Directions for Supervised Machine Learning CART and kNN

Part 1: CART

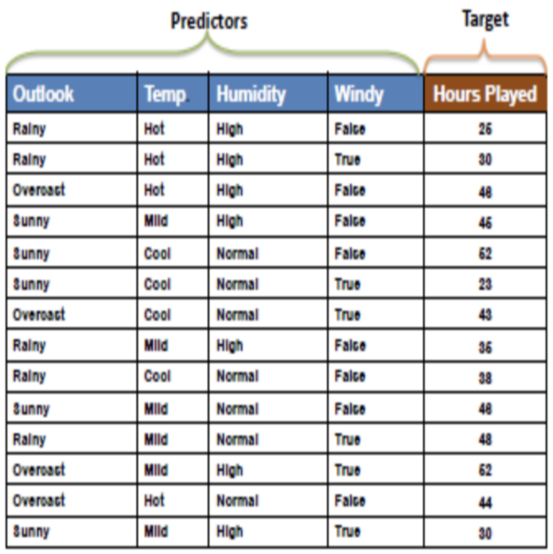
Supervised Machine Learning CART

CART Regression with Scikit-Learn: Overview

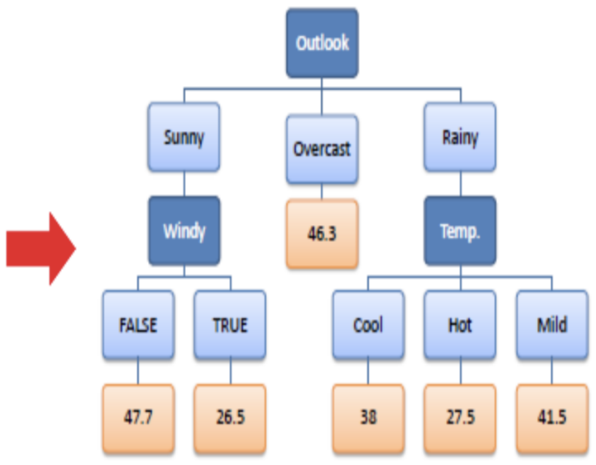
A decision tree builds regression or classification models in the form of a tree structure.

Regression trees are needed when the response variable is numeric or continuous.

In either case, the predictors or independent variables may be categorical or numeric. It is the target variable that determines the type of decision tree needed.



Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	26
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	62
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



Python: Scikit-Learn Library

The scikit-learn project started as scikit.learn, a Google Summer of Code project by David Cournapeau.

Its name stems from the notion that it is a "SciKit' (SciPy Toolkit), a separately developed and distributed third-party extension to SciPy.

The original codebase was later rewritten by other developers:

- In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel, all from INRIA, took leadership of the project.
- They made the first public release on February the 1st 2010.
- As of 2018, scikit-learn is under active development.

Scikit-leam provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-leam.

This stack includes:

- NumPy: Base n-dimensional array package
- SciPy: Fundamental library for scientific computing
- Matplotlib: Comprehensive 2D/3D plotting
- IPython: Enhanced interactive console
- Sympy: Symbolic mathematics
- Pandas: Data structures and analysis

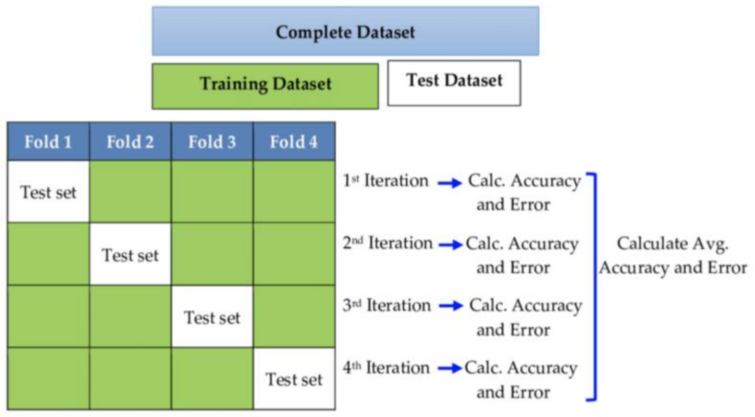
K-Fold Cross-Validation

In K Fold cross-validation, the data is divided into k subsets.

One of the k subsets is used as the test set/ validation set and the other k-l subsets are put together to form a training set.

The error estimation is averaged over all k trials to get the total effectiveness of our model.

In the following illustration K-Fold = 4


Complete Dataset
Training Dataset
Test Dataset
Fold 1
Fold 2
Fold 3
Fold 4
Test set
Test set
Test set
Test set
1st Iteration → Calc. Accuracy and Error
2nd Iteration → Calc. Accuracy and Error
3rd Iteration → Calc. Accuracy and Error
4th Iteration → Calc. Accuracy and Error
Calculate Avg. Accuracy and Error

Negative Mean Squared Error (Negative MSE)

In statistics, the mean squared error (MSE) or mean squared deviation (MSD) of an estimator (a procedure for estimating an unobserved quantity) measures the average of the squares of the errors or deviation, i.e., the difference between the estimator and what is estimated.

MSE is a risk function, corresponding to the expected value of the squared error loss or quadratic loss.

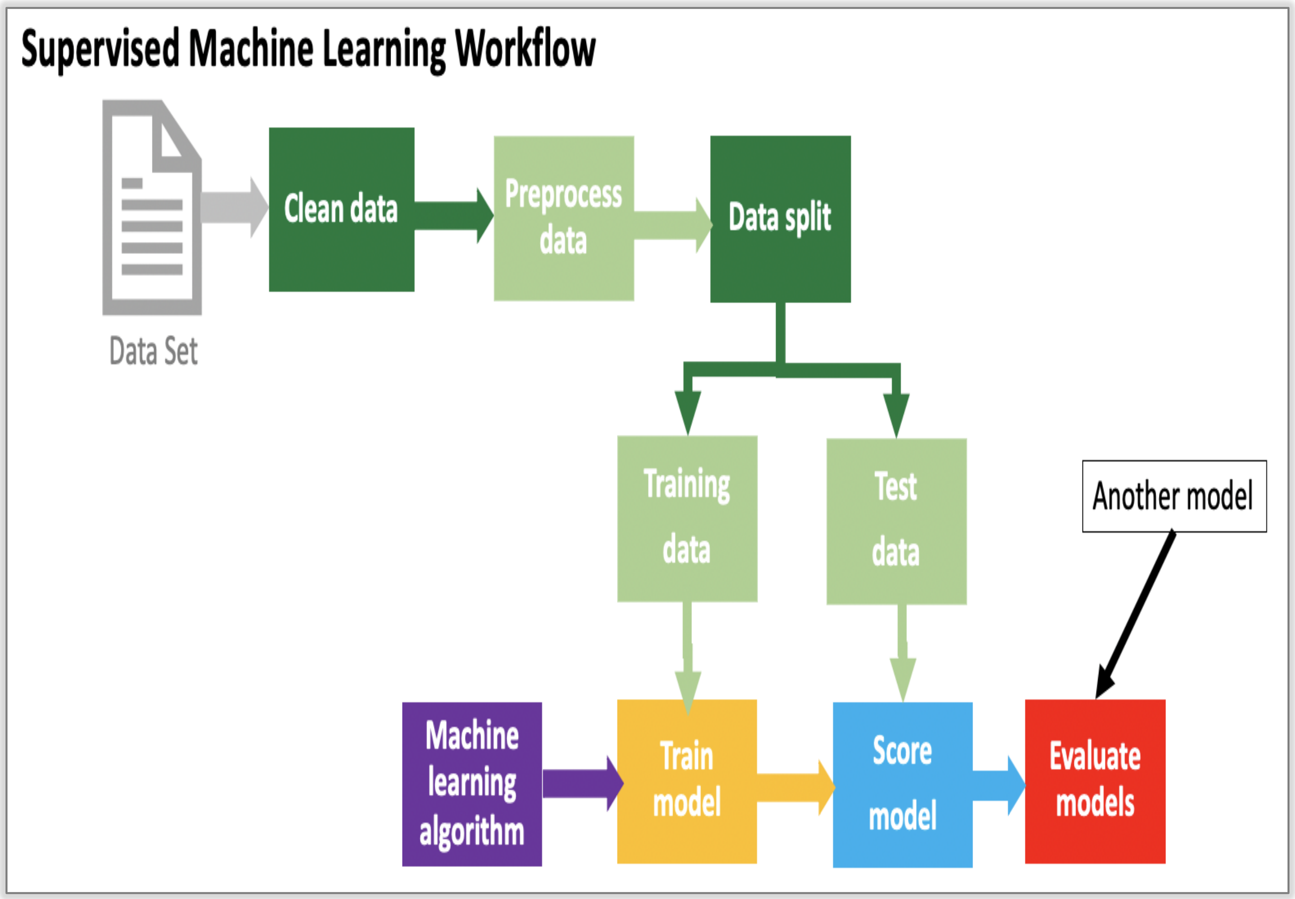
The difference occurs because of randomness or because the estimator doesn't account for information that could produce a more accurate estimate.

The MSE is a measure of the quality of an estimator.

In the general case, it is non-negative, and values closer to zero are better.

IMPORTANT NOTES: In scikit-learn, mean squared error values are inverted, i.e., negative

Let’s Get Started with Machine Learning Supervised CART



STEP 1: Import Libraries

- import pandas and numpy libraries
- import scatter_matrix from pandas.plotting
- import DecisionTreeRegressor from sklearn.tree
- import tree from sklearn
- import train_test_split, KFold, and cross_val_score from sklearn.model_selection
- import matplotlib
- import seaborn
- import pyplot from matplotlib

Python Code In [ ]:

```
# Import Python Libraries: NumPy and Pandas

import pandas as pd
import numpy as np

# Import Libraries & modules for data visualization

from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Import scit-Learn module for the algorithm/model: DecisionTreeRegressor and tree to plot

from sklearn.tree import DecisionTreeRegressor
from sklearn import tree

# Import scikit-Learn module to split the dataset into train/ test sub-datasets

from sklearn.model_selection import train_test_split

# Import scikit-Learn module for K-fold cross-validation - algorithm/modeL evaluation &
validation

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

WORKFLOW: DATA SET

STEP 2: Read data description and Load the Data

Data Set is the housing_boston.csv

We will investigate the Boston House Price dataset as you did with the linear regression homework. Each record in the database describes a Boston suburb or town. The data was drawn from the Boston Standard Metropolitan Statistical Area (SMSA) in 1970. The attributes are defined as follows:

- CRIM: per capita crime rate by town
- ZN: proportion of residential land zoned for lots over 25,000 sq. ft.
- INDUS: proportion of non-retail business acres per town
- CHAS: Charles River dummy variable (= 1 tract bounds river; 0 otherwise)
- NOX nitric oxides concentration (parts per 10 million)
- RM: average number of rooms per dwelling
- AGE: proportion of owner-occupied units built prior to 1940
- DIS weighted distances to five Boston employment centers
- RAD: index of accessibility to radial highways
- TAX: full-value property-tax rate per 10,000 dollars
- PTRATIO: pupil-teacher ratio by town
- AA: 1000(AA -0:63)^2 where AA is the proportion of African Americans by town
- LSTAT: % lower status of the population
- MEDV: Median value of owner-occupied homes in 1000 dollars.

Note: For this assignment, we use a subset of the original dataset.

- CRIM: per capita crime rate by town
- INDUS: proportion of non-retail business acres per town
- TAX: full-value property-tax rate per 10,000 dollars
- MEDV: Median value of owner-occupied homes in 1000 dollars.

Python Code In [ ]:
```
# Specify location of the dataset.
housingfile = 'housing_boston.csv'
```
Python Code In [ ]:
```
# Load the data into a Pandas DataFrame
df= pd.read_csv (housingfile, header=None)
```
Python Code In [ ]:
```
# Specify the fields with their names
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
'TAX', 'PTRATIO', 'AA','LSTAT', 'MEDV']
```
Python Code In [ ]:
```
# Load the data into a Pandas DataFrame
df = pd.read_csv(housingfile, names=names)
#  Look at the first 5 rows of data
df.head()
```

WORKFLOW: Clean and Preprocess the Dataset

STEP 3: Clean the data

Python Code In [ ]:
```
df.isnull().sum()

# We see there are no missing data points
```
Python Code In [ ]:
```
# Now let's say we want to decrease the number of variables in our heatmap.
# We would use the following code.
#  Remember how to make a subset.  Try using different variables.

df2= df[['CRIM','INDUS', 'TAX','MEDV']]

# We will use df2 for the rest of the calculations.
```
Python Code In [ ]:
```
df2.head()
```

STEP 4: Performing the Exploratory Data Analysis (EDA)

Python Code In [ ]:
```
# Get the number of records/rows, and the number of variables/columns

print(df2.shape)
```
Python Code In [ ]:
```
# Get the data types of all variables

print(df2.dtypes)
```
Python Code In [ ]:
```
# Obtain the summary statistics of the data

print(df2.describe())
```

STEP 4A: Create Histograms

Python Code In [ ]:
```
# Plot histogram for each variable. I encourage you to work with the
histogram. Remember what you did in the previous homework.

df2.hist(edgecolor= 'black',figsize=(14,12))
plt.show()
```

STEP 4B: Create Scatter Plot

Python Code In [ ]:
```
# Create scatter plot matrix

scatter_matrix(df2, alpha=0.8, figsize=(15, 15))
plt.show()
```

STEP 4C: Join Plots with Seaborn

IMPORTANT NOTE: You can find more information on joint plots here http://seaborn.pydata.org/generated/seaborn.jointplot.html

Python Code In [ ]:
```
sns.jointplot(data=df2, x="CRIM", y="MEDV", kind="reg")
```
Python Code In [ ]:
```
#Join plot with CRIM and MEDV

sns.jointplot(x = 'CRIM', y = 'MEDV', data = df2, kind = 'kde', height = 5,
joint_kws={'color':'green'})
plt.show()
```

Python Code In [ ]:
```
#Join plot with TAX and MEDV

sns.jointplot(x = 'TAX', y = 'MEDV', data = df2, kind = 'hex', height = 5,
joint_kws={'color':'purple'})
plt.show()
```

Python Code In [ ]:
```
# Join plot with TAX and MEDV

sns.jointplot(x = 'INDUS', y = 'MEDV', data = df2, kind = 'hist', height = 5,
joint_kws={'color':'orange'}, binwidth=(3,5), cbar=True)
plt.show()
```

Python Code In [ ]:
```
# Now we will combine the join plots

g = sns.PairGrid(df2, height= 10)
g.map_upper(sns.histplot, bins= 20, binwidth=3, cbar=True)
g.map_lower(sns.kdeplot, fill=True, cbar=True)
g.map_diag(sns.histplot, kde=True, cbar=True)
```

WORKFLOW: DATA SPLIT

STEP 5: Separate the Dataset into Input & Output NumPy Arrays

Python Code In [ ]:
```
# Store the dataframe values into a numPy array

array = df2.values

# Separate the array into input and output components by slicing (you used
this in your Python fundamental homework)
# For X (input) [:,3] --> All the rows and columns from 0 up to 3

X = array [:, 0:3]

# For Y (output) [:3] --> All the rows in the last column (MEDV)

Y = array [:,3]
```

STEP 6: Split into Input/Output Array into Training/Testing Datasets

Python Code In [ ]:
```
# Split the dataset --> training sub-dataset:  67%, and test sub-dataset:
33%

test_size = 0.33

# Selection of records to inclue in which sub-dataset must be done randomly -
use the for seed radomization

seed = 7

# Split the dataset (both input & output) into training/testing datasets

X_train, X_test, Y_train, Y_test= train_test_split(X,Y, test_size=test_size,
random_state=seed)
```

WORKFLOW: TRAIN MODEL

STEP 7: Build and Train the Model

Python Code In [ ]:
```
# Build the model
model = DecisionTreeRegressor(random_state=seed)
```

Python Code In [ ]:

```
# Train the model using the training sub-dataset
model.fit(X_train,Y_train)

# Non-Linear --> NO coefficients and the intercept
```

```
DecisionTreeRegressor (criterion='mse', max_depth=None, max_features=None,
max_leaf_nodes=None, min_impurity_decrease=0.0, min_samples_split=100,
min_weight_fraction_leaf=0.0, random_state=seed, splitter='best')
```

Python Code In [ ]:

```
#Plot tree

tree.plot_tree(model, feature_names=X_train, class_names=Y_train, filled =
True, fontsize=10)

plt.show()
```

WORKFLOW: SCORE MODEL 1

STEP 8: Calculate R-Squared

Python Code In [ ]:
```
R_squared = model.score(X_test, Y_test)
print('R-Squared = ', R_squared)
```

Notes: The higher the R-squared, the better (0 – 100%). Depending on the model, the best models score above 83%. The R-squared value tells us how well the independent variables predict the dependent variable. This is very low. Think about how you could increase the R-squared.

Step 9: Prediction

Python Code In [ ]:
```
model.predict([[12,10,450]])
```
Python Code In [ ]:
```
model.predict([[2,30,50]])
```

We have now trained the model. Let's use the trained model to predict the "Median value of owner-occupied homes in 1000 dollars" (MEDV).

We are using the following predictors for the 1st prediction:

- CRIM: per capita crime rate by town: 12
- INDUS: proportion of non-retail business acres per town: 10
- TAX: full-value property-tax rate per $10,000: 450

Notes: So, the model predicts that the median value of owner-occupied homes in 1000 dollars in the above suburb should be around $12,600.

We are using the following predictors for the 2nd prediction:

- CRIM: per capita crime rate by town: 2
- INDUS: proportion of non-retail business acres per town: 30
- TAX: full-value property-tax rate per $10,000: 50

Notes: So, the model predicts that the median value of owner-occupied homes in 1000 dollars in the above suburb should be around $15,700.

WORKFLOW: EVALUATE MODELS

Step 10: Train & Score Model 2 Using K-Fold Cross Validation Data Split

Python Code In [ ]:
```
# Evaluate the algorithm
# Specify the K-size
num_folds = 10

# Fix the random seed
# must use the same seed value so that the same subsets can be obtained
# for each time the process is repeated
seed = 7

# Split the whole data set into folds
kfold= KFold(n_splits=num_folds, random_state=seed, shuffle=True)

scoring = 'neg_mean_squared_error'
```

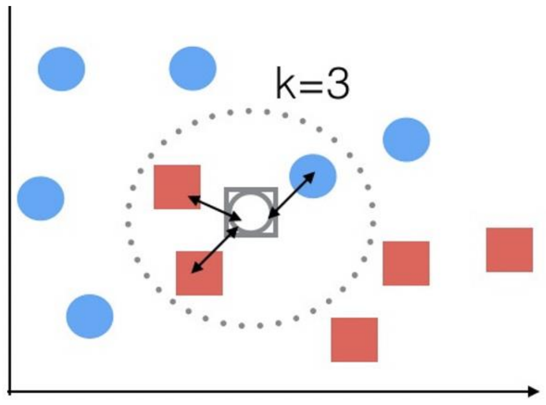
Python Code In [ ]:

```
# Train the model and run K-foLd cross-validation to validate/evaluate the
model
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)

# Print out the evaluation results
# Result: the average of all the results obtained from the k-fold cross
validation
print("Average of all results from the K-fold Cross Validation, using
negative mean squared error:",results.mean())
```

Notes: After we train, we evaluate. We are using K-fold to determine if the model is acceptable. We pass the whole set since the system will divide it for us. This value would traditionally be a positive value but scikit reports this value as a negative value. If you want a positive number, you may calculate the square root of the Negative Mean Squared Error value.

Part 2: k-Nearest Neighbors (kNN)



Supervised Machine Learning: k-Nearest Neighbors Overview

- The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.
- The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. You may have heard the old adage, “Birds of a feather flock together.”
- An object is classified by a majority vote of its neighbors with the new object being assigned to the class most common among its k nearest neighbors.
- K is a positive integer, typically small: k = 3, 5, or 10.
- KNN can do multiple (more than two) class predictions.
- In binary (two-class) classification problems, it is helpful to choose k to be an odd number as this can avoid a tie vote

Supervised Machine Learning: k-Nearest Neighbors

- Let's begin Part 2 using the same Supervised Learning Workflow used in part 1.

STEP 1: Import Libraries

- import pandas and numpy libraries
- import scatter_matrix from pandas.plotting
- import matplotlib
- import seaborn
- import pyplot from matplotlib
- import KNeighborsClassifier from sklearn.neighbors
- import train_test_split, KFold, and cross_val_score from sklearn.model_selection
- import classification_report from sklearn.metrics

Python Code In [ ]:

