Process Mining and Management Project

Davide Donà - 264467 Andrea Blushi - 264468

December 2025

Contents

1 Introduction

In modern business process, predicting outcomes and providing recommendations based on historical data is crucial for improving efficiency and decision-making. In this project, using process monitoring and machine learning techniques, we analyze a real-life *production* log to predict whether a process instance will lead to a positive (fast trace, cycle-time < avg-cycle-time) or 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. Then, we evaluate the effectiveness of our recommendation system by comparing the predicted outcomes with the actual outcomes in a test dataset.

2 Implementation

We implemented a recommendation system using a decision tree classifier to analyze production event logs. The implementation was achieved using Python, leveraging libraries such as *PM4Py* for process mining tasks and *scikit-learn* for machine learning functionalities. 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 in order to take into consideration only the prefixes of each trace. To do this, each trace in the log was truncated to *prefix_length* events. Since in a real-world scenario the partial ongoing trace is incomplete, we need to train the classifier only on the prefixes of the traces.

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

To use this data for training a decision tree classifier, we encoded the pruned event using a *Boolean encoding* scheme. This encoding involved creating binary features for each unique activity name. For each trace in the event log, 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 to build a feature set with a fixed number of columns, regardless of the varying lengths of the traces. This implies that we don't need to worry about padding the 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

After encoding the pruned event log, a decision tree classifier was trained using the *scikit-learn* library. We split the encoded log into features (by removing the trace identifier and label columns from the original encoding) and labels, which were the ground truth extracted directly from the log. The *XES* file was already partitioned into separate training and testing sets; we used these to train and evaluate the classifier, respectively.

After training the classifier, it was possible to visualize 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 iterates through various combinations of hyperparameters, such as maximum tree depth, maximum features, and criterion (e.g., Gini or entropy), to identify the configuration that yields the optimal performance. At every iteration, 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.

2.4 Generating Predictions

After training the classifier, we used it 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. Those feature were cleaned by removing the trace identifier and ground truth label columns, similar to the training phase. Then we evaluated the classifier's performance using standard metrics such as accuracy, precision, recall, and F1-score, provided by the *scikit-learn* library.

2.5 Extracting Recommendations

Given the predictions obtained from the decision tree classifier, using the transparent structure of the tree, we can provide recommendations for traces predicted to have a negative outcome. To achieve this, we needed to:

1. Extract the positive paths from the decision tree, which are the paths leading to leaf nodes with a positive outcome.

2. For each prefix trace predicted as negative:
    - Filter the positive paths to find the ones that are compliant with the current trace;
    - From the compliant paths, take the one with the highest confidence score;
    - The recommended activities are the ones that need to be added to the current trace to follow the selected positive path.

Extracting Positive Paths from the Decision Tree: To extract the positive paths from the decision tree, we traversed the tree structure starting from the root node in a *depth-first* manner. At each node in the path, we recorded the feature and the boolean condition (true or false) that leads to the next node. When we reached a leaf node, we checked if it corresponded to a positive outcome. If it did, we stored the entire path taken to reach that leaf, with the associated confidence score, as a positive path.

Algorithm 1 Get Positive Paths

```
function GetPositivePaths(node, current_path, positive_paths)
    if node is leaf then
        if node.label == positive then
            Add (current_path, node.confidence) to positive_paths
        end if
    else
        Append (node.feature, False) to current_path
        Call GetPositivePaths(node.left, current_path, positive_paths)
        Remove last element from current_path
        Append (node.feature, True) to current_path
        Call GetPositivePaths(node.right, current_path, positive_paths)
        Remove last element from current_path
    end if
end function
```

Filter compliant paths: After extracting all positive paths from the decision tree, for each prefix trace predicted as negative, we filtered only the *compliant* paths. A path is considered *compliant* with a trace if none of the conditions along the path are violated by the current prefix trace. Among all the compliant paths, we selected the one with the highest confidence score. If two or more paths had the same confidence score, we selected the shortest one.

Generate Recommendations: Once we identified the most suitable positive compliant path, we generated recommendations by determining the activities that needed to be added to the current trace to follow the selected path. For each prefix seen in the test set, its recommendation is defined as follows:

- The empty set if the prefix was predicted as positive or if no compliant positive path was found;
- The set of activities corresponding to the conditions in the selected positive path that were not satisfied by the current prefix trace otherwise.

2.6 Evaluation of Recommendations

To evaluate the quality of the recommendations generated, we compared them against the actual full traces in the test set, where the encoded test set served as the ground truth for comparison.

For each complete trace in the test set, we first checked if the trace's prefix matched the specific conditions leading to a particular recommendation, as defined by the decision tree classifier. Only if a prefix match occurred, we then proceeded to check if the recommended activity was actually performed in the subsequent step of the actual trace following that prefix. This recommendation outcome was then compared with the ground truth outcome of the trace to determine the predictive accuracy of the recommendation. Note that if a matched prefix had an empty recommendation, it was interpreted as no changes being necessary, and thus the recommendation was considered to be followed by default.

With those checks we can create an approximated confusion matrix, where:

- True Positives: The recommended activity was followed in the actual trace, and the ground truth outcome is positive.
- False Positives: The recommended activity was not followed in the actual trace, but the ground truth outcome is positive.
- True Negatives: The recommended activity was not followed in the actual trace, and the ground truth outcome is negative.
- False Negatives: The recommended activity was followed in the actual trace, but the ground truth outcome is negative.

Starting from this confusion matrix, we computed standard evaluation metrics such as accuracy, precision, recall, and F1-score to assess the effectiveness of the recommendations provided by our system.

3 Evaluation

In this section, we present the evaluation results, for both $\textit{prefix\_length} = 5$ and $\textit{prefix\_length} = 10$, 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* = 10, 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 number of maximum 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* = 10.

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.

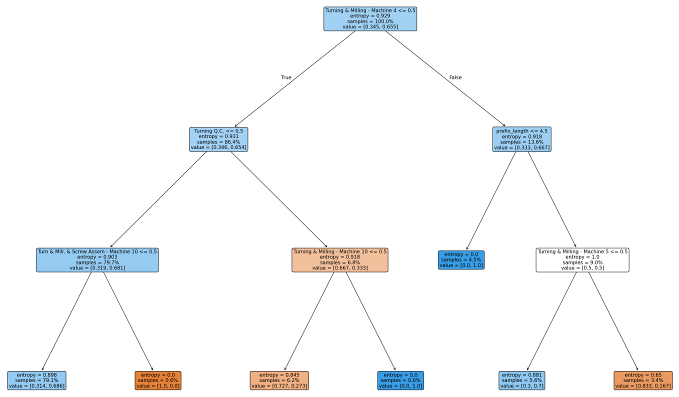

Figure 1: Visualization of the trained decision tree classifier for *prefix_length* = 5. Each node represents a decision based on the presence or absence of specific activities, leading to recommendations at the leaf nodes.

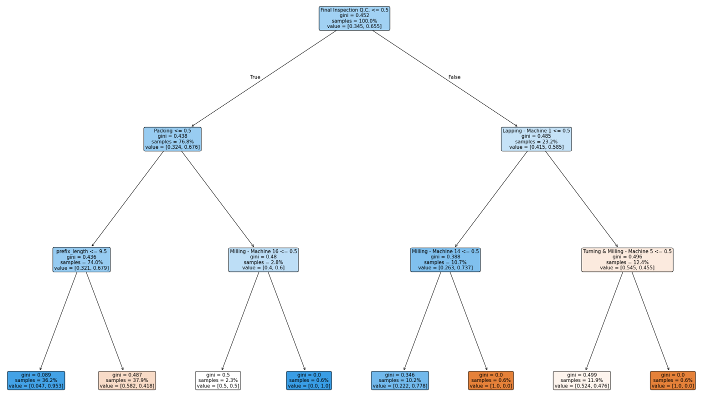

Figure 2: Visualization of the trained decision tree classifier for *prefix_length* = 10.

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 a 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* $= 10$, 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.

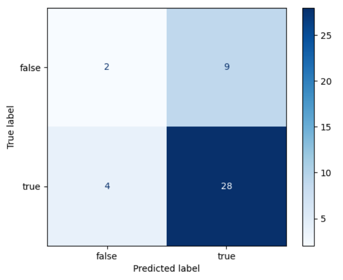

Figure 3: Confusion matrix of predictions in the test set (*prefix_length* $= 5$).

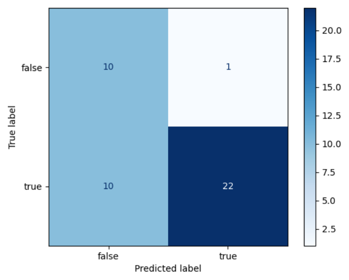

Figure 4: Confusion matrix of predictions in the test set (*prefix_length* $= 10$).

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	74.42%
Precision	83.97%
Recall	74.42%
F1-Score	76.04%

Table 4: Evaluation metrics for the decision tree classifier on the test set for *prefix_length* $= 10$.

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.

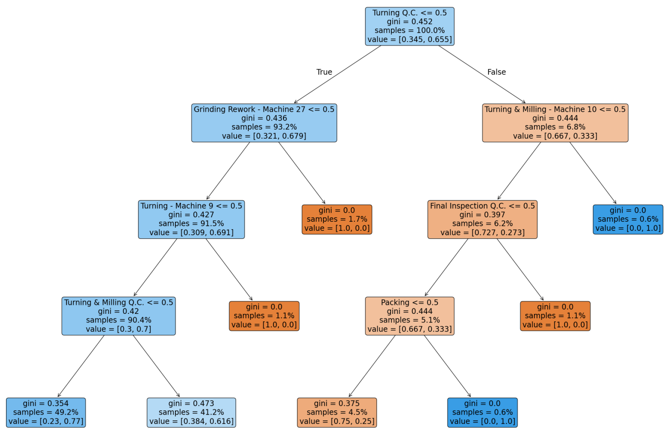

Figure 5: Simplified decision tree used to illustrate the recommendation extraction process. Given a node, the left child branch is taken if the condition is true (activity is absent in the trace), while the right child branch is taken if the condition is false (activity is present in the trace).

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 them. This is illustrated in Figure 6, with the green dotted nodes.

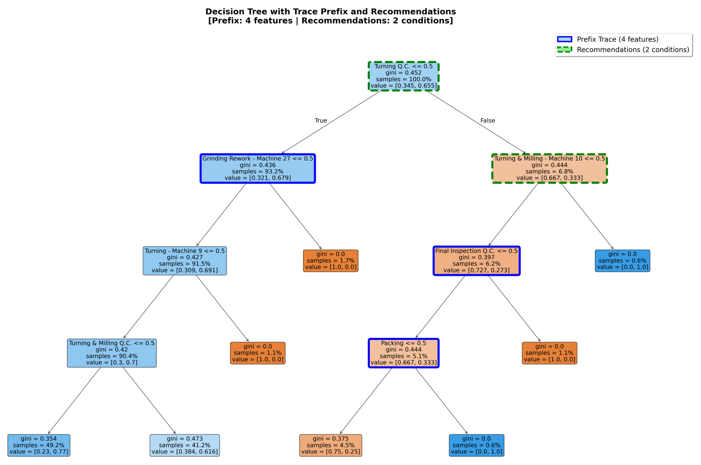

Figure 6: The blue squared nodes represent the activities present in the given prefix trace. The green highlighted node represent the recommended activities

Recommendations Quality Tables 5 and 6 present the evaluation of our recommendation system. This 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.

For *prefix_length* = 5, the recommendation system achieves a very high accuracy, coupled with a good recall. This indicates that, even with limited information, the system is often able to identify recommendations that align with positive outcomes. The lower precision reflects the fact that some recommended actions may not always be necessary, but overall the system demonstrates strong reliability in this setting.

At *prefix_length* = 10, the accuracy decreases, while precision remains high. This behavior suggests a more conservative recommendation strategy: the system proposes fewer but more targeted actions, which increases precision at the expense of overall accuracy. Despite this trade-off, the results confirm that the recommendation system remains effective and meaningful when more contextual information is available, supporting its usage.

Metric	Value
Accuracy	93.75%
Precision	75.00%
Recall	83.33%
F1-Score	72.09%

Table 5: Evaluation metrics for the recommendation system on the test set for $\textit{prefix\_length} = 5$.

Metric	Value
Accuracy	68.75%
Precision	81.48%
Recall	74.58%
F1-Score	65.12%

Table 6: Evaluation metrics for the recommendation system on the test set for $\textit{prefix\_length} = 10$.