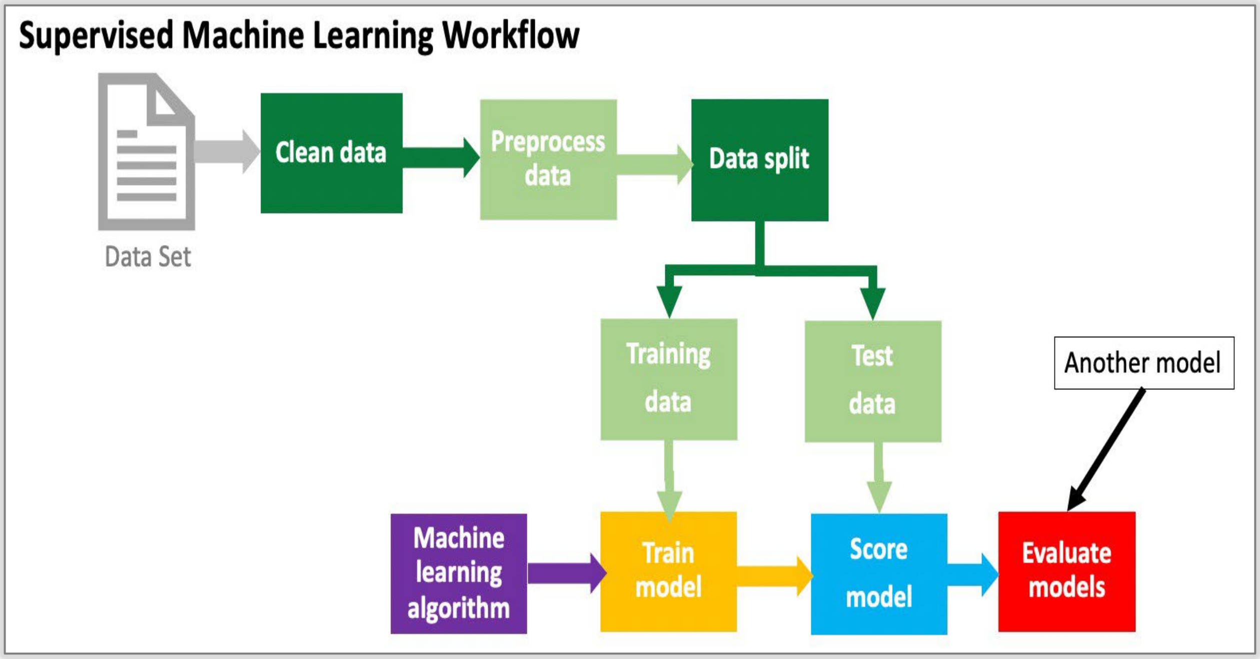
Directions for Machine Learning Supervised Learning

Linear and Logistic Regression

Part 1: Linear Regression

Machine Learning Supervised Linear Regression



- You can see in the workflow; supervised learning starts with the data set. Remember, since it is supervised, the data is labeled. Then there is some data preprocessing (cleaning) to be done. Next, you will declare your input (X/Independent variables) and output (Target Variable/Dependent or Y) NumPy Arrays. Then the data is split into a testing and training set. Then you will build and train the model, use the model for predictions, and evaluate/validate the model. So let's begin.

STEP 1: Import Libraries

- import pandas and numpy libraries
- import scatter_matrix from pandas.plotting
- import LinearRegression from sklearn.linear_model
- import train_test_split, KFold, and cross_val_score from sklearn.model_selection
- import matplotlib
- import seaborn

Python Code In [ ]:

```
import pandas as pd
import numpy as np

from pandas.plotting import scatter_matrix

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

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



import seaborn as sns
sns.set(color_codes=True)

import matplotlib.pyplot as plt
```

WORKFLOW: DATA SET

STEP 2: Read data description and Load the Data

- Read the description of the dataset listed below
- Dataset is provided in the module and assignment. It is called housing_boston.csv.
- Load the data into Pandas dataframe called df
- View the first five rows of the dataframe

Description of Boston Housing Dataset

- CRIM: This is the per capita crime rate by town
- ZN: This is the proportion of residential land zoned for lots larger than 25,000 sq. ft.
- INDUS: This is the proportion of non-retail business acres per town.
- CHAS: This is the Charles River dummy variable (this is equal to 1 if tract bounds river; 0 otherwise)
- NOX: This is the concentration of the nitric oxide (parts per 10 million)
- RM: This is the average number of rooms per dwelling
- AGE: This is the proportion of owner-occupied units built prior to 1940
- DIS: This is the weighted distances to five Boston employment centers
- RAD: This is the index of accessibility to radial highways
- TAX: This is the full-value property-tax rate per 10,000 dollars
- PTRATIO: This is the pupil-teacher ratio by town
- AA: This is calculated as $1000(AA — 0.63)^2$, where AA is the proportion of people of African American descent by town
- LSTAT: This is the percentage lower status of the population
- MEDV: This is the median value of owner-occupied homes in $1000s

Python Code In [ ]:

```
# Specify the 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 [ ]:

```
df.head()
```

STEP 3: Give names to the columns since there are no headers

- Give the following names to the columns:
  'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX','RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'AA', 'LSTAT', 'MEDV'
- Verify columns names were added
- View the first five rows of the dataframe

Python Code In [ ]:

```
#give names to the columns

col_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX','RM', 'AGE', 'DIS', 'RAD',
'TAX', 'PTRATIO', 'AA', 'LSTAT', 'MEDV']
```

Python Code In [ ]:

```
# Let's check to see if the column names were added

df.columns = col_names
```

Python Code In [ ]:

```
#  Look at the first 5 rows of data

df.head()
```

WORKFLOW: Clean and Preprocess the Dataset

STEP 4: Clean the data

- Find and Mark Missing Values
- If there are no missing data points, then proceed to Step 5.

Python Code In [ ]:

```
df.isnull().sum()

# We see there are no missing data points
```

STEP 5: Performing the Exploratory Data Analysis (EDA)

- Print a count of the number of rows (observations) and columns (variables)
- Print the data types of all variables
- Print a summary statistics of the data

Python Code In [ ]:

```
# Get the number of records/rows and the number of variables/columns

print(df.shape)
```

Python Code In [ ]:

```
# Get the data types of all variables

print(df.dtypes)
```

Python Code In [ ]:

```
# Obtain the summary statistics of the data

print(df.describe())
```

STEP 5A: Create Histograms

- Create histograms for each variable from the dataframe df with a figure size of 14 x 12
- Plot the histograms

Python Code In [ ]:

```
# Plot histogram for each variable. I encourage you to work with the
histogram.

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

STEP 5B: Create Density Plots

- Create density plots from the dataframe df that 14 numeric variables, at least 14 plots, layout (5,3): 5 rows, each row with 3 plots
- Plot the density plot

Python Code In [ ]:

```
# Density plots
# Notes:  14 numeric variables, at least 14 plots, layout (5,3): 5 rows, each
row with 3 plots

df.plot(kind='density', subplots=True, layout= (5,3), sharex=False,
legend=True, fontsize=1, figsize= (12,16))
plt.show()
```

STEP 5C: Create Boxplots

- Create boxplots from the dataframe df with a layout (5,3) and figure size (20,18). Ensure subplots is True and sharex is False.
- Plot the box plot

Python Code In [ ]:

```
# Boxplots

df.plot(kind="box", subplots=True, layout=(5,3), sharex=False,
figsize=(20,18))
plt.show()
```

STEP 5D: Pair Plots - Correlation Analysis and Feature Selection

- Create pair plots of the dataframe with a height of 1.5
- Plot the pair plots
- Use the format function to decrease the number of decimal places to three
- Obtain the correlation of the dataframe

Python Code In [ ]:

```
#Obtain pair plots of the data. I know this is a lot of information, but I
wanted you to see what is possible with the Seaborn library.

sns.pairplot(df, height=1.5);
plt.show()
```

Python Code In [ ]:

```
# We will decrease the number of decimal places with the format function.

pd.options.display.float_format = '{:,.3f}'.format
```

Python Code In [ ]:

```
# Here we will get the correlations, with only 3 decimals.

df.corr()
```

STEP 5E: Creating Heatmaps

- Create a heatmap of the dataframe with a figure size of 16x10, use the dataframe correlation and ensure annot is True
- Plot the heatmap
- For help type sns.heatmap?
- Create another dataframe df2 with less variables
- Obtain the correlation of the dataframe df2
- Plot pair plot for df2 with a height of 5.5
- Create a heatmap of the dataframe df2 with a figure size of 20x12, use the dataframe correlation and ensure annot is True
- Change the color to blue and increase the font size 20

Python Code In [ ]:

```
# We could simply look at the correlations, but a heatmap is a great way to
present to the general audience.

plt.figure(figsize =(16,10))
sns.heatmap(df.corr(), annot=True)
plt.show()
```

Python Code In [ ]:

```
# If you get stuck on what can be done with the heatmap, you can use the
following code to get help

sns.heatmap?
```

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']]
```

Python Code In [ ]:

```
# Here we will look at the correlations for only the variables in df2.

df2.corr()
```

Python Code In [ ]:

```
# Let's try the pairplot with only the variables in df2

sns.pairplot(df2, height=5.5);
plt.show()
```

Python Code In [ ]:

```
# Now, we will make a heatmap with only the variables in the df2 subset.

plt.figure(figsize =(20,12))
sns.heatmap(df2.corr(), annot=True)
plt.show()
```

Python Code In [ ]:

```
#If you want to change the color and font to make the labels easier to read,
use this code.

plt.figure(figsize =(20,12))
sns.heatmap(df2.corr(), cmap="Blues", annot=True, annot_kws={"fontsize":20})
plt.show()
```

WORKFLOW: DATA SPLIT

STEP 6: Separate the Dataset into Input & Output NumPy Arrays

- Store the dataframe d2 values into a NumPy array
- Separate the array into input and output components by slicing

Python Code In [ ]:

```
# Store the dataframe values into a NumPy array
array= df2.values

# Separate the array into input and output components by slicing
# 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 7: Split into Input/Output Array into Training/Testing Datasets

- Split the dataset into training at 67% and test at 33% with the seed = 7

Python Code In [ ]:

```
# Split the dataset --> training sub-dataset:  67%, and test sub-dataset:
33%

test_size = 0.33

# Selection of records to include in which sub-dataset must be done randomly
- use the following for seed randomization

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=0.2,
random_state=seed)
```

WORKFLOW: TRAIN MODEL

STEP 8: Build and Train the Model

- Assign LinearRegression to the model
- Train the model
- Print the intercept and coefficients
- Print the list of the coefficients with their correspondent variable name

Python Code In [ ]:

```
# Build the model

model=LinearRegression()

# Train the model using the training sub-dataset

model.fit(X_train, Y_train)

# Print intercept and coefficients

print ("Intercept:", model.intercept_)
print ("Coefficients:", model.coef_)
```

Python Code In [ ]:

```
# Print out the list of the coefficients with their correspondent variable
name
# Pair the feature names with the coefficients

names_2 = ["CRIM", "INDUS", "TAX"]

coeffs_zip = zip(names_2, model.coef_)

# Convert iterator into a set

coeffs = set(coeffs_zip)

# Print (coeffs)

for coef in coeffs:
    print (coef, "\n")
```

Python Code In [ ]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

WORKFLOW: SCORE MODEL 1

STEP 9: Calculate R-Squared

- Calculate the R-Squared
- Print the score

Note: 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, which is very low. Think about how you could increase the R-squared. What variables would you use?

Python Code In [ ]:

```
R_squared = model.score(X_test, Y_test)
print("R-squared: ", R_squared)
```

Step 10: Prediction

- Execute model prediction
- 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:

  - 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

Note: The model predicts that the median value of owner-occupied homes in 1000 dollars in the above suburb should be around $24,144.

Python Code In [ ]:

```
model.predict([[12,10,450]])
```

WORKFLOW: EVALUATE MODELS

Step 11: Train & Score Model 2 Using K-Fold Cross Validation Data Split

- Specify the k-size to 10
- Fix the random seed to 7
- Split the entire data set
- Obtain the Mean squared error
- Train the model and run K-fold cross-validation
- Print results

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)

# For Linear regression, we can use MSE (mean squared error) value
# to evaluate the model/algorithm

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())
```

Note: 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. We see a -64 avg of all errors (mean of square errors). This value would traditionally be positive, but scikit reports this value as a negative value. If the square root had been evaluated, the value would have been around 8.

Step 12: Score Using Explained Variance

- **Let's use a different scoring parameter. Here we use the Explained Variance. The best possible score is 1.0; lower values are worse.**
  - Specify the k-size to 10
  - Set the seed to 7
  - Split the entire data set
  - Obtain the explained variance score
  - Train the model and run K-fold cross-validation
  - Print results

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)

# For Linear regression, we can use explained variance value to evaluate the
model/algorithm

scoring = 'explained_variance'
```

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 exlpained variance:",results.mean())
```

To learn more about Scikit Learning scoring [https://scikit-learn.org/stable/modules/model_evaluation.html (Links to an external site.)]

Part 2: Logistic Regression

Machine Learning Supervised Logistic Regression

- Let's begin Part 2 using logistic regression 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 LogisticRegression from sklearn.linear_model
- 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
```

Python Code In [ ]:

```
# Import Libraries & modules for data visualization

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

Python Code In [ ]:

```
# Import scikit-Learn module for the algorithm/modeL: Logistic Regression
from sklearn.linear_model import LogisticRegression
```

Python Code In [ ]:

```
# Import scikit-Learn module to split the dataset into train/ test sub-
datasets
from sklearn.model_selection import train_test_split
```

Python Code In [ ]:

```
# 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
```

Python Code In [ ]:

```
# Import scikit-Learn module classification report to later use for
information about how the system

from sklearn.metrics import classification_report
```

WORKFLOW: DATA SET

STEP 2: Read data description and Load the Data

- Read the description of the dataset listed below
- Dataset is provided in the module and assignment. It is called iris.csv.
- Load the data into Pandas dataframe called df
- View the first five rows of the dataframe

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. Fisher's 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

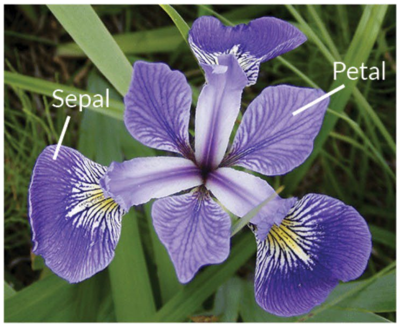
Number of Instances: 150 (50 in each of three classes)

Number of predictors: 4 numeric

Predictive attributes and the class attribute information:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm

class:



Iris Versicolor

Iris Setosa

Iris Virginica

Python Code In [ ]:

```
# Specify the location of the dataset

filename = 'iris.csv'

# Load the data into a Pandas DataFrame

df = pd.read_csv(filename)
```

Python Code In [ ]:

```
# Look at the data frame

df.head()
```

WORKFLOW: Clean and Preprocess the Dataset

STEP 3: Clean the data

- Find and Mark Missing Values
- If there are no missing data points, then proceed to Step 4.

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)

- Print a count of the number of rows (observations) and columns (variables)
- Print the data types of all variables
- Print a summary statistics of the data
- Print the number of records in each class

Python Code In [ ]:

