SUPERVISED AND EXPERIENTIAL LEARNING

MASTER IN ARTIFICIAL INTELLIGENCE

Universitat Politècnica de Catalunya

PRACTICAL WORK 2:

IMPLEMENTATION AND EVALUATION OF A RANDOM FOREST CLASSIFIER

Albert Espín

April 29th, 2019

Table Of Contents

1. Introduction

The Random Forest classifier was introduced by Leo Breiman from University of California in 2001 [1]. The innovative characteristic of this algorithm is the usage of a set of decision trees that consider only a random subset of features for each split node, and classify new instances according to the majority vote of each tree.

The objective of this work is the implementation of the Random Forest classifier in Python 3, and the evaluation of the algorithm in the classification task using different combinations of hyper-parameters, applied it to 5 data sets of different sizes and features. The accuracy of each trained model is computed, as well as the importance of each feature.

2. The Random Forest classifier algorithm

As any other classifier algorithm in machine learning, the Random Forest classifier is trained with a set of training instances, described by a feature matrix and a class vector, and can be evaluated with a set of test instances, whose class is predicted based only on the feature matrix.

Figure 1 represents the training algorithm for the Random Forest classifier. Apart from the instances (described, as mentioned, with a feature matrix and class vector), this algorithm has two configuration parameters: the number of trees and the number of features to consider at each split node. The algorithm creates as many decision trees as specified by the tree number parameter. Each tree uses a Bootstrap sample of the training instances, i.e. a set with as many elements as the original instances, each of them being a random selection among the original instances, which means that repetition is allowed. This procedure is useful to simulate variability on the distribution of the training data, without explicitly discarding any instance (but in practice using approximately two thirds of the original set in each sample, due to repetitions). The set of trees form the random forest, from which a list of the features of the data, sorted in descending order of importance, is obtained. To do so, each tree is recursively traversed, and all split-decision nodes (i.e. non-terminal ones) are checked to see which is the feature they use to do the split. The importance of each feature is then computed as the proportion of nodes that use that feature in their split decision: this way, the sum of the importance of all features is 1.

Algorithm 1: Random Forest classifier (training)

```
Input  : instances /*List of training instances, with features and class*/,
         tree_num /*Number of trees in the forest*/,
         split_feature_num /*Number of features to randomly select at each split node*/
Output: random_forest /*Decision trees of the forest*/,
         features_by_importance /*Features in their order of importance*/
random_forest ← list();
/*Create the trees*/
for i ← 1 to tree_num do
    /*Use a bootstrap sample (allowing repetition) of the instances, of the same size*/
    bootstrap_instances ← get_samples_with_repetition(instances, len(instances));
    /*Build a decision tree using the bootstrap sample, and add it to the forest*/
    random_tree ← build_random_tree(bootstrap_instances, split_feature_num);
    random_forest.append(random_tree);
/*Find (feature, importance) pairs, sorted by descending importance: the importance of each
  feature is the number of times it is used in split nodes divided by the number of nodes*/
features_by_importance ← get_features_by_importance(random_forest);
```

Figure 1: Algorithm for training the Random Forest classifier.

Figure 2 presents in more detail the recursive algorithm used to build a decision tree (and all its subtrees, i.e. children nodes given a reference node). The algorithm receives as parameters the training instances that should be considered to take the split decision, as well as the number of features to randomly select. The decision trees are binary, based on the Classification And Regression Trees (CART [2]) model. The best split point among these features is determined, by selecting the one that provides the lowest value of impurity, defined using the Gini index, that is 0 when all instances belong to the same class, and greater the more class dispersion there is, representing a probability of classifying instances wrongly. For each of the randomly selected features, a set of possible split values is considered. For numeric features, the candidate split points are the midpoints between each pair of sorted continuous values, while categorical split points are all the possible combinations of set membership (included or not) among the categories present in the instances. As mentioned, the split point with the lowest impurity value is used to split the data, according to the evaluation criterion of a new split node. For numeric features, the instances with value lower to the split point are directed to the left of the node, and those instances with value higher or equal to the split point are assigned to the right split; for categorical values, if the category of the split feature in an instance is included in the category set of the split point, the instance is directed to the right, otherwise to the left. This binary node is then used as root node of a new subtree, since the algorithm is called again for the right and left splits of the data, to create the right and left child nodes of the algorithm, respectively. Each recursive

exploration ends when all the instances that have reached there have the same class. At that point, a terminal node with associated to the class is created.

