Supervised and experiential learning - Work 2

Combining multiple classifiers

Natalia Jakubiak
May 12, 2021

Master in Artificial Intelligence

Universitat Politècnica de Catalunya

Contents

1 Introduction **1**

2 Algorithms **1**
2.1 The base-learner 1
2.2 Decision Forest 2
2.3 Random Forest 2

3 Implementation **2**
3.1 Decision Forest 3
3.2 Random Forest 3
3.3 Main program 6
3.4 Implementation demonstration 6

4 Databases **8**
4.1 Iris dataset 8
4.2 Car evaluation dataset 8
4.3 Nursery dataset 9
4.4 Data preparation methods 9

5 Evaluation **10**

6 Conclusion **15**

7 Execution of the code **16**

1 Introduction

Ensemble methods are techniques that create multiple models and then combine them to produce improved results. Ensemble methods usually produces more accurate solutions than a single model would. In this work two combinations of multiple classifier, a random forest and a decision forest, are implemented, compared and validated .

2 Algorithms

2.1 The base-learner

The base-learner for inducing the trees is the CART (Classification And Regression Tree) method. It is a classification algorithm for building a decision tree based on Gini's impurity index as splitting criterion [1]. CART is a binary tree build by splitting node into two child nodes repeatedly. The algorithm works repeatedly in three steps:

1. Find all possible subsets of all the possible values of each attribute A. Given $V(A) = a1, ..., a_v$, there are $\frac{2^v-2}{2} = 2^{v-1} - 1$ possible subsets. Each subset S_A is a possible binary split of attribute A. For continuous attributes the midpoint between each pair of sorted adjacent values is taken as a possible split-point.

2. Find the node's best split. Among all possible subsets from $Step_1$ find the one, which maximizes the splitting criterion. (The selection of which input variable to use and the specific split or cut-point.)

3. Split the node using best node split from $Step_2$ and repeat from $Step_1$ until stopping criterion is satisfied (after all train instances are processed).

A splitting criterion is based on Gini-index of impurity, which provides an indication of how "pure" the leaf nodes are (how mixed the training data assigned to each node is). It is defined as follows:

$$Gini(X) = 1 - \sum_{i=1}^{k} p^2_{x \in Ci}$$

Where X is set of all instances to be discriminated at each node and k is the number of different labels of the class attribute. The Gini index calculation for each node is weighted by the total number of instances in the parent node. The Gini score for a chosen split point in a binary classification problem is therefore calculated as follows:

$$Gini(X, A) = \frac{|X_1|}{X} * Gini(X_1) + \frac{|X_2|}{X} * Gini(X_2)$$

The attribute which is chosen as the splitting attribute maximizes the reduction in impurity expressed by the equation:

$$\Delta Gini(A) = Gini(X) - Gini(X, A)$$

So to find the best split, a calculation of the weighted sum of Gini Impurity for both child nodes is needed.
In implementation I do this for all possible splits and then take the one with the lowest Gini Impurity as the best split.

2.2 Decision Forest

The decision tree algorithm is quite easy to understand and interpret. But often, a single tree is not sufficient for producing effective results. This is where the ensemble methods come into the picture. The Decision Forest algorithm is a tree-based machine learning classifier that leverages the power of multiple decision trees for making decisions to obtain better predictive performance than could be obtained from any of the constituent decision tree algorithms alone. As the name suggests, it is a "forest" of trees. In the Decision Forest, each node in the decision tree works on a random subset of features (the subset of feature is chosen per tree) to calculate the output. The outputs of individual decision trees are later combined using majority voting technique and the final output is generated. In this work each tree is trained using the same original training set.

2.3 Random Forest

Random forest algorithm is a forest of randomly created decision trees. In this model, large number of relatively uncorrelated trees operating as a committee will outperform any of the individual constituent models. The low correlation between trees is the key. Such trees protect each other from their individual errors. Each node of those decision trees considers only a random subset of a fixed size of attributes for making a split. It forces the classifier to use other attributes for splitting each time and increases the uncorrelatedness of the resulting trees. A strategy to achieve the low correlation between decision trees is also to train each classifier on a different training set. Each training set for each tree is a Bootstrapped Sampling of the original training set. It means that given a training set of size n, the new training sets from the original training set is derived by randomly taking n samples from the training set. By that we get a training set with some duplicate samples and some missing samples. Duplicate samples make the associated classifier prioritize the correct classification of that sample. By introducing randomization into its construction procedure, the variance of the classifier is reduced. To get a final prediction of the ensemble of trees, as for the Decision Forest, majority voting is used.

3 Implementation

This work consists of three python files:

- *decisionforest.py*: implementation of the Decision Forest classifier.
- *randomforest.py*: implementation of the Random Forest classifier.
- *main.py*: main program used to generate experiments.

