1.0
Class 0	1	N/A	N/A
Class 1	1	N/A	N/A
Class Total	2	N/A	N/A
Decision Forest - D=30; NT=50; F=10			
AGE	463	0.4	15.0
ALK_PHOSPHATE	366	0.1	15.6
SGOT	288	0.1	17.4
PROTIME	222	0.4	17.0
ALBUMIN	180	0.3	16.0
BILIRUBIN	138	0.2	10.5
ASCITES_no	26	0.5	9.0
LIVER_FIRM_no	23	0.5	10.7
SPLEEN_PALPABLE_yes	23	0.5	12.9
SPIDERS_yes	21	0.5	10.6
MALAISE_yes	20	0.5	5.9
HISTOLOGY_no	20	0.5	16.4

STEROID_no	19	0.5	15.7
STEROID_yes	18	0.5	11.2
VARICES_yes	18	0.5	12.6
ANOREXIA_no	17	0.5	9.4
HISTOLOGY_yes	17	0.5	11.3
ANTIVIRALS_no	16	0.5	10.6
MALAISE_no	16	0.5	17.1
VARICES_no	15	0.5	15.9
ASCITES_yes	14	0.5	10.9
SEX_female	13	0.5	7.6
SPIDERS_no	13	0.5	15.8
LIVER_BIG_Unknown	12	0.5	11.2
ANOREXIA_yes	11	0.5	10.2
SPLEEN_PALPABLE_no	11	0.5	13.5
LIVER_BIG_yes	10	0.5	11.2
LIVER_FIRM_yes	10	0.5	12.8
FATIGUE_yes	8	0.5	7.2
LIVER_BIG_no	8	0.5	13.5
LIVER_FIRM_Unknown	6	0.5	10.3
SPLEEN_PALPABLE_Unknown	4	0.5	7.2
SPIDERS_Unknown	3	0.5	4.0

SEX_male	3	0.5	8.7
ANTIVIRALS_yes	3	0.5	16.3
FATIGUE_no	3	0.5	18.0
ASCITES_Unknown	1	0.5	5.0
VARICES_Unknown	1	0.5	20.0
Class 0	626	N/A	N/A
Class 1	1380	N/A	N/A
Class Total	2006	N/A	N/A

Table A.2 – Accuracy in the Tic-Tac-Toe Endgame dataset.

Random Forest - D=2; NT=75; F=5			
Feature	**Frequency**	**Average Split Point**	**Average Depth**
top-left-square_o	107	0.5	1.2
top-left-square_x	96	0.5	1.7
top-right-square_x	93	0.5	1.5
middle-middle-square_o	82	0.5	0.5
bottom-right-square_b	34	0.5	1.9
Class 0	26	N/A	N/A
Class 1	67	N/A	N/A
Class Total	93	N/A	N/A
Decision Forest - D=10; NT=100; F=20			
bottom-middle-square_x	1128	0.5	6.7

top-left-square_x	1106	0.5	6.8
bottom-right-square_x	1087	0.5	6.7
middle-left-square_x	1084	0.5	6.8
middle-right-square_x	1033	0.5	6.5
middle-middle-square_x	935	0.5	6.5
bottom-left-square_x	914	0.5	6.5
top-middle-square_x	870	0.5	6.7
middle-right-square_o	843	0.5	6.9
top-right-square_x	834	0.5	6.7
middle-left-square_o	768	0.5	6.9
middle-left-square_b	733	0.5	7.4
bottom-middle-square_o	726	0.5	7.0
top-middle-square_b	725	0.5	7.6
bottom-right-square_o	716	0.5	6.7
top-middle-square_o	713	0.5	6.8
bottom-middle-square_b	678	0.5	7.6
top-right-square_o	669	0.5	6.8
middle-right-square_b	649	0.5	7.3
top-right-square_b	480	0.5	7.4
top-left-square_o	423	0.5	5.7
bottom-right-square_b	383	0.5	7.5

bottom-left-square_b	382	0.5	7.5
top-left-square_b	378	0.5	7.0
bottom-left-square_o	314	0.5	5.8
middle-middle-square_b	221	0.5	7.6
middle-middle-square_o	80	0.5	0.0
Class 0	7756	N/A	N/A
Class 1	8990	N/A	N/A
Class Total	16746	N/A	N/A

Table A.3 – Accuracy in the Cure the Princess dataset.

Random Forest - D=30; NT=25; F=3			
Feature	**Frequency**	**Average Split Point**	**Average Depth**
Phoenix Feather	548	0.2	17.2
Fairy Dust	406	0.1	16.4
Unicorn Horn	242	0.1	17.4
Class 0	539	N/A	N/A
Class 1	580	N/A	N/A
Class Total	1119	N/A	N/A
Decision Forest - D=100; NT=100; F=6			
Basilisk Scale	11513	0.2	50.6
Chimera Fang	11365	0.2	52.9
Witch's Brew	10870	0.2	51.4

Griffin Claw	9713	0.2	54.5
Fairy Dust	9145	0.2	51.1
Phoenix Feather	8791	0.2	53.4
Goblin Toes	8553	0.2	55.1
Minotaur Horn	8546	0.2	54.4
Kraken Ink	7548	0.3	53.8
Dragon's Blood	7284	0.2	52.1
Troll Hair	7102	0.3	57.7
Unicorn Horn	6774	0.2	53.2
Mermaid Tears	6119	0.2	52.1
Class 0	58214	N/A	N/A
Class 1	53419	N/A	N/A
Class Total	111633	N/A	N/A