Algorithm 2: Recursive random tree creation (build_random_tree)

```
Input  : instances /*List of instances, with features and class*/,
         split_feature_num /*Number of features to randomly select at each split node*/
Output: node /*Root node of the decision tree*/
/*Find the unique classes of the instances*/
unique_classes ← unique_elements(instances.classes);
/*If all the instances belong to the same class, create a terminal node for that class*/
if len(unique_classes) == 1 then
    node ← TerminalNode(unique_classes[0]);
/*Otherwise create the a node with the best split*/
else
    /*Find the best possible split according to randomly selected features (as many as indicated
      by the parameter, selecting the value that minimizes Gini impurity, considering all
      midpoints of sorted values for numeric features and the membership of all combinations of
      categories for categorical features), generating a SplitNode and the data split with the two
      resulting partitions of the instances*/
    node, data_split = create_best_split_node(instances, split_feature_num);
    /*If it was not possible to split with the randomly selected features, create a terminal node
      with the majority class*/
    if node == null then
        node ← TerminalNode(mode(instances.classes));
    /* Otherwise keep dividing the tree left and right*/
    else
        left_child ← build_random_tree(data_split.left_instances, split_feature_num);
        if left_child != null then
            node.left_child ← left_child;
        right_child ← build_random_tree(data_split.right_instances, split_feature_num);
        if right_child != null then
            node.right_child ← right_child;
```

Figure 2: Recursive algorithm to create a decision tree of the Random Forest.

The test set evaluation algorithm is depicted in Figure 3. All trees in the Random Forest predict the class of each instance, by evaluating it at the root node, and recursively to the rest of nodes of the tree (following the left or right direction at each split node according to the criteria explained in the previous paragraph), until a terminal node is reached: the class given by the terminal node is the class predicted by the tree. For each instance, the final assigned class is the one voted by a majority of trees (i.e. the mode), with random tie break. Each prediction of the Random Forest is compared with the ground truth classes, to determine the accuracy of the classifier as the proportion of correct classifications.

Algorithm 3: Random Forest classifier (prediction)

```
Input  : instances /*List of test instances, with features but not class*/,
         classes /*True classes of each instance*/
Output: assigned_classes /*Predicted classes for the instances*/,
         accuracy /*Classification accuracy*/
assigned_classes ← list();
correct_class_count ← 0;
/*Analyze each instance*/
for i, instance in enumerate(instances) do
    /*Let all trees vote*/
    voted_classes ← list();
    for tree in random_forest do
        /*The prediction of a tree is found descending through its nodes, by evaluating the
          binary condition of nodes based on the instance value for the split feature (greater-than
          check for numeric features, set-membership check for categorical ones), until a terminal
          node with a class is reached*/
        voted_classes.append(predict_class_with_tree(instance, tree));
    /*The assigned class is the one that obtains the majority vote of the trees*/
    assigned_class ← mode(voted_classes);
    assigned_classes.append(assigned_class);
    /*Check if the classification is correct*/
    if assigned_class == classes[i] then
        correct_class_count ← correct_class_count + 1;
accuracy ← correct_class_count / len(classes);
```

Figure 3: Algorithm to make predictions with the Random Forest classifier.

3. Experimental methodology

The Random Classifier algorithm has been tested with 5 data sets from the UCI repository [3] to evaluate its performance. To enrich the experimental variability, these data sets include different proportions of numeric and categorical features, balanced and unbalanced class distributions, different number of classes, and some of the data sets contain missing values.

