**My Assignment topic**

**Assignment 3**

**Model Evaluation**

**By**

**Fahad Mughal**

**ROLL #:1220100509**

**ADP (IT)**

**Fall 2024**

**COURSE TITLE:** **Artifical intelligence**

**SUBMITTED TO : Sir Zubair**

****

**Department of Law International Institute of Science, Arts and Technology (IISAT), Gujranwala**

**Task 1**

**Accuracy Metrics Calculation**

**Question 01**

**What are the calculated values for accuracy, precision, recall, and F1-score?**

**What do these metrics tell you about your model&#39;s performance?**

**Anwer :**

**Calculated Values**

- Accuracy: 1.0

- Precision: 1.0

- Recall: 1.0

-F1-Score: 1.0

**Interpretation of Metrics**

1. **Accuracy:**

- **Value:** 1.0 (or 100%)

**Meaning**: This means that the model correctly predicted all instances in the test set. It indicates perfect overall correctness.

**Implication:** The model is highly accurate for this dataset, showing no errors in classification.

**2. Precision:**

**- Value:** 1.0 (or 100%)

- **Meaning:** This indicates that when the model predicts a certain class, it is always correct. There are no false positives.

**- Implication**: The model is precise, meaning it doesn't mistakenly classify instances into the wrong class.

**3. Recall:**

- **Value:** 1.0 (or 100%)

- **Meaning:** This shows that the model successfully identified all actual instances of each class. There are no false negatives.

**- Implication:** The model is highly sensitive and able to capture all relevant instances of each class.

4**. F1-Score:**

**- Value:** 1.0 (or 100%)

- **Meaning:** As the harmonic mean of precision and recall, this indicates a perfect balance between the two.

- **Implication:** The model performs excellently in both precision and recall, making it a robust and reliable classifier.

**What These Metrics Tell You About Your Model's Performance**

The perfect scores across all metrics indicate that the model performs exceptionally well on the Iris dataset. Specifically:

**- High Accuracy:** The model is very reliable in making correct predictions overall.

**- High Precision:** The model is very good at avoiding false positives, meaning it makes very few mistakes when predicting a class.

- **High Recall**: The model does not miss any relevant instances, meaning it makes very few false negatives.

**- High F1-Score:** The model maintains a good balance between precision and recall, suggesting it is effective at both identifying correct instances and avoiding incorrect ones.

**Potential Caveats**

While these results are excellent, they might raise some concerns in a real-world context:

- **Dataset Simplicity**: The Iris dataset is relatively small and simple, which might not represent the complexity of real-world data.

**- Model Overfitting**: Perfect scores might indicate that the model overfitted the training data, especially if the dataset is not diverse enough. In practice, such perfect scores are rare, and further validation with more complex datasets would be necessary to ensure the model's robustness and generalizability.

**Task 2**

**Confusion Matrix Interpretation**

**Question 2**

**Anwer**

**Confusion Matrix Generation**

First, let's generate the confusion matrix using the Logistic Regression model trained on the Iris dataset. Here is the Python code for that:

python

# Import necessary libraries

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

# Load the Iris dataset

iris = load\_iris()

X, y = iris.data, iris.target

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a Logistic Regression model

model = LogisticRegression(max\_iter=200)

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Generate the confusion matrix

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

# Display the confusion matrix

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=iris.target\_names)

disp.plot(cmap=plt.cm.Blues)

plt.show()

**Confusion Matrix**

Assuming the resulting confusion matrix looks like this:

lua

Copy code

Confusion Matrix:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

**Explanation of Each Value**

The confusion matrix for the Iris dataset has three classes: setosa, versicolor, and virginica. The matrix is structured as follows:

markdown

Copy code

Predicted

setosa versicolor virginica

Actual

setosa 10 0 0

versicolor 0 9 0

virginica 0 0 11

* **True Positives (TP)**: The diagonal elements (10, 9, 11) represent the number of correct predictions for each class.
  + **Setosa**: 10 instances were correctly predicted as setosa.
  + **Versicolor**: 9 instances were correctly predicted as versicolor.
  + **Virginica**: 11 instances were correctly predicted as virginica.
* **False Positives (FP)**: The off-diagonal elements in each row represent the number of incorrect predictions for that class (predicted as another class).
  + **Setosa**: 0 instances were incorrectly predicted as versicolor or virginica.
  + **Versicolor**: 0 instances were incorrectly predicted as setosa or virginica.
  + **Virginica**: 0 instances were incorrectly predicted as setosa or versicolor.
* **False Negatives (FN)**: The off-diagonal elements in each column represent the number of instances that were incorrectly predicted as another class.
  + **Setosa**: 0 instances of setosa were predicted as another class.
  + **Versicolor**: 0 instances of versicolor were predicted as another class.
  + **Virginica**: 0 instances of virginica were predicted as another class.

**How the Confusion Matrix Helps in Understanding the Model's Performance**

The confusion matrix is a powerful tool for understanding the performance of a classification model:

1. **Detailed Breakdown**:
   * Unlike overall metrics like accuracy, the confusion matrix provides a detailed breakdown of how the model performs on each class. This helps identify specific classes where the model may be underperforming.
2. **Identifying Errors**:
   * By looking at false positives and false negatives, we can understand the types of errors the model is making. For example, if a class has high false negatives, the model may need improvement in recognizing instances of that class.
3. **Class Imbalance**:
   * It helps in identifying issues related to class imbalance. If one class has significantly more instances than others, it might dominate the accuracy score. The confusion matrix helps to see if the model is biased towards the majority class.
4. **Performance Metrics**:
   * Precision, recall, and F1-score for each class can be derived from the confusion matrix, providing more granular insights into model performance beyond overall accuracy.

**Summary**

* **True Positives** (TP): Correctly predicted instances for each class.
* **False Positives** (FP): Incorrectly predicted instances as the class.
* **False Negatives** (FN): Instances of the class incorrectly predicted as another class

**Task 3**

**ROC/AUC Calculation**

**Question**

**Answer**

**ROC Curve and AUC Calculation**

Here's how we can compute and interpret the ROC curve and AUC:

python

# Import necessary libraries

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

# Load the Iris dataset

iris = load\_iris()

X, y = iris.data, iris.target

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a Logistic Regression model

model = LogisticRegression(max\_iter=200)

model.fit(X\_train, y\_train)

# Predict probabilities for the test set

probs = model.predict\_proba(X\_test)

# Compute ROC curve and AUC for each class

fpr = {}

tpr = {}

roc\_auc = {}

for i in range(len(iris.target\_names)):

fpr[i], tpr[i], \_ = roc\_curve(y\_test, probs[:, i], pos\_label=i)

roc\_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class

plt.figure(figsize=(8, 6))

colors = ['blue', 'red', 'green']

for i, color in zip(range(len(iris.target\_names)), colors):

plt.plot(fpr[i], tpr[i], color=color, lw=2,

label='ROC curve of class {0} (AUC = {1:0.2f})'

''.format(iris.target\_names[i], roc\_auc[i]))

plt.plot([0, 1], [0, 1], color='black', lw=1, linestyle='--') # Plotting the diagonal line

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 (ROC) Curve')

plt.legend(loc="lower right")

plt.show()

# Print AUC values

for i in range(len(iris.target\_names)):

print(f'AUC for class {iris.target\_names[i]}: {roc\_auc[i]:.2f}')

**Interpretation of ROC Curve and AUC**

* **ROC Curve**:
  + The ROC curve plots the True Positive Rate (sensitivity) against the False Positive Rate (1 - specificity) for different threshold values. Each point on the ROC curve represents a sensitivity/specificity pair corresponding to a particular decision threshold.
  + It shows the trade-off between sensitivity and specificity: a good model will have a curve that is closer to the top-left corner, indicating higher true positive rates and lower false positive rates across different thresholds.
* **AUC (Area Under the Curve)**:
  + AUC quantifies the overall performance of the model across all possible classification thresholds. It represents the probability that the model will correctly classify a randomly chosen positive instance higher than a randomly chosen negative instance.
  + AUC ranges from 0 to 1, where 0.5 indicates a model that performs no better than random guessing, and 1 indicates perfect performance.

**How These Metrics Help in Evaluating Model Performance**

1. **ROC Curve**:
   * **Visual Assessment**: It provides a visual representation of the model's performance, showing how well the model can distinguish between classes.
   * **Threshold Selection**: Helps in selecting the optimal threshold depending on the specific use case (e.g., balancing sensitivity and specificity).
2. **AUC**:
   * **Overall Performance**: AUC provides a single metric to compare different models or different configurations of the same model.
   * **Model Comparison**: Higher AUC values indicate better discrimination between classes and better overall performance of the model.

**Example Interpretation**

* Suppose the ROC curve shows that for the class 'setosa', the curve is close to the top-left corner with an AUC of 1.0, indicating perfect discrimination. For other classes like 'versicolor' and 'virginica', the curves might not be as steep but still show good discrimination (AUC close to 1.0).
* If the AUC values are lower or closer to 0.5, it suggests that the model may not be effectively distinguishing between classes, indicating potential areas for improvement.

**Task 4**

**Cross-Validation Reporting**

**Question**

**Anwer**

### Mean and Standard Deviation of Cross-Validation Accuracy

Cross-validation involves splitting the dataset into multiple folds, training the model on some folds, and evaluating it on the remaining fold. This process is repeated multiple times to obtain a more reliable estimate of the model's performance.

Here's how we can calculate the mean and standard deviation of cross-validation accuracy using Python and scikit-learn:

python

# Import necessary libraries

from sklearn.datasets import load\_iris

from sklearn.model\_selection import cross\_val\_score, train\_test\_split

from sklearn.linear\_model import LogisticRegression

import numpy as np

# Load the Iris dataset

iris = load\_iris()

X, y = iris.data, iris.target

# Initialize the Logistic Regression model

model = LogisticRegression(max\_iter=200)

# Perform cross-validation

# Here, we use 5-fold cross-validation and specify 'accuracy' as the scoring metric

cv\_scores = cross\_val\_score(model, X, y, cv=5, scoring='accuracy')

# Calculate mean and standard deviation of cross-validation accuracy

mean\_accuracy = np.mean(cv\_scores)

std\_accuracy = np.std(cv\_scores)

mean\_accuracy, std\_accuracy

### Explanation:

1. **cross\_val\_score**: This function from scikit-learn performs cross-validation by splitting the dataset into specified number of folds (cv=5 in this case) and evaluates the model performance using the specified scoring metric (accuracy).
2. **mean**: np.mean(cv\_scores) calculates the mean of the cross-validation scores, providing an average measure of the model's performance across different folds.
3. **standard deviation**: np.std(cv\_scores) computes the standard deviation of the cross-validation scores, indicating the variability or consistency of the model's performance across folds.

### Interpretation:

* **Mean Accuracy**: It represents the average accuracy of the model across all cross-validation folds. A higher mean accuracy indicates that the model is generally performing well on the dataset.
* **Standard Deviation of Accuracy**: It shows how much the cross-validation accuracy scores vary from the mean. A lower standard deviation suggests that the model's performance is consistent across folds, while a higher standard deviation might indicate variability in performance.

### Importance of Cross-Validation in Model Evaluation

Cross-validation is crucial in model evaluation for several reasons:

1. **Better Estimation of Performance**: It provides a more reliable estimate of the model's performance by averaging results from multiple train-test splits of the data.
2. **Reduces Overfitting**: Cross-validation helps in detecting overfitting because it evaluates the model on multiple splits of the data, ensuring that the model generalizes well to unseen data.
3. **Optimizes Hyperparameters**: It aids in hyperparameter tuning by selecting the best parameters that maximize model performance across different cross-validation folds.
4. **Handles Data Variability**: Cross-validation mitigates the impact of data variability by testing the model's robustness on different subsets of the data, thus providing a more comprehensive evaluation.
5. **Provides Confidence Intervals**: Standard deviation of cross-validation scores gives insights into the stability and consistency of the model's performance, providing a range of possible performance metrics.