|  |
| --- |
| Master in Artificial Intelligence, UPC-URV-UB |
| Random Forest |
| SUPERVISED AND EXPERIENTIAL LEARNING |
|  |
| **Hrizantema Stefanova Stancheva** |
|  |
|  |

[Hrizantema.st@gmail.com](mailto:Hrizantema.st@gmail.com)

Sofia University “St. Kliment Ohridski”

**12/14/2017**

|  |
| --- |
| Under the guidance of Dr. Miquel Sànchez i Marrè |

Content

[Algorithm of Random Forest 3](#_Toc501055030)

[Abstract. 3](#_Toc501055031)

[The ID3 algorithm 3](#_Toc501055032)

[Pseudocode 4](#_Toc501055033)

[The Random Forest algorithm 4](#_Toc501055034)

[Algorithm 4](#_Toc501055035)

[Variable importance 5](#_Toc501055036)

[Classification 5](#_Toc501055037)

[Experiments with the algorithms 5](#_Toc501055038)

[Evaluation of results for all the tested databases: 6](#_Toc501055039)

[Results for the car dataset: 6](#_Toc501055040)

[Results for the mushrooms dataset: 8](#_Toc501055041)

[Results for the nursery dataset: 10](#_Toc501055042)

[Instructions on how to execute the code 12](#_Toc501055043)

[Other comments 13](#_Toc501055044)

# Algorithm of Random Forest

# Abstract.

 Random decision forest is an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. In this report Random forest is used for the task of [classification](https://en.wikipedia.org/wiki/Statistical_classification).

Among the three possibilities for the implementation of the decision tree to be the base learner for inducing the trees, the ID3 algorithm is the chosen one.

## The ID3 algorithm

The ID3 algorithm is used by training on a dataset S to produce a decision tree. This decision tree is used to classify new unseen test cases by working down the decision tree using the values of this test case to arrive at a terminal node that tells us what class this test case belongs to.

To build a decision tree we take a set of possible features. Then we take one feature create tree node for it and split training data. Once training data is split into n sublists same thing is repeated on those sublists with recursion until whole tree is built. The ID3 algorithm builds decision trees using a top­down approach. Moreover ID3 is a greedy algorithm and at each step it selects the best attribute to split on. The best attribute is considered the one which is the most discriminant, which means the one that is potentially more useful according to some measure.

Information IG(A) is the measure of the difference in entropy from before to after the set S is split on an attribute A. In other words, how much uncertainty in S was reduced after splitting set S on attribute A.

Briefly, the steps to the algorithm are:

1. Start with a training data set, which we’ll call S. It should have attributes and classifications

2. Determine the best attribute in the data set S.

3. Split S into subsets that correspond to the possible values of the best attribute.

4. Make a decision tree node that contains the best attribute.

5. Recursively make new decision tree nodes with the subsets of data created in step #3. Attributes can’t be reused. If a subset of data agrees on the classification, choose that classification. If there are no more attributes to split on, choose the most popular classification.

The following pseudocode describes the implementation which I followed.

### Pseudocode

ID3 (Examples, Target\_Attribute, Attributes)

Create a root node for the tree

If all examples are positive, Return the single-node tree Root, with label = +.

If all examples are negative, Return the single-node tree Root, with label = -.

If number of predicting attributes is empty, then Return the single node tree Root,

with label = most common value of the target attribute in the examples.

Otherwise Begin

A ← The Attribute that best classifies examples.

Decision Tree attribute for Root = A.

For each possible value, *vi*, of A,

Add a new tree branch below Root, corresponding to the test A = *vi*.

Let Examples(*vi*) be the subset of examples that have the value *vi* for A

If Examples(*vi*) is empty

Then below this new branch add a leaf node with label = most common target value in the examples

Else below this new branch add the subtree ID3 (Examples(*vi*), Target\_Attribute, Attributes – {A})

End

Return Root

## The Random Forest algorithm

A random forest is a collection of random decision trees.

### Algorithm

1. Sample *N* cases at random are selected from the data The subset is 66% of the total set.
2. At each node:
   1. For some number*F, F* features are selected at random from all the features.
   2. The feature that provides the best split, according to some objective function, is used to do a split on that node.
   3. At the next node, choose another F features at random from all features and do the same.

### Variable importance

Random forests can be used to rank the importance of variables in a regression or classification problem in a natural way. After inducing the random forest we can produce an *ordered list of the features* used in the forest, according to their importance. The importance is be estimated as the frequency of its appearance in the random forest constructed. The level in the tree at which the feature is positioned is not taken into account.

### Classification

Final classification is done by majority vote across trees. After all the trees in the forest classify a given example the result for the class of the example is the mode of all the results.

# Experiments with the algorithms

Java programming language is chosen for implementing the task. The implementation of the algorithm can be found at:

<https://github.com/hrizantema-st/AI-homeworks/tree/master/HW_SEL_2>

The results of the conducted experiments with the algorithm are dependant both on the data set and the hyper-parameters F (number of random features used in the splitting of the nodes) and the number of trees (NT) desired.

We measure the algorithm's classification accuracy by splitting the data into a training set and a test set. Given a **csv** file containing the data which will be used for testing the algorithm we split the examples into two parts – 66% of the data is used for training the algorithm and 33% is used for testing it.

The induced forest is evaluated by the following measurement: the accuracy of the classification process of the forest is equal to the percentage of correctly classified examples.

The experiments are conducted on the three datasets for different combination of values of F and NT; M is the total number of features:

* NT = 50, F = 1
* NT = 50, F = 3
* NT = 50, F = int(log2𝑀+1)
* NT = 50, F = √𝑀
* NT = 100, F = 1
* NT = 100, F = 3
* NT = 100, F = int(log2𝑀+1)
* NT = 100, F = √𝑀

# Evaluation of results for all the tested databases:

For the evaluation of the algorithm are used the following 3 data sets:

* car: <https://archive.ics.uci.edu/ml/datasets/Car+Evaluation>
* mushrooms: <https://archive.ics.uci.edu/ml/datasets/Mushroom>
* nursery: <https://archive.ics.uci.edu/ml/datasets/Nursery>

## Results for the car dataset:

Total number of instances in the dataset: 1728

Number of attributes: 6

Attribute names: "buying", "maint", "doors", "persons", "lug\_boot", "safety".

* NT = 100, F = 1

Accuracy: 0.77018

FEATURES SIZE: ---------------------------- 6

feature: Attr\_5 values: low, med, high,

feature: Attr\_3 values: 2, 4, more,

feature: Attr\_1 values: vhigh, high, med, low,

feature: Attr\_0 values: vhigh, high, med, low,

feature: Attr\_4 values: small, med, big,

feature: Attr\_2 values: 2, 3, 4, 5more,

* NT = 50, F = 1

Accuracy: 0.80175

FEATURES SIZE: ---------------------------- 6

feature: Attr\_5 values: low, med, high,

feature: Attr\_3 values: 2, 4, more,

feature: Attr\_0 values: vhigh, high, med, low,

feature: Attr\_1 values: vhigh, high, med, low,

feature: Attr\_4 values: small, med, big,

feature: Attr\_2 values: 2, 3, 4, 5more,

* NT = 100, F = 3

Accuracy: 0.92193

FEATURES SIZE: ---------------------------- 6

feature: Attr\_0 values: vhigh, high, med, low,

feature: Attr\_2 values: 2, 3, 4, 5more,

feature: Attr\_4 values: small, med, big,

feature: Attr\_1 values: vhigh, high, med, low,

feature: Attr\_3 values: 2, 4, more,

feature: Attr\_5 values: low, med, high,

* NT = 50, F = 3

Accuracy: 0.92193

FEATURES SIZE: ---------------------------- 6

feature: Attr\_0 values: vhigh, high, med, low,

feature: Attr\_2 values: 2, 3, 4, 5more,

feature: Attr\_4 values: small, med, big,

feature: Attr\_1 values: vhigh, high, med, low,

feature: Attr\_3 values: 2, 4, more,

feature: Attr\_5 values: low, med, high,

Since the data set used has only 6 features there is no sense to make experiments with other possibilities for the hyperparameters.

All the test are conducted on 33% of the data which is shuffled before splitting to train and test data, so that at each run of the experiment we obtain n different results depending on the distribution of the data samples.

What we can observe that using higher number of features in the algorithm increases the accuracy of the classification. This is obvious because if the number of features F=1, at which node we select an attribute in which to split randomly.

Another observation which we could make is that when we double the number of the decision trees in the forest the accuracy increases again, but slightly.

Also the order of the features changes depending on the different values for the number of trees, but we see that the order is independent on the number of features in the current tests.

The results of the conducted tests are summarized in the following table:

|  |  |  |
| --- | --- | --- |
| **Table of accuracy for the cars dataset** | | |
| **NT / F** | **1** | **3** |
| **50** | **0.80175** | **0.92193** |
| **100** | **0.77018** | **0.92193** |

## Results for the mushrooms dataset:

Total number of instances in the dataset: 8124

Number of attributes: 22

Attribute names: "buying", "maint", "doors", "persons", "lug\_boot", "safety".

It is interesting to test the performance of the algorithm on a data set with a bigger number of attributes.

* NT = 50, F = 1

Accuracy: 0.65529

FEATURES SIZE: ---------------------------- 22

feature: Attr\_4 values: p, a, l, n, f, c, y, s, m,

feature: Attr\_10 values: e, c, b, r, ?,

feature: Attr\_9 values: e, t,

feature: Attr\_18 values: p, e, l, f, n,

feature: Attr\_7 values: n, b,

feature: Attr\_12 values: s, f, y, k,

feature: Attr\_14 values: w, p, g, b, n, e, y, o, c,

feature: Attr\_5 values: f, a,

feature: Attr\_20 values: s, n, a, v, y, c,

feature: Attr\_2 values: n, y, w, g, e, p, b, u, c, r,

feature: Attr\_15 values: p,

feature: Attr\_0 values: x, b, s, f, k, c,

feature: Attr\_21 values: u, g, m, d, p, w, l,

feature: Attr\_3 values: t, f,

feature: Attr\_13 values: w, g, p, n, b, e, o, c, y,

feature: Attr\_6 values: c, w,

feature: Attr\_1 values: s, y, f, g,

feature: Attr\_17 values: o, t, n,

feature: Attr\_11 values: s, f, k, y,

feature: Attr\_16 values: w, n, o, y,

feature: Attr\_19 values: k, n, u, h, w, r, o, y, b,

feature: Attr\_8 values: k, n, g, p, w, h, u, e, b, r, y, o,

* NT = 100, F = 1

Accuracy: 0.65529

FEATURES SIZE: ---------------------------- 22

feature: Attr\_4 values: p, a, l, n, f, c, y, s, m,

feature: Attr\_10 values: e, c, b, r, ?,

feature: Attr\_7 values: n, b,

feature: Attr\_21 values: u, g, m, d, p, w, l,

feature: Attr\_9 values: e, t,

feature: Attr\_3 values: t, f,

feature: Attr\_18 values: p, e, l, f, n,

feature: Attr\_17 values: o, t, n,

feature: Attr\_6 values: c, w,

feature: Attr\_15 values: p,

feature: Attr\_14 values: w, p, g, b, n, e, y, o, c,

feature: Attr\_12 values: s, f, y, k,

feature: Attr\_13 values: w, g, p, n, b, e, o, c, y,

feature: Attr\_20 values: s, n, a, v, y, c,

feature: Attr\_1 values: s, y, f, g,

feature: Attr\_16 values: w, n, o, y,

feature: Attr\_5 values: f, a,

feature: Attr\_11 values: s, f, k, y,

feature: Attr\_2 values: n, y, w, g, e, p, b, u, c, r,

feature: Attr\_19 values: k, n, u, h, w, r, o, y, b,

feature: Attr\_0 values: x, b, s, f, k, c,

feature: Attr\_8 values: k, n, g, p, w, h, u, e, b, r, y, o,

* NT = 50, F = 3

Accuracy: 0.65529

FEATURES SIZE: ---------------------------- 22

feature: Attr\_4 values: p, a, l, n, f, c, y, s, m,

feature: Attr\_10 values: e, c, b, r, ?,

feature: Attr\_21 values: u, g, m, d, p, w, l,

feature: Attr\_7 values: n, b,

feature: Attr\_20 values: s, n, a, v, y, c,

feature: Attr\_3 values: t, f,

feature: Attr\_8 values: k, n, g, p, w, h, u, e, b, r, y, o,

feature: Attr\_19 values: k, n, u, h, w, r, o, y, b,

feature: Attr\_18 values: p, e, l, f, n,

feature: Attr\_9 values: e, t,

feature: Attr\_2 values: n, y, w, g, e, p, b, u, c, r,

feature: Attr\_1 values: s, y, f, g,

feature: Attr\_11 values: s, f, k, y,

feature: Attr\_0 values: x, b, s, f, k, c,

feature: Attr\_12 values: s, f, y, k,

feature: Attr\_13 values: w, g, p, n, b, e, o, c, y,

feature: Attr\_6 values: c, w,

feature: Attr\_14 values: w, p, g, b, n, e, y, o, c,

feature: Attr\_17 values: o, t, n,

feature: Attr\_16 values: w, n, o, y,

feature: Attr\_15 values: p,

feature: Attr\_5 values: f, a,

* NT = 50, F = 3

Accuracy: 0.65529

FEATURES SIZE: ---------------------------- 20

feature: Attr\_4 values: p, a, l, n, f, c, y, s, m,

feature: Attr\_7 values: n, b,

feature: Attr\_19 values: k, n, u, h, w, r, o, y, b,

feature: Attr\_21 values: u, g, m, d, p, w, l,

