In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, precision_recall_fscore_support, confusion_matrix, f1_score

seed = 1234
```

1. Classification and hold-out

1.1 Load 'abalone' dataset

In [2]:
```
# Load labels
ds = np.loadtxt('abalone.csv')
X = ds[:,:-1]
y_truth = ds[:,-1]

# Count items for each class
np.unique(y_truth, return_counts=True)
```

Out[2]:
```
(array([0., 1., 2.]), array([1306, 1342,  416]))
```

In [3]:
```
X.shape
```

Out[3]:
```
(3064, 8)
```

In [4]:
```
X
```

Out[4]:
```
array([[ 0.365 ,  0.255 ,  0.08  , ...,  0.0345,  0.053 ,  5.    ],
       [ 0.465 ,  0.38  ,  0.135 , ...,  0.1095,  0.22  , 14.    ],
       [ 0.365 ,  0.27  ,  0.085 , ...,  0.042 ,  0.058 ,  6.    ],
       ...,
       [ 0.61  ,  0.485 ,  0.165 , ...,  0.232 ,  0.38  , 11.    ],
       [ 0.62  ,  0.485 ,  0.17  , ...,  0.3045,  0.33  , 15.    ],
       [ 0.36  ,  0.265 ,  0.075 , ...,  0.0365,  0.055 ,  7.    ]])
```

In [5]:
```
y_truth.shape
```

Out[5]:
```
(3064,)
```

1.2 Create train and test splits

- Use the train_test_split() method
- Random state to make results repeatable

In [6]:
```
# Separate data into training and test set
# Default test_size = 0.25
X_train, X_test, y_train, y_test = train_test_split(X, y_truth,
                                                    test_size=0.2, random_state=42)
```

In [7]:
```
X_train.shape
```

Out[7]:
```
(2451, 8)
```

In [8]:
```
X_test.shape
```

Out[8]:
```
(613, 8)
```

In [9]:
```
y_train.shape
```

Out[9]:
```
(2451,)
```

1.3 Train classifier and make predictions

- Use Gaussian Naive Bayes classifier

In [10]:
```
clf = DecisionTreeClassifier(max_depth=2)
```

In [11]:
```
clf.fit(X_train, y_train)
```

Out[11]:
```
DecisionTreeClassifier(max_depth=2)
```

In [12]:
```
y_test_pred = clf.predict(X_test)
```

In [13]:
```
y_test
```

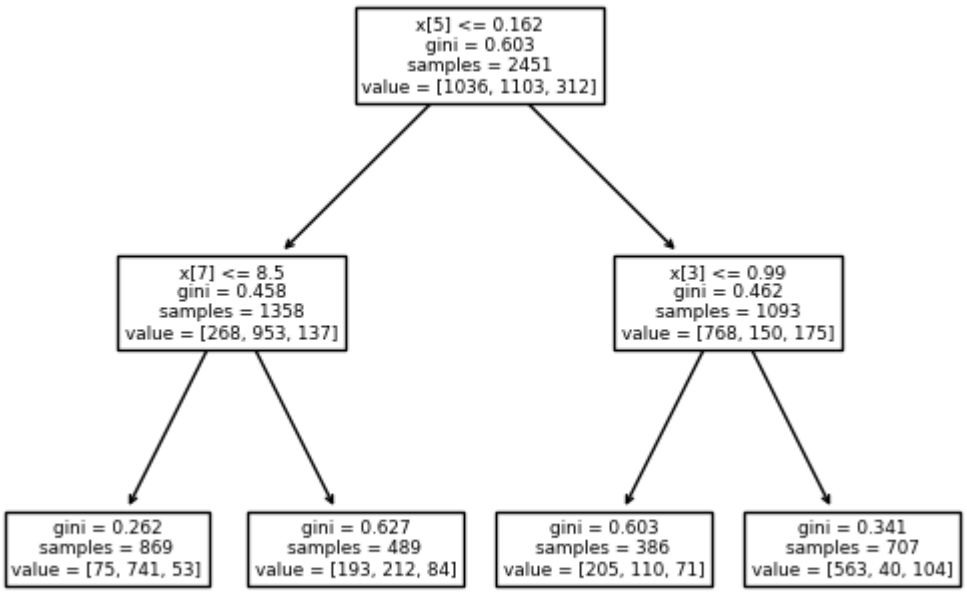
Out[13]:
```
array([0., 0., 2., 1., 0., 1., 1., 2., 2., 0., 1., 2., 1., 1., 1., 0., 1.,
       0., 0., 1., 0., 0., 0., 2., 0., 1., 1., 1., 0., 1., 2., 1., 0., 0.,
       1., 1., 2., 1., 1., 0., 2., 2., 1., 1., 1., 0., 1., 1., 0., 0., 1.,
       0., 0., 2., 0., 0., 2., 2., 0., 0., 1., 0., 1., 2., 1., 1., 0., 2.,
       1., 0., 2., 0., 0., 0., 2., 0., 1., 1., 2., 1., 1., 1., 0., 2., 1.,
       1., 1., 2., 2., 1., 1., 0., 1., 1., 1., 2., 0., 0., 0., 0., 2., 1.,
       0., 2., 0., 0., 1., 2., 0., 2., 0., 1., 1., 1., 0., 1., 0., 1., 1.,
       0., 0., 1., 1., 1., 2., 0., 0., 1., 1., 2., 0., 2., 0., 1., 1., 0.,
       2., 1., 0., 2., 0., 1., 0., 1., 1., 0., 0., 2., 2., 1., 0., 0., 1.,
