Evaluation Metrics for Classification Tasks in Machine Learning: A brief overview of Accuracy, Error Rate, Precsion, Recall, F1 Score, Specificity, ROC, and AUC

Abdi Vicenciodelmoral

Introduction

Classification is a fundamental task in machine learning that involves assigning a discrete or categorical label to an observation based on its features, using a model trained on a dataset with known labels. Its applications can include tasks such as object classification, bot detection, and medical diagnosis. Evaluating the performance of classification models is vital to ensure their effectiveness in making accurate predictions. Model evaluation is primarily conducted through various metrics, each providing insights into different aspects of its performance.

The Role of Evaluation Metrics

Evaluation metrics offer quantitative measures that assess the performance of models, enabling comparisons, guiding improvements, and supporting their overall development. In classification tasks, these metrics demonstrate a model's ability to correctly distinguish between classes, displaying their strengths and identifying areas that may need improvement. Metrics serve multiple purposes, from aligning the model with its objectives to navigating the trade-offs required in model tuning.

This review provides a brief overview of different measures used to evaluate classification models, such as accuracy, error rate, precision, recall, Receiver Operating Characteristic (ROC) curve, and Area Under the Curve (AUC). Each metric offers a different perspective on performance; for instance, accuracy provides a general indication of success. Precision and recall, on the other hand, offer detailed insights into predictive reliability, which is critical when the consequences of false positives or negatives are significant.

The selection of appropriate metrics depends on the application's specific requirements and the dataset's characteristics. The following sections detail each metric's significance, application, and associated trade-offs. This survey aims to deepen the understanding of evaluation metrics and to provide guidance on selecting and interpreting these metrics effectively, ensuring comprehensive and accurate model evaluation.

Accuracy

Definition and Formula

Accuracy is one of the most direct metrics used to evaluate the performance of a machine learning model, especially in classification tasks [28, 10]. It quantifies the proportion of correct predictions (both true positives and true negatives) out of all predictions made by the model. The formula for accuracy is given by:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

where:

- TP is the number of true positives
- TN is the number of true negatives
- FP is the number of false positives
- FN is the number of false negatives

Accuracy is highly intuitive and simple to understand, making it a popular choice for initial model evaluation. It provides a quick snapshot of the model's overall effectiveness in making correct predictions. This simplicity is particularly appealing in educational settings or when explaining the performance of the model to individuals without technical expertise.

Evaluating Model Accuracy in Different Scenarios

Email Classification

Consider the task of classifying emails into two categories: "Spam" and "Not Spam". After developing and testing a model on a set of 100 emails, we obtain the following results:

- True Positives (TP): 30 (emails correctly identified as Spam)
- True Negatives (TN): 60 (emails correctly identified as Not Spam)
- False Positives (FP): 5 (emails incorrectly identified as Spam)
- False Negatives (FN): 5 (emails incorrectly identified as Not Spam)

With these values, we calculate the model's accuracy using the formula:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Plugging in the numbers:

$$\text{Accuracy} = \frac{30 + 60}{30 + 60 + 5 + 5} = \frac{90}{100} = 0.90$$

Therefore, the accuracy of our email classification model is 90%. This demonstrates the model's effectiveness in correctly identifying emails as either Spam or Not Spam.

Disease Diagnosis

Consider a different scenario involving the diagnosis of a disease from medical test results. Evaluating the model on 200 patients, we obtain:

- True Positives (TP): 40 (patients correctly identified as having the disease)
- True Negatives (TN): 150 (patients correctly identified as not having the disease)
- False Positives (FP): 5 (patients incorrectly identified as having the disease)
- False Negatives (FN): 3 (patients incorrectly identified as not having the disease)

We calculate the diagnostic model's accuracy:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Substituting the given values:

$$\text{Accuracy} = \frac{40 + 150}{40 + 150 + 5 + 3} = \frac{190}{198} = 0.9596$$

Thus, the accuracy of our disease diagnosis model is 95.96%, indicating its high effectiveness in correctly diagnosing patients when tested on this set of 200 patients.

These examples illustrate the use of accuracy as a measure of model performance in different scenarios. However, it is crucial to also consider other factors, such as the balance of datasets and the significance of minimizing certain types of errors, to fully assess a model's effectiveness.

Limitations of Accuracy

Despite its intuitiveness, accuracy has significant limitations, particularly in cases involving imbalanced datasets:

- **Imbalanced Datasets:** In cases where one class significantly outnumbers the other class or classes, accuracy can be misleading. A model could achieve high accuracy by simply predicting the majority class for all instances, while failing to capture the minority class effectively. This can result in the model's inability to classify the less prevalent class accurately, to be overlooked.
- **Ignores the Cost of Different Types of Errors:** Accuracy treats all types of errors equally, without considering the varying costs associated with false positives and false negatives. In many real-world applications, the impact of these errors can differ significantly.
- **Threshold Dependency:** Sensitivity to threshold choices is a general characteristic of evaluation metrics for binary classification. For models that output probabilities, such as logistic regression, several performance metrics, depend on the threshold chosen to classify predictions as positive or negative. Different thresholds can lead to significantly different accuracy measurements, making it difficult to compare models or to understand a model's performance without considering the threshold used. The decision on what probability threshold to use for classifying an instance as belonging to a particular class can significantly impact the model's perceived performance as measured by the accuracy rate.

  A higher threshold could lead to more false negatives, by being overly cautious about predicting positives, while a lower threshold could increase false positives, by being too lenient in predicting positives. Since the accuracy rate is calculated based on these binary outcomes, the choice of threshold directly impacts it.

Examples Illustrating When Accuracy Might Be Misleading

- **Disease Diagnosis:** In a medical dataset where 95% of the samples are of non-diseased individuals and only 5% are of diseased individuals, a model that predicts 'non-diseased' for all patients would achieve an accuracy of 95%. Despite the high accuracy, this model fails entirely to identify diseased individuals, which is critical in medical diagnostics.
- **Fraud Detection:** In financial transactions where only a small fraction might be fraudulent, such as 1%, a model designed to detect fraud that predicts all transactions as legitimate could still achieve 99% accuracy. This model, while accurate in a general sense, is practically useless for fraud detection purposes as it fails to identify any fraudulent transactions.

Accuracy is a useful metric for assessing a machine learning model's overall performance, but understanding its limitations, especially with imbalanced datasets, is critical. Relying solely on accuracy may yield misleading conclusions about the effectiveness of the model. Complementing accuracy with other metrics is often necessary for a more detailed view of performance, particularly when the cost of errors varies or in the presence of imbalanced classes.

Misclassification Rate (Error Rate)

Definition and Formula

The misclassification rate, or error rate, measures the frequency at which a classification model makes incorrect predictions [28, 7]. It's a straightforward metric that assesses the overall accuracy of the model by considering how often it fails to predict the correct labels. The formula for calculating the error rate is as follows:

$$\text{Error rate} = \frac{FP + FN}{TP + TN + FP + FN}$$

Where,

- FP represents false positives (instances incorrectly predicted as positive)
- FN represents false negatives (instances incorrectly predicted as negative)
- TP represents true positives (instances correctly predicted as positive)
- TN represents true negatives (instances correctly predicted as negative)

The appeal of the error rate lies in its clarity and straighforwardness, as in the accuracy metric. It directly tells you the proportion of predictions your model got wrong, providing a clear and concise measure of performance. For educators, students, and practitioners new to machine learning, this metric can be an excellent starting point for evaluating model accuracy since it doesn't require deep statistical knowledge to understand or calculate.

Evaluating Error Rate in Different Scenearios

Email Classification

As an example scenario, a machine learning model was employed to filter out spam emails. After testing the model on a dataset of 1,000 emails, the following results were attained:

- True Positives (TP): 200 emails correctly identified as spam.
- True Negatives (TN): 700 emails correctly identified as not spam.
- False Positives (FP): 50 emails incorrectly identified as spam (legitimate emails marked as spam).
- False Negatives (FN): 50 emails incorrectly identified as not spam (spam emails not caught).

Using the formula for the error rate:

$$\text{Error rate} = \frac{FP + FN}{TP + TN + FP + FN}$$

Plugging in the numbers from our scenario:

$$\text{Error rate} = \frac{50 + 50}{200 + 700 + 50 + 50} = \frac{100}{1000} = 0.10$$

This means the model has a error rate of 10%. In other words, 10% of the email classifications made by this model are incorrect.

Disease Diagnosis

Another example could be the application of a diagnostic model designed to identify a specific disease from patient test results. After evaluating the model on a dataset of 1,000 patients, the following outcomes were produced:

- True Positives (TP): 150 patients correctly identified as having the disease.
- True Negatives (TN): 780 patients correctly identified as not having the disease.
- False Positives (FP): 20 patients incorrectly identified as having the disease (healthy patients misdiagnosed).
- False Negatives (FN): 50 patients incorrectly identified as not having the disease (sick patients missed).

The formula for the error rate is the same as before:

$$\text{Error rate} = \frac{FP + FN}{TP + TN + FP + FN}$$

And substituting our values:

$$\text{Error rate} = \frac{20 + 50}{150 + 780 + 20 + 50} = \frac{70}{1000} = 0.07$$

This calculation indicates an error rate of 7%, meaning that 7% of the model's diagnoses are incorrect.

Considerations of Error Rate in the context of Disease Diagnosis

In the context of disease diagnosis, a 7% error rate signifies that for every 100 patients, 7 are incorrectly diagnosed. This error rate encompasses both false positives and false negatives, each having distinct implications:

- False Positives (FP): Misdiagnosing healthy patients as having the disease can lead to unnecessary worry, further testing, and potentially unwarranted treatment, impacting the patient's quality of life and incurring unnecessary healthcare costs.
- False Negatives (FN): Failing to diagnose patients who actually have the disease can be particularly dangerous, as it may prevent them from receiving timely and potentially life-saving treatments. In diseases where early intervention is critical, the cost of false negatives can be extremely high, both in terms of patient health outcomes and subsequent healthcare costs.

Limitations of Error Rate

Similar to the accuracy metric, the error rate has several limitations that can impact its effectiveness as a performance metric:

- **Imbalanced Classes:** In the presence of imbalanced datasets, like the accuracy metric, the error rate can be misleading. For instance, in a dataset with 95% of one class, a naive model that always predicts the majority class will have a 5% error rate, which superficially appears excellent despite the model not learning anything meaningful.
- **Class Importance Ignored:** It treats all misclassifications equally, ignoring the potential varying costs of different types of errors. For example, in medical diagnosis, the cost of a false negative is often much higher than a false positive.
- **Threshold Dependency:** This issue of threshold dependency isn't unique to the accuracy rate, but applies to all evaluation metrics that rely TP, FP, TN, and FN. This includes the error rate. Changes in the probability threshold significantly impacts the model's perceived performance, and must therefore

be taken into consideration for a comprehensive understanding of model performance evaluation in probabilistic settings.

Examples Illustrating of When Error Rate Might Be Misleading

Imbalanced Classes: Consider a medical diagnosis example where a machine learning model is utilized to predict a rare disease. In the dataset, the disease is present in 1% of the population. A model that always predicts "healthy" would have a 99% accuracy, or a 1% error rate, which might seem outstanding at first glance. However, this model fails entirely at its primary task of identifying the diseased cases, illustrating how the error rate can be deceptive with imbalanced classes.

Class Importance Ignored: In another scenario, Model A has a 5% error rate primarily due to false positives, while Model B has the same error rate but mostly because of false negatives. Despite having the same error rates, the implications are vastly different in a medical context, where false negatives are far more critical than false positives, as failure to detect the disease could be life threatening. The error rate doesn't capture this distinction.

Understanding both the utility and the limitations of the error rate can help in more effectively navigating its application in machine learning projects and better communicating these nuances in educational settings.

Introduction to Precision and Recall

Precision and recall provide valuable metrics in the evaluation of classification models. These metrics are of particular importance in situations where the impact of consequences associated with false positives are distinct from those related to false negatives. This difference in impact is crucial in fields such as pedestrian detection in automated vehicles and bone fracture detection in medical settings. For example, in medical diagnosis, the consequences of a machine learning model missing a serious injury (a false negative) can have far more grave implications than mistakenly identifying someone as having a fracture (a false positive), which might only lead to further tests. Therefore, it is vital to evaluate models carefully, taking into account the specific nature and potential severity of errors in various contexts.

