

# COMP 6721 Applied Artificial Intelligence (Fall 2021)

## Lab Exercise #5: Decision Trees & K-means

**Question 1** Given the training instances below, use information theory to find whether ‘Outlook’ or ‘Windy’ is the best feature to decide when to play a game of golf.

| Outlook  | Temperature | Humidity | Windy | Play / Don't Play |
|----------|-------------|----------|-------|-------------------|
| sunny    | 85          | 85       | false | Don't Play        |
| sunny    | 80          | 90       | true  | Don't Play        |
| overcast | 83          | 78       | false | Play              |
| rain     | 70          | 96       | false | Play              |
| rain     | 68          | 80       | false | Play              |
| rain     | 65          | 70       | true  | Don't Play        |
| overcast | 64          | 65       | true  | Play              |
| sunny    | 72          | 95       | false | Don't Play        |
| sunny    | 69          | 70       | false | Play              |
| rain     | 75          | 80       | false | Play              |
| sunny    | 75          | 70       | true  | Play              |
| overcast | 72          | 90       | true  | Play              |
| overcast | 81          | 75       | false | Play              |
| rain     | 71          | 80       | true  | Don't Play        |

**Question 2** It's time to leave the calculations to your computer: Write a Python program that uses `scikit-learn`'s *Decision Tree Classifier*.<sup>1</sup>

---

```
import numpy as np
from sklearn import tree
from sklearn import preprocessing
```

---

Here is the training data from the first question:

---

```
dataset = np.array([
    ['sunny', 85, 85, 0, 'Don\'t Play'],
    ['sunny', 80, 90, 1, 'Don\'t Play'],
    ['overcast', 83, 78, 0, 'Play'],
    ['rain', 70, 96, 0, 'Play'],
    ['rain', 68, 80, 0, 'Play'],
    ['rain', 65, 70, 1, 'Don\'t Play'],
    ['overcast', 64, 65, 1, 'Play'],
    ['sunny', 72, 95, 0, 'Don\'t Play'],
    ['sunny', 69, 70, 0, 'Play'],
    ['rain', 75, 80, 0, 'Play'],
    ['sunny', 75, 70, 1, 'Play'],
    ['overcast', 72, 90, 1, 'Play'],
    ['overcast', 81, 75, 0, 'Play'],
    ['rain', 71, 80, 1, 'Don\'t Play'],
])
```

---

Note that we changed `True` and `False` into `1` and `0`.

For our feature vectors, we need the first four columns:

---

```
X = dataset[:, 0:4]
```

---

and for the training labels, we use the last column from the dataset:

---

```
y = dataset[:, 4]
```

---

However, you will not be able to use the data as-is: All features must be numerical for training the classifier, so you have to transform the strings into numbers. Fortunately, `scikit-learn` has a preprocessing class for *label encoding* that we can use:<sup>2</sup>

---

```
le = preprocessing.LabelEncoder()
X[:, 0] = le.fit_transform(X[:, 0])
```

---

(Note: you will need to transform `y` as well.)

---

<sup>1</sup>See <https://scikit-learn.org/stable/modules/tree.html#classification>

<sup>2</sup>See [https://scikit-learn.org/stable/modules/preprocessing\\_targets.html#preprocessing-targets](https://scikit-learn.org/stable/modules/preprocessing_targets.html#preprocessing-targets)

Now you can create a classifier object:

---

```
dtc = tree.DecisionTreeClassifier(criterion="entropy")
```

---

Note that we are using the *entropy* option for building the tree, which is the method we've studied in class and used on the exercise sheet. Train the classifier to build the tree:

---

```
dtc.fit(X, y)
```

---

Now you can predict a new value using `dtc.predict(test)`, just like you did for the Naïve Bayes classifier last week. Note: if you want the string output that you transformed above, you can use `inverse_transform(predict)` function of a label encoder to change the `predicted` result back into a string.

You can also print the tree:

---

```
tree.plot_tree(dtc)
```

---

but this can be a bit hard to read; to get a prettier version you can use the *Graphviz* visualization software,<sup>3</sup> which you can call from Python like this:

---

```
# print a nicer tree using graphviz
import graphviz
dot_data = tree.export_graphviz(dtc, out_file=None,
    feature_names=['Outlook', 'Temperature', 'Humidity', 'Windy'],
    class_names=['Don't Play', 'Play'],
    filled=True, rounded=True)
graph = graphviz.Source(dot_data)
graph.render("mytree")
```

---

The result will be stored in a file `mytree.pdf` and should look like Figure 1.

---

<sup>3</sup>See <https://www.graphviz.org/>

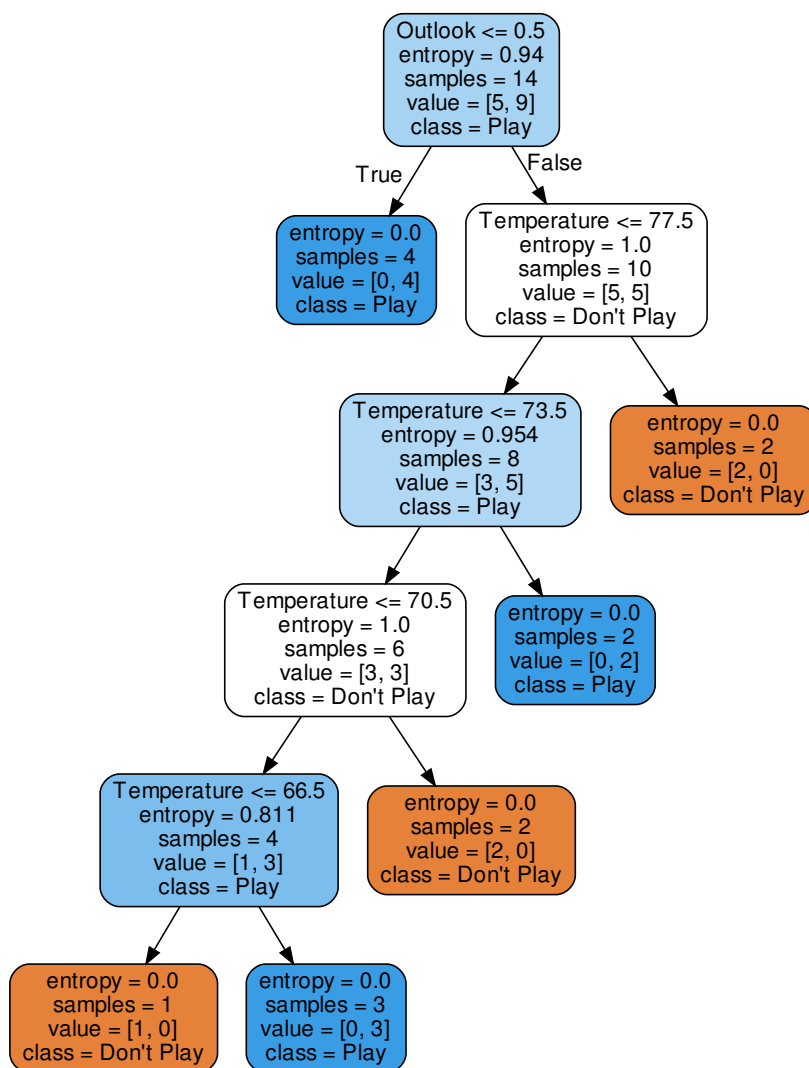


Figure 1: Generated Decision Tree using scikit-learn: Note that the string values for *Outlook* have been changed into numerical values ('overcast'= 0, 'rain'= 1, 'sunny' = 2)

**Question 3** Let's start working with some “real” data: `scikit-learn` comes with a number of example datasets, including the *Iris* flower dataset. If you do not know this dataset, start by reading [https://en.wikipedia.org/wiki/Iris\\_flower\\_dataset](https://en.wikipedia.org/wiki/Iris_flower_dataset).<sup>4</sup>

- (a) Change your program from the previous question to work with this new dataset:

---

```
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
```

---

Train a decision tree and print it out like before.

- (b) Now you have to *evaluate* the performance of your classifier. Use `scikit-learn`'s `train_test_split` helper function<sup>5</sup> to split the Iris dataset into a *training* and *testing* subset, as discussed in the lecture. Now run an *evaluation* to compute the *Precision*, *Recall*, *F<sub>1</sub>-measure*, and *Accuracy* of your classifier using the evaluation tools in `scikit-learn`.<sup>6</sup>
- (c) Finally, compute and print out the *confusion matrix*.<sup>7</sup>



Figure 2: An *Iris versicolor*<sup>8</sup>

---

<sup>4</sup>Also see [https://scikit-learn.org/stable/auto\\_examples/datasets/plot\\_iris\\_dataset.html](https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html) for the `scikit-learn` documentation of this dataset.

<sup>5</sup>See [https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html)

<sup>6</sup>See [https://scikit-learn.org/stable/modules/model\\_evaluation.html](https://scikit-learn.org/stable/modules/model_evaluation.html)

<sup>7</sup>See [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion\\_matrix.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)

<sup>8</sup>Photo taken by Danielle Langlois in July 2005 at the Forillon National Park of Canada, Quebec, Canada. Licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license. [https://commons.wikimedia.org/wiki/File:Iris\\_versicolor\\_3.jpg](https://commons.wikimedia.org/wiki/File:Iris_versicolor_3.jpg)

**Question 4** Consider the following data set with two attributes for each of seven individuals.

|       | a1  | a2  |
|-------|-----|-----|
| Data1 | 1.0 | 1.0 |
| Data2 | 1.5 | 2.0 |
| Data3 | 3.0 | 4.0 |
| Data4 | 5.0 | 7.0 |
| Data5 | 3.5 | 5.0 |
| Data6 | 4.5 | 5.0 |
| Data7 | 3.5 | 4.5 |

- (a) The data set is to be grouped into two clusters. Initialize the clusters using Data1 and Data4 as initial centroids to the two clusters. That is, allocate the remaining individual to one of the two clusters using the Euclidean distance.

|         | Individuals | Centroid   |
|---------|-------------|------------|
| Group 1 | 1           | (1.0, 1.0) |
| Group 2 | 4           | (5.0, 7.0) |

- (b) Recalculate the centroids based on the current partition, reassign the individuals based on the new centroids. Which individuals (if any) changed clusters as a result?

**Question 5** scikit-learn also has an implementation for K-means.<sup>9</sup>

- (a) Complete clustering the data from the previous question using scikit-learn's K-means implementation:

---

```
dataset = np.array([
    [1.0, 1.0],
    [1.5, 2.0],
    [3.0, 4.0],
    [5.0, 7.0],
    [3.5, 5.0],
    [4.5, 5.0],
    [3.5, 4.5]])
```

---

Note that with K-means, you do not need any labels, since we are doing unsupervised learning:

---

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2)
kmeans.fit(dataset)
```

---

To print the results, you can use:

---

```
print(kmeans.labels_)
```

---

This will show you for each element in the `dataset` which cluster it belongs to.

- (b) Now cluster the *Iris* flower dataset: How well do the clusters represent the three species of *Iris*?

Using *Matplotlib*,<sup>10</sup> you can create a 3D-plot of your clusters:

---

```
# Plot the results using matplotlib
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10, 8))
plot = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azimuth=134)
plot.set_xlabel('Petal width')
plot.set_ylabel('Sepal length')
plot.set_zlabel('Petal length')
plot.scatter(X[:, 3], X[:, 0], X[:, 2], c=kmeans.labels_, edgecolor='k')
```

---

The result should look similar to Figure 3.

---

<sup>9</sup>See <https://scikit-learn.org/stable/modules/clustering.html#k-means>

<sup>10</sup>See <https://matplotlib.org/>

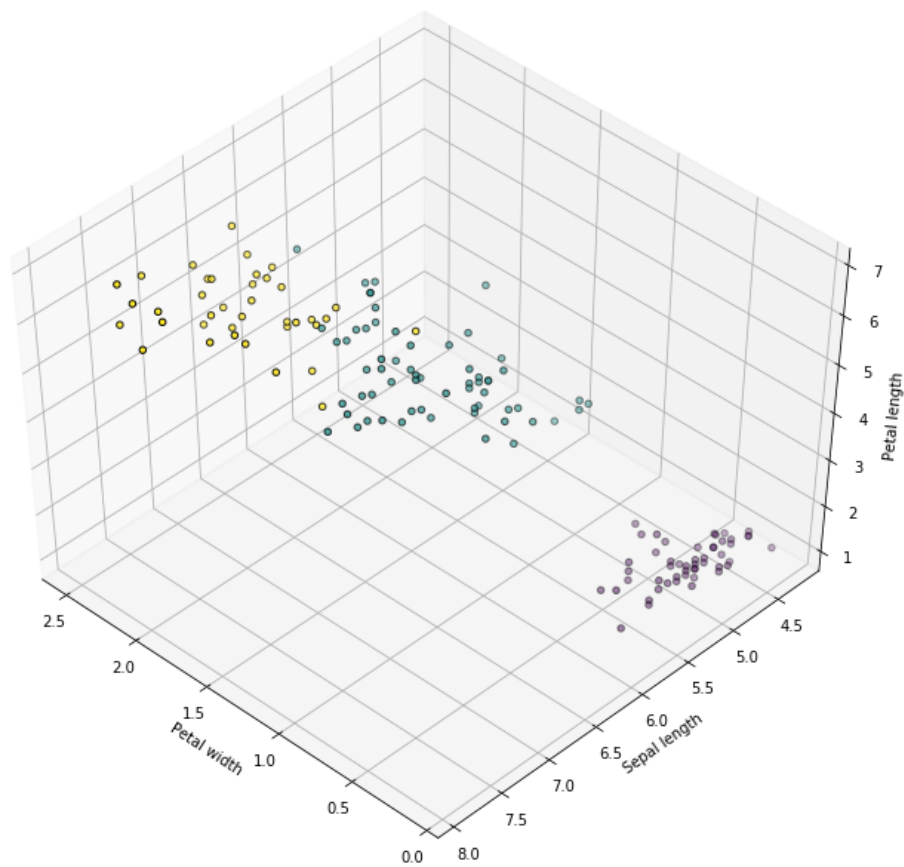


Figure 3: Plotting the results of clustering with K-means on the *Iris* dataset with Matplotlib