Feature Subset Selection Using Decision Tree on Abalone Dataset

We can use a decision tree to identify the most important features in a dataset. This notebook will demonstate this using the Abalone dataset.

About the dataset

The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope -- a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict the age. Further information, such as weather patterns and location (hence food availability) may be required to solve the problem. **Rings: +1.5 gives the age in years**

The dataset consists of the following columns:

- `Sex` : Gender of the abalone (M, F, I)
- `Length` : Length measurement
- `Diameter` : Diameter measurement
- `Height` : Height measurement
- `Whole weight` : Weight of the whole abalone
- `Shucked weight` : Weight after removing the shell
- `Viscera weight` : Weight of the gut
- `Shell weight` : Weight of the shell
- `Rings` : Age of the abalone (target variable)

To perform Feature Subset Selection we will take the following Steps:

1. **Load and preprocess the Data**: Read the CSV file to understand its structure and convert enerything to numerical columns.
2. **Decision Tree**: Train a decision tree model on the dataset.
3. **Feature Importance**: Sort all the feature by their importances from the decision tree model.
4. **Subset Selection**: Select a subset of features based on their importance.

In [1]:
```
# Step 1: Load and preprocess the Data: Read the CSV file to understand its structure and convert enerything to numerical columns.

import pandas as pd

# Load the data
file_path = 'abalone.csv'
df = pd.read_csv(file_path)

# Show the first few rows of the dataframe to understand its structure
df.head()
```

Out[1]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

Encode nominal attribute `sex` into a `numeric` attribute using LabelEncoder

We can use the `LabelEncoder` from the `sklearn` library to convert a nominal column to a numeric one.

LabelEncoder Example: Encode target labels with value between 0 and n_classes-1.

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(["paris", "paris", "tokyo", "amsterdam"])
print(list(le.classes_)) # prints ['amsterdam', 'paris', 'tokyo']
le.transform(["tokyo", "tokyo", "paris"])
```

In [2]:
```
from sklearn.preprocessing import LabelEncoder

# Encode the 'Sex' column
label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])
df.head()
```

Out[2]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	2	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	2	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	0	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	2	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	1	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

In [3]:
```
# Step 2: Decision Tree: Train a decision tree model on the dataset.

from sklearn.tree import DecisionTreeRegressor # For training a Decision Tree Regressor

# Split data into features and target variable
X = df.drop('Rings', axis=1)
y = df['Rings']

dt_regressor = DecisionTreeRegressor(random_state=0)
dt_regressor.fit(X, y)
```

Out[3]:
```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=0)
```

The `feature_importances_` attribute in a decision tree model gives you information about the importance of each feature used in making predictions. Here's a breakdown:

- **Feature Importance Values**:
  - Each feature in your dataset gets a score.
  - The scores show the relative importance of each feature.
  - Higher scores mean more important features.
- **How It's Calculated**:
  - Based on how much each feature contributes to splitting the data.
  - Features that better separate the classes get higher scores.
- **Uses**:
  - Understand which features are most influential in the model.
  - Potentially remove less important features to simplify the model.
- **Example**:
  - Imagine analyzing student performance.
  - Features might include study hours, attendance, and participation.
  - `feature_importances_` might show that study hours are the most important feature in predicting student grades.

In [4]:
```
# Step 3: Feature Importance: Sort all the feature by their importances from the decision tree model.

# Extracting feature importances
feature_importances = dt_regressor.feature_importances_

# Creating a DataFrame for feature importances
features_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

features_df.sort_values(by='Importance', ascending=False)
```

Out[4]:

	Feature	Importance
7	Shell weight	0.499676
5	Shucked weight	0.176053
4	Whole weight	0.081031
6	Viscera weight	0.068467
1	Length	0.054298
2	Diameter	0.049838
3	Height	0.049328
0	Sex	0.021309

In [5]:
```
# Step 4: Subset Selection: Select a subset of features based on their importance

feature_subset = features_df.sort_values(by='Importance', ascending=False).query('Importance > 0.1')
X_subset = X[list(feature_subset.Feature)]
```

X_subset only contains 2 columns but those are the 2 most important columns.

Part 2: Build models and compare

Now we will build two models. The first one will utilize all the 8 features and the second one will utilize the smaller dataset with only 2 features. Finally, we will calculate the error rate and compare them.

Part 2.1 Build a model using all the data

In [6]:
```
from sklearn.tree import DecisionTreeRegressor # For training a Decision Tree Regressor
from sklearn.metrics import mean_squared_error  # For calculating MSE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Load the data
file_path = 'abalone.csv'
df = pd.read_csv(file_path)

# Encode the 'Sex' column
label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])

# Split data into features and target variable
X = df.drop('Rings', axis=1)
y = df['Rings']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [7]:
```
# Training the Decision Tree Regressor
dt_regressor = DecisionTreeRegressor(max_depth = 3, random_state=0)
dt_regressor.fit(X_train, y_train)
```

Out[7]:
```
DecisionTreeRegressor(max_depth=3, random_state=0)
```

In [8]:
```
# Predict the target values for the same dataset
y_pred = dt_regressor.predict(X_test)
```

In [9]:
```
# Calculate the Mean Absolute Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 6.189694429815185
```

Part 2.2: Build a model using a subset of the data

In [10]:
```
from sklearn.tree import DecisionTreeRegressor # For training a Decision Tree Regressor
from sklearn.metrics import mean_squared_error  # For calculating MSE
from sklearn.model_selection import train_test_split
import pandas as pd

# Load the data
file_path = 'abalone.csv'
df = pd.read_csv(file_path)

df_subset = df[['Shucked weight', 'Shell weight', 'Rings']] # Obtained previously as the 2 most important features

# Split data into features and target variable
X = df_subset.drop('Rings', axis=1)
y = df_subset['Rings']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [11]:
```
# Training the Decision Tree Regressor
dt_regressor = DecisionTreeRegressor(max_depth = 3, random_state=0)
dt_regressor.fit(X_train, y_train)
```

Out[11]:
```
DecisionTreeRegressor(max_depth=3, random_state=0)
```

In [12]:
```
# Predict the target values for the same dataset
y_pred = dt_regressor.predict(X_test)
```

In [13]:
```
# Calculate the Mean Absolute Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 6.178891055534098
```

Notes:

As you can see, in both the cases, the final error rate was about the same. In fact, we see a small improvement in the perf

To generate a visualization

```
import graphviz
from sklearn import tree

# Generate visualization for the limited depth tree
dot_data_limited = tree.export_graphviz(dt_regressor, out_file=None,
                                        feature_names=X_train.columns,
                                        filled=True)
graphviz.Source(dot_data_limited, format="png")
```

In [ ]: