Calculate Balanced Accuracy in Python Using sklearn

Authored by

Mohammed looti

November 2, 2025

Understanding model performance is critical in machine learning, and while standard **accuracy** is often the first metric considered, it can be misleading, especially when dealing with complex datasets. This is where **Balanced accuracy** steps in, providing a robust and reliable measure for assessing the quality of a **classification model**.

Balanced accuracy is particularly useful because it accounts for potential class distributions, ensuring that the model performs equally well across all categories. Unlike traditional accuracy, which simply calculates the ratio of correct predictions to total predictions, balanced accuracy adjusts for scenarios where one class significantly outweighs the others. This adjustment prevents overly optimistic evaluations derived from models that simply predict the majority class most of the time.

Mathematically, balanced accuracy is defined as the arithmetic mean of sensitivity (True Positive Rate) and specificity (True Negative Rate). This definition highlights its focus on evaluating performance independently for both positive and negative classes, ensuring a fair assessment regardless of the dataset structure.

The Mathematical Foundation: Sensitivity and Specificity

To fully grasp balanced accuracy, we must first understand its core components: **Sensitivity** and **Specificity**. These terms are derived directly from the results summarized in a **confusion matrix**, which maps out the four possible outcomes of any binary classification task: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

The calculation relies on the following fundamental relationship:

Balanced accuracy = (Sensitivity + Specificity) / 2

Let's define these key rates more formally:

Sensitivity (or the True Positive Rate): This measures the proportion of actual positive cases that were correctly identified by the model. It is calculated as TP / (TP + FN). Sensitivity is crucial when minimizing false negatives is paramount.

Specificity (or the True Negative Rate): This measures the proportion of actual negative cases that were correctly identified. It is calculated as TN / (TN + FP). Specificity is essential when minimizing false positives is the primary objective.

By averaging these two rates, **balanced accuracy** ensures that a model cannot achieve a high score by performing excellently on one class while completely failing on the other. It demands competence across the board, providing a holistic and robust measure of predictive capability.

The Problem of Imbalanced Data

The necessity of using balanced accuracy becomes strikingly clear when dealing with **imbalanced data**. An imbalanced dataset is characterized by a significant disparity in the number of observations belonging to each class. For instance, in rare disease diagnosis or fraud detection, genuine cases vastly outnumber the minority class of interest.

In such skewed scenarios, standard accuracy is highly misleading. A model biased toward the majority class (e.g., predicting 0s correctly 99% of the time but failing to detect any 1s) would appear successful based on the overall accuracy score. However, this model has poor predictive power for the minority class, which is often the class we care about most.

Balanced accuracy addresses this flaw directly. By calculating the average of the class-specific recall rates, it automatically penalizes models that exhibit strong performance on the majority class but poor performance on the minority class. A high balanced accuracy score (close to 1.0) indicates that the model is performing well on both the positive and negative classes simultaneously, making it the preferred metric for robust evaluation in real-world applications where **imbalanced data** is common.

Practical Example: Predicting NBA Draft Outcomes

Consider a scenario where a sports analyst develops a **classification model** to predict whether 400 college basketball players will be drafted into the NBA (Class 1) or not (Class 0). In this highly selective environment, the dataset is severely imbalanced, as only a small fraction of players are actually drafted.

The following **confusion matrix** summarizes the predictions made by the analyst's model. Note that the total number of negative cases (Not Drafted) far exceeds the positive cases (Drafted), illustrating the imbalance:

		Predicted	
		Drafted = Yes	**Drafted = No**
Actual	**Drafted = Yes**	**15** (True Positive)	**5** (False Negative)
	Drafted = No	**5** (False positive)	**375** (True Negative)

To calculate the balanced accuracy manually, we first need to determine the **Sensitivity** (True Positive Rate) and **Specificity** (True Negative Rate) based on the figures presented in the matrix:

Sensitivity (True Positive Rate): Calculates how well the model identifies drafted players (15 TP / (15 TP + 5 FN)) = 15 / 20 = 0.75

Specificity (True Negative Rate): Calculates how well the model identifies non-drafted players (375 TN / (375 TN + 5 FP)) = 375 / 380 ≈ 0.9868

Using these intermediate results, we proceed to calculate the final balanced accuracy score:

Balanced accuracy = (Sensitivity + Specificity) / 2

Balanced accuracy = (0.75 + 0.9868) / 2

Balanced accuracy = 0.8684

The resulting **balanced accuracy** for this model is **0.8684**. If we had only relied on standard accuracy, which would be (15+375)/400 = 0.975, we might mistakenly assume the model is near perfect. Balanced accuracy correctly reveals that while the model is excellent at identifying the majority class (Specificity), its performance is slightly weaker on the minority class (Sensitivity), providing a much clearer picture of its limitations.

Implementing Balanced Accuracy in Python using sklearn

While manual calculation is essential for understanding the underlying mechanics, in a practical data science workflow, utilizing optimized libraries is standard practice. The **sklearn** library (scikit-learn) provides the powerful `balanced_accuracy_score()` function, which handles this calculation efficiently and scales well for large datasets.

The following Python code demonstrates how to define the arrays representing the predicted and actual classes from our NBA draft example and subsequently calculate the balanced accuracy score using **sklearn** metrics. This automation ensures accuracy and simplifies the model evaluation process significantly.

import numpy as np
from sklearn.metrics import balanced_accuracy_score

#define array of actual classes
actual = np.repeat(, repeats=)

#define array of predicted classes
pred = np.repeat(, repeats=)

#calculate balanced accuracy score
balanced_accuracy_score(actual, pred)

0.868421052631579

As confirmed by the output, the score calculated by the **balanced_accuracy_score()** function is **0.8684**. This perfectly aligns with the result derived from our earlier step-by-step manual calculation, validating the implementation and demonstrating the convenience of using specialized libraries.

For advanced use cases, the `balanced_accuracy_score()` function also supports an optional parameter to adjust the calculation to return the macro-averaged recall, which is equivalent to balanced accuracy when sample weights are uniform. Understanding the official documentation is key to leveraging its full power within your data science projects.

Conclusion and Further Reading

Balanced accuracy is an indispensable metric for anyone working with **imbalanced data** or datasets where equal importance must be placed on the correct classification of all categories. By averaging the recall for each class, it provides a far more honest and informative evaluation of model performance than standard accuracy can offer.

Mastering the use of metrics like balanced accuracy, **sensitivity**, and **specificity** is a foundational skill in data science, ensuring that model deployment decisions are based on reliable performance indicators rather than misleading results from skewed data distributions.

For those interested in exploring related classification metrics or diving deeper into the functionality of the Python implementation, the following resources are highly recommended:

Additional Resources

Detailed documentation for the **balanced_accuracy_score()** function, including implementation notes and use cases, can be found on the official **sklearn** website. Additionally, exploring other robust metrics like the F1-score and Matthews correlation coefficient can further enhance your model evaluation toolkit.