Machine Learning Model Evaluation

Machine Learning Model does not require hard-coded algorithms. We feed a large amount of data to the model and the model tries to figure out the features on its own to make future predictions. So we must also use some techniques to determine the predictive power of the model.

Machine Learning Model Evaluation

Model evaluation is the process that uses some metrics which help us to analyze the performance of the model. As we all know that model development is a multi-step process and a check should be kept on how well the model generalizes future predictions. Therefore evaluating a model plays a vital role so that we can judge the performance of our model. The evaluation also helps to analyze a model's key weaknesses. There are many metrics like Accuracy, Precision, Recall, F1 score, Area under Curve, Confusion Matrix, and Mean Square Error. Cross Validation is one technique that is followed during the training phase and it is a model evaluation technique as well.

Cross Validation and Holdout

Cross Validation is a method in which we do not use the whole dataset for training. In this technique, some part of the dataset is reserved for testing the model. There are many types of Cross-Validation out of which K Fold Cross Validation is mostly used. In K Fold Cross Validation the original dataset is divided into k subsets. The subsets are known as folds. This is repeated k times where 1 fold is used for testing purposes. Rest k-1 folds are used for training the model. So each data point acts as a test subject for the model as well as acts as the training subject. It is seen that this technique generalizes the model well and reduces the error rate

Holdout is the simplest approach. It is used in neural networks as well as in many classifiers. In this technique, the dataset is divided into train and test datasets. The dataset is usually divided into ratios like 70:30 or 80:20. Normally a large percentage of data is used for training the model and a small portion of the dataset is used for testing the model.

Evaluation Metrics for Classification Task

In this Python code, we have imported the iris dataset which has features like the length and width of sepals and petals. The target values are Iris setosa, Iris virginica, and Iris versicolor. After importing the dataset we divided the dataset into train and test datasets in the ratio 80:20. Then we called Decision Trees and trained our model. After that, we performed the prediction and calculated the accuracy score, precision, recall, and f1 score. We also plotted the confusion matrix.

Importing Libraries and Dataset

Python libraries make it very easy for us to handle the data and perform typical and complex tasks with a single line of code.

- **Pandas** – This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.
- **Numpy** – Numpy arrays are very fast and can perform large computations in a very short time.
- **Matplotlib/Seaborn** – This library is used to draw visualizations.
- Sklearn – This module contains multiple libraries having pre-implemented functions to perform tasks from data preprocessing to model development and evaluation.

- Python3

```
import pandas as pd

import numpy as np

from sklearn import tree

from sklearn import datasets

from sklearn.datasets import load_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import precision_score,\

recall_score, f1_score, accuracy_score
```

Now let's load the toy dataset iris flowers from the sklearn.datasets library and then split it into training and testing parts (for model evaluation) in the 80:20 ratio.

- Python3

```
iris = load_iris()

X = iris.data

y = iris.target



# Holdout method.Dividing the data into train and test

X_train, X_test,\

    y_train, y_test = train_test_split(X, y,

                                       random_state=20,

                                       test_size=0.20)
```

Now, let's train a Decision Tree Classifier model on the training data, and then we will move on to the evaluation part of the model using different metrics.

- Python3

```
tree = DecisionTreeClassifier()

tree.fit(X_train, y_train)

y_pred = tree.predict(X_test)
```

Accuracy

Accuracy is defined as the ratio of the number of correct predictions to the total number of predictions. This is the most fundamental metric used to evaluate the model. The formula is given by

```
Accuracy = (TP+TN)/(TP+TN+FP+FN)
```

However, Accuracy has a drawback. It cannot perform well on an imbalanced dataset. Suppose a model classifies that the majority of the data belongs to the major class label. It yields higher accuracy. But in general, the model cannot classify on minor class labels and has poor performance.

- Python3

```
print("Accuracy:", accuracy_score(y_test,
                                   y_pred))
```

Output:

```
Accuracy: 0.9333333333333333
```

Precision and Recall

Precision is the ratio of true positives to the summation of true positives and false positives. It basically analyses the positive predictions.

```
Precision = TP/(TP+FP)
```

The drawback of Precision is that it does not consider the True Negatives and False Negatives.

Recall is the ratio of true positives to the summation of true positives and false negatives. It basically analyses the number of correct positive samples.

```
Recall = TP/(TP+FN)
```

The drawback of Recall is that often it leads to a higher false positive rate.