       1., 0., 0., 1., 2., 1., 2., 0., 0., 0., 0., 2., 0., 1., 0., 0., 0.,
       0., 1., 1., 0., 2., 2., 0., 2., 2., 1., 0., 0., 2., 1., 1., 0., 1.,
       1., 1., 0., 0., 1., 0., 2., 0., 1., 0., 1., 0., 0., 1., 0., 1., 2.,
       0., 0., 1., 0., 2., 1., 0., 1., 0., 0., 1., 0., 0., 1., 1., 0., 2.,
       1., 1., 0., 0., 2., 2., 0., 0., 2., 2., 0., 1., 1., 1., 1., 1., 1.,
       1., 1., 2., 1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 0., 1., 2., 0.,
       0., 1., 2., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0.,
       1., 2., 1., 2., 0., 1., 2., 1., 0., 0., 2., 0., 0., 1., 0., 2., 0.,
       2., 0., 1., 0., 1., 0., 0., 0., 1., 2., 0., 1., 1., 0., 1., 2., 0.,
       1., 0., 1., 1., 2., 0., 2., 1., 0., 1., 0., 0., 1., 1., 2., 0., 1.,
       0., 1., 1., 0., 1., 1., 0., 0., 2., 0., 1., 1., 0., 0., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 1., 2., 1., 1., 1., 0., 2., 2., 2., 1., 0.,
       1., 1., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 1., 2., 2., 0., 1.,
       1., 0., 1., 1., 1., 1., 1., 0., 1., 0., 1., 0., 0., 0., 1., 2., 2.,
       2., 1., 1., 0., 0., 0., 0., 1., 1., 0., 1., 0., 2., 2., 0., 2., 2.,
       0., 1., 1., 0., 0., 2., 0., 2., 0., 0., 1., 2., 0., 0., 0., 2., 0.,
       0., 2., 2., 0., 0., 0., 0., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1.,
       0., 0., 1., 1., 2., 0., 1., 1., 1., 2., 1., 1., 0., 1., 1., 2., 1.,
       1., 2., 1., 0., 2., 2., 1., 1., 0., 1., 0., 0., 2., 0., 1., 1., 2.,
       1., 1., 2., 1., 0., 0., 1., 1., 1., 0., 1., 0., 1., 0., 0., 0., 1.,
       2., 0., 0., 1., 1., 0., 0., 0., 2., 1., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 1., 0., 0., 1., 1., 0., 1., 2., 2., 1., 0., 1., 1., 0., 0.,
       1., 0., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 0., 0., 1., 2., 0.,
       1., 0., 2., 0., 0., 0., 1., 0., 0., 0., 1., 2., 0., 1., 0., 0., 1.,
       1., 0., 0., 0., 2., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 2., 1.,
       2., 1., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0., 2., 1.,
       0., 1., 0., 0., 0., 1., 1., 0., 0., 2., 1., 0., 0., 1., 1., 1., 2.,
       1.])
```

In [14]:
```
y_test_pred
```

Out[14]:
```
array([0., 0., 0., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 0., 1.,
       0., 0., 1., 0., 0., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0.,
       0., 1., 0., 0., 1., 0., 1., 0., 1., 1., 1., 0., 1., 1., 0., 0., 1.,
       0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 1., 1., 1., 0., 0.,