The contact lenses data set is a tiny, purely categorical data set where the requirement of lenses (none, soft or hard) should be predicted from vision-related features of the individuals. The labor data set is a small data set combining numeric and categorical features that describe contract elements and the output is binary: whether the contract should be considered as acceptable or not. Despite of its small size, it is challenging due to the high percentage of missing values (35.7%). The hepatits data sets is larger, and also combines categorical and numeric features related with hepatitis patients to determine whether they will live or die. The breast cancer data set is also two-class and with both numeric and categorical classes, but is larger than the previous ones and has the special characteristic of having inconsistent

instances, i.e. examples with same features but different class. Finally, the car evaluation data set, which is purely categorical, has been added to see how the algorithm performs in a considerably large set of instances (1728).

As a pre-processing operation, the missing values of the data sets have been replaced with the mean of the feature in the case of numeric attributes, and the mode for categorical ones. To ensure that all numeric attributes have the same relevance when computing rule-to-instance distances, the ranges of numeric features have been linearly normalized between 0 and 1: the minimum value has become 0, the maximum 1, and the rest lie in between.

The model has been trained with a 90% of the data of each data set, while the 10% has been used for test (except for contact lenses, where 5 of the 24 instances have been used for test, to make the results more reliable). The data splits are stratified, in the sense that for both training and test the proportion of each class is approximately preserved, with all classes appearing in both splits most of the times.

As mentioned in the previous section, the Random Forest classifier has been given two configuration parameters, and different values are tested for each of them: 50 and 100 trees per forest, with both values combined with different numbers of features to consider at each split node: 1, 3, the base-2 logarithm of the total number of features (if different to 1 or 3), and the square root of the absolute number of features (if different to the previous values). The objective of these experiments is to analyze the impact of these parameters in the accuracy results of each data set, calculated as the proportion of correctly classified test instances, so it is a value between 0 and 1. The results are compared with a reputed implementation of the algorithm, from the scikit-learn library [4].

4. Results and discussion

4.1. Contact lenses

The mean accuracy obtained with the implemented algorithm is 0.933 ± 0.094. Particularly, as shown in Table 1, when the number of features considered to decide the best split at a node is 1, the accuracy has been 0.8, while in the rest of cases, independently of the number of trees, an accuracy value of 1 has been obtained. From these results it can be induced that using more than 1 feature to select the split is the most suitable scenario for decision trees applied to this data set. This idea is strengthen by the fact that various features have a similar level of importance, and

therefore discrimination capacity. Namely, the age and the tear production rate have been proven to be crucial to predict if a person should have contact lenses, with importance proportions close to 0.25 in all the combinations. The rest of features (asigmatism and spectacles prescription), however, are also always above 0.20, which means that their relevance in decisions is also remarkable.

Table 1: Accuracy for each Random Forest configuration for the contact lenses data set.

Tree number	Split feature number	Accuracy	Features by importance
50	1	0.8	('age', 0.291), ('spectacle-prescrip', 0.246), ('tear-prod-rate', 0.246), ('astigmatism', 0.218)
50	2	1	('tear-prod-rate', 0.289), ('age', 0.265), ('astigmatism', 0.225), ('spectacle-prescrip', 0.221)
50	3	1	('age', 0.278), ('tear-prod-rate', 0.27), ('astigmatism', 0.235), ('spectacle-prescrip', 0.217)
100	1	0.8	('age', 0.289), ('tear-prod-rate', 0.245), ('astigmatism', 0.234), ('spectacle-prescrip', 0.231)
100	2	1	('tear-prod-rate', 0.29), ('age', 0.284), ('astigmatism', 0.224), ('spectacle-prescrip', 0.201)
100	3	1	('age', 0.264), ('tear-prod-rate', 0.262), ('astigmatism', 0.247), ('spectacle-prescrip', 0.227)

