Assignment 5

Computer Science Department, University of Crete
MACHINE LEARNING - CS 577, Fall 2022

Student: Mourouzidou Eleni, Msc Bioinformatics

Exercise 1 - Entropy and Decision Trees (Theoretical)

	Body Length	Bold letters	susp. adress	Virus
s1	23	0	Yes	¬Virus
s2	18	1	No	¬Virus
s3	43	0	Yes	¬Virus
s4	68	0	No	Virus

Table 1: **Train Set**

	Body Length	Bold letters	susp. adress	Virus
s5	20	1	Yes	Virus
s6	25	1	No	¬Virus
s7	60	0	Yes	Virus
s8	35	0	No	¬ Virus

Table 2: **Test Set**

What is the entropy of Virus at the root?

We need to calculate the entropy of the attribute Virus at the root, the starting point of the decision tree before any further splits take place. The goal is to minimize the entropy while traversing the decision tree by selecting the best attributes to split in every node.

We will calculate the entropy of the virus in the root, using the following formula:

$Entropy_{Virus\ at\ the\ Root} = -p_{virus} * \log_2(p_{virus}) - p_{\neg virus} * \log_2(p_{\neg virus})$

So, using the train set we have 4 emails and 3 out of them are classified as “no-virus” while 1 out of 4 is classified as “virus”.
p(virus) = ¼
p(¬virus) = ¾
now if we plug in these values to the entropy formula we get:

$Entropy_{Virus\ at\ the\ Root} = -¼ * log_2(¼) - ¾ * log_2(¾)$

$Entropy_{Virus\ at\ the\ Root} = -¼ * (-2) - ¾ * (-0.415)$
$Entropy_{Virus\ at\ the\ Root} = 0.5 + 0.31125$
$Entropy_{Virus\ at\ the\ Root} = 0.811125$

Here we could assume that since the entropy of the class virus at the root is relatively high the training set is good for learning.

Which attribute should you choose as the root of a decision tree?

In order to decide which attribute to choose as the root of the decision tree we need to compute the information gain for each attribute and finally pick the one with the highest information gain. as it is considered the most “useful” for discriminating.

The first attribute is numerical so we should avoid splitting the branches for each value to not overfit the model. Thus, we need to choose a threshold “t” in a way that will make the decision binary (short body length vs long body length) as :
branch 1 : value for attribute $X < t$
branch 2 : value for attribute $X \geq t$

We will calculate the threshold as the median value for attribute “body length”.
body length = {23, 18, 43, 68}
sorted body length = {18, 23, 43, 68}
median(body length) = $\frac{23+43}{2}$ = 33, t = 33
and in the train dataset the emails for attribute “body length” could be characterized as: S1, S2 : short (<t), S3, S4 : long (>t).

Now we can calculate the entropy and information gain for this attribute as we handle categorical variables.

Body Length:

Short : $\frac{2}{4}$, p(virus) = $\frac{0}{2}$, p(¬virus) = $\frac{2}{2}$

Long : $\frac{2}{4}$, p(virus) = $\frac{1}{2}$, p(¬virus) = $\frac{1}{2}$

Entropy $_{\text{Body Length}}$ = $\frac{2}{4}$ * (-$\frac{0}{2}$log$_2$($\frac{0}{2}$) - $\frac{2}{2}$*log($\frac{2}{2}$)) +$\frac{2}{4}$* -$\frac{1}{2}$log$_2$($\frac{1}{2}$) - $\frac{1}{2}$*log($\frac{1}{2}$))=

= $\frac{2}{4}$ * (-0.5*(-1) - 0.5*(-1)) = 0.5,

Information Gain $_{\text{Body Length}}$ = Entropy$_{\text{Virus at the Root}}$ - Entropy$_{\text{Body Length}}$ = 0.811 - 0.5 = 0.311

Information Gain $_{\text{Body Length}}$ = 0.311

Bold Letters

Bold : $\frac{1}{4}$, p(virus) = $\frac{0}{1}$, p(¬virus) = $\frac{1}{1}$

Not Bold : $\frac{3}{4}$, p(virus) = $\frac{1}{3}$ $p(\neg virus)$ $\frac{2}{3}$

Entropy $_{\text{Bold Letters}}$ = $\frac{1}{4}$*(-$\frac{0}{1}$$log(\frac{0}{1}$ − $\frac{1}{1}$*log($\frac{1}{1}$))+ $\frac{3}{4}$*(- $\frac{1}{3}$ * $log(\frac{1}{3})$ -$\frac{2}{3}$ * $log(\frac{2}{3})$) =

= $\frac{3}{4}$ * (- ⅓ * (-1.5875) - ⅔ * (-0.585)) = ¾ * (0.529 + 0.39) = ¾*0.919 = 0.6892

Information Gain $_{\text{Bold Letters}}$ = Entropy$_{\text{Virus at the Root}}$ - Entropy$_{\text{Bold Letters}}$ = 0.811 - 0.689 = 0.1217

Information Gain $_{\text{Bold Letter}}$ = 0.1217

Suspicious Address

Suspicious : $\frac{2}{4}$, p(virus) = $\frac{0}{2}$, p(¬virus) = $\frac{2}{2}$

Not suspicious : $\frac{2}{4}$, p(virus) = $\frac{1}{2}$, p(¬virus) = $\frac{1}{2}$

Entropy $_{\text{Suspicious}}$ = $\frac{2}{4}$ * (-$\frac{0}{2}$* log$_2$($\frac{0}{2}$) - $\frac{2}{2}$ *log$_2$($\frac{2}{2}$))+$\frac{2}{4}$(-$\frac{1}{2}$*log$_2$($\frac{1}{2}$) - $\frac{1}{2}$*log$_2$($\frac{1}{2}$))=

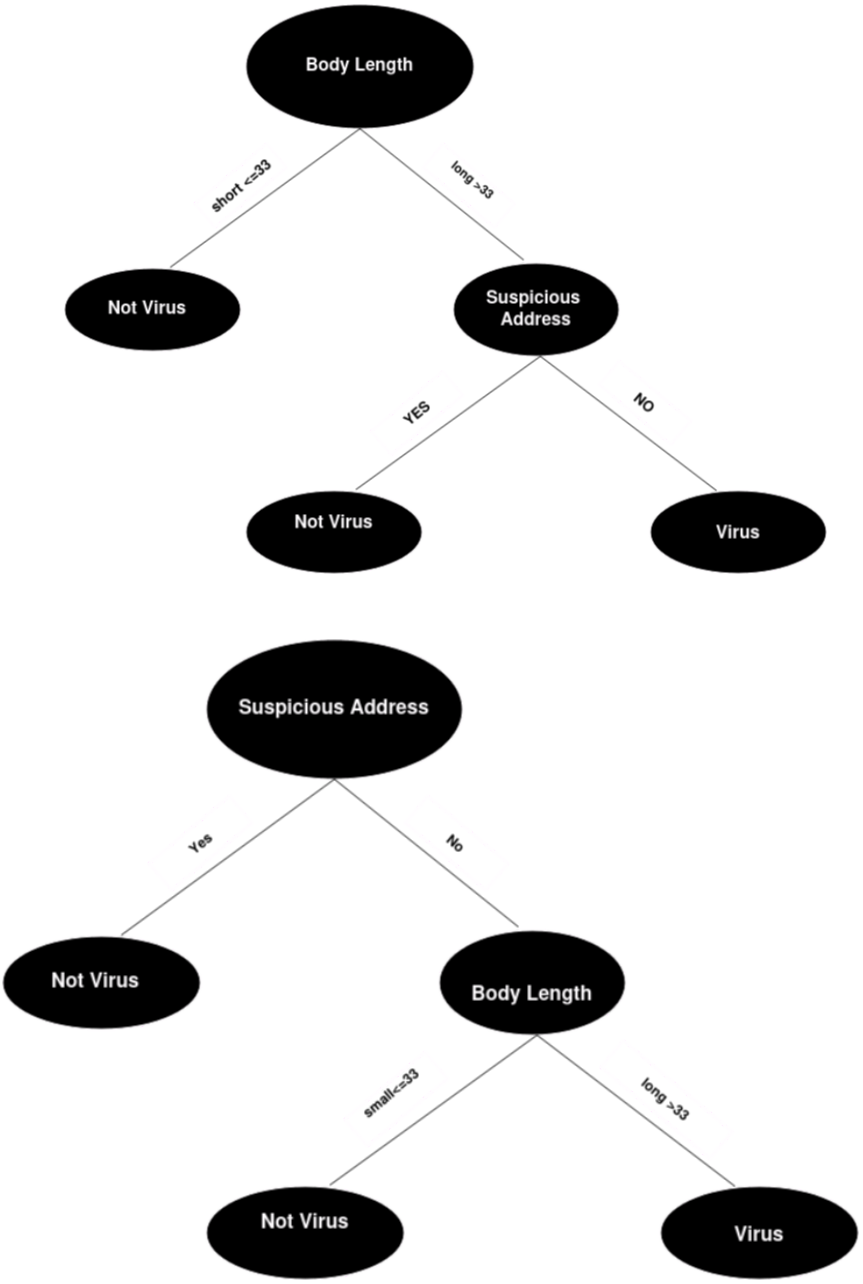
= $\frac{2}{4}$ *($\frac{1}{2}$ + $\frac{1}{2}$) = 0.5

