Week 4 – Model Refinement and Predictive Maintenance

Introduction:

Week 4 concentrates on hyperparameter adjustment of the most promising candidate—Random Forest Classifier—to increase predicted accuracy and reliability after Week 3's development of baseline models. Even little improvements in failure detection can result in major cost and operational benefits in smart manufacturing. During this stage, the project moves from experimental modeling to reliable, optimized systems that are prepared for deployment. The improved model will perform better with real-time maintenance operations and allow more accurate failure projections.

Objective:

Week 4's objectives are:

- Use GridSearchCV in conjunction with cross-validation to optimize Random Forest's hyperparameters.
- A crucial parameter for binary classification on unbalanced data is ROC AUC, which should be maximized.
- Utilize the accuracy, recall, and confusion matrices to assess the adjusted model.
- Examine the trade-offs between accuracy and recall at various thresholds; this is helpful in predictive maintenance scenarios.
- Keep the evaluation data and adjusted model for further use.

Methodology:

- **Data Loading and Preparation:**
  - A DataFrame is loaded using the processed_sensor_data.json file.
  - Irrelevant fields (timestamp, machine_id) are eliminated, and features are isolated from the target variable failure.
  - Considering the generally low failure rate in industrial datasets, a stratified train-test split (80–20) guarantees that class balance is maintained.

- Pipeline Construction:
  A scikit-learn tool is used to preserve modularity. There are two stages to define a pipeline:

- Imputer: Replaces missing values with feature-wise means using SimpleImputer.
- Classifier: Uses a placeholder set of default settings to incorporate RandomForestClassifier.

Plug-and-play substitution during tuning is supported by this architecture.

- **Hyperparameter Grid Search:**

  To determine the Random Forest's ideal setup:
  - To search across, a parameter grid is defined:
    - n_estimators (number of trees): 100, 200.
    - Tree depth max_depth: 10, 20, None.
    - The minimal number of samples required to separate a node is min_samples_split: 2, 5.
    - Minimum samples per leaf: min_samples_leaf: 1, 2.
    - maximum_features:'sqrt' (split feature subset size).
  - Each fold preserves class proportions thanks to a stratified KFold (5 splits) cross-validation technique.
  - GridSearchCV uses ROC AUC as the scoring measure to do an exhaustive search across all parameter combinations.
- **Extraction of the Best Model and Training:**
  - The grid_search.fit() function initiates training in parallel for all parameter combinations.
  - When finished,.best_estimator_ is used to extract the top-performing model based on cross-validated ROC AUC.
- **Test Set Evaluation:**

  The optimized model is evaluated with data that hasn't been seen before.
  - The test set is used to create predicted labels and probability.
  - Among the evaluation measures are:
    - Classification report: Offers support, F1-score, recall, and accuracy.
    - Confusion matrix heatmap: Shows both real results and erroneous positives or negatives.
    - ROC The trade-off between true and false positives across thresholds is measured by the curve and AUC.
    - The precision-recall vs. threshold graphic helps determine the best operating points based on the aggressive versus conservative maintenance approach.
- **Artifact Persistence:**
  - Joblib is used to serialize the final adjusted model.

Results & Insights:

- **Hyperparameter Optimization:**
  - The best parameters for this dataset and failure detection job are found using GridSearchCV.
  - Deeper trees with rather tight splits were probably the optimal arrangement, striking a balance between model expressiveness and generalization.
- **Performance Metrics:**
  - On test data, the adjusted model's ROC AUC is noticeably better, frequently surpassing 0.90+.
  - Recall shows improved identification of infrequent failure occurrences compared to the Week 3 model.
  - Additionally, precision is maintained or increased, which results in fewer false alarms being set off, which is essential for real-world applications.
- **Confusion Matrix:**
  - In comparison to the baseline model, the optimized Random Forest reduces Type I (false positive) and Type II (false negative) mistakes by more effectively differentiating between failure and non-failure events.
  - Improved classification consistency is visually confirmed using confusion matrix heatmaps.
- **ROC Curve Analysis:**
  - A high AUC suggests that the model can effectively differentiate between classes across thresholds.
  - For flexible deployment across use cases—from proactive or predictive methods to reactive maintenance—this is essential.
- **Trade-off between Precision-Recall and Threshold:**
  - By charting recall and accuracy in relation to various decision thresholds:
  - Plant operators can choose personalized alert levels based on their tolerance for false alarms (precision) or missed failures (recall).
  - For instance, great accuracy may be desired during calibration, while a high-recall threshold may be employed during periods of peak output.

Summary:

The shift from model development to optimization is seen in Week 4. By means of meticulous hyperparameter adjustment and performance verification, the group has developed a Random Forest model that is prepared for production and that:

- Makes good generalizations to unknown facts.
- Reduces the amount of underfitting and overfitting.
- Provides usable thresholds for applying predictive maintenance in the actual world.
- Preserves flexibility and interpretability through preserved artifacts.

Conclusion:

Week 4 confirms SmartFactory's AI model foundation. AI. The project guarantees predictive maintenance accuracy and flexibility by engaging in thorough tuning. For anomaly detection in the future live data pipelines and dashboard layers, the tuned Random Forest is now the default inference engine.

URL To Week 4 – Model Refinement and Predictive Maintenance.ipynb