4.2. Labor

The mean accuracy obtained with the implemented algorithm in the labor data set is 1.0 ± 0.0, which means that all test instances are correctly predicted in the majority vote of the trees, meaning that many of the decision trees, if not all, are extremely successful in this data set, which is especially remarkable considering the large presence of missing values, that are replaced with the mode category or mean numeric value. The high number of attributes in this data set, that has a small number of instances, allows to perform a very detailed, instance-driven division of the tree sections. Since the test instances follow very similar patterns to those of training examples, the test instances can be precisely classified. The most relevant feature, with roughly 0.1 importance value, is the wage of the first year, determinant to determine the success of contract, agreement, closely followed by other elements, such as the second year wage and the amount of holidays.

Table 2: Accuracy for each Random Forest configuration for the labor data set.

Tree number	Split feature number	Accuracy	Features by importance
50	1	1	('wage1', 0.091), ('holidays', 0.088), ('wage2', 0.08), ('shift_diff', 0.077), ('dur', 0.077), ('vacation', 0.077), ('wage3', 0.069), ('dntl_ins', 0.064), ('cola', 0.064), ('hours', 0.056), ('stby_pay', 0.056), ('educ_allw', 0.051), ('empl_hplan', 0.051), ('pension', 0.048), ('lngtrm_disabil', 0.035), ('bereavement', 0.016)
50	3	1	('wage1', 0.126), ('shift_diff', 0.101), ('hours', 0.095), ('holidays', 0.087), ('vacation', 0.07), ('wage2', 0.067), ('empl_hplan', 0.067), ('cola', 0.059), ('pension', 0.056), ('dur', 0.053), ('dntl_ins', 0.053), ('lngtrm_disabil', 0.053), ('stby_pay', 0.047), ('educ_allw', 0.028), ('wage3', 0.022), ('bereavement', 0.017)

50	4	1	('wage1', 0.133), ('holidays', 0.108), ('wage2', 0.099), ('shift_diff', 0.091), ('pension', 0.082), ('hours', 0.071), ('empl_hplan', 0.068), ('dntl_ins', 0.059), ('cola', 0.057), ('vacation', 0.054), ('dur', 0.051), ('educ_allw', 0.042), ('lngtrm_disabil', 0.042), ('wage3', 0.017), ('stby_pay', 0.017), ('bereavement', 0.008)
50	5	1	('wage1', 0.153), ('wage2', 0.1), ('holidays', 0.1), ('pension', 0.087), ('hours', 0.077), ('empl_hplan', 0.077), ('lngtrm_disabil', 0.067), ('vacation', 0.063), ('shift_diff', 0.063), ('dntl_ins', 0.053), ('cola', 0.047), ('educ_allw', 0.03), ('dur', 0.03), ('stby_pay', 0.03), ('wage3', 0.017), ('bereavement', 0.007)
100	1	1	('hours', 0.085), ('empl_hplan', 0.082), ('wage1', 0.08), ('holidays', 0.073), ('wage2', 0.073), ('dur', 0.067), ('stby_pay', 0.067), ('shift_diff', 0.065), ('vacation', 0.064), ('pension', 0.064), ('dntl_ins', 0.059), ('cola', 0.058), ('educ_allw', 0.053), ('wage3', 0.053), ('lngtrm_disabil', 0.036), ('bereavement', 0.022)
100	3	1	('wage1', 0.138), ('holidays', 0.104), ('shift_diff', 0.093), ('wage2', 0.09), ('pension', 0.083), ('empl_hplan', 0.062), ('cola', 0.061), ('lngtrm_disabil', 0.061), ('hours', 0.056), ('vacation', 0.053), ('dntl_ins', 0.053), ('educ_allw', 0.04), ('dur', 0.04), ('wage3', 0.036), ('stby_pay', 0.027), ('bereavement', 0.003)
100	4	1	('wage1', 0.132), ('holidays', 0.119), ('wage2', 0.09), ('pension', 0.086), ('hours', 0.077), ('lngtrm_disabil', 0.071), ('shift_diff', 0.066), ('empl_hplan', 0.061), ('dntl_ins', 0.061), ('cola', 0.055), ('vacation', 0.047), ('dur', 0.039), ('educ_allw', 0.039), ('wage3', 0.03), ('stby_pay', 0.022), ('bereavement', 0.003)
100	5	1	('wage1', 0.139), ('holidays', 0.113), ('wage2', 0.103), ('pension', 0.091), ('hours', 0.081), ('shift_diff', 0.08), ('empl_hplan', 0.075), ('lngtrm_disabil', 0.061), ('vacation', 0.061), ('cola', 0.045), ('dntl_ins', 0.041), ('wage3', 0.036), ('dur', 0.028), ('educ_allw', 0.025), ('stby_pay', 0.017), ('bereavement', 0.003)