Precision

Definition and Formula

Precision is a metric that evaluates the accuracy of the positive predictions made by a classification model [28, 8]. It answers the question: "Of all the instances the model labeled as positive, how many were actually positive?". Precision essentially quantifies the model's proficiency in reducing false positives, instances where a negative instance is incorrectly identified as positive. It is particularly crucial when the cost of a false positive is high, as it helps to minimize these incorrect positive predictions [2]. The "cost" of a false positive in machine learning refers to the negative outcomes, such as financial loss, health risks, reputational damage, operational inefficiency, or social and psychological impacts, that arise from incorrect positive predictions by a model. Minimizing these costs is crucial, making precision an important metric in contexts where false positives have significant adverse consequences.

The formula for precision is straightforward:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Where:

- TP stands for True Positives (cases where the model correctly predicts the positive class)
- FP stands for False Positives (cases where the model incorrectly predicts the negative class as positive)

This formula gives us a direct measure of the model's accuracy in predicting positive instances out of all instances it predicted as positive.

Limitations of Precision in Evaluating Model Performance

The effectiveness of precision as an isolated measure is constrained and limited by several factors that need to be considered in any machine learning task in order to ensure the results accurately represent the model performance. One significant limitation is its inability to account for the number of false negatives, which can lead to an overestimation of a model's accuracy in scenarios where incorrectly classifying positive cases is critical. Furthermore, precision does not provide insight into the model's ability to correctly identify negative instances, potentially overlooking its overall effectiveness across different classes within the dataset. This neglect results in a distorted perception of a model's performance. While high precision suggests effectiveness in decreasing false positive, it does not reflect the model's capability in recognizing all actual positives (True Positives + False Negatives).

The Challenge of Minimizing False Positives

The primary objective to enhancing precision is to minimize a model's false positives predictions, which carry significant implications across various domains:

In medical diagnostics, reducing false positives is crucial to prevent unnecessary anxiety and the potential for patients to undergo unnecessary, possibly invasive, follow-up examinations.

In spam detection, lowering false positives is vital to ensure that legitimate emails are not misclassified as spam, safeguarding important communications.

In legal and surveillance systems, it is critical to diminish false positives to avoid mistakenly implicating innocent individuals or subjecting them to unwarranted scrutiny.

Nonetheless, focusing solely on reducing false positives without considering the impact on false negatives may result in a model that, despite being precise, fails to identify a considerable number of positive instances. This issue is acutely significant in areas such as disease screening, where not detecting a positive case can lead to severe repercussions.

Imbalanced Datasets

The accuracy of a model's predictions can be less reliable when dealing with imbalanced datasets. This imbalance can make the model's predictions less accurate, especially for the minority class. In these scenarios, a model may achieve high precision through a strategy that results in it predicting positive cases less frequently. This happens because the model can be designed to identify positive cases only when there's a strong likelihood, minimizing the occurrence of false positives. This is typically done by increasing the model's probability threshold, during classification of the predictions. However, this approach leads to a significant increase in false negatives, where true positive cases are incorrectly labeled as negative instead. These methods, while effective in reducing incorrect positive predictions, inadvertently misses many actual positive instances. As a result, even though the model demonstrates high precision by accurately identifying a higher proportion of true positives among its positive predictions, it becomes less practical. Its effectiveness is compromised because it fails to recognize many genuine positive cases, highlighting a trade-off between precision and the ability to capture all positive instances accurately.

Recall

Recall, also known as the true positive rate or sensitivity, is another essential metric in machine learning, particularly for classification tasks. It measures how accurately a model identifies all the true positive instances within a dataset [28, 8]. Conceptually, recall seeks to address the query: "Of all the actual positives, how many has the model correctly identified?"

The mathematical expression for recall is defined as follows:

$$\text{Recall} = \frac{TP}{TP + FN}$$

where:

- TP represents True Positives (instances accurately identified as positive)
- FN represents False Negatives (instances incorrectly classified as negative)

This formula illustrates recall's emphasis on the minimization of false negatives.

Limitations of Recall

Although recall is important, focusing solely on it can lead to negative consequences, especially if it causes the model to ignore precision. This can lead to development of a model that identifies almost all positive cases, exhibiting high recall, but also makes many false positive predictions, exhibiting low precision. This situation can be especially problematic in some cases. For example, in spam detection, a model with high recall but low precision could wrongly mark many legitimate emails as spam. This could annoy users and block important messages. Therefore, it's important to find a careful balance between recall and other evaluation measures like precision to make sure the model works well for the specific needs and consequences of the task. We often achieve this balance by using metrics like the F1 score, which combines precision and recall. This gives us a fuller view of the model's effectiveness.

Minimizing False Negatives

Recall is significantly important in scenarios where the consequences of overlooking a true positive can be detrimental. For instance, in the domain of disease screening within the healthcare sector, failing to identify a true positive (resulting in a false negative) can harm patient health. It might delay or even miss the chance for the needed treatment. Therefore, having a high recall rate is critical to make sure we identify all positive cases.

Significant fields where the recall metric offers invaluable insights:

- **Medical Diagnosis and Screening:** In the field of healthcare, especially in diagnostics and screening for diseases such as cancer, high recall is essential. Missing a true positive case could mean a delay in the diagnosis and treatment of a serious condition, potentially leading to life-threatening situations. Therefore, it's vital to capture as many true positive cases as possible to ensure timely intervention, even if it means accepting a higher number of false positives that can be ruled out with further testing.
- **Fraud Detection:** In financial services, especially in identifying fraud or illicit transactions, a high recall rate ensures that nearly all fraudulent activities are detected. The cost of missing such activities can be very high, both financially and in terms of customer trust. Businesses often prefer to investigate a larger number of potential fraud cases, including some false positives, to ensure that they catch as many true instances of fraud as possible.
- **Public Safety and Security:** When it comes to public safety and security measures, such as surveillance for terrorist activities or identifying safety threats, recall is crucial. The risk of not detecting a genuine threat could have catastrophic consequences. Hence, authorities often opt for systems with higher recall to ensure comprehensive threat detection, even at the expense of higher false positive rates, which can be subjected to further scrutiny.

In each of these scenarios, the emphasis on recall is driven by the imperative to minimize the risk associated with false negatives.

The Relationship Between Precision and Recall

The trade-off between precision and recall arises from their definitions and the way they measure model performance. Precision, the ratio of true positive predictions to all positive predictions, measures how accurately the model identifies positive instances that are actually correct. In other words, it reflects the model's ability to avoid labeling negative instances as positive. Recall, conversely, assesses the model's ability to capture all actual positive instances, calculated as the ratio of true positive predictions to the total of true positives and false negatives. This means it evaluates the model's effectiveness in not missing any positive instances.

This inherent tension between precision and recall stems from the model's classification threshold: adjusting this threshold to reduce one type of classification error inevitably increases the other. Thus, enhancing precision often leads to a decrease in recall, and vice versa, reflecting a fundamental compromise in model performance optimization [11].

Threshold Definition

In classification tasks, the term "threshold" refers to a cut-off value used to determine the classification of instances based on a model's output.

Many classification models, especially binary classifiers, produce a probability score or some continuous output that represents the confidence level of an instance belonging to a positive class versus a negative class.

The threshold is a predefined value within the range of the model's output, often between 0 and 1. If the model's output for an instance is above the threshold, that instance is classified as belonging to the positive class. Conversely, if the output is below the threshold, the instance is classified as belonging to the negative class. Figure 1, presents a visual representation of a threshold cut-off at 0.5.

Example: In a spam detection model, if the threshold is set at 0.5, and an email has a spam probability score of 0.7, it would be classified as spam since 0.7 is above the threshold.

The choice of threshold value has a significant impact on the model's precision and recall. Lowering the threshold means more emails are classified as spam, increasing recall but possibly decreasing precision. Raising the threshold means fewer emails are classified as spam, potentially increasing precision but decreasing recall [24].

Figure 1: Visual representation of the probability threshold determining class assignment in a predictive model.

Balancing Precision and Recall

Selecting the appropriate threshold is crucial because it directly influences the trade-off between catching as many positive instances as possible (high recall) and ensuring the correctness of those positive predictions (high precision). Careful calibration of the threshold allows model developers to balance precision and recall according to the specific needs and priorities of their application, thereby optimizing the model's overall performance and utility.

Precision and Recall Example: Disease Screening Scenario

Let's assume that a community health program conducts a screening for Disease X among 1,000 individuals. The outcomes of this screening are as follows:

- True Positives (TP): 50 individuals who actually have Disease X are correctly identified by the screening test.
- False Positives (FP): 150 individuals do not have Disease X but are incorrectly identified by the screening test as having the disease.

- True Negatives (TN): 780 individuals do not have Disease X and are correctly identified by the screening test as not having the disease.
- False Negatives (FN): 20 individuals actually have Disease X but are incorrectly identified by the screening test as not having the disease.

Precision and Recall Definitions and Calculations

Precision focuses on the accuracy of positive predictions. As defined:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Plugging in the numbers, the precision calculation is:

$$\text{Precision} = \frac{50}{50 + 150} = \frac{50}{200} = 0.25 \text{ or } 25\%$$

Recall focuses on the identification of actual positive predictions. As defined:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Using our scenario, the recall calculation is:

$$\text{Recall} = \frac{50}{50 + 20} = \frac{50}{70} \approx 0.71 \text{ or } 71\%$$

The results show that the recall is relatively high and the precision is low. This indicates that the model is capturing as many true positive cases as possible and accepting a higher number of false positives in its predictions.

Changing the Threshold

The decision threshold determines the probability or score needed to classify a case as positive. Changing this threshold impacts True Positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TN):

- Lowering the threshold makes it easier to classify a case as positive, generally increasing TP and FP, which can increase recall but decrease precision.
- Raising the threshold makes the criteria for classifying a case as positive stricter, usually decreasing TP and FP, potentially increasing precision but lowering recall.

Threshold Adjustment Impact On Precision and Recall

Consider the scenario where the decision threshold of a predictive model is modified either by lowering or raising it slightly. This section explores the effects of such adjustments on the screening criteria and their subsequent outcomes:

Scenario 1 (Lowering the Threshold): Decreasing the decision threshold results in the model identifying 10 additional true positives, increasing TP to 60, but it also introduces 40 more false positives, increasing FP increases to 190.

Scenario 2 (Raising the Threshold): Conversely, increasing the decision threshold leads to a reduction of 50 in false positives, decreasing FP to 100, but it also results in overlooking 10 true positive cases, increasing FN to 30.

Recalculating Metrics

Original Precision and Recall:

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{50}{50 + 150} = \frac{50}{200} = 0.25$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{50}{50 + 20} = \frac{50}{70} \approx 0.7143$$

After Lowering Threshold:

$$\text{New Precision} = \frac{60}{60 + 190} = \frac{60}{250} = 0.24$$

$$\text{New Recall} = \frac{60}{60 + 20} = \frac{60}{80} = 0.75$$

After Raising Threshold:

$$\text{New Precision} = \frac{50}{50 + 100} = \frac{50}{150} \approx 0.3333$$

$$\text{New Recall} = \frac{50}{50 + 30} = \frac{50}{80} = 0.625$$

After recalculating the metrics based on hypothetical adjustments to the decision threshold, a mathematical illustration of the precision-recall trade-off becomes evident. Lowering the threshold results in an increase in recall but a decrease in precision due to the inclusion of more false positives. Conversely, raising the threshold enhances precision but leads to missing more true positives, thereby decreasing recall. This interplay is crucial in scenarios such as disease screening, where it is essential to balance the costs associated with false positives and negatives.

F1 Score: A Comprehensive Evaluation Metric in Machine Learning

The F1 Score stands out as a comprehensive evaluation metric in classification tasks. It combines precision and recall, into a single, unified score by calculating their harmonic mean [12, 18]. This calculation helps balance the trade-offs between them, offering a more extensive view of a model's accuracy.

The Harmonic Mean and F1 Score: Definition and Mathematical Formulation

The F1 Score is formally defined as:

$$F1\ Score = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Calculating the F1 Score by using the harmonic mean tends to draw the result closer to the lower of the two measures of precision and recall. This means if one of these measures is significantly lower than the other, it will have a greater influence on the F1 Score. This approach highlights areas that require improvement by emphasizing the lower value. This characteristic makes it a conservative metric that ensures both high precision and high recall are necessary for a high F1 Score. Key aspects of this approach include:

- **Sensitivity to Low Values:** The harmonic mean emphasizes the lower of the two values of precision and recall, making it naturally inlcined towards the lesser of the two. If precision or recall is low, this significantly lowers the F1 Score. This approach is notably different from the arithmetic mean, which might produce a higher, potentially misleading score [16]. A 'misleading' score suggests that the arithmetic mean might make the performance of a model look better than it actually is by not adequately penalizing low values in either precision or recall.

