**Classification Metrics**

* How much accuracy is good
* Accuracy score
* Confusion Matrix
* When accuracy score can miss lead you
* Precision, recall, F1 Score
* Multi-class with Precision, recall, F1 Score
* Micro precision
* Weighted precision
* Micro recall
* Weighted recall
* Micro F1 Score
* Weighted F1 score
* Or use classification report
* Formulas for all
* When to use which matrix

**Understanding Classification Evaluation Metrics**

Understanding classification [evaluation metrics](https://www.geeksforgeeks.org/metrics-for-machine-learning-model/) is crucial for assessing the performance of machine learning models, especially in tasks like binary or multiclass classification. Some common metrics are:

* Accuracy
* [Confusion Matrix](https://www.geeksforgeeks.org/confusion-matrix-machine-learning/)
* [Precision, Recall and F1 Score](https://www.geeksforgeeks.org/f1-score-in-machine-learning/)
* [AUC-ROC Curve](https://www.geeksforgeeks.org/auc-roc-curve/)

Let's consider the MNIST dataset and try to understand the metrics based on the classifier. MNIST has a set of 70,000 small, handwritten-digit images. Let's go through the dataset before we start.

**from** **keras.datasets** **import** mnist

**from** **keras.utils** **import** to\_categorical

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Conv2D, MaxPooling2D, Flatten, Dense

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, precision\_score, recall\_score, f1\_score, roc\_auc\_score, roc\_curve

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

(train\_X, train\_y), (test\_X, test\_y) = mnist.load\_data()

train\_X = train\_X.reshape((train\_X.shape[0], 28, 28, 1)).astype('float32') / 255

test\_X = test\_X.reshape((test\_X.shape[0], 28, 28, 1)).astype('float32') / 255

train\_y = to\_categorical(train\_y)

test\_y = to\_categorical(test\_y)

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D((2, 2)),

Flatten(),

Dense(100, activation='relu'),

Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

model.fit(train\_X, train\_y, epochs=3, batch\_size=200, validation\_split=0.2, verbose=2)

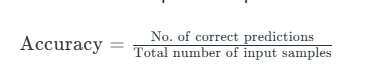
y\_pred = model.predict(test\_X)

y\_pred\_classes = y\_pred.argmax(axis=1)

y\_test\_classes = test\_y.argmax(axis=1)

**1. Accuracy**

**Accuracy**is a fundamental metric for evaluating the performance of a classification model, providing a quick snapshot of how well the model is performing in terms of correct predictions. It is calculated as the ratio of correct predictions to the total number of input samples.



It works great if there are an equal number of samples for each class. For example, we have a 90% sample of *class A* and a 10% sample of *class B* in our training set. Then, our model will predict with an accuracy of 90% by predicting all the training samples belonging to *class A*. If we test the same model with a test set of 60% from class A and 40% from class B. Then the accuracy will fall, and we will get an accuracy of 60%.

Accuracy is good but it gives a False Positive sense of achieving high accuracy. The problem arises due to the possibility of misclassification of minor class samples being very high.

from sklearn.metrics import accuracy\_score

accuracy = accuracy\_score(y\_test\_classes, y\_pred\_classes)

print(f'Accuracy: {accuracy:.4f}')

**2. Confusion Matrix**

The confusion matrix is another way to evaluate the performance of a classifier. Here, it counts the number of times instances of class A are classified as class B. For example, the number of times the classifier confused images of 5s with non-5s.

This is a table that is often used to describe the performance of a classification model. It presents a summary of the predictions made by the model against the actual class labels. The confusion matrix is a matrix with four different combinations of predicted and actual classes: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

Let's compute the confusion matrix to evaluate the performance of a classifier. We can make use of MNIST dataset to compute the confusion matrix. The stepsare as follows:

cm = confusion\_matrix(y\_test\_classes, y\_pred\_classes)

plt.figure(figsize=(10, 4))

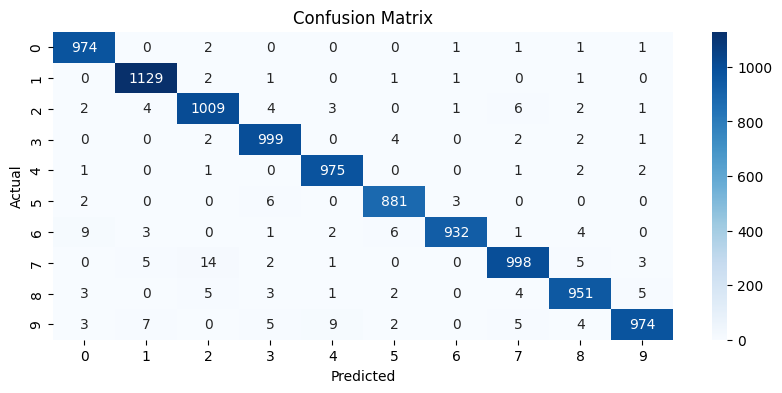
sns.heatmap(cm, annot=**True**, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()



**3. Precision, Recall and F1 Score**

A confusion matrix is a great way to evaluate the performance of a classifier, but sometimes we may need a more concise metric. Here comes the importance of precision.

**3.1 Precision:** Precision provides the accuracy of the positive prediction made by the classifier. The equation is as follows:

**Precision = True Positive / (True Positive + False Positive)**

***When to choose precision?***

In some cases, we need high precision. For example, consider that we trained a classifier to detect videos that are safe for kids. Here, we prefer a classifier that keeps only the safe one (high precision), irrespective of whether the classifier rejects many good videos (low recall).

Precision is typically used with another metric called recall (sensitivity, or the true positive rate ━ TPR).

**3.2 Recall**: Recall is the ratio of number of true positive predictions (correctly detected by the classifer) to the total number of actual positive instances in the dataset. It measures the completeness of positive predictions. The equation is as follows:

**Recall = True Positve / (True Positive + False Negative)**

***When to choose recall?***

In some cases, high recall is given importance instead of high precision. Suppose you train a classifier for fire detection with high precision; certain actual cases were not considered. So it is important to maintain a high recall. Here, security guards will get a few false alarms, but they will be alarmed in almost every actual case.

**3.3 F1 Score**: The F1 score is the harmonic mean of precision and recall. It favors classifiers that have similar precision and recall. Here, the classifier will only get a high F1 score if both recall and precision are high. The equation is as follows:

F1 = 2 \* (Precision \* Recall) / (Precision + Recall)

**When to choose F1 Score?**

F1 Score is invaluable in binary classification tasks, especially with imbalanced datasets, where accuracy can be misleading. It strikes a balance between precision and recall, crucial in scenarios where both are equally important, like medical diagnosis. This metric effectively captures the trade-off between precision and recall, offering a comprehensive evaluation of model performance.

For implementation refer to code below:

precision = precision\_score(y\_test\_classes, y\_pred\_classes, average='macro')

recall = recall\_score(y\_test\_classes, y\_pred\_classes, average='macro')

f1 = f1\_score(y\_test\_classes, y\_pred\_classes, average='macro')

print(f'Precision: **{**precision**:**.4f**}**')

print(f'Recall: **{**recall**:**.4f**}**')

print(f'F1 Score: **{**f1**:**.4f**}**')

**Output:**

Precision: 0.9823  
Recall: 0.9821  
F1 Score: 0.9822

In the above code, we make use of the f1\_score() method from the sklearn metric to calculate the F1 score.

**4. ROC Curve**

The Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a classification model at various thresholds. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR).***The Area Under the ROC Curve (AUC-ROC) is a metric to evaluate the performance of a binary classification model. AUC-ROC value lies between 0 and 1, where a higher value indicates better performance***. AUC-ROC is insensitive to class distribution and gives an aggregate measure of performance across all possible classification thresholds.

The true positive rate is calculated as:

**TPR = True Positives / (True Positives + False Negatives)**

It defines how good the model is at predicting the positive class for a positive outcome. It is also known as sensitivity.

The false positive rate is calculated as:

**FPR = False Positives / (False Positives + True Negatives)**

It is also referred to as inverted specificity (1 - specificity), where specifity is calculated as:

***Specificity = True Negative / (True Negaive + False Positive)***

Let's get to the implementation part using Sklearn.The code is as follows:

*# Assuming y\_test and y\_pred\_prob are the true labels and predicted probabilities respectively*

y\_pred\_prob = model.predict(test\_X)

roc\_auc = roc\_auc\_score(test\_y, y\_pred\_prob, multi\_class='ovr')

print(f'ROC AUC: **{**roc\_auc**:**.4f**}**')

*# Plotting ROC Curve for one class (e.g., class 0)*

fpr, tpr, \_ = roc\_curve(y\_test\_classes == 0, y\_pred\_prob[:, 0])

plt.plot(fpr, tpr, label='Class 0 ROC curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc='best')

plt.show()