**Confusion Matrix |**

**Summarizing the Performance of a Classification Algorithm**

Understanding what metrics to use to evaluate machine learning models is very important in a machine learning project.

Problem Statement for this article, we will examine the results on a dataset for a Binary Classification problem, which has two classes: YES (1) or NO **(**0**)**. We have test our model on

1,000 samples, and use the output from y\_true and y\_pred to walk through some key metrics

definitions. The following sklearn libraries have been imported to help perform these evaluations:

**from** **sklearn.metrics** **import** confusion\_matrix

**from** **sklearn.metrics** **import** accuracy\_score, precision\_score, recall\_score, f1\_score

Confusion Matrix is a tool for summarizing the performance of a classification

algorithm on a set of test data for which **true values** are known. It provides a matrix as output and

describes the complete performance of a model by giving us a clear picture of the performance and the

types of errors produced by the classifier. These values are presented in the form of a matrix where the Y-axis shows the True Label and the X-axis shows the Predicted Label, as shown below (Figure 1):

CODE:

# Note: y\_true are the true values of y provided by the input data; y\_pred are the predicted values of y.

cm = confusion\_matrix(y\_true, y\_pred)

print(f'Confusion Matrix:**\n** {confusion\_matrix(y\_true, y\_pred)}')

[Output:] Confusion Matrix:

[[722 37]

[102 139]]

f, ax = plt.subplots(figsize =(3,3))

sns.heatmap(cm,annot = True,linewidths=0.5,linecolor="white",fmt = ".0f",ax=ax)

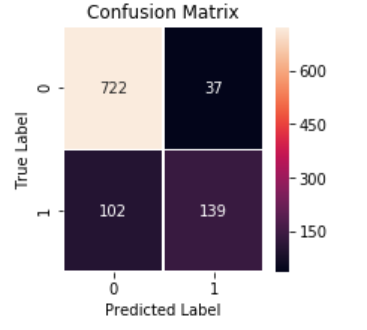
plt.title("Confusion Matrix")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.show()

[Output: **Figure 1**]



TP = cm[1,1] *# True Positives*

TN = cm[0,0] *# True Negatives*

FP = cm[0,1] *# False Positives - Type I Errors*

FN = cm[1,0] *# False Negatives - Type II Errors*

ttl\_obsv = TP+TN+FP+FN *# Total Observations*

print(f'Confusion Matrix Metrics: TP = **{TP}** | TN = **{TN}** | FP = **{FP}** | FN = **{FN}** || Total Obsv = **{** ttl\_obsv **}**')

[Output:] Confusion Matrix Metrics:

TP = 139 | TN = 722 | FP = 37 | FN = 102 || Total Obsv = 1000

Four types of outcomes are possible when evaluating a confusion matrix. These four outcomes are described below:

* **True Positives (TP):** The cases in which we predicted YES, and the actual output was also YES.
* **True Negatives (TN):** The cases in which we predicted NO, and the actual output was NO.
* **False Positives (FP):** The cases in which we predicted YES, and the actual output was NO. This type of error is also called “Type I errors”, which raises “false” alerts.
* **False Negatives (FN):** The cases in which we predicted NO, and the actual output was YES. This type of error is also called “Type II errors”.

Using the outcomes listed above, we can evaluate our classifier performance with

the metrics below:

Accuracy

**How often is the classifier “correct” when predicting both positive and negative**

**class?**

* Use accuracy when every class value is “equally” important in your performance evaluation.
* Accuracy does not tell you what “types” of errors your model is not

predicting correctly (misclassified predictions).

* Accuracy is a valid choice of evaluation for binary and multiclass classification problems.
* Accuracy can be a misleading metric for imbalanced data sets. Accuracy works well only if there are equal number of samples belonging to each class - well balanced and not skewed.
* Accuracy is the ratio of correct predictions to the total number of input samples.

**Formula:**

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

[Output:] 0.861

**Using sklearn.metric:**

display(accuracy\_score(y\_true, y\_pred))

[Output:] 0.861

Misclassification Rate

**How often is the classifier “incorrect” when predicting the class?**

* Use misclassification rate when you want to know the percentage of missed

predictions.

* Misclassification is the ratio of incorrect predictions to the total number of input samples.

**Formula:**

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

[Output:] 0.139

**Using sklearn.metric:**

display(1- accuracy\_score(y\_true, y\_pred))

[Output:] 0.139

Precision

**When a positive value is predicted, how often is the prediction “correct”?**

* Precision is often termed “**Positive Predictive Value**”.
* Use precision when we want to be “very sure” of our prediction.
* Usually, coupled with other metrics such as “recall”.
* Precision is the ratio of the number of true positives to the total number of “predicted” positives.

**Formula:**

precision = TP / (TP + FP)

[Output:] 0.7897727272727273

**Using sklearn.metric:**

display(precision\_score(y\_true, y\_pred))

[Output:] 0.7897727272727273

Recall

**When the actual value is positive, how often is the prediction “correct”?**

* Recall is often termed “**True** **Positive Rate**” or “**Sensitivity**”.
* Usually, coupled with other metrics such as “precision”.
* Widely used in the following domains: Medicine
* Recall is the ratio of the number of true positives to the total number of “Actual” positives.

**Formula:**

recall = TP / (TP + FN)

[Output:] 0.5767634854771784

**Using sklearn.metric:**

display(recall\_score(y\_true, y\_pred))

[Output:] 0.5767634854771784

Specificity

**When the actual value is negative, how often is the prediction “correct”?**

* Specificity is often termed “**False** **Positive Rate**”.

**Formula:**

recall = TN / (TN + FP)

[Output:] 0.9512516469038208

F1 Score

**Combines precision and recall into one metric.**

* F1 Score is the harmonic mean between precision and recall, it tries to find the balance between precision and recall.
* The F1 score is calculated based on the precision and recall of each class.
* It is the weighted average of the precision and the recall scores.
* The F1 score reaches its perfect value at 1 and worst at 0.
* It is a very good way to show that a classifier has good recall and precision values.
* F1 Score can also be used for multiclass problems.

**Formula:**

f1 = 2\*((precision\*recall)/(precision+recall))

[Output:] 0.6666666666666666

**Using sklearn.metric:**

display(f1\_score(y\_true, y\_pred))

[Output:] 0.6666666666666666

**References:**

* [Scikit-learn: Machine Learning in Python](http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html), Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
* [<https://towardsdatascience.com/evaluation-metrics-for-classification-problems-in-machine-learning-d9f9c7313190>](https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62)
* <https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>
* <https://towardsdatascience.com/common-classification-model-evaluation-metrics-2ba0a7a7436e>
* <https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226>
* <https://medium.com/@yashwant140393/the-3-pillars-of-binary-classification-accuracy-precision-recall-d2da3d09f664>
* <https://towardsdatascience.com/the-ultimate-guide-to-binary-classification-metrics-c25c3627dd0a#2f5c>

**Written By**

Sharonda Warner | Data Science Student at Flatiron School