Universiteit Leiden
The Netherlands

# BSc Data Science and Artificial Intelligence

Truly Unordered Rule Sets for Delivery Delay Prediction:
Analyzing the Performance–Interpretability Trade-off

Jesse Kamerbeek

Supervisors:
Dr. Matthijs van Leeuwen & Dr. Francesco Bariatti

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

28/06/2026

**Abstract**

Predicting delivery delay in e-commerce is becoming more important. Current approaches mainly employ black box machine learning models with strong predictive performance. However, these models lack interpretability, making it harder to understand which risk factors contribute to these delays.

This study benchmarks the intrinsically interpretable Truly Unordered Rule Sets (TURS) against CART, Random Forest, and XGBoost. These models are compared on the task of binary delivery delay classification, using the Olist Brazilian E-Commerce dataset. We quantify the trade-off between predictive performance and interpretability, and perform a subgroup analysis on TURS' learned rules. We contribute the first application of a truly unordered probabilistic rule set to delivery delay prediction. We find that TURS scores lower on PR-AUC than the black box models (Random Forest and XGBoost), but approaches them on ROC-AUC. TURS performs similarly to CART, slightly lower on PR-AUC and higher on ROC-AUC. Furthermore, we find that TURS offers a more compact model than CART, while having no hierarchical structure. Finally, TURS discovers interpretable subgroups, which highlight operational risk factors. For instance, we find that orders with tightly estimated delivery windows have increased delivery delay risk.

The main takeaway is that we quantify the cost of interpretability, and find that this cost may be worth it when the need for interpretability is high. This is especially relevant for high-stakes domains, such as healthcare or criminal justice, but also in lower-stakes environments where discovering risk factors is a priority.

# Contents

# 1 Introduction

Machine learning in supply chain optimization is increasingly applied [1]. This includes the use case of delivery delay prediction in an e-commerce context. Predicting delivery delay in e-commerce is important as it allows for the identification of the underlying risk factors, which could be used to mitigate these bottlenecks. Ultimately, predicting delivery delay may lead to better mitigation and therefore less delay, enhancing customer satisfaction and operational efficiency [1, 2].

Previous research has shown that machine learning models can predict delivery delay with high accuracy. Salari et al. [3] have proposed a data-driven framework generating real time delivery duration forecasts. This framework was developed on roughly 220,000 JD.com orders, aiming to provide customers a promised delivery date. The authors utilized a Quantile Regression Forest ensemble model for this framework, which according to their simulations, could improve sales volume by 3.7-6.1% over current policies. Lochbrunner [4] also took a regression approach, where they combined machine learning models such as XGBoost with human knowledge to predict delivery lead time. In this study pure machine learning models achieved a higher prediction accuracy than the ML-Human combination.

In contrast to the aforementioned regression approaches, recent research frequently frames the delivery time problem as a classification problem. For instance, Küp et al. achieved an AUC of 0.997 predicting on-time delivery as a binary class for real e-commerce orders at a logistics company using CatBoost [1]. Karakaya [2] employed ensemble methods such as ExtraTrees to classify last-mile delivery time in multiple classes from Amazon data and achieved an F-score of 0.978.

Even though previous research has shown significant accuracy in predicting delivery time and delay, most methods including the aforementioned approaches have one thing in common: they make use of so-called "black-box" machine learning models. Because these models are often superior in performance, they are currently being used widely in various domains. However, they lack interpretability, that is, their predictions cannot be explained in a way that the human mind can grasp [5]. This usage of black-boxes gives rise to severe consequences in high-stakes domains such as healthcare or criminal justice. Stiglic et al. [6] illustrate that a lack of model interpretability prevents broader adoption of ML in healthcare, as healthcare workers lack trust in complex ML models because they do not understand them. These domain experts must fully understand a model's predictions in order to accept or reject them.

The lack of interpretability is also problematic in lower-stakes fields such as supply chain optimization. This is because without interpretability, risk factors that are necessary for mitigating delivery delay cannot be discovered.

It is often argued that we can obtain interpretability from black-boxes through the field of Explainable AI with post-hoc explanation methods. However, Rudin [5] states that we should not try to explain black-box models for high-stakes decision making and knowledge discovery, but that we should instead create and apply intrinsically interpretable models, such as rule-based models. She also argues that interpretable models can often achieve an accuracy similar to complex models when data is structured and contains meaningful features [5]. Furthermore, explanations generated by post-hoc methods cannot be absolutely faithful to the model they explain [5].

Although intrinsically interpretable models can address the shortcomings of black-box models and post-hoc explanation methods, they often still contain their own interpretability limitations. Yang and Van Leeuwen [7] illustrate that current interpretable rule-based models often impose orders among rules, making them less comprehensible. They also find that probabilistic rules are not

often explored due to difficulties that arise with rule overlaps. To address these issues, Yang and Van Leeuwen [7] propose TURS (Truly Unordered Rule Sets). This interpretable rule-based ML model ensures rules remain unordered by only allowing rule overlap if two rules share the same probabilistic output, i.e. they agree on a prediction. Yang et al. [8] show that Truly Unordered Rule Sets are effective: the authors predict patient readmission risks with a ROC-AUC that approaches black-box models using only five interpretable, unordered rules, while also discovering important subgroups of patients.
Given the operational need for transparency in delivery delays, we identify a gap in the current literature on the use of intrinsically interpretable models for delivery delay prediction. This study addresses this gap by benchmarking the TURS model against three baseline models: the interpretable rule model CART and two black-box models: XGBoost and Random Forest. The benchmark is performed on the task of delivery delay classification rather than predicting delivery time itself. We define delivery delay as an order arriving later than the platform's estimated delivery date. Specifically, we aim to answer the following research questions:

(i) How does TURS compare to the baseline models (CART, Random Forest, XGBoost) in predictive performance for delivery delay classification?

(ii) How does TURS compare to the interpretable CART model in terms of model complexity?

(iii) Which subgroups of orders with substantially deviating delivery delay risk does TURS discover, and which operational risk factors emerge from them?

Ultimately, this study evaluates the trade-off between the predictive performance of black-box models and the comprehensibility, model complexity, and actionable insights gained from intrinsically interpretable models.

## 1.1 Contributions

This study contributes, to our knowledge, the first application of a truly unordered probabilistic rule set (TURS), a modern rule-based method, to delivery delay prediction. Furthermore, we quantify the trade-off between predictive performance and interpretability by benchmarking TURS against an interpretable baseline (CART) and two black-box ensembles (Random Forest and XGBoost). Finally, we discover subgroups of orders with substantially deviating delivery delay risk and illustrate how these subgroups can be interpreted to identify operational risk factors.

## 1.2 Thesis Overview

Chapter 2 discusses the preliminary background concepts on intrinsic interpretability, rule-based models, TURS, tree ensembles and related work. In Chapter 3 we present the Olist dataset, the preprocessing steps and engineered features. Chapter 4 describes the experimental setup, the models and hyperparameter tuning, and the complexity and subgroup analysis setup. Chapter 5 reports the findings for the three research questions: predictive performance, model complexity, and discovered subgroups. Finally, Chapter 6 concludes and discusses the findings, operational insights, limitations, and directions for future work.

# 2 Background and Related work

## 2.1 Intrinsic Interpretability vs. Post-hoc Explainability

There are two routes to interpreting machine learning models: (i) using models that are intrinsically interpretable, and (ii) using post-hoc explanation methods. A model is intrinsically interpretable if its structure can directly be interpreted by a human, such as a small decision tree or a small set of rules. Conversely, a post-hoc explanation method is detached from the original model, and tries to approximate the behavior and explain the prediction of the model in a comprehensible manner.
Post-hoc explanation methods are often chosen over intrinsically interpretable models, as they allow for the use of black-box models with stronger predictive performance. However, as Rudin [5] illustrates, the main problem with post-hoc explanation methods compared to intrinsic interpretability is that post-hoc explanation methods can never be truly faithful to the model they try to explain, as that would mean the original model would have already been interpretable. Because of this, Rudin argues [5] that in critical domains and high-stakes decision making, intrinsically interpretable models should be considered over post-hoc explanation methods. A notable family of intrinsically interpretable models are rule-based models. We discuss a range of rule-based models in the next section.

## 2.2 Rule-based Classification Models

Rule-based machine learning models are generally considered to be intrinsically interpretable. This is because the rules follow the if-then format, which aligns well with human intuition. However, rule-based models can vary in interpretability, depending on how the rules are structured and what they predict.
One factor of rule-based models is whether the rules are ordered. Ordered rule-based models are models where order is imposed among the rules. For instance, a decision list is an ordered if-then-else sequence that has to be read in order, meaning that to evaluate a single rule, every preceding rule must also be considered. In contrast, unordered rule-based models are models where each rule can be interpreted individually. This, however, can result in a conflict where two rules produce different predictions for the same instance.
Another factor in rule-based models is the output they produce. Rule models can produce a probability distribution over all classes, or they can predict a single class label.
These factors determine how well the model's predictions can be interpreted by humans. In the next section, we discuss TURS, which learns unordered, probabilistic rules and addresses the conflict where two rules differ in their prediction for the same instance.

## 2.3 Truly Unordered Rule Sets

Probabilistic Truly Unordered Rule Sets was developed by Yang and Van Leeuwen [7] to address several deficiencies in existing rule-based models, mainly regarding interpretability.

### 2.3.1 Probabilistic Rules and Rule Sets

The TURS model learns probabilistic rules in the form of **IF** $X$ **meets certain condition, THEN** $P(Y) = \hat{P}(Y)$, where $X$ represents features from the dataset, $Y$ the target feature, and $\hat{P}(Y)$ the

estimated probability of the instance belonging to a certain class.
Each rule's condition is a logical conjunction of literals, where a literal is a tuple consisting of (i) a feature from the dataset; (ii) a logical operator (e.g., "<", "=" or "≥"); and (iii) a feature value. For instance, consider the rule $S$: IF (product_weight $\geq 20\,\text{kg}$) AND (distance $\geq 100\,\text{km}$) THEN $\hat{P}(\text{Late}) = 0.4$. In this rule, the literals are "product_weight $\geq 20\,\text{kg}$" and "distance $\geq 100\,\text{km}$" and they make up the condition: (product_weight $\geq 20\,\text{kg}$) AND (distance $\geq 100\,\text{km}$), which if true for an instance, means that there is an estimated 40% chance of delivery delay. We define the *cover* of a rule as the set of instances that satisfy its condition, and the *coverage* as the size of this set.
For a given dataset, TURS produces a set of $K$ rules, such that rule set $M = \{S_1, \ldots, S_K\}$ together form the entire model. For each individual rule, the probability estimate $\hat{P}_S(Y)$ is computed using all instances in its cover, including those also covered by other rules. This makes each rule's estimate independent of the rest of the rule set, i.e., self-standing.
Given an instance with feature vector $X$, the model makes a class probability prediction in one of three ways, namely (i) an instance is covered by one rule; (ii) an instance is not covered by any rule, therefore the class probability prediction becomes the class distribution within the remaining dataset not covered by any rules; or (iii) an instance is covered by more than one rule. The latter case becomes problematic when multiple rules covering the same instance give varying probability distributions, i.e., the rules disagree with each other. Resolving this overlap is the main challenge that TURS addresses.

### 2.3.2 The TURS Model: Handling Overlaps

Rather than preventing rule-overlap, TURS allows rules to overlap only if they produce similar probability estimates, i.e., the rules agree with each other. Through this design choice, rules are able to be interpreted independently, without the need for ranking overlapping rules. TURS achieves this by incorporating the probabilistic output differences into the goodness-of-fit of the probabilistic model.
For instances where multiple rules apply, one might expect the model to estimate the class probabilities by taking the intersection of the covers of these rules, i.e. instances where all these rules apply, and looking at the class distribution of this cover. Although it may seem counterintuitive, TURS takes the union of the covers of the involved rules, i.e. instances where any of the rules apply. The probability of class $y$ is then estimated as the proportion of instances in this union whose target value equals $y$.
The reason TURS chooses this union-based estimate is because it inherently favors overlapping rules with similar probabilistic outputs when optimizing for likelihood. This is the case because for each individual rule estimate, $\hat{P}_S(Y)$ is computed over its full cover, including overlap with other rules. When two overlapping rules disagree, the empirical class distribution in their overlap differs from that in each rule's individual region. Since $\hat{P}_S(Y)$ averages over both regions, i.e. the overlap between rules and a rule's individual region, the probability estimate becomes a compromise between these distributions and the estimate does not match either distribution well. The union-based estimate used for the overlap itself shows similar behavior, as it lies between the two individual rule estimates, and therefore typically does not match the empirical class distribution in the overlap. Therefore, the likelihood of the data given such a model is low. On the other hand, when overlapping rules produce similar probability estimates, all quantities, i.e., the individual estimates and the union estimate, align with their empirical class distributions. This results in the likelihood being maximized.

In summary, this design causes the likelihood metric to penalize conflicting overlaps without requiring hyperparameters to tune the allowed difference between overlapping rules.

### 2.3.3 Learning TURS: Model Selection via MDL

Having defined the TURS model and its likelihood, the remaining task is to learn the rule set that describes the data best. Only optimizing for the maximum likelihood results in overfitting, which TURS addresses through the minimum description length (MDL) principle. MDL provides a formal trade-off between goodness-of-fit and model complexity without the need for a regularization parameter.

TURS uses the Kraft inequality to connect code length with probability. Then the Normalized Maximum Likelihood (NML) distributions are used to encode the data given the model. Because MDL requires a single probability distribution, NML produces a single representative probability distribution $P_M^{NML}$. This probability distribution is used to obtain the code length of the data $-\log_2 P_M^{NML}(Y^n = y^n \mid X^n = x^n)$ through the Kraft inequality, where $x^n = (x_1, \ldots, x_n)$ represents the feature vectors of all $(n)$ instances, and $y^n = (y_1, \ldots, y_n)$ represents their corresponding target values.

After that, the code length $L(M)$ encodes the model rule set, containing the number of rules in $M$, and for each rule the number of literals, the features used, and the operators and threshold values of those literals. The exact encoding scheme is described in detail by Yang and van Leeuwen [7]. Because $L(M)$ grows with the amount of rules and average amount of literals per rule, the MDL criterion naturally favors simpler rule sets that are complex enough to fit the data.

Combining the code length of the data given the model and the code length of the model rule set, TURS selects the rule set $M^*$ that minimizes the total code length:

$$M^* = \underset{M}{argmin}[-\log_2 P_M^{NML}(Y^n = y^n \mid X^n = x^n) + L(M)],$$

where the first term measures the goodness-of-fit of M to the data, and the second term measures the complexity of the rule set model M itself. As previously mentioned, the goodness-of-fit term contains the property of rule sets with overlapping rules and conflicting probabilistic output yielding a lower $P_M^{NML}(Y^n = y^n \mid X^n = x^n)$, therefore a larger code length. The complexity term $L(M)$ penalizes overly large or complex rule sets. As both terms are expressed in bits, they are minimized together without requiring a hyperparameter to balance them.

### 2.3.4 Algorithm Overview

The TURS model is not learned exhaustively as the combinatorial search space is too large. Exact methods such as branch-and-bound are not applicable to TURS. This is because TURS' rule quality depends on overlap with the rest of the rule set, meaning candidate rules cannot be evaluated individually. TURS learns rule sets iteratively, i.e., rule by rule. At each iteration, the algorithm searches for the single rule whose addition decreases the MDL-based score the most, relative to the number of additional instances it covers. This rule is then added to the rule set, and the process repeats until no further rule can be found that improves the MDL score.

A beam search is used to find each individual rule, forming a growing set by adding literals one by one, where the main beam keeps the most promising partial rules (conditions) at each step. However, allowing rule overlap introduces a challenge: existing rules in the rule set can block the

discovery of new useful rules, because a literal that would otherwise be a good starting point may create a conflicting overlap and be rejected by the main beam. To handle this, Yang and van Leeuwen [7] employ a dual-beam search, namely an auxiliary beam is introduced alongside the main beam. Its task is to score the candidate literals individually, ignoring overlap conflict with the current rule set, allowing the algorithm to still consider them for further growth.

## 2.4 Tree-based Ensembles as Black-box Baselines

Tree ensembles are models that combine a large number of decision trees into a single strong model. There are multiple approaches to build these ensembles. Random Forest uses *bagging*: training multiple decision trees on different subsets of the data, and then aggregating their predictions, resulting in a model that can generalize better than any of the individual trees. XGBoost uses *gradient boosting*: decision trees are trained sequentially, where each new tree minimizes the error of all preceding trees.
We choose tree ensembles as black-box baselines because they excel in tabular data, which applies to our dataset, and because these models are widely used in a large range of industries. Tree ensembles are not directly interpretable because of the large number of trees that are needed to build the final ensemble. While a few trees can be interpreted by humans, ensembles often use hundreds or thousands of trees, completely removing any interpretability.

## 2.5 Related Work

We review related work in two categories. First, we discuss alternative interpretable rule-based methods, including traditional and modern models. Second, we look deeper into current approaches, specifically aimed at delivery time and delay prediction.
CART (Classification and Regression Trees) [9] learns a binary decision tree, where each rule corresponds to a root-to-leaf path. Because of this, when evaluating a leaf node's prediction, every split leading up to that leaf node also has to be accounted for in the interpretation.
Rule learners avoid the tree's hierarchical structure, but they often introduce a different order. RIPPER (Repeated Incremental Pruning to Produce Error Reduction) [10] learns rule sets one class at a time, removing covered instances before moving on to the next class. This results in an ordered list of per-class rule sets, therefore imposing explicit order between these rule sets. Furthermore, only class labels are produced rather than probability estimates.
Unordered CN2 [11] differentiates itself by learning rules without imposing order between them, while also producing a probabilistic rule set. This makes it the closest competitor to TURS among the methods discussed here. However, CN2 resolves rule overlap conflicts at prediction time, by taking the weighted average of the individual rule estimates, while ignoring these conflicts during learning. It also uses one-versus-rest learning, where TURS uses direct multi-class learning.
More recent methods have explored alternative approaches. CLASSY [12], similar to TURS, uses the MDL principle for model selection. However, it produces an ordered rule list rather than an unordered rule set, which means that each rule's interpretation depends on all rules that precede it.
DRS (Diverse Rule Sets) [13], like TURS, directly learns a rule set for multi-class targets. However, DRS resolves rule overlap by minimizing it, while TURS allows overlap when the involved rules share a similar probabilistic output.

In contrast to the discussed methods, TURS is, to our knowledge, the only rule-based method that produces a truly unordered probabilistic rule set using MDL-based model selection.

The prediction of delivery time has been approached using various ensemble methods. Regression-based methods apply Quantile Regression Forests [3] and gradient boosting [4], while classification-based approaches use boosting algorithms (e.g. XGBoost, CatBoost [1]), and tree ensembles such as ExtraTrees [2]. Although these approaches achieve strong predictive performance, they are not intrinsically interpretable.
In the closest study to our work, Baryannis et al. [14] also evaluate the trade-off between performance and interpretability for delivery delay prediction. However, they use a decision tree (CART) as their interpretable model, in which a hierarchical structure between node splits exists. Furthermore, they evaluate the interpretability and performance trade-off using a combination of AI and supply chain experts.
Our work differs in two ways: (i) instead of using CART as the main interpretable model, we use it as a baseline, and test a more modern rule-based model that learns a flat, unordered probabilistic rule set as our main interpretable contender, and (ii) we evaluate a model that intrinsically learns an interpretable rule set without an expert-in-the-loop step. To our knowledge no truly unordered probabilistic rule set has been applied to the problem of delivery delay prediction.

# 3 Data

## 3.1 Dataset

This study uses the Brazilian E-Commerce Public Dataset [15], made available by Olist Store. The dataset contains approximately 100,000 anonymized e-commerce orders from 2016 to 2018 at multiple marketplaces in Brazil.
The dataset is a relational database consisting of eight tables, with each table being available as a CSV file. The main table is *olist_orders_dataset*, which we merge with the other tables through their respective identifiers, which can be seen in Figure 1. The tables we use contain important features such as seller and customer geodata, order timestamps, product information and delivery date estimations.
The temporal coverage of orders from the dataset after processing starts at October 2016 through August 2018, with almost all orders placed in 2017 and 2018 (94,353 out of 94,617). This is partly due to the platform's growth accelerating in 2017 and onward.

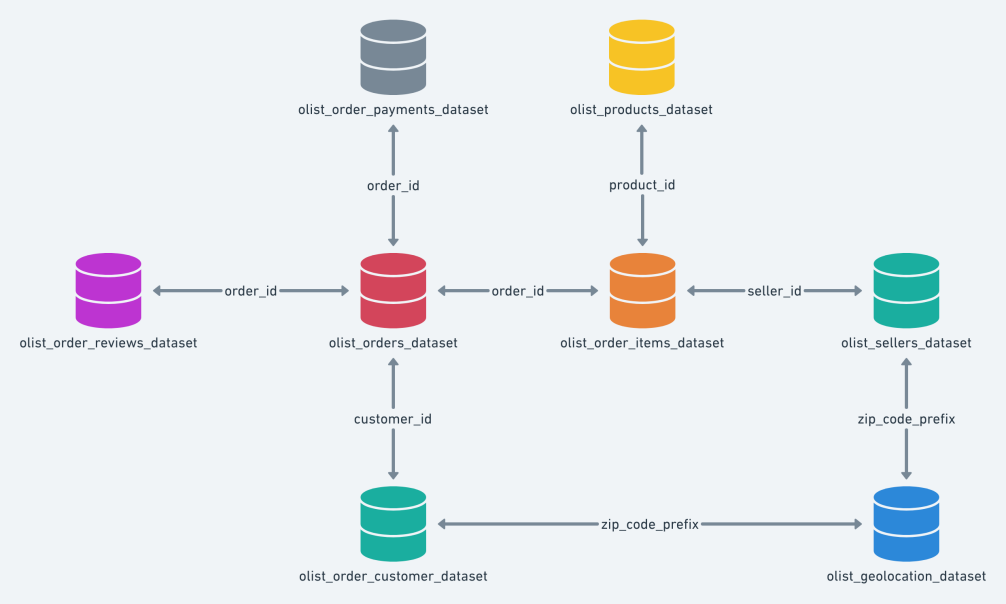



Figure 1: Brazilian E-Commerce Relational Database Schema, taken from [15].

## 3.2 Preprocessing

The goal of the preprocessing step is to transform the tables from the dataset into one merged dataset, with each instance corresponding to a unique *order_id* without any missing values in columns. We disregard *olist_order_reviews_dataset* as sentiment analysis is out-of-scope for the study.
We enrich the main orders dataset by merging it with the other datasets of interest, through their shared primary and foreign keys, according to the data schema (Figure 1).
The raw geolocation dataset contains instances of non-unique zip codes with corresponding coordinates. For each unique zip code, a set of its corresponding coordinates is created, of which we

compute the centroid by taking the mean of these coordinates. These centroids are merged with the main dataset as approximate geographical locations for the seller and customer, allowing the *seller_customer_distance* feature to be created.
The merging steps introduce duplicate *order_id* instances in our main dataset. This is caused by orders with sequential payments, multiple products or multiple sellers. To address these multi-item, multi-payment and multi-seller orders, we aggregate them to single order instances. We combine values of features that are scattered over duplicate orders to obtain their correct value, and aggregate them to a single order. For features associated with logistical friction, we take the worst-case values, where we assume that larger products and longer distances increase transportation difficulty. The remaining features are aggregated to maintain their intended meaning. Table 1 lists the aggregation method used for each feature.

Table 1: Aggregation rules used to collapse duplicate `order_id` rows into single order instances.

| Feature | Aggregation |
|---|---|
| *Correct order-level value* | |
| `price` | sum |
| `freight_value` | sum |
| `product_weight_g` | sum |
| *Largest single constraint (worst-case logistical friction)* | |
| `product_length_cm` | max |
| `product_height_cm` | max |
| `product_width_cm` | max |
| `seller_customer_distance` | max |
| *Meaning-preserving* | |
| `product_photos_qty` | mean |
| `payment_sequential` | max (= number of payments) |
| `payment_type` | combined category |

These aggregations leave us with a few more duplicate *order_id* instances, which contain missing values in payment, price and freight features. Due to the incomplete information we drop these instances from the dataset. Even though the *order_id* instances are now all unique, there are still missing values in 20 features of which the largest amount occur in *product_photos_qty* and *order_delivered_carrier_date* (1545 and 1003 respectively). Since there are 3547 instances with missing values left, we drop these instances, because the reduction in dataset size is only $\sim 3.6\%$. We assume that dropping these instances does not remove meaningful information for the target feature, though we have not verified this.
The last preprocessing step includes filtering the dataset on feature *order_status* being "delivered", which leads to 6 less instances in our dataset. This results in our final processed dataset with 94,617 unique order instances.
Finally, we finish preprocessing and define our binary target feature as follows:

$$Target = \begin{cases} 1 \text{ (Late)} & \text{if } t_{actual} > t_{estimated} \\ 0 \text{ (On Time)} & \text{if } t_{actual} \leq t_{estimated} \end{cases}$$

## 3.3 Feature Engineering

Beyond the aggregations described in Section 3.2, we engineer additional features based on potential drivers of delivery delay. These features represent multiple facets related to an order such as geographical, order complexity, and temporal attributes. The goal of feature engineering is to create new features that allow TURS to generate rules with more predictive accuracy and greater coverage. The final feature set is shown in Table 2.

The feature *seller_customer_distance*, computed in Section 3.2, along with the binary feature *same_state* are created to catch potential supply chain delays at greater order route distances, or smoother order deliveries at shorter distances.

We create and add the features *number_of_products* and *number_of_sellers*, based on the assumption that orders with a higher value of these features could have a higher chance of delivery delay due to the possibility of multiple supply chains being involved.

We compute binary features *is_mixed_category_order* and *multiple_payments*, assuming that multiple product categories and multiple payments could be associated with a more complex or multi-item order.

Temporal features we create include *purchase_month*, *purchase_dayofweek*, *purchase_hour*, and *is_weekend*. These features aim to catch temporal factors in e-commerce orders, e.g., an order made in December on a Friday night will probably have a longer delivery time than an order made on a regular Tuesday at 9 am in April.

Furthermore, we create *estimated_delivery_days* and *shipping_window_days* from order timestamps. We assume that the estimated delivery window will be informative given that our target definition includes the estimated delivery date.

We derive the feature *product_volume_cm*3 from the dimensional features aggregated to their per-axis maximum to capture the worst-case bounding box volume for a given order.

Finally, we create *price_per_product* and $freight\_price\_ratio$ as candidate features without a strong assumption, leaving the model to determine whether they are informative.

Table 2: Final Feature Set used for Model Training.

| Feature | Type | Description | Source |
|---|---|---|---|
| *Order value and payment* | | | |
| price | Numerical | Total order price (sum over items) | Aggregated |
| freight_value | Numerical | Total freight cost (sum over items) | Aggregated |
| payment_type | Categorical | Payment method (e.g., credit card, boleto, voucher) | Aggregated |
| multiple_payments | Binary | 1 if order was paid in more than one transaction | Engineered |
| price_per_product | Numerical | price / number_of_products | Engineered |
| freight_price_ratio | Numerical | freight_value / price | Engineered |
| *Order composition* | | | |
| number_of_products | Numerical | Count of distinct items in the order | Engineered |
| number_of_sellers | Numerical | Count of distinct sellers in the order | Engineered |
| is_mixed_category_order | Binary | 1 if order contains products from more than one category | Engineered |
| *Product attributes* | | | |
| product_weight_g | Numerical | Total order weight (sum over items) | Aggregated |
| product_length_cm | Numerical | Maximum item length (cm) | Aggregated |
| product_height_cm | Numerical | Maximum item height (cm) | Aggregated |
| product_width_cm | Numerical | Maximum item width (cm) | Aggregated |
| product_volume_cm3 | Numerical | length $\times$ height $\times$ width | Engineered |
| product_photos_qty | Numerical | Mean number of product photos across items | Aggregated |
| *Geographic* | | | |
| seller_customer_distance | Numerical | Haversine distance (km) between seller and customer centroids | Engineered |
| same_state | Binary | 1 if seller and customer reside in the same Brazilian state | Engineered |
| *Temporal* | | | |
| purchase_month | Numerical | Calendar month of order_purchase_timestamp (1–12) | Engineered |
| purchase_dayofweek | Numerical | Day of week of purchase (0 = Monday, 6 = Sunday) | Engineered |
| purchase_hour | Numerical | Hour of purchase (0–23) | Engineered |
| is_weekend | Binary | 1 if purchase was on a Saturday or Sunday | Engineered |
| *Time-window* | | | |
| estimated_delivery_days | Numerical | order_estimated_delivery_date − order_purchase_timestamp (days) | Engineered |
| shipping_window_days | Numerical | shipping_limit_date − order_purchase_timestamp (days) | Engineered |

# 4 Methodology

## 4.1 Experimental Setup

All experiments use a fixed random seed for reproducibility. We split the data into a training and test dataset, containing 75,693 and 18,924 instances respectively (approximately 80%/20% split). We use stratification during the train test split to accurately represent the class imbalance in both train and test datasets (also approx. 8.1%).

We compare performance metrics on the holdout test dataset using the area under the precision-recall curve (PR-AUC) and the area under the receiver operating characteristic (ROC-AUC). We choose the PR-AUC as the primary metric because we have a binary classification problem with class imbalance [16], where the positive class only accounts for $\sim 8.1\%$. We approximate PR-AUC through the `average_precision_score` function (sklearn) rather than the trapezoidal `auc` function (sklearn). The trapezoidal estimate linearly interpolates between adjacent points on the PR-curve, which results in the method being overly optimistic for the TURS model, given its small number of rules. The `average_precision_score` estimate is more accurate, as it computes the average precision as a weighted mean of the precision at each recall threshold instead of interpolating between points.

Hyperparameters for the baseline models (XGBoost, Random Forests, CART) are selected on the training set through 5-fold stratified cross validation. For XGBoost and Random Forests we select hyperparameters using randomized search (RandomizedSearchCV, n_iter=50) over the grid in Tables 6 and 7 in Appendix A. The hyperparameter space for CART is smaller; therefore we use exhaustive grid search (GridSearchCV) over the values in Table 5 in Appendix A. Note that we cap the depth hyperparameter of CART at 6 in the grid, since we want to keep this baseline interpretable for the upcoming comparison with TURS. For each model, the configuration with the highest cross-validated PR-AUC is refit on the full training set and evaluated on the held-out test set. The selected hyperparameters are displayed in Appendix B.

We did not apply hyperparameter tuning to TURS and we used the default parameters of the TURS2 implementation [17]. Furthermore, we modify the TURS code to omit the cross validation and add a train test split with our seed. We report metrics on the test dataset using 5, 20, and unbounded (MDL) rule counts. We retrieve all MDL rules (34 rules) that TURS learned on the training dataset as a basis for the subgroup analysis.

## 4.2 Implementation

We access the baseline models through their respective Python libraries; `DecisionTreeClassifier` [18], an implementation of the CART algorithm [9], `RandomForestClassifier` [18], and `XGBClassifier` [19].

For all three baseline models, categorical features are one-hot encoded. We did not scale any data as rule models and tree-based ensembles generally do not benefit from it. For TURS, we use the TURS2 reference implementation [17] with several modifications described below. Unlike the baseline models, TURS does not need preemptive processing of input features, except for placing the target feature in the last column of the input dataset. The TURS2 implementation encodes features internally, where it one-hot encodes features with a cardinality $\leq 20$. Therefore, we pass the unscaled numerical features along with label-encoded categorical features to TURS, allowing

the implementation to perform the encoding. Passing the numerical features unscaled also allows for unit preservation (e.g., distance in km, volume in $cm^3$), which is preferable for the subgroup analysis. We make the following modifications to the TURS2 implementation [20]:

- **Feature name preservation.** The default implementation assigns generic feature identifiers (`X0`, `X1`, ...) to learned rules. We extract column names from the input DataFrame and propagate them through the data loader, run configuration, and `DataInfo` object so that learned rules are represented with the original feature names.
- **Maximum-rules early stopping.** The `max_num_rules` parameter is defined in TURS2's configuration but is not implemented as a stopping criterion, as the implementation relies on the MDL-based stopping criterion. We add an explicit check that terminates rule addition when `max_num_rules` is reached, such that we can report results at our desired rule counts alongside the default MDL-determined count.
- **Train/test evaluation.** The reference implementation uses an internal cross-validation procedure for evaluation. We replace this with a single train/test split using our seed, so that TURS is evaluated on the same held-out test set as the baseline models.

## 4.3 Subgroup and Complexity Analysis

We analyze the final TURS-MDL rule set (34 rules) learned by TURS to uncover subgroups of orders with deviating chances of delivery delay, aiming to gain insights into risk factors. For each rule, we report its conditions, estimated late-delivery probability $P(\text{Late} = 1 \mid \text{Rule})$, and coverage. We focus on rules that meet two conditions: (i) coverage of at least 500 training instances, and (ii) estimated late-delivery probability either at or above 0.15 (elevated-risk) or at or below 0.04 (reduced-risk), relative to the training baseline prevalence of 8.1%. We organize these subgroups into trends that emerge in the full rule list based on shared literals. For each trend, we discuss representative rules that meet the specified criteria. We compare these subgroups' rule conditions and estimated risks to the overall population to identify operationally relevant risk factors.
Complexity of the different TURS models is evaluated by the number of rules in the set and the average rule length. We compare these to the complexity of the CART baseline.

# 5 Results

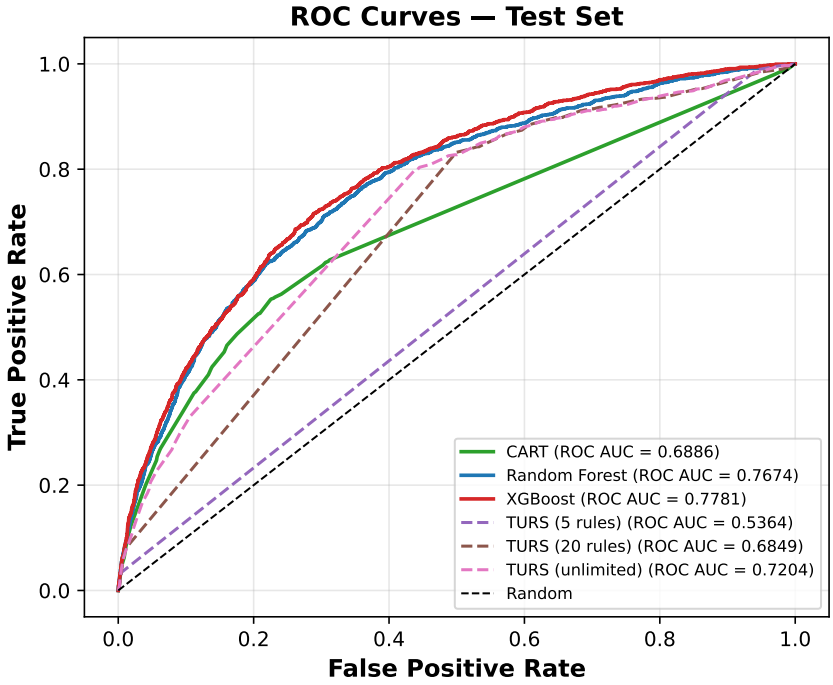
In this chapter we address the following research questions: (i) How does TURS compare to the baseline models (CART, Random Forest, XGBoost) in predictive performance for delivery delay classification? (ii) How does TURS compare to the interpretable CART model in terms of model complexity? (iii) Which subgroups of orders with substantially deviating delivery delay risk does TURS discover, and which operational risk factors emerge from them?

## 5.1 Predictive Performance

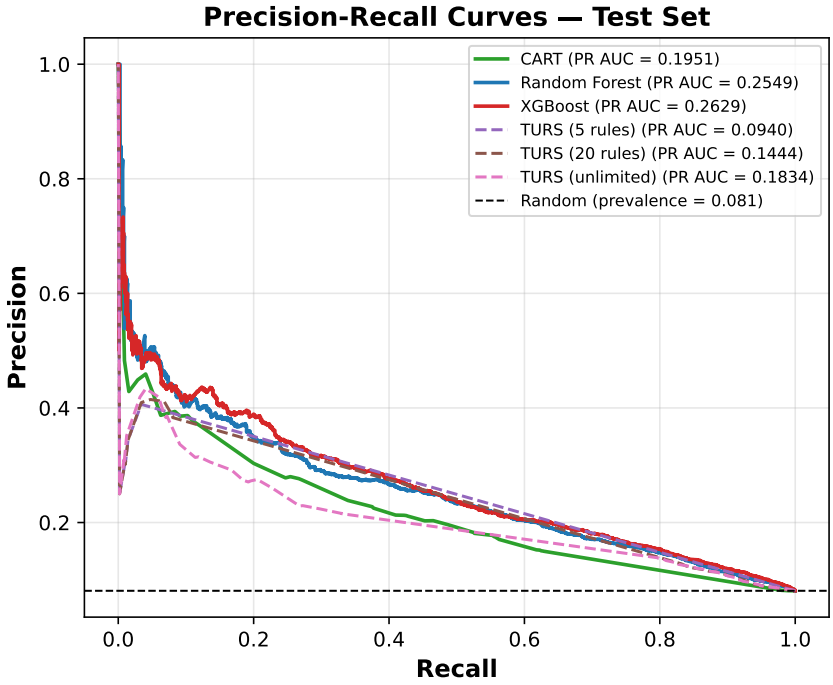
Table 3 reports the test set performance of all models and Figures 2a and 2b show the corresponding ROC and precision-recall curves. For TURS, metrics are reported for 5, 20 and 34 rules, where the 34 rule model is capped by the MDL principle.

Table 3: Evaluation Metrics Across Models on Hold-out Set.

| **Model** | **PR-AUC** | **ROC-AUC** |
|---|---|---|
| CART | 0.1951 | 0.6886 |
| Random Forest | 0.2549 | 0.7674 |
| XGBoost | **0.2629** | **0.7781** |
| TURS-5 | 0.0940 | 0.5364 |
| TURS-20 | 0.1444 | 0.6849 |
| TURS-MDL (34 rules) | 0.1834 | 0.7204 |




(a) ROC curves




(b) Precision-Recall curves

Figure 2: ROC (a) and Precision-Recall (b) Curves for all Models on the Hold-out Set.

Given that this is an imbalanced classification task (positive class = 8.1%), these results align with our expectations, namely that all PR-AUC values are limited. XGBoost scores highest on PR-AUC

(0.2629) and ROC-AUC (0.7781), closely followed by Random Forest.
The interpretable rule-based models CART and TURS score lower than the black box models. Even though TURS-MDL outperforms CART in ROC-AUC (TURS-MDL = 0.7204, CART = 0.6886), CART slightly beats TURS-MDL on our primary metric PR-AUC (CART = 0.1951, TURS-MDL = 0.1834) for this imbalanced classification task.
Within the TURS models, we observe the performance climb substantially as we permit more rules: from TURS-5 being just above random (PR-AUC = 0.0940, ROC-AUC = 0.5364) to TURS-MDL not being extremely far away from black boxes (PR-AUC = 0.1834, ROC-AUC = 0.7204). This pattern shows that overly restrictive rule limits prevent the model from capturing sufficient signal. Allowing TURS to decide its rule count by balancing signal and complexity through its MDL-based stopping criterion leads to the best fit to the data, in our case outperforming both manually created limits.
The performance gap between TURS-MDL and the best black box ensemble (approx. 0.08 in PR-AUC and 0.06 in ROC-AUC) can be represented as the cost of intrinsic interpretability within this dataset. The complexity and interpretable insights that TURS provides in exchange are examined in the following sections.

## 5.2 Complexity Analysis

This section evaluates and compares the complexity costs of the TURS and CART models. We exclude the black box ensembles (Random Forest and XGBoost) as these models are not inherently interpretable. Even though each individual decision tree of these ensembles can be interpreted to an extent, the final prediction of the ensemble cannot be reduced to an interpretable structure, making rule-level complexity metrics inapplicable.
Table 4 shows the number of rules (or leaves) and the average rule length (or leaf depth) for each model. TURS-MDL is more compact compared to CART in both condition count (TURS-MDL = 34, CART = 46) and in their average complexity (TURS-MDL = 4.06 literal average, CART = 5.74 average depth). Among the TURS models, the average rule length remains stable across the different rule counts, indicating that TURS produces rules of similar average complexity on this dataset, regardless of a rule count limit.

Table 4: Complexity Comparison between TURS Models and CART.

| **Model** | **# Rules / Leaves** | **Avg. Rule Length / Leaf Depth** |
|---|---|---|
| TURS-5 | 5 | 4.2 |
| TURS-20 | 20 | 4.25 |
| TURS-MDL | 34 | 4.06 |
| CART | 46 | 5.74 |

The comparison between rule length and leaf depth requires nuance, as these quantities measure different forms of complexity. In a decision tree such as CART, every instance traverses a path from the root node to a leaf node. This means interpreting any leaf node requires considering the full sequence of splits leading to it from the root. This negatively impacts interpretability as the tree grows deeper, because a hierarchy of conditions arises. For instance, take a leaf node that classifies

instances at depth 6. This means the last condition before the leaf node is dependent on the five conditions before it.
In contrast, TURS' rules are non-hierarchical and self-standing, meaning they can be interpreted individually without considering other rule outcomes. Although TURS still requires multiple literals to hold jointly, these rules contain no hierarchy between them, and the rules themselves remain interpretable in isolation. This gives TURS, in combination with the smaller average rule length, a better interpretability profile.

## 5.3 Subgroup Analysis

This analysis starts by illustrating trends that emerge in the full rule list. The exhaustive rule list can be found in Appendix C, Table 9. After that we look into rules that represent these trends using the two criteria mentioned in Section 4.3: (i) Coverage of at least 500 training instances, and (ii) estimated late-delivery probability either at or above 0.15 (elevated-risk) or at or below 0.04 (reduced-risk), relative to the 8.1% baseline prevalence.

**Pattern 1: Tight estimated delivery windows combined with structural risk factors. (increased risk)** Several elevated-risk rules share a literal of the form `estimated_delivery_days` $< X$, with $X$ ranging from 10 to 28 days. Rule 32 demonstrates this effect outside the peak-season months identified in Pattern 2, in combination with a large order distance, suggesting that tight estimation windows could act as an independent risk factor:

> **Rule 32:**
> IF seller-customer distance $\geq 1093.35$ $\wedge$ estimated delivery days $< 23$ $\wedge$ purchase month $\notin$ {Feb, Mar, Jul, Aug, Nov}
> THEN $\hat{P}(\text{Late}) = 0.238$ (coverage: 533)

A second representative example combines the window literal with a month literal and a price-per-product condition, showing that the window effect persists when other features are added:

> **Rule 30:**
> IF estimated delivery days $< 10$ $\wedge$ price per product $\geq 59.99$ $\wedge$ purchase month = August
> THEN $\hat{P}(\text{Late}) = 0.229$ (coverage: 936)

Recall that the target is defined as $t_{\text{actual}} > t_{\text{estimated}}$. The elevated late-delivery rates in subgroups with short estimated windows may therefore reflect systematic over-optimism in Olist's delivery date estimation, rather than physically slower deliveries. Other qualifying rules sharing this pattern (Rules 12, 23, and 26; see Appendix C) combine tight windows with other literals such as peak months and longer distances; we revisit Rule 23 in Pattern 2 as an illustration of compound risk.

**Pattern 2: Peak-season months combined with structural risk factors. (increased risk)** All elevated-risk rules that adhere to our criteria contain a `purchase_month` literal, either as a positive selector or as an exclusion of months (R32), suggesting that it is a major associative factor for delivery delay risk. The most prevalent months are March (3x), November (2x) and

February (2x). Furthermore, December and August also come up once as high-risk literals. The month November is to be expected as a busy month, as it contains Black Friday (late November), which often causes logistical challenges. A feasible explanation for March being a prevalent value is that on March 15th in Brazil Dia do Consumidor (day of the consumer) takes place. Order counts from the training set in Figure 3 partly support this hypothesis: November 2017 shows a clear peak in volume relative to surrounding months, consistent with a Black Friday peak. March 2018 has, in combination with January, the most orders in 2018. However, the gap with other months is relatively small. December and August show no noticeable patterns relative to the other months. Note that neither the `purchase_month` literal nor the `estimated_delivery_days` literal (Pattern 1) ever stands in isolation within a rule. These literals are always combined with at least one other literal, such as order distance, product features (weight, photo qty), estimated delivery day and shipping window day. The two main risk patterns also overlap in several rules, a clear example being Rule 23, where tight delivery estimation window (Pattern 1) co-occurs with a peak-season month (Pattern 2).

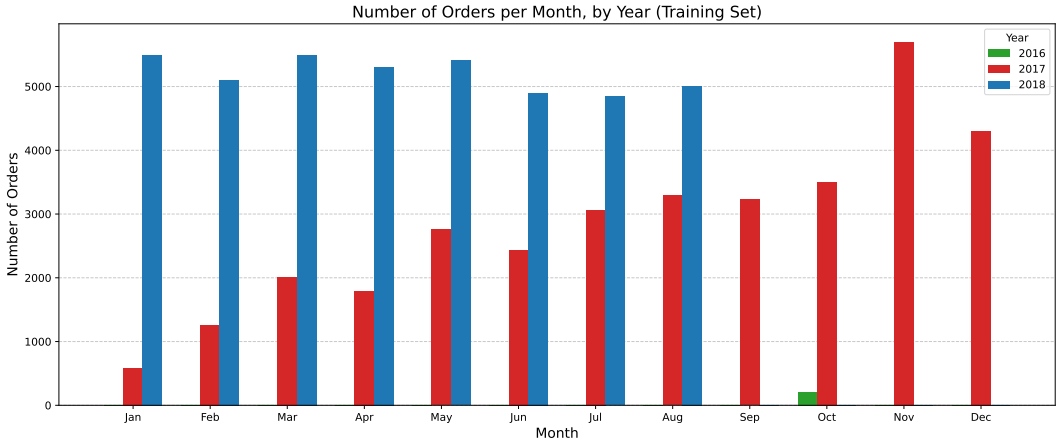



Figure 3: Number of Orders per Month, colored by Year, taken from the Training Set.

**Rule 23:**
IF estimated delivery days $< 23 \ \wedge$ same state $= 0 \ \wedge \ 499.82 \leq$ seller-customer distance $< 1093.35 \ \wedge$ purchase month = March
THEN $\hat{P}(\text{Late}) = 0.269$ (coverage: 994)

Rule 23's estimated delay probability of 0.269 exceeds that of rules only containing a tight delivery estimation window with structural factors (e.g., R32: 0.238) or rule only containing peak-season month with structural factors (e.g., R28: 0.204), suggesting that these two main patterns compound rather than substitute each other. We illustrate the seasonal amplification effect with two other representative rules below.

**Rule 28:**
IF product photos quantity $< 2 \ \wedge$ product volume $cm^3 \geq 15{,}750 \ \wedge$ seller-customer distance $< 1472.78 \ \wedge$ purchase month = November
THEN $\hat{P}(\text{Late}) = 0.204$ (coverage: 1,171)

Rule 28 combines the November peak with physical product attributes and order distance. The subgroup contains large items (`product_volume_cm3` $\geq$ 15,750) with few product images, shipped over small to moderately large distances. Unlike rules in Pattern 1 and 3, this rule does not contain the `estimated_delivery_days` literal, suggesting a logistical and possibly seller related effect rather than an estimation calibration effect. The combination of larger products in a busy period may cause logistical friction, while the low photo count may indicate a lower quality product listing, though this cannot be verified from the data.

**Rule 34:**
IF freight value $< 19.01$ $\wedge$ same state $= 0$ $\wedge$ purchase month = November
THEN $\hat{P}(\text{Late}) = 0.165$ (coverage: 2,175)

Rule 34 combines the November peak with low freight value and cross-state delivery. A low `freight_value` may reflect less expensive or slower shipping methods, which, in combination with cross-state deliveries that typically involve longer distances, might be associated with additional delay risk during the Black Friday period. A similar pattern, combining a specific `freight_value` range with `purchase_month` = March, appears in Rule 31 (see Appendix C). As with the other patterns, these are observed associations and not causal relationships.

**Pattern 3: Generous estimated delivery windows combined with other risk-reducing factors (reduced risk).** Almost all low-risk subgroups that adhere to our criteria contain a literal `estimated_delivery_days` $\geq X$. This pattern can be seen as the inverse of pattern 1, that is, tight estimated delivery window $\rightarrow$ increased risk, generous estimated delivery window $\rightarrow$ reduced risk. R3 and R13 are representative examples that fit the criteria, with Rule 3 having the lowest $\hat{P}(Late)$ among all rules in the ruleset and Rule 13, excluding the default rule, having the biggest coverage out of all rules, while also having a low $\hat{P}(Late)$. Note that R3 also contains the month June as a literal, which according to Figure 3 is a relatively calm month volume-wise. R13 takes a different approach, by excluding six specific months, which partly overlap with the elevated-risk months identified in Pattern 2. Furthermore, R3 and R13 also both contain literals that cap the maximum `seller_customer_distance`, further contributing to a reduced delay risk.

**Rule 3:**
IF seller-customer distance $< 816.48$ $\wedge$ estimated delivery days $\geq 28$ $\wedge$ purchase month = June
THEN $\hat{P}(\text{Late}) = 0.006$ (coverage: 1,448)

**Rule 13:**
IF seller-customer distance $< 1093.35$ $\wedge$ estimated delivery days $\geq 22$ $\wedge$ purchase month $\notin$ {Jan, Feb, Mar, Apr, Nov, Dec}
THEN $\hat{P}(\text{Late}) = 0.019$ (coverage: 12,748)

These insights suggest that the reduced-risk profile arises from a combination of generous delivery estimation, short to moderately large delivery routes, and the absence of seasonal logistic pressure rather than the generous window factor alone.

**Counter-intuitive finding: multi-seller orders (reduced risk).** In Section 3.3 we hypothesized that orders with a greater value of the feature `number_of_sellers` would contribute to increased delivery delay risk, as these often involve multiple supply chains. However, Rule 6 shows a complete opposite effect by finding a subgroup in which orders with more than one seller, have a reduced risk of delivery delay.

**Rule 6:**
IF number of sellers $\neq 1$
THEN $\hat{P}(\text{Late}) = 0.015$ (coverage: 996)

Looking at the coverage of Rule 6, we find that multi-seller orders form a small fraction of the data. Out of the 75,693 training instances only 996 ($\sim 1.3\%$) have more than one seller. Furthermore, we find that in the dataset multi-seller orders have a slightly higher delivery estimate on average (25.2 vs 23.4).
Within the 996 multi-seller orders, 620 have `estimated_delivery_days` $\geq 23$. These more generous window orders show a lower late-delivery rate (0.010) as opposed to the orders with `estimated_delivery_days` $< 23$ (0.024). This finding is consistent with Patterns 1 and 3. Because the target is defined as $t_{\text{actual}} > t_{\text{estimated}}$, a more generous delivery window reduces the likelihood of an order being classified as delayed.
Therefore, the prevalence of generous delivery windows within the `number_of_sellers` $\neq 1$ subgroup offers a partial explanation for the reduced risk observed. This finding shows a practical advantage of intrinsically interpretable models: learned rules can be directly inspected against domain hypotheses, without the need for post-hoc analysis.

# 6 Conclusions & Discussion

This chapter serves as a reflection and synthesis of the research conducted within this thesis, structured in five components.
First, the proposed research questions are individually answered. Second, we discuss our results in the context of the literature and methodology. Third, we discuss potential operational insights gained from the subgroup analysis. Fourth, we highlight limitations of our work. Finally, we suggest possible future directions that address the limitations and fulfill existing gaps in our work.

## 6.1 Research Questions & Conclusions

For this evaluation, we consider the strongest TURS model we trained: TURS-MDL with 34 rules, meaning we refer to TURS-MDL when stating TURS.

**How does TURS compare to the baseline models (CART, Random Forest, XGBoost) in predictive performance for delivery delay classification?** We observe from the results that TURS scores lower on metrics PR-AUC and ROC-AUC than the black box baselines. However, TURS' relative performance is asymmetrical across both metrics. On our main metric PR-AUC, TURS scores roughly 30.2% lower than the best black box baseline, XGBoost. On ROC-AUC, TURS approaches the strongest black box baseline, trailing by a small gap compared to the PR-AUC difference. TURS scores similarly to the interpretable baseline CART, scoring slightly lower on PR-AUC and slightly higher on ROC-AUC.
These results show that the main interpretability cost of TURS is concentrated in PR-AUC, i.e., how precisely late order instances are identified across recall thresholds, while mostly maintaining its ROC-AUC, i.e., the ability to rank orders on risk. Furthermore, TURS approaches a tuned CART model in the ability to classify late order instances, without any hyperparameter tuning applied.

**How does TURS compare to the interpretable CART model in terms of model complexity?** The complexity analysis shows that TURS has a smaller number of rules and a shorter average rule length compared to CART's leaf count and average leaf depth. However, as mentioned before, this is not an equal comparison as the compared metrics function differently. The real advantage of TURS is that there is no hierarchical structure among its rules, whereas interpreting a CART leaf requires considering the full root-to-leaf path. Therefore, TURS offers a more compact model than CART. Combining this compactness with the unordered nature of TURS' rules, we conclude that TURS is more interpretable and structurally less complex than the CART baseline.

**Which subgroups of orders with substantially deviating delivery delay risk does TURS discover, and which operational risk factors emerge from them?** We consider subgroups that meet the criteria specified in Section 4.3. TURS discovers multiple subgroups whose delivery-delay risk deviates substantially from the dataset's base delivery-delay rate in both directions, ranging from almost zero up to roughly 32% (Appendix C).

These subgroups illustrate a few associative risk factors: the estimated delivery window, the purchase month, and structural factors such as order distance and product attributes. We examine these factors more deeply in Section 6.3.
These findings show that TURS offers the most favorable interpretability profile among the evaluated models at a moderate cost of predictive performance, and unlike the baseline models, TURS reaches its performance without any hyperparameter tuning. Furthermore, TURS' unordered nature allows for the interpretation of individual rules. For instance, in our dataset, Rule 6 shows that orders with multiple sellers have a reduced delivery-delay risk, even though we hypothesized that this subgroup would have an increased delivery-delay risk.
Therefore, the performance cost of TURS may be worth it when the need for interpretability is high enough, especially in high-stakes domains such as healthcare or criminal justice, or even in lower-stakes environments where discovering risk factors is a priority.

## 6.2 Discussion

The discovered cost of interpretability of TURS on this prediction task aligns with those found in the literature. Baryannis et al. [14] also discovered a cost of interpretability in the form of average precision (PR-AUC). Their depth restricted tree, capped for interpretability, found a 37% PR-AUC cost compared to black box model SVM. They also saw a decrease of only 5% in recall. These findings align with our results: in both studies the cost of interpretability is asymmetric and concentrated on the PR-AUC axis rather than being balanced across metrics. In our results, we see PR-AUC being the main cost, with ROC-AUC being mostly maintained. Their framework differs in that they include an expert-in-the-loop step during evaluation. Our study trains a modern flat unordered rule set for our interpretable benchmarking model, without an expert-in-the-loop step. This illustrates that the asymmetric interpretability cost pattern recurs across studies with substantially varying methods.
The cost of interpretability we found was obtained on TURS' default parameters without the need to optimize hyperparameters. This resulted in a model that performs almost as well as a tuned CART model on our main metric. Conversely, the baseline models were tuned on grids from 96 (CART) up to 104,976 (XGBoost) combinations. These grids were searched to find the best balance in model complexity and goodness of fit. TURS finds this balance internally using the MDL principle, eliminating the need for hyperparameter tuning and grid search. TURS therefore offers a competitive interpretable model that does not require hyperparameter tuning. This, however, is not the main advantage of TURS. The main strength of TURS lies in its interpretability benefits and the insights it provides. The next section examines the operational insights TURS highlights.

## 6.3 Operational Insights

Section 5.3 described the three main patterns that emerged from the discovered subgroups. This section discusses if and how these patterns could be interpreted operationally. Specifically, we interpret the identified patterns as potential operational risk factors and discuss if they could be actionable.

**Tight estimated delivery windows combined with structural risk factors.** Pattern 1 illustrates that subgroups containing a tight estimated delivery window, have an elevated delivery

delay risk. Since the target variable is defined as $t_{actual} > t_{estimated}$, this pattern could point to two factors, or a combination of the two: (i) an order with inherent structural risk factors, such as distance, product weight and seasonal amplification, or (ii) a miscalibration of the delivery estimate. This pattern suggests that a tighter estimated delivery window has a greater chance of being classified as late. However, we cannot distinguish which factor dominates, though it is likely a combination of the two.
A potential actionable step could be to include a buffer in the delivery estimate, whenever the order already contains other structural risk factors or takes place in a busy month. However, there remains a trade-off between providing customers a realistic delivery date window and one that is unnecessarily late.

**Peak-season months combined with structural risk factors.** Pattern 2 shows increased risk for certain months in combination with structural risk factors.
We find five different months that come up as high risk literals, however, we observed that out of those months only November is noticeably busier in terms of order counts. This suggests that November contributes to the delivery delay risk.
Furthermore, we observe that in some rules, such as Rule 23 (March), a month literal is combined with Pattern 1, i.e., tight delivery estimation. This observation is consistent with the compounding effect observed in Section 5.3, i.e., that tight delivery estimation windows in combination with other risk factors increase the chance of an order being classified as late.
Busy months can be anticipated by looking at historical order counts and external data sources such as a retail calendar. Therefore, an actionable step could be to plan logistical capacity for November, as it is the only month where there is a volume-based explanation for the increased risk.

**Generous estimated delivery windows combined with other risk-reducing factors.** Pattern 3 can be seen as the inverse of Pattern 1, namely that subgroups with a broader delivery window have a reduced risk profile. This pattern also includes structural risk factors, however, they are often also inverted, meaning that these factors associate with lower delivery delay risk. For instance, Rule 13 restricts the order distance from small to moderately large orders (1093 km). Inversely to Pattern 1, this pattern does suggest that a broader delivery estimation window has a lower chance of being classified as late, however, we still cannot distinguish if the dominating factor is the window itself, or a better logistic process. This pattern supports the actionable insight made in Pattern 1, specifically that a broader delivery window correlates with a substantially reduced delivery delay risk in our dataset.

In conclusion, the discovered risk factors correspond to three operational risk factors: the estimated delivery window, seasonal load, and structural order attributes such as distance and weight. These factors vary in actionability. The estimated delivery window can deliver the most direct reduction in delay risk, however, this only reduces the measured delay instead of any underlying logistical factors.
The association between seasonal load and increased delivery delay is not actionable, except November, which shows a visible peak in historic order counts. This peak could be addressed by preparing logistical capacity based on internal order count data.
The structural factors are not addressable themselves, but rather serve as flags for a more challenging

order. These flags become visible at order time and could be used to calibrate the delivery window estimate with a potential buffer.
We see the estimated delivery window associated across all main patterns in the subgroups, i.e. a tight window with increased risk and a broad window with reduced risk. These insights are associative risk indicators gathered from one dataset, the discovered relationships are not causal.

## 6.4 Limitations

**Single Task and Dataset** While TURS discovered valuable subgroups, from which operational insights emerged, this is only achieved on one dataset, with one trained TURS model for binary delivery delay prediction. This also applies to the cost of interpretability we measured, as it is not clear what this cost would be for other problems on different datasets.

**No Causality** The findings in this study are associative, as causal discovery and inference are out of scope for this study. The discovered associative relationships support the identification of risks, but they do not show that taking action reduces the risk.

**No Bootstrap Confidence Intervals** The predictive performance for TURS and the baseline models are measured on the same hold-out test set, however, bootstrap CIs are not computed. Therefore, there exists a range of uncertainty in the predictive performance results.

## 6.5 Future Work

Future work could look at different tasks such as multi-class classification, and other or multiple datasets to study if TURS retains its ability to discover valuable subgroups among a large array of tasks and varying datasets.
Furthermore, an in-depth comparison between TURS and other modern rule-based models (CN2, CLASSY, DRS) could extend our work by not only benchmarking predictive performance, but by including interpretability benchmarking.
Finally, to further investigate the discovered risk factors, a causal analysis could find out if for instance the increased risk in tight delivery window orders is caused by faulty estimation calibration, or by a logistical effect.

# AI Usage Statement

The Large Language Model Claude (Anthropic) was used during the course of this thesis in the following ways:

- To discuss my ideas and explain complex concepts. All explanations are verified against the original sources and the literature.
- Debugging TURS2's source code after changes I made. All changes in the final fork of TURS2 were verified before being published.
- To improve my writing style by asking for feedback about sentence structure, and finding spelling and grammatical errors. The final text was written by me.

# References


[1] B. Ü. Küp, E. T. Küp, A. D. Yücekaya, M. Hekimoğlu, and G. Koçak, "Real-time prediction of delivery delay in supply chains using machine learning approaches," *Book of Abstracts and Full texts*, p. 46, 2024. [Online]. Available: https://dx.doi.org/10.2139/ssrn.5062672

[2] Karakaya, "Evaluation of machine learning and ensemble learning models for classification using delivery data," *Verimlilik Dergisi*, no. PRODUCTIVITY FOR LOGISTICS, p. 89–104, 2025. [Online]. Available: https://izlik.org/JA53SU63RA

[3] N. Salari, S. Liu, and Z.-J. M. Shen, "Real-time delivery time forecasting and promising in online retailing: When will your package arrive?" *Manufacturing & Service Operations Management*, vol. 24, no. 3, pp. 1421–1436, 2022. [Online]. Available: https://doi.org/10.1287/msom.2022.1081

[4] M. Lochbrunner, "Combining machine learning with human knowledge for delivery time estimations," Olten, 2021. [Online]. Available: https://irf.fhnw.ch/handle/11654/48594

[5] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," 2019. [Online]. Available: https://arxiv.org/abs/1811.10154

[6] G. Stiglic, P. Kocbek, N. Fijacko, M. Zitnik, K. Verbert, and L. Cilar, "Interpretability of machine learning-based prediction models in healthcare," *WIREs Data Mining and Knowledge Discovery*, vol. 10, no. 5, p. e1379, 2020. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1379

[7] L. Yang and M. van Leeuwen, "Probabilistic truly unordered rule sets," 2024. [Online]. Available: https://arxiv.org/abs/2401.09918

[8] L. Yang, S. L. van der Meijden, S. M. Arbous, and M. van Leeuwen, "Interpretable machine learning for identifying icu readmission risk in subgroups with probabilistic rules," *Journal of the American Medical Informatics Association*, p. ocaf171, 10 2025. [Online]. Available: https://doi.org/10.1093/jamia/ocaf171

[9] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth, 1984.

[10] W. W. Cohen, "Fast effective rule induction," in *Machine Learning Proceedings 1995*, A. Prieditis and S. Russell, Eds. San Francisco (CA): Morgan Kaufmann, 1995, pp. 115–123. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781558603776500232

[11] P. Clark and R. Boswell, "Rule induction with cn2: Some recent improvements," in *Machine Learning — EWSL-91*, Y. Kodratoff, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 151–163.

[12] H. M. Proença and M. van Leeuwen, "Interpretable multiclass classification by mdl-based rule lists," *Information Sciences*, vol. 512, pp. 1372–1393, 2020.

[13] G. Zhang and A. Gionis, "Diverse rule sets," *CoRR*, vol. abs/2006.09890, 2020. [Online]. Available: https://arxiv.org/abs/2006.09890

[14] G. Baryannis, S. Dani, and G. Antoniou, "Predicting supply chain risks using machine learning: The trade-off between performance and interpretability," *Future Generation Computer Systems*, vol. 101, pp. 993–1004, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X19308003

[15] Olist and A. Sionek, "Brazilian e-commerce public dataset by olist," 2018. [Online]. Available: https://www.kaggle.com/dsv/195341

[16] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," *PLOS ONE*, vol. 10, no. 3, pp. 1–21, 03 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0118432

[17] L. Yang, "TURS2: A refined implementation of probabilistic truly unordered rule sets," https://github.com/ylincen/TURS2, 2024.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[19] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[20] J. Kamerbeek, "TURS2: Fork with modifications for bachelor thesis on delivery delay prediction," https://github.com/Jessek43/TURS2/tree/thesis-delivery-delay, 2026, fork of https://github.com/ylincen/TURS2; modifications include feature name preservation, maximum-rules early stopping, and train/test split evaluation.

# A Hyperparameter Grids

## A.1 CART

Table 5: Hyperparameter Grid for CART.

| Hyperparameter | Values |
|---|---|
| `max_depth` | {3, 4, 5, 6} |
| `min_samples_leaf` | {20, 50, 100, 200} |
| `min_samples_split` | {50, 100, 200} |
| `criterion` | {gini} |
| `class_weight` | {None, balanced} |
| **Total combinations** | 96 |

## A.2 Random Forests

Table 6: Hyperparameter Grid for Random Forest.

| Hyperparameter | Values |
|---|---|
| `n_estimators` | {300, 500, 800, 1000} |
| `max_depth` | {None, 10, 15, 20, 30} |
| `min_samples_leaf` | {1, 3, 5, 10, 20} |
| `min_samples_split` | {2, 5, 10, 20} |
| `max_features` | {sqrt, log2, 0.3, 0.5} |
| `class_weight` | {None, balanced, balanced_subsample} |
| **Total combinations** | 4,800 |

## A.3 XGBoost

Table 7: Hyperparameter Grid for XGBoost. `Spw` $\approx 11.37$ is the ratio of negative to positive instances in the training set.

| **Hyperparameter** | **Values** |
|---|---|
| `n_estimators` | {300, 500, 800} |
| `max_depth` | {3, 4, 5, 6, 8, 10} |
| `learning_rate` | {0.01, 0.03, 0.05, 0.1} |
| `subsample` | {0.6, 0.8, 1.0} |
| `colsample_bytree` | {0.6, 0.8, 1.0} |
| `min_child_weight` | {1, 5, 10} |
| `reg_alpha` | {0, 0.1, 1.0} |
| `reg_lambda` | {1, 5, 10} |
| `gamma` | {0, 0.1, 1.0} |
| `scale_pos_weight` | {1, spw} |
| **Total combinations** | 104,976 |

# B Selected Hyperparameters

Table 8: Hyperparameters selected by 5-fold stratified cross-validation, using PR-AUC Scoring on the Training Set.

| **Model** | **Hyperparameter** | **Value** |
|---|---|---|
| CART | `max_depth` | 6 |
| | `min_samples_leaf` | 200 |
| | `min_samples_split` | 50 |
| | `class_weight` | "balanced" |
| | `criterion` | "gini" |
| Random Forest | `n_estimators` | 800 |
| | `max_depth` | None |
| | `min_samples_leaf` | 5 |
| | `min_samples_split` | 5 |
| | `max_features` | 0.5 |
| | `class_weight` | "balanced" |
| XGBoost | `n_estimators` | 500 |
| | `max_depth` | 10 |
| | `learning_rate` | 0.01 |
| | `subsample` | 0.6 |
| | `colsample_bytree` | 0.8 |
| | `min_child_weight` | 5 |
| | `reg_alpha` | 0 |
| | `reg_lambda` | 1 |
| | `gamma` | 1 |
| | `scale_pos_weight` | 1 |

# C Complete Rule Set TURS-MDL

Table 9 lists the complete TURS-MDL rule set learned on the training set. For each rule, the conditions, estimated late-delivery probability, and number of training instances covered are reported. The final entry shows the default prediction used when no rule applies. Rules are listed in the order they were learned by the algorithm.

Table 9: Complete TURS-MDL rule set (34 rules).

| # | **Conditions** | $P$(**Late**) | **Coverage** |
|---|---|---|---|
| 1 | estimated_delivery_days $<$ 22 $\wedge$ same_state $=$ 0 $\wedge$ seller_customer_distance $\geq$ 1093.35 $\wedge$ purchase_month $=$ 3 | 0.500 | 136 |
| 2 | estimated_delivery_days $<$ 19 $\wedge$ freight_value $\geq$ 16.11 $\wedge$ same_state $=$ 0 $\wedge$ number_of_sellers $=$ 1 $\wedge$ purchase_month $=$ 3 | 0.436 | 319 |
| 3 | estimated_delivery_days $\geq$ 28 $\wedge$ seller_customer_distance $<$ 816.48 $\wedge$ purchase_month $=$ 6 | 0.006 | 1,448 |
| 4 | estimated_delivery_days $\geq$ 23 $\wedge$ same_state $=$ 1 $\wedge$ shipping_window_days $<$ 10 $\wedge$ purchase_month $\notin \{2, 3, 11\}$ | 0.011 | 2,318 |
| 5 | estimated_delivery_days $<$ 28 $\wedge$ seller_customer_distance $\geq$ 1472.78 $\wedge$ purchase_month $=$ 3 | 0.377 | 321 |
| 6 | number_of_sellers $\neq$ 1 | 0.015 | 996 |
| 7 | estimated_delivery_days $<$ 30 $\wedge$ seller_customer_distance $\geq$ 1472.78 $\wedge$ purchase_month $=$ 11 | 0.312 | 333 |
| 8 | estimated_delivery_days $<$ 10 $\wedge$ seller_customer_distance $<$ 253.45 $\wedge$ payment_type $=$ boleto $\wedge$ purchase_month $=$ 8 | 0.355 | 248 |
| 9 | estimated_delivery_days $\geq$ 32 $\wedge$ seller_customer_distance $<$ 1093.35 $\wedge$ purchase_month $\notin \{1, 2, 3, 11, 12\}$ | 0.011 | 3,229 |
| 10 | estimated_delivery_days $\geq$ 16 $\wedge$ seller_customer_distance $<$ 551.72 $\wedge$ shipping_window_days $<$ 10 $\wedge$ purchase_month $\notin \{1, 2, 3, 4, 5, 11, 12\}$ | 0.019 | 10,075 |
| 11 | estimated_delivery_days $\geq$ 10 $\wedge$ freight_value $<$ 13.61 $\wedge$ payment_type $=$ credit_card $\wedge$ purchase_month $\notin \{2, 3, 5, 11\}$ | 0.024 | 6,736 |
| 12 | estimated_delivery_days $<$ 21 $\wedge$ freight_value $\geq$ 13.61 $\wedge$ same_state $=$ 0 $\wedge$ purchase_month $=$ 2 | 0.320 | 531 |
| 13 | estimated_delivery_days $\geq$ 22 $\wedge$ seller_customer_distance $<$ 1093.35 $\wedge$ purchase_month $\notin \{1, 2, 3, 4, 11, 12\}$ | 0.019 | 12,748 |

*Continued on next page*

*Table 9 continued from previous page*

| # | **Conditions** | $P$(**Late**) | **Coverage** |
|---|---|---|---|
| 14 | estimated_delivery_days $\geq 32 \wedge$ purchase_month $= 8$ | 0.010 | 580 |
| 15 | same_state $= 1 \wedge$ purchase_month $= 4$ | 0.029 | 2,644 |
| 16 | estimated_delivery_days $\geq 32 \wedge$ purchase_month $= 6$ | 0.008 | 1,657 |
| 17 | estimated_delivery_days $\geq 10 \wedge$ product_weight_g $< 1{,}000 \wedge$ seller_customer_distance $< 67.11$ | 0.026 | 5,301 |
| 18 | estimated_delivery_days $\geq 10 \wedge$ freight_value $< 19.01 \wedge$ purchase_month $= 7$ | 0.024 | 3,880 |
| 19 | estimated_delivery_days $\geq 38 \wedge$ product_weight_g $< 10{,}750$ | 0.022 | 3,229 |
| 20 | $26.00 \leq$ estimated_delivery_days $< 38.00 \wedge$ seller_customer_distance $< 1472.78 \wedge$ purchase_month $= 4$ | 0.031 | 1,695 |
| 21 | estimated_delivery_days $\geq 19 \wedge$ same_state $= 1 \wedge$ shipping_window_days $< 12 \wedge$ purchase_month $\notin \{2, 11\}$ | 0.021 | 6,393 |
| 22 | estimated_delivery_days $\geq 24 \wedge$ freight_value $< 13.61$ | 0.020 | 2,039 |
| 23 | estimated_delivery_days $< 23 \wedge$ same_state $= 0 \wedge 499.82 \leq$ seller_customer_distance $< 1093.35 \wedge$ purchase_month $= 3$ | 0.269 | 994 |
| 24 | freight_value $< 17.65 \wedge$ purchase_month $= 6$ | 0.021 | 3,389 |
| 25 | $30.00 \leq$ estimated_delivery_days $< 38.00 \wedge$ freight_value $< 34.15 \wedge$ seller_customer_distance $< 627.64 \wedge$ purchase_month $\notin \{3, 6\}$ | 0.024 | 2,230 |
| 26 | estimated_delivery_days $< 28 \wedge$ seller_customer_distance $\geq 816.48 \wedge$ purchase_month $= 2$ | 0.251 | 668 |
| 27 | estimated_delivery_days $\geq 12 \wedge$ seller_customer_distance $\geq 410.27 \wedge$ purchase_month $= 8$ | 0.031 | 3,634 |
| 28 | product_photos_qty $< 2 \wedge$ product_volume_cm3 $\geq 15{,}750 \wedge$ seller_customer_distance $< 1472.78 \wedge$ purchase_month $= 11$ | 0.204 | 1,171 |
| 29 | estimated_delivery_days $\geq 28 \wedge$ shipping_window_days $< 10 \wedge$ purchase_month $= 3$ | 0.173 | 960 |
| 30 | estimated_delivery_days $< 10 \wedge$ price_per_product $\geq 59.99 \wedge$ purchase_month $= 8$ | 0.229 | 936 |
| 31 | $12.74 \leq$ freight_value $< 16.11 \wedge$ purchase_month $= 3$ | 0.152 | 1,758 |
| 32 | estimated_delivery_days $< 23 \wedge$ seller_customer_distance $\geq 1093.35 \wedge$ purchase_month $\notin \{2, 3, 7, 8, 11\}$ | 0.238 | 533 |

*Continued on next page*

*Table 9 continued from previous page*

| # | **Conditions** | $P$(**Late**) | **Coverage** |
|---|---|---|---|
| 33 | product_weight_g $\geq$ 1,200 $\wedge$ same_state = 0 $\wedge$ purchase_month = 12 | 0.160 | 1,050 |
| 34 | freight_value $<$ 19.01 $\wedge$ same_state = 0 $\wedge$ purchase_month = 11 | 0.165 | 2,175 |
| – | *Default (no rule applies)* | 0.104 | 26,798 |