Bishop's University

CS 405/CS 505 – Data Mining

Assignment 1: Decision Trees

1. Introduction

The objective of this assignment is to demonstrate the implementation of decision trees for classification and regression problems. This document freely reproduces some examples shown in the excellent scikit-learn documentation.

2. Decision Trees

Decision trees are nonparametric learning methods used for classification and regression problems. The goal is to create a model that predicts the values of the target variable, based on a set of sequences of decision rules inferred from the training data. The tree therefore approximates the target by a succession of if-then-else rules. This paradigm applies to both categorical and numerical data. The more complex the tree generated, the better the model "explains" the learning data but the higher the risk of over-fitting.

Decision trees have several advantages that make them interesting in contexts where it is useful to understand the sequence of decisions made by the model:

- They are easy to understand and visualize.
- They require little data preparation (normalization, etc.).
- The cost of using trees is logarithmic.
- They can use categorical and numerical data.
- They can deal with multi-class problems.
- White box model: the result is easy to conceptualize and visualize.

However, these models have two major disadvantages:

- Over-fitting: sometimes the generated trees are too complex and generalize badly. Choosing good values for the maximum depth (*max_depth*) and minimum number of samples per leaf (*min_samples_leaf*) parameters avoids this problem.
- It may happen that the generated trees are not balanced, which implies that the travel time is no longer logarithmic. It is therefore recommended to adjust the dataset before construction, to avoid that one class largely dominates the others in terms of the number of training examples.

3. Trees for classification

In scikit-learn, the class sklearn.tree.DecisionTreeClassifier allows to perform a multi-class classification using a decision tree.

We start by importing the right modules and building the tree object:

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
```

For the example, we can define a minimalist dataset (two points, each in a class):

```
X = [[0, 0], [1, 1]]
y = [0, 1]
```

The tree is built using the *.fit(X, y)* method:

```
clf = clf.fit(X, y)
```

Prediction on new samples is done in the usual way with *.predict(X)*:

```
clf.predict([[2., 2.]])
```

One can also predict the probability of each class for a sample:

```
clf.predict_proba([[2., 2.]])
```

4. Classification of Iris data

DecisionTreeClassifier is able to handle multi-class classification problems (eg with labels 0, 1, … K-1). In this example we will work with the Iris dataset, easily accessible in sklearn. This dataset contains 150 instances of iris (a type of plant, each observation describes its morphology). The objective is to classify each instance into one of three categories: Iris setosa, Iris virginica or Iris versicolor.

One of the classes is linearly separable with respect to the other two, but the other two are not separable with respect to each other.

The attributes of the dataset are:

- sepal length,
- sepal width,
- petal length,
- petal width,
- class: Iris Setosa, Iris Versicolor or Iris Virginica.

The Iris dataset being very common, scikit-learn offers a native function to load it into memory:

```
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target
```

Question 1: Calculate the statistics (mean and standard deviation) of the four explanatory variables: sepal length, sepal width, petal length and petal width.

Question 2: How many examples of each class are there?

Before building the model, let's split the dataset into two: 70% for training, 30% for testing.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
random_state=0)
```

We can now build a decision tree on this data:

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

Once the training is finished, we can visualize the tree, either with matplotlib using the *plot_tree* method, or with the *graphviz* tool (dot command). For example, with *matplotlib*:

```
tree.plot_tree(clf, filled=True)
```

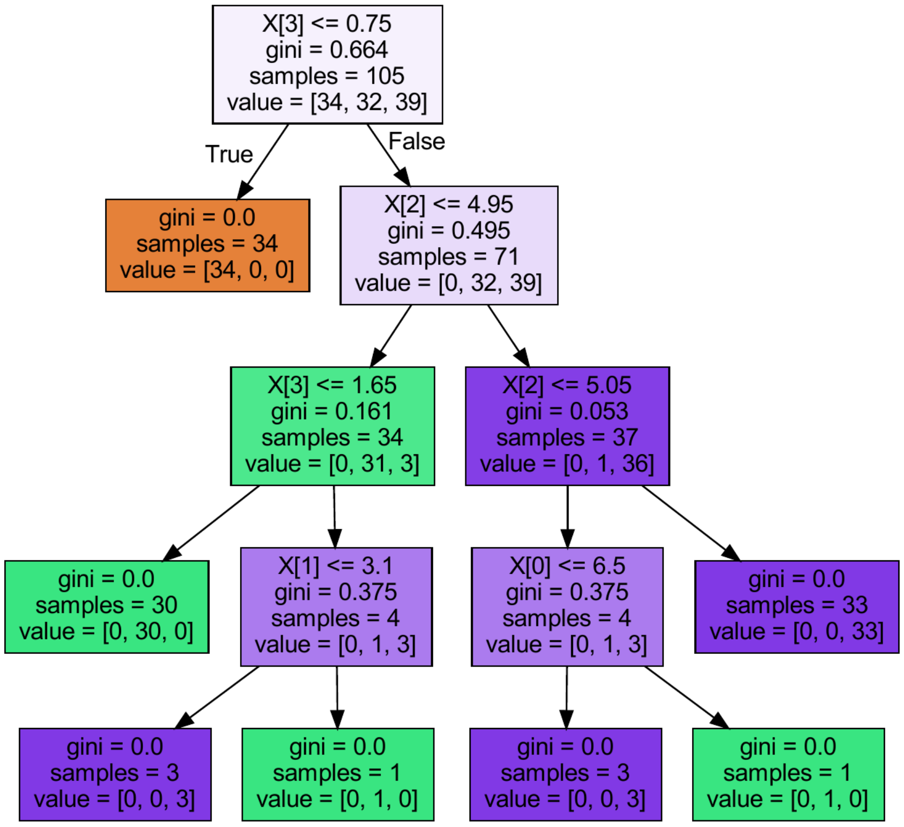
Alternatively, it is possible to export by producing a .dot file which is the default format of *graphviz*:

```
# Export the graph to the iris.dot file
with open("iris.dot", 'w') as f:
    f = tree.export_graphviz(clf, out_file=f, filled=True)
```

Then, on the command line, it is possible to convert this file into many formats, for example into *PDF* (shell command):

```
%%bash
dot -Tpdf iris.dot -o iris.pdf
```

The generated image should look like this:



Once the model is built, it is possible to use it for prediction on new data:

```
clf.predict(X_test)
```

The test score can be calculated this way:

```
clf.score(X_test, y_test)
```

Question 3: Change the *max_depth* and *min_samples_leaf* parameter values. What do you notice?

Question 4: The problem here being particularly simple, redo a training/test division with 5% of the data in training and 95% test. Calculate the rate of misclassified items on the test set. Vary (or better, perform a grid search with *GridSearchCV*) the values of the *max_depth* and *min_samples_leaf* parameters to measure their impact on this score.

5. Display of the decision surface

For a pair of attributes, i.e., for two-dimensional observations, we can visualize the decision surface in 2 dimensions. First, we discretize the two-dimensional domain with a constant step and then we evaluate the model on each point of the grid.

In this example, we only keep the length and width of the petals.

```
import numpy as np
import matplotlib.pyplot as plt

# Settings
n_classes = 3
plot_colors = "bry" # blue-red-yellow
plot_step = 0.02

# Choose the length and width attributes of the petals
pair = [2, 3]

# We only keep the two attributes
X = iris.data[:, pair]
y = iris.target

```

```python
# Tree learning
clf = tree.DecisionTreeClassifier().fit(X, y)

# Display of the decision surface
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step), np.arange(y_min,
y_max, plot_step))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
plt.xlabel(iris.feature_names[pair[0]])
plt.ylabel(iris.feature_names[pair[1]])
plt.axis("tight")

# Display of learning points
for i, color in zip(range(n_classes), plot_colors):
    idx = np.where(y == i)
    plt.scatter(X[idx,        0],        X[idx,        1],        c=color,
label=iris.target_names[i], cmap=plt.cm.Paired)
plt.axis("tight")
plt.suptitle("Decision surface of a decision tree using paired features")
plt.legend()
plt.show()
```

Question 5: Redo the display for the other pairs of attributes. On which pair is the separation between the classes the most marked?

6. Decision trees for regression

For regression with decision trees, scikit-learn offers the *DecisionTreeRegressor* class. As for the classification, the *fit(...)* method takes as input the parameter *X* (attributes of the observations). Warning: the y are not class labels but real values.

```python
from sklearn import tree

X = [[0, 0], [2, 2]]
y = [0.5, 2.5]
```

```
clf = tree.DecisionTreeRegressor()
clf = clf.fit(X, y)
clf.predict([[1, 1]])
```

In the following example, we will construct a sinusoidal signal affected by white noise and we will train a regression tree on this training data.

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeRegressor

# Create the training data
np.random.seed(0)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel()

fig = plt.figure(figsize=(12, 4))
fig.add_subplot(121)
plt.plot(X, y)
plt.title("Pure sine wave")

# We add a random noise every 5 samples
y[::5] += 3 * (0.5 - np.random.rand(16))
fig.add_subplot(122)
plt.plot(X, y)
plt.title("Noisy sine wave")
```

The objective is to regress this signal y from the values of x. For this, we use a regression tree.

```
# Learn the model
reg = DecisionTreeRegressor(max_depth=2)
reg.fit(X, y)

# Prediction on the same range of values
X_test = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
y_pred = reg.predict(X_test)

# Display of results
plt.figure()
plt.scatter(X, y, c="darkorange", label="Training Examples")
plt.plot(X_test,  y_pred,  color="cornflowerblue",  label="Prediction",
linewidth=2)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Decision Tree Regression")
plt.legend()
plt.show()
```

Question 6: Change the value of the *max_depth* parameter. What happens if we take too large a value? Too small? Change the rate of elements affected by noise (the *y[::5]*). When all elements are affected by noise, should a high or low value for *max_depth* be preferred?

Question 7: To deepen, load the Diabetes dataset from the *sklearn.datasets* module and make a random partition into learning part and test part (70% learning, 30% testing). Build a regression tree model on this basis. Calculate the root mean square error on the test set. Do a grid search to find the *max_depth* parameter value that minimizes this error.

7. Submission

You must submit the *pdf* file of your report. In fact, it must contain the responses for the seven questions. Indeed, your explanation must justify which one of the four visualization techniques will help doctors to make a relevant decision. In addition, you have to submit your *.ipynb*. ***Please, do not submit your assignment in .zip or .rar files.***

8. Useful external references

- NumPy documentation
- SciPy documentation
- MatPlotLib documentation
- scikit-learn
- Python programming language