       1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 0., 1., 1., 1., 0., 0., 0.,
       1., 0., 0., 0., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 1.,
       0., 1., 0., 1., 1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 1., 1., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0.,
       0., 1., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 1., 1., 1., 1., 1.,
       0., 0., 1., 1., 0., 1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 0., 1.,
       0., 0., 1., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1.,
       1., 1., 0., 1., 1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 1., 1.,
       0., 0., 1., 0., 1., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 0.,
       1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 1., 1., 1., 1., 1., 1.,
       1., 0., 0., 1., 1., 1., 0., 0., 1., 1., 0., 1., 0., 1., 1., 1., 0.,
       0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0.,
       1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0.,
       1., 1., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 1., 1., 1., 1., 1.,
       0., 1., 1., 0., 0., 2., 0., 2., 0., 0., 1., 2., 0., 0., 0., 2., 0.,
       0., 2., 2., 0., 0., 0., 0., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1.,
       0., 0., 1., 1., 2., 0., 1., 1., 1., 2., 1., 1., 0., 1., 1., 2., 1.,
       1., 2., 1., 0., 2., 2., 1., 1., 0., 1., 0., 0., 2., 0., 1., 1., 2.,
       1., 1., 2., 1., 0., 0., 1., 1., 1., 0., 1., 0., 1., 0., 0., 0., 1.,
       2., 0., 0., 1., 1., 0., 0., 0., 2., 1., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 1., 0., 0., 1., 1., 0., 1., 2., 2., 1., 0., 1., 1., 0., 0.,
       1., 0., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 0., 1., 1., 1., 0.,
       0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0.,
       1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 0., 0., 1., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0.,
       1., 1., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 1., 1., 0., 0., 1.,
       1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1., 1.,
       0., 1., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0., 1., 0., 1., 1., 0.,
       1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 0., 1., 1., 1., 1.,
       1.])
```

Visualiza Decision Tree

In [15]:
```
tree.plot_tree(clf)
plt.show()
```



1.4 Evaluate the results

- Evaluation using accuracy score

In [16]:
```
# Compute accuracy
acc = accuracy_score(y_test, y_test_pred)
print(f"Accuracy = {acc:.2f}")
```

```
Accuracy = 0.68
```

- **Accuracy** seems good, but if we look at the scores separately for each class...

In [17]:
```
# Precision, recall, f1, support: for each class
p, r, f1, support = precision_recall_fscore_support(y_test, y_test_pred)

for c in range(p.shape[0]):
    print(f"\nClass {c}:")
    print(f"number of items: {support[c]}")
    print(f"p = {p[c]:.2f}")
    print(f"r = {r[c]:.2f}")
    print(f"f1 = {f1[c]:.2f}")

# Macro average f1
macro_f1 = f1.mean()

# This score is important when you have class imbalancing
print(f"\nF1, macro-average: {macro_f1:2f}")
```

```
Class 0:
number of items: 270
p = 0.71
r = 0.74
f1 = 0.73

Class 1:
number of items: 239
p = 0.65
r = 0.90
f1 = 0.75

Class 2:
number of items: 104
p = 0.00
r = 0.00
f1 = 0.00

F1, macro-average: 0.493006
/Users/salvatorephd/opt/anaconda3/envs/mlds-env/lib/python3.8/site-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

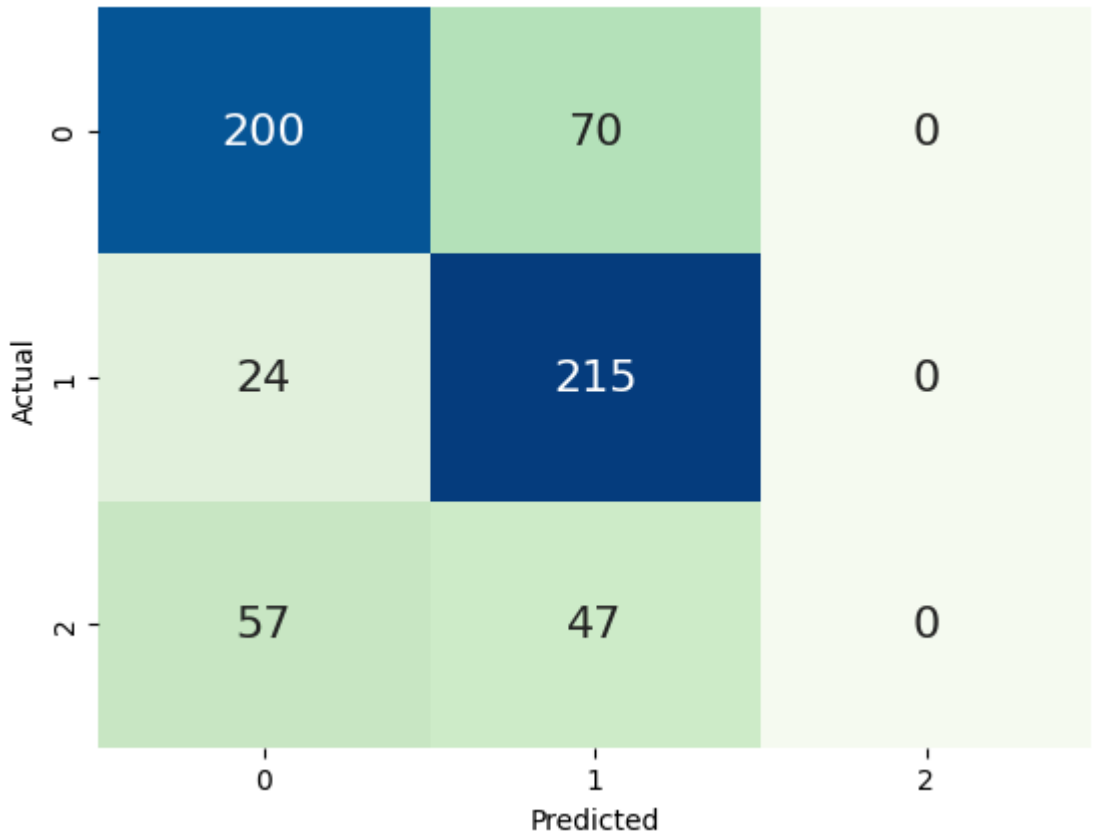
- Accuracy was good because of class imbalancing
- The **minority class** (c2) has a very low recall
- Indeed, the **macro-averaged** F1 is quite low.

Let's verify this with a confusion matrix:

In [18]:
```
# Build the confusion matrix
conf_mat = confusion_matrix(y_test, y_test_pred)

# Plot the result
```

```
label_names = np.arange(p.shape[0])
conf_mat_df = pd.DataFrame(conf_mat, index = label_names, columns = label_names)
conf_mat_df.index.name = 'Actual'
conf_mat_df.columns.name = 'Predicted'
sns.heatmap(conf_mat_df, annot=True, cmap='GnBu',
            annot_kws={"size": 16}, fmt='g', cbar=False)
plt.show()
```



1.5 Accuracy with unbalanced dataset

Assume we have an unbalanced binary classification problem with 100 samples (1 sample with class 0 and 99 samples with class 1).