3.1 Decision Forest

The Decision Forest classifier is implemented as a class named DecisionForest. This class takes two hyper-parameters - *number_of_features, number_of_trees*. After initializing an object, the main function of the algorithm *make_and_test_forest(train_data, test_data)* can be called. This function allows to fit the model with training dataset and the training target values using *plant_forest* function and to assign the labels to a list of observations from test set using *make_prediction_forest* function which returns accuracy score for the dataset, assigned winning labels and outputs of multiple classification.
plant_tree function is iterative based, it repeatedly forms the tree classifier calling *plant_tree()* function. It starts the process of recursion on the training data and let the tree grow using subset of random features (chosen per tree) and *recursive_splitter* function. The best split for each node is chosen by *find_best_split_point()* which iterates over all possible subsets and looks for a min $Gini(X, A)$ value that maximizes splitting criterion as described in subsection 2.1. Moreover the algorithm produce the feature importance list based on $\Delta Gini(A)$. When the selected feature is used to make decision how to divide the data set into two separate sets the $\Delta Gini(A)$ of this split is added to the value of the importance of this splitting attribute. These values are cumulated during the whole process of creating the forest (for each split in each decision tree) and in the last step are normalized. The algorithm returns ordered list of the features according to their importance.

3.2 Random Forest

The class implementation of the Random Forest algorithm inherits from DecisionForest class and differs in just two aspects:

1. The different random subset of fixed max size is chosen for each node in the process of growing the decision tree (in the case of DF the same feature subset for the whole tree).
2. Each decision tree is learned on a bootstrapped sampling of the original training set (not the same original training set as in the case of DF). The bagging method is implemented using buil-in *resample* function from *sklearn.utils* library.

Algorithm 1 Decision Forest

INPUT: S, where S = set of train classified instances
OUTPUT: Decision Forest
Require: S != $\emptyset, num\_attributes > 0$
NT $\leftarrow$ number of trees
NF $\leftarrow$ number of random features

```
procedure PlantForest
    for i=1 to NT do
        F ← NF randomly selected features
        procedure PlantTree
            repeat
                procedure CreateNode
                    minGini ← 0
                    splitA ← null
                    P ← possible binary partitions of F
                    for all A in F do
                        gini ← Gini(X, A)
                        if gini < minGini then
                            minGini ← gini
                            splitA ← A
                        end if
                    end for
                    Partition(S, splitA)
                end procedure
            until all instances classified
        end procedure
    end for
end procedure
```

Algorithm 2 Random Forest

INPUT: S, where S = set of train classified instances
OUTPUT: Decision Forest
Require: S != $\emptyset, num\_attributes > 0$
NT $\leftarrow$ number of trees
NF $\leftarrow$ number of random features

```
procedure PlantForest
    for i=1 to NT do
        D ← randomly sampled S with replacement
        procedure PlantTree
            repeat
                F ← NF randomly selected features
                procedure CreateNode
                    minGini ← 0
                    splitA ← null
                    P ← possible binary partitions of F
                    for all A in F do
                        gini ← Gini(X, A)
                        if gini < minGini then
                            minGini ← gini
                            splitA ← A
                        end if
                    end for
                    Partition(D, splitA)
                end procedure
            until all instances classified
        end procedure
    end for
end procedure
```

3.3 Main program

The main function when executed generates all possible parameters (F - number features used in the splitting of the nodes in RF or in each tree in DF; NT - number of desired trees) combination for DF or RF algorithm. With M being the total number of features, the parameters for Decision Forest take values:

- NT = 1, 10, 25, 50, 75, 100
- F = int($\frac{M}{4}$), int($\frac{M}{2}$, $\frac{3*M}{4}$, $Runif(1, M)$)

Parameters for Random Forest are as follows:

- NT = 1, 10, 25, 50, 75, 100
- F = 1, 3, $\log_2 M + 1$, $\sqrt{M}$

For each combination of the parameters main function create the object of the chosen algorithm and execute the *make_and_test_forest* function - 24 executions for one dataset. For each execution it gets achieved accuracy and the ordered list of the features used in the forest, according to its importance. Finally, this function generates a Data-frame with all the combinations and its metrics and store it in a .csv file DatasetName_AlgorithmNames.csv

3.4 Implementation demonstration

Besides the datasets used to evaluate the prediction results of the implemented algorithms, presented later in this report, I also used additional dataset from lecture's slides for debugging purposes: human-identify dataset table 1.

Eye	Hair	Height	class
Blue	Blonde	Tall	C+
Blue	Brown	Medium	C+
Brown	Brown	Medium	C-
Green	Brown	Medium	C-
Green	Brown	Tall	C+
Brown	Brown	Low	C-
Green	Blonde	Low	C-
Blue	Brown	Medium	C+

Table 1: Human identification problem

I would like to demonstrate the output values of derived single tree. For Decision Forest classifier trained on the whole dataset from table 1 the following tree was obtained:

```
{'column_id': 0,                # id of splitting attribute
  'column_name': 'Eye',         # name of splitting attribute
  'type': 0,                    # type: {0: categorical, 1: continuous}
  'dsplit_value': [['Blue'], ['Green', 'Brown']], # binary split
  'gini': 0.19999999999999999,
  'left': 'C+',                 # assigned class
  'right': {'column_id': 2,
   'column_name': 'Height',
   'type': 0,
   'dsplit_value': [['Tall'], ['Medium', 'Low']],
   'gini': 0.0,
   'left': 'C+',
   'right': 'C-'}
```

As can be noticed, for the first node of decision tree, the best split is for the Eye attribute [blue]+[green, brown]. All people with blue eyes are immediately classified to class C+. People with green and brown eyes are divided due to the Height value [Tall]+[Medium, Low]. Tall people are classified as C+ and medium and low people as C-.

The output for feature importance is:

```
[('Height', 0.516), ('Eye', 0.484), ('Hair', 0.0)]
```

Feature importance refers to the techniques for assigning scores to input features to a predictive model that indicates the relative importance of each feature when making a prediction. The score achieved for Eye attribute is lower than one for Height, because Eye splitting point was not enough to split the original dataset exactly into two classes. For height attribute the further splitting was not needed, because all instances were already classified. Hair attribute importance has value equals to 0, because this feature was not chosen as a splitting point for creating a partitions at any node.

4 Databases

To analyze behavior of implemented algorithm I decided to use following datasets from from UCI ML: iris dataset, car dataset and nursery dataset. Details about them are presented in the table 2.

Name of dataset	Size	Type of attributes	Number of instances	Number of Attributes	Number of classes	Missing values
Iris	Small	Real	150	4	3	No
Car	Medium	Categorical	1728	6	4	No
Nursery	Large	Categorical	12960	8	5	No

Table 2: Details of selected datasets

4.1 Iris dataset

This data set is perhaps the best known database to be found in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant: Irist Setosa, Iris Versicolour, Iris Virginica. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.
Attributes:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm

4.2 Car evaluation dataset

Car Evaluation Database was created to evaluate cars according to chosen parameters. Test subjects were asked about their opinion about certain cars. Simple features like safety, number of doors, etc. were extracted from these cars which can be used to predict the subjects opinion.

Class Values: unacc, acc, good, vgood
Attributes:

- buying: vhigh, high, med, low.
- maint: vhigh, high, med, low.
- doors: 2, 3, 4, 5more.
- persons: 2, 4, more.
- lug_boot: small, med, big.
- safety: low, med, high.

4.3 Nursery dataset

Nursery Database was derived from a hierarchical decision model originally developed to rank applications for nursery schools. The final decision depended on three subproblems: occupation of parents and child's nursery, family structure and financial standing, and social and health picture of the family.
Class Values:not_recom, recommend, very_recom, priority, spec_prior.
Attributes:

- parents: usual, pretentious, great_pret
- has_nurs: proper, less_proper, improper, critical, very_crit
- form: complete, completed, incomplete, foster
- children: 1, 2, 3, more
- housing: convenient, less_conv, critical
- finance: convenient, inconv
- social: non-prob, slightly_prob, problematic
- health: recommended, priority, not_recom

4.4 Data preparation methods

In the preprocessing step the dataset is splitted into train and test sets. For all datasets 80% of the examples were used for training classifier, so the remaining 20% could be used for testing.

5 Evaluation

Each algorithm was run on each dataset for each combination of values of F and NT (24 combinations for one dataset).
Decision Forest:

- NT = 1, 10, 25, 50, 75, 100
- F = int($\frac{M}{4}$), int($\frac{M}{2}$, $\frac{3*M}{4}$, $Runif(1, M)$)

Random Forest:

- NT = 1, 10, 25, 50, 75, 100
- F = 1, 3, $\log_2 M + 1$, $\sqrt{M}$

The following tables contain the achieved results - accuracy and the ordered list of the features used in the forest, according to its importance. The column name *1* indicates the most important feature.

NT	F	NF	Accuracy	1	2	3	4
1	M/4	1	0.733	SepalLengthCm: 1.0	SepalWidthCm: 0.0	PetalLengthCm: 0.0	PetalWidthCm: 0.0
1	M/2	2	0.933	PetalWidthCm: 0.503	SepalLengthCm: 0.497	SepalWidthCm: 0.0	PetalLengthCm: 0.0
1	3*M/4	3	1.000	PetalWidthCm: 0.555	SepalWidthCm: 0.269	PetalLengthCm: 0.176	SepalLengthCm: 0.0
1	Runif(1,M)	uniform	0.967	PetalWidthCm: 0.725	SepalWidthCm: 0.275	SepalLengthCm: 0.0	PetalLengthCm: 0.0
10	M/4	1	1.000	PetalLengthCm: 0.389	PetalWidthCm: 0.324	SepalWidthCm: 0.287	SepalLengthCm: 0.0
10	M/2	2	0.867	SepalWidthCm: 0.428	SepalLengthCm: 0.318	PetalWidthCm: 0.159	PetalLengthCm: 0.095
10	3*M/4	3	1.000	PetalWidthCm: 0.377	PetalLengthCm: 0.303	SepalWidthCm: 0.181	SepalLengthCm: 0.14
10	Runif(1,M)	uniform	1.000	PetalWidthCm: 0.376	SepalLengthCm: 0.27	PetalLengthCm: 0.244	SepalWidthCm: 0.11
25	M/4	1	1.000	PetalLengthCm: 0.365	SepalLengthCm: 0.294	PetalWidthCm: 0.228	SepalWidthCm: 0.112
25	M/2	2	0.967	SepalWidthCm: 0.306	SepalLengthCm: 0.285	PetalWidthCm: 0.239	PetalLengthCm: 0.17
25	3*M/4	3	1.000	PetalWidthCm: 0.326	PetalLengthCm: 0.287	SepalWidthCm: 0.216	SepalLengthCm: 0.17
25	Runif(1,M)	uniform	1.000	SepalWidthCm: 0.304	PetalWidthCm: 0.283	SepalLengthCm: 0.238	PetalLengthCm: 0.176
50	M/4	1	0.900	PetalLengthCm: 0.417	SepalLengthCm: 0.403	SepalWidthCm: 0.115	PetalWidthCm: 0.065
50	M/2	2	0.967	PetalWidthCm: 0.314	SepalLengthCm: 0.278	PetalLengthCm: 0.256	SepalWidthCm: 0.152
50	3*M/4	3	1.000	PetalWidthCm: 0.395	SepalLengthCm: 0.216	PetalLengthCm: 0.215	SepalWidthCm: 0.174
50	Runif(1,M)	uniform	1.000	SepalLengthCm: 0.277	PetalWidthCm: 0.272	PetalLengthCm: 0.248	SepalWidthCm: 0.203
75	M/4	1	0.900	SepalLengthCm: 0.416	PetalLengthCm: 0.336	SepalWidthCm: 0.131	PetalWidthCm: 0.117
75	M/2	2	0.967	SepalWidthCm: 0.301	SepalLengthCm: 0.247	PetalWidthCm: 0.239	PetalLengthCm: 0.214
75	3*M/4	3	1.000	PetalWidthCm: 0.287	PetalLengthCm: 0.265	SepalLengthCm: 0.24	SepalWidthCm: 0.208
75	Runif(1,M)	uniform	1.000	PetalWidthCm: 0.263	PetalLengthCm: 0.259	SepalWidthCm: 0.24	SepalLengthCm: 0.238
100	M/4	1	0.933	PetalLengthCm: 0.403	SepalLengthCm: 0.32	PetalWidthCm: 0.15	SepalWidthCm: 0.127
100	M/2	2	0.967	SepalWidthCm: 0.32	SepalLengthCm: 0.301	PetalWidthCm: 0.193	PetalLengthCm: 0.186
100	3*M/4	3	1.000	PetalWidthCm: 0.279	PetalLengthCm: 0.268	SepalWidthCm: 0.235	SepalLengthCm: 0.218
100	Runif(1,M)	uniform	1.000	SepalLengthCm: 0.278	PetalWidthCm: 0.262	PetalLengthCm: 0.25	SepalWidthCm: 0.21