- Python3

```
print("Precision:", precision_score(y_test,
                                    y_pred,
                                    average="weighted"))
```

```
print('Recall:', recall_score(y_test,

                               y_pred,

                               average="weighted"))
```

Output:

```
Precision: 0.9435897435897436

Recall: 0.9333333333333333
```

F1 score

The F1 score is the harmonic mean of precision and recall. It is seen that during the precision-recall trade-off if we increase the precision, recall decreases and vice versa. The goal of the F1 score is to combine precision and recall.

```
F1 score = (2×Precision×Recall)/(Precision+Recall)
```

- Python3

```
# calculating f1 score

print('F1 score:', f1_score(y_test, y_pred,

                            average="weighted"))
```

Output:

```
F1 score: 0.9327777777777778
```

Confusion Matrix

A confusion matrix is an N x N matrix where N is the number of target classes. It represents the number of actual outputs and the predicted outputs. Some terminologies in the matrix are as follows:

- True Positives: It is also known as TP. It is the output in which the actual and the predicted values are YES.
- True Negatives: It is also known as TN. It is the output in which the actual and the predicted values are NO.
- False Positives: It is also known as FP. It is the output in which the actual value is NO but the predicted value is YES.

- False Negatives: It is also known as FN. It is the output in which the actual value is YES but the predicted value is NO.

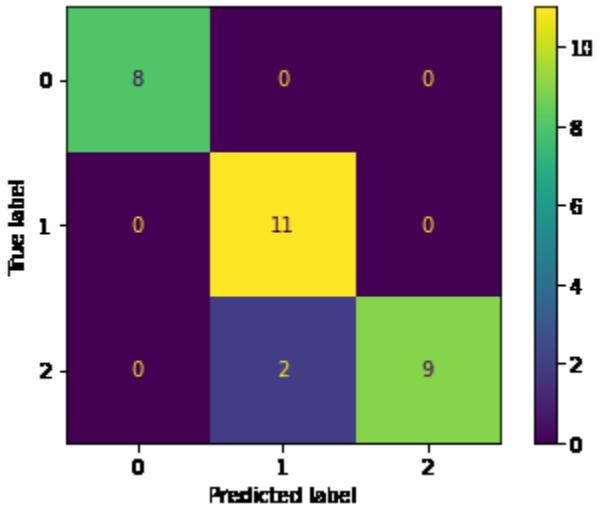
- Python3

```
confusion_matrix = metrics.confusion_matrix(y_test,
                                            y_pred)


cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix,
    display_labels=[0, 1, 2])


cm_display.plot()
plt.show()
```

Output:



Confusion matrix for the output of the model

In the output, the accuracy of the model is 93.33%. Precision is approximately 0.944 and Recall is 0.933. F1 score is approximately 0.933. Finally, the confusion matrix is plotted. Here class labels denote the target classes:

```
0 = Setosa
1 = Versicolor
2 = Virginica
```

From the confusion matrix, we see that 8 setosa classes were correctly predicted. 11 Versicolor test cases were also correctly predicted by the model and 2 virginica test cases were misclassified. In contrast, the rest 9 were correctly predicted.

AUC-ROC Curve

AUC (Area Under Curve) is an evaluation metric that is used to analyze the classification model at different threshold values. The Receiver Operating Characteristic(ROC) curve is a probabilistic curve used to highlight the model's performance. The curve has two parameters:

- TPR: It stands for True positive rate. It basically follows the formula of Recall.
- FPR: It stands for False Positive rate. It is defined as the ratio of False positives to the summation of false positives and True negatives.

This curve is useful as it helps us to determine the model's capacity to distinguish between different classes. Let us illustrate this with the help of a simple Python example

- Python3

```
import numpy as np

from sklearn .metrics import roc_auc_score



y_true = [1, 0, 0, 1]

y_pred = [1, 0, 0.9, 0.2]

auc = np.round(roc_auc_score(y_true,
```

```
                                y_pred), 3)

print("Auc", (auc))
```

Output:

```
Auc 0.75
```

AUC score is a useful metric to evaluate the model. It basically highlights a model's capacity to separate the classes. In the above code, 0.75 is a good AUC score. A model is considered good if the AUC score is greater than 0.5 and approaches 1. A poor model has an AUC score of 0.

Evaluation Metrics for Regression Task

Regression is used to determine continuous values. It is mostly used to find a relation between a dependent and an independent variable. For classification, we use a confusion matrix, accuracy, f1 score, etc. But for regression analysis, since we are predicting a numerical value it may differ from the actual output. So we consider the error calculation as it helps to summarize how close the prediction is to the actual value. There are many metrics available for evaluating the regression model.

In this Python Code, we have implemented a simple regression model using the Mumbai weather CSV file. This file comprises Day, Hour, Temperature, Relative Humidity, Wind Speed, and Wind Direction. The link for the dataset is here.

We are basically interested in finding a relationship between Temperature and Relative Humidity. Here Relative Humidity is the dependent variable and Temperature is the independent variable. We performed the Linear Regression and used the metrics to evaluate the performance of our model. To calculate the metrics we make extensive use of sklearn library.

- Python3

```
# importing the libraries

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_absolute_error,\
```

```
    mean_squared_error, mean_absolute_percentage_error
```

Now let's load the data into the panda's data frame and then split it into training and testing parts (for model evaluation) in the 80:20 ratio.

- Python3

```
df = pd.read_csv('weather.csv')

X = df.iloc[:, 2].values

Y = df.iloc[:, 3].values

X_train, X_test,\

    Y_train, Y_test = train_test_split(X, Y,

                                       test_size=0.20,

                                       random_state=0)
```

Now, let's train a simple linear regression model. On the training data and we will move to the evaluation part of the model using different metrics.

- Python3

```
X_train = X_train.reshape(-1, 1)

X_test = X_test.reshape(-1, 1)

regression = LinearRegression()

regression.fit(X_train, Y_train)

Y_pred = regression.predict(X_test)
```

Mean Absolute Error(MAE)

This is the simplest metric used to analyze the loss over the whole dataset. As we all know the error is basically the difference between the predicted and actual values. Therefore MAE is defined as the average of the errors calculated. Here we calculate the modulus of the error, perform the summation and then divide the result by the number of data points. It is a positive quantity and is not concerned about the direction. The formula of MAE is given by

$$MAE = \sum |y_{pred} - y_{actual}| \ / \ N$$

- Python3

```
mae = mean_absolute_error(y_true=Y_test,
                          y_pred=Y_pred)
print("Mean Absolute Error", mae)
```

Output:

```
Mean Absolute Error 1.7236295632503873
```

Mean Squared Error(MSE)

The most commonly used metric is Mean Square error or MSE. It is a function used to calculate the loss. We find the difference between the predicted values and the truth variable, square the result and then find the average over the whole dataset. MSE is always positive as we square the values. The small the MSE better is the performance of our model. The formula of MSE is given:

$$MSE = \sum (y_{pred} - y_{actual})^2 \ / \ N$$

- Python3

```
mse = mean_squared_error(y_true=Y_test,
                         y_pred=Y_pred)
print("Mean Square Error", mse)
```

Output:

```
Mean Square Error 3.9808057060106954
```

Root Mean Squared Error(RMSE)

RMSE is a popular method and is the extended version of MSE(Mean Squared Error). This method is basically used to evaluate the performance of our model. It indicates how much the data points are spread around the best line. It is the standard deviation of the Mean squared error. A lower value means that the data point lies closer to the best fit line.

$$RMSE=\sqrt{(\sum(y_{pred} - y_{actual})^2 / N)}$$

- Python3

```
rmse = mean_squared_error(y_true=Y_test,
                          y_pred=Y_pred,
                          squared=False)

print("Root Mean Square Error", rmse)
```

Output:

```
Root Mean Square Error 1.9951956560725306
```

Mean Absolute Percentage Error (MAPE)

MAPE is basically used to express the error in terms of percentage. It is defined as the difference between the actual and predicted value. The error is then divided by the actual value. The results are then summed up and finally, we calculate the average. Smaller the percentage better the performance of the model. The formula is given by

$$MAPE = \sum((y_{pred}-y_{actual}) / y_{actual}) / N * 100 \%$$

- Python3

```
mape = mean_absolute_percentage_error(Y_test,
                                      Y_pred,
                                      sample_weight=None,
                                      multioutput='uniform_average')
```

```
print("Mean Absolute Percentage Error", mape)
```

Output:

```
Mean Absolute Percentage Error 0.02334408993333347
```