Decision Trees

Decision Trees are classification methods that are able to extract simple rules about the data features which are inferred from the input dataset. Several algorithms for decision tree induction are available in the literature. Scikit-learn contains the implementation of the CART (Classification and Regression Trees) induction algorithm.

Practical examples

Fist of all, we do all necessary imports.

In [1]:

```
import pandas as pd
import graphviz

from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Setting random seed.
seed = 10
```

Dataset with continuous features

Next, we load the Iris dataset, extract its values and labels and split them into train and test sets.

In [2]:

```
# Loading Iris dataset.
data = pd.read_csv('data/iris.csv')

# Creating a LabelEncoder and fitting it to the dataset labels.
le = LabelEncoder()
le.fit(data['Name'].values)

# Converting dataset str labels to int labels.
y = le.transform(data['Name'].values)

# Extracting the instances data.
X = data.drop('Name', axis=1).values

# Splitting into train and test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.34, stratify=y, random_state=seed)
```

Then, we will fit and test a DecisionTreeClassifier. Scikit-learn does not implement any post-prunning step. So, to avoid overfitting, we can control the tree size with the parameters min_samples_leaf, min_samples_split and max_depth.

In [3]:

```
# Creating a DecisionTreeClassifier.
# The criterion parameter indicates the measure used (possible values: 'gini' for the Gini index and
# 'entropy' for the information gain).
# The min_samples_leaf parameter indicates the minimum of objects required at a leaf node.
# The min_samples_split parameter indicates the minimum number of objects required to split an internal node.
# The max_depth parameter controls the maximum tree depth. Setting this parameter to None will grow the
# tree until all leaves are pure or until all leaves contain less than min_samples_split samples.
tree = DecisionTreeClassifier(criterion='gini',
                              min_samples_leaf=5,
                              min_samples_split=5,
                              max_depth=None,
                              random_state=seed)

tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('DecisionTreeClassifier accuracy score: {}'.format(accuracy))
```

```
DecisionTreeClassifier accuracy score: 0.9615384615384616
```

Finally, we can plot the obtained tree to visualize the rules extracted from the dataset.

In [4]:

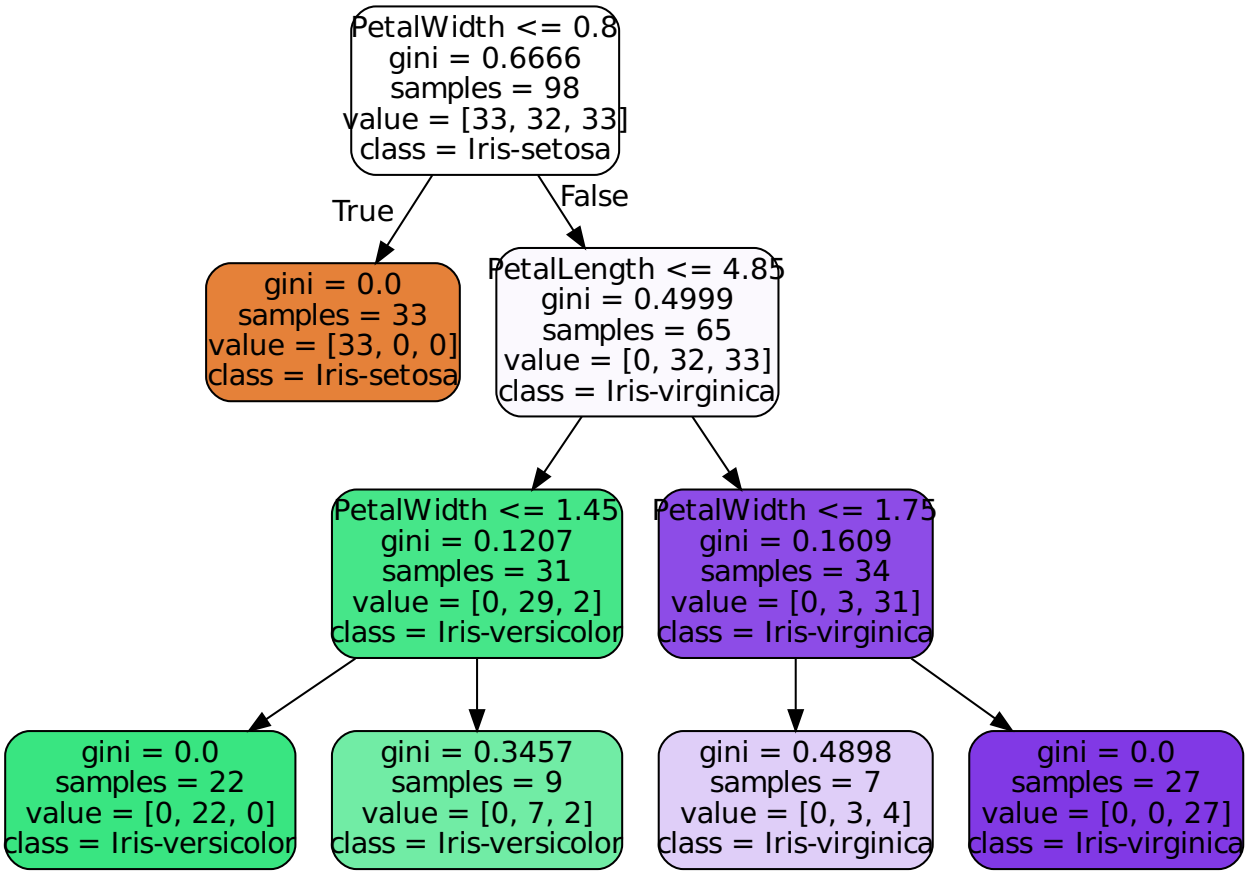
```
def plot_tree(tree, dataframe, label_col, label_encoder, plot_title):
    label_names = pd.unique(dataframe[label_col])

    # Obtaining plot data.
    graph_data = export_graphviz(tree,
                                 feature_names=dataframe.drop(label_col, axis=1).columns,
                                 class_names=label_names,
                                 filled=True,
                                 rounded=True,
                                 out_file=None)

    # Generating plot.
    graph = graphviz.Source(graph_data)
    graph.render(plot_title)
    return graph

tree_graph = plot_tree(tree, data, 'Name', le, 'Iris')
tree_graph
```