4.3. Hepatitis

The mean accuracy obtained with the implemented algorithm is 0.852 ± 0.043, with the highest value, 0.938, obtained with 50 trees and 4 split features considered per node, followed by 0.875 accuracy by different combinations, such as 50 trees with 1 split feature, 100 with 1 split feature and 100 trees and 4 features; in the rest of cases the accuracy is 0.813. It can be seen that using 4 or 1 features per node is the best choice, unless for the performed experiments, while using 3 or 5 leads to lower accuracy, independently of the number of trees. Due to the standard variation of 0.043, the conclusions about these results should be rather conservative: it is not intuitively clear why those 1 or 4 features are better that 3, and it may be caused due to the inherent randomness of the algorithm. In any case, the most important features to differentiate between patients that have or do not have hepatatis are the level bilirubin and alubimin, both surpassing the 0.1 value in different combinations.

Table 3: Accuracy for each Random Forest configuration for the hepatitis data set.

Tree number	Split feature number	Accuracy	Features by importance
50	1	0.875	('Sgot', 0.082), ('Bilirubin', 0.071), ('Protime', 0.063), ('Age', 0.058), ('Alubimin', 0.056), ('Alk phosphate', 0.054), ('Liver firm', 0.054), ('Histology', 0.054), ('Malaise', 0.051), ('Liver big', 0.049), ('Anorexia', 0.049), ('Sex', 0.049), ('Steroid', 0.049), ('SPiders', 0.048), ('Varices', 0.046), ('Spleen palpable', 0.044), ('Fatigue', 0.043), ('Ascites', 0.04), ('Antivirals', 0.04)
50	3	0.813	('Bilirubin', 0.107), ('Alubimin', 0.1), ('Age', 0.09), ('Sgot', 0.089), ('Alk phosphate', 0.086), ('Protime', 0.073), ('SPiders', 0.054), ('Malaise', 0.046), ('Steroid', 0.044), ('Liver big', 0.043), ('Varices', 0.041), ('Liver firm', 0.036), ('Ascites', 0.034), ('Spleen palpable', 0.033), ('Histology', 0.033), ('Anorexia', 0.032), ('Sex', 0.026), ('Fatigue', 0.023), ('Antivirals', 0.01)
50	4	0.938	[('Alubimin', 0.111), ('Bilirubin', 0.101), ('Alk phosphate', 0.099), ('Sgot', 0.097), ('Age', 0.087), ('Protime', 0.083), ('SPiders', 0.053), ('Liver firm', 0.044), ('Histology', 0.041), ('Varices', 0.036), ('Malaise', 0.036), ('Ascites', 0.035), ('Liver big', 0.035), ('Spleen palpable', 0.032), ('Sex', 0.028), ('Steroid', 0.027), ('Anorexia', 0.024), ('Fatigue', 0.022), ('Antivirals', 0.008)
50	5	0.813	('Bilirubin', 0.162), ('Sgot', 0.109), ('Alubimin', 0.106), ('Age', 0.106), ('Alk phosphate', 0.102), ('Protime', 0.075), ('SPiders', 0.056), ('Liver firm', 0.033), ('Liver big', 0.032), ('Malaise', 0.031), ('Ascites', 0.03), ('Varices', 0.03), ('Anorexia', 0.025), ('Spleen palpable', 0.022), ('Histology', 0.021), ('Fatigue', 0.02), ('Sex', 0.018), ('Steroid', 0.015), ('Antivirals', 0.007)
