Confusion Matrix and Classification Report

Understanding two powerful techniques to evaluate a classification model.

Giulio Laurenti, PhD Follow
Nov 26, 2020 · 10 min read ★

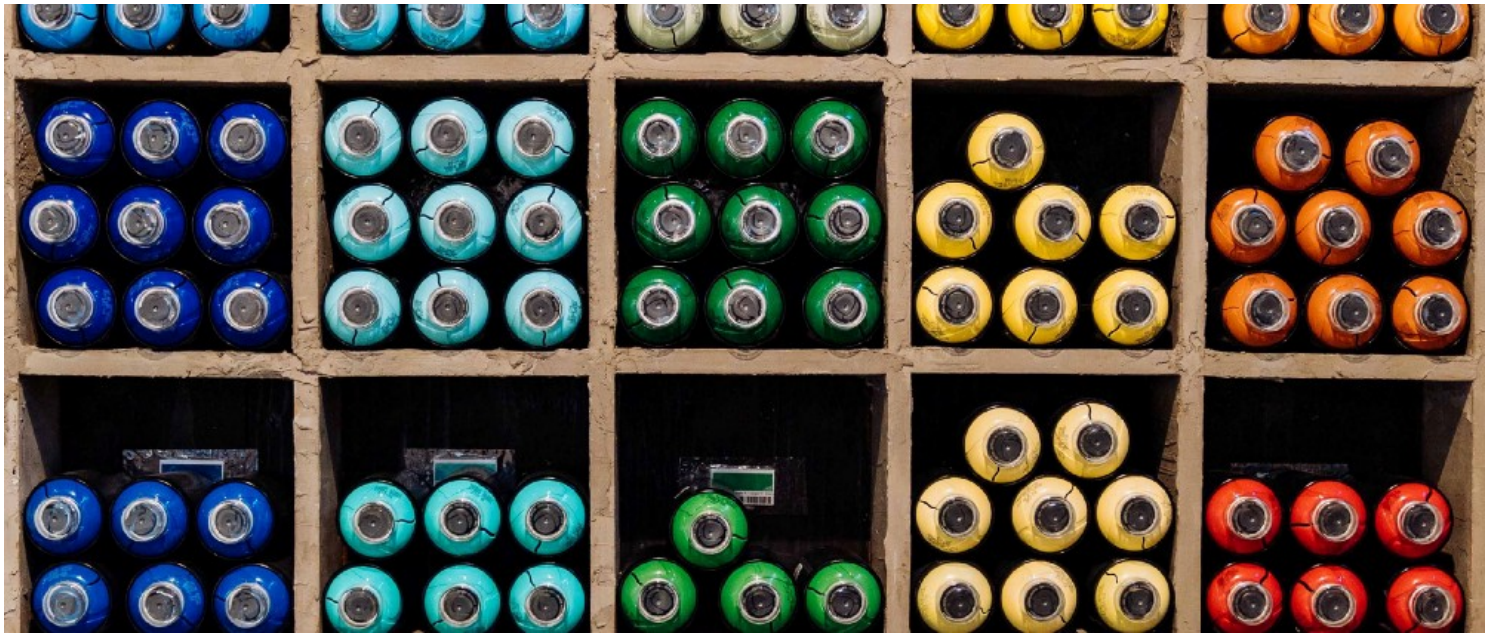


Photo by Edgar Chaparro on Unsplash

Classification is the task of assigning an observation to a category based on specific criteria.

In machine learning, classification is part of supervised learning, which means that the data used to train the model have labels that identify each category.

A critical step in the life cycle of a machine learning model is the evaluation of its performance.

Two techniques used to evaluate a classification model are the confusion matrix and the classification report.

In this post, we will learn to interpret the confusion matrix and the classification report while using them to evaluate the performance of a Support Vector Machine model on two common types of classification problems:

- **Binary classification**
- **Multiclass classification**

Let's import all the necessary libraries in Python.

```
# Libraries for data manipulation and visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Libraries and modules for Machine Learning
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

# Libraries and modules for model evaluation
from sklearn.metrics import confusion_matrix, classification_report
```

Binary classification

Breast cancer classification

In our binary classification problem, we will use the Wisconsin breast cancer dataset from sci-kit learn (sklearn). The goal is to train a model to predict whether biopsies of breast tissue are benign or malignant based on characteristics of the nuclei of the cells.

In binary classification, we refer to the class we want to predict (malignant) as Positive and the second class (benign) as Negative.

I will label the benign samples as 0 and the malignant samples as 1.

The dataset comes as a dictionary that also includes the following keys: data (independent variables), target(dependent variable), feature_names. In the section below, I will create the data frame and assign the proper labels to benign and malignant samples.

```
# Import the dataset Wisconsin breast cancer dataset from skelarn.

from sklearn.datasets import load_breast_cancer
cancer =  load_breast_cancer()

# Create a DataFrame.

cancer = pd.DataFrame(np.c_[cancer['target'], cancer['data']],
                      columns = np.append('target',
cancer['feature_names']))
```

```
# Assign the label 0 to benign samples and 1 to malignant samples
# change the datatype of target from float to integer.

cancer['target'] = cancer['target'].map({0: 1, 1:0}).astype('int64')

# Check the structure of the dataset

cancer.shape
```

```
(569, 31)
```

The dataset has 569 rows and 31 columns.

```
cancer.head(2)
```

	target	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points
0	1	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710
1	1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017

2 rows × 31 columns

Check the proportion of benign (0) and malignant (1) observations in the dependent variable target.

```
print(round(cancer['target'].value_counts()/len(cancer), 2))
```

```
0    0.63
1    0.37
Name: target, dtype: float64
```

63% of the observations are benign, and 37% are malignant.

Split the dataset into a train set (80%) and a test set(20%). Train the model on the train set and then use it to predict the class of the samples of the test set.

```
# Separate the independent variables from the dependent variable.

X = cancer.drop('target', axis = 1)
y = cancer['target']

# Divide the dataset into train (80%) and test (20%).

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 0)

# Create a Support Vector Machine object.

svm = SVC()

# Fit the model to the train set.

svm.fit(X_train, y_train)

# Predict the classes on the test set.

y_predict = svm.predict(X_test)
```

Now that we have the predictions, we need to evaluate the performance of our model.

Evaluating the model

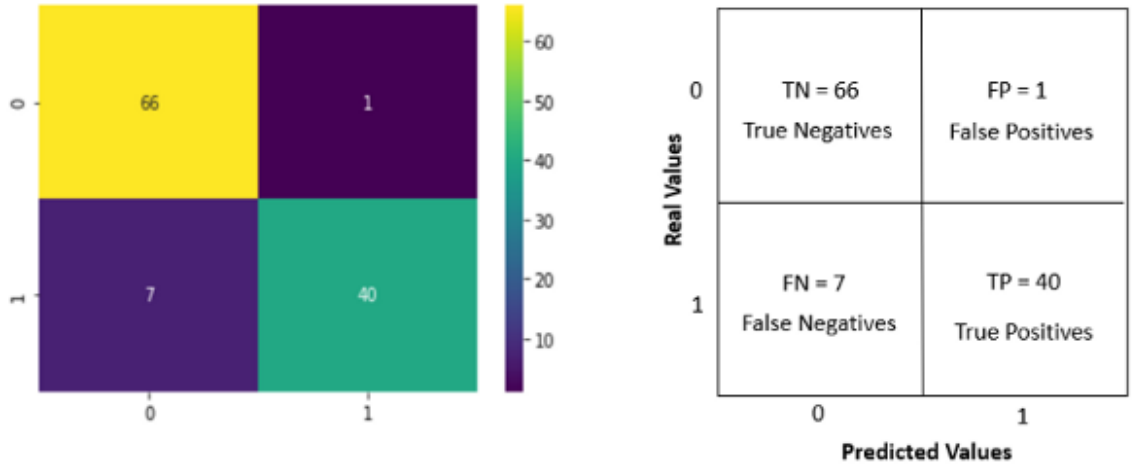
Confusion Matrix and Classification Report

The confusion matrix is an N x N table (where N is the number of classes) that contains the number of correct and incorrect predictions of the classification model.

To create the confusion matrix, we can use sklearn confusion_matrix(), which takes the real values (y_test) and the predicted values (y_predict).

We can use seaborn to print a heatmap of the confusion matrix.

```
sns.heatmap(confusion_matrix(y_test, y_predict), annot = True)
```



Confusion matrix heatmap(left) and scheme(right). Image by Author

The rows of the matrix represent the real classes, while the columns represent the predicted classes.

The values returned by the confusion matrix are divided into the following categories:

- **True Positive (TP):**
  The model predicted positive, and the real value is positive.
- **True Negative (TN):**
  The model predicted negative, and the real value is negative.
- **False Positive (FP):**
  The model predicted positive, but the real value is negative (Type I error).
- **False Negative (FN):**
  The model predicted negative, but the real value is positive (Type II error).

The diagonal from the top left to the bottom right contains the observations correctly predicted.

Let's evaluate our model using the data of the confusion matrix.

- Total predictions: 114
- Correct predictions: 106 (66 benign (TN) and 40 malignant (TP))

- Incorrect predictions: 8 (1 benign classified as malignant (FP) and 7 malignant classified as benign (FN))

Although detailed, the data above make it difficult to understand how good is the model at classifying cancer samples.

We can use the data of the confusion matrix to compute metrics that quantify the performance of the model.

Metrics for the evaluation of a classification model

Accuracy:
The accuracy returns the proportion of correct predictions.

Accuracy = (TP + TN)/(TP + TN + FP + FN) = (66 + 40)/(66 + 40 + 1 + 7) = 0.93 = 93%

Precision:
The precision returns the proportion of true positives among all the values predicted as positive.

Precision = TP/(TP + FP) = 40/(40 + 1) = 0.98 = 98%

Recall:
The recall returns the proportion of positive values correctly predicted.

Recall = TP/(TP + FN) = 40/(40 + 7) = 0.85 = 85%

Specificity:
The specificity returns the proportion of negative values correctly predicted.

Specificity: TN/ (TN + FP) = 66/(66 + 1) = 0.99 = 99%

F1-score:
The f1-score is the harmonic mean of precision and recall. It is often used to compare classifiers.

F1-score = (2 x Precision x Recall)/(Precision + Recall) = 0.91 = 91%

The harmonic mean gives more weight to the lower value, so a high F1-score means that both precision and recall are high.

For the rest of the post, we will focus on accuracy, precision, recall and f1-score.

We can use sklearn to compute the metrics above.

```
from sklearn.metrics import precision_score, accuracy_score,
recall_score, f1_score

print(f"Accuracy: {round(accuracy_score(y_test, y_predict), 2)}")
print(f"Precision: {round(precision_score(y_test, y_predict), 2)}")
print(f"Recall: {round(recall_score(y_test, y_predict), 2)}")
print(f"F1_score: {round(f1_score(y_test, y_predict), 2)}")
```


sklearn classification_report() returns precision, recall and F1-score for each class.

```
print(classification_report(y_test, y_predict))
```


Classification report breast cancer diagnosis

Apart from the evaluation metrics, the classification report includes some additional information:

Support: number of observations for each class.

Macro average: the arithmetic average of a metric between the two classes.

Macro average(precision) = (p0 + p1)/2 = (0.90 + 0.98)/2 = 0.94 = 94%

Weighted average: the weighted average is calculated by dividing sum(metric of interest x weight) by sum(weights).

Here the weights are the number of observation for each class.

Weighted average (precision)= (p0 x s0 + p1 x s1)/(s0 + s1) = (0.9 x 67) + (0.98 x 47)/(67 + 47) = 0.93 = 94%

What metric should we use to evaluate a classification model?

The metric we use to evaluate the model depends on two factors:

- Class balance
- Reason for the classification

The accuracy is extensively used to evaluate a classification model. It works well when the classes are balanced but it can be misleading when the classes are unbalanced.

Let's consider our cancer diagnosis problem again, but this time let's assume that only 5% of the observations are malignant. In this case, a model that returns benign for every sample will have an accuracy of 95%, but it will be of no use.

In this situation, it is crucial to detect all the malignant observations, even if it means classifying some benign samples as malignant.

Here, false negatives (type II errors) are more dangerous than false positives (type I errors).

In this scenario**, recall** is the best metric to evaluate the model.

Our model has a recall of 85%, which is not bad considered that we did not scale the data and did not optimize the model on the train set.

Let's now consider another binary classification problem: spam detection.

In this case, the goal is to identify as many spams as possible while not losing emails of interest.

Here type II errors (false negative) are more acceptable than type I errors (false positive).

In other words, it is better having a few spams than losing an email of interest.

In this scenario, **precision** is the best metric to evaluate the model.

Multiclass classification

Classification of Iris Species

In our multiclass classification problem, we will use the iris dataset from pydataset to assign the iris plant to one of three species: Setosa, Versicolor or Virginica based on length and width of both petals and sepals.

```
from pydataset import data
iris = data('iris')

iris.shape
```


The dataset has 150 rows and 5 columns. Let's print the first two rows.

```
iris.head(2)
```

I am going to replace the labels Setosa, Versicolor and Virginica with the numbers 0, 1 and 2, respectively, and then check the proportion of iris plants in each class.

```
iris['Species'].replace({'setosa': 0, 'versicolor': 1, 'virginica':
2}, inplace = True)

print(round(iris[‘Species’].value_counts()/len(iris), 2))
```


The dataset is balanced with about 33% of observations in each class.

```
# Separate the dependent variable from the independent variables.

Xi = iris.drop('Species', axis = 1)
yi = iris['Species']

# Divide the dataset into train (60%) and test (40%).
# SVM works very well on this dataset so I used a train-test ratio of
60-40 to introduce some misclassified values.

Xi_train, Xi_test, yi_train, yi_test = train_test_split(Xi, yi,
test_size = 0.4, random_state = 0)

# Create a Support Vector Machine object.
svmi = SVC()

# Fit the model to the train set.
svm.fit(Xi_train, yi_train)

# Predict the classes on the test set.
yi_predict = svmi.predict(Xi_test)
```

Model evaluation

As before, the real values are on the rows, and the predicted values are on the columns.

```
sns.heatmap(confusion_matrix(yi_test, yi_predict), annot = True)
```

Confusion matrix heatmap(left), scheme of TP, FP and FN(right). Image by Author

Each row contains one cell with TP and two cells with FN, while each column contains one cell with TP and two cells with FP.

Consider class 1:

TP1 are at the intersection row1-column1, FP1 are on column 1, and FN1 are on row 1.

$FP1 = 0 + 3 = 3$

$FN1 = 0 + 1 = 1$

Let's check the position of TN in the 3x3 matrix.

Confusion matrix heatmap(left), scheme of TP and TN(right). Image by Author.

To find the TN, we can eliminate the column and the row of the class of interest from the matrix. The cells that remain contain the TN.

For example, if we remove row 1 and column 1 from the matrix, the four cells that remain (the ones at the corners of the matrix) contain TN1.

TN1 = 18 + 0 + 16 + 0 = 34

Thinking in terms of rows and columns will make it easier to remember how to find all the elements that belong to a specific class.

Let's compute the accuracy of the model, and the precision, recall and F1-score for class 2.

Accuracy= (TP0 + TP1 + TP2)/(TP0 + TP1+ TP2 + FP0 + FP1 + FP2)= (16 + 22 + 18)/(16 + 22 + 18 + 0 + 3 + 1) = 0.93 = 93%

Precision(2) = TP2/(TP2 + FP2) = (18)/(18 + 1) = 0.95 = 95%

Recall(2) = TP2/(TP2 + FN2) = 18/(18 +3) = 0.86 = 86%

F1-score(2) = (2 x Precision x Recall)/ (Precision + Recall) = (2x 0.95 x 0.86)/(0.95 x 0.86) = 0.90 = 90%

we can get all the information about the performance of the model by using sklearn classification_report().

```
print(classification_report(yi_test, yi_predict))
```

The model correctly predicted all the observations of class 0, so both precision and recall are 1, and the F1-score is 1.

For class 1, the recall is higher than precision, which means that there are fewer FN1 (1) than FP1 (3).

For class 2, precision is higher than recall, which means that there are more FN2 (3) than FP2 (1).

Let's compute the macro average and the weighted average for recall.

Macro average (recall) = $(R0 + R1 + R2)/3 = (1.00 + 0.96 + 0.86)/3 = 0.94 = 94\%$

Weighted average(recall) $= ((R0 \times s0)+(R1 \times s1) + (R2 \times s2))/(s0 +s1 +s2) = ((1.00 \times 16) + (0.96 \times 23) + (0.86 \times 21))/ (16 + 23 + 21) = 0.93 = 93\%$

Everything we said about the 3x3 matrix is also valid for matrices with $N > 3$.

We are now ready to interpret the confusion matrix and the classification report for any number of classes.

Conclusions

In this post, we learned how to read the confusion matrix and compute accuracy, precision, recall and F1-score for both a binary and multiclass classification problem.

We also learned that no metric is good enough to evaluate all classification problems, but that each metric is the best choice in specific circumstances.

With this knowledge, we can interpret any classification report and use the information in it to confidently evaluate a model or compare different models.

Thank you for reading.

Classification Confusion Matrix Precision Recall Classification Report