- **Necessity for High Precision and Recall:** For a model to attain a high F1 Score, it must exhibit high precision and recall. The F1 Score is reduced if either metric is considerably lower, as the harmonic mean penalizes the lower value more heavily. This ensures a model is recognized for high performance only if it demonstrates balanced capabilities in both identifying positive instances accurately and capturing a comprehensive set of actual positives [15].
- **Promotion of Balanced Model Performance:** The emphasis on both precision and recall makes the F1 Score a valuable metric for scenarios that require a balanced approach to model evaluation. It motivates the optimization of models to excel in both precision and recall, fostering the development of more effective and balanced models [27].

Example: Disease Screening Scenario

Let's take a hypothetical scenario where doctors are diagnosing a particular disease using a new test.

- **True Positives (TP)**: 80 patients who actually have the disease and are correctly identified by the test.
- **False Positives (FP)**: 20 patients who do not have the disease but are incorrectly identified by the test as having it.
- **True Negatives (TN)**: 880 patients who do not have the disease and are correctly identified by the test as not having it.
- **False Negatives (FN)**: 20 patients who have the disease but are incorrectly identified by the test as not having it.

Precision

Precision measures the accuracy of positive predictions. It is the ratio of true positives to the sum of true and false positives.

$$Precision = \frac{TP}{TP + FP} = \frac{80}{80 + 20} = \frac{80}{100} = 0.8$$

This means that when the test predicts the disease, it is correct 80% of the time.

Recall

Recall measures the ability of the test to detect all positive cases. It is the ratio of true positives to the sum of true positives and false negatives.

$$Recall = \frac{TP}{TP + FN} = \frac{80}{80 + 20} = \frac{80}{100} = 0.8$$

This means that the test identifies 80% of all actual disease cases.

F1 Score

F1 Score is the harmonic mean of precision and recall, providing a balance between them.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} = 2 \times \frac{0.8 \times 0.8}{0.8 + 0.8} = 2 \times \frac{0.64}{1.6} = \frac{1.28}{1.6} = 0.8$$

The F1 Score here is also 0.8, showing a balanced performance between precision and recall.

F1 Score in Less Ideal Conditions

Now let's consider a different scenario for the same disease diagnosis, where the test outcomes are not as ideal in terms of precision and recall.

In this new scenario, the outcomes from the test are as follows:

- True Positives (TP) = 50
- False Positives (FP) = 150
- True Negatives (TN) = 750
- False Negatives (FN) = 21

Precision

Precision is lower in this scenario, indicating a higher rate of false positives:

$$Precision = \frac{TP}{TP + FP} = \frac{50}{50 + 150} = \frac{50}{200} = 0.25$$

Recall

Recall in this scenario, is higher than precision:

$$Recall = \frac{TP}{TP + FN} = \frac{50}{50 + 21} = \frac{50}{71} = 0.7$$

F1 Score

The F1 Score is affected by the significant difference of precision and recall:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} = 2 \times \frac{0.25 \times 0.7}{0.25 + 0.7} = 2 \times \frac{0.175}{0.95} = \frac{0.35}{0.95} = 0.368$$

The calculated F1 Score of 0.368 indicates a lower effectiveness of the test, primarily due to the significant gap between precision (0.25) and recall (0.7). This low score demonstrates the model's challenge in reducing incorrect positive predictions and increasing the accuracy of positive identifications. In essence, the low precision indicates that a substantial portion of positive predictions were incorrect, while the moderate recall suggests that the model correctly identified only 70% of the actual positives. By taking the harmonic mean of precision and recall, the F1 Score illlustrates the impact of this discrepancy more sharply than either metric alone. The result is a low F1 Score, which signals a critical need for model adjustment to improve either precision, recall, or ideally both, to enhance the model's overall predictive accuracy and reliability.

Limitations and Considerations in the Application of the F1 Score

The F1 score is a useful metric for evaluating the performance of classification models, especially in cases where the data is imbalanced. However, like all metrics, the F1 score has its limitations and considerations that need to be taken into account when applying it. Here are some of the key ones:

Imbalanced Classes Sensitivity

The F1 score is often used in scenarios with imbalanced classes because it balances the precision and recall. However, it might not always provide a clear picture of the model's performance in extreme cases of class imbalance [29, 14].

In cases with extremely imbalanced datasets, a model might achieve high precision by making very few positive predictions, most of which are correct, but this comes at the expense of a low recall, as it fails to identify a majority of positive instances. Conversely, strategies aimed at boosting recall can lead to a higher rate of false positives, diminishing precision. Therefore, although the F1 score is a balanced metric combining precision and recall, it might not completely reflect how extreme class imbalance affects the model differently when it comes to false positives and false negatives.

This is due to the averaging nature of the F1 score, which can sometimes mask biases in the model against the minority class. A model might perform moderately well on the majority class but poorly on the minority

class. The F1 score, especially if averaged across classes in a multi-class setting, might not reveal this disparity. This can lead to misleading interpretations of model improvement. Even a slight improvement in the handling of the minority class can lead to a significant increase in the F1 score. This might give the impression of substantial improvement in model performance, whereas, in practical terms, the model might still fail to adequately address the imbalance

Not Sensitive to True Negatives

The F1 score considers the positive class predictions, such as true positives and false positives, and actual positives, such as true positives and false negatives.

As defined:

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{1}$$

Noticeably absent from this formula are True Negatives. This absence means the F1 score doesn't take into account the model's ability to identify negative instances correctly. In certain applications, particularly where negative instances are significant, this might not provide a full picture of the model's performance. [20]

For example, in a medical screening test, false negatives indicate a failure to identify a disease when it is present, and false positives indicate an incorrect classification of a disease when it is absent. True negatives entail correctly identifying the absence of a disease. The F1 score's focus on positive predictions and actual positives means it might not fully represent the model's performance in scenarios where negative predictions are equally important.

In such cases, other metrics like specificity, which measures the proportion of actual negatives that are correctly identified as such, might be used alongside the F1 score to give a more complete picture of the model's performance.

Difficulty in Interpreting Across Different Domains

The F1 score's interpretation is significantly influenced by both contextual and methodological constraints. The significance and value of the F1 Score may vary depending upon each application's unique balance of the consequences or severity of false positives and false negatives. For instance, in spam detection, a high F1 score is acceptable even with some false negatives, as the cost of misclassification is relatively low. However, in medical diagnostics, the same F1 score might be deemed insufficient due to the grave consequences of missing a positive diagnosis.

Moreover, the F1 score presupposes that precision and recall are equally important, a principle not universally applicable across all scenarios. In contexts like medical diagnoses, the harm from failing to identify a disease (a high false negative rate or low recall) vastly overshadows the repercussions of false positives. This fundamental bias of the F1 score illuminates the critical need for flexible metrics that can be fine-tuned to prioritize precision or recall as needed, accommodating the varied importance these factors hold in different contexts. Such adaptability is crucial for a more accurate assessment of model performance, given the diverse requirements and implications of errors in various applications.

Threshold Dependency

Models output a probability score indicating the likelihood of an instance being positive, and it's the adjustable threshold value that converts these scores into class labels—positive or negative. Model metrics like precision, recall, and the F1 score are highly dependent on the probability threshold. Changing this threshold has significant effects: lowering it boosts recall by capturing more true positives at the cost of precision due to increased false positives, while raising it enhances precision by limiting false positives but may lower recall by missing true positives. The F1 score, a harmonic mean of precision and recall, reflects the balance achieved by adjusting this threshold according to task-specific needs.

This threshold-dependent nature of precision, recall, and the F1 score illustrates the importance of careful threshold selection. The optimal threshold is one that aligns with the specific objectives of the model, whether it prioritizes minimizing false positives (precision), minimizing false negatives (recall), or seeks a balance between the two. Moreover, the choice of threshold can heavily influence the perceived performance of the model. An appropriately chosen threshold can reveal a model's effectiveness in distinguishing between classes under particular operational criteria, while an ill-chosen threshold can mask the model's true capabilities or shortcomings.

In practice, the process of determining the optimal threshold to maximize the F1 score often involves empirical methods, utilizing validation data to select a threshold that offers the best performance on this subset. However, this approach is fraught with challenges. The empirical selection of thresholds is notably susceptible to the "winner's curse," where the chosen threshold, while optimal on the validation set, may not perform as well in broader applications due to sampling variability and limited data [19]. This phenomenon is particularly pronounced in the optimization of the F1 score due to its non-linear nature and the asymmetric treatment of positive and negative class labels.

Consequently, evaluating model performance involves not only examining precision, recall, and the F1 score at a single threshold but also considering how these metrics change across a range of thresholds. This approach provides a more comprehensive understanding of model behavior and effectiveness, facilitating the identification of a threshold that best serves the specific needs and constraints of the application at hand.

Challenges in Multi-Class Classification Settings

Transitioning from binary classification, where outcomes are predicted as either positive or negative, to multi-class classification scenarios, characterized by more than two possible outcomes, complicates the direct application of the F1 score. The standard formulation of the F1 score is tailored for binary contexts, predicated on the binary distinction of "positive" and "negative" classes for the computation of precision and recall. This binary framework does not readily apply to multi-class contexts, wherein multiple categories are considered "positive."

To accommodate multi-class classification, the F1 score can be adapted using various averaging methods, each with distinct implications for class imbalance [26, 1]:

- **Macro-Averaging**: Computes the F1 score for each class independently before calculating the average. This method assumes equal significance across classes, potentially misleading in the presence of class imbalance, as it does not reflect class proportions.
- **Micro-Averaging**: Aggregates the contributions of all classes to derive the average F1 score, weighting each instance equally. This method is preferable for datasets with class imbalance, as it incorporates the frequency of each class.
- **Weighted Averaging**: Similar to macro-averaging, but each class's F1 score is weighted by the class's prevalence before averaging. This approach accounts for class imbalance by weighting the scores according to the relative importance or prevalence of each class.

Adapting the F1 score for multi-class classification introduces interpretative complexities. While the F1 score in binary classification clearly indicates the balance between precision and recall, its extension to multi-class settings—dependent on the chosen averaging method (macro, micro, or weighted)—necessitates a robust understanding of these methods' biases and how they handle class imbalance.

F-beta Score

The *F-beta Score* introduces a way to weigh precision and recall differently, using a parameter called β. This results in the weighted harmonic mean of the precision and recall, which reaches its optimal value at 1 and worst value at 0 [23]. The β parameter allows for the adjustment the importance of recall relative to precision in the evaluation metric. The formula for the F-beta Score is defined as follows:

$$F_\beta = \frac{(1+\beta^2) \cdot \text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

When $\beta = 1$, the F-beta Score becomes the F1 Score, treating precision and recall as equally important. When $\beta < 1$, precision is given more weight than recall. This setting is useful in scenarios where false positives are more costly than false negatives. Conversely, when $\beta > 1$, recall is considered more important, which is valuable in situations where missing a positive instance (false negative) carries a higher cost than mistakenly labeling a negative instance as positive (false positive).

Practical Implications

The flexibility of the F-beta Score makes it an invaluable tool for tailoring model evaluation to specific application needs. For example, in medical diagnostics, a high recall might be prioritized to ensure as few cases of a disease are missed as possible, even at the expense of some false positives. Conversely, in email spam detection, precision might be more critical because falsely categorizing important emails as spam could be more disruptive to users than occasionally missing a spam email.

By adjusting the β parameter, developers and researchers can fine-tune their evaluation metrics to better reflect the trade-offs and priorities inherent in their specific applications, leading to more comprehensive and appropriate assessments of model performance.

Specificity Metric

Specificity is a valuable performance metric, particularly when the cost of false positives is high. It is defined as the proportion of true negatives (TN) out of the total actual negatives (TN + FP), where FP denotes false positives [3, 17]. In essence, specificity measures a model's ability to correctly identify negative instances as negative. This metric is pivotal in scenarios where the absence of a condition is as significant as its presence.

Mathematically, specificity is defined as:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

A higher specificity value indicates that the model is highly effective at identifying negative cases as such, reducing the likelihood of falsely categorizing an instance as positive. This is particularly vital in fields like healthcare, where falsely diagnosing a healthy patient with a disease (a false positive) could lead to unnecessary stress, further testing, and treatment. Similarly, in fraud detection systems, specificity helps minimize the number of legitimate transactions flagged as fraudulent, ensuring a smooth user experience while safeguarding against actual fraud.

Limitations and Considerations

Specificity, like precision and recall, has its own set of limitations and considerations, especially when used in isolation to evaluate the performance of classification models. Here are some key points to consider:

- **Imbalance Sensitivity:** Specificity may not provide a complete picture in imbalanced datasets where the number of negative instances significantly outweighs the positive ones.
- **Trade-off with Sensitivity:** Specificity has an intrinsic trade-off with sensitivity (also known as recall).

While specificity is an integral metric for assessing the performance of classification models, it should not be considered in isolation. Relying solely on specificity can lead to a misleading evaluation of a model's overall effectiveness. This is because specificity does not account for the model's ability to correctly identify positive instances, nor does it reflect the precision or the balance between precision and recall, such as the F1 score. Additionally, the significance of specificity must be contextualized within the specific application it is being used for, as its importance varies depending on the relative costs of false positives and false negatives in different scenarios.

ROC & AUC

The Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC) are essential tools for evaluating the performance of classification models. These metrics are crucial for measuring a model's ability to accurately distinguish between classes, guiding the selection of optimal decision thresholds that strike a balance between sensitivity (true positive rate; Recall) and specificity (true negative rate). Achieving this balance is vital to ensure models perform effectively and align with the practical requirements and constraints of various applications, ranging from healthcare, where they might predict the presence of disease in patients, to manufacturing, where they forecast maintenance needs. Fundamentally, ROC and AUC are indispensable in any binary classification scenario, serving to differentiate between two distinct classes based on predictive modeling

Introduciton to ROC

The ROC curve is a graphical representation used to assess the performance of classification models at various threshold settings. This curve primarily used to evaluate the general diagnostic effectiveness of a test and allows for the comparison between the effectiveness of multiple diagnostic tests [21]. In classification, it is constructed by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings.

True Positive Rate (TPR)

TPR, also known as sensitivity (recall), measures the proportion of actual positives that are correctly identified by the model [9]. It is calculated as:

$$\text{TPR} = \frac{TP}{TP + FN},$$

where TP represents the number of true positives and FN represents the number of false negatives.

False Positive Rate (FPR)

FPR measures the proportion of actual negatives that are incorrectly classified as positive by the model [9]. It is calculated as:

$$\text{FPR} = \frac{FP}{FP + TN},$$

where FP is the number of false positives and TN is the number of true negatives.

ROC Curve

The ROC curve is plotted by placing the FPR on the X-axis and the TPR on the Y-axis, corresponding to various threshold levels. As the threshold that distinguishes between positive and negative classifications changes, the FPR and TPR also shift. This dynamic adjustment traces the ROC curve, providing a clear visual representation of the trade-off between the advantages of correctly identifying true positives and the disadvantages of inaccurately identifying false positives.

Key Points about the ROC Curve:

Interpretation: The closer the ROC curve is to the top left corner, the higher the overall accuracy of the model at distinguishing between the positive and negative classes. A curve that falls below the diagonal line indicates a model performing worse than random guessing, which is typically seen as a model with serious issues.

Threshold Selection: The ROC curve can also help in selecting an optimal threshold for classification, which balances the trade-off between the True Positive Rate and False Positive Rate according to the specific requirements of the application, such as prioritizing minimizing false positives over capturing as many positives as possible.

ROC Curve Plot

Figure 2 illustrates the ROC curve representing the relationship between the FPR on the x-axis and the TPR on the y-axis, providing insight into the classifier's performance across varying thresholds. The orange diagonal line in ROC curve plots acts a baseline for comparison, representing the performance of a random classifier. Extending from the bottom left to the top right, it depicts random guessing with an AUC of 0.5, exhibiting no ability to differentiate between classes. A classifier is considered effective if its ROC curve lies above this line, indicating better than random performance. The line serves as a visual guide to quickly gauge a classifier's discriminative power, where a curve nearer to the top left corner signifies higher accuracy. This plot was constructed using randomly generated data, and employing the scikit-learn library [22] for model training and evaluation, and matplotlib [13] for visualization. The ROC curve's close proximity to the top left corner, as evidenced in the figure, signifies the model's effective discrimination between the positive and negative classes.

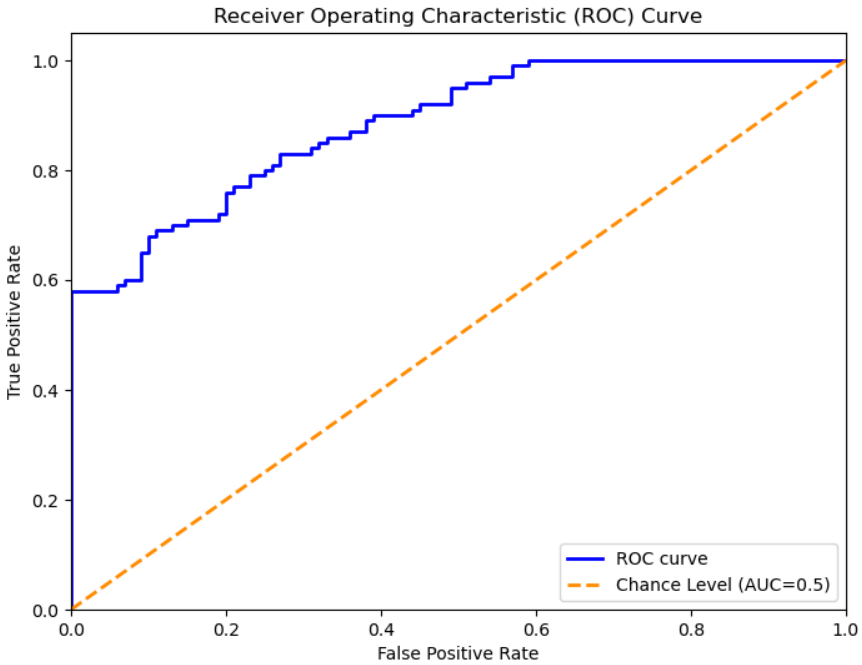


Figure 2: ROC curve analysis demonstrating the classifier's ability to distinguish between positive and negative classes. The curve was generated using scikit-learn with data visualized through matplotlib.

Introduction to AUC

AUC is a metric used to evaluate the performance of classification models. It measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1). It provides an aggregate measure of performance across all possible classification thresholds [9]. The value of AUC ranges from 0 to 1, where an AUC of 1 represents a perfect model that perfectly classifies all positives and negatives correctly. An AUC of 0.5 suggests a model that has no discriminative ability and performs no better than random guessing.

The AUC value gives you an aggregated measure of the model's ability to correctly classify the positive and negative cases across all possible thresholds. A higher AUC indicates a better model performance. For practical purposes, an AUC over 0.8 is usually considered acceptable [21], though this can vary based on the specific context and application. A key advantage of the AUC is that it is not affected by the proportion of class distribution. This makes it particularly useful for evaluating models on imbalanced datasets.

If the ROC curve were plotted on a perfectly continuous scale, the AUC could be found through integration.

However, in practical applications with discrete thresholds, the AUC is often calculated using the trapezoidal rule to approximate the area under the curve, summing the areas of trapezoids under the curve [25]. The trapezoidal rule is a numerical method for approximating the definite integral of a function [5]. It works by breaking down the area under a curve into a series of adjacent trapezoids, rather than rectangles as in the simpler Riemann sums approach. The area of each trapezoid is calculated, and then these areas are summed to approximate the total area under the curve. Specifically, for a function $f(x)$ that is defined at discrete intervals $[a, b, c, \ldots]$, the area of a single trapezoid between any two points x_i and x_{i+1} is given by:

$$\text{Area} = \frac{(f(x_i) + f(x_{i+1}))}{2} \cdot (x_{i+1} - x_i)$$

By applying this formula to each pair of adjacent points that define the trapezoids under the ROC curve and summing all these areas, we obtain an approximation for the AUC.

$$\text{AUC} = \sum_{i=0}^{n-1} \frac{(y_i + y_{i+1})}{2} \cdot (x_{i+1} - x_i)$$

This approach is particularly useful in machine learning for ROC curves, where the curve is constructed from a finite set of points representing different classification thresholds. The trapezoidal rule provides an efficient and accurate method for estimating the AUC, offering a quantitative measure of a model's ability to discriminate between classes across all possible thresholds and encapsulating the trade-off between sensitivity and specificity (1 - FPR).

Emphasizing 1-FPR over FPR in the context of AUC is imperative because it directly evaluates the model's precision in classifying negatives accurately, a capability as important as identifying positives. This focus on specificity, alongside sensitivity, demonstrates the a model's ability to avoid false alarms, enhancing its comparative evaluation.

Ultimately, AUC's significance lies in its capacity to gauge a model's performance across all thresholds, indicating a well-tuned balance between identifying true positives and correctly dismissing negatives. A higher AUC suggests a model's proficiency in both aspects, marking it as effective in situations requiring careful discrimination between classes.

ROC Curve Plot with AUC score

Figure 3 illustrates the ROC curve with the AUC distinctly shaded. This visualization demonstrates the performance of the classification model and quantifies it through the AUC score presented in the legend. The depicted ROC curve and AUC score offer a comprehensive measure of the model's ability to accurately predict between classes. This plot was generated using the Scikit-learn library [22] and Matplotlib [13] in Python, based on a dataset of randomly generated data. Such a graphical representation is instrumental in evaluating the model's diagnostic ability in a binary classification context.

Merits of AUC in Predictive Modeling

Performance Measurement: It gives you a single number that lets you compare models more easily, even when the distribution of classes in your test data changes.

Imbalanced Classes: AUC is especially useful in situations where you have imbalanced classes, as it is less sensitive to class imbalance than accuracy.

No Need to Set a Threshold: It evaluates the model's ability to discriminate between positive and negative classes without needing to set a specific threshold. This is useful because choosing an optimal threshold that works well for all circumstances is challenging. The AUC gives a summary of how well a model can separate positive from negative examples without relying on a fixed cut-off point. It measures whether the model is likely to rank a randomly chosen positive example higher than a randomly chosen negative one, across all possible cut-off points. It's particularly valuable in the early stages of model testing

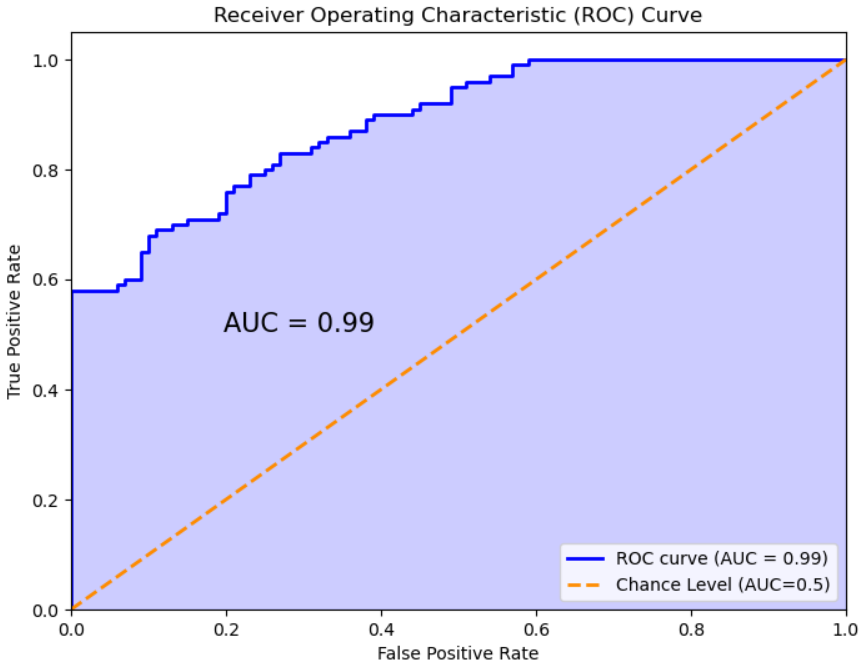


Figure 3: Receiver Operating Characteristic (ROC) curve with shaded Area Under the Curve (AUC).

or in situations where the cost of false positives versus false negatives varies, and an immediate choice of threshold might not be ideal.

However, in practice when actually using a model, choosing a specific threshold to turn probability scores into definite class labels (like "spam" or "not spam") is often necessary. Here, the ROC curve can guide the choice of a threshold that balances well between identifying true positives and avoiding false positives, according to what's most important for the task at hand. AUC is a valuable, threshold-independent way to compare the performance of different models. Meanwhile, the ROC curve offers detailed insights into how choice of threshold affects model performance.

