Validating Machine Learning Models with scikit-learn

Introduction

Building machine learning models is an important element of predictive modeling. However, without proper model validation, the confidence that the trained model will generalize well on the unseen data can never be high. Model validation helps in ensuring that the model performs well on new data, and helps in selecting the best model, the parameters, and the accuracy metrics.

In this guide, we will learn the basics and implementation of several model validation techniques, mentioned below:

```
Hold Out Validation
K-fold Cross-Validation.
Stratified K-fold Cross-Validation
Leave One Out Cross-Validation.
Repeated Random Test-Train Splits
```

Steps

In this guide, we will follow the following steps:

Step 1 - Loading the required libraries and modules.

Step 2 - Reading the data and performing basic data checks.

Step 3 - Creating arrays for the features and the response variable.

Step 4 - Trying out different model validation techniques.

Step 1 - Loading the Required Libraries and Modules

In [1]:

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn

import warnings
warnings.simplefilter('ignore')

# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression


from sklearn.model_selection import KFold
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import LeavePOut
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import StratifiedKFold
```

Step 2 - Reading the Data and Performing Basic Data Checks

In [2]:

```
dat = pd.read_csv('diabetes.csv')
print(dat.shape)
dat.describe().transpose()
```

(2000, 9)

Out[2]:

	count	mean	std	min	25%	50%	75%	ma
Pregnancies	2000.0	3.70350	3.306063	0.000	1.000	3.000	6.000	17.(
Glucose	2000.0	121.18250	32.068636	0.000	99.000	117.000	141.000	199.(
BloodPressure	2000.0	69.14550	19.188315	0.000	63.500	72.000	80.000	122.(
SkinThickness	2000.0	20.93500	16.103243	0.000	0.000	23.000	32.000	110.(
Insulin	2000.0	80.25400	111.180534	0.000	0.000	40.000	130.000	744.(
BMI	2000.0	32.19300	8.149901	0.000	27.375	32.300	36.800	80.6
DiabetesPedigreeFunction	2000.0	0.47093	0.323553	0.078	0.244	0.376	0.624	2.4
Age	2000.0	33.09050	11.786423	21.000	24.000	29.000	40.000	81.(
Outcome	2000.0	0.34200	0.474498	0.000	0.000	0.000	1.000	1.(

In [3]:

```
dat.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	2	138	62	35	0	33.6	0.12
1	0	84	82	31	125	38.2	0.23
2	0	145	0	0	0	44.2	0.63
3	0	135	68	42	250	42.3	0.36
4	1	139	62	41	480	40.7	0.53

Step 3 - Creating Arrays for the Features and the Response Variable

In [4]:

```
x1 = dat.drop('Outcome', axis=1).values
y1 = dat['Outcome'].values
```

Step 4 - Trying out Different Model Validation Techniques

1. Holdout Validation Approach

- The holdout validation approach refers to creating the training and the holdout sets, also referred to as the 'test' or the 'validation' set. The training data is used to train the model while the unseen data is used to validate the model performance. The common split ratio is 70:30, while for small datasets, the ratio can be 90:10.

In [5]:

```
# Evaluate using a train and a test set
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(x1, y1, test_size=0.30,
model = LogisticRegression()
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
print("Accuracy: %.2f%%" % (result*100.0))
```

```
Accuracy: 78.17%
```

We can see that the accuracy for the model on the test data is approximately 74 percent. The above technique is useful but it has pitfalls. The split is very important and, if it goes wrong, it can lead to model overfitting or underfitting the new data. This problem can be rectified using resampling methods, which repeat a calculation multiple times using randomly selected subsets of the complete data. We discuss the popular cross-validation techniques in the following sections of the guide.

2. K-fold Cross-Validation

- In k-fold cross-validation, the data is divided into k folds. The model is trained on k-1 folds with one fold held back for testing. This process gets repeated to ensure each fold of the dataset gets the chance to be the held back set. Once the process is completed, we can summarize the evaluation metric using the mean or/and the standard deviation.