Information Gain $_{\text{Suspicious Adress}}$ = Entropy$_{\text{Virus at the Root}}$ - Entropy$_{\text{Suspicious Adress}}$ = 0.811 - 0.5 = 0.311

Information Gain $_{\text{Suspicious Address}}$ = 0.311

Suspicious Address and Body Length attributes, have the highest Information Gain score, so we should choose one of these as the root of the decision tree. To decide which one of these two attributes we could use as a root, maybe it would be reasonable to calculate again the Information Gain after the first split for both cases and then choose the one that seems to be more effective.

Build the decision tree to classify as virus or not.



Classify the test set samples and report the accuracy.

S5:
Body length : short → No Virus (Failed to classify ✖)

S6:
Body length : short → No Virus (Classified correctly ✓)

S7:
Body length : long → Suspicious Address : Yes → No virus (Failed to classify ✖)

S8:
Body length : long → Suspicious Address : No → Virus (Failed to classify ✖)

The accuracy is ¼ or 25%

Exercise 2 - Random Forest (Programming)

Part A

Function TrainRF(X, Y, n_trees, min_samples_leaf):

This function is defined to train the random forest classifier for continuous data. The function takes the following inputs:

X: a numpy array with the sample data which will be used to train the model
Y: a 1-D numpy array containing the labels - class variable to be predicted
n_trees: the number of decision trees to grow in the ensemble procedure
min_samples_leaf: the minimum number of observations - samples that are required to create a leaf node for each decision tree

Functionality:
As required in the question, we first determine the maximum number of features to be considered for each split of a decision tree as the square root of the total feature number. We initialize the list *trees* to store every decision tree, and we then perform iterations over the specified n_trees. Then, np.random.choice will be used to perform the bootstrapping - generate random indices of the samples to be used in each tree - with replacement. The function DecisionTreeClassifier from sklearn.tree is called to generate the required number of decision trees with the following parameters:

min_samples_leaf, max_features, as explained above. Finally, tree.fit will train the decision trees on the sample data and the trained trees will be appended to the list *trees*. The function returns the model - a dictionary containing the list of decision trees, the mean_samples leaf and max_features parameters.

Function PredictRF(model, X):

This function is defined to make predictions of labels - classes of new data using the output Random Forest model trained from the TrainRF function. The function takes the following inputs:
model: the model - dictionary trained by the function TrainRF
X: the numpy array with the new data to be classified

Functionality:
First we initialize an np.zeros array to store the predictions of each individual tree of the Random Forest. Then, we iterate over every decision tree and use tree.predict method (sklearn.trees) to make predictions for the new data based on each decision tree. The predictions will be stored in the corresponding column of the *predictions* array. Finally apply argmax function along the axis of decision trees to the predictions array. This will determine the most frequent class for each data sample and return it as the "most voted" final prediction - return *predictions* array.

Part B

Test your implementations on the provided dataset: "Dataset5_XY "
- **Split the data into a training (70%) and a test sample (30%)**
- **Estimate the accuracies of each individual tree in the forest, and compare them against the one obtained using the prediction from the Random Forest.**

We defined a function called *calculate_accuracy* that takes the following inputs:
tree: an individual decision tree
X: a numpy array with the observed dataset
Y: a numpy array with the labels of the observed dataset
The function returns the percentage of the correctly classified samples out of the total number of samples according to the specified input decision tree

We also defined the function *plot_accuracy_histogram(ax, individual_accuracy, rf_accuracy, title)*. The function generates a histogram of individual decision tree

accuracies and adds vertical lines that indicate the mean accuracy of the individual decision trees and the accuracy of the Random Forest model.

Data preparation

- Load the csv dataset using pandas
- Separate the dataset to X features and Y labels- is house valuable (last column of the dataset).
- Split the data to 70 % training set and 30% test set.

