Process Mining and Management Project

Davide Donà - 264467 Andrea Blushi - 264468

January 2026

Contents

1 Introduction

In modern business processes, predicting outcomes and providing recommendations based on historical data is crucial for improving efficiency and decision-making. In this project, we use process monitoring and machine learning techniques to analyze a real-life production log and predict whether a process instance will lead to a positive outcome (fast trace: cycle-time < avg-cycle-time) or a negative outcome (slow trace). When a negative outcome is predicted, we generate recommendations that suggest changes in the sequence of activities to steer the process towards a positive result. We then evaluate the effectiveness of our recommendation system by comparing the predicted outcomes with the actual outcomes in a test dataset.

2 Implementation

A recommendation system was implemented using a decision tree classifier to analyze production event logs. The implementation was achieved in Python, leveraging libraries such as *PM4Py* for process mining tasks and *scikit-learn* for machine learning functionality. The execution flow of our implementation resides within the *notebook.ipynb* file, which calls all the designed utility functions.

2.1 Encoding the Event Log

We began by importing the event log file, which is in *XES* format, using the *PM4Py* library. We then converted this file into a *PM4Py* event log object suitable for subsequent process mining tasks.

A pruned copy of the event log was created to consider only the prefixes of each trace. To achieve this, each trace in the log was truncated to *prefix_length* events. Since the partial ongoing trace in a real-world scenario is incomplete, we trained the classifier exclusively on the prefixes of the traces.

Next, we extracted the unique activity names from the complete event log to serve as columns (features) for the encoding step. In addition to these, we included the trace identifier, the prefix length, and the ground truth label, which indicates whether the trace outcome was positive or negative.

To prepare this data for training a decision tree classifier, the pruned events were encoded using a *Boolean encoding* scheme. This encoding involved creating binary features for each unique activity name. For each trace, if an activity was present within the trace, the corresponding feature in that row was set to true (1); otherwise, it was set to false (0). This approach allows for the creation of a feature set with a fixed number of columns, eliminating the need to pad traces to a uniform length. Upon completion, we obtained a *DataFrame* in which each row represented a trace, while the feature columns indicated the presence or absence of specific activities within that trace.

2.2 Training the Decision Tree Classifier

Following the encoding of the pruned event log, a decision tree classifier was trained using the *scikit-learn* library. We separated the encoded log into features ($\boldsymbol{X}$), which were derived by removing the trace identifier and label columns, and labels ($\boldsymbol{y}$), which represented the ground truth extracted directly from the log. The dataset had been pre-partitioned in the original *XES* file into distinct training and testing sets; these partitions were used to train and evaluate the classifier, respectively.

After training the classifier, we visualized the learned decision rules by plotting the decision tree using built-in *scikit-learn* functionality.

2.3 Optimizing the Classifier Hyperparameters

To enhance the performance of our decision tree classifier, we employed the *Hyperopt* library to perform hyperparameter optimization. This tool systematically explores a defined search space, iteratively testing combinations of critical hyperparameters—such as *max_depth*, *max_features*, and the split criterion (e.g., *Gini* or *entropy*)—to identify the optimal configuration. During each trial, the model was trained and evaluated using the F_1-score as the primary performance metric. Only the configuration that resulted in the highest F_1-score was retained for the final model. The *Hyperopt* library also allowed us to define a search space and to specify the maximum number of iterations to perform.
In order to achieve reproducible results, a fixed random seed was set for all stochastic processes involved in the optimization and training phases.

2.4 Generating Predictions

After training, the optimized classifier was used to generate predictions on the test set. The predicted labels were obtained by applying the trained model to the feature set derived from the encoded test log. Similar to the training phase, the features ($\boldsymbol{X}_{test}$) were prepared by removing the trace identifier and ground truth label columns. The classifier's performance was subsequently evaluated using standard scikit-learn metrics, including *Accuracy*, *Precision*, *Recall*, and the *F1-score*.

2.5 Extracting Recommendations

Given the interpretability inherent in the decision tree structure, the model can provide proactive recommendations specifically for traces predicted to result in a negative outcome. The process for generating these recommendations involved three main steps:

1. **Extracting Positive Paths**: identify all distinct paths in the decision tree that lead to a leaf node associated with a positive outcome.
2. **Filtering Compliant Paths**: for a given negatively-predicted trace prefix, filter the positive paths to find those that are compliant (i.e., whose conditions are not violated by the activities already present in the trace).
3. **Generating Recommendations**: from the compliant positive paths, select the one with the highest confidence score. Determine the set of activities required to complete that path.

Note that the function signature differs from the originally requested one. In particular, the *class_values* parameter was removed to simplify the implementation, since our setting is restricted to binary classification only.

Extracting Positive Paths from the Decision Tree: To extract all positive paths from the decision tree, we traversed the tree structure starting from the root node using a *Depth-first search* (DFS) approach. At each decision node in the path, we recorded the feature, the operator (e.g., $\leq$, $>$), and the threshold that leads to the next node.
Upon reaching a leaf node, if the node's label corresponded to a positive outcome, the entire path, along with its associated confidence score, was stored as a positive path.
A weighted confidence score was introduced for decision tree paths to mitigate the issue of paths achieving a confidence of 1 with only a single supporting instance. The weighting was based on the number of samples reaching each node, ensuring that recommendations were not derived from statistically unrepresentative outlier paths.

Algorithm 1 Get Positive Paths

```
1: function GetPositivePaths(node, current_path, positive_paths)
2:     if node is leaf then
3:         if node.label == positive then
4:             Add (current_path, node.confidence) to positive_paths
5:         end if
6:     else
7:         Append (node.feature, $\leq$, node.value) to current_path
8:         Call GetPositivePaths(node.left, current_path, positive_paths)
9:         Remove last element from current_path
10:
11:        Append (node.feature, $>$, node.value) to current_path
12:        Call GetPositivePaths(node.right, current_path, positive_paths)
13:        Remove last element from current_path
14:    end if
15: end function
```

Filtering Compliant Paths: After all positive paths were extracted, for each prefix trace predicted as negative, we filtered the set to retain only the *compliant* paths. A path is defined as compliant if and only if none of the feature conditions along the path are violated by the set of activities currently present in the prefix trace. Among all compliant paths, the one with the highest confidence score was selected. In case of a tie in confidence, the shortest path was chosen as the tie-breaker.

Generating Recommendations: Once the most suitable positive compliant path was identified, recommendations were generated by determining the activities that needed to be added to the current trace to follow the selected path. For each test-set prefix, the recommendation set is defined as:

- *None* if the trace was already predicted as positive;
- The empty set if no compliant positive path was found;
- Otherwise, the set of activities corresponding to the conditions in the selected positive path that were not satisfied by the current prefix trace.

2.6 Evaluation of Recommendations

To evaluate the quality of the generated recommendations, we compared them against the actual full traces in the test set, using the encoded test set as ground truth. For each trace in the test set, we matched it with its corresponding prefix and the recommendation generated for that prefix.

If a recommendation was defined as *None*, it was skipped, as this indicates that no action was necessary because the trace was already predicted to have a positive outcome. Conversely, if a recommendation was an empty set, it was considered *not followed*, since the system was unable to suggest any corrective action for the negative prediction. A recommendation was considered *followed* if and only if all the associated Boolean conditions were satisfied in the full trace. Each condition specifies whether a given activity should be present or absent; if any single condition was violated, the recommendation was marked as *not followed*.

The recommendation outcome (followed or not followed) was then compared with the ground truth label of the trace to derive an approximate confusion matrix defined as follows:

- **True Positives**: the recommendation was followed in the actual trace and the ground truth outcome is positive.
- **False Positives**: the recommendation was followed in the actual trace but the ground truth outcome is negative.

- **True Negatives**: the recommendation was not followed in the actual trace and the ground truth outcome is negative.
- **False Negatives**: the recommendation was not followed in the actual trace but the ground truth outcome is positive.

Based on this confusion matrix, we computed standard evaluation metrics, such as accuracy, precision, recall, and F1-score, to assess the effectiveness of the generated recommendations.

3 Evaluation

In this section, we present the evaluation results for both *prefix_length* = 5 and *prefix_length* = 11 of the decision tree classifier and the recommendation system built on top of it.

3.1 Hyperparameters

Tables 1 and 2 show the optimized hyperparameters obtained through *Hyperopt* for both prefix lengths.

For *prefix_length* = 5, the optimizer selects a shallow tree with a maximum depth of 3 and a small number of features considered at each split. This choice limits overfitting in a setting where only partial information about the trace is available, while still allowing the model to exploit the most informative activities. The use of the entropy criterion leads to more balanced and interpretable decision paths.

For *prefix_length* = 11, the optimal configuration still constrains the maximum depth to 3, confirming that deeper trees do not provide additional benefits even when more information is available. However, the maximum number of features increases, indicating that longer prefixes allow the model to effectively leverage a richer set of activities. In this case, the Gini criterion is selected, suggesting that it better captures class separation when more contextual information is present.

Hyperparameter	**Value**
Criterion	Entropy
Max Depth	3
Max Features	2

Table 1: Optimized hyperparameters for the decision tree classifier for *prefix_length* = 5.

Hyperparameter	**Value**
Criterion	Gini
Max Depth	3
Max Features	3

Table 2: Optimized hyperparameters for the decision tree classifier for *prefix_length* = 11.

3.2 Decision Tree Structure

Figures 1 and 2 illustrate the structure of the trained decision trees. The color-coded leaf nodes (blue for positive outcomes, orange for negative outcomes) clearly show the distribution of predictions across different decision paths. Both trees maintain interpretability thanks to their depth, as the controlled *max_depth* and *max_features* parameters prevent excessive branching. The tree structures provide valuable insights into which activities are most discriminative for predicting case outcomes, making the model transparent.

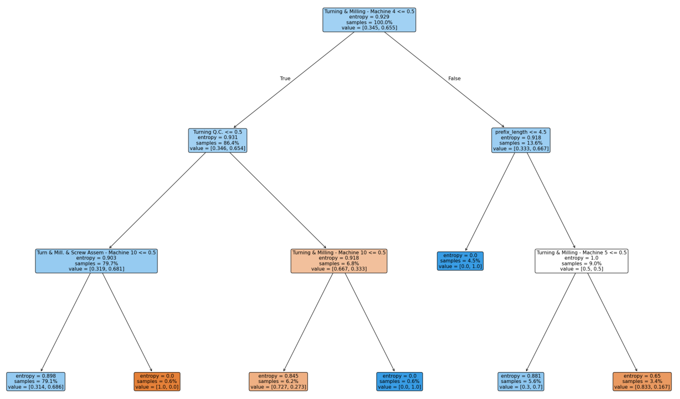

Figure 1: Visualization of the trained decision tree classifier for *prefix_length* $= 5$. Each node represents a decision based on the presence or absence of specific activities, leading to recommendations at the leaf nodes.

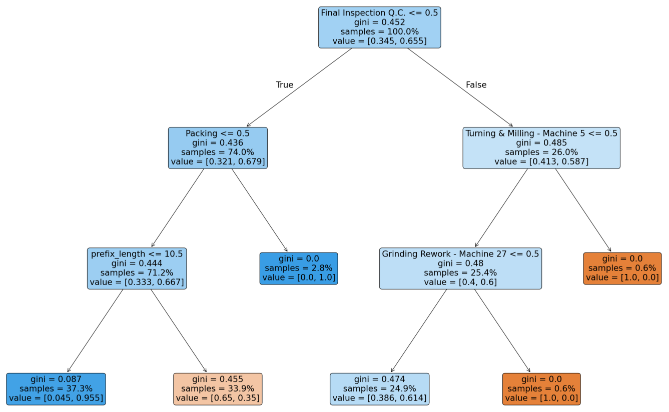

Figure 2: Visualization of the trained decision tree classifier for *prefix_length* = 11.

3.2.1 Evaluation Metrics

The classification results in Tables 3 and 4 demonstrate the predictive performance of our decision tree models.

For *prefix_length* = 5, these results indicate reasonably balanced performance when only a limited portion of each trace is available. In this setting, the model is already able to capture meaningful patterns, but the reduced amount of information limits its power.

The performance further improves for *prefix_length* = 11, where the results suggest that longer prefixes provide more informative context, enabling the model to make more confident positive predictions. Overall, the results confirm that increasing the *prefix_length* has a positive impact on outcome prediction, as more activities contribute to defining the case trajectory.

The confusion matrices in Figures 3 and 4 provide further insight into the distribution of predictions and misclassifications.

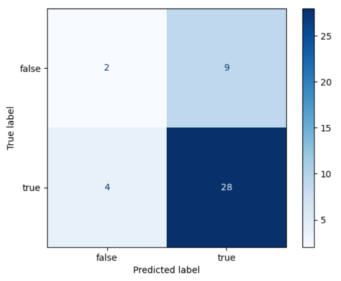

Figure 3: Confusion matrix of predictions in the test set ($prefix\_length = 5$).

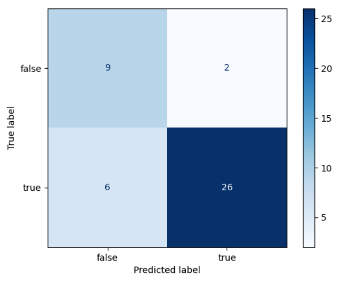

Figure 4: Confusion matrix of predictions in the test set ($prefix\_length = 11$).

Metric	**Value**
Accuracy	69.77%
Precision	64.84%
Recall	69.77%
F1-Score	66.42%

Table 3: Evaluation metrics for the decision tree classifier on the test set for $prefix\_length = 5$.

Metric	**Value**
Accuracy	81.40%
Precision	84.45%
Recall	81.40%
F1-Score	82.21%

Table 4: Evaluation metrics for the decision tree classifier on the test set for $prefix\_length = 11$.

3.3 Recommendation Analysis

Recommendation Generation For the sake of simplicity in explaining the recommendation extraction process, we chose to consider the tree in Figure 5.

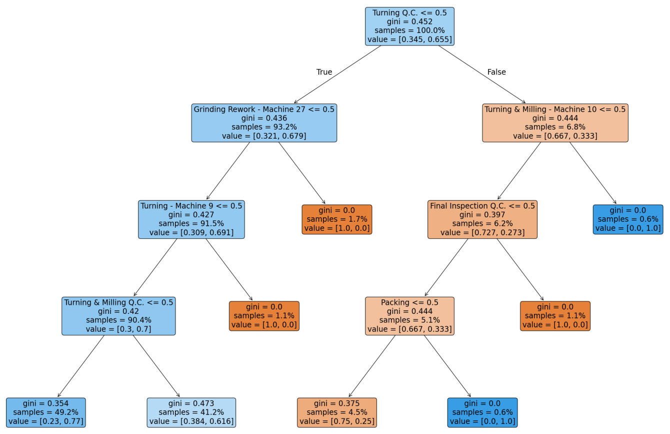

Figure 5: Simplified decision tree used to illustrate the recommendation extraction process. Given a node, the left child branch is taken if the condition is true (activity is absent in the trace), while the right child branch is taken if the condition is false (activity is present in the trace).

Applying the extraction of positive paths (Algorithm 2.5) to this tree, we obtain the following positive paths with their confidence scores:

- Path 1: *Turning Q.C.* = F $\wedge$ *Grinding Rework - Machine 27* = F $\wedge$ *Turning - Machine 9* = F $\wedge$ *Turning & Milling Q.C.* = F with confidence 0.77;
- Path 2: *Turning Q.C.* = F $\wedge$ *Grinding Rework - Machine 27* = F $\wedge$ *Turning - Machine 9* = T with confidence 0.80;
- Path 3: *Turning Q.C.* = T $\wedge$ *Turning & Milling - Machine 10* = F $\wedge$ *Final Inspection Q.C.* = F $\wedge$ *Packing* = T with confidence 1.0;
- Path 4: *Turning Q.C.* = T $\wedge$ *Turning & Milling - Machine 10* = T with confidence 1.0;

Now, suppose a prefix trace that has the following activities: {*Final Inspection Q.C.*: True, *Packing*: True, *Grinding Rework - Machine 27*: True}, represented by the orange path in Figure 5, is predicted as false. To find the recommendations for this trace, we first filter the positive paths to find the compliant ones.

- Path 1 and Path 2 are not compliant because they require *Grinding Rework - Machine 27* = F, which is violated by the trace;
- Path 3 is compliant because none of its conditions are violated by the trace;

- Path 4 is also not compliant because it requires *Final Inspection Q.C.* = F, which is violated by the trace.

Given this, we can derive the recommendations for the given prefix trace as the activities that need to be added to follow Path 3. Since both *Turning Q.C.* and *Turning & Milling - Machine 10* need to be true to follow Path 3, the recommended activities will be those. This is illustrated in Figure 6, with the green dotted nodes.

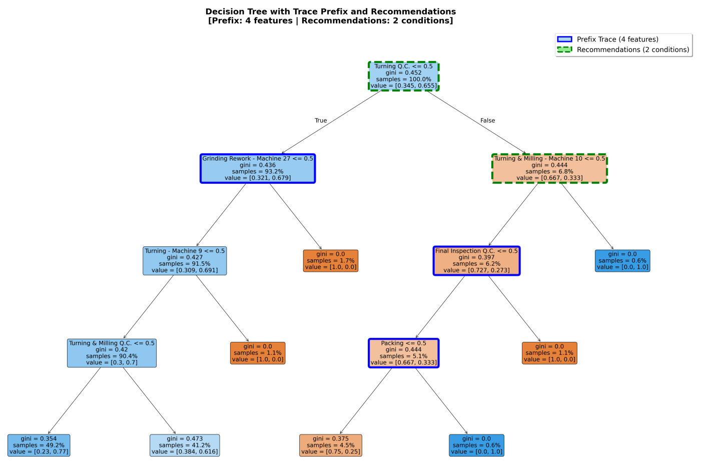

Figure 6: The blue squared nodes represent the activities present in the given prefix trace. The green highlighted nodes represent the recommended activities.

Recommendations Quality Tables 5 and 6 present the evaluation of our recommendation system. These values have to be considered as an approximation, since the ground truth is based on actual traces rather than hypothetical scenarios where recommendations were followed.

Overall, the results indicate that the limited size of the test dataset influences the number of recommendations and, consequently, their statistics.

For *prefix_length* = 5, we can see that we achieve 0% across all metrics, and the few recommendations generated were considered not followed. This outcome suggests that with very short prefixes, the system struggles to provide effective recommendations, likely due to insufficient context about the trace. They then consistently failed to align with the actual trace outcomes. At this early stage of process execution, the activities observed may not yet be discriminative enough to distinguish between successful and unsuccessful process paths, resulting in recommendations that, while structurally valid according to the decision tree, do not reflect the actual dynamics of successful traces.

At *prefix_length* = 11, the recommendation system demonstrates substantial improvement across all metrics compared to shorter prefix lengths. The improved performance at this prefix length confirms that the recommendation system requires substantial contextual information to reliably identify patterns that distinguish successful from unsuccessful process executions and translate them into actionable recommendations. The higher number of recommendations followed indicates that the system is better able to suggest changes that align with actual successful outcomes in the traces.

Metric	**Value**
Accuracy	0.00%
Precision	0.00%
Recall	0.00%
F1-Score	0.00%

Table 5: Evaluation metrics for the recommendation system on the test set for *prefix_length* = 5.

Metric	**Value**
Accuracy	46.67%
Precision	60.00%
Recall	33.33%
F1-Score	42.86%

Table 6: Evaluation metrics for the recommendation system on the test set for *prefix_length* = 11.

For a more detailed understanding of the system's performance, we present the confusion matrices in Figures 7 and 8.

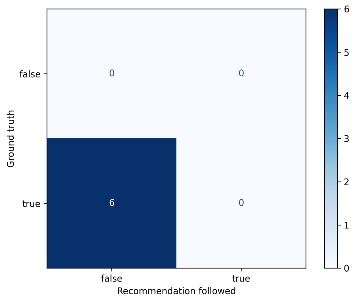

Figure 7: Confusion matrix of recommendations (*prefix_length* = 5).

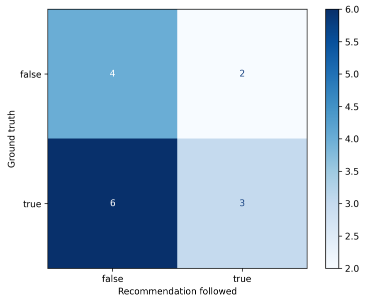

Figure 8: Confusion matrix of recommendations (*prefix_length* = 11).

3.4 Performance by Prefix Length

As shown in Figure 9, the F1-score for the decision tree classifier remains relatively stable across different prefix lengths, while the recommendation system shows a more pronounced improvement as the prefix length increases. This trend reveals a fundamental difference between the two components: while the classifier can maintain consistent predictive performance regardless of prefix length, the recommendation system requires a minimum threshold of contextual information before it can generate meaningful recommendations.

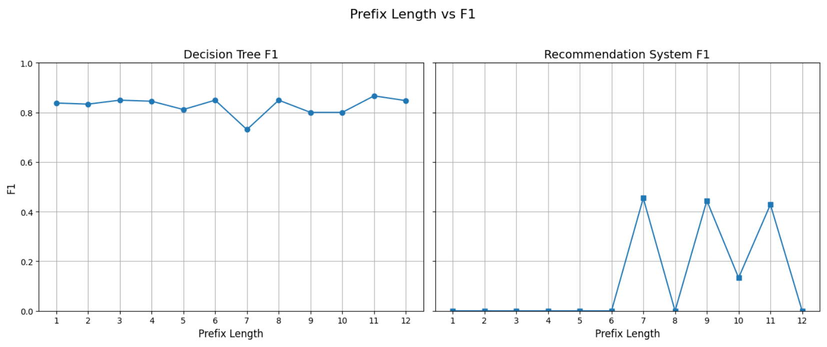

Figure 9: The box plots illustrate the distribution of F1-scores across different prefix lengths for both the decision tree classifier and the recommendation system.

4 Conclusion

Given the limited number of samples in our dataset, the results should be considered preliminary. Nevertheless, they are promising and suggest that, with the availability of more data, the proposed recommendation system could become a valuable tool for process improvement. Our evaluation indicates that the effectiveness of recommendations is highly dependent on the amount of process context available. Longer prefixes capture more detailed information about the process execution, enabling the recommendation system to generate more accurate and relevant suggestions.

Overall, this project demonstrates the feasibility of combining process mining and machine learning techniques to predict process outcomes and generate actionable recommendations. With further refinement and larger datasets, such systems have the potential to significantly enhance business process management and optimization.