Generating ROC Curves and AUC Score

To construct the ROC curve, a series of methodical procedures is required, typically performed in a sequential order [6, 4]. It involves systematically varying the threshold for classifying observations and calculating the TPR and FPR at each threshold. The stages outlined below detail the process involved in constructing an ROC curve.

Stage 1: Initial Model Predictions

The classification model predicts a score or probability for each sample in the dataset, indicating the likelihood that each sample belongs to the positive class. These scores do not directly classify samples as belonging to a specific class, but rather indicate the confidence level of the model in its prediction that the sample is positive.

Stage 2: Thresholds Determination

The threshold values are not randomly selected but are typically derived from the prediction scores themselves. A common approach is to use each unique score from the model's predictions as a potential threshold. This method ensures that every possible cutoff is evaluated, providing a comprehensive assessment of the model's performance across all thresholds.

The thresholds are then sorted based on their prediction scores, with the highest scores first. It is also common practice to include a threshold slightly below the lowest score (0) and slightly above the highest score (1.0). This inclusion ensures that the ROC curve starts at (0,0) and ends at (1,1), encapsulating the full spectrum of possible FPR and TPR values.

The quantity of thresholds selected is dependent on the unique probabilities produced by the model's predictions. A large dataset with finely grained model scores (probabilities) could result in a significant number of unique thresholds.

Stage 3: Calculating TPR and FPR at Each Threshold

For each threshold chosen in Stage 2, the dataset is divided based on whether the model's score for an observation is above or below the threshold. Observations with scores above the threshold are classified as positive, and those below as negative. This classification is then compared to the actual labels to calculate the TPR and FPR at each threshold. By evaluating each unique prediction probability as a potential threshold, from highest to lowest, it's possible to simulate how adjusting the thresholds impacts the classification outcomes.

Stage 4: Plotting the ROC Curve

The TPR is plotted on the y-axis, and the FPR is plotted on the x-axis for each threshold. When plotting the ROC curve, primary procedure is to connect the points in the order of descending thresholds. This starts from the most conservative (Highest) classification threshold, which would classify most instances as negative, down to the most liberal (Lowest), which would classify most instances as positive. This method ensures the curve begins at the bottom left corner of the plot. Which represents a very high threshold where both the TPR and FPR are near zero.The curve then ends at the top right corner, which represents a very low threshold where both rates are near one.

Stage 5: Model Evaluation with AUC

The AUC score of the ROC plot can then be calculated to provide a single measure of the model's performance across all thresholds. A higher AUC indicates a better model performance.

Efficiency Considerations

In practice, to generate an ROC curve and AUC score, the model itself does not need to be retrained multiple times with different thresholds. Instead, the model is trained once, and its prediction scores for the validation or test set are used to evaluate the TPR and FPR at various thresholds post-prediction. This approach allows for efficient evaluation of the model's performance at distinguishing between classes without the need for multiple model executions.

Conclusion

In conclusion, this paper has presented a brief survey of several essential evaluation metrics used in classification models, namely precision, recall, error rate, specificity, F1 Score, ROC curve, and AUC. Each metric offers a unique perspective through which the performance of a predictive model can be assessed, offering different aspects of prediction quality and class imbalances inherent in real-world datasets. Precision and recall focus on the model's accuracy in predicting positive cases, demonstrating the trade-off between capturing relevant instances and the proportion of accurate positive predictions. The error rate provides a straightforward measure of the model's overall error, while specificity addresses the model's ability to correctly identify negative cases.

The F1 Score is calculated as the harmonic mean of precision and recall. This method of calculation ensures that the F1 Score provides a balanced measure, making it especially valuable in situations where both precision and recall are equally important. It helps to understand how well the model performs in identifying true positive cases while minimizing false positives and false negatives. The ROC curve and AUC expand the methods for evaluating model performance by showing how the model operates under different

threshold settings. This provides a clearer view of the model's capacity to distinguish accurately between classes.

Understanding and appropriately applying these metrics is vital for the development, tuning, and comparison of machine learning models. Each metric illustrates differant aspects of model performance and, therefore, must be chosen with consideration of the specific tasks and objectives of the model being evaluated.

A Appendix A: Practical Implementation of Classification Evaluation Metrics in Machine Learning

[1]:
```
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import NearestNeighbors
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

[2]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,␣
↪roc_auc_score, roc_curve, auc, confusion_matrix
```

[3]:
```
#Read the data from a csv file onto a pandas dataframe
df=pd.read_csv("data/diabetes.csv")
df
```

[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
..	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1

```
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...   ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[5]:
```
df.describe()
```

[5]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

It's important to check for null or missing values in the data, as that can affect the performance of a model.

[6]:
```
df.isnull().sum()
```

[6]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Using the isnull() method showed no nulls in the dataset, however it is evident that there 0.0 values in some of the columns. In two of the columns it is appropriate to have 0, as this is relevant to the representation of the data. For example in the pregnancy columns, a 0 indicates that the patient has had no pregnancies. Also the outcome 0 indicates the class which determines that a person does not have diabetes.

Knowing that the columns for 'Pregnancies' and 'Outcome' have appropriate zeros, we can get convert the zeros on all other columns in to NaN values, this will potentially make further manipulation and data processing far simpler. First we can grab the names for the other columns.

[7]:
```
cols_valid_zeros = ['Pregnancies','Outcome']
cols_null_zeros = df.columns[(df == 0).any() & ~df.columns.isin(cols_valid_zeros)]
display(cols_null_zeros)
```

```
Index(['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI'], dtype='object')
```

The following function goes through the dataframe and looks at the columns specified in cols, then replaces any 0 values to Nan values

[8]:
```
def nanConvert(data_df,cols):
    for c in cols:
        df[c] = df[c].replace(0, np.nan)
    return df
```

[9]:
```
df = nanConvert(df,cols_null_zeros)
df
```

[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148.0	72.0	35.0	NaN	33.6	
1	1	85.0	66.0	29.0	NaN	26.6	
2	8	183.0	64.0	NaN	NaN	23.3	
3	1	89.0	66.0	23.0	94.0	28.1	
4	0	137.0	40.0	35.0	168.0	43.1	
..	...	...	...	...	...	...	...
763	10	101.0	76.0	48.0	180.0	32.9	
764	2	122.0	70.0	27.0	NaN	36.8	
765	5	121.0	72.0	23.0	112.0	26.2	
766	1	126.0	60.0	NaN	NaN	30.1	
767	1	93.0	70.0	31.0	NaN	30.4	

	DiabetesPedigreeFunction	Age	Outcome

```
0                        0.627   50        1
1                        0.351   31        0
2                        0.672   32        1
3                        0.167   21        0
4                        2.288   33        1
..                         ...   ...      ...
763                      0.171   63        0
764                      0.340   27        0
765                      0.245   30        0
766                      0.349   47        1
767                      0.315   23        0

[768 rows x 9 columns]
```

Once we replaced all 0 values with NaN, we can do some data imputations to fill the NaNs.

Mean/Median/Mode Imputation: This is the simplest form of imputation. Missing values are filled with the mean, median, or mode (most frequent value) of the column. Mean is used for continuous data, while median is more robust to outliers. Mode is used for categorical data. It's the simplest form of imputation, requiring minimal computation, but doesn't account for any correlations between features.

- Complexity: Very Low
- Overhead: Very Low

Random Imputation: Missing values are filled with a random observation from the same column. This method preserves the distribution but can add randomness to the dataset, which might not always be desirable. Slightly more complex due to the randomness but still very straightforward and fast.

- Complexity: Low
- Overhead: Low

Last Observation Carried Forward (LOCF) & Next Observation Carried Backward (NOCB): Common in time series data, LOCF fills missing values with the last observed value, and NOCB fills them with the next observed value. These methods assume the data points are closely related in time.

- Complexity: Low
- Overhead: Low

Mainly used for time series data. The complexity is still low, but the method is slightly more sophisticated than mean/median/mode because it considers the order of data.

Linear Interpolation: Another technique used in time series data, where missing values are filled based on linear interpolation between observed values. It's suitable for data with a linear trend.

- Complexity: Moderate
- Overhead: Low to Moderate

For time series data, this method considers the trend between observations. It's more complex but still relatively fast.

K-Nearest Neighbors (KNN) Imputation: This method fills missing values based on the K-nearest neighbors. It calculates the similarity between observations and uses the mean or median of the K most similar observations for imputation.

- Complexity: High
- Overhead: High

KNN requires calculating distances between observations, which can be computationally intensive, especially for large datasets.

Regression Imputation: Missing values are estimated using a regression model. The model is built using the observed values, and the missing values are filled based on the predictions of this model.

- Complexity: High
- Overhead: High

Involves building regression models for each feature with missing values, which can be computationally intensive, especially with many features or complex models.

Multiple Imputation: It involves creating multiple imputations (sets of values) for missing data points. The analysis is performed on each set, and the results are pooled to get final estimates. It accounts for the uncertainty in the imputations.

- Complexity: Very High
- Overhead: Very High

Creating multiple imputations for missing data and then analyzing each to pool results significantly increases complexity and computational load.

Hot-Deck Imputation: A randomly selected value from an individual in the dataset who has similar values on other variables is used to fill in the missing value. It's similar to KNN but doesn't explicitly use a distance metric.

H-Deck Imputation

- Complexity: Moderate
- Overhead: Moderate

This method requires matching profiles within a dataset, increasing complexity and computational time

Cold-Deck Imputation: Similar to hot-deck, but the value to fill in the missing data is taken from another dataset that is believed to be similar to the one with missing data.

- Complexity: Moderate
- Overhead: Moderate

Similar to hot-deck, but potentially requires accessing and processing external datasets, which can add to the complexity.

Imputation Using Deep Learning: Advanced methods like autoencoders can be used for imputation, where the model learns to predict missing values based on patterns learned from the data.

- Complexity: Very High
- Overhead: Very High

Designing, training, and using deep learning models for imputation is the most complex and computationally intensive, especially for large datasets and complex models.ationally intensive, especially for large datasets and complex models.r large datasets and complex models.

Due to it's low complexity and low overhead, we will use the mean imputations

B SimpleImputer

class sklearn.impute.SimpleImputer(*, missing_values=nan, strategy='mean', fill_value=None, copy=True, add_indicator=False, keep_empty_features=False)

Univariate imputer for completing missing values with simple strategies.

Replace missing values using a descriptive statistic (e.g. mean, median, or most frequent) along each column, or using a constant value.

Source: Pedregosa, F., Varoquaux, G., Gramfort, A., et al. "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, vol. 12, 2011, pp. 2825–2830.

Typically, the data should be split into training and testing sets before any imputation is performed. This is crucial for preventing data leakage. Data leakage occurs when information from the test set is inadvertently used to influence the training process, leading to overly optimistic performance estimates.

After splitting the data, the imputer should be fit on the training set only. This means calculating the imputation values based on the training data alone.mance.

[10]:
```
X = df.drop('Outcome', axis=1)  # Features
y = df['Outcome']
```

[11]:
```
display(X)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6    148.0           72.0           35.0      NaN  33.6
1              1     85.0           66.0           29.0      NaN  26.6
2              8    183.0           64.0            NaN      NaN  23.3
3              1     89.0           66.0           23.0     94.0  28.1
4              0    137.0           40.0           35.0    168.0  43.1
..           ...      ...            ...            ...      ...   ...
763           10    101.0           76.0           48.0    180.0  32.9
764            2    122.0           70.0           27.0      NaN  36.8
765            5    121.0           72.0           23.0    112.0  26.2
766            1    126.0           60.0            NaN      NaN  30.1
767            1     93.0           70.0           31.0      NaN  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

[12]:
```
display(y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Split the data into the training and testing sets.

[13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[14]:
```
# mean imputation using the SimpleImputer Class from Scikit Learn
imp_mean = SimpleImputer(strategy='mean')

# Fit on the training data and transform it
X_train_imputed = imp_mean.fit_transform(X_train)
X_train_imputed = pd.DataFrame(X_train_imputed, columns=X_train.columns, index=X_train.
↪index)