In [6]:

```
kfold = model_selection.KFold(n_splits=10, random_state=100)
model_kfold = LogisticRegression()
results_kfold = model_selection.cross_val_score(model_kfold, x1, y1, cv=kfold)
print("Accuracy: %.2f%%" % (results_kfold.mean()*100.0))
```

```
Accuracy: 77.55%
```

i am use 10-fold cross-validation for our problem statement. The first line of code uses the 'model_selection.KFold' function from 'scikit-learn' and creates 10 folds. The second line instantiates the LogisticRegression() model, while the third line fits the model and generates cross-validation scores. The arguments 'x1' and 'y1' represents the predictor and the response array, respectively. The 'cv' argument specifies the number of cross-validation splits. The fourth line prints the mean accuracy result.

3. Stratified K-fold Cross-Validation

- Stratified K-Fold approach is a variation of k-fold cross-validation that returns stratified folds, i.e., each set containing approximately the same ratio of target labels as the complete data.

In [7]:

```
skfold = StratifiedKFold(n_splits=3, random_state=100)
model_skfold = LogisticRegression()
results_skfold = model_selection.cross_val_score(model_skfold, x1, y1, cv=skfold)
print("Accuracy: %.2f%%" % (results_skfold.mean()*100.0))
```

```
Accuracy: 77.80%
```

The lines of code below repeat the steps as discussed above for k-fold cross-validation, except for a couple of changes. The first line creates the Stratified KFolds instead of the k-fold, and this adjustment is then passed to the 'cv' argument in the third line of code.

4. Leave One Out Cross-Validation (LOOCV)

- LOOCV is the cross-validation technique in which the size of the fold is “1” with “k” being set to the number of observations in the data. This variation is useful when the training data is of limited size and the number of parameters to be tested is not high.

In [8]:

```
loocv = model_selection.LeaveOneOut()
model_loocv = LogisticRegression()
results_loocv = model_selection.cross_val_score(model_loocv, x1, y1, cv=loocv)
print("Accuracy: %.2f%%" % (results_loocv.mean()*100.0))
```

```
Accuracy: 77.80%
```

The lines of code below repeat the steps as discussed above, except for a couple of changes. The first line creates the leave-one-out cross-validation instead of the k-fold, and this adjustment is then passed to the 'cv' argument in the third line of code.

5. Repeated Random Test-Train Splits

- This technique is a hybrid of traditional train-test splitting and the k-fold cross-validation method. In this technique, we create random splits of the data in the training-test set manner and then repeat the process of splitting and evaluating the algorithm multiple times, just like the cross-validation method.

In [9]:

```
kfold2 = model_selection.ShuffleSplit(n_splits=10, test_size=0.30, random_state=100)
model_shufflecv = LogisticRegression()
results_4 = model_selection.cross_val_score(model_shufflecv, x1, y1, cv=kfold2)
print("Accuracy: %.2f%% (%.2f%%)" % (results_4.mean()*100.0, results_4.std()*100.0))
```

```
Accuracy: 76.43% (1.18%)
```

Conclusion

- In this guide, you have learned about the various model validation techniques using scikit-learn. The guide used the diabetes dataset and built a classifier algorithm to predict the detection of diabetes.
- The mean accuracy result for the various techniques is summarised below:

```
Holdout Validation Approach: Accuracy of 78.17%
K-fold Cross-Validation: Mean Accuracy of 77.55%
Stratified K-fold Cross-Validation: Mean Accuracy of  77.80%
Leave One Out Cross-Validation: Mean Accuracy of 77.80%
Repeated Random Test-Train Splits: Mean Accuracy of 76.43%
```

- We can conclude that the cross-validation technique improves the performance of the model and is a better model validation strategy. The model can be further improved by doing exploratory data analysis, data pre-processing, feature engineering, or trying out other machine learning algorithms instead of the logistic regression algorithm we built in this guide.