feature: Attr\_20 values: s, n, a, v, y, c,

feature: Attr\_10 values: e, c, b, r, ?,

feature: Attr\_9 values: e, t,

feature: Attr\_2 values: n, y, w, g, e, p, b, u, c, r,

feature: Attr\_3 values: t, f,

feature: Attr\_18 values: p, e, l, f, n,

feature: Attr\_12 values: s, f, y, k,

feature: Attr\_11 values: s, f, k, y,

feature: Attr\_8 values: k, n, g, p, w, h, u, e, b, r, y, o,

feature: Attr\_1 values: s, y, f, g,

feature: Attr\_6 values: c, w,

feature: Attr\_14 values: w, p, g, b, n, e, y, o, c,

feature: Attr\_13 values: w, g, p, n, b, e, o, c, y,

feature: Attr\_0 values: x, b, s, f, k, c,

feature: Attr\_17 values: o, t, n,

feature: Attr\_16 values: w, n, o, y,

### For this database we observe a very strange behavior – the accuracy of the classification does not change when we increase the hyperparameters NT and F, only the list of features ordered by importance changes

### Results for the nursery dataset:

Total number of instances in the dataset: 12960

Number of attributes: 8

Attribute names: "parents", "has\_nurs", "form", "children", "housing", "finance", "social", "health".

* NT = 50, F = 1

Accuracy: 0.54273

FEATURES SIZE: ---------------------------- 8

feature: Attr\_7 values: recommended, priority, not\_recom,

feature: Attr\_1 values: proper, less\_proper, improper, critical, very\_crit,

feature: Attr\_0 values: usual, pretentious, great\_pret,

feature: Attr\_5 values: convenient, inconv,

feature: Attr\_6 values: nonprob, slightly\_prob, problematic,

feature: Attr\_4 values: convenient, less\_conv, critical,

feature: Attr\_3 values: 1, 2, 3, more,

feature: Attr\_2 values: complete, completed, incomplete, foster,

We can notice that for this data set with only one feature in the set from which to choose a split at each node the algorithm performs very poor, classifying correctly only half of the data.

* NT = 100, F = 1

Accuracy: 0.75728

FEATURES SIZE: ---------------------------- 8

feature: Attr\_7 values: recommended, priority, not\_recom,

feature: Attr\_1 values: proper, less\_proper, improper, critical, very\_crit,

feature: Attr\_0 values: usual, pretentious, great\_pret,

feature: Attr\_5 values: convenient, inconv,

feature: Attr\_4 values: convenient, less\_conv, critical,

feature: Attr\_6 values: nonprob, slightly\_prob, problematic,

feature: Attr\_3 values: 1, 2, 3, more,

feature: Attr\_2 values: complete, completed, incomplete, foster,

Increasing the number of trees increases the accuracy, but the results are still not satisfactory.

* NT = 50, F = 3

Accuracy: 0.93219

FEATURES SIZE: ---------------------------- 8

feature: Attr\_2 values: complete, completed, incomplete, foster,

feature: Attr\_3 values: 1, 2, 3, more,

feature: Attr\_5 values: convenient, inconv,

feature: Attr\_4 values: convenient, less\_conv, critical,

feature: Attr\_0 values: usual, pretentious, great\_pret,

feature: Attr\_7 values: recommended, priority, not\_recom,

feature: Attr\_6 values: nonprob, slightly\_prob, problematic,

feature: Attr\_1 values: proper, less\_proper, improper, critical, very\_crit,

* NT = 100, F = 3

Accuracy: 0.93266

FEATURES SIZE: ---------------------------- 8

feature: Attr\_2 values: complete, completed, incomplete, foster,

feature: Attr\_5 values: convenient, inconv,

feature: Attr\_3 values: 1, 2, 3, more,

feature: Attr\_4 values: convenient, less\_conv, critical,

feature: Attr\_0 values: usual, pretentious, great\_pret,

feature: Attr\_7 values: recommended, priority, not\_recom,

feature: Attr\_6 values: nonprob, slightly\_prob, problematic,

feature: Attr\_1 values: proper, less\_proper, improper, critical, very\_crit,

With F = 3 we obtain accuracy > 90%. The difference between the accuracy for 100 and 50 decision trees in the forest is insignificant so we can conclude that. Given the fact that increasing the NT, increases the time for the calculations a good trade-off point has to be found.

Again, as in the previous experiments the list of the features ordered by importance changes when we change the number of features F, but it seems that it is independent of the number of decision trees.

* NT = 50, F = 5

Accuracy: 0.93195

FEATURES SIZE: ---------------------------- 8

feature: Attr\_2 values: complete, completed, incomplete, foster,

feature: Attr\_3 values: 1, 2, 3, more,

feature: Attr\_5 values: convenient, inconv,

feature: Attr\_4 values: convenient, less\_conv, critical,

feature: Attr\_0 values: usual, pretentious, great\_pret,

feature: Attr\_6 values: nonprob, slightly\_prob, problematic,

feature: Attr\_7 values: recommended, priority, not\_recom,

feature: Attr\_1 values: proper, less\_proper, improper, critical, very\_crit,

* NT = 100, F = 5

Accuracy: 0.91746

FEATURES SIZE: ---------------------------- 8

feature: Attr\_2 values: complete, completed, incomplete, foster,

feature: Attr\_3 values: 1, 2, 3, more,

feature: Attr\_5 values: convenient, inconv,

feature: Attr\_4 values: convenient, less\_conv, critical,

feature: Attr\_0 values: usual, pretentious, great\_pret,

feature: Attr\_6 values: nonprob, slightly\_prob, problematic,

feature: Attr\_7 values: recommended, priority, not\_recom,

feature: Attr\_1 values: proper, less\_proper, improper, critical, very\_crit,

Increasing the number of features F to 5 does not result in a much better results.

A table summarizing the tests follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Table of accuracy for the nursery s dataset** | | | |
| **NT/F** | **1** | **3** | **5** |
| **50** | **0.54273** | **0.93219** | **0.93195** |
| **100** | **0.75728** | **0.93266** | **0.91746** |

Since the total number of features in the given data set is 8, is does not make sense to test the classification algorithm on different values for F.

# Instructions on how to execute the code

In order to run a the JAR file containing the Random Forest classifier you need to have Java 8 installed on your system, since the implementation uses some of the features of the Java programming language presented in the 1.8 version. To execute the jar file you should run the following command in cmd on Windows:

**java -jar Main.jar [dataset] [NT] [F]**

where **[dataset]** is the parameter which should contain one of the dataset names:

*car.data, agaricus-lepiota.data, nursery.data*

which are included in the resources folder of the project.

**[NT]** should be an integer which specifies the number of decision trees which will be induces in order to create the forest

**[F]** should be an integer which specifies the number of random features, which will be selected for every step in the generation of the decision tree

All parameters are mandatory in order for the program to execute.

After executing the jar file with the desired arguments on the console is printed the most important information for the specified data set: The selected dataset, the number of the examples in the dataset, the trees induces, the accuracy of the classification, list of features ordered by importance

# Other comments

* All the data sets that are used for testing the algorithm consist only of features with categorical values.
* All the data sets that are used for testing are without any noise. But even with noisy data presented in the data sets we can handle it easily by substituting the missing value with the mode value for the current attribute.
* Since my idea was the algorithm to work with all possible data sets with categorical attributes, but there is no uniform way to extract the names of the attributes of the data set the method responsible for extracting all the selectors is assigning names with the structure *“Attr\_i”* where *i* is the index of the current attribute. Furthermore the implementation suggests that the class of each data set is in the last column of the data set. This can also be easily modified to handle data sets which have class label in a different position but for the purpose of this task this is not needed; For the given task – to test the algorithm on 3 specific data sets there is a possibility to hardcode the names of the attributes so that the representation of the rules is more readable. (tree static arrays are added in the end of Main.java file which could be used for more readable printing of the decision trees but this is not implemented yet).
* Depending on the nature of the dataset we observe different results –the nursery data set is the most suitable one on which to check how the accuracy of the algorithm depends on the hyperparameters.