# Transform the test data based on the training fit
X_test_imputed = imp_mean.transform(X_test)
X_test_imputed = pd.DataFrame(X_test_imputed, columns=X_test.columns, index=X_test.
↪index)
```

Standard scaling should typically be done after imputation.

Since scaling involves adjusting the distribution of data based on its mean and standard deviati. If e da scaling is done a before imputing missing value then the model is e working with an incomplete dataset. The calculated mean and standard deviation would only reflect the observed (non-missing) data points, which might not represent the true distribution of the complete dataset.

If mean or median imputation is being used, the process will be influenced by the scale of the features. Imputing before scaling ensures that the imputation reflects the true scale of the data. For instance, imputing the mean of a feature that hasn't been scaled might be more meaningful, as the mean will be in the same units as the original dng data.

Scaling the features is crucial for some models, but not necessary for others.

The appropriateness of StandardScaler for different machine learning models depends on how each model handles feature scaling and the distribution of the features. For example:

Logistic Regression: Yes, feature scaling is beneficial because logistic regression coefficients can be influenced by the scale of the features, especially when using regularization.

Naive Bayes: Not typically necessary. Gaussian Naive Bayes is designed to handle data in its natural distribution since it assumes that the features follow a Gaussian distribution. However, for other types of Naive Bayes (e.g., MultinomialNB for count data), scaling is not applicable.

K-Nearest Neighbour (KNN): Yes, KNN is distance-based, meaning that the scale of the features directly impacts the calculation of distances between data points. Standard scaling is often crucial for models like KNN.

Decision Tree: No, decision trees and tree-based models (like Random Forests and gradient boosting models) do not require feature scaling. These models are not sensitive to the variance in the data and can handle varied feature scales.

Support Vector Machine (SVM) - Linear and RBF kernels: Yes, SVMs are sensitive to the scale of the data, especially with kernel methods (like RBF). Feature scaling can greatly influence the performance of SVMs by affecting the decision boundary.

Random Forest: No, similar to decision trees, Random Forest models are not sensitive to the scale of the features. Scaling does not generally impact their performance.

eXtreme Gradient Boost (XGBClassifier): No, XGBoost is also a tree-based model and is typically robust to the scale of the features. Feature scaling is not usually required.

For distance-based models (like Logistic Regression, KNN, and SVM), scaling is important because these models are sensitive to the scale of the features. For probability-based models (like Naive Bayes), scaling

might not be necessary and depends on the specific type of Naive Bayes used. For tree-based models (like Decision Trees, Random Forest, and XGBoost), scaling is generally not needed as these models are not sensitive to the feature scale.

[15]:
```
# Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imputed)
X_test_scaled = scaler.transform(X_test_imputed)
```

[16]:
```
models_scaled = []
models_scaled.append(("Logistic Regression", LogisticRegression()))
models_scaled.append(("K-Nearest Neighbour", KNeighborsClassifier(n_neighbors=3)))
# Enable probability estimates for SVM models
models_scaled.append(("Support Vector Machine-linear", SVC(kernel="linear", C=0.2,␣
↪probability=True)))
models_scaled.append(("Support Vector Machine-rbf", SVC(kernel="rbf",␣
↪probability=True)))
```

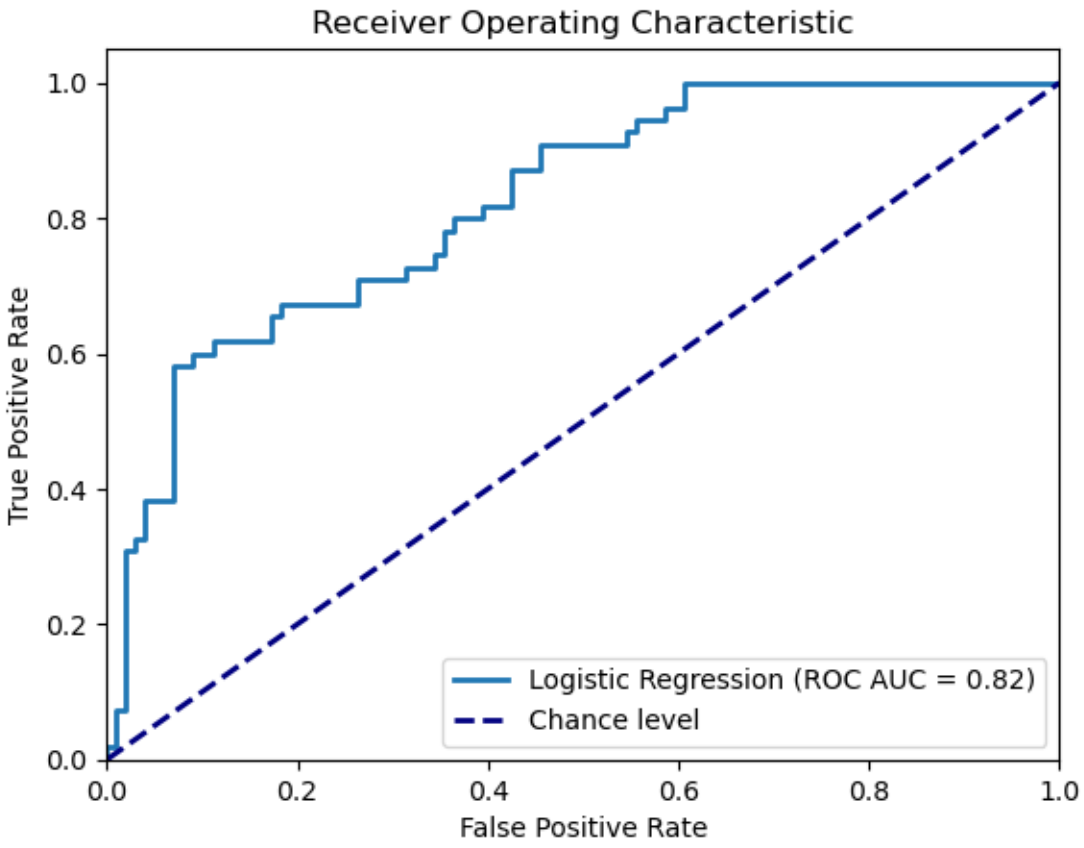
[17]:
```
models_unscaled = []
models_unscaled.append(("Naive Bayes", GaussianNB()))
models_unscaled.append(("Decision Tree", DecisionTreeClassifier()))
models_unscaled.append(("Random Forest", RandomForestClassifier(n_estimators=5)))
models_unscaled.append(("eXtreme Gradient Boost", XGBClassifier()))
```

[18]:
```
import matplotlib.pyplot as plt

def evaluate(model, name, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, y_proba)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, lw=2, label=f'{name} (ROC AUC = {roc_auc:.2f})')
    else:
        print(f"{name} does not support predict_proba. ROC AUC cannot be calculated.")
        roc_auc = "N/A"

    # Calculate evaluation metrics
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    accuracy = accuracy_score(y_test, y_pred)
    Error = (fp + fn)/(tn + fp + fn + tp)
    precision = precision_score(y_test, y_pred, average='binary')
    recall = recall_score(y_test, y_pred, average='binary')
    specificity = tn / (tn + fp)
    f1 = f1_score(y_test, y_pred, average='binary')
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()


    print("-"*30)
    print(f"Model: {name}")
    print(f"Accuracy: {accuracy*100:.2f}%")
    print(f"Error: {Error*100:.2f}%")
```

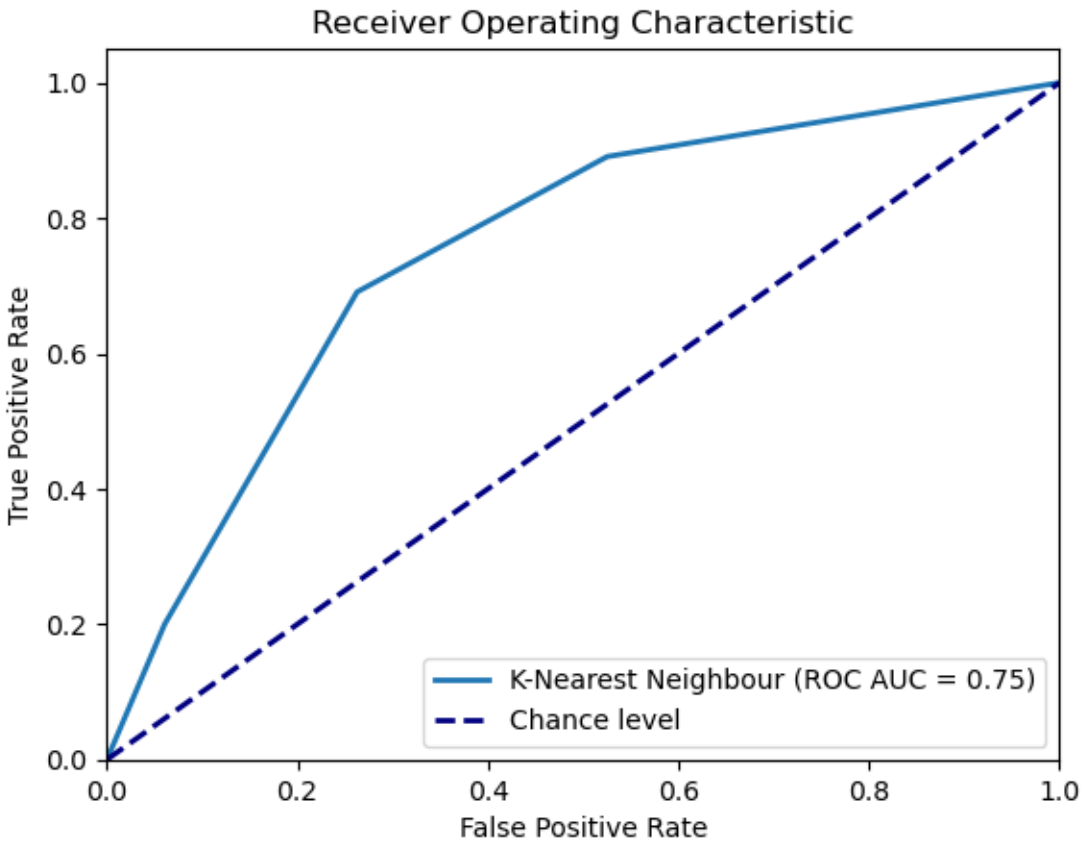
```
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"Specificity: {specificity:.2f}")
    print(f"F1 Score: {f1:.2f}")
    print(f"ROC AUC: {roc_auc}" if roc_auc != "N/A" else "ROC AUC: N/A")
    print("-"*30)

    if roc_auc != "N/A":
        plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Chance␣
↪level')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver Operating Characteristic')
        plt.legend(loc="lower right")
        plt.show()
```

[19]:
```
# Iterate over each model, train, and evaluate it
for name, model in models_scaled:
    evaluate(model,name,X_train_scaled,y_train,X_test_scaled,y_test)
```

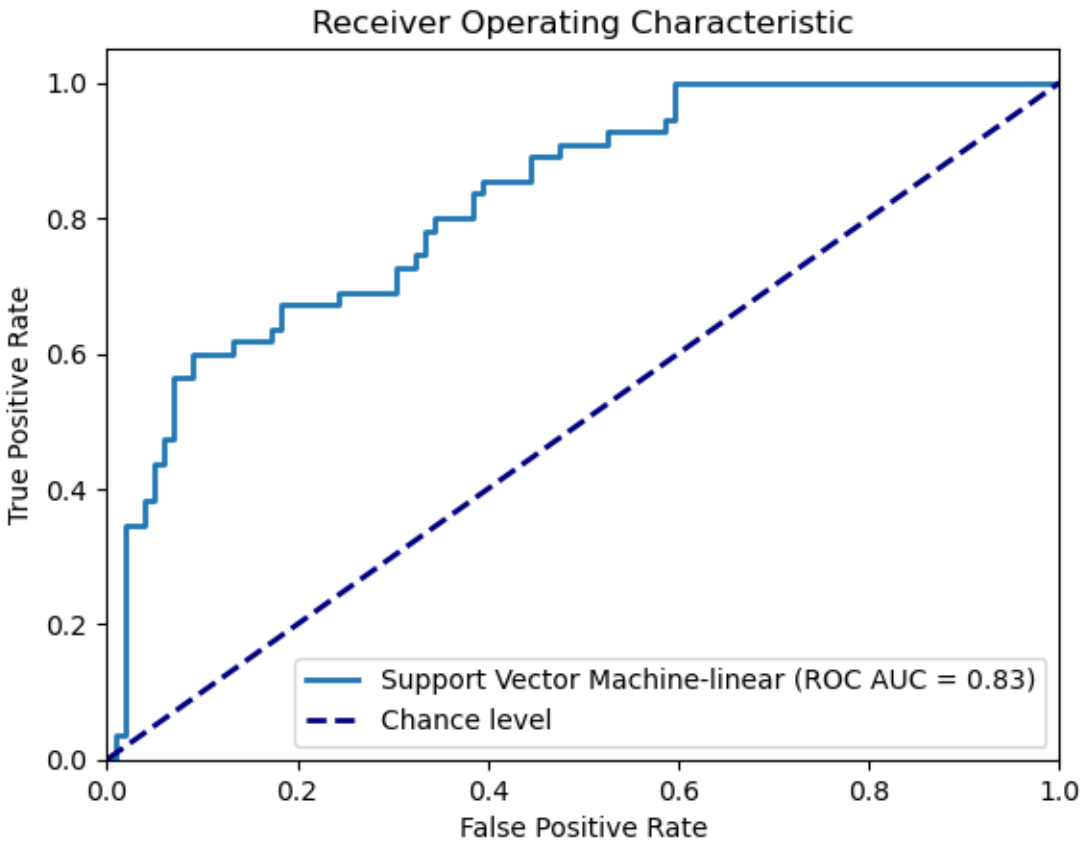
```
------------------------------
Model: Logistic Regression
Accuracy: 75.32%
Error: 24.68%
Precision: 0.67
Recall: 0.62
Specificity: 0.83
F1 Score: 0.64
ROC AUC: 0.8240587695133149
------------------------------
```



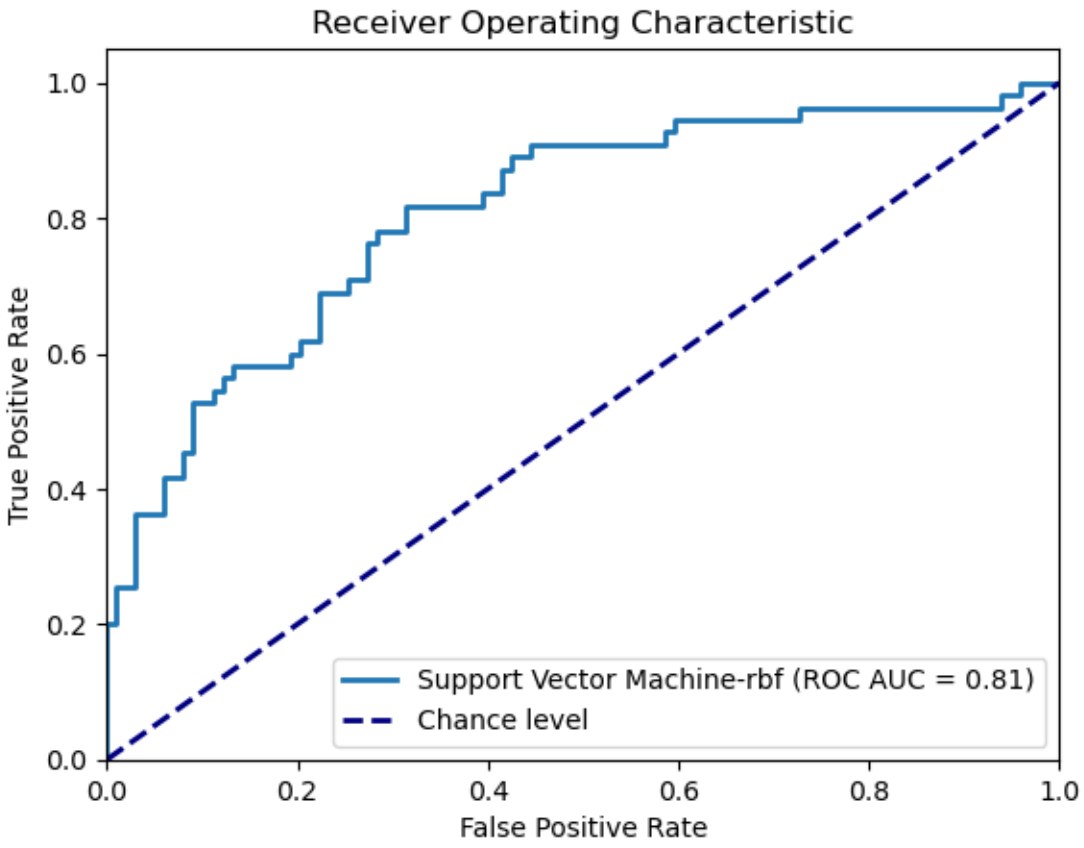

```
------------------------------
Model: K-Nearest Neighbour
Accuracy: 72.08%
Error: 27.92%
Precision: 0.59
Recall: 0.69
Specificity: 0.74
F1 Score: 0.64
ROC AUC: 0.7526170798898071
------------------------------
```




```
------------------------------
Model: Support Vector Machine-linear
Accuracy: 75.32%
Error: 24.68%
Precision: 0.67
Recall: 0.62
Specificity: 0.83
F1 Score: 0.64
ROC AUC: 0.8268135904499541
------------------------------
```

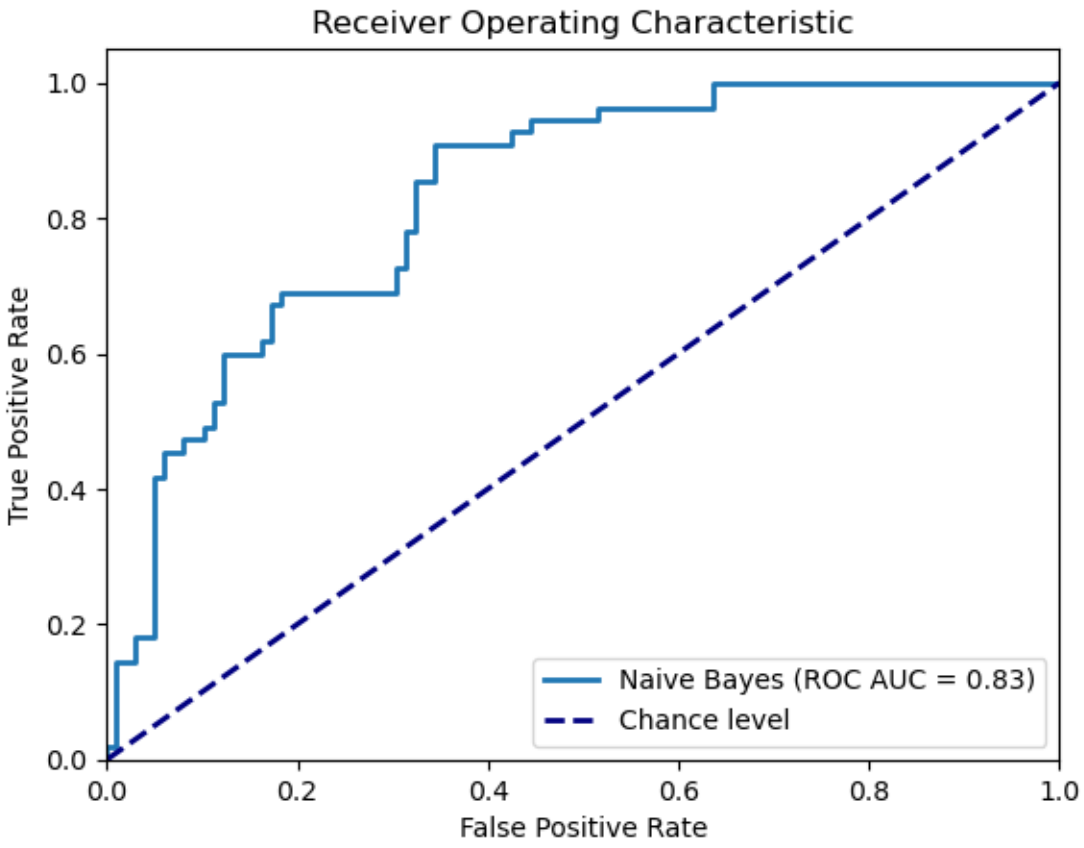



```
------------------------------
Model: Support Vector Machine-rbf
Accuracy: 75.32%
Error: 24.68%
Precision: 0.68
Recall: 0.58
Specificity: 0.85
F1 Score: 0.63
ROC AUC: 0.8101010101010101
------------------------------
```

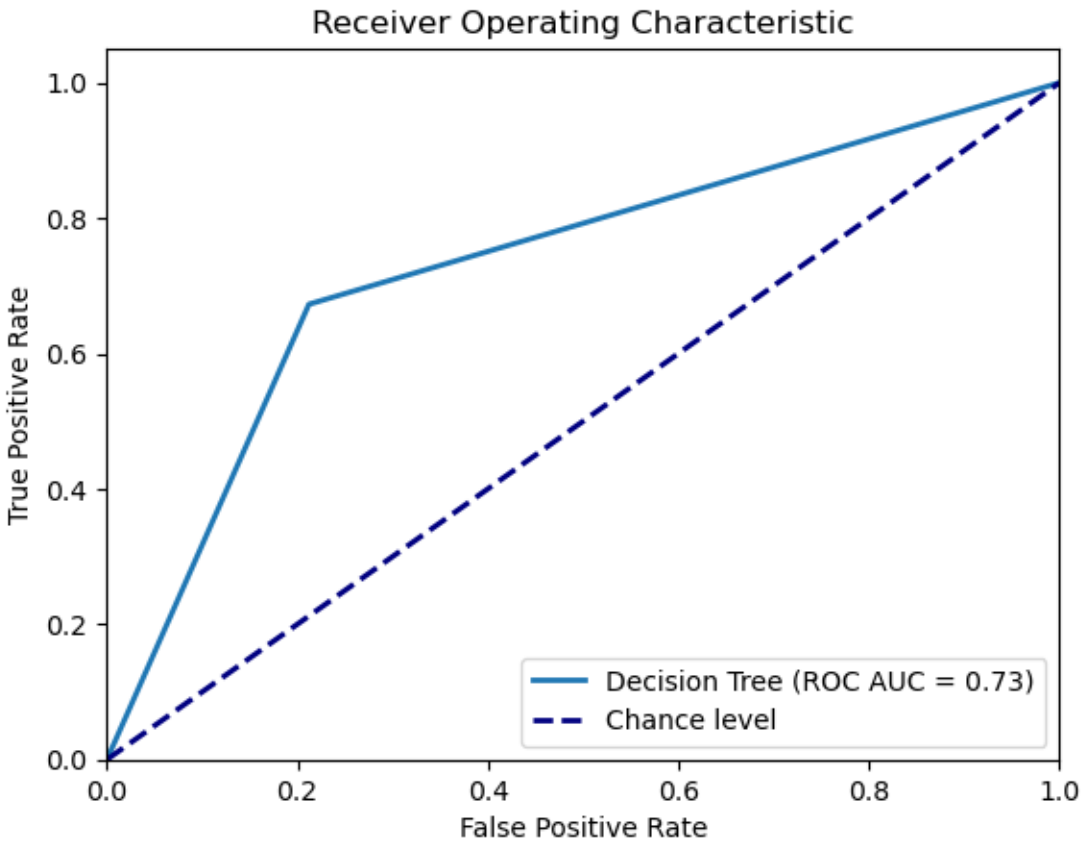


[20]:
```
# Iterate over each model, train, and evaluate it
for name, model in models_unscaled:
    evaluate(model,name,X_train_imputed,y_train,X_test_imputed,y_test)
```

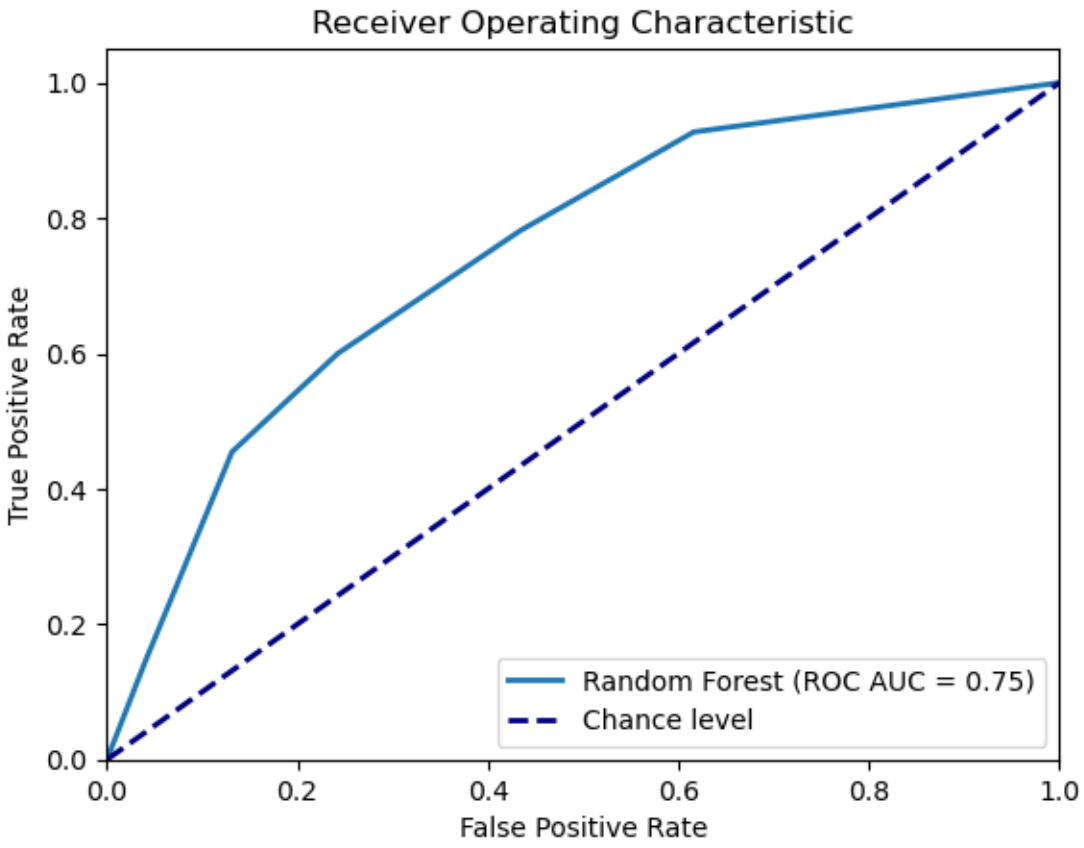
```
------------------------------
Model: Naive Bayes
Accuracy: 75.32%
Error: 24.68%
Precision: 0.64
Recall: 0.69
Specificity: 0.79
F1 Score: 0.67
ROC AUC: 0.8330578512396694
------------------------------
```



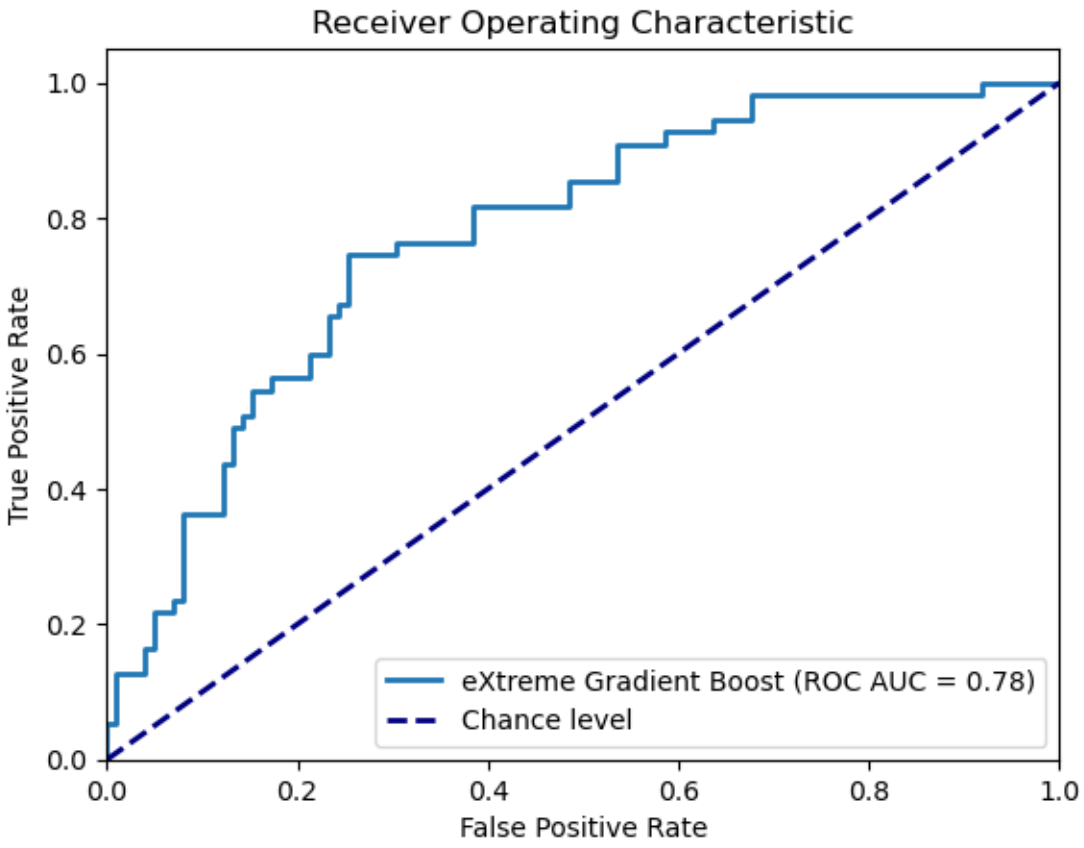

```
------------------------------
Model: Decision Tree
Accuracy: 73.38%
Error: 26.62%
Precision: 0.62
Recall: 0.65
Specificity: 0.78
F1 Score: 0.64
ROC AUC: 0.7161616161616162
------------------------------
```




```
------------------------------
Model: Random Forest
Accuracy: 72.73%
Error: 27.27%
Precision: 0.62
Recall: 0.62
Specificity: 0.79
F1 Score: 0.62
ROC AUC: 0.7505968778696052
------------------------------
```




```
------------------------------
Model: eXtreme Gradient Boost
Accuracy: 72.08%
Error: 27.92%
Precision: 0.60
Recall: 0.67
Specificity: 0.75
F1 Score: 0.63
ROC AUC: 0.7763085399449036
------------------------------
```



B.1 Model Evaluation: Logistic Regression

Let's evaluate the logistic regression model, as its scores for most of its metrics were relatively high.

- **Accuracy (75.32%)**: Accuracy is the proportion of all predictions (both diabetic and non-diabetic) that the model got right. With an accuracy of 75.32%, it means that about 75 out of every 100 predictions accurately reflect the actual condition of the individuals being tested, encompassing both TP (correctly identified diabetic cases) and TN (correctly identified non-diabetic cases).
- **Error (24.68%)**: The error rate represents the proportion of all predictions that were incorrect. An error rate of 24.68% means that nearly 25 out of every 100 predictions are wrong, which includes both FP (individuals incorrectly identified as diabetic) and FN (individuals with diabetes who were not correctly identified).
- **Precision (0.67)**: Precision focuses on the positive (diabetic) predictions and measures the fraction that was correct. A precision of 0.67 indicates that out of every 100 individuals the model labels as diabetic, 67 truly have diabetes (TP), while 33 do not (FP). This metric is critical in healthcare to minimize the psychological and financial impact of incorrect diagnoses.
- **Recall (0.62)**: Recall, or sensitivity, assesses the model's ability to correctly identify actual diabetic cases. A recall of 0.62 means the model successfully detects 62% of all real diabetic cases (TP) but misses 38% (FN). High recall is vital to ensure that most patients with diabetes receive the necessary care and intervention.
- **Specificity (0.83)**: Specificity measures the proportion of actual non-diabetic cases correctly identified as such. With a specificity of 0.83, this means 83% of non-diabetic individuals are correctly recognized

(TN), reducing the risk of 17% being incorrectly labeled as diabetic (FP). High specificity is crucial to prevent unnecessary stress and medical procedures for those without the condition.

- **F1 Score (0.64)**: The F1 Score is a balance between precision and recall, useful when you need a single metric to evaluate the model's overall performance in identifying positive cases correctly. An F1 Score of 0.64 suggests a moderate balance, indicating the model is reasonably effective but with room for improvement in both precision and recall.
- **ROC AUC (0.82)**: The ROC curve illustrates the model's ability to differentiate between diabetic and non-diabetic individuals at various threshold settings, and the AUC represents the model's discrimination capability. An AUC of 0.82 is considered good, implying the model has a strong ability to distinguish between individuals with and without diabetes across different decision thresholds.

References

[1] sklearn.metrics.f1_score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html. Accessed: 2024-03-21.

[2] Accuracy vs. precision vs. recall in machine learning: what's the difference? https://www.evidentlyai.com/classification-metrics/accuracy-precision-recall, 2024. Accessed: 2024-03-06.

[3] Max Bramer. *Principles of Data Mining*. Undergraduate Topics in Computer Science. Springer London, 3 edition, 2016. 123 b/w illustrations.

[4] Carmen Chan. What is a roc curve and how to interpret it. https://www.displayr.com/what-is-a-roc-curve-how-to-interpret-it/, 2024.

[5] Stephen H. Curry and Robin Whelpton. Appendix 1: Mathematical concepts and the trapezoidal method, 2016. Accessed on 04/04/2024. Terms and Conditions: https://onlinelibrary.wiley.com/terms-and-conditions.

[6] DATAtab Team. Datatab: Online statistics calculator. https://datatab.net/tutorial/roc-curve, 2024.

[7] DeepAI. Accuracy & error rate - machine learning glossary and terms. https://deepai.org/machine-learning-glossary-and-terms/accuracy-error-rate, 2024. Accessed: 2024-03-05.

[8] Google. Classification: Precision and recall. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall, 2021. Accessed: 2024-03-06.

[9] Google Developers. Classification: Roc and auc | machine learning crash course. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc, 2022. Accessed: 2024-03-23.

[10] Google Developers. Classification: Accuracy - machine learning crash course. https://developers.google.com/machine-learning/crash-course/classification/accuracy, 2024. Accessed: 2024-03-05.

[11] Murat Goksenin Guzel. Precision, recall trade-off for accurate learning model in ai/ml. https://www.linkedin.com/pulse/precision-recall-trade-off-accurate-learning-model-aiml-guzel/, 2023. Accessed: 2024-03-06.

[12] Steven A. Hicks, Inga Strümke, Vajira Thambawita, Michael Hammou, Mathias Alexander Riegler, Pål Halvorsen, and Sathishkumar V. E. Parasa. On evaluation metrics for medical applications of artificial intelligence. *Scientific Reports*, 12(1):5979, 2022.

[13] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[14] Mostafa Ibrahim. An introduction to the f1 score in machine learning: What the f1 score is and why it matters in machine learning, 3 2024. 1 star.

[15] Stephen M. Walker II. F-score: What are accuracy, precision, recall, and f1 score? https://klu.ai/glossary/accuracy-precision-recall-f1, 2024. Accessed: 2024-03-14.

[16] Teemu Kanstrén. A look at precision, recall, and f1-score: Exploring the relations between machine learning metrics. https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec, 9 2020. Accessed: 2024-03-14.

[17] Ajitesh Kumar. Ml metrics: Sensitivity vs. specificity, 9 2018. Accessed: 2024-03-21.

[18] Scikit learn developers. sklearn.metrics.f1_score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html, 2024. Accessed: 2024-03-14.

[19] Zachary C. Lipton, Charles Elkan, and Balaji Narayanaswamy. Optimal thresholding of classifiers to maximize f1 measure. In *Mach Learn Knowl Discov Databases*, volume 8725, pages 225–239. Springer, 2014.

[20] Inna Logunova. A guide to f1 score, 7 2023.

[21] FS Nahm. Receiver operating characteristic curve: overview and practical use for clinicians. *Korean Journal of Anesthesiology*, 75(1):25–36, 2022.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[23] Saurabh Saxena. F1 to f-beta. https://towardsai.net/p/l/f1-to-f-beta, 2022. Last updated: October 10, 2022. Accessed: March 21, 2024.

[24] scikit-learn developers. Plot precision-recall curve for binary classifiers. https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html, 2024. Accessed: 2024-03-06.

[25] scikit-learn developers. sklearn.metrics.auc - Area Under the Curve. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html, 2024. Accessed on 04/04/2024.

[26] Amir Masoud Sefidian. Understanding micro, macro, and weighted averages for scikit-learn metrics in multi-class classification with example. https://iamirmasoud.com/2022/06/19/understanding-micro-macro-and-weighted-averages-for-scikit-learn-metrics-in-multi-class-classificat 2022. Accessed: 2024-03-21.

[27] Natasha Sharma. Understanding and applying f1 score: A deep dive with hands-on coding. https://arize.com/blog-course/f1-score/, 6 2023. Accessed: 2024-03-14.

[28] Xavier Vasques. *Machine Learning Theory and Applications: Hands-on Use Cases with Python on Classical and Quantum Machines*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2024. Published simultaneously in Canada.

[29] Jingyao Wu, Zhibin Zhao, Chuang Sun, Ruqiang Yan, and Xuefeng Chen. Learning from class-imbalanced data with a model-agnostic framework for machine intelligent diagnosis. *Reliability Engineering & System Safety*, 216:107934, 2021.