Table 3: Output of DF algorithm for Iris dataset

NT	F	NF	Accuracy	1	2	3	4
1	1	1	0.933	PetalWidthCm: 0.416	SepalLengthCm: 0.405	SepalWidthCm: 0.122	PetalLengthCm: 0.056
1	3	3	1.000	PetalWidthCm: 0.377	SepalLengthCm: 0.273	SepalWidthCm: 0.194	PetalLengthCm: 0.155
1	log2(M+1)	3	0.967	SepalLengthCm: 0.402	PetalWidthCm: 0.362	PetalLengthCm: 0.197	SepalWidthCm: 0.04
1	sqrt(M)	2	0.900	SepalLengthCm: 0.319	PetalWidthCm: 0.311	PetalLengthCm: 0.254	SepalWidthCm: 0.116
10	1	1	1.000	PetalLengthCm: 0.378	PetalWidthCm: 0.27	SepalWidthCm: 0.204	SepalLengthCm: 0.148
10	3	3	1.000	PetalWidthCm: 0.347	SepalLengthCm: 0.237	PetalLengthCm: 0.226	SepalWidthCm: 0.19
10	log2(M+1)	3	1.000	PetalWidthCm: 0.421	PetalLengthCm: 0.208	SepalLengthCm: 0.198	SepalWidthCm: 0.173
10	sqrt(M)	2	1.000	PetalWidthCm: 0.318	PetalLengthCm: 0.264	SepalWidthCm: 0.216	SepalLengthCm: 0.201
25	1	1	1.000	PetalWidthCm: 0.34	PetalLengthCm: 0.283	SepalWidthCm: 0.194	SepalLengthCm: 0.184
25	3	3	1.000	PetalWidthCm: 0.361	PetalLengthCm: 0.27	SepalWidthCm: 0.192	SepalLengthCm: 0.177
25	log2(M+1)	3	1.000	PetalWidthCm: 0.353	PetalLengthCm: 0.311	SepalWidthCm: 0.181	SepalLengthCm: 0.155
25	sqrt(M)	2	1.000	PetalWidthCm: 0.33	PetalLengthCm: 0.287	SepalWidthCm: 0.224	SepalLengthCm: 0.159
50	1	1	1.000	PetalLengthCm: 0.317	PetalWidthCm: 0.317	SepalWidthCm: 0.186	SepalLengthCm: 0.181
50	3	3	1.000	PetalWidthCm: 0.38	PetalLengthCm: 0.221	SepalWidthCm: 0.205	SepalLengthCm: 0.193
50	log2(M+1)	3	1.000	PetalWidthCm: 0.338	PetalLengthCm: 0.267	SepalWidthCm: 0.2	SepalLengthCm: 0.195
50	sqrt(M)	2	1.000	PetalWidthCm: 0.31	PetalLengthCm: 0.287	SepalWidthCm: 0.226	SepalLengthCm: 0.177
75	1	1	1.000	PetalLengthCm: 0.321	PetalWidthCm: 0.312	SepalLengthCm: 0.185	SepalWidthCm: 0.181
75	3	3	1.000	PetalWidthCm: 0.372	PetalLengthCm: 0.225	SepalWidthCm: 0.209	SepalLengthCm: 0.194
75	log2(M+1)	3	1.000	PetalWidthCm: 0.36	PetalLengthCm: 0.236	SepalLengthCm: 0.207	SepalWidthCm: 0.198
75	sqrt(M)	2	1.000	PetalWidthCm: 0.307	PetalLengthCm: 0.293	SepalWidthCm: 0.217	SepalLengthCm: 0.183
100	1	1	1.000	PetalWidthCm: 0.336	PetalLengthCm: 0.313	SepalLengthCm: 0.183	SepalWidthCm: 0.168
100	3	3	1.000	PetalWidthCm: 0.377	PetalLengthCm: 0.235	SepalWidthCm: 0.197	SepalLengthCm: 0.191
100	log2(M+1)	3	0.967	PetalWidthCm: 0.35	PetalLengthCm: 0.251	SepalWidthCm: 0.206	SepalLengthCm: 0.193
100	sqrt(M)	2	1.000	PetalWidthCm: 0.337	PetalLengthCm: 0.285	SepalWidthCm: 0.202	SepalLengthCm: 0.176

Table 4: Output of RF algorithm for Iris dataset

NT	F	NF	Accuracy	1	2	3	4	5	6
1	M/4	1	0.685	persons: 1.0	buying: 0.0	maint: 0.0	doors: 0.0	lug_boot: 0.0	safety: 0.0
1	M/2	3	0.685	buying: 0.444	maint: 0.29	lug_boot: 0.266	doors: 0.0	persons: 0.0	safety: 0.0
1	3*M/4	4	0.783	buying: 0.279	persons: 0.251	lug_boot: 0.244	safety: 0.225	maint: 0.0	doors: 0.0
1	Runif(1,M)	uniform	0.772	doors: 0.35	buying: 0.236	lug_boot: 0.204	maint: 0.129	safety: 0.081	persons: 0.0
10	M/4	1	0.685	safety: 0.786	buying: 0.094	lug_boot: 0.061	maint: 0.06	doors: 0.0	persons: 0.0
10	M/2	3	0.746	safety: 0.255	buying: 0.234	persons: 0.213	maint: 0.174	lug_boot: 0.096	doors: 0.028
10	3*M/4	4	0.815	persons: 0.243	doors: 0.237	buying: 0.205	maint: 0.142	safety: 0.105	lug_boot: 0.067
10	Runif(1,M)	uniform	0.691	doors: 0.345	lug_boot: 0.194	buying: 0.189	safety: 0.144	maint: 0.095	persons: 0.033
25	M/4	1	0.685	safety: 0.46	maint: 0.174	buying: 0.165	persons: 0.133	lug_boot: 0.044	doors: 0.023
25	M/2	3	0.697	safety: 0.257	buying: 0.224	persons: 0.208	maint: 0.126	doors: 0.11	lug_boot: 0.075
25	3*M/4	4	0.876	buying: 0.218	persons: 0.217	safety: 0.159	doors: 0.15	maint: 0.143	lug_boot: 0.113
25	Runif(1,M)	uniform	0.705	doors: 0.262	persons: 0.262	lug_boot: 0.145	buying: 0.12	maint: 0.112	safety: 0.097
50	M/4	1	0.685	safety: 0.489	persons: 0.298	maint: 0.1	buying: 0.088	doors: 0.015	lug_boot: 0.011
50	M/2	3	0.702	persons: 0.296	safety: 0.25	buying: 0.155	maint: 0.111	lug_boot: 0.111	doors: 0.077
50	3*M/4	4	0.803	persons: 0.267	safety: 0.161	buying: 0.159	maint: 0.159	doors: 0.153	lug_boot: 0.101
50	Runif(1,M)	uniform	0.691	doors: 0.267	persons: 0.23	lug_boot: 0.173	buying: 0.167	safety: 0.084	maint: 0.079
75	M/4	1	0.685	safety: 0.475	persons: 0.241	buying: 0.121	maint: 0.117	lug_boot: 0.03	doors: 0.016
75	M/2	3	0.702	safety: 0.278	persons: 0.262	buying: 0.174	maint: 0.118	lug_boot: 0.09	doors: 0.077
75	3*M/4	4	0.824	persons: 0.216	buying: 0.212	doors: 0.176	safety: 0.143	maint: 0.129	lug_boot: 0.124
75	Runif(1,M)	uniform	0.697	persons: 0.285	doors: 0.21	buying: 0.17	lug_boot: 0.145	maint: 0.098	safety: 0.092
100	M/4	1	0.685	persons: 0.507	safety: 0.375	maint: 0.047	buying: 0.041	lug_boot: 0.018	doors: 0.011
100	M/2	3	0.688	persons: 0.268	buying: 0.219	safety: 0.204	maint: 0.136	doors: 0.091	lug_boot: 0.083
100	3*M/4	4	0.803	persons: 0.212	buying: 0.194	doors: 0.169	safety: 0.149	maint: 0.148	lug_boot: 0.128
100	Runif(1,M)	uniform	0.697	persons: 0.261	doors: 0.245	buying: 0.194	lug_boot: 0.119	maint: 0.096	safety: 0.084