In [19]:
```
gt_arr = [0]*1 + [1]*99
```

A classifier that always predict class 1

In [20]:
```
y_pred = [1]*len(gt_arr)
```

In [21]:
```
acc = accuracy_score(gt_arr, y_pred)
print(f"Accuracy = {acc:.2f}")
```

```
Accuracy = 0.99
```

In [22]:
```
f1 = f1_score(gt_arr, y_pred, average='macro')
print(f"F1 = {f1:.2f}")
```

```
F1 = 0.50
```

2. Cross-Validation

2.1 With kfold.split()

In [23]:
```
from sklearn.model_selection import KFold
# K-Fold with 5 splits
kfold = KFold(n_splits=5, shuffle=True)

print("Scores for each kfold iteration.")
i = 0
for train_indices, test_indices in kfold.split(X, y_truth):
    # Prepare splits
    X_train = X[train_indices]     # Use fancy indexing to extract data
    y_train = y_truth[train_indices]
    X_test = X[test_indices]
    y_test = y_truth[test_indices]

    # Train and evaluate
    clf = GaussianNB()
    clf.fit(X_train, y_train)
    y_test_pred = clf.predict(X_test)

    # Compute macro average f1
    _, _, f1, _ = precision_recall_fscore_support(y_test, y_test_pred)
    macro_f1 = f1.mean()

    print(f"Iteration {i}. macro-f1 = {macro_f1}")
    i+=1
```

```
Scores for each kfold iteration.
Iteration 0. macro-f1 = 0.5011657818978951
Iteration 1. macro-f1 = 0.5252797818682935
Iteration 2. macro-f1 = 0.5100069954327872
Iteration 3. macro-f1 = 0.4844360832032448
Iteration 4. macro-f1 = 0.5360837714020309
```

2.2 With cross_val_score()

- Use scoring = 'f1_macro'

In [24]:
```
from sklearn.model_selection import cross_val_score
```

In [25]:
```
clf = GaussianNB()
f1_cv = cross_val_score(clf, X, y_truth, cv=5, scoring='f1_macro')
```

In [26]:
```
print(f"Macro-f1 for each iteration: {f1_cv}")
mean_macro_f1 = f1_cv.mean()
std_macro_f1 = f1_cv.std() * 2
print(f"Macro-f1 (statistics): {mean_macro_f1:.2f} (+/- {std_macro_f1:.2f})")
```

```
Macro-f1 for each iteration: [0.50716433 0.49471464 0.50663244 0.52182634 0.51496524]
Macro-f1 (statistics): 0.51 (+/- 0.02)
```

2.3 Leave-One-Out and scoring: cross_val_predict()

- The previous approach (average of F1 for each iteration) cannot be used with leave one out.
  - Iteration 0: y_test = [1] -> F1?
  - Iteration 1: y_test = [0] -> F1?
  - ...
  - Iteration 2: y_test = [1] -> F1?
- When test set has only 1 sample, F1, precision and recall cannot be properly computed.
- The following solution trains N models with leave one out, fits them on test data to obtain the vector y_pred (each model predicts 1 single value inside y_pred). Finally, it computes a single score by comparing y_pred with y_truth

In [27]:
```
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import LeaveOneOut

clf = GaussianNB()
y_pred = cross_val_predict(clf, X, y_truth, cv=LeaveOneOut())
_, _, f1_loo, _ = precision_recall_fscore_support(y_truth, y_pred)
macro_f1_loo = f1_loo.mean()
print(f"F1, for each class: {f1_loo}")
print(f"Macro-f1 = {macro_f1_loo:.2f}")
```

```
F1, for each class: [0.71562952 0.76223533 0.04968944]
Macro-f1 = 0.51
```

In [ ]:
```
```