100	1	0.875	('Alubimin', 0.073), ('Sgot', 0.07), ('Bilirubin', 0.066), ('Protime', 0.065), ('Alk phosphate', 0.064), ('SPiders', 0.06), ('Age', 0.057), ('Liver firm', 0.057), ('Steroid', 0.052), ('Histology', 0.048), ('Ascites', 0.047), ('Varices', 0.047), ('Malaise', 0.046), ('Liver big', 0.045), ('Anorexia', 0.043), ('Spleen palpable', 0.043), ('Fatigue', 0.043), ('Sex', 0.038), ('Antivirals', 0.034)
100	3	0.813	('Alubimin', 0.101), ('Bilirubin', 0.101), ('Alk phosphate', 0.1), ('Sgot', 0.098), ('Age', 0.08), ('Protime', 0.077), ('SPiders', 0.054), ('Histology', 0.049), ('Malaise', 0.044), ('Liver big', 0.04), ('Liver firm', 0.038), ('Ascites', 0.036), ('Varices', 0.035), ('Spleen palpable', 0.032), ('Steroid', 0.032), ('Anorexia', 0.03), ('Fatigue', 0.022), ('Sex', 0.019), ('Antivirals', 0.012)
100	4	0.875	('Alubimin', 0.123), ('Bilirubin', 0.114), ('Age', 0.099), ('Alk phosphate', 0.092), ('Sgot', 0.088), ('Protime', 0.077), ('SPiders', 0.046), ('Liver firm', 0.039), ('Malaise', 0.039), ('Ascites', 0.038), ('Liver big', 0.037), ('Varices', 0.034), ('Histology', 0.034), ('Anorexia', 0.031), ('Spleen palpable', 0.027), ('Fatigue', 0.026), ('Steroid', 0.026), ('Sex', 0.019), ('Antivirals', 0.011)
100	5	0.813	('Bilirubin', 0.128), ('Alk phosphate', 0.113), ('Age', 0.103), ('Alubimin', 0.095), ('Protime', 0.094), ('Sgot', 0.076), ('SPiders', 0.051), ('Histology', 0.042), ('Liver firm', 0.042), ('Malaise', 0.037), ('Varices', 0.037), ('Anorexia', 0.03), ('Liver big', 0.029), ('Steroid', 0.027), ('Spleen palpable', 0.025), ('Ascites', 0.024), ('Sex', 0.019), ('Fatigue', 0.017), ('Antivirals', 0.008)

4.4. Breast cancer

The mean accuracy obtained with the implemented algorithm is 0.964 ± 0.007, with a very small difference between the worst result (0.9571) and the best one (0.971), which leads to think that the number of trees and the number of split features considered per node are not determinant parameters for this data set. The results are remarkably high considering the presence of inconsistent examples, i.e. instances with the same features but different class, that inevitably lead to some wrong classifications: if different of these examples form the instances for which to create a node, they cannot be separated since their features are equal, so ultimately a terminal node with the mode class among them is created. The most important features to

determine if a patient has cancer are those related to normal nucleoli and bare nuclei, followed by the clump thickness: all of them have a value higher than 0.1 in most cases.

Table 4: Accuracy for each Random Forest configuration for the breast cancer data set.