Table 5: Output of DF algorithm for Car dataset

NT	F	NF	Accuracy	1	2	3	4	5	6
1	1	1	0.665	safety: 0.611	persons: 0.248	doors: 0.083	buying: 0.022	maint: 0.022	lug_boot: 0.015
1	3	3	0.951	buying: 0.293	maint: 0.244	persons: 0.193	doors: 0.136	lug_boot: 0.108	safety: 0.026
1	log2(M+1)	3	0.942	persons: 0.362	doors: 0.269	buying: 0.158	lug_boot: 0.109	maint: 0.068	safety: 0.035
1	sqrt(M)	2	0.879	buying: 0.407	lug_boot: 0.342	persons: 0.137	doors: 0.052	maint: 0.032	safety: 0.03
10	1	1	0.754	safety: 0.321	maint: 0.253	persons: 0.196	buying: 0.16	lug_boot: 0.053	doors: 0.017
10	3	3	0.960	safety: 0.208	lug_boot: 0.184	buying: 0.183	persons: 0.181	doors: 0.171	maint: 0.072
10	log2(M+1)	3	0.968	lug_boot: 0.248	buying: 0.227	persons: 0.156	safety: 0.151	doors: 0.142	maint: 0.077
10	sqrt(M)	2	0.951	safety: 0.375	persons: 0.184	lug_boot: 0.149	buying: 0.126	maint: 0.095	doors: 0.071
25	1	1	0.798	persons: 0.298	safety: 0.245	buying: 0.182	maint: 0.158	lug_boot: 0.075	doors: 0.042
25	3	3	0.954	maint: 0.208	persons: 0.182	lug_boot: 0.175	buying: 0.171	doors: 0.151	safety: 0.113
25	log2(M+1)	3	0.957	lug_boot: 0.235	persons: 0.18	maint: 0.169	doors: 0.158	safety: 0.135	buying: 0.123
25	sqrt(M)	2	0.957	safety: 0.242	maint: 0.19	lug_boot: 0.174	buying: 0.152	persons: 0.151	doors: 0.09
50	1	1	0.803	safety: 0.317	persons: 0.211	maint: 0.195	buying: 0.163	lug_boot: 0.082	doors: 0.031
50	3	3	0.957	lug_boot: 0.2	persons: 0.191	buying: 0.166	doors: 0.16	safety: 0.146	maint: 0.138
50	log2(M+1)	3	0.960	buying: 0.216	lug_boot: 0.195	persons: 0.164	doors: 0.152	safety: 0.147	maint: 0.126
50	sqrt(M)	2	0.951	safety: 0.237	persons: 0.175	buying: 0.174	lug_boot: 0.172	maint: 0.142	doors: 0.099
75	1	1	0.786	safety: 0.303	persons: 0.29	buying: 0.157	maint: 0.143	lug_boot: 0.07	doors: 0.037
75	3	3	0.960	lug_boot: 0.214	persons: 0.171	maint: 0.163	safety: 0.158	doors: 0.149	buying: 0.145
75	log2(M+1)	3	0.962	lug_boot: 0.205	safety: 0.177	persons: 0.165	buying: 0.154	doors: 0.152	maint: 0.148
75	sqrt(M)	2	0.954	safety: 0.221	buying: 0.204	lug_boot: 0.171	persons: 0.161	maint: 0.156	doors: 0.088
100	1	1	0.743	safety: 0.356	persons: 0.24	buying: 0.179	maint: 0.11	lug_boot: 0.083	doors: 0.03
100	3	3	0.962	lug_boot: 0.194	persons: 0.174	safety: 0.166	maint: 0.16	doors: 0.154	buying: 0.152
100	log2(M+1)	3	0.962	lug_boot: 0.231	maint: 0.168	safety: 0.164	persons: 0.157	doors: 0.145	buying: 0.136
100	sqrt(M)	2	0.960	safety: 0.286	buying: 0.176	persons: 0.165	maint: 0.159	lug_boot: 0.133	doors: 0.082

Table 6: Output of RF algorithm for Car dataset

NT	F	NF	Accuracy	1	2	3	4	5	6	7	8
1	M/4	2	0.491	has_nurs: 0.875	form: 0.125	parents: 0.0	children: 0.0	housing: 0.0	finance: 0.0	social: 0.0	health: 0.0
1	M/2	4	0.777	health: 0.59	social: 0.218	form: 0.098	parents: 0.095	has_nurs: 0.0	children: 0.0	housing: 0.0	finance: 0.0
1	3*M/4	6	0.912	finance: 0.273	form: 0.229	housing: 0.171	has_nurs: 0.144	health: 0.112	parents: 0.071	children: 0.0	social: 0.0
1	Runif(1,M)	uniform	0.367	children: 0.371	form: 0.341	social: 0.167	housing: 0.098	parents: 0.022	has_nurs: 0.0	finance: 0.0	health: 0.0
10	M/4	2	0.620	health: 0.704	has_nurs: 0.166	parents: 0.06	housing: 0.032	social: 0.014	form: 0.012	finance: 0.012	children: 0.0
10	M/2	4	0.529	health: 0.294	parents: 0.145	has_nurs: 0.123	finance: 0.112	form: 0.099	social: 0.092	housing: 0.089	children: 0.047
10	3*M/4	6	0.922	form: 0.244	children: 0.177	finance: 0.151	housing: 0.123	has_nurs: 0.106	social: 0.088	health: 0.075	parents: 0.037
10	Runif(1,M)	uniform	0.734	finance: 0.172	housing: 0.166	children: 0.147	social: 0.146	health: 0.11	form: 0.109	has_nurs: 0.108	parents: 0.041
25	M/4	2	0.542	health: 0.472	has_nurs: 0.311	parents: 0.103	housing: 0.037	social: 0.037	children: 0.03	form: 0.007	finance: 0.004
25	M/2	4	0.868	health: 0.353	has_nurs: 0.177	children: 0.114	parents: 0.089	social: 0.074	finance: 0.067	form: 0.066	housing: 0.06
25	3*M/4	6	0.898	form: 0.275	children: 0.237	social: 0.138	has_nurs: 0.095	finance: 0.081	housing: 0.077	health: 0.054	parents: 0.043
25	Runif(1,M)	uniform	0.919	form: 0.279	finance: 0.169	children: 0.163	social: 0.126	housing: 0.092	has_nurs: 0.082	health: 0.046	parents: 0.044
50	M/4	2	0.596	health: 0.718	has_nurs: 0.145	parents: 0.058	children: 0.024	social: 0.021	housing: 0.014	finance: 0.011	form: 0.009
50	M/2	4	0.740	health: 0.313	has_nurs: 0.22	housing: 0.116	parents: 0.095	form: 0.09	children: 0.071	finance: 0.053	social: 0.042
50	3*M/4	6	0.952	children: 0.212	form: 0.187	housing: 0.131	has_nurs: 0.12	social: 0.114	finance: 0.112	health: 0.075	parents: 0.048
50	Runif(1,M)	uniform	0.765	form: 0.253	children: 0.17	finance: 0.156	social: 0.142	has_nurs: 0.105	housing: 0.078	parents: 0.053	health: 0.042
75	M/4	2	0.613	health: 0.655	has_nurs: 0.182	parents: 0.088	social: 0.025	housing: 0.021	children: 0.012	form: 0.011	finance: 0.006
75	M/2	4	0.878	health: 0.314	has_nurs: 0.187	children: 0.128	form: 0.109	housing: 0.088	social: 0.068	parents: 0.067	finance: 0.04
75	3*M/4	6	0.952	form: 0.244	children: 0.212	social: 0.125	has_nurs: 0.111	finance: 0.107	housing: 0.105	health: 0.053	parents: 0.044
75	Runif(1,M)	uniform	0.811	form: 0.295	children: 0.217	social: 0.137	has_nurs: 0.099	finance: 0.094	housing: 0.083	parents: 0.056	health: 0.019
100	M/4	2	0.568	health: 0.662	has_nurs: 0.174	parents: 0.07	children: 0.024	housing: 0.024	social: 0.023	form: 0.015	finance: 0.008
100	M/2	4	0.872	health: 0.352	has_nurs: 0.166	housing: 0.115	children: 0.098	form: 0.073	finance: 0.067	parents: 0.065	social: 0.063
100	3*M/4	6	0.963	form: 0.23	children: 0.194	finance: 0.125	housing: 0.12	has_nurs: 0.115	social: 0.114	health: 0.058	parents: 0.044
100	Runif(1,M)	uniform	0.905	form: 0.231	children: 0.211	social: 0.121	finance: 0.117	housing: 0.113	has_nurs: 0.093	parents: 0.073	health: 0.042

