Project Activity: Automated Decision Trees

Group Names and Roles

- David (driver)
- Lauren (proposer)
- rashi (gone :( )

Introduction

Last time, you used `pandas` summary tables to create some informed guesses about good *decision trees* -- flow-chart-like rules for making a guess about the species of a penguin. In this activity, we'll use `scikit-learn` to automatically create superior decision trees.

Let's begin by importing all the libraries we'll need, and by downloading the penguins dataset:

If you experience `ConnectionRefused` errors when doing this, instead copy/paste the url into your browser. Save the data in the same directory as this notebook in a file called `penguins.csv`, and then replace `url` with `"penguins.csv"` in the block below.

In [44]:
```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn import tree, preprocessing
import numpy as np
```

In [45]:
```
url = "https://philchodrow.github.io/PIC16A/datasets/palmer_penguins.csv"
penguins = pd.read_csv(url)
```

§1. Preparing your data

For this activity, we will use only the following columns: `"Species"`, `"Flipper Length (mm)"`, `"Body Mass (g)"`, `"Sex"`. (Use the square brackets operator on the list of these strings, and **assign the result back to `penguins`**.)

In [46]:
```
cols = penguins[['Species', 'Flipper Length (mm)', 'Body Mass (g)', 'Sex']]
```

Next, inspect the `penguins` data frame. You should have 344 rows and 4 columns.

In [47]:
```
cols.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 4 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Species              344 non-null    object 
 1   Flipper Length (mm)  342 non-null    float64
 2   Body Mass (g)        342 non-null    float64
 3   Sex                  334 non-null    object 
dtypes: float64(2), object(2)
memory usage: 10.9+ KB
```

You might have noticed that your dataframe contains rows with `NaN` values. Calling `.dropna()` on the dataframe will remove these rows. Do this below, and reassign the result back to `penguins`.

In [48]:
```
penguins = cols.dropna()
```

Look at your dataframe once again. You should have 334 rows and 4 columns.

Note: The sex of one of the penguins was not recorded, and the corresponding entry in the `Sex` column is `.`. This won't cause issues, but feel free if you like to remove this row. Now is a good time to do this.

In [49]:
```
penguins.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 334 entries, 0 to 343
Data columns (total 4 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Species              334 non-null    object 
 1   Flipper Length (mm)  334 non-null    float64
 2   Body Mass (g)        334 non-null    float64
 3   Sex                  334 non-null    object 
dtypes: float64(2), object(2)
memory usage: 13.0+ KB
```

Run the next cell. Doing this will make sure that the random values that your code will generate will be the same every time you run the code.

In [50]:
```
np.random.seed(3354354524)
```

Our goal is to build a model that *predicts the species of a penguin based on the other features* that you now have in the `penguins` dataframe. With this in mind, clean the data, as follows:

- obtain slices `X` and `y` (predictor variables and target variable) from the `penguins` dataframe
- encode the sex (inside `X`) and the species (`y`) of the penguins as integers

In [51]:
```
x_slice = penguins[["Flipper Length (mm)", "Body Mass (g)", "Sex"]]
y_slice = penguins[['Species']]
```

To make sure that you know what is going on, look at your `X` and `y` variables by running the next cells.

In [52]:
```
x_slice.head()
```

Out[52]:

	Flipper Length (mm)	Body Mass (g)	Sex
0	181.0	3750.0	MALE
1	186.0	3800.0	FEMALE
2	195.0	3250.0	FEMALE
4	193.0	3450.0	FEMALE
5	190.0	3650.0	MALE

In [53]:
```
y_slice.head()
```

Out[53]:

	Species
0	Adelie Penguin (Pygoscelis adeliae)
1	Adelie Penguin (Pygoscelis adeliae)
2	Adelie Penguin (Pygoscelis adeliae)
4	Adelie Penguin (Pygoscelis adeliae)
5	Adelie Penguin (Pygoscelis adeliae)

In [32]:
```
le = preprocessing.LabelEncoder()
le.fit_transform(x_slice['Sex'])
```

Out[32]:
```
array([2, 1, 1, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2,
       1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2,
       1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
       1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2,
       1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
       1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
       1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2,
       1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
       1, 2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2, 1, 1, 2,
       1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 1,
       1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 1, 2, 2, 1,
       2, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 1, 2, 1,
       2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 2, 1,
       1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
       2, 1, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 0, 2, 1,
       1, 2, 1, 2])
```

In [57]:
```
#x_slice['Sex'] = le.fit_transform(x_slice['Sex'])
x_slice
```

Out[57]:

	Flipper Length (mm)	Body Mass (g)	Sex
0	181.0	3750.0	2
1	186.0	3800.0	1
2	195.0	3250.0	1
4	193.0	3450.0	1
5	190.0	3650.0	2
...	...	...	...
338	214.0	4925.0	1
340	215.0	4850.0	1
341	222.0	5750.0	2
342	212.0	5200.0	1
343	213.0	5400.0	2

334 rows × 3 columns

```
for sex, 2 represents male and 1 represents female
```

In [60]:
```
le.fit_transform(y_slice['Species'])
```

Out[60]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2])
```

```
for species, 0 = adelie, 1 = chinstrap 2, = gentoo
```

In [63]:
```
y_slice['Species'] = le.fit_transform(y_slice['Species'])
y_slice
```

```
<ipython-input-63-6e1106d3ef8d>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  y_slice['Species'] = le.fit_transform(y_slice['Species'])
```

Out[63]:

	Species
0	0
1	0
2	0
4	0
5	0
...	...
338	2
340	2
341	2
342	2
343	2

334 rows × 1 columns

Now split `X` and `y` into training and test data (80/20% of the rows).

Note: *You should conduct all splits using a single call to the function* `train_test_split` *from* `sklearn.model_selection` *.*

In [35]:
```
from sklearn.model_selection import train_test_split
```

In [36]:
```
x_train, x_test, y_train, y_test = train_test_split(x_slice, y_slice, test_size = 0.2)
```

In [38]:
```
```

Out[38]:
```
(     Flipper Length (mm)  Body Mass (g)     Sex
 238                209.0         4800.0  FEMALE
 123                202.0         3875.0    MALE
 233                213.0         5850.0    MALE
 176                195.0         3300.0  FEMALE
 124                186.0         3050.0  FEMALE
 ..                   ...            ...     ...
 262                210.0         4300.0  FEMALE
 200                187.0         3250.0    MALE
 224                215.0         5400.0    MALE
 145                185.0         3650.0    MALE
 111                191.0         4600.0    MALE

 [267 rows x 3 columns],
                                        Species
 238         Gentoo penguin (Pygoscelis papua)
 123       Adelie Penguin (Pygoscelis adeliae)
 233         Gentoo penguin (Pygoscelis papua)
 176  Chinstrap penguin (Pygoscelis antarctica)
 124       Adelie Penguin (Pygoscelis adeliae)
 ..                                         ...
 262         Gentoo penguin (Pygoscelis papua)
 200  Chinstrap penguin (Pygoscelis antarctica)
 224         Gentoo penguin (Pygoscelis papua)
 145       Adelie Penguin (Pygoscelis adeliae)
 111       Adelie Penguin (Pygoscelis adeliae)

 [267 rows x 1 columns])
```

§2. Training a model

Using the training data you generated in the previous part, train a decision tree classification model `T` with a `max_depth` value of 20. Score your model **against the training data**. Then score your model again, this time **against the test data**. Print both scores and observe the output.

In [ ]:
```
```

Again, using the training data you generated in the previous part, train another model, also named `T`. This time, use a `max_depth` value of 3. Just as above, score your model **against the training data**, and again **against the test data**. Print both scores and observe the output.

In [ ]:
```
```

Discuss your observations in the next cell. Which model is better? Is a model better when it performs better against the training data or the test data? Why does one model perform better against the training data while the other performs better against the test data?

---

your discussion here

Run the next cell to visualize your model.

In [ ]:
```
fig, ax = plt.subplots(1, figsize = (20, 20))
p = tree.plot_tree(T, filled = True, feature_names = X.columns)
```

§3. Cross-validation

Now estimate the optimal tree depth using cross-validation, and plot the results, as follows.

Make an empty plot. The x-axis will be the tree depth, and the y-axis will be the cross-validation score. Label your axes.

Make a `for` loop that will test a particular tree depth, between 1 and 30. On each iteration, train a decision tree model of the given depth and calculate its cross-validation score. Plot the depth and the score in your scatterplot. Then compare your score to the best score you have so far, and update the best score and best max depth if the new score is better.

In [ ]:
```
```

Lastly, train a decision tree classification model `T` using the best max depth. Score the model against the test data. Print the score and observe the output.

In [ ]:

```

```