```
# Import Python Libraries: NumPy and Pandas

import pandas as pd
import numpy as np

# Import Libraries & modules for data visualization

from pandas.plotting import scatter_matrix
from matplotlib import pyplot


# Import scikit-Learn module for the algorithm/modeL: Nearest Neighbors

from sklearn.neighbors import KNeighborsClassifier
```

Python Code In [ ]:

```
# Import scikit-Learn module to split the dataset into train/ test sub-
datasets

from sklearn.model_selection import train_test_split


# Import scikit-Learn module for K-fold cross-validation - algorithm/modeL
evaluation & validation

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


# Import scikit-Learn module classification report to later use for
information about how the system try to classify / label each record

from sklearn.metrics import classification_report
```

WORKFLOW: DATA SET

STEP 2: Read data description and Load the Data

Description Iris Dataset

Data Set: Iris.csv Title: Iris Plants Database Updated Sept 21 by C. Blake -Added discrepancy information Sources:

- Creator: RA_ Fisher
- Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
- Date: 1988

Relevant Information: This is perhaps the best-known database to be found in the pattern recognition literature. Fishers paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example)

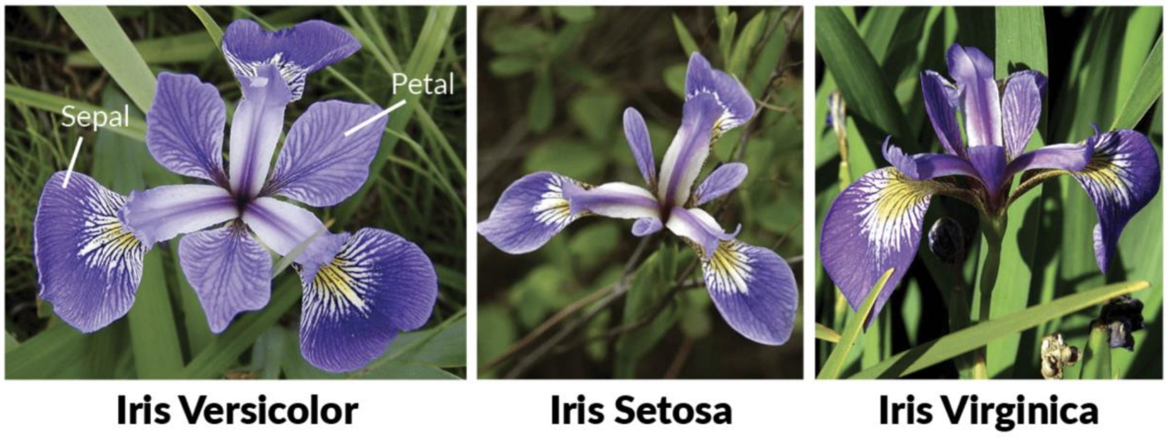
The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

Predicted attribute: class of iris plant

Number of Instances: 150 (50 in each of three classes)

Number of predictors: 4 numeric, predictive attributes and the class Attribute Information:

- 1.sepal length in cm
- 2.sepal width in cm
- 3.petal length in cm
- 4.petal width in cm
- 5.class:
-



Python Code In [ ]:

```
# Specify location of the dataset
filename = 'iris.csv'

# Load the data into a Pandas DataFrame
df = pd.read_csv(filename)
```

WORKFLOW: Clean and Preprocess the Dataset

STEP 3: Clean the data

Python Code In [ ]:

```
# mark zero values as missing or NaN
df[[ 'SepalLengthCm' , 'SepalWidthCm' , 'PetalLengthCm' ,'PetalWidthCm' ]] \
= df[['SepalLengthCm' , 'SepalWidthCm' ,'PetalLengthCm' , 'PetalWidthCm'
]].replace(0,np.NaN)

# count the number of NaN values in each column
print (df.isnull().sum())
```

STEP 4: Performing the Exploratory Data Analysis (EDA)

Python Code In [ ]:

```
# get the dimensions or shape of the dataset
# i.e. number of records / rows X number of variables / columns
print(df.shape)
```

Python Code In [ ]:

```
#get the data types of all the variables / attributes in the data set
print(df.dtypes)
```

Python Code In [ ]:

```
#return the first five records / rows of the data set
print(df.head(5))
```

Python Code In [ ]:

```
#return the summary statistics of the numeric variables / attributes in the
data set
print(df.describe())
```

Python Code In [ ]:

```
#class distribution i.e. how many records are in each class
print(df.groupby('Species').size())
```

STEP 4A: Create Histograms

Python Code In [ ]:

```
#plot histogram of each numeric variable / attribute in the data set
df.hist(figsize=(12, 8))
pyplot.show()
```

STEP 4B: Density Plots

Python Code In [ ]:

```
# generate density plots of each numeric variable / attribute in the data set
df.plot(kind='density', subplots=True, layout=(3, 3), sharex=False,
legend=True, fontsize=1,
figsize=(12, 16))
pyplot.show()
```

Step 4C: Creating Boxplots

Python Code In [ ]:

```
# generate box plots of each numeric variable / attribute in the data set
df.plot(kind='box', subplots=True, layout=(3,3), sharex=False,
figsize=(12,8))
pyplot.show()
```

Step 4D: Creating Scatter plots

Python Code In [ ]:

```
# generate scatter plot matrix of each numeric variable / attribute in the
data set
scatter_matrix(df, alpha=0.8, figsize=(15, 15))
pyplot.show()
```

WORKFLOW: DATA SPLIT

STEP 5: Separate the Dataset into Input & Output NumPy Arrays

Python Code In [ ]:

```
# store dataframe values into a numpy array
array = df.values
# separate array into input and output by slicing
# for X(input) [:, 1:5] --> all the rows, columns from 1 - 4 (5 - 1)
# these are the independent variables or predictors
X = array[:,1:5]
# for Y(input) [:, 5] --> all the rows, column 5
# this is the value we are trying to predict
Y = array[:,5]
```

STEP 6: Split into Input/Output Array into Training/Testing Datasets

Python Code In [ ]:

```
# split the dataset --> training sub-dataset: 67%; test sub-dataset: 33%
test_size = 0.33
#selection of records to include in each data sub-dataset must be done
randomly
seed = 7
```

Python Code In [ ]:

```
#split the dataset (input and output) into training / test datasets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=test_size,
random_state=seed)
```

WORKFLOW: TRAIN MODEL

STEP 7: Build and Train the Model

Python Code In [ ]:

```
#build the model
model = KNeighborsClassifier()

# train the model using the training sub-dataset
model.fit(X_train, Y_train)

#print the classification report
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print("Classification Report: ", "\n", "\n", report)
```

WORKFLOW: SCORE MODEL 1

STEP 8: Score the Accuracy of the Model

Accuracy is the most intuitive performance measure, and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost the same. Therefore, you have to look at other parameters to evaluate the performance of your model.

Python Code In [ ]:
```
#score the accuracy leve
result = model.score(X_test, Y_test)
#print out the results
print(("Accuracy: %.3f%%") % (result*100.0))
```

Step 9: Prediction

Python Code In [ ]:

```
model.predict([[5.3, 3.0, 4.5, 1.5]])
```

WORKFLOW: EVALUATE MODELS

Step 10: Train & Score Model 2 Using K-Fold Cross Validation Data Split

Python Code In [ ]:
```
# evaluate the algorithm
# specify the number of time of repeated splitting, in this case 10 folds
n_splits = 10
```

Python Code In [ ]:
```
# fix the random seed
# must use the same seed value so that the same subsets can be obtained
# for each time the process is repeated

seed = 7
```

Python Code In [ ]:
```
# split the whole dataset into folds
# In k-fold cross-validation, the original sample is randomly partitioned
into k equal sized subsamples. Of the k subsamples, a single subsample is
retained as the validation data for testing the model, and the remaining k −
1 subsamples are used as training data. The cross-validation process is then
repeated k times, with each of the k subsamples used exactly once as the
validation data. The k results can then be averaged to produce a single
estimation. The advantage of this method over repeated random sub-sampling is
that all observations are used for both training and validation, and each
observation is used for validation exactly once.

kfold = KFold(n_splits, random_state=seed, shuffle=True)
```

Python Code In [ ]:
```
# we can use the accuracy level to evaluate the model / algorithm

scoring = 'accuracy'
```

Python Code In [ ]:
```
# train the model and run K-fold cross validation to validate / evaluate the
model

results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
```

Python Code In [ ]:

```
# print the evaluation results
# result: the average of all the results obtained from the K-fold cross
validation

print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))
```

Compare this outcome to last week's Supervised Logistic Regression exercise and assess which model is superior.