INSA
ROUEN NORMANDIE

Informatique et Technologie de l'Information 4
Année universitaire 2024-2025

Practical Work

Decision Trees

simon.bernard@univ-rouen.fr

1 Decision tree classification with Scikit-learn

In this practical session, you will use the scikit-learn library to test decision tree classifiers. This first part gives you some code snippets that show you step-by-step how to do it, using the Iris dataset as a toy example. Please, read the online documentation of this dataset before testing the following scripts.

1. Load the Iris dataset and check that it corresponds to the description given in the documentation.

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Desc. of the dataset
print(iris.feature_names)
print(iris.target_names)
print(...)
```

2. Plot the data in a 2D plot using the two features 'petal length' and 'petal width'.

```
import matplotlib.pyplot as plt

# 2D Plot using the first the features: petal length and petal width
plt.figure(figsize=(10, 6))
...
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.legend()
plt.show()
```

3. Build a decision tree classifier on the entire Iris dataset and plot the resulting tree structure.

```
from sklearn.tree import plot_tree

# Build a decision tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X, y)

# Plot the decision tree
plt.figure(figsize=(15, 10))
tree.plot_tree(clf, filled=True, feature_names=iris.feature_names,
    class_names=iris.target_names)
plt.show()
```

   Check that you obtain the exact same structure (first 4 levels) as the one shown in the lecture slides.

4. Plot the decision frontier of the decision tree classifier in the 2D plot.

```
from sklearn.inspection import DecisionBoundaryDisplay

# Retrain the classifier on the 2D data
X_2d = X[:, 2:4]
clf = DecisionTreeClassifier(random_state=42).fit(X_2d, y)

# Plot the decision boundary with the data
_, ax = plt.subplots(1, 1, figsize=(10, 6))
DecisionBoundaryDisplay.from_estimator(clf, X_2d, ax=ax,
    xlabel=iris.feature_names[2], ylabel=iris.feature_names[3], alpha=0.6)
plt.scatter(X_2d[:, 0], X_2d[:, 1], c=y, edgecolor='k', s=20)
plt.show()
```

2 Decision trees and overfitting

This second part aims to study the overfitting situations when using decision trees. To do so, we consider a synthetic dataset made up with two non-linearly separable classes, generated using the `make_classification` function from scikit-learn :

1. Referring to the documentation, generate the dataset containing 2000 instances from two classes and described with 20 informative features. To make the classification task a little more difficult, set the `class_sep` parameter to a value slightly less than 1.0.

```
from sklearn.datasets import make_classification
X, y = make_classification(..., random_state=11)
```

2. Split the dataset into a training set and a test set using the `train_test_split` function from scikit-learn. Keep 20% of the data for the test set.

```
from sklearn.model_selection import train_test_split
...
```

The next step is to observe overfitting situations when training a decision tree. We saw during the course that the deeper a tree is, the more likely it is to overfit the training data. We thus propose to monitor the evolution of the error as a function of the depth of a tree :

1. Read the scikit-learn documentation to see how to set the maximum depth of a decision tree.
2. Train 20 decision trees on the training set, with depths ranging from 1 to 20.

```
from sklearn.tree import DecisionTreeClassifier

# Train 20 decision trees with depths ranging from 1 to 20
for depth in range(1,20):
    depth.append(i)
    dt = DecisionTreeClassifier(...)
   ...
```

3. For each tree, compute the training and test errors and plot them as a function of the depth of the tree.
4. Which of the 20 trees would you say overfit ? Which of these trees do you think performs best ?

3 Combining decision trees

For the last part, we propose to test the combination of several decision trees and to compare the results obtained with those of each tree taken individually. To do this, we will use the `make_moons` dataset from scikit-learn, which generates two moon-shaped classes.

- First, train 16 decision trees on random replicas of the dataset. Then, display the decision frontier of each tree on a separate figure to observe the differences from one tree to another.

```
_, axes = plt.subplots(4, 4, figsize=(15, 10))
for i in range(16):
   # Generate a random dataset of 100 instances.
   # Set a different random seed for each one, but always keep the same noise level
    X, y = make_moons(...)

   # Train a decision tree classifier on X and y
   # (Do not use any depth limitation during learning)
   ...

   # Plot the data on the axes[i//4, i%4] subplot
   ...

   # plot the decision frontier on the same subplot
   ...
```

- Give a brief analysis of the phenomenom illustrated by these figures.

- Second, combine 16 similar decision trees (same hyperparametrization) by majority voting. To do this, use the `BaggingClassifier` class from scikit-learn (cf. documentation) and set the number of trees to 16.

```
# Generate a random dataset of 100 instances.
X, y = make_moons(...)

# Train a combination à 16 decision trees, each one trained on a random
# replica of the dataset containing the same amount of instances as in X
forest = BaggingClassifier(n_estimators=16).fit(X, y)

# Plot the 'soft' decision frontier of the forest
# 'Soft' decision frontiers shows uncertainty regions with lighter colors
...
```

  Note : Bagging is a method that we will explain in detail in the second chapter of the course, but for the moment remember that using scikit-learn's default parameterisation, one can train `n_estimators` decision trees on random replicas of the dataset and combine them by majority voting.

- Rerun this experiments with different number of instances and different noise levels and observe the impact on the decision frontier.
- Finally, rerun this experiments several times with an increasing number of decision trees in the combination and observe the impact on the decision frontier.
- What can you conclude from this about the minimum number of decision trees to combine that you would recommend for this problem ?