Universiteit Leiden

Master Computer Science

Truly Unordered Rule Sets for Interpretable Risk Estimation in the Retail Sector

Name:	Levi Peeters
Student ID:	2011174
Date:	February 13, 2025
Specialisation:	Artificial Intelligence
1st supervisor:	Francesco Bariatti
2nd supervisor:	Matthijs van Leeuwen

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In this thesis, we address the challenge of improving the Retail Risk Index, a metric used to assess the survival chances of new stores. We do so in collaboration with Locatus, who maintain a comprehensive dataset of 4.4 million retail venues in the Netherlands. Our goal is to use TURS (Truly Unordered Rule Sets), an interpretable machine learning method, to generate models that are accurate and understandable. However, the scale of this dataset presents significant computational challenges. To enable TURS to handle such large-scale data, we focus on improving its space and time complexity.

We first enhance TURS' memory efficiency by implementing sparse arrays, allowing it to process large datasets without memory issues. We then address the time complexity by introducing parallel processing techniques. Parallelizing the expansion of candidate rules results in a 7–8x speedup over the non-parallel case. Despite these improvements, we encounter challenges when applying TURS to the retail dataset and reduce the problem to the municipality of Utrecht, which consists of 70,000 instances. In the resulting rulesets, 66%–87% of instances are not covered by any rule, severely reducing the interpretability of the results. We propose and test three strategies to mitigate this issue, of which one results in minor improvements.

Finally, we evaluate TURS' predictive power against competing methods. Although TURS achieves ROC_AUC scores between 0.565 and 0.628, outperforming the Retail Risk Index (0.295–0.440), it still lags behind state-of-the-art methods like XGBoost (0.672-0.810). Given that the predictive scores are this low, we argue that interpretability of the results is secondary to accuracy for improving the trustworthiness of the model. As such, we conclude that TURS is not ready to be applied to this problems of this scale.

Contents

Chapter 1

Introduction

In recent years, machine learning has become a powerful and valuable tool in an increasing number of fields, including finances, healthcare and law. As machine learning models increase in size and complexity, their decision-making process becomes less transparent, making it harder for domain experts to follow the reasoning of the model and, subsequently, to trust the model's results. As such, there is a constantly rising need for more transparent models, especially in fields where the reasoning behind a decision is as important as the decision itself.

1.1 Interpretable Machine Learning

In the field of Transparent AI, we distinguish Explainable Artificial Intelligence (XAI) and Interpretable Machine Learning (IML). While the two terms are often used interchangeably, they represent different approaches towards the same goal [Mohseni et al., 2021]. Explainable AI concerns itself with so-called black box models, such as neural networks. They are powerful and widely applicable, but their decision-making process is incredibly complex and opaque. In order to aid in understanding the results produced by such models, XAI methods aim to give the user some understanding in the local search space around the result and which factors were important in reaching this result, without fully explaining the internal logic of the model.

In contrast, the philosophy behind Interpretable Machine Learning is not to retroactively explain the decisions made by an opaque model, but rather to design models with interpretability in mind from the start.

In this work, we focus on methods that discover probabilistic rules in the search space. Given a set of feature values X and a target feature Y, a probabilistic rule takes the form

If X meets certain conditions **THEN** $P(Y) = \hat{P}(Y)$.

Where $\hat{P}(Y)$ is the estimated probability distribution of instances that satisfy the rule. Each condition is a *literal*, which takes the form $X_i \in R_i$. For numerical variables, R_i represents a single interval. For categorical variables, R_i represents a value or a set of values.

Probabilistic rules are powerful in IML for two reasons. Primarily, their simple structure is relatively easy to understand and interpret for both data scientists and domain experts. Secondarily, the output of the rule is a probability distribution which gives immediate insight into

its uncertainty when used for prediction tasks, whereas non-probabilistic methods only output the prediction itself.

Models that make use of probabilistic rules are generally structured in one of three ways; decision trees, rule lists and rule sets. In a decision tree, each node contains a literal which is used to split the data into two subsets. Therefore, a decision tree can be seen as a set of rules, where each rule represents a path from the root node all the way to a leaf node. This seems like a very interpretable structure at first glance, but suffers from drawbacks as problems become more complicated. Decision trees for complicated problems grow in size and depth very quickly. In addition, rules in a tree structure share many nodes high up in the tree and will often contain multiple literals for the same variable, making it more difficult to understand the paths instances take.

Rule lists connect the rules in a $IF \dots THEN \dots ELSE\, IF \dots THEN \dots ELSE \dots THEN \dots$ structure. The main advantage of this approach is that, once a rule is found, the instances covered by this rule are no longer considered for future rules. This allows for efficient divide-and-conquer algorithms to learn rule lists effectively. However, the structure of a rule list is harder to interpret, as each rule in the list does not stand on its own. Instead, each rule consists of its own literals as well as the negation of all rules that precede it. As we look farther down the list, rules become harder to interpret as we need to keep track of an increasing number of negated rules.

In a rule set, each rule is a separate $IF \dots THEN \dots$ structure which requires no other rules to be understood, making them very easy to interpret for data scientists and domain experts. However, contrary to decision trees and rule lists, it is possible for rules to overlap. When an instance satisfies multiple rules, the results become harder to understand, as this instance then satisfies multiple distinct probability distributions. To some extent, this can be valuable to a user as it provides multiple arguments to support the result, giving insight into which elements are the most promising to change in order to improve a user's results. On the other hand, overlapping rules can also be contradictory or redundant, which makes interpreting them more difficult.

When it comes to the quality of individual rules, there are a few properties we look for. Rules with a high probability towards either outcome are preferable, as this informs the user with high certainty that their situation is valuable or not. We also prefer rules with high coverage, as this decreases the effects of variance on the probability distribution of the rule. However, more important is the amount of literals and their content. Rules with a single condition can be effective for prediction, but will not tell the user anything that they could not have learned using simpler methods. However, if the rule consists of a lot of conditions the rule can become difficult to interpret, especially if the literals concern features that the user considers unrelated.

The interpretability of the ruleset as a whole is mostly dependent on the quality of the individual rules, as the user will only need to work with the rules that apply to their situation. However, it is important that the ruleset covers as much of the search space as possible. Any instances not covered by any rules are part of the *else rule*. The else rule can be interpreted as the "background": Without any other rules, its probability distribution is the prior distribution of the data. As rules are added to the model, the leftover background distribution is refined further and further. As such, the else rule is unique in that it does not stand alone; it is the negation of all other rules combined. This makes the else rule nearly impossible to interpret in a meaningful way.

1.2 Locatus and the Retail Sector

Retail venues come in many shapes and sizes, ranging from small corner shops to giant clothing warehouses. When planning to open a new store, an entrepreneur needs to make sure this business is likely to succeed in making a profit. A big factor in this is their business plan, but there are many other factors to consider, such as properties of the new building, the state of the area they plan to open in and presence of competitors. Gathering all of this information and interpreting it correctly is a tedious task, which is made easier by Locatus[1].

Locatus is a company based in Utrecht that gathers its own data on stores and retail venues, including geographical data, store surface area, branch and footfall[2]. This information is prepared and made available to clients via online databases and digital maps. A team of field workers travels through the Netherlands in order to maintain the data by recording stores that change, measuring the floor area and taking pictures. The information provided by Locatus is of interest not only to new entrepreneurs, but to project developers, investors and consultants as well. In addition, Locatus tracks vacancy of storefronts, which is valuable to municipalities. Besides gathering all of this data, Locatus also helps clients interpret it through models and visualisations.

A question Locatus often gets is to present advice on the survival chances of retail venues. Perhaps the client wishes to open a new store, and wants to gain insight in how the surrounding area will impact their business. Or, a client with existing stores may ask why some of their stores are not doing well compared to others. In order to answer this question, Locatus currently makes use of a prediction model called the Retail Risk Index (RRI). This model, which is explained in more detail in Section 3.1, aggregates four indices that represent specific risks for the potential new store; a building score, a street score, an industry score and a market score. Based on these four indices, the client will gain some insight in how their final risk score came to be, but no further interpretation can be gained from the model.

While the most important factor for Locatus is accurately predicting the risk of a store failing, there is a secondary focus on interpretability of the model. When the RRI predicts that a store is at risk of failing, the client will naturally want to know where this risk comes from and what can be done to improve their chances. The current model offers some interpretability by dividing the RRI into four different areas. An explanation such as "this street received a low score" is actionable, but still quite vague. By making use of modern Transparent AI techniques, our goal is to design a more interpretable model for estimating the risk of failure, without sacrificing predictive performance of the model.

1.3 Thesis Outline

For this thesis, we collaborate with Locatus to design a new interpretable model to predict the survival chance of new stores. We make use of TURS [Yang and van Leeuwen, 2022], a novel method to discover rule sets. In Chapter 2, we review a number of other approaches to transparent AI. In Chapter 3, we further elaborate on the Retail Risk Index and how it is used for prediction tasks. We also explain some key properties of TURS that are important to understand the rest of the thesis. In Chapter 4, we explore Locatus' exhaustive retail

[1] `https://locatus.com/en/`

[2] Footfall is a measure of passersby in front of the store

database and construct the target feature for this prediction task. Chapter 5 is dedicated to our contributions to increase the scalability of TURS and allow it to process a larger subset of this large database. In Chapter 6, we apply this version of TURS to a subset of the Locatus data and evaluate the predictive power and interpretability of the resulting rule set models. Finally, in Chapter 7, we discuss a number of interesting results of this thesis and present some promising avenues for future work.

Chapter 2

Related Work

Methods in the field of transparent AI can be divided into two general approaches: *post-hoc* methods treat the model as a black box and aim to explain their results after training, whereas *ante-hoc* models are designed to be inherently interpretable. In the following section, we will give a short overview of both approaches. Furthermore, we give a more in-depth overview of rule-based methods, including decision trees, rule list methods and rule set methods.

Post-hoc. Models such as neural networks or random forests are considered black boxes: Data is entered into the box and the model returns a result, but the underlying process is far too complex for a human to be able to follow along. Post-hoc models for interpretability leave this black box intact, attempting to illustrate how the input and output of the model interact without explaining the internal decision-making.

LIME [Ribeiro et al., 2016] is one such method, which is used to explain individual predictions done by a model. LIME generates a new set of data consisting of perturbations of a single instance of input data. The generated data points are weighted based on their proximity to the original data point. The black box is probed with this data and produces labels, which LIME uses to train an interpretable surrogate model. This model can then be used to do interpretable predictions. Due to the distance-based weighting, the surrogate model becomes less trustworthy as input data moves out of the local area.

SHAP [Lundberg and Lee, 2017] also aims to explain a single prediction instance by comparing it to a baseline prediction. The black box model is probed with instances where a number of features are set to their baseline value. Repeating this for every subset of features gives an indication of how important each feature is in reaching this prediction.

Another approach is to generate a counterfactual explanation [Goodman and Flaxman, 2017], which is a statement in the form "If X had not occurred, then Y would not have occurred". In this case, Y is the prediction that we are trying to explain and X is one of the features that were used to make the prediction. For each of the features, this method perturbs the input until the output changes in a meaningful way, such as turning a negative prediction into a positive one. The counterfactual explanation to a prediction is thus defined as the smallest change in input that changes the prediction. The approaches provided in [Goodman and Flaxman, 2017], as well as a number of other approaches, are unstable: Small changes to input could have unpredictable results on the output, which again harms interpretability [Artelt et al., 2021].

More recent work has been done to make the process of generating counterfactuals more robust [Guyomard et al., 2023] by adding a robustness term to the optimisation problem. The counterfactual approach is particularly interesting for our problem because it is actionable: When a store is predicted to fail, a counterfactual can show the user how to increase their chances at success.

Ante-hoc. While post-hoc methods are effective at explaining a model's output, they can only be used to compare input configurations that are close to each other in the search space. The underlying model is still a black box, which does not explain its decision-making process and is therefore hard for domain experts to trust. In contrast, ante-hoc methods are designed with interpretability in mind from the start, meaning the model itself is interpretable instead of just local results. Linear models, such as linear regression, logistic regression or Support Vector Machines, are examples of interpretable models because of their simple mathematical structure. Clustering methods such as k-nearest neighbour also fall in this category, as the decision-making process is very simple to follow.

Decision trees. Methods based on decision rules, such as decision trees, also fall into this category as the models are easily interpreted by following the decision process. Algorithms to construct decision trees differ mostly on the metric they use to decide how to split the data: ID3 [Quinlan, 1986] uses information gain, C4.5 [Salzberg, 1994] uses gain ratio and CART [Breiman et al., 2017] uses the Gini index. Information gain is the total reduction in entropy after the split is performed, and thus shows how much information this split provides us about the distribution of the data. Information gain tends to choose features which have a large number of distinct values. When this is not preferred, the Gain Ratio can be used instead, which is Information Gain normalised by the number of instances. The Gini index is the probability of misclassifying a randomly labelled element, and tends to perform similarly to the Gain Ratio.

Rule lists. Rule lists structure the decision process as a sequence of IF-THEN-ELSE statements. CN2 [Clark, 1989] expands on ID3 to generate rule lists and is especially reliable in noisy domains. PART [Witten et al., 1998] expands on CART by inducing rule sets from decision trees. The lack of a global postprocessing step makes PART very efficient, while still achieving accuracy comparable to other methods in the field, such as C4.5. Bayesian Rule Lists [Yang et al., 2016] search for probabilistic rule lists that fit the data while also conforming to a set of priors, which favour short, interpretable rule lists. Particularly interesting for this thesis are Classy [Proença and van Leeuwen, 2020], which introduces the Minimal Description Length as the optimisation criterion for a greedy search, and SSD++ [Proença et al., 2022], which expands on this by using a heuristic beam search instead.

Rule sets. It is generally agreed that rule sets are more interpretable compared to trees and lists, as only rules that cover the instance are needed to explain a result. However, learning a set of rules that is unordered and diverse is a challenging problem. CN2 can be modified to produce unordered sets of rules [Clark and Boswell, 1991], but the rule sets are large and tend to contain many overlapping rules. RIPPER [Cohen, 1995] learns rules for each class sequentially, starting with the least frequent class and removing instances after all rules for that class have been learned. Because of this process, the rules for each class are mined with

different data, and the majority class simply becomes everything that is left over. As such, the rule sets RIPPER produces are not truly unordered. FURIA [Hühn and Hüllermeier, 2009] expands on RIPPER by learning fuzzy rules, which have soft decision boundaries that can stretch to cover uncovered instances. As a result of this rule stretching, many rules are often needed to explain the results of FURIA. IDS [Lakkaraju et al., 2016] uses an objective function which favours interpretable sets, i.e. smaller sets of smaller rules that have little overlap. DRS [Zhang and Gionis, 2020] uses a similar approach to generate diverse sets of rules by adding a diversity factor into the objective function. While both methods achieve good results, the rules they find are not probabilistic, which harms interpretability.

TURS [Yang and van Leeuwen, 2022], for Truly Unordered Rule Sets, is a recently developed algorithm that mines probabilistic rulesets with no inherent ordering.

Chapter 3

Background Knowledge

This section introduces a few topics that become important in the later chapters of this thesis. We first discuss the Retail Risk Index, the current method Locatus uses to measure risk, and how it is compared to TURS for evaluation. We then discuss the Minimum Description Length principle via a basic example, and how TURS uses this principle to discover rulesets.

3.1 Retail Risk Index

The Retail Risk Index was first developed by Joris van der Loo [J.M. van de Loo, 2013]. This is a decision tree model, where data is split on three conditions. The first split is made on how well the store's branch of industry is doing. One can imagine that, for instance, with today's shift towards of digital shopping and distribution, a video rental store has a very high risk of going out of business. In contrast, entertainment businesses such as restaurants and services such as hairdressers are not affected by this shift nearly as much. Then, the building that the store will open in is evaluated based on the history of businesses that have operated from it. Venues receive a good score if the stores that occupied them stayed for a long period of time, and will receive penalties for long vacancies or frequently switching businesses. The final split is made on the state of the surrounding area. While almost all stores will benefit from being situated in a busy area, it is highly dependent on the industry which factors are important for it. For example, clothing stores generally benefit from an area where there are already a number of clothing stores present. Most shoppers do not visit a single store to buy what they need, but will prefer to browse many stores before deciding what to buy. As such, a clothing store with no competitors near it will not attract many shoppers. In different industries such as supermarkets, where shoppers can buy everything they require in one store without a need or desire to browse, this effect is reversed and the presence of many competitors is seen as a disadvantage.

These scores are calculated based on the historical performance of stores and discretized into five categories, from very low to very high risk of failure. Once per year, the Retail Risk Index is evaluated by comparing the number of stores that failed in each category. The details of this process are not shared publicly.

In order to compare the Retail Risk Index to other methods, the risk score needs to be converted to a normalized metric that we can use in a prediction task. The RRI ranges from 50 (low

risk) to 150 (high risk). Locatus reports that stores with the lowest risk have a probability of 5.1% to go out of business. For stores with the highest risk, the probability is 31.5%. For fair comparison to other methods, we scale and invert the RRI to range between 0 and 1. After this transformation, the highest risk score will correspond to the lowest survival chance reported by Locatus and vice-versa. Using this probability, we can evaluate the RRI on prediction tasks in Chapter 6. Note that, as the highest probability for failure presented by Locatus is only 31.5%, the Retail Risk Index will always predict that a store will succeed for at least one year.

3.2 Model selection using MDL

The Minimal Description Length (MDL) principle is a model selection criterion first introduced by [Rissanen, 1998]. The idea behind the method is a common property of statistical models and compression schemes; both aim to discover regular patterns in data in order to produce a good model. The principle of MDL is to translate statistical models to encoding schemes for data, and select the model that minimises the total code length $L(D)$:

$$L(D) = L(M) + L(D|M) \tag{3.1}$$

Here, $L(M)$ is the code length of the encoded model and $L(D|M)$ is the code length of the data given the model. By including the length of the model L(M) in the computation, the MDL principle aims to avoid overly complex and specific models that fit the data very well (and thus have a low $L(D|M)$), but would be difficult to interpret. Considering both terms strikes a balance between complexity of the model and its ability to fit the data, and ensures that a simpler model will always be preferred if it has the same predictive power. This way, MDL models are naturally resistant to overfitting.

This method works on probabilistic models, such as probabilistic rulesets, by assigning shorter codes to outcomes that are more likely. We calculate the code length of each outcome i, with probability $p(i)$ by using:

$$l_i = -log_2(p_i) \tag{3.2}$$

Note that this equation does not produce an integer code length, which means the resulting encoding scheme could not actually be used to transmit the information. Fortunately, in MDL, we don't actually have to transmit any information; we are only interested in minimising the *theoretical* code length of our data and model. Codes in MDL-based approaches are solely used as a means to compare models.

We illustrate the usage of MDL on a probability distribution using a simple example; a series of coin tosses. Suppose we toss our coin ten times and observe the sequence $HHTHHHHHTH$. In order to encode this sequence, we could naively assign a 1 to heads and a 0 to tails, which means the encoding would take 10 bits. However, we can compress the sequence more efficiently by making use of the observed probability distribution. Using equation 3.2, we can compute the code lengths l_H and l_T associated with the outcomes.

$$l_H = -log_2(0.8) \approx 0.32$$

$$l_T = -log_2(0.2) \approx 2.32$$

The total code length of the sequence now becomes $8 \cdot 0.32 + 2 \cdot 2.32 = 7.2$, reducing the total code length of the sequence.

The more imbalanced the sequence becomes, the more compression we can gain by implementing the above method. In the worst case, where both outcomes are equally likely, both outcomes are assigned a code length of 1 and we are back to the naive solution. In other words, *the more certain we are of the outcome of our probabilistic model, the more compression we can achieve*.

3.2.1 TURS

The TURS algorithm [Yang and van Leeuwen, 2022] is a rule-based model which uses MDL for evaluating candidate rule sets. This section describes the process by which TURS grows a single rule. We first cover how we cut features into discrete chunks to be used as literals in rules. Then, we walk through the process of growing a rule, describing which properties TURS prioritises in choosing which rules to expand. Finally, we cover the two-beam approach and the role of the "auxiliary beam".

Cut Points As discussed in Chapter 1, rules are built out of literals which cut the search space into two parts; one part satisfies the literal and belongs to the rule's coverage and the other part does not. In order to perform its rule expansion, TURS requires a predefined list of all cut points that can be made.

Categorical features are one-hot encoded, creating a binary column for every value the feature can take. Each of these binary columns has a single cut point, splitting data into the part that has that value and the part that hasn't.

Numerical variables can have a very large or even infinite number of potential cut points, which need to be discretised into a small set for TURS to iterate over. Naively, we could decide to space the cut points evenly over the range of the feature. This will work fine if the values are distributed uniformly, but if the distribution is skewed we encounter a problem. If there are few instances between two cut points, there will be very little difference in the coverage between the two options if they are used in a rule. In the most extreme case where the segment is completely empty, the two literals will be the same in practice. To avoid this issue, TURS uses quantiles to decide the cut points, so that each resulting segment is populated with an equal number of instances. That way, we can reasonably expect there to be a significant difference in coverage between two neighbouring instances. While any number of quantiles will theoretically work, TURS generally uses 20.

Rule Growing TURS grows rules step by step, starting with an empty rule. The algorithm iterates through every literal and calculates the learning speed score $r(S)$, defined as the reduction in code length this rule would achieve per extra covered instance. The score is calculated per extra covered instance to prevent the algorithm from selecting rules with large overlap. As TURS iterates through the literals, it keeps track of the literals with the highest $r(S)$ through a beam search. After all potential literals have been evaluated, the most promising W candidates are kept and the rest are discarded.

Should we just choose the W candidates with the highest $r(S)$, we can run into an issue with diversity between rules: If we take our best rule and change one of its literals to the next available cut point, the resulting new rule is likely to have a very similar coverage to the original rule, and as such will have a very similar $r(S)$. As TURS always considers every potential literal, it is reasonable to expect that some of the W rules with highest $r(S)$ will be very similar. As a consequence, the search process becomes greedier and less likely to escape a local minimum.

TURS prevents this issue by implementing *patience diversity*. Because TURS starts with an empty rule and adds literals one by one, every added literal reduces the coverage of the growing rule. As such, as the rule's coverage becomes smaller, there is fewer room for future improvement and the rule becomes greedy. If the coverage is still large, there is more room for improvement and the rule is 'patient'. TURS encourages diversity in the search process by categorising the candidates based on coverage and keeping the rule with the largest $r(S)$ in each category for the next iteration. These rules are expanded further in the next iteration, and are also kept in a list containing all candidate rules. When a stopping criterion is met, the rule with the highest $r(S)$ from these two sets is chosen and added to the ruleset, after which the whole process repeats.

The Auxiliary Beam The learning speed score $r(S)$ is designed to deter TURS from selecting rules with a large overlap in coverage. While this is desirable behaviour, it can lead to a problem during the rule-growing process, which is shown in Figure 3.1. In this figure, a simulated search space is shown in which two rules have already been added. There is a very promising region for a third rule in the bottom right. However, because TURS adds one literal per step, both of the literals that can lead TURS to this third rule will have a large overlap with the existing rules. This means the $r(S)$ of the candidate rule will suffer and the new rule is not likely to be found.

The role of the auxiliary beam is to keep track of potential rules with this property. The beam search follows the same process as the regular beam, but using the complementary score $R(S)$ instead of $r(S)$. $R(S)$ is defined as the reduction in code length achieved if only previously uncovered instances are considered. As such, it does not punish potential rules for overlapping with existing rules and allows them to be expanded further. From each coverage category, the rule with the largest $R(S)$ is kept for the next iteration as well as the rule with the best $r(S)$. At the end of the search process, the rule with the largest $r(s)$ is added to the ruleset.

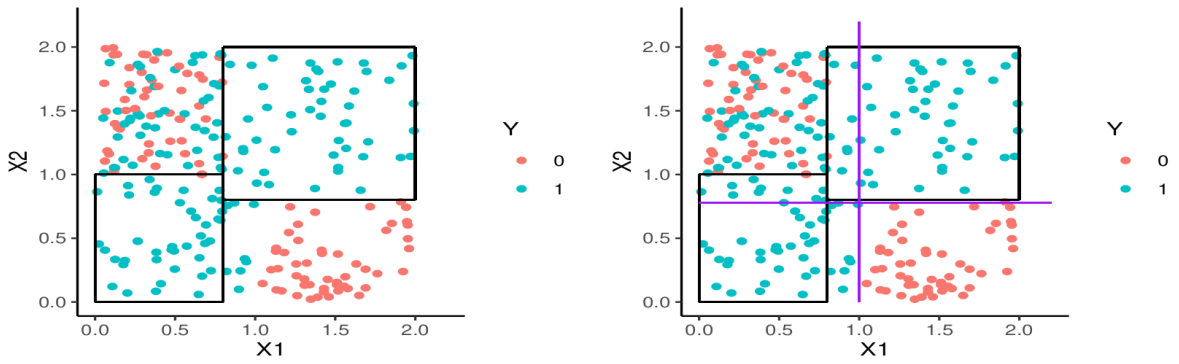


Figure 3.1: (Left) Simulated data with a rule set containing two rules (black outlines).(Right) Growing a rule to describe the bottom-right instances will create conflicts with existing rules. E.g., adding either X1 $>$ 1 (vertical purple line) or X2 $<$ 0.8 (horizontal purple line) would create a huge overlap that deteriorates the likelihood. Figure and caption from [Yang and van Leeuwen, 2022].

Chapter 4

Data

The data studied in this paper is an extensive dataset of historical information on retail venues. This section describes some relevant properties of the data and preprocessing steps.

Locatus has been maintaining the historical retail dataset since 2004. Each year, over 200.000 records are added, putting the total number of records at 4.4M in 2023. Of the 68 features present in the database, sixteen are selected to be used for our classification problem. Features are excluded for being too specific (e.g. street and house number), irrelevant to the problem (e.g. names of stores) or having too many missing values. A list of all features is included in Appendix A.

Features While some features in the included list are self-explanatory (e.g. house number), some others are defined by Locatus and require some explanation.

The *Retail Floor Area* (RFA) is defined as the area of the store that is available to customers and used for shopping. Back rooms, elevator shafts and bathrooms are not considered part of the RFA. while the area behind the counter, showcases and fitting rooms are.

Footfall is an approximation of the amount of people that walk by the store. Note that this is different from the number of people that enter a store, which is not a metric that Locatus can record.

Locatus records the *shopping area* that a store is in. A shopping area is defined as any grouping of at least five stores in close proximity. In this work, we make use of the names of the areas as well as three features that describe them: *Main type*, *subtype* and *visit motive*.

Main Shopping Area Type classifies shopping areas. The most important shopping area in a city is designated as a "central shopping area", and others are designated "supporting areas". Stores that are not considered part of a shopping area are designated as "dispersed retail". *Shopping Area Subtype* further divides shopping areas based on the number of stores they contain.

Visit Motive is a different classification of shopping areas based on the way customers use them. Most stores fall into the *comparison* type. These are, for example, clothing stores, where customers tend to browse many items while only purchasing a few. The second largest category is *convenience*, such as supermarkets, where customers go to buy items they need without browsing much. Shopping areas are classified on the dominant type of store they

contain and their total surface. A third category, *specialized*, contains shopping areas with large-scale retail venues, such as home improvement stores, car dealerships or large furniture stores. These types of shopping areas are typically found on the outskirts of cities.

We also make use of the *subcentre* that a store is in. Subcentres are also groupings of stores, but they differ from shopping areas in that they are designed to be so, usually by the municipality, and they have an assigned name. A subcentre is always considered a shopping area, or a part of one. Malls and train stations with retail venues are examples of subcentres.

Missing values Most features are recorded for every store. In some cases, however, there are missing values which need to be considered. Footfall is only recorded in areas where it is expected to be high, such as large city centres. As such, when footfall is missing it can be assumed that it should be low. As such, missing footfall values are set to 100 passerby per hour.

The database records which shopping area and mall each store is part of. When a store is not in a shopping area or mall, this feature is set to a separate category "missing".

We use the median to fill missing values for Retail Floor Area, because the distribution is skewed; most stores have a small RFA, but large stores such as warehouses have such a large RFA that they strongly influence the mean.

4.1 Construction of the Target Feature

In order to construct a model that predicts the survival chance of retail venues, a target feature needs to be constructed. This work investigates two different approaches to the target feature.

Short-term target Each building in the dataset is given a separate entry each year, recording which store occupied the building at that time. The entry for the same building in the next year indicates whether this store survived in this building or not. Therefore, the yearly target feature is set to 1 if the store is still present one year later, and set to 0 if it isn't. Stores that disappear from their building have not necessarily closed; it is also possible that the store occupying this building moved to a different location. Whether to assign a positive or a negative score to the old building can be debated. A store moving to a different venue can be positive, indicating that the store was doing very well and moved to a larger or more expensive venue. However, it could also be an indication that the store was not doing well at the old building and moved out of necessity. Based on the experience of domain experts at Locatus, we hypothesise that a store moving is more likely to be a negative indicator for the old building, and as such they are assigned a negative target value.

Vacant retail buildings are registered by Locatus as well, but they have no predictive power for this problem, and are removed from the dataset.

An important consequence of this approach to the target feature is that it becomes skewed; most stores survive much longer than a year before closing, which causes a store to have multiple records with a positive target and one, or none, with a negative target. As a result the target feature is 88% positive. A skewed data distribution makes data mining problems more challenging, which becomes apparent in Chapter 6, where we study the resulting rulesets.

Long-term target In order to mitigate the effects of this skewed distribution, we investigate a different way of constructing a target feature. Instead of looking only a year ahead, we look at the total number of years during which a store operates from a venue. We then produce a binary target which is positive if it survived at this location for three years. This way, the target feature is still skewed, but less so than with the yearly target, with 75% of instances positive. This approach is also beneficial because it reduces the total size of the dataset, thus allowing us to process larger subsets of data with a given time budget. We also argue that this feature is more useful to this problem: In the context of retail business, one year is not a particularly long time. In the first year of operation, a business is investing and establishing itself in its environment. Even if the store is going to fail, it it not unreasonable to expect it to survive at least one year. As such, failing stores can still have positive entries in the dataset. Aggregating each store to one entry and setting a threshold for three years of survival prevents this undesirable effect.

The downside of this approach is that the other features need to be aggregated from an entry each year to one entry per store. This is not a problem for all features. For example, we would not expect the branch of a store to change during its lifetime. For many features, however, staticness of the features is not so obvious. Footfall and population of the city can change over time, as can Retail Floor Area should the building be renovated. The shopping area a store belongs to is not likely to change, but may still do so when shopping areas are rezoned or redefined.
In constructing the dataset, the median is used to aggregate numerical features, and the mode is used to aggregate categorical features, as this gives an indication of this store over its lifetime at this location. An argument can also be made for using the most recent entry instead, as they are the most indicative of the current situation.

4.2 Size analysis

In Chapter 5, we show the performance of TURS on subsets of Locatus data. The time it takes to complete a TURS run is heavily dependent on which features are included; when expanding a rule, TURS evaluates each potential literal that can be added to the rule in question. Numerical features have a fixed number of potential cut points that is set as one of its hyperparameters. For categorical features, however, no cut point can be defined; instead TURS has to evaluate each category separately. As such, categorical features with a lot of possible values have a large impact on the runtime of TURS. Table 4.1 shows the number of possible values each feature has in the full dataset. This table shows that the number of possible values ranges from tens of values to tens of thousands. Removing only the largest of the features reduces the number of cut points to be considered substantially.

With this in mind, we define three sets of features to be investigated. The first is the set of *relevant* features, the features that will be used in Chapter 6 to mine and evaluate rulesets on Locatus data. These are features we deem the most relevant for this problem and that do not add an excessive amount of cut points. To produce the *reduced* set of features, we remove the categorical features with a large number of possible values, as well as two numerical features, in order to reduce the total number of cut points. For the *inflated* set of features, we add postcodes to the set of relevant features, adding a large number of cut points. This set is used to test the limits of our approach.

Feature	# of values	Relevant	Reduced	Inflated
Sh. Area Main Type	4	X	X	X
Inner City	4	X	X	X
Group	9	X	X	X
Sh. Area Type	12	X	X	X
Visit Motive	15	X	X	X
Main Sector	35	X	X	X
Sector	217	X		X
Chain	3.220	X		X
Sh. Area	5.052	X		X
Subcentre	5.199	X		X
Postcode	84.964			X
Total Cut Points for Utrecht		1 631	84	3 210

Table 4.1: The categorical features that are considered in this work, with the number of values that each feature can take in the full dataset.

As running TURS on the entire dataset is infeasible, as further discussed in Chapter 5, we limit this work to one municipality: Utrecht. Utrecht is small enough that experiments can be performed in a reasonable time, but also large enough to expect to find interesting patterns and rules. Utrecht spans 71.150 entries for the short-term target, which when reduced to one entry per store becomes 11.631 retail venues.

Chapter 5

Scaling TURS for Larger Datasets

TURS is a relatively new approach to rule-based learning, and it has not been widely applied. Experiments on sample datasets provided by the UCI Machine Learning Repository [Kelly et al.,] show that TURS can mine accurate and interpretable rulesets effectively, but given the small size of the UCI datasets, the scalability of the algorithm to larger data remains uncertain. When we apply TURS to the dataset provided by Locatus, we encounter two main bottlenecks: Memory usage and time complexity. In the following sections, we describe our contributions to the scalability of TURS.

5.1 Reducing Memory Complexity

As described in Section 3.2.1, categorical features need to be one-hot encoded in order for TURS to work with them. On large datasets with categorical features that can have many values, this increases the size of the dataset substantially as each possible value takes up a full column of space. The Locatus dataset, with all rows and relevant features, already takes up 4.7 GB in memory. Were we to one-hot encode it entirely, this space increases to an estimated 2.47 TB. However, the one-hot encoded array is very sparse. It can be stored much more efficiently if we do not record every one-hot encoded value, but only the values that are not zero, alongside the indices of their location in the array.

An implementation of this type of sparse array is provided by the Scipy library [Virtanen et al., 2020]. Because TURS evaluates the data column by column, we use the column variant `csc_array`, which allows for fast column-wise slicing. Table 5.1 shows the memory requirements to load the datasets used in this thesis, with and without one-hot encoding.

Additionally, in the original implementation of TURS, each candidate rule would contain an array containing the instances that it covers. This is useful, because the covered instances are used in subsequent rule evaluations. With large datasets, however, this causes extra problems with memory usage, especially because a new rule, with no literals, covers the entire search space and therefore contains a full copy of the dataset. To prevent this, each rule now no longer contains its own covered instances, but retrieves them from the full array when needed. This takes extra time, but prevents memory issues from crashing TURS.

After implementing sparse arrays and preventing TURS from making copies of the dataset, no further memory issues are present. While these improvements are nothing major, they are an

		One-Hot Encoded	
Feature set	Original	Dense	Sparsified
Reduced	5.69	49.52	7.68
Relevant	9.68	933.49	13.66
Inflated	10.25	1832.25	14.51

Table 5.1: The amount of memory (GB) each dataset requires when normally loaded, one-hot encoded and sparsified. While the original size is relatively similar, the one-hot encoded size increases substantially when large categorical features are added. The sparsified matrices reduce this effect, but still take a bit more memory than the original array, as they need to store indices as well as the data.

important step in applying TURS to real-life data, allowing the method to move from research into real-life applications.

5.2 Non-Deterministic Results

In the final stages of this master project, we discovered an issue where the results of TURS are not fully deterministic. This is not an inherent property of TURS, which is a deterministic algorithm and in the earlier versions used in this project, the problem does not occur. This issue went unnoticed for the duration of the project, as the cause of this non-deterministic behaviour does not occur frequently and the divergent results will all be valid and fairly similar rulesets. As such, we were unable to fix this problem, but we have narrowed down where the problem seems to occur.

In Appendix B, we provide three rulesets produced on the same dataset. The first and second rulesets are almost identical, except for the cut point that was chosen for the first literal of the first rule, where Ruleset B.1 covers a strictly smaller area in the search space compared to Ruleset B.2. However, their coverage is the same. We can only conclude that, in the area covered by Ruleset B.2 but not Ruleset B.1, there are no instances at all. As such, the two rules are practically identical.

Unfortunately, this behaviour can cause cascading effects in some cases. Rule three in Appendix B is also a result of the same experiment. The same two rules are found again, but this time TURS also discovers two extra rules. Upon following the decision-making in the early stages, we notice that the *inhabitants* feature is again the start of the diverging behaviour.

The *inhabitants* feature has an interesting property when used on this dataset. The municipality of Utrecht consists primarily of the city of Utrecht, and a few smaller towns in the area. Most instances will therefore have the number of inhabitants of Utrecht in this feature, which changes every year, as seen in Figure 5.1. Because of the quantile method used to decide on the cut points for numerical features, this can cause cut points to be very close to one another or even overlap.

We believe that this property of the way cut points are decided is part of the reason why the non-deterministic behaviour occurs. With that being said, the problem did not occur in earlier versions of our implementation, and is unlikely to be an inherent flaw in the algorithm but rather a mistake made in this project.

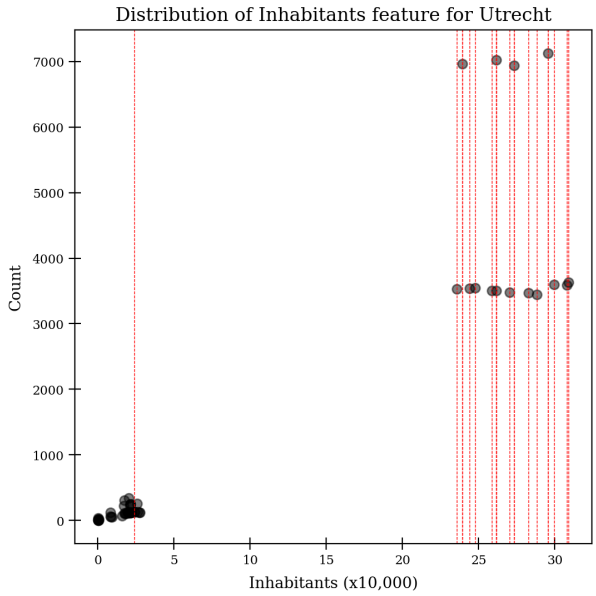


Figure 5.1: Distribution of the number of inhabitants over the Utrecht dataset. Red dashed lines show the cut points calculated by TURS. Notice how there are 15 dashed lines, while TURS calculates 20 cut points, indicating that some are duplicated.

Due to time constraints, the issue was not resolved. As the diverging rulesets produced by our implementation are similar and all valid models, we believe the following results are still valuable. In the rest of this chapter, we present various experiments that show the runtime of TURS. Because the runtime of TURS is largely dependent on the amount of rules it finds, we present these runtimes normalised by the number of rules, so that a fair comparison on runtime is possible.

5.3 Parallel Processing

The second bottleneck that prevents us from applying TURS to the Locatus dataset is the time required to run even small subsets. Figure 5.2 shows the runtime, depending on an increasing number of datapoints, of TURS on the municipality of Utrecht for the three feature sets defined in Chapter 4. TURS will only be able to process this dataset within the time budget of twelve hours when using the reduced feature set. If using all relevant or inflated features, the amount of rows needs to be reduced further to stay within the time budget.

To increase the amount of data TURS can reasonably process, we make use of parallelisation. We consider various approaches to implementing this, which we discuss in the following sections.

5.3.1 Parallel Beam Searches

The first approach to parallel processing is to perform the main beam search and the auxiliary beam search in parallel, as shown in Figure 5.3. Each of the two beams evaluates a set of candidate rules and selects a set of most promising rules. They require no information from the other beam, and two resulting sets of rules are combined after the beam searches are complete.

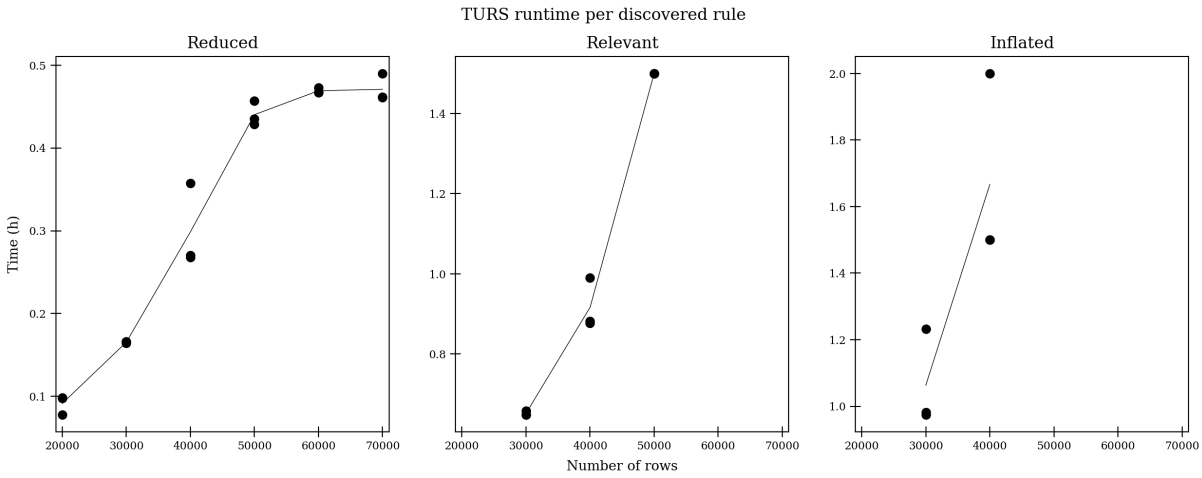


Figure 5.2: Runtime of TURS before implementing any parallel processing. When using the relevant or inflated feature set, the timeout limit of 12 hours is reached before a full ruleset is discovered.

As such, this is an easy approach to implement. It also carries a theoretical limit: By running two parallel processes, the processing speed can be at most doubled. Upon implementing this approach, a speedup close to two was indeed observed.

5.3.2 Parallel Generation of Literals

In order to reach a more significant speedup, parallel processing can be applied to different areas of TURS. As TURS expands a rule, each literal that could be added is evaluated and added to the beam search. Because there are many literals to be evaluated for each rule, and the process of evaluating them is completely isolated from the other rules, we implement the process of expanding a rule in parallel. A schematic view is shown in Figure 5.4a.

One of the main challenges of parallel processing is the overhead caused by the serialisation of required data and the communication of this data to worker processes. As this has to be done for every task, parallel processing becomes more efficient when there is little data to be shared and more calculations to be done. As evaluating a single literal is a relatively small task, and there are many of these tasks to accomplish, sending each task to a worker individually would result in a large amount of overhead. To reduce this effect, we send the tasks to each worker in chunks.

Naively, we might choose to divide the number of tasks by the number of available workers and send chunks of this size to each worker. In the ideal case, where each task takes the same amount of time, this would be the optimal strategy. However, if the time taken per task varies, this strategy will cause some processes to finish earlier than others. As there are no more tasks to do, all processes will have to wait on the slowest process to finish before the algorithm can move on to its next iteration.

As such, a balance needs to be struck in the size of the chunks. It needs to be large enough to cause a significant reduction in overhead, while still being small enough to allow for efficient scheduling so that workers don't spend as much time waiting on one another.

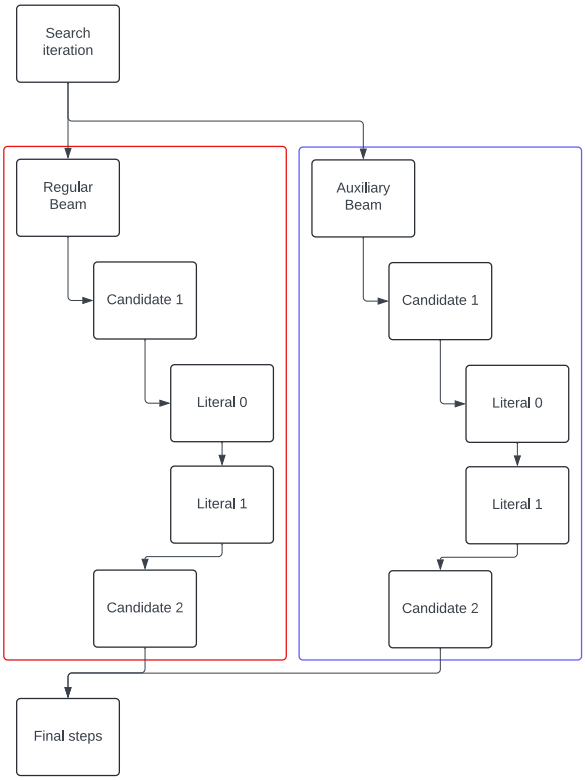
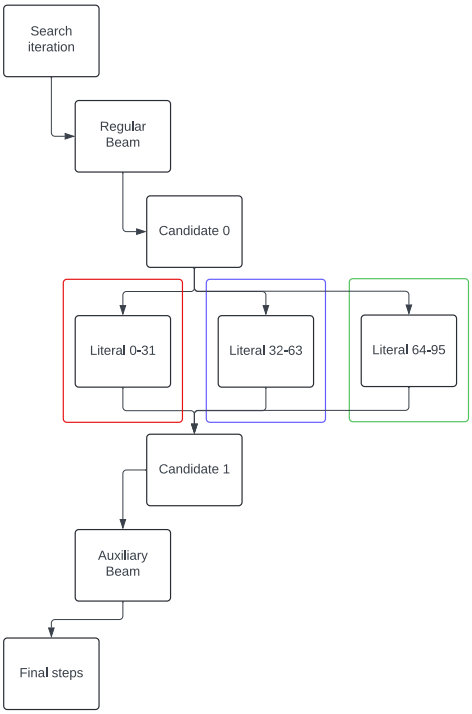


Figure 5.3: A schematic view of TURS with the two beam searches ran in parallel. Each beam processes the candidate rules, and each of their candidate literals, sequentially.

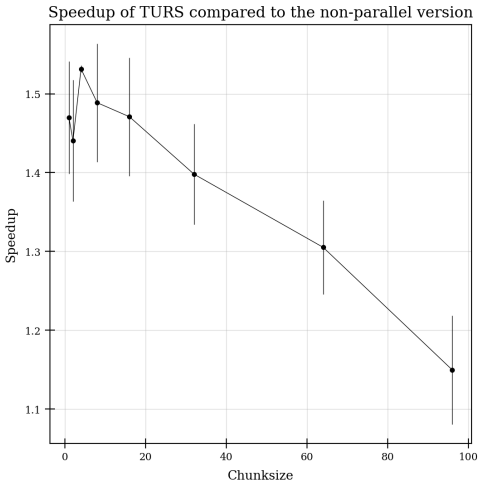
We show the effect of the chunksize by running the following experiment. We run TURS on the Locatus dataset, using 50 000 instances located in Utrecht. Each run has 16 workers available, and we run each chunksize setting three times. Figure 5.4b shows the speedup compared to the sequential case. From this figure, we see that the speedup with respect to the non-parallel version is not high. Performing the beam searches in parallel yields a speedup of roughly 2 times, while this approach reaches about 1.5 times at the peak. The highest speedup is obtained with a low chunksize, with values between 1 and 20 scoring similarly well. This is remarkable, as in this dataset each rule will have 1 631 cut points to be evaluated. This indicates that the scheduling issues from choosing a large chunksize are much more important for the runtime than the extra overhead from sending multiple chunks of data.

5.3.3 Parallel Expansion of Candidate Rules

The third approach to parallel processing that we investigate is to assign each worker a complete rule to expand, as shown in Figure 5.5a. The process of expanding a rule involves evaluating each potential literal that can be added, which requires no information about the other candidate rules. Only at the end of the expansion process are the most promising candidate rules combined into the beam. Because each candidate needs to be expanded twice (once for the main beam and once for the auxiliary beam), we can at most use $2 \cdot n\_candidates$ workers. This introduces an upper limit to the speedup that can be achieved, but a much higher one than running only the beams in parallel. In this work, we consider ten candidate rules per iteration. As such, the theoretical limit on our speedup is 20 times faster than the non-parallel

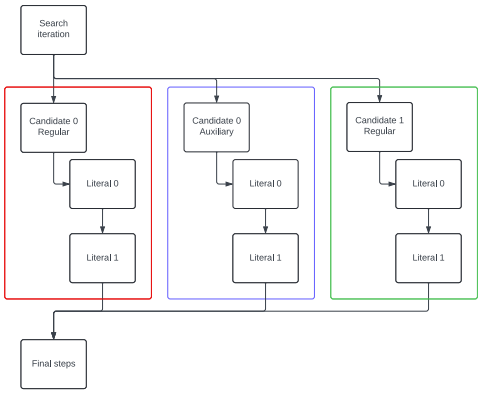


(a) A schematic view of TURS where the literals that could be added are evaluated in parallel. Because each literal is a small task and there are many literals to be evaluated, the tasks are processed in chunks.

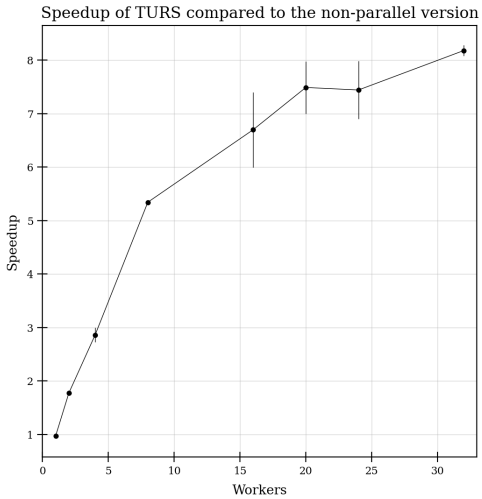


(b) Speedup of TURS when literal generation is performed in parallel, compared to the non-parallelised version.

Figure 5.4



(a) A schematic view of TURS where each rule is evaluated in parallel. Each rule must be evaluated twice, once in the main beam and once in the auxiliary beam.



(b) Speedup of TURS when rules are generated in parallel, compared to the non-parallelised version. Each point is averaged over three runs. Error bars represent the standard deviation.

Figure 5.5

case.

This approach to parallel processing can introduce the same problem that was discussed when parallelising the literals in Section 5.3.2: If the time taken to expand a rule varies, the workers with quicker tasks will have to wait for the longest task to finish. To investigate the effect of this, we run our experimental setup again, this time varying the amount of workers available. Results of this experiment are shown in Figure 5.5b.

When only given access to a single worker, this experiment is equivalent to the sequential implementation with the added overhead from transmitting and receiving data from the child process. As we see a speedup close to 1 for this case, we can conclude that this approach does not cause a significant amount of overhead of this type. The speedup does not increase proportional to the amount of workers, indicating that there is overhead caused by processes waiting on one another. As the number of workers approaches 20, we see diminishing returns, as expected. While there appears to be another increase in speedup when the number of workers is increased to 32, we believe this to be caused by variance, as there is no reason for workers beyond the number of tasks to add anything.

5.4 Time Complexity of Different Parallelisation Approaches

We investigate the duration of a run of TURS for various configurations. As discussed in Chapter 4, we use three different feature sets to see the effect of the number of features. The relevant features are the features we also select for the data mining problem in Chapter 6. The reduced feature set removes some of the categorical features with many cut points. The inflated set adds the categorical feature *postcodes*, which can take a large number of values, thus adding a large number of binary columns. We investigate the Utrecht dataset with an increasing number of rows and compare TURS without parallel processing, with parallel processing by literals and parallel processing by rules.

Results are shown in Figure 5.6. Due to the non-deterministic results, discussed in Section 5.2, we present the runtime normalized by the number of rules that were found, so that results can be compared fairly. From these results, we can conclude that the parallel expansion of rules is much faster than the other two approaches, regardless of the size of the dataset.

From these experiments, we conclude that the best configuration for TURS is to use the implementation where rules are expanded in parallel, with at least 20 available workers. This configuration is used in subsequent experiments.

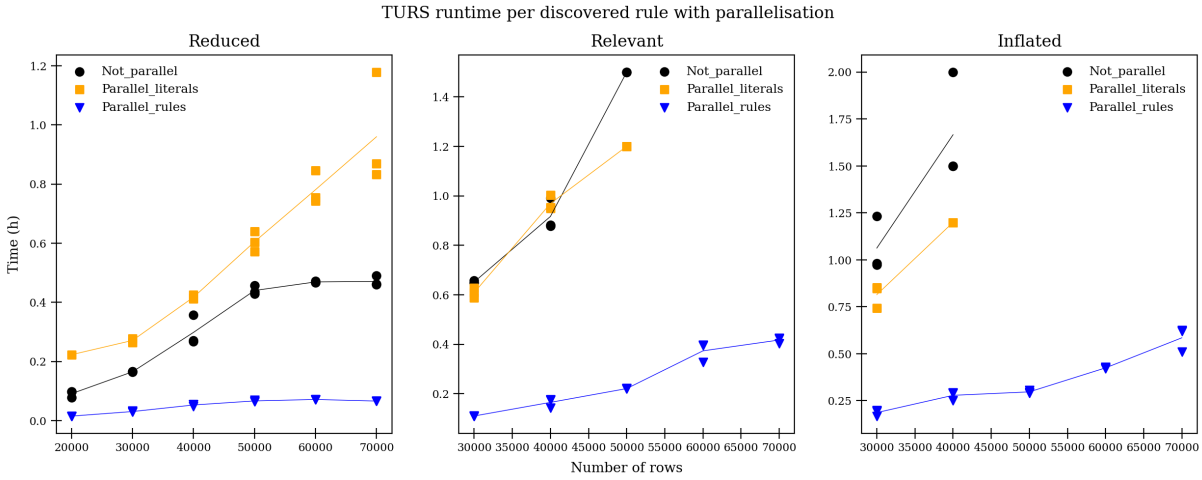


Figure 5.6: Comparison between the non-parallel version of TURS and the two approaches to parallelisation. Each configuration is run three times, and the mean is shown.

Chapter 6

Applying TURS to Locatus data

This chapter investigates the contents of the rulesets that are produced by TURS on Locatus data. We use the same experimental setup as before; investigating the municipality of Utrecht. Utrecht strikes a good balance between being small enough to be able to run experiment in a reasonable time, while also being a large enough city where interesting pattern are likely to emerge. We use both versions of the target feature discussed in Section 4.1, one where each store is represented by an instance for every year that it existed, and one where each store is condensed into a single instance. We use the set of relevant features as shown in Table 4.1.

In Section 1, we discuss some properties of rules that we consider beneficial for interpretability. We prefer rules with high coverage and a probability distribution that is as non-uniform as possible, as these properties show certainty of the result. The most important property for interpretability is the amount and content of the literals: Rules with more literals are preferred, as they provide more information. Specifically rules with a single literal are too simple, as they are easily produced using much easier data analysis methods.

We investigate the rulesets TURS produces on this setup, and find that the skewed data distribution discussed in Chapter 4 harms the quality of the resulting rulesets. As such, we also investigate the effect of various modifications to the algorithm. We evaluate the resulting models on interpretability and predictive performance.

6.1 Rulesets produced by unmodified turs

Short-term target When applying TURS to the Locatus data where the target feature looks one year ahead, we receive the ruleset as shown in Listing 6.1.

Of the 18 discovered rules, we see that six have a probability distribution that is more certain than that of the else rule. Most of the rules have a probability distribution that is "weaker"; a set of instances with a positive probability of 0.51, such as rule 01, can't be encoded efficiently and will contribute more code length to the model. TURS selects these rules because it takes these difficult instances out of the else rule. Due to the imbalanced distribution of our target feature (see Chapter 4) this small improvement to the else rule's probability distribution is worth adding a "weak" rule to the ruleset.

There are also a few rules (e.g. Rule 11) whose probability distribution is more certain

compared to the else rule. These rules each consist of a single value that can be taken by a categorical variable, in some cases accompanied by a literal on Population. While these rules are strong from an MDL perspective, they are rather uninteresting from a subgroup discovery perspective, as showing the success rate of stores from a given store chain or shopping area is easily done using simpler methods.

Listing 6.1: Ruleset produced by an unmodified TURS on the short-term target

```
01: Population >= 261550.0;
    Store Chain == Independent;
    Subcentre == Hoog Catharijne;
    Main Branch != Food;
    Then: P(1) = 0.51, coverage: 168

02: Population >= 295445.0;
    Sh. Area Main Type ==
        Spread out shopping;
    Longitude >= 5.06;
    Inner City != Miscellaneous;
    GROEP != Cultuur & Ontspanning;
    Then: P(1) = 0.65, coverage: 591

03: Store Chain == EAR&EYEMUSIC;
    Then: P(1) = 0.0, coverage: 5

04: 258690.0 <= Population < 295445.0;
    Subcentre == Miscellaneous Utrecht;
    Then: P(1) = 0.99, coverage: 717

05: Branch == Misc. Foodstuffs;
    Then: P(1) = 0.57, coverage:49

06: 282740.0 <= Population < 288120.0;
    Sh. Area Main Type ==
        Spread out shopping;
    Then: P(1) = 0.73, coverage:709

07: Store Chain == Albert Heijn;
    Then: P(1) = 0.98, coverage: 332

08: Visit Motive Type == Convenience - XL
;
    Then: P(1) = 0.76, coverage: 243

09: Subcentre == Miscellaneous Utrecht;
    Then: P(1) = 0.96, coverage: 745

10: 270175.0 <= Population < 299395.0;
    Branch == Employment Agency;
    Then: P(1) = 0.69, coverage: 207

11: Population >= 299395.0;
    Subcentre == Miscellaneous Utrecht;
    Then: P(1) = 0.97, coverage: 726

12: Store Chain == Kruidvat;
    Then: P(1) = 0.97, coverage: 272

13: Longitude < 5.13;
    Inner City == City with Inner City;
    Store Chain == Independent;
    Main Branch !=
        Jewellery & Optics, Fastservice;
    Branch != Hairdressers;
    Visit Motive Type != missing;
    Then: P(1) = 0.84, coverage: 8303

14: 273130.0 <= Population < 295445.0;
    Visit Motive Type == Comparison - XL;
    Subcentre != none;
    Then: P(1) = 0.66, coverage: 369

15: Population >= 261550.0;
    5.11 <= Longitude < 5.12;
    Store Surface Area < 100.0;
    Visit Motive Type == Comparison - XL ;
    Subcentre == none;
    Main Branch != Drinks;
    Branch != Restaurants, Hairdressers;
    Then: P(1) = 0.81, coverage: 2797

16: 239425.0 <= Population < 244095.0;
    Sh. Area Main Type ==
        Spread out shopping;
    Store Surface Area < 174.0;
    Then: P(1) = 0.72, coverage: 597

17: 270175.0 <= Population < 273130.0;
    Sh. Area Main Type ==
        Spread out shopping;
    Subcentre == none;
    Then: P(1) = 0.82, coverage: 1393

18: 235745.0 <= Population < 270175.0;
    Main Branch == Private Services;
    Shopping area !=
        Miscellaneous Utrecht;
    Then: P(1) = 0.82, coverage: 1343

If none of above,
    Then: P(1) = 0.9, coverage: 38226 (66%)
```

Long-term target When using the three-year target, the rules produced are different in nature. With the short-term target, we see TURS discover weak rules which are worth adding because they reinforce the else rule. With this target, we see "stronger" rules, with a probability distribution that is less uniform. A number of subcentres appear to have no stores that survived for three years (e.g. Rule 1), and five other rules have found a subgroup of stores with a very low change to survive for three years (e.g. Rule 4). Even though the prior distribution for this target is weaker, we still see a large fraction of stores remain in the else rule: $8\,124$ of $9\,305$

Listing 6.2: Ruleset produced by an unmodified TURS on the long-term target

```
01: Subcentre == Groeneweg
       /Laan van Nieuw Guinea;
    Then: P(1) = 0.0, coverage: 16

02: Subcentre == Nachtegaalstraat
       /Burgemeester Reigerstraat;
    Then: P(1) = 0.0, coverage: 15

03: Subcentre == Balijelaan/Rijnlaan;
    Then: P(1) = 0.0, coverage: 12

04: Visit Motive Type == missing;
    Sh. Area Main Type !=
       Spread out shopping;
    Then: P(1) = 0.14, coverage: 69

05: Subcentre == GWC Kanaleneiland (s);
    Then: P(1) = 0.0, coverage: 9

06: Population < 239425.0;
    Visit Motive Type == Comparison - XS;
    Then: P(1) = 0.2, coverage: 99

07: Population < 239425.0;
    Sh. Area Type == Inner City;
    Then: P(1) = 0.32, coverage: 357

08: 282740.0 <= Population < 299395.0;
    Inner City == Inner City;
    Subcentre != none;
    Then: P(1) = 0.27, coverage: 81

09: Subcentre == overig Utrecht;
    Then: P(1) = 0.99, coverage: 196

10: Branch == Garage;
    Then: P(1) = 0.94, coverage: 232

11: Population < 239425.0;
    Sh. Area Main Type == Supporting;
    Longitude >= 5.11;
    Visit Motive Type != Convenience - M;
    Then: P(1) = 0.23, coverage: 123

If none of above,
    Then: P(1) = 0.79, coverage: 8124 (87%)
```

6.2 Modifications to TURS

While the rulesets we investigate in Section 6.1 contain some interesting patterns, we see a lot of instances remain in the else rule. Because the else rule is the negation of all other rules, it is the only rule that cannot be interpreted by itself. As such, instances in the else rule are harder to interpret compared to instances that are in a rule. In addition, most of the rules TURS does discover consist of one or two literals corresponding to a simple pattern that we could expect a domain expert to already be aware of.
In order to produce more interpretable models using TURS, we investigate three modifications designed to discourage TURS from reinforcing the else rule: Forcing TURS to consider the else rule to be perfectly uniform, a second run of TURS using only instances from the else rule and a probability threshold which forces discovered rules to be less uniform. Table 6.1 contains descriptive metrics of each ruleset for easy comparison. The rulesets produced using each of the following methods are included in Appendix C.

Table 6.1: Descriptive metrics of the rulesets, to compare them on interpretability.

Model	# of rules	# of literals per rule	Coverage of rules	Coverage of else rule
Short-term target				
Unmodified	18	3.22±2.17	1 087±1 863	38 226 (66%)
Probability Threshold	4	1.00±0.00	7 418±9 386	27 839 (48%)
Rerun Else Rule	26	3.00±1.96	903±1 596	35 125 (61%)
Force Else Rule Probability	101	1.14±0.49	86±222	49 919 (88%)
Long-term target				
Unmodified	11	1.82±1.11	109±106	8 124 (87%)
Probability Threshold	7	2.00±1.41	54±64	8 962 (96%)
Rerun Else Rule	19	1.79±1.00	107±121	7 563 (81%)
Force Else Rule Probability	5	1.00±0.00	599±800	7 081 (76%)

6.2.1 Force Else Rule Probability

We can discourage TURS from reinforcing the else rule by forcibly setting its probability distribution to $[0.5, 0.5]$, the least efficient distribution for MDL encoding. By doing this, we completely prevent TURS from rejecting rules because they weaken the else rule and it will keep accepting the best candidate rule.

Short-term target On the dataset with an entry for each year, we see clearly the downside of this approach: TURS finds 101 rules, but nearly all of them are very specific, consisting of a single literal that is often a single store chain or subcentre that is doing very well. This type of rule covers very few instances, so the else rule remains very populated despite the number of discovered rules.

Long-term target It is surprising that the same effect does not occur when we use this method on the three-year target. Instead of finding many rules with low coverage, it only discovers five, less than any other modification, and the rules have relatively high coverage. Of the four approaches, this removes the most instances from the else rule (see Table 6.1), which is positive. However, the rules are not that helpful to a beginning entrepreneur: Rules 1 and 3 give a single successful branch or shopping area, respectively, while Rules 2, 4 and 5 just apply to miscellaneous stores around the city.

6.2.2 Rerun Else Rule

We can extend the ruleset discovered by TURS by running the algorithm a second time and only giving it the instances that ended up in the else rule. In doing so, we reset the code length of the model, forcing TURS to find additional patterns within the instances that were initially grouped together. We observe a similar result for both datasets, where a small number of additional rules is mined. We can see that the coverage and number of literals of the rules are very similar to the models produced by an unmodified TURS. While these extra rules do improve the model somewhat, from an interpretability standpoint, the problems with the models from the unmodified TURS still persist; many instances are still left in the else rule

and many rules consist of one or two literals that would not explain much to a beginning entrepreneur.

6.2.3 Probability Threshold

TURS is predisposed to discover rules with a weak probability distribution, because these rules strengthen the else rule. We are interested in learning whether there are possible strong rules to be learned as well; are the patterns being overshadowed by the weak rules, or are they not there at all? In order to learn this, we can add a threshold to TURS so that rules can only be added if their highest probability is higher than that of the else rule.

Short-term target On the dataset with one instance per year per store, we see that TURS has discovered four rules. Two of these represent a single store chain each, one of whom is doing very well while the other has no records of surviving stores. The other two rules are very broad and have a large coverage, leading to this model having the fewest instances in the else rule. While this is a positive, the two rules that the instances end up in instead are not that helpful either, describing a very general property that a lot of stores follow (e.g. Rule 1 applies to all stores outside of the inner city).

Long-term target For the dataset with one instance per store, the constraint of having a major probability above that of the else rule is much more lenient, as the prior distribution of the data is less imbalanced. As such, it is not surprising to see that in this case, the modified version performs similar to the unmodified version. The modified version finds four fewer rules, and the rules it does find are very similar or identical to those the unmodified model finds. No additional rules are discovered, but four of them are prevented from being added, causing this version to have the largest else rule of each of the versions on this dataset.

6.2.4 Summary of Modification Results

The main problem with using TURS for this project remains across all modifications; many instances are not part of any discovered patterns and end up in the else rule. Only the probability threshold on the short-term target has less than half of the instances in the else rule, and even then the four rules that are found have very little information to interpret. In addition, on the long-term target this approach has close to all instances in the else rule, indicating that the modification is not a reliable solution. Rerunning with only instances in the else rule improves on the unmodified case by discovering a few more rules which remove instances from the else rule, but the resulting model still has the same problems as the unmodified case. Forcing the else rule to be uniform shows wildly different behaviour on the two targets, flooding the ruleset with specific, low-coverage rules in the short-term case while only finding a few very broad rules in the long-term case.
For interpretability, rerunning the else rule results in the most valuable ruleset. However, no ruleset discovered by any modification is really able to mitigate the problems when using TURS on this dataset.

6.3 Comparing to CLASSY

CLASSY [Proença and van Leeuwen, 2020] is a method that uses MDL to discover rule lists. Rule lists differ from rule sets in that, whenever a rule is discovered, the covered instances are removed from the dataset and no longer considered for future rules. Rules cannot be interpreted alone, as they also imply the negation of all rules that precede it. Comparing the content of rules is difficult, as the two approaches diverge as soon as the first rule is found.
Interestingly, even the first rule is not the same between both approaches. TURS first discovers a nearly uniform rule with a coverage of 168, whereas CLASSY discovers a with much higher coverage which almost entirely consists of positive instances. When searching for the first rule, the data still looks the same, so it is surprising to see two MDL-based models find rules that are so different.
On the long-term target, the two approaches behave more similarly. TURS first discovers a few values for *subcentre* that contain no positive instances, which CLASSY is unable to do. TURS rule 4, however, looks very similar to the first rule discovered by CLASSY, although CLASSY adds an extra literal that removes 21 instances, half of which were positive, to make the rule only contain negative instances.
On the short-term target, the else rules of CLASSY and TURS look very similar, with a similar coverage and probability distribution. TURS discovers 18 rules, where CLASSY discovers 6. On the long-term target, CLASSY is able to remove far more instances from the else rule, although most of the rules with high coverage appear near the end of the list, making these rules hard to interpret.

Listing 6.3: CLASSY, Short-Term Target

```
If
    258690.0 <= Population < 261550.0;
    Visit Motive Type == missing;
    Store Chain == Independent;
    THEN Pr(1) = 0.99, coverage: 997
ELSE IF
    Visit Motive Type ==
        Comparison - XL;
    Subcentre != none;
    Population >= 282740.0;
    Footfall < 150.0;
    THEN Pr(1) = 0.61, coverage: 241
ELSE IF
    Inner City != City met Inner City;
    Shopping area ==
        Miscellaneous Utrecht;
    THEN Pr(1) = 0.96, coverage: 1385
ELSE IF
    Footfall >= 150.0;
    Store Chain != Independent;
    Inner City != Inner City ;
    THEN Pr(1) = 0.93, coverage: 3585
ELSE IF
    261550.0 <= Population < 295445.0;
    Subcentre != none;
    THEN Pr(1) = 0.81, coverage: 2237
ELSE IF
    261550.0 <= Population < 295445.0;
    Store Surface Area < 80.0;
    THEN Pr(1) = 0.83, coverage: 3296
ELSE
    Pr(1) = 0.88, coverage: 38064 (67%)
```

Listing 6.4: CLASSY, Long-Term Target

```
If
    Sh. Area Type !=
        Spread out shopping;
    Visit Motive Type == missing;
    Inner City !=
        City with Inner City;
    THEN Pr(1) = 0.0, coverage: 48
ELSE IF
    Population < 241645.0
    Longitude >= 5.1137;
    Subcentre == none;
    THEN Pr(1) = 0.09, coverage: 68
ELSE IF
    Population < 241645.0
    Visit Motive Type == missing;
    5.0881 <= Longitude < 5.1191;
    100.0 <= Footfall < 7125.0
    THEN Pr(1) = 0.28, coverage: 248
ELSE IF
    Sh. Area Type == Inner City;
    Population < 241645.0;
    THEN Pr(1) = 0.38, coverage: 231
ELSE IF
    Population >= 282740.0;
    Visit Motive Type == missing;
    THEN Pr(1) = 0.96, coverage: 744
ELSE IF
    282740.0 <= Population < 297420.0;
    5.1031 <= Longitude < 5.1138;
    Inner City !=
        Woonplaats met binnenstad;
    THEN Pr(1) = 0.29, coverage: 86
ELSE IF
    Visit Motive Type == missing;
    Store Chain == Other;
    THEN Pr(1) = 0.89, coverage: 1019
ELSE IF
    Population >= 297420.0;
    Store Chain == Independent;
    THEN Pr(1) = 0.91, coverage: 382
ELSE IF
    241645.0 <= Population < 270175.0;
    Store Chain != Independent;
    THEN Pr(1) = 0.87, coverage: 660
ELSE IF
    270175.0 <= Population < 297420.0;
    Store Chain == Independent
    THEN Pr(1) = 0.67, coverage: 1479
ELSE
    Pr(1) = 0.75, coverage: 3176 (34%)
```

6.4 Evaluating Predictive Power

Table 6.2: Evaluation metrics for the short-term target.

Name	Accuracy	ROC_AUC	Precision	Recall	F1 score
Majority Class	**0.878**	0.500	0.878	**1.000**	**0.935**
CART	0.815	0.547	**0.889**	0.902	0.896
RandomForest	0.863	0.581	0.879	0.979	0.926
XGBoost	**0.878**	**0.672**	0.879	0.999	**0.935**
CLASSY	**0.878**	0.432	0.878	**1.000**	**0.935**
RRI	0.867	0.404	0.867	**1.000**	0.929
RRI (imputed)	**0.878**	0.440	0.878	**1.000**	**0.935**
Unmodified TURS	**0.878**	0.565	0.878	0.999	**0.935**
Forced Else Rule Probability	**0.878**	0.534	0.878	**1.000**	**0.935**
Probability Threshold	**0.878**	0.556	0.878	**1.000**	**0.935**
Rerun Else Rule	**0.878**	0.516	0.878	**1.000**	**0.935**

Table 6.3: Evaluation metrics for the long-term target.

Name	Accuracy	ROC_AUC	Precision	Recall	F1 score
Majority Class	0.752	0.500	0.752	**1.000**	0.859
CART	0.756	0.667	**0.834**	0.843	0.839
RandomForest	0.767	0.627	0.783	0.960	0.861
XGBoost	**0.808**	**0.810**	0.826	0.943	**0.881**
CLASSY	0.790	0.292	0.798	0.965	0.874
RRI	0.777	0.295	0.777	**1.000**	0.874
RRI (imputed)	0.752	0.363	0.752	**1.000**	0.859
Unmodified TURS	0.782	0.628	0.785	0.976	0.871
Forced Else Rule Probability	0.752	0.589	0.752	**1.000**	0.859
Probability Threshold	0.776	0.590	0.776	0.989	0.869
Rerun Else Rule	0.760	0.516	0.763	0.988	0.861

Tables 6.2 and 6.3 show evaluative metrics of TURS compared to a naive baseline and a number of comparable methods. These scores are obtained using 5-fold cross-validation with 20% of data reserved for the test set in each fold. Implementations for CART and RandomForest are from the Scikit-learn package [Pedregosa et al., 2011], XGBoost [Chen and Guestrin, 2016] and CLASSY [Proença and van Leeuwen, 2020] are from their packages. Each method uses the default hyperparameters from their respective implementation.
For the short-term target, what immediately stands out is the scores of the majority class. Its accuracy, recall and F1-score are matched by some methods, but never improved.
These metrics only evaluate the predictions made by the model, without considering how certain the model is of these predictions. The ROC_AUC measures how robust these decisions are, and can therefore be considered the most important metric. For the ROC_AUC, XGBoost is the clear best performer, with random forest following with a score that is nearly 10 percentage point lower and rule-based models scoring even lower.
For the long-term target, XGBoost again scores significantly higher on ROC_AUC, and this time for accuracy as well. As such, when predictive performance is a priority, XGBoost should be chosen over TURS or other methods
Comparing the different implementations of TURS, we see that the modifications perform worse than the unmodified version. This is not surprising, as the modifications are designed to improve interpretability without predictive performance in mind. Rerunning on the else rule, the modification which is most promising for interpretability, suffers the most in terms of ROC_AUC for both targets.
In terms of computation time, all of the methods except for TURS compute their results in a matter of minutes.

6.5 RRI as a predictive model

This section will evaluate how the Retail Risk Index performs as a predictive model. The RRI was introduced in 2014, and as such any entries in the dataset from before 2014 do not have an RRI score. These scores cannot retroactively be calculated, as the precise implementation of the RRI is not public. The fact that these scores are missing was not considered when we made the datasets used for evaluation. As such, when we use them to evaluate the RRI, the data that is actually used for evaluation is different then when we use

these datasets to evaluate TURS and other methods. As time does not allow rerunning all of the experiments, we present two evaluations of the RRI. The row "RRI" in Tables 6.2 and 6.3 only evaluates the instances that actually have an RRI score, disregarding the rest. This is the most fair evaluation of the RRI, but is hard to compare to the other methods. The row "RRI (imputed)" instead evaluates the RRI when missing values are filled with the mean score.

Most of the scores obtained by the RRI are identical to the Majority Class. This is expected, as discussed in Section 3.1, as the RRI always predicts positively by definition. The ROC_AUC shows how well the RRI performs when the positive threshold is shifted. For both targets, the RRI performs similar to CLASSY and significantly worse than all other methods, even performing worse than the majority class. It is reasonable to state that, even if the instances from 2004-2014 were included in the evaluation of the RRI, that the method would underperform competing methods.

Chapter 7

Discussion and Conclusion

In Section 6.2, we evaluate the improvement to interpretability when applying various modifications to TURS, concluding that the most valuable modification for interpretability is to rerun the else rule with a blank MDL score. In Section 6.4, we compare the performance of various TURS modifications to those of XGBoost, Random Forest and CART, concluding that all modifications to TURS perform worse than the unmodified version. XGBoost outperforms all version of TURS easily, while CART and Random Forest perform slightly better or equally to TURS, depending on the situation.

We should now ask the question: Is it worth using rerunning the else rule to gain more interpretable results at the cost of predictive power? The modification adds eight rules for both targets, removing about 5% of instances from the else rule. In return, this modification performs the worst out of all in terms of ROC_AUC (barely outperforming the majority class) and second worst in term of the other metrics. Therefore, we believe that in its current state, the modification is not worth using. Interpretable results are valuable as a way to make results more trustworthy, but there is little merit in interpretability if the correctness of the result can't be trusted to begin with. Because TURS and its modifications still perform far below their alternatives, correctness of the predictions should still have priority over extra interpretability.

This leads into the next question: Is TURS worth using over XGBoost at all? While the predictive power of TURS on the long-term target is respectable, it still performs a lot worse than XGBoost. The resulting ruleset allows for some interpretability of the results, but 87% of the instances have ended up in the else rule, which offers no interpretability without knowledge of every rule in the model. Given all of this, we believe that the extra predictive performance of XGBoost is much more valuable in increasing the trustworthiness of a prediction.

The most important reason for TURS' inability to discover good rules is the skewed distribution of data, which is discussed in Chapter 4. MDL-based methods like TURS discover rules by identifying areas in the search space where the probability distribution is more certain, which allows for efficient encoding and thus reduces the total code length of the model. As such, MDL-based models suffer badly from the skewed distribution of this data, as any patterns in the data need to be very certain in order to actually improve the overall distribution and be accepted as a rule. In addition, the total scale of the

dataset is far too large for TURS to process, and only a fraction of the data was used in this work. We believe that TURS is a very promising method for interpretable machine learning, but that applying it to a dataset this large and this skewed is currently too ambitious.

From our results in Section 6.4, however, we find that the Retail Risk Index performs rather badly as a predictive model, obtaining the worst ROC_AUC scores across all similar methods. While TURS is not an eligible candidate to replace the RRI, we do believe that improvement is very achievable via a method like XGBoost. While XGBoost in itself does not allow for much interpretability, post-hoc methods discussed in Chapter 2 could mitigate this issue.

7.1 Future Work

This section discusses a number of venues for future work following this thesis.

To improve upon the RRI with modern data science techniques, we conclude that TURS is not sufficient. However, our results show that current state-of-the-art methods, particularly XGBoost, can improve upon the RRI significantly. These methods do not struggle with the size of this dataset nearly as much as TURS and can be realistically be trained on the entire dataset. XGBoost does not offer much for interpretability of the results, but we argue that accuracy of the predictions should take precedent over interpretability techniques to increase the trustworthiness of the result. Post-hoc techniques such as LIME can offer limited interpretation of results.

Further testing of TURS on real-world scenarios should continue on a smaller scale. We believe TURS is a powerful and useful method for data science applications, but it is simply not ready to be applied on this scale. While parallelisation improved the significant runtime issues of TURS, experiments on large volumes of data still require long computation times. As a result of this, we were unable to test TURS on available data, instead focussing on a small subset where not all available information could be used.

MDL-based methods such as TURS struggle with data that has a skewed target distribution, as this means the data can be very efficiently encoded before any rules are added. Data preprocessing techniques, such as resampling, can be applied to reduce the effect of skewed data. Undersampling (removing instances in the majority class) could be particularly interesting for this problem, as it reduces the total amount of data. As TURS currently is incapable of using all data, this could be advantageous. However, it also leads to a loss of information. In contrast, oversampling increases the size of the minority class by duplicating instances, further increasing the amount of data but conserving all information.

Finally, we see some potential for applying pruning techniques on the rulesets TURS produces. Generally, it is advantageous for TURS to discover "nested rules", where one rule is fully contained in another rule in the search space, if the probability distributions of the two rules are different. This can be seen as an exception, allowing for a reasoning such as: If (condition A), you are not likely to succeed, unless also (condition B), then you are likely to succeed. Consider for example Rules 9 and 11 from Listing C.3. Rule 9 consists of a single

literal (miscellaneous subcentre), while Rule 11 uses that same literal as well as a second one, population. However, the probability distributions and coverages of the two rules are nearly identical. Reducing these rules to one rule reduces the length of the model encoding, allowing for a less redundant rule to be added.
Pruning can also be considered for literals. TURS adds literals to a candidate rule sequentially. It is not unlikely that later additions to the rule may make earlier literals redundant. For example, TURS could select a range for "Population" as the first literal, and then a city within this range as its second literal, thus making the first literal redundant. Pruning rules just before adding them to the final ruleset could again free up code length for additional rules.

7.1.1 Conclusion

From this work, we derive the following conclusions.
Our first main contribution is to improve the space and time complexity of TURS so that it can be applied to real-world data science problems. We improve TURS' memory usage by implementing sparse arrays and a number of smaller improvements, allowing TURS to process large datasets without memory issues. To improve time complexity, we implement parallel processing via two different approaches. Parallelizing the expansion of literals produces a significant amount of overhead and yields little to no improvement on runtime. Parallelizing the expansion of rules does lead to improvement, where we observe a speedup of 7 to 8 times compared to the non-parallel case.
We then apply TURS to a real-world data science application in cooperation with Locatus. Using their exhaustive retail dataset, containing 4.4M instances of retail venues in the Netherlands, we attempt to use TURS to improve the Retail Risk Index, a risk factor for the survival chances of new stores, and evaluate the interpretability and predictive power of the resulting models. With the above solutions implemented, we are not able to process all of this data and reduce the problem to the municipality of Utrecht, consisting of 70 000 instances.
We find that TURS has issues producing interpretable rulesets on this data. While a number of interesting rules are discovered, we also see 66% to 87% of instances classified in the else rule, severely harming interpretability. We implement three modifications to mitigate this effect: Running TURS a second time on instances in the else rule, forcing TURS to consider the else rule distribution to be uniform and setting a threshold on the major probability of discovered rules. Of these modifications, only rerunning the else rule shows (minor) improvements to the issue.
We evaluate the predictive power of TURS compared to other methods in the field. Without modifications, TURS achieves a ROC_AUC between 0.565 and 0.628, with the modifications each scoring worse. This is an improvement over the Retail Risk Index, which scores between 0.295 and 0.440. However, competing methods in the field score much better, especially XGBoost, which achieves ROC_AUC scores between 0.672 and 0.810. In addition, XGBoost computes its model in a matter of minutes.
Given that the predictive scores are this low, we argue that interpretability of the results is secondary to accuracy for improving the trustworthiness of the model. As such, we conclude that TURS is not ready to be applied to this problems of this scale.

Bibliography

[Artelt et al., 2021] Artelt, A., Vaquet, V., Velioglu, R., Hinder, F., Brinkrolf, J., Schilling, M., and Hammer, B. (2021). Evaluating Robustness of Counterfactual Explanations.

[Breiman et al., 2017] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (2017). *Classification And Regression Trees*. Routledge.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, New York, NY, USA. ACM.

[Clark, 1989] Clark, P. (1989). The CN2 Induction Algorithm. Technical report.

[Clark and Boswell, 1991] Clark, P. and Boswell, R. (1991). Rule Induction with CN2: Some Recent Improvements. *Lecture Notes in Computer Science*, 482.

[Cohen, 1995] Cohen, W. W. (1995). Fast Effective Rule Induction. In