Random Forest Train-Predict

- Train multiple Random Forest models with different min_samples_leaf(1, 10) in this case
- Calculate the individual accuracies of each tree and the overall Random Forest's accuracy, as calculated in the function *calculate_accuracy*

Observations

- Discrepancy between Average Tree Accuracy and Forest Accuracy:
    - The overall Random Forest's accuracy seems to be higher than the mean individual tree accuracy. This can be observed both in the case of min_samples_leaf=1 and mean_samples_leaf=10
    - This, indicates that the ensemble method applied in the Random Forest classifier improves the prediction performance. Individual decision trees in the forest are trained on different subsets after applying the bootstrapping method (randomly sample the data with replacement). Thus, there is a higher diversity in individual trees since the nodes are splitted based on different patterns of the data. As we can also observe in the histograms below, Random Forest outperforms the individual decision trees, since the forest has a higher ability to generalize and avoid overfitting to the training set.
- Impact of changes in min_samples_leaf
    - Given the histograms shown below (Fig1), we can tell that the discrepancy between average tree accuracy and forest accuracy was larger for min_samples_leaf=1 compared to min_samples_leaf=10. This indicates that with a smaller minimum number of samples required in each leaf node, the individual trees tend to be less accurate, leading to a

wider gap between their average accuracy and the overall forest accuracy.

- For min_samples_leaf=1, individual trees are more likely to have higher complexity (more nodes) as they could easily split a leaf node based on outliers, compared to min_samples_leaf=10 where a certain amount of samples is required in order to make a decision.
- Trees with lower min_samples_leaf tend to overfit more to the training data.

Thus, based on the observations above, the min_samples_leaf parameter plays a crucial role in balancing the complexity and accuracy of random forest models. While lower values may lead to overfitting, higher values can result in underfitting. The observed discrepancy between average tree accuracy and forest accuracy highlights the importance of selecting an appropriate min_samples_leaf value to optimize the performance of our random forest model.

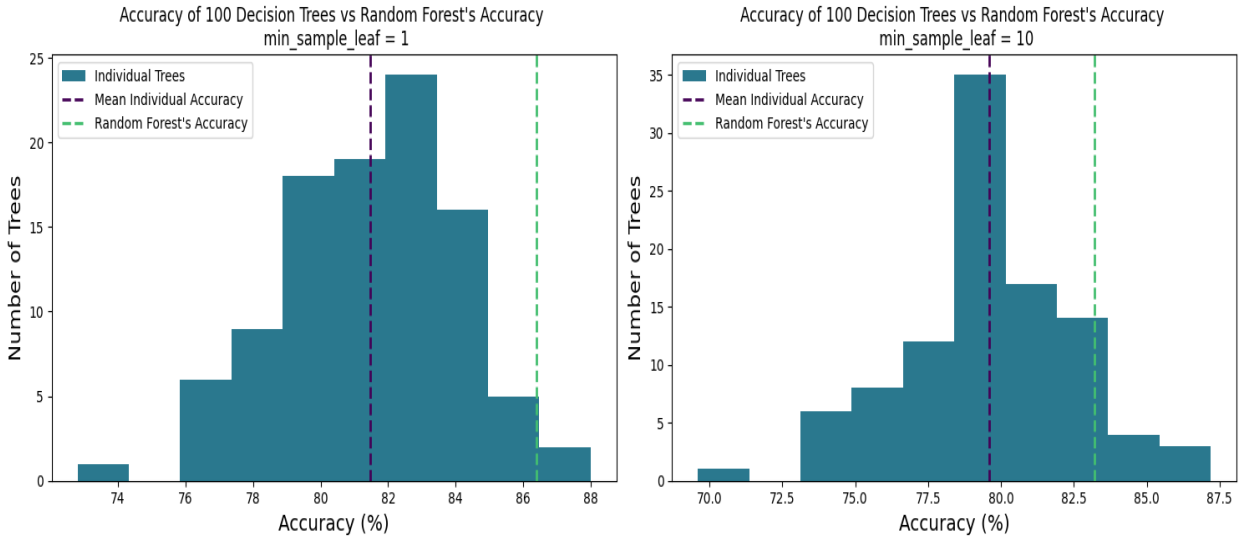


Fig1. Visualization of the comparison between accuracies of individual trees and random forest, for mean_samples_leaf=1(left histogram) and mean_samples_leaf=10 (right histogram).

BONUS

Perform the proper comparison between the accuracy obtained from a basic decision tree, and your Random Forest algorithm. Create an histogram for the accuracies of the single decision trees, and use an arrow to indicate the histogram mean value and the accuracy of the Random Forest.Calculate the probability that if you used a single tree, you would obtain a similar or better accuracy than using the Random Forest

To implement this comparison (accuracy between individual trees - without ensembling them - and random forest) and also define the probability of achieving similar or better accuracy with a single tree, we followed these steps:

1. Train the desired (num_trees = 1000) number of decision trees
2. Evaluate their individual accuracies (without having performed bootstrap sampling before)
3. Train Random Forest model as we implemented before (*TrainRf* function) - store the predicted classes (output of *PredictRf* function), and calculate the overall accuracy of random forest.
4. Calculate the probability of achieving similar or better accuracy with a single decision tree by iterating through the *individual_accuracies* list and count the number of accuracies that are greater than or equal to the Random Forest accuracy. This count is then divided by the total number of trees to get the probability.

```
Probability of achieving similar or better accuracy with a single decision tree: 0.015
```

5. Visualize the distribution of individual tree accuracies by also plotting a vertical line at the mean accuracy of the individual trees and the accuracy of the random forest (*Fig2*).

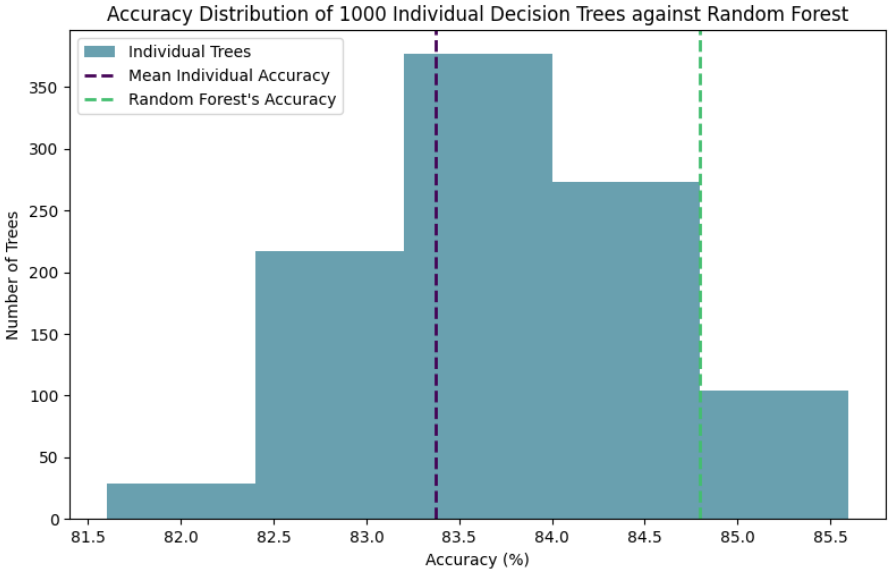


Fig2. Distribution of individual tree accuracies compared to the Random Forest accuracy

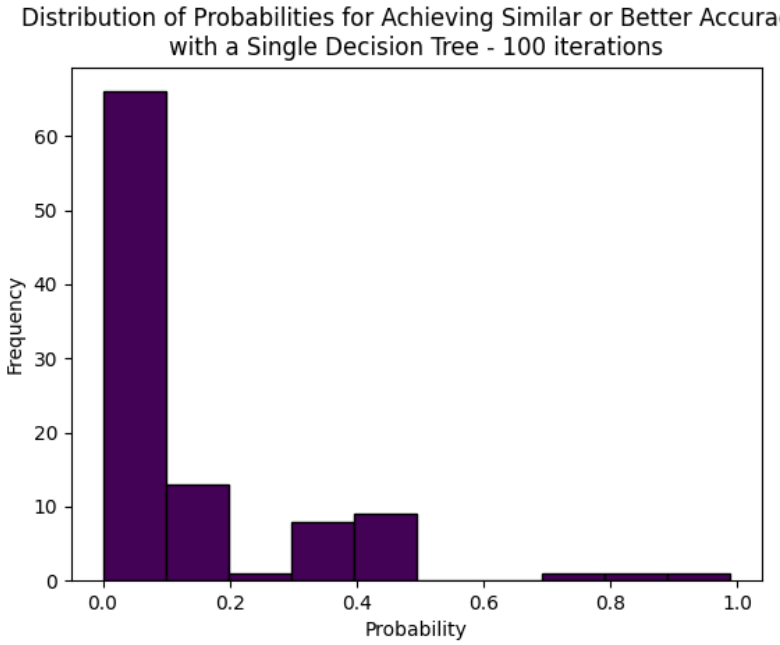


Fig3. Distribution of probabilities of achieving higher or similar accuracy of single decision tree comparing to Random forest - after 100 iterations

The probability of achieving a similar or better accuracy with a single decision tree as compared to the Random Forest appears to be very low for the majority of the iterations performed (Fig3). This suggests that it is not very likely to achieve similar or higher accuracy with a single decision tree. The histogram (*Fig2*) shows that the distribution of individual decision tree accuracies is wide, with some trees achieving very low accuracies and others achieving very high accuracies. This suggests that the performance of a single decision tree is highly variable. The Random Forest, on the other hand, is able to achieve a more consistent level of accuracy by averaging the predictions of multiple trees.