Table 7: Output of DF algorithm for Nursery dataset

NT	F	NF	Accuracy	1	2	3	4	5	6	7	8
1	1	1	0.562	health: 0.705	parents: 0.201	has_nurs: 0.042	housing: 0.039	social: 0.006	children: 0.004	form: 0.002	finance: 0.0
1	3	3	0.965	has_nurs: 0.323	parents: 0.16	social: 0.134	form: 0.129	children: 0.088	housing: 0.059	finance: 0.058	health: 0.049
1	log2(M+1)	4	0.978	children: 0.31	form: 0.266	housing: 0.157	social: 0.125	finance: 0.075	health: 0.047	parents: 0.011	has_nurs: 0.009
1	sqrt(M)	2	0.861	health: 0.55	housing: 0.158	form: 0.094	has_nurs: 0.063	parents: 0.042	children: 0.037	finance: 0.028	social: 0.027
10	1	1	0.732	health: 0.599	parents: 0.147	has_nurs: 0.118	finance: 0.038	social: 0.032	housing: 0.029	form: 0.02	children: 0.016
10	3	3	0.989	health: 0.209	form: 0.161	children: 0.155	housing: 0.141	finance: 0.09	social: 0.09	parents: 0.084	has_nurs: 0.07
10	log2(M+1)	4	0.995	form: 0.21	children: 0.192	housing: 0.19	finance: 0.116	has_nurs: 0.109	social: 0.076	health: 0.069	parents: 0.037
10	sqrt(M)	2	0.951	health: 0.263	has_nurs: 0.231	parents: 0.139	children: 0.085	form: 0.083	housing: 0.076	finance: 0.063	social: 0.06
25	1	1	0.894	health: 0.433	has_nurs: 0.215	parents: 0.178	housing: 0.051	social: 0.044	children: 0.033	finance: 0.025	form: 0.02
25	3	3	0.994	health: 0.234	form: 0.147	housing: 0.132	has_nurs: 0.129	children: 0.117	parents: 0.092	finance: 0.086	social: 0.062
25	log2(M+1)	4	0.997	form: 0.245	children: 0.175	housing: 0.148	finance: 0.122	has_nurs: 0.093	health: 0.077	parents: 0.076	social: 0.066
25	sqrt(M)	2	0.978	health: 0.43	has_nurs: 0.153	parents: 0.111	housing: 0.075	form: 0.064	children: 0.064	social: 0.059	finance: 0.044
50	1	1	0.902	health: 0.521	has_nurs: 0.19	parents: 0.126	housing: 0.047	children: 0.037	social: 0.029	form: 0.028	finance: 0.022
50	3	3	0.995	health: 0.164	has_nurs: 0.158	form: 0.149	children: 0.129	parents: 0.124	housing: 0.117	finance: 0.079	social: 0.079
50	log2(M+1)	4	0.998	form: 0.224	children: 0.169	housing: 0.16	finance: 0.126	health: 0.106	parents: 0.077	has_nurs: 0.069	social: 0.069
50	sqrt(M)	2	0.972	health: 0.338	has_nurs: 0.176	parents: 0.132	housing: 0.089	form: 0.078	children: 0.067	social: 0.065	finance: 0.055
75	1	1	0.853	health: 0.56	has_nurs: 0.206	parents: 0.109	housing: 0.04	children: 0.025	finance: 0.021	social: 0.02	form: 0.019
75	3	3	0.995	health: 0.199	form: 0.157	children: 0.131	housing: 0.117	has_nurs: 0.112	parents: 0.109	finance: 0.093	social: 0.083
75	log2(M+1)	4	0.996	form: 0.219	children: 0.183	housing: 0.143	finance: 0.12	has_nurs: 0.102	parents: 0.082	social: 0.08	health: 0.072
75	sqrt(M)	2	0.972	health: 0.405	has_nurs: 0.164	parents: 0.124	housing: 0.071	form: 0.067	children: 0.063	finance: 0.054	social: 0.053
100	1	1	0.904	health: 0.526	has_nurs: 0.223	parents: 0.111	housing: 0.034	social: 0.031	children: 0.028	form: 0.025	finance: 0.022
100	3	3	0.995	health: 0.229	form: 0.148	has_nurs: 0.123	housing: 0.12	children: 0.118	parents: 0.106	finance: 0.085	social: 0.071
100	log2(M+1)	4	0.998	form: 0.237	children: 0.182	housing: 0.141	finance: 0.12	health: 0.091	has_nurs: 0.084	parents: 0.074	social: 0.071
100	sqrt(M)	2	0.977	health: 0.344	has_nurs: 0.161	parents: 0.144	housing: 0.088	form: 0.078	children: 0.069	social: 0.06	finance: 0.057

Table 8: Output of RF algorithm for Nursery dataset

6 Conclusion

The Random Forest classifier significantly outperformed results obtained for DF for all the datasets. In general more trees in the forest and more features used, the higher accuracy was obtained. However thanks to randomization techniques Random Forest achieved good results even if only values of one attribute were considered for a split at time. Random Forest is a great algorithm for classification problems, to produce a predictive model. It returned great results and it avoided overfitting. Moreover, it is a pretty good indicator of the importance it assigns to features.

The feature importance was counted based on Gini Index. It measures how much the tree nodes that use that feature decrease impurity across all the trees in a forest. It depicts the contribution made by every feature in the training phase and scales all the scores such that it sums up to 1. This, in turn, helps in shortlisting the important features and dropping the ones that do not make a huge impact (no impact or less impact) on the model building process. The reason behind considering only a few features is to reduce overfitting that usually materializes when there is a good deal of attributes. The drawbacks of the method is to tendency to prefer (select as important) numerical features and categorical features with high cardinality. What is more, in the case of correlated features it can select one of the feature and neglect the importance of the second one (which can lead to wrong conclusions).

For example, taking into account the importance of features for the nursery dataset, for majority of cases social and finance attribute take the last and next to last places, so it could be worth to consider deletion of them from training set what in many cases will lower computational time.

7 Execution of the code

The implementation of this assignment was performed using Python 3.6. The included packages that were exploited and are required to execute the script:

- **Numpy:** mathematical operations
- **Pandas:** reading databases
- **Sklearn:** splitting the dataset into test and train sets; resample function
- **More_itertools:** set_partitions used for finding all possible proposals of potential splitting values

To execute the algorithm following command line arguments must be specified:

- '-a', '–algorithm' - Decision forest [df] or random forest [rf] algorithm. Possible values: [df, rf]
- '-d', '–dataset_size' - Size of the dataset. Possible values: [small, medium, large, demo]
- '-t', '–test_percentage' - Percentage of the dataset that will be used for testing. Possible values: [between 0.0 and 1.0] Default: 0.2

To reproduce the results from this report the following commands should be executed:

```
python main.py -a df -d small -t 0.2
python main.py -a rf -d small -t 0.2
python main.py -a df -d medium -t 0.2
python main.py -a rf -d medium -t 0.2
python main.py -a df -d large -t 0.2
python main.py -a rf -d large -t 0.2
python main.py -a [df/]rf] -d demo
```

For more detailed information you can call the help function:

```
python main.py -h
```

References

[1] Roger Lewis. An introduction to classification and regression tree (cart) analysis. 01 2000.