Out[4]:



Dataset with categorical features

Unfortunately, the DecisionTreeClassifier class does not handle categorical features directly. So, we might consider to transform them to dummy variables. **However, this approach must be taken with a grain of salt because decision trees tend to overfit on data with a large number of features.**

In [5]:

```
# Loading Mushroom dataset.
data = pd.read_csv('data/mushroom.csv')

# We drop the 'stalk-root' feature because it is the only one containing missing values.
data = data.drop('stalk-root', axis=1)

# Creating a new DataFrame representation for each feature as dummy variables.
dummies = [pd.get_dummies(data[c]) for c in data.drop('label', axis=1).columns]

# Concatenating all DataFrames containing dummy variables.
binary_data = pd.concat(dummies, axis=1)

# Getting binary_data as a numpy.array.
X = binary_data.values

# Getting the labels.
le = LabelEncoder()
y = le.fit_transform(data['label'].values)

# Splitting the binary dataset into train and test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.34, stratify=y, random_state=seed)

# Creating a DecisionTreeClassifier.
tree = DecisionTreeClassifier(criterion='gini',
                              min_samples_leaf=5,
                              min_samples_split=5,
                              max_depth=None,
                              random_state=seed)

tree.fit(X_train, y_train)
```

Out[5]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=5,
            min_samples_split=5, min_weight_fraction_leaf=0.0,
            presort=False, random_state=10, splitter='best')
```

Now, we will apply the obtained tree on the test set.

In [6]:

```
y_pred = tree.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('DecisionTreeClassifier accuracy score: {}'.format(accuracy))
```

```
DecisionTreeClassifier accuracy score: 0.9992761491132827
```

We can observe that the above decision tree is pretty accurate.

Now, let's check its depth.

In [7]:

```
print('DecisionTreeClassifier max_depth: {}'.format(tree.tree_.max_depth))
```

```
DecisionTreeClassifier max_depth: 6
```

What if we fit a decision tree with a smaller depth?

In [8]:

```
# Creating a DecisionTreeClassifier.
tree = DecisionTreeClassifier(criterion='gini',
                              min_samples_leaf=5,
                              min_samples_split=5,
                              max_depth=3,
                              random_state=seed)

tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('DecisionTreeClassifier accuracy score: {}'.format(accuracy))
```

```
DecisionTreeClassifier accuracy score: 0.9659790083242852
```

We can observe that the new tree is almost as accurate as the first one. Apparently both trees are able to handle the mushroom data pretty well. The second three might be preferred, since it is a simpler and computationally cheaper model.

Finally, we plot the second tree.

In [9]:

```
# Appending 'label' column to binary DataFrame.
binary_data['label'] = data['label']

tree_graph = plot_tree(tree, binary_data, 'label', le, 'Mushroom')
tree_graph
```

Out[9]: