Predictive Maintenance Case Study in R

Overview

An asset is comprised of **100 machines**, each containing **4 critical components**. The goal is to predict the failure of a specific component within the next 24 hours. This task is framed as a multi-class classification problem, where each class corresponds to one of the four components.

Machines and Components

- **Machines:**
    - **Identification:** Each machine is uniquely identified by a machineID (1 to 100).
    - **Metadata:**
        - **Model type**
        - **Age:** Number of years in service.
- **Components:**
    - Every machine has 4 components.
    - When a component fails, it is replaced.
    - The failure type detection provides information about which component is expected to fail, enabling the maintenance team to replace it accurately.

Sensors and Data Collection

- **Sensors:** Each machine is equipped with 4 sensors that continuously record:
    - Voltage
    - Rotation
    - Pressure
    - Vibration

Types of Events

- **Faults (Errors):**
    - These are non-critical errors occurring while the machine remains operational.
    - There are five types of errors, identified by errorIDs: "error1", "error2", "error3", "error4", and "error5".
- **Failures:**
    - A failure results in the machine becoming non-operational.
    - Failures can occur in any one of the four components, and multiple components may fail simultaneously.

- Failure events are identified by component IDs: "comp1", "comp2", "comp3", and "comp4".

Data Acquisition

- **Telemetry Data:**
  - Time-series data of voltage, rotation, pressure, and vibration measurements.
  - Collected hourly from all 100 machines throughout the year 2015 (starting from January 1, 2015).
- **Errors Data:**
  - Time-series records of faults (non-critical errors) occurring in each machine during 2015.
- **Maintenance Records:**
  - Generated when a component is replaced either during scheduled inspections (preventive maintenance) or after a breakdown.
  - Each record includes the date and time (all records are timestamped at 06:00:00), machine ID, and the type of replaced component.
  - Data spans from June 1, 2014, to 2015.
- **Machines Data:**
  - Contains machine-specific metadata such as model type and age.
- **Failures Data:**
  - Focuses on records of component replacements due solely to breakdowns.
  - Includes the date and time, machine ID, and the failed component type.
  - Although most failure records are also present in the maintenance data, some records are unique to this dataset.
  - This data is used exclusively for labeling the failure events.

Available Files (CSV Format)

- **PdM_telemetry.csv:** Telemetry data
- **PdM_errors.csv:** Errors data
- **PdM_maint.csv:** Maintenance records
- **PdM_machines.csv:** Machines metadata
- **PdM_failures.csv:** Failures data

Data Preprocessing

1. Data Type Transformation

- **Telemetry Data Frame**
    - **Datetime Field:** Initially provided as a character string.
    - **Action:** Convert to a calendar date/time format (e.g., using the "Y/M/D H/M/S UTC" format).
- **Errors Data Frame**
    - **Datetime Field:** Initially provided as a character string.
        - **Action:** Transform into a calendar date/time format (e.g., "Y/M/D H/M/S UTC").
    - **errorID Field:** Initially provided as a character string.
        - **Action:** Convert to a categorical variable (factor) with five categories: "error1", "error2", "error3", "error4", and "error5".
- **Maintenance (Maint) Data Frame**
    - **Datetime Field:** Provided as a character string.
        - **Action:** Convert to a calendar date/time format.
    - **comp Field:** Provided as a character string.
        - **Action:** Transform into a categorical variable (factor) with four categories: "comp1", "comp2", "comp3", and "comp4".
- **Failure Data Frame**
    - **Datetime Field:** Provided as a character string.
        - **Action:** Convert to a calendar date/time format.
    - **failure Field:** Provided as a character string.
        - **Action:** Convert to a categorical variable (factor) with four categories: "comp1", "comp2", "comp3", and "comp4". *(Note: In the Maintenance data set, this field is called comp, whereas in the Failures data set it is called failure.)*
- **Machine Data Frame**
    - **Model Field:** Initially provided as a character string.
        - **Action:** Although it could remain as a string, for this experiment it is converted into a categorical variable (factor) with four machine models: "model1", "model2", "model3", and "model4".

2. Cleaning Missing Data

- **Missing Values Check:**
    - **Procedure:** Count the total number of NA/NaN values in each data frame.
    - **Result:** There are no missing values detected.

3. Removing Duplicate Rows

- **Duplicate Check:**

- **Procedure:** Count the total number of duplicated rows.
- **Result:** No duplicated rows were found.

4. Identifying Outliers

- **Outlier Detection:**
  - **Procedure:** Use boxplots for each data frame to visually inspect for outliers.
  - **Result:** No outliers were identified for deletion.

Feature Engineering

Overview

While feature selection appears to be unnecessary (as can be confirmed via a correlation analysis), feature extraction is required. Different extraction strategies will be applied to each data frame, with a key focus on using rolling aggregates to enhance historical context and reduce dataset size.

Rolling Aggregate Technique

Purpose:

1. **Enhance Historical Context:**
   - Improve the representation of both short- and long-term history for each record.
   - **Solution:** Apply a rolling window to compute aggregate statistics.
2. **Reduce Dataset Size:**
   - Original dataset: 876,100 rows × 4 features (voltage, rotation, pressure, vibration) for 100 machines (sampled every hour).
   - **Solution:** Compute rolling aggregates only for selected time units (e.g., every 3 hours).

Hypotheses and Settings:

- **Aggregating Functions:** Mean and standard deviation.
- **Aggregation Interval:** Every 3 hours (by = 3).
- **Rolling Windows:**
  - **Short-term:** Window size (W) = 3 hours.

- **Long-term:** Window size (W) = 24 hours.

R Implementation:

- **Function:** rollapply from the **zoo** library.
- rollapply(v, width = X, FUN = f, align = "right", fill = NA, by = 3)
  - **v:** Series of observations (e.g., voltage, rotation, pressure, vibration).
  - **width:** Window size (either 3 or 24 hours).
  - **FUN:** Aggregation function (mean or standard deviation).
  - **align:** "right" ensures that for each observation, only previous data points within the window are considered.
  - **fill:** Uses NA for windows that cannot be computed due to insufficient data.
  - **by:** Aggregation is computed every 3 hours.

Feature Engineering for Telemetry

New Columns Created:

- **For a 3-hour window:**
  - **Mean:** voltmean, rotatemean, pressuremean, vibrationmean
  - **Standard Deviation:** voltsd, rotatesd, pressuresd, vibrationsd
- **For a 24-hour window:**
  - **Mean:** voltmean_24h, rotatemean_24h, pressuremean_24h, vibrationmean_24h
  - **Standard Deviation:** voltsd_24h, rotatesd_24h, pressuresd_24h, vibrationsd_24h

Processing Steps

1. **Grouping by Machine and Sorting:**
   - **Step 1:** Order the data frame by machine and datetime.
   - **Step 2:** Group data by machine before applying the rolling aggregate.
2. **Adding Rolling Aggregates:**
   - For each machine group, apply rollapply on each sensor column.
   - Add new columns to store the computed aggregates.
3. **Filtering and Ungrouping:**
   - Remove rows containing NA values resulting from the rolling window computation.
   - Select the relevant columns (datetime, machine, and the newly computed values).
   - Ungroup the data to proceed with further processing.

Merging the Telemetry Feature Sets

After computing the rolling aggregates, the next step is to merge the resulting data frames into a final feature set for telemetry. The challenge is that:

- **Pair 1:** Data frames containing the 3-hour rolling aggregates (telemetrymean and telemetrysd) have the same datetime values and number of rows.
- **Pair 2:** Data frames with the 24-hour rolling aggregates (telemetrymean_24hrs and telemetrysd_24hrs) also match in size and datetime.

Merging Strategy:

1. **First Step: Merge Each Pair Separately**
    - Merge the two data frames of the same dimension (e.g., merge telemetrymean with telemetrysd).
    - Similarly, merge the 24-hour pair.
2. **Second Step: Join the Two Merged Data Frames**
    - The two merged data frames (say, telemetryfeat and telemetryfeat_24hrs) might have different sizes and datetime values.
    - Use the left_join() function in R to merge these data frames:

left_join(x, y, by = "datetime")

        - **x:** The data frame with the larger number of rows (e.g., telemetryfeat with 292,000 rows).
        - **y:** The other data frame (telemetryfeat_24hrs).
    - **Result:** The joined data frame retains all rows from x and includes columns from y where matching datetime values exist. Rows without a match in y will have NA in the joined columns.
3. **Final Filtering:**
    - Apply a filtering function to remove rows with NA values specifically in the telemetryfeat_24hrs columns.
    - This step ensures that the final feature set is clean and consistent.

Feature Engineering for Errors

Unlike telemetry data, the errors data come with timestamps that use different time units and do not have a fixed sampling period. The goal is to capture the **long-term history** of errors (since a short-term view is less meaningful when errors occur sporadically over days or several hours apart) and to align the errors data with the telemetry data that has been aggregated every 3 hours.

Objectives

1. **Capture Long-Term History:**
    - **Solution:** Apply a rolling aggregate over a 24-hour lag window.

- **Aggregation Function:** Since errorID is a categorical feature, instead of computing means or standard deviations, count the number of occurrences for each error type within the window.

2. **Align Datetime Values:**
    - Ensure that the errors data matches the datetime structure of the final telemetry dataset (i.e., one record every 3 hours).

Steps to Engineer Features for Errors

1. **Transform the Structure:**
    - **Create Binary Columns:**
      For each error type, use a function like mutate() to generate new integer columns (e.g., error1, error2, error3, error4, error5).
        - **Value Assignment:**
            - 0 if an error of that type does not occur.
            - 1 if an error of that type occurs.
2. **Aggregate by Datetime and Machine:**
    - **Combine Multiple Records:**
        - If there are several rows for the same timestamp and machine, use the summarise() function to sum the values across these rows.
        - **Grouping Variables:**
          Group by machineID and datetime so that each group reflects the total count of each error type at that timestamp.
3. **Align Timestamps with Telemetry Data:**
    - **Extract Alignment Keys:**
        - From the original telemetry data (sampled every hour), extract the datetime and machineID columns.
    - **Left Join:**
        - Perform a left_join() between these extracted columns and the summarized errors data.
        - This step ensures that the errors data is aligned to the same hourly timestamps as the telemetry data.
4. **Apply Rolling Aggregation:**
    - **Rolling Window Configuration:**
        - **Window Size:** 24 hours
        - **Step Size:** by = 3 hours to match the final telemetry feature set.
    - **Aggregation Function:**
        - Sum the error counts for each error type over the rolling window.

- **Outcome:**
    - You obtain, for every 3-hour interval, the total number of each error type recorded over the previous 24 hours.

Feature Engineering for Maintenance

To extract meaningful features from maintenance records, we leverage the **data.table** R package for efficient data manipulation. Maintenance data capture component replacements (both scheduled and breakdown-driven) from 2014 and 2015. Given that the time intervals are not constant and there can be multiple replacements at the same timestamp, special techniques such as rolling joins are applied.

Key Functions and Concepts

- **data.table Package Functions:**
    - **as.data.table():** Convert a data frame into a data table. This function returns a copy of the original data.
    - **setkey():** Sorts the data table by specified columns (e.g., machineID and datetime). Sorting is done in ascending order by default.
    - **Filtering and Column Operations:** Functions to filter, select, and add or swap columns.
- **Rolling Joins:**
    - **Purpose:** Used for time series analysis to join two data tables so that each element in one table is matched with the most recent (closest previous) record in another.
    - **Parameter:** roll = TRUE (rolling forward) ensures that the join picks the last available record from the replacement table.

Steps for Feature Engineering from Maintenance Data

1. **Create Binary Indicator Columns:**
    - For each component type, add a new binary column indicating if a replacement occurred:
        - Value 1 if a replacement occurred.
        - Value 0 if not.
    - Remove the original comp column after this transformation.
2. **Convert and Sort the Data Table:**
    - Convert the maintenance data frame into a data table using as.data.table().
    - Sort the data table by machineID and datetime to ensure proper chronological ordering.
3. **Split the Data Table by Component Type:**

- Create separate data tables for each of the four components (e.g., comp1rep, comp2rep, etc.).
- Each table should include:
  - **machineID**
  - **datetime**
  - A new column (e.g., lastrepcomp1) representing the datetime of the replacement for that component.

4. **Prepare the Reference Time Table:**
   - From the final telemetry dataset, extract the columns datetime and machineID.
   - Create a new data table (e.g., compdate) that is sorted by machineID and datetime. This table serves as the reference for aligning maintenance events with telemetry and error data.
5. **Apply Rolling Joins:**
   - Perform a rolling join between compdate and each component-specific replacement table.
   - **Outcome:** Each record in compdate will be matched with the most recent (closest previous) replacement record for that component.
6. **Calculate Days Since Last Replacement:**
   - For each joined table, add a new column that calculates the time difference (in days) between the datetime in compdate and the matched replacement datetime (lastrepcompX).
7. **Merge the Results:**
   - Merge the four separate component tables to form a complete maintenance feature set.
   - Convert the final merged data table back into a data frame.
   - **Result:** A feature set (compfeat_final) with the same time units as the final telemetry and error datasets.

Feature Engineering for Machine Data

- **Machine Features:**
  - Use the machine metadata (e.g., model type and age in years) directly without further modifications.
  - This dataset provides descriptive attributes for the 100 machines.

Final Feature Matrix

To build the comprehensive feature matrix for the predictive maintenance model, merge the following datasets:

- **telemetryfeat_final:** Processed telemetry data with rolling aggregates (e.g., 291,300 rows).
- **errorfeat_final:** Processed error data aligned to the same 3-hour time units.
- **compfeat_final:** Maintenance-derived features (days since last replacement for each component) aligned with the telemetry data.
- **machines:** Machine metadata (100 rows).

Merging Process

- **Left Join:**
  Use left_join() to merge the datasets.
    - **Note:** The machines data set is merged differently due to its distinct structure.
- **Outcome:**
  The final feature matrix combines all the engineered features into one data frame (finalfeat) ready for further modeling.

Data Labelling

Introduction

- **Prediction Window:**
  A prediction window is defined to alert the system 24 hours before a failure occurs.
- **Objective:**
  Add a new label column to the final feature set (i.e., finalfeat) so that the machine learning model can learn to predict the type of failure (if any) for each timestamp.
- **States:**
  A state-driven labelling approach is used, with two primary states:
    - **Normal State:** Labelled as none (or 0), indicating no imminent failure.
    - **Failure State:** Labelled as one of comp1, comp2, comp3, or comp4, indicating a specific component failure.
- **Handling Overlapping Failures:**
  If two failures of different types occur at the same time, the prediction window will contain duplicate rows for that timestamp—with the same features but different labels.

Labelling Process Steps

1. **Initial Join with Failure Data:**
    - **Action:** Use a left_join() to merge the transformed final features dataset (finalfeat_t) with the failures dataset (failures_t), joining on machineID.

- **Purpose:** Incorporate failure event information (from the failures data frame) into the feature set.

2. **Compute Time Difference:**
   - **Action:** Add a new column that calculates the time difference between each record's timestamp and the corresponding failure event's timestamp.
   - **Purpose:** This time difference indicates how far a record is from a failure event.
3. **Filter by Prediction Window:**
   - **Action:** Filter the rows to retain only those where the time difference falls within the prediction window.
     - **Note:** Although the introduction states a 24-hour window, the provided instructions mention filtering for a 7-hour difference. Confirm the appropriate window length—if using a 24-hour window, adjust the filtering accordingly.
   - **Purpose:** Identify and label records that are within the window in which a failure is expected to occur.
4. **Merge Label Information:**
   - **Action:** Perform a second left_join() between the original features dataset (finalfeat_t) and the filtered failure information (using both datetime and machineID as keys).
   - **Purpose:** Attach the corresponding failure label (e.g., comp1, comp2, etc.) to each record that falls within a prediction window.
5. **Assign Default (Normal) Label:**
   - **Action:** For records where no failure information was joined (i.e., resulting in NA), replace the NA values with the default label none (or 0).
   - **Purpose:** Clearly indicate that no failure is predicted for these records.

Outcome

- **Final Labeled Dataset:**
  The resulting dataset (often referred to as labeledfeatures) contains an added column with the failure labels. Due to overlapping failure events (e.g., two different failures occurring at the same time), the labeled dataset might contain more rows than the original finalfeat dataset. Each row now indicates whether it falls within a prediction window for a specific component failure or if it represents normal operation.

Training, Modelling, and Evaluation

1. Data Splitting: Time-Dependent Partitioning

- **Data Origin:**
  All training and testing instances are derived from the fully preprocessed and labelled dataset (labeledfeatures). This dataset combines historical data, engineered features, and data labels.

- **Time-Ordering Requirement:**
  Feature instances are sorted according to their timestamps. This ordering ensures that the model is trained on past data and tested on future data, which is crucial for time-series analysis.
- **Exclusion of Specific Features:**
  Information such as Datetime and MachineID is excluded from the model training process because they are either time stamps or too asset-specific.
- **Prediction Window Handling:**
  Records that fall within the prediction window prior to the training cutoff point are removed from the training set. This avoids any leakage from future information.
- **Scenario-Based Splits:**
  Three split scenarios are considered to compare performance:
    - **Scenario 1:** Split at 2015-08-01 01:00:00
        - Train on the first 7 months, test on the last 5 months.
    - **Scenario 2:** Split at 2015-09-01 01:00:00
        - Train on the first 8 months, test on the last 4 months.
    - **Scenario 3:** Split at 2015-10-01 01:00:00
        - Train on the first 9 months, test on the last 3 months.

---

2. Model Training with Gradient Boosting Machine (GBM)

- **Chosen Algorithm:**
  The model is built using the **GBM** package available on CRAN. This implementation of the Gradient Boosting Machine is well-suited for multi-class classification tasks.
- **Model Parameters:**
    - **Formula:**
      Define the model using a formula in R, excluding Datetime and MachineID.
      Example:

formula = Response ~ Feature1 + Feature2 + Feature3 + ...

    - **Distribution:**
      Since this is a multi-class problem, set the distribution to "multinomial".
    - **n.trees:**
      Specify the total number of trees to be fitted. Increasing the number of trees can reduce training error but might lead to overfitting.
    - **interaction.depth:**
      Controls the maximum number of splits (tree depth). A default value is 1, but this may be tuned.

- **shrinkage (Learning Rate):**
  Typically set between 0 and 1. Lower values (e.g., below 0.01) may improve performance but slow the learning process.

- **Training Outcome:**
  The trained model is stored in a variable (e.g., gbm_model) for further prediction and testing.

---

3. Making Predictions

- **Predict Function:**
  Once the model is trained, predictions on the test dataset are made using:

```
predictions <- predict(gbm_model, newdata = testindata, n.trees = n_trees, type = "response")
```

  - **Parameters:**
    - gbm_model: The trained GBM model.
    - testindata: The test dataset.
    - n.trees: The number of trees to use for prediction (must be ≤ the number used in training).
    - type: "response" to get predicted probabilities.

- **Prediction Outcome:**
  The output is a vector of probabilities for each class. The predicted class for each instance is the one with the highest probability.

---

4. Model Evaluation Metrics

- **Evaluation is Key:**
  Proper evaluation metrics help understand the performance of the model on unseen data.

- **Common Evaluation Metrics:**
  - **Confusion Matrix:**
    Provides a detailed breakdown of actual versus predicted classifications.
  - **Accuracy:**
    The overall proportion of correct predictions.
  - **Precision:**
    The fraction of true positive predictions among all positive predictions.
  - **Recall (Sensitivity):**
    The fraction of true positives detected among all actual positives.
  - **F1 Score:**
    The harmonic mean of precision and recall.

- **Averaging for Multi-Class:**
  Metrics like precision, recall, and F1 can be computed for each class and then macro-

averaged for an overall measure. Alternatively, one-versus-all confusion matrices can be used to assess performance for each class separately.

- **Final Assessment:**
  The selected metrics allow for a detailed comparison between different split scenarios and model parameter configurations to ensure robust predictive performance.

Conclusion

In summary, this experiment demonstrates a comprehensive approach to predictive maintenance by integrating multiple data sources—telemetry, error logs, maintenance records, and machine metadata—into a unified framework. Starting with rigorous data preprocessing and transformation, we engineered meaningful features through rolling aggregates and advanced grouping techniques, tailored individually for telemetry, errors, and maintenance datasets. The state-driven data labeling method allowed us to accurately mark failure events within a 24-hour prediction window, even handling overlapping failure instances.

Subsequently, we applied time-dependent splitting of the labeled dataset to ensure that our training and testing sets reflected the natural temporal order, thereby preventing data leakage. Leveraging a Gradient Boosting Machine (GBM) model, we trained on historical data while excluding asset-specific identifiers such as timestamps and machine IDs. The model's performance was evaluated using metrics such as confusion matrices, accuracy, precision, recall, and F1 scores, which confirmed its robustness in predicting component failures.