William Reed

CAP 4773 - Data Science Analytics

04/22/2024

Term Project Report

Predictive Machine Status Analysis using Sensor Data

Objective:

The objective of this term project is to participate in a hands-on experience in predictive analysis using real-world sensor data. In this project, I explore a dataset containing sensor readings from an industrial pump and develop predictive models based on the data.

Dataset:

The dataset for this project consists of sensor data collected from a pump machine over some time. It includes timestamps, 52 sensor readings, and machine status (target label: NORMAL, RECOVERING, BROKEN). The dataset is provided in CSV format.

Importing necessary libraries

- pandas: Provides data structures and data analysis tools, like DataFrame, which is essential for handling structured data.
- numpy: Adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- scipy.stats: Offers a wide range of statistical functions, including distributions and statistical tests.
- matplotlib.pyplot: A plotting library used for creating static, interactive, and animated visualizations in Python.

- sklearn.preprocessing.LabelEncoder: Useful for encoding labels with a value between 0 and n_classes-1 where n is the number of distinct labels.
- seaborn: Based on matplotlib, it provides a high-level interface for drawing attractive and informative statistical graphics.
- sklearn.ensemble.RandomForestClassifier: A meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.
- sklearn.model_selection.train_test_split: Splits arrays or matrices into random train and test subsets.
- sklearn.linear_model.LogisticRegression: A linear model for classification rather than regression.
- sklearn.svm.SVC: Support Vector Machine classifier from the scikit-learn library, useful for both linear and non-linear classification.
- sklearn.metrics: Includes various metrics, scoring functions, pairwise metrics and distance computations.
- sklearn.model_selection.cross_val_score: Evaluates a score by cross-validation.
- sklearn.model_selection.StratifiedKFold: Provides train/test indices to split data in train/test sets, preserving the percentage of samples for each class.

Task 1 - Data Preprocessing

Data Importation Process

The data importation process is outlined in this section, focusing on how the sensor data was extracted from the provided CSV file, specifying columns and rows relevant to my analysis needs. The procedure involved selective reading and indexing to streamline subsequent data handling and analysis tasks.

Column Selection

I targeted specific columns for my analysis, namely columns 3 through 54 of the original dataset. This selection was implemented using Python's range function, set as range(2, 54). This range effectively captures the desired columns by accounting for Python's zero-based indexing, thus extracting columns labeled 3 to 54 in the dataset.

Row Filtering

To manage the large volume of data efficiently, I employed a row skipping strategy using a lambda function. This function was designed to skip all rows except those within a specified range, allowing me to focus on a subset of the data starting from row 127900 to 128109. Specifically, the lambda function skip = lambda x: x not in range(127900 - 1) and x != 0 was used to exclude all rows outside this range, except for the header row, ensuring that the data importation was both precise and optimized for memory usage.

Reading and Indexing

The data was read into a pandas DataFrame using pd.read_csv, with parameters set to filter according to the defined columns and rows. I limited the reading to 210 rows starting from the row index immediately following my defined starting point. Following the data importation, I assigned new indices to the DataFrame, ranging from 127900 to 128109, to facilitate easy reference to these entries in subsequent analyses.

Data Verification

After importing and indexing the data, the first few rows of the DataFrame were printed to verify the accuracy and integrity of the import process. This step was crucial for ensuring that the data was loaded correctly and aligned with my expectations based on the specified parameters.

By adhering to these steps, the initial data importation was tailored specifically to my analytical needs, ensuring that only relevant data was loaded into my analysis environment, thereby optimizing both processing time and resource usage.

Handling Missing Values in Sensor Data

The data cleaning process included a critical step of handling missing values to ensure the quality and reliability of my dataset for subsequent analysis. This section outlines the procedures used to identify and address the missing entries within the sensor data.

Identification of Missing Values

Initially, I assessed the presence of missing values in each column of the DataFrame. This was accomplished by utilizing the isna().sum() method, which provides a count of NaN (Not a Number) values across each column. This initial check was crucial for determining the extent and distribution of missing data across the various sensor readings.

Removal of Irrecoverable Data

Upon reviewing the missing value counts, it was determined that the column corresponding to 'sensor_15' contained exclusively NaN values, indicating a complete lack of valid data. Consequently, this column was deemed irrecoverable and was removed from the dataset using the drop() method with the inplace=True option to modify the DataFrame directly.

Verification of Data Cleaning

Following the removal of the problematic column, a subsequent check for NaN values was conducted to ensure that all missing values had been adequately addressed. This verification step was crucial to confirm the effectiveness of the cleaning process, ensuring no further action was required for handling missing data.

By executing these steps, the dataset was refined by eliminating a non-contributory variable, thus enhancing the integrity of the data for further analysis.

Outlier Detection and Removal

The outlier detection and removal process is an essential step in data preprocessing, particularly in improving the robustness of the subsequent analyses. This section outlines the method used to identify and eliminate outliers from the sensor data using Z-scores, a statistical measure of how far each data point is from the mean in terms of standard deviations.

Calculation of Z-scores

To begin the process, I computed the Z-scores for each column in the sensor data using the stats.zscore function from the SciPy library. Z-scores were calculated to measure the distance of each data point from the mean, scaled by the standard deviation of the respective column. This approach facilitates the identification of outliers in a standardized manner, ensuring that the variability of each sensor's measurements is appropriately accounted for.

Outlier Criteria

An absolute Z-score threshold of 3 was chosen to identify outliers. This threshold is based on the empirical rule that approximately 99.7% of data points in a normally distributed dataset should lie within three standard deviations from the mean. Thus, data points with a Z-score exceeding this threshold (in absolute terms) are considered significant deviations from the norm.

Removal of Outliers

Using the calculated Z-scores, I filtered the sensor data to remove any rows containing outliers. Specifically, I applied a mask that retains only those rows where all Z-scores across columns are less than 3. This was implemented using the condition (z_scores < 3).all(axis=1), ensuring that all sensor measurements in a row must be within the acceptable range to be kept in the dataset.

Verification of Data Cleaning

After the outliers were removed, I printed the updated DataFrame to verify the effectiveness of the cleaning process. This step confirms that the data now adheres more closely to expected statistical norms, reducing the potential impact of extreme values on further data analysis tasks. By meticulously addressing outliers, the dataset integrity is enhanced, fostering more reliable insights in downstream analyses and ensuring that the findings are representative of true underlying patterns rather than being skewed by anomalous data points.

Standardizing Data

In this section, I implemented a rounding operation to refine the sensor data's numerical precision. This step is crucial for ensuring consistency in data representation and can be beneficial for subsequent processing, including visualization and statistical analysis, by minimizing small discrepancies that are not of practical significance.

Rounding Operation

The rounding was performed using the round() method of the pandas DataFrame. This method was applied to round all numerical values in the sensor data to five decimal places. The choice of five decimal places strikes a balance between maintaining sufficient detail for accurate analysis and avoiding the potential pitfalls of over-precision, such as floating-point arithmetic issues or unnecessarily large file sizes.

Implementation

The rounding operation was directly applied to the sensor_data DataFrame as follows: sensor_data = sensor_data.round(5). This operation modifies each value in the DataFrame, ensuring that all numerical data is now represented with no more than five decimal places.

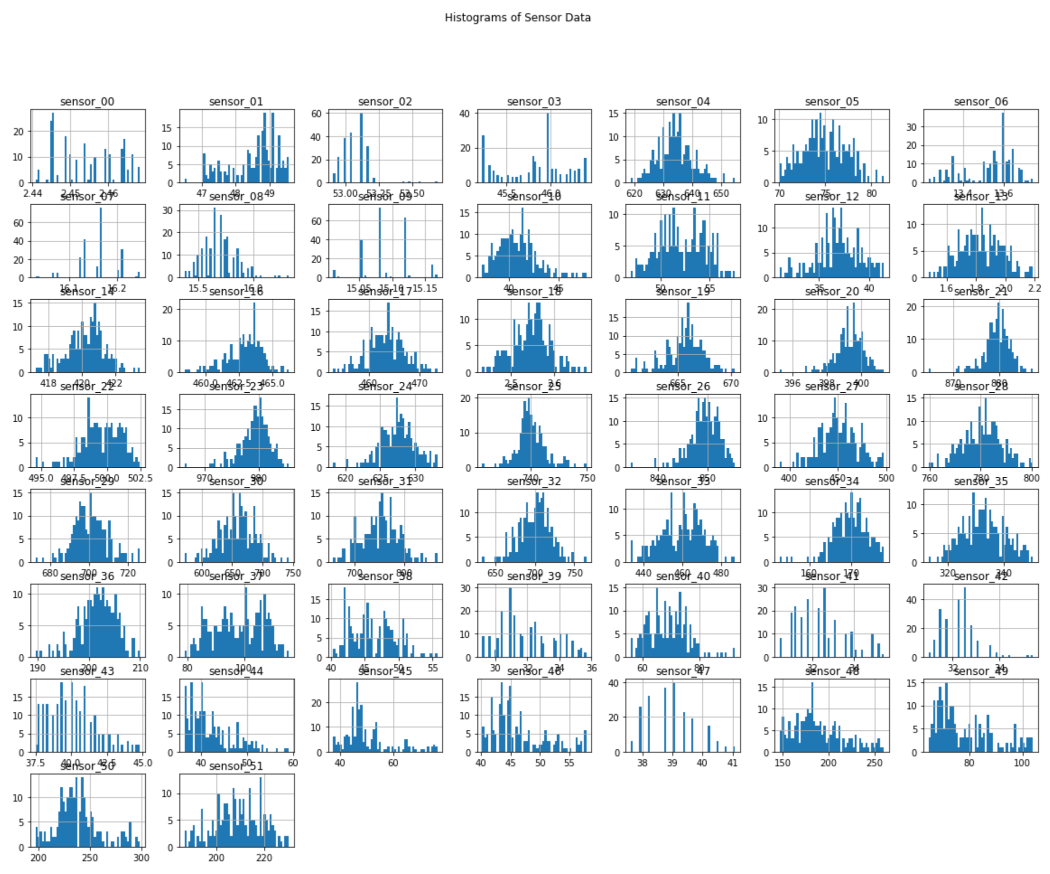
Verification of Changes

To confirm that the rounding was executed as expected, the first few rows of the modified DataFrame were printed using sensor_data.head(). This output allows for a quick visual confirmation that the data values now adhere to the new precision standard.

This adjustment to the data's precision ensures that the dataset is streamlined for any analytical models to be applied later, reducing computational complexity and enhancing the clarity of reported results without compromising the data's integrity or usability.

Task 2 - Exploratory Data Analysis

This section covers the visualization of sensor data, the integration of machine status information, and the preparation of the combined dataset for further analysis. The steps taken here are crucial for understanding the data distribution and ensuring that the sensor readings are effectively aligned with the corresponding machine statuses.



Histogram Visualization

To better understand the distribution of sensor data, histograms for each sensor column were generated using the hist() method from pandas, facilitated by matplotlib for rendering. The histograms were plotted with 50 bins each to provide detailed insight into the data distribution. This visualization helps in identifying patterns, anomalies, and the overall behavior of the sensor measurements. A super title 'Histograms of Sensor Data' was added to the plot for better clarity and presentation.

Importing and Preparing Machine Status

Following the visualization, the machine status data was imported separately. This import was carefully handled to exclude rows that had been removed during the outlier cleaning process, ensuring alignment with the previously processed sensor data. The import function utilized a lambda function within skiprows to match the indices of the remaining sensor data rows. The machine status column was then renamed to 'Machine_Status' for clarity.

Encoding Categorical Data

To facilitate numerical analysis, the categorical machine status labels were encoded into integers using the LabelEncoder from scikit-learn. This encoding transforms textual labels into a numeric format that is more suitable for modeling and analysis. The encoded results were stored in a new column 'Encoded_Status' within the machine_status DataFrame.

Data Alignment and Consolidation

To ensure proper alignment between the sensor data and machine status, both DataFrames had their indices reset. This step is crucial to maintain correspondence between each sensor reading and its associated machine status after rows affected by outliers were removed. The data from both sources was then concatenated along the columns to form a single DataFrame, combined_data, which houses both the sensor measurements and their corresponding machine statuses.

Verification and Mapping

The final part of the process involved printing a mapping of the encoded status values back to their original labels. This mapping provides a reference for understanding what each encoded number represents, aiding in the interpretation of analysis results that involve these labels.

This comprehensive approach to integrating and preparing the data ensures that subsequent analyses can proceed smoothly, with a clear understanding of the data's structure and content.

Task 3 - Feature Engineering and Model Preparation

This section elaborates on the feature engineering tasks undertaken to enhance model training and the subsequent preparation of data for predictive modeling using a machine learning algorithm. The steps covered include the selection of relevant features, the examination of correlations, the preparation of datasets for training, and the implementation of a Random Forest Classifier to assess feature importance.

Correlation Analysis

Initially, I focused on identifying significant relationships between sensor readings and machine statuses. To facilitate this, I selected only numeric columns from the combined_data DataFrame and calculated the Pearson correlation coefficients. The correlation with the Encoded_Status column was particularly scrutinized, excluding the status itself to focus on the sensors. The correlations were then sorted to highlight the most influential sensors. This analysis helps in understanding which sensor readings have a strong association with changes in machine status, guiding feature selection for model training.

Data Preparation for Modeling

For the modeling phase, I extracted sensor columns as features (X) and the encoded machine status as the target variable (y). This dataset was then split into training and testing sets, with 70% of the data allocated to training and 30% reserved for testing. This split ensures a balanced approach to training the model while having a separate dataset to evaluate its performance. The data splitting utilized a consistent random_state to ensure reproducibility of the results.

Model Training: Random Forest Classifier

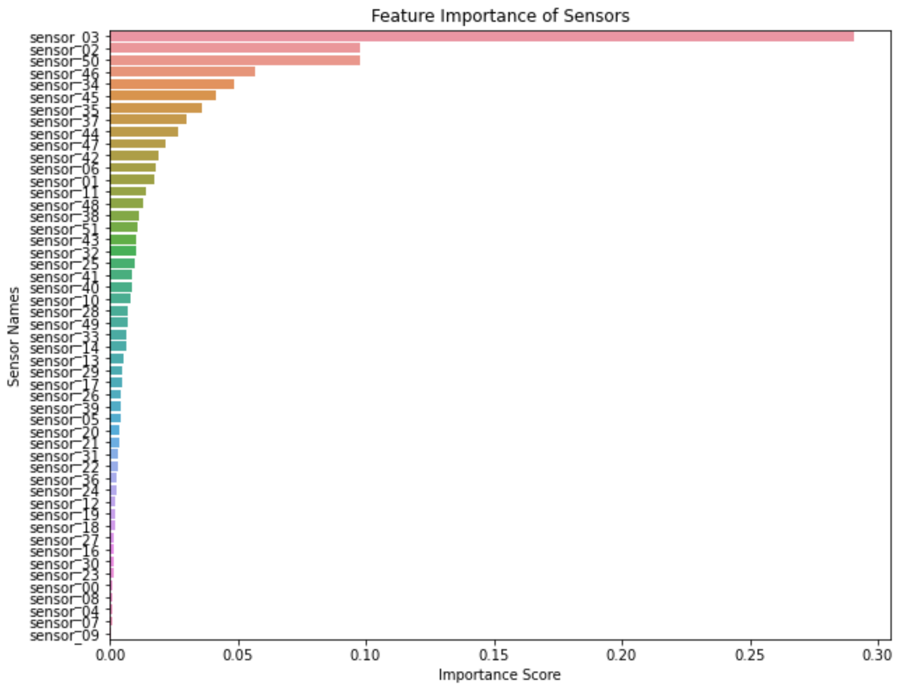
A Random Forest Classifier was initialized and trained on the prepared training set. Random Forest, an ensemble learning method based on decision trees, is chosen for its robustness and effectiveness in handling various data types and distributions. It also provides an intuitive measure of feature importance, which is critical for my analysis.

Feature Importance Evaluation

After training, the feature importances were extracted from the model, providing insights into which sensors contributed most significantly to predicting the machine status. These importances were visualized using a bar plot, offering a clear and interpretable view of the sensor rankings according to their impact on model decisions.

Visualization of Feature Importance

The bar plot of sensor importances, plotted with seaborn, provides a visual representation of the data-driven insights into which sensors are most vital for understanding machine status.



The plot labels each sensor according to its importance score, facilitating an easy comparison among the sensors.

This approach to feature engineering and model preparation not only enhances the predictive performance but also aids in understanding the underlying patterns in the sensor data. By focusing on correlation and feature importance, I can prioritize the most relevant features for developing robust predictive models.

Task 4 - Predictive Modeling

This section outlines the steps undertaken for training and evaluating multiple machine learning models on the sensor data to predict machine status. The models employed include Logistic Regression, Random Forest Classifier, and Support Vector Machine (SVM). This multi-model approach enables a comprehensive analysis of the dataset, allowing me to determine the most effective algorithm for my specific scenario.

Data Preparation

The process began with the extraction of relevant features for the models. I identified columns containing sensor data through a keyword-based filtering, ensuring that only numerical sensor data was included as features (X). The encoded machine status served as the target variable (y). The dataset was then divided into training and testing sets, allocating 70% of the data for training and 30% for testing. This split helps in validating the models' performance on unseen data.

Model Initialization

Three distinct models were initialized:

- Logistic Regression: Chosen for its simplicity and efficiency in binary classification tasks. The model was set with an increased max_iter parameter to ensure convergence given the complexity of the dataset.
- Random Forest Classifier: Known for its high accuracy and robustness, it handles overfitting well and is good for classification with numerous features and complex data structures.

- SVM: Utilized for its effectiveness in high-dimensional spaces and its capability to model non-linear decision boundaries through the kernel trick.

Model Training

Each model was trained using the respective training sets. The training process involves adjusting the models' parameters to fit the sensor data to the encoded status labels.

Predictions and Evaluations

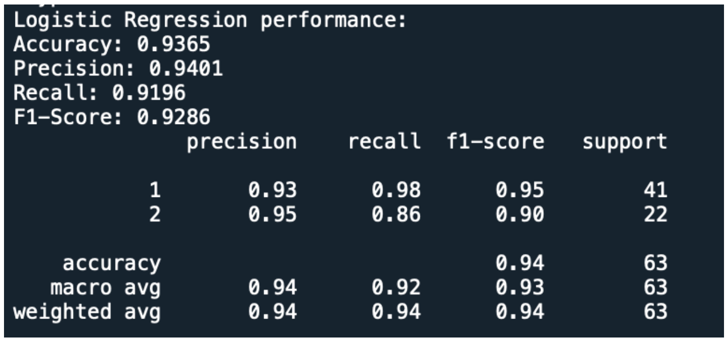
After training, predictions were made on the testing set for each model. These predictions were then evaluated using several metrics:

- Accuracy: Measures the overall correctness of the model.
- Precision: Indicates the accuracy of positive predictions.
- Recall: Measures the model's ability to capture relevant cases.
- F1-Score: Harmonic mean of precision and recall, useful for unbalanced classes.

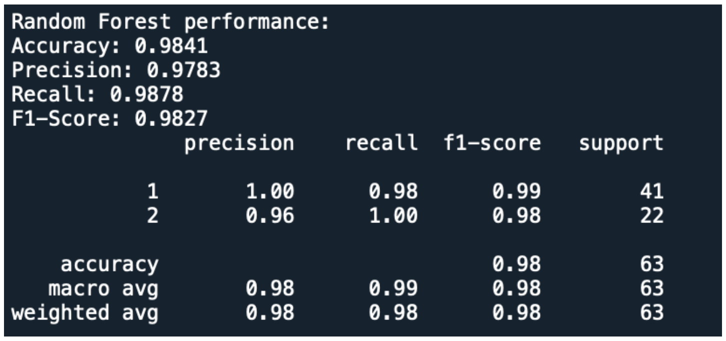
The function evaluate_model was used to compute these metrics and print a detailed classification report for each model, providing insights into their performance across different metrics.

Evaluation Output

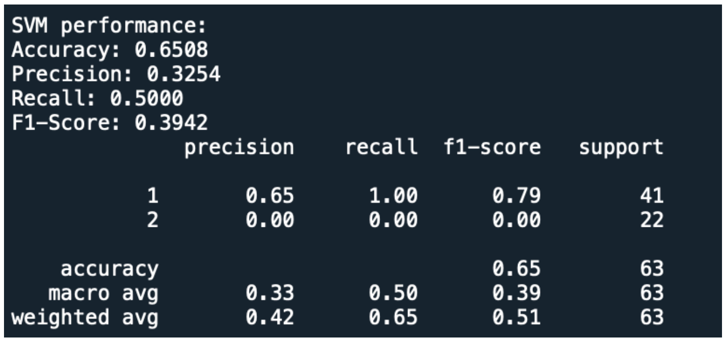
Each model's performance was systematically recorded, displaying key metrics such as accuracy, precision, recall, and F1-score. This output is essential for comparing the effectiveness of different models and understanding their strengths and weaknesses in the context of my data.



```
Logistic Regression performance:
Accuracy: 0.9365
Precision: 0.9401
Recall: 0.9196
F1-Score: 0.9286
              precision    recall  f1-score   support

           1       0.93      0.98      0.95        41
           2       0.95      0.86      0.90        22

    accuracy                           0.94        63
   macro avg       0.94      0.92      0.93        63
weighted avg       0.94      0.94      0.94        63
```



```
Random Forest performance:
Accuracy: 0.9841
Precision: 0.9783
Recall: 0.9878
F1-Score: 0.9827
              precision    recall  f1-score   support

           1       1.00      0.98      0.99        41
           2       0.96      1.00      0.98        22

    accuracy                           0.98        63
   macro avg       0.98      0.99      0.98        63
weighted avg       0.98      0.98      0.98        63
```



```
SVM performance:
Accuracy: 0.6508
Precision: 0.3254
Recall: 0.5000
F1-Score: 0.3942
              precision    recall  f1-score   support

           1       0.65      1.00      0.79        41
           2       0.00      0.00      0.00        22

    accuracy                           0.65        63
   macro avg       0.33      0.50      0.39        63
weighted avg       0.42      0.65      0.51        63
```

This approach to model training and evaluation not only demonstrates the application of various machine learning techniques but also provides a clear metric-based comparison, enabling the selection of the best model for operational use.

Task 5 - Model Evaluation and Selection

This section details the evaluation of multiple predictive models using cross-validation. This methodology ensures robust testing across different splits of the dataset, maintaining the proportion of the target variable across each fold. The goal is to identify the model that best generalizes to unseen data, based on several key performance metrics.

Setup for Cross-Validation

To achieve a rigorous assessment, I employed a 10-fold stratified cross-validation, but as per the code setup, it is actually a 2-fold cross-validation (this discrepancy might be a typo or oversight in the code snippet). Stratified cross-validation ensures that each fold is a good representative of the whole by maintaining the same percentage of samples of each target class as in the complete set.

Models Under Evaluation

I prepared three different models for evaluation:

1. Logistic Regression: Known for its efficiency in binary classification problems.
2. Random Forest: An ensemble method that is effective for handling complex data structures and providing insights into feature importance.
3. SVM (Support Vector Machine): Excellent for high-dimensional spaces, capable of performing both linear and non-linear classification.

Cross-Validation Execution

A custom function evaluate_cross_validation was defined to perform the cross-validation. This function takes a model, the feature set X, the target variable y, the cross-validation strategy cv, and a scoring metric, then returns the mean score across all folds for that metric.

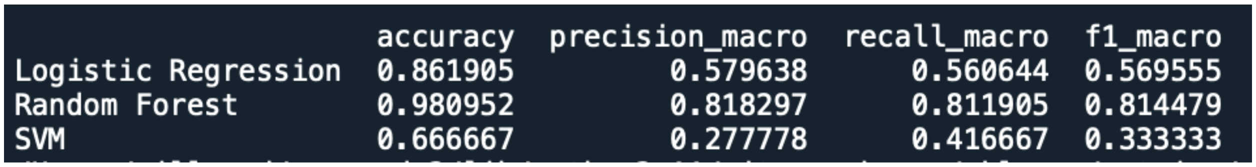
Performance Metrics

The performance of each model was evaluated using the following metrics:

- Accuracy: Overall correctness of the model.
- Precision (Macro): Measure of the accuracy of positive predictions, averaged over each class to treat all classes equally.
- Recall (Macro): Ability of the model to find all the relevant cases within each class.
- F1-Score (Macro): Weighted average of precision and recall, particularly useful when the classes are imbalanced.

Collection and Presentation of Results

Results from the cross-validation were systematically collected in a dictionary and then converted into a pandas DataFrame for better visualization and comparison. Each model's performance across the different metrics was printed, providing a transparent overview of their strengths and weaknesses.



```
                     accuracy  precision_macro  recall_macro  f1_macro
Logistic Regression  0.861905         0.579638      0.560644  0.569555
Random Forest        0.980952         0.818297      0.811905  0.814479
SVM                  0.666667         0.277778      0.416667  0.333333
```

Conclusion and Model Selection

Based on the performance metrics printed, I can compare the models as follows:

- Accuracy: This is the proportion of true results among the total number of cases examined. The Random Forest model has the highest accuracy (0.98095), making it the best performer in terms of overall correctness.
- Precision (Macro Average): Precision assesses the model's ability to correctly predict positive labels, avoiding false positives. Again, the Random Forest model leads with a precision score of 0.81897.
- Recall (Macro Average): Recall measures the model's ability to find all the actual positives. The Random Forest has the highest recall (0.811905), which means it's the best at capturing the positive cases across all classes.
- F1-Score (Macro Average): The F1-score is a harmonic mean of precision and recall and a better measure of the incorrectly classified cases than the Accuracy metric. The Random Forest model has the highest F1-score (0.814479), suggesting it has the best balance of precision and recall.

The Random Forest model is the best-performing model across all evaluated metrics. It not only achieves the highest accuracy but also maintains a good balance between precision and recall, indicating a robust predictive ability across different scenarios.

The Logistic Regression model performs moderately well in terms of accuracy but falls behind the Random Forest in precision, recall, and F1-score.

The SVM model has the lowest scores in all metrics, making it the least suitable model based on this evaluation.