An Introduction to Scikit-Learn: Machine Learning in Python

This tutorial/notebook is made by :

Pranav Vinod Chaudhari (BTech-AIML)

Reference : https://www.simplilearn.com/tutorials/python-tutorial/scikit-learn (https://www.simplilearn.com/tutorials/python-tutorial/scikit-learn)

What is Python Scikit-Learn?

It's a simple and efficient tool for data mining and data analysis It is built on NumPy, SciPy, and Matplotlib It's an open-source, commercially available BSD license

What Can We Achieve Using Python Scikit-Learn?

For the most part, users accomplish three primary tasks with scikit-learn:

1. Classification Identifying which category an object belongs to.

Application: Spam detection

2. Regression Predicting a continuous variable based on relevant independent variables.

Application: Stock price predictions

3. Clustering Automatic grouping of similar objects into different clusters.

Application: Customer segmentation

What is Scikit Data Set?

For this tutorial, we will use the wine quality-red data set available on Kaggle, where you can also download the .csv file. Save the file in the same location where your Python file is saved.

Scikit-learn provides several in-built data sets for our convenience. You can visit https://scikit-learn.org/stable/datasets/index.html (https://scikit-learn.org/stable/datasets/index.html) to learn the names of those data sets. Let's see how to import the widely used iris plant data set.

In [6]:
```
from sklearn.datasets import load_iris
```

The data set contains details about the composition of wine, as well as it's quality.

Importing the Data Set and Modules

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.externals import joblib
from sklearn import preprocessing
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

wine = pd.read_csv('winequality-red.csv')
```

```
D:\Anaconda\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.cor
e.umath_tests is an internal NumPy module and should not be imported. It will be removed in a futur
e NumPy release.
  from numpy.core.umath_tests import inner1d
```

Let's discuss each of these modules one-by-one:

NumPy is used for algebraic and numerical calculations We have included pandas for working with data frames The model_selection module helps us to select between different models The preprocessing module gives us the ability to scale and transform our data The RandomForestRegressor is used to compare the performance metrics of our data set

Now that you have imported the data set from its source and converted that into a pandas DataFrame, let's display a few records from this DataFrame. For this, we will use the head() method.

In [3]:
```
import pandas as pd
wine = pd.read_csv('winequality-red.csv')
wine.head()
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

The head() method gives us the first five records from the data set.

Now, let's look at the total number of rows and columns in the data.

In [4]:
```
print(wine.shape)
```

```
(1599, 12)
```

Our data set consists of 1599 samples and 12 features, including our target feature.

All features include the following:

- quality(target)
- fixed acidity
- volatile acidity
- citric acid
- residual sugar
- chlorides
- free sulfur dioxide
- total sulfur dioxide
- density
- pH
- sulfates
- alcohol

Training Sets and Test Sets

Splitting the data into training and test sets are vital to estimating your model's performance.

A training set is used to test our algorithm to build a model.

A testing set is used to test our model to see how accurate our predictions are.

Let's separate our target (y) and our training (x) features, and split them into the train and test sets. We will use the scikit-learn train_test_split() function for splitting.

In [5]:
```
y = wine.quality
x = wine.drop('quality', axis=1)
x_train, x_test, y_train, y_test =  train_test_split(x, y, test_size=0.1, random_state=123, stratify=
```

Preprocessing Data

Data preprocessing is the process in which we make the data suitable to be performed over a model with less effort. It is the initial and most important process that enhances the quality of the model.

What is Standardization?

Standardization is a technique that is performed as a preprocessing step—before machine learning models are applied—to standardize the range of input representing data features.

We will be using Transformer API for preprocessing code, which makes the model performance more realistic.

In [11]:
```
#Transformer API
scaler = preprocessing.StandardScaler().fit(x_train)

#Applying transformer to training dataPython
x_train_scaled = scaler.transform(x_train)

print(x_train_scaled.mean(axis=0))

print(x_train_scaled.std(axis=0))
```

```
[-1.85165758e-17  5.80186042e-17  8.76451255e-17  2.34851903e-16
  1.08630578e-16 -8.14729336e-17  1.48132606e-17 -3.06566601e-14
  1.40664254e-15 -1.77759128e-16  1.25048609e-15]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

What is Hyperparameter?

- Hyperparameters define the higher-level concepts, such as complexity or capacity to learn
- It cannot be learned directly from the data in the standard model training process and needs to be predefined

Examples of hyperparameters include:

- Learning rate
- Number of clusters in clustering algorithms

You can see the list of tunable hyperparameters in the following way:

In [13]:
```
#List tunable Hyperparameters
pipeline = make_pipeline(preprocessing.StandardScaler(), RandomForestRegressor(n_estimators=10))
print(pipeline.get_params())
```

```
{'memory': None, 'steps': [('standardscaler', StandardScaler(copy=True, with_mean=True, with_std=Tr
ue)), ('randomforestregressor', RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=No
ne,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False))], 'standardscaler': St
andardScaler(copy=True, with_mean=True, with_std=True), 'randomforestregressor': RandomForestRegres
sor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False), 'standardscaler__cop
y': True, 'standardscaler__with_mean': True, 'standardscaler__with_std': True, 'randomforestregress
or__bootstrap': True, 'randomforestregressor__criterion': 'mse', 'randomforestregressor__max_dept
h': None, 'randomforestregressor__max_features': 'auto', 'randomforestregressor__max_leaf_nodes': N
one, 'randomforestregressor__min_impurity_decrease': 0.0, 'randomforestregressor__min_impurity_spli
t': None, 'randomforestregressor__min_samples_leaf': 1, 'randomforestregressor__min_samples_split':
2, 'randomforestregressor__min_weight_fraction_leaf': 0.0, 'randomforestregressor__n_estimators': 1
0, 'randomforestregressor__n_jobs': 1, 'randomforestregressor__oob_score': False, 'randomforestregr
essor__random_state': None, 'randomforestregressor__verbose': 0, 'randomforestregressor__warm_star
t': False}
```

The make_pipeline() function is used to combine a preprocessor with a classifier.

Let's declare the hyperparameters.

In [14]:
```
#Declare hyperparameters to tune
hyperparameters = {'randomforestregressor__max_features' : ['auto' , 'sqrt'],
                   'randomforestregressor__max_depth' : [None, 1, 2, 4]}
```

What is Cross-Validation?

Cross-validation is an important evaluation technique used to assess the generalization performance of a machine learning model. To avoid overfitting, the data set is usually divided into N random parts with equal volume.

In [15]:
```
#Cross-validation
clf = GridSearchCV(pipeline, hyperparameters, cv=3)
#Fit and tune model
clf.fit(x_train, y_train)
```

Out[15]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=Pipeline(memory=None,
     steps=[('standardscaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('randomf
orestregressor', RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decr...imators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False))]),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'randomforestregressor__max_features': ['auto', 'sqrt'], 'randomforestregressor_
_max_depth': [None, 1, 2, 4]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
GridSearchCV performs the cross-validation across the entire grid.
```

Evaluate Model Pipeline

Now it's time to evaluate the model performance. For this, we import the metrics we used earlier

In [16]:
```
y_pred = clf.predict(x_test)
print (r2_score(y_test, y_pred))
print (mean_squared_error(y_test, y_pred))
```

```
0.5028356516028893
0.3253125
```

The r2_score function is used to calculate the variance of the dependent variable for the independent variable.

Mean_squared_error calculates the average of the square of the errors.

To assess if the performance is sufficient, we return to the goal of the model that it was designed for.

Do not forget to save the model for future use.

In [17]:
```
joblib.dump(clf, 'rf_regressor.pkl')
```

```
D:\Anaconda\lib\site-packages\sklearn\externals\joblib\numpy_pickle.py:93: DeprecationWarning: to
string() is deprecated. Use tobytes() instead.
  pickler.file_handle.write(chunk.tostring('C'))
D:\Anaconda\lib\site-packages\sklearn\externals\joblib\numpy_pickle.py:93: DeprecationWarning: to
string() is deprecated. Use tobytes() instead.
  pickler.file_handle.write(chunk.tostring('C'))
D:\Anaconda\lib\site-packages\sklearn\externals\joblib\numpy_pickle.py:93: DeprecationWarning: to
string() is deprecated. Use tobytes() instead.
  pickler.file_handle.write(chunk.tostring('C'))
D:\Anaconda\lib\site-packages\sklearn\externals\joblib\numpy_pickle.py:93: DeprecationWarning: to
string() is deprecated. Use tobytes() instead.
  pickler.file_handle.write(chunk.tostring('C'))
D:\Anaconda\lib\site-packages\sklearn\externals\joblib\numpy_pickle.py:93: DeprecationWarning: to
string() is deprecated. Use tobytes() instead.
  pickler.file_handle.write(chunk.tostring('C'))
D:\Anaconda\lib\site-packages\sklearn\externals\joblib\numpy_pickle.py:93: DeprecationWarning: to
string() is deprecated. Use tobytes() instead.
  pickler.file_handle.write(chunk.tostring('C'))
```

Out[17]:
```
['rf_regressor.pkl']
```

Conclusion

In this Python scikit-learn article, we discussed the basic concepts of scikit-learn. We looked at how to import a data set and its different functions. We went through hyperparameters, preprocessing, and cross-validation techniques.