Tree number	Split feature number	Accuracy	Features by importance
50	1	0.971	('Normal Nucleoli', 0.122), ('Clump Thickness', 0.122), ('Uniformity of Cell Shape', 0.117), ('Single Epithelial Cell Size', 0.116), ('Bare Nuclei', 0.115), ('Bland Chromatin', 0.114), ('Uniformity of Cell Size', 0.11), ('Marginal Adhesion', 0.104), ('Mitoses', 0.08)
50	3	0.957	('Bare Nuclei', 0.158), ('Clump Thickness', 0.151), ('Uniformity of Cell Shape', 0.132), ('Normal Nucleoli', 0.119), ('Uniformity of Cell Size', 0.118), ('Marginal Adhesion', 0.104), ('Single Epithelial Cell Size', 0.094), ('Bland Chromatin', 0.087), ('Mitoses', 0.037)
50	4	0.971	('Bare Nuclei', 0.16), ('Clump Thickness', 0.147), ('Uniformity of Cell Shape', 0.12), ('Normal Nucleoli', 0.109), ('Uniformity of Cell Size', 0.108), ('Single Epithelial Cell Size', 0.105), ('Bland Chromatin', 0.101), ('Marginal Adhesion', 0.099), ('Mitoses', 0.052)
100	1	0.957	('Clump Thickness', 0.119), ('Single Epithelial Cell Size', 0.118), ('Uniformity of Cell Shape', 0.117), ('Normal Nucleoli', 0.117), ('Bare Nuclei', 0.114), ('Bland Chromatin', 0.112), ('Uniformity of Cell Size', 0.111), ('Marginal Adhesion', 0.111), ('Mitoses', 0.08)
100	3	0.957	('Bare Nuclei', 0.156), ('Clump Thickness', 0.138), ('Uniformity of Cell Size', 0.117), ('Single Epithelial Cell Size', 0.115), ('Uniformity of Cell Shape', 0.113), ('Marginal Adhesion', 0.109), ('Normal Nucleoli', 0.107), ('Bland Chromatin', 0.102), ('Mitoses', 0.042)
100	4	0.971	('Bare Nuclei', 0.185), ('Clump Thickness', 0.147), ('Uniformity of Cell Shape', 0.116), ('Uniformity of Cell Size', 0.111), ('Marginal Adhesion', 0.108), ('Normal Nucleoli', 0.106), ('Bland Chromatin', 0.097), ('Single Epithelial Cell Size', 0.095), ('Mitoses', 0.036)

4.5. Car evaluation

The mean accuracy obtained with the implemented algorithm is 0.936 ± 0.071. Using just 50 trees with a single feature per node is not sufficient, leading to an accuracy of 0.786, much lower than the one obtained with 4 features, both with 50 and 100 trees, reaching an accuracy value of 0.983. Analogous to the contact lenses case, the features of this data set are almost evenly distributed in terms of importance, which makes it crucial to be able to consider as many as them as possible in every split to maximize the classification accuracy. In fact, accuracy is proportional to the number of split features (0.983 with 4 and with 3 in the case of 50 trees, 0.977 with different combinations of 3 and 2 features, 0.954 with 2 features and 50 trees, 0.85 with 1 feature and 100 trees, yet 0.786 with 1 feature and 50 trees). The most important features to evaluate the quality of a car are its number of doors, safety and maintenance, all of them close to an importance value of 0.2.

Table 5: Accuracy for each Random Forest configuration for the car evaluation data set.

Tree number	Split feature number	Accuracy	Features by importance
50	1	0.786	('safety', 0.193), ('persons', 0.175), ('buying', 0.172), ('doors', 0.162), ('maint', 0.158), ('lug_boot', 0.141)
50	2	0.954	('maint', 0.18), ('doors', 0.179), ('persons', 0.165), ('safety', 0.164), ('lug_boot', 0.162), ('buying', 0.151)
50	3	0.983	('doors', 0.206), ('lug_boot', 0.184), ('maint', 0.176), ('buying', 0.163), ('persons', 0.161), ('safety', 0.111)
50	4	0.983	('doors', 0.227), ('lug_boot', 0.202), ('maint', 0.17), ('persons', 0.153), ('buying', 0.15), ('safety', 0.099)
100	1	0.85	('safety', 0.189), ('persons', 0.176), ('maint', 0.169), ('buying', 0.164), ('doors', 0.159), ('lug_boot', 0.143)
100	2	0.977	('doors', 0.18), ('maint', 0.173), ('buying', 0.173), ('lug_boot', 0.162), ('persons', 0.159), ('safety', 0.153)
100	3	0.977	('doors', 0.205), ('maint', 0.183), ('lug_boot', 0.18), ('buying', 0.167), ('persons', 0.156), ('safety', 0.108)
100	4	0.983	('doors', 0.224), ('lug_boot', 0.207), ('maint', 0.165), ('buying', 0.153), ('persons', 0.151), ('safety', 0.099)

4.6. Global comparison with the scikit-learn implementation

Table 6 shows a comparison between the mean accuracy obtained for all the data sets with the current implementation of the Random Forest classifier, and the implementation provided by the scikit-learn library, when set to use Gini index as decision criterion. The mean accuracies have been obtained using the same training and test splits and the same configurations of the two Random Forest parameters described in the previous sections. It can be seen that the results are very similar (equal in labor and with less than 0.05 of difference in hepatitis, where the distance is highest), since they are based on the same core Random Forest algorithm, and both use the Gini index as decision criteria, as done in traditional CART models. The only factor that may explain that the results of this implementation are slightly higher are the fact that the scikit-learn algorithm has a minimum instance number to perform a split (with the recommended default value of 2), which is not present in this implementation (so the implicit value is 1). This allows to obtain more precise, instance-specific splits, that would probably lead to overfitting in a single decision tree, but that are compensated by the usage of different trees, so that in the end this extra level of detail becomes a valuable prediction resource. The scikit-learn model, however, is significantly faster due to the parallelization of the decision tree calculations, divided in multiple threads that take advantage of multi-processing functionality.

Table 6: Comparison of this implementation and the scikit-learn Random Forest classifier, in terms of the mean accuracy results for each data set.

Data set	Accuracy (this implementation)	Accuracy (scikit-learn)
Contact lenses	**0.933 ± 0.094**	0.9 ± 0.1
Labor	**1.0 ± 0.0**	**1.0 ± 0.0**
Hepatitis	**0.852 ± 0.043**	0.805 ± 0.021
Breast cancer	**0.964 ± 0.007**	0.955 ± 0.005
Car evaluation	**0.936 ± 0.071**	0.928 ± 0.018

5. Code execution instructions

The code files of this work is found in the "Source" folder. In particular, the "main.py" is the one to be executed, by running the following shell command: "python3 -m main". It is required to have Python 3 installed, e.g using "sudo apt-get install python3.6" in a Linux machine. An alternative to the terminal is to open "main.py" in a Python IDE (e.g. PyCharm) and press the run button.

6. Conclusions and future work

The experiments performed in this work show how a Random Forest classifier built as the combination of many decision trees that perform a majority vote to classify instances allows to obtain high prediction accuracy, even in challenging data sets such as labor (with many missing values) and breast cancer (with inconsistent instances with same features but different class). While individual decision trees can suffer overfitting when creating splits based on a very small number of instances, the generalization capacity obtained by having multiple decision trees built with different samples of the same data eliminate the problem, and take advantage of the extra level of depth that individual trees can now have.

As future work, it would be interesting to test the developed algorithm in a greater number of data sets, under a larger number of parameter combinations, including aspects such as the minimum sample size to perform a split, to try to empirically confirm that the lower it is the higher accuracy can be obtained by learning more details while the majority vote compensates the over-specification of individual trees. Additionally, it would be a good idea to parallelize the algorithm, as done by the scikit-learn library, so that multiple threads are used to build different decision trees at the same time, and synchronized to form a single forest in the end.

7. References

[1] Breiman L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.

[2] Chipman, Hugh A., Edward I. George, Robert E. McCulloch (1998). Bayesian CART model search. *Journal of the American Statistical Association*. 93.443: 935-948.

[3] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[4] Pedregosa F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *JMLR 12*, pp. 2825-2830.