```
# Get the number of records/rows and the number of variables/columns

print("Shape of the dataset(rows, columns):",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 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

- Create histograms from the dataframe df that is black with a figure size of 14 x 12
- Plot the histograms

Python Code In [ ]:

```
# Plot histogram for each variable.

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

Step 4B: Creating Boxplots

- Create boxplots from the dataframe df with a layout (5,3) and figure size (20,18). Ensure subplots is True and sharex is False.
- Plot the boxplots

Python Code In [ ]:

```
# Boxplots

df.plot(kind="box", subplots=True, layout=(5,3), sharex=False,
figsize=(20,18))
plt.show()
```

Step 4C: Create Pair Plots

- Create pair plots of the dataframe with a height of 3.5
- Plot the pair plots
- Add color

Note: Please click on the above URL to learn more about Pair Plots
https://seaborn.pydata.org/generated/seaborn.pairplot.html

Python Code In [ ]:

```
# Create pair plots

sns.pairplot(df, height=3.5);
plt.show()
```

Python Code In [ ]:

```
# Let's try that again using color. Notice: assigning a hue variable adds a
semantic mapping and changes the default marginal plot to a layered kernel
density estimate (KDE)

sns.pairplot(df, hue='Species', height=3, aspect= 1);
```

Step 4D: Creating Violin Plots

- Create violin plots
- Plot the violin plots
- Add color

Note: Please click on the above URL to learn more about Violin Plots
https://seaborn.pydata.org/generated/seaborn.violinplot.html

Python Code In [ ]:

```
#  Violin Plot

plt.figure(edgecolor="black", linewidth= 1.2,figsize=(12,8));
plt.subplot(2,2,1)
sns.violinplot(x='Species', y = 'SepalLengthCm', data=df)
plt.subplot(2,2,2)
sns.violinplot(x='Species', y = 'SepalWidthCm', data=df)
plt.subplot(2,2,3)
sns.violinplot(x='Species', y = 'PetalLengthCm', data=df)
plt.subplot(2,2,4)
sns.violinplot(x='Species', y = 'PetalWidthCm', data=df);
```

WORKFLOW: DATA SPLIT

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

- Store the dataframe values into a NumPy array
- Separate the array into input and output components by slicing

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 - 5
# 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

- Split the dataset into training at 67% and test at 33% with the seed = 7

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

- Assign LogisticRegression to the model
- Train the model
- Print the classification report

Python Code In [ ]:

```
#build the model

model = LogisticRegression(random_state=seed, max_iter=1000)

# train the model using the training sub-dataset

model.fit(X_train, Y_train)
```

Python Code In [ ]:

```
#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

- Calculate accuracy score
- Print the score

Python Code In [ ]:

```
#score the accuracy level

result = model.score(X_test, Y_test)

#print out the results

print(("Accuracy: %.3f%%") % (result*100.0))
```

Step 9: Prediction

- Execute model prediction

Note: We have now trained the model and using that trained model to predict the type of flower we have with the listed values for each variable.

Python Code In [ ]:

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

Python Code In [ ]:

```
model.predict([[5, 3.6, 1.4, 1.5]])
```

WORKFLOW: EVALUATE MODELS

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

- Specify the k-size to 10
- Fix the random seed to 7
- Split the entire data set
- Obtain the accuracy level
- Train the model and run K-fold cross-validation
- Print results

Python Code In [ ]:

```
# Evaluate the algorithm and specify the number of times of repeated
splitting, in this case, 10 folds

n_splits=10

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

seed=7

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

# for logistic regression, we can use the accuracy level to evaluate the
model

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)

# print the evaluation results. The result is the average of all the results
obtained from the K-fold cross-validation
print("Accuracy: %.3f (%.3f)"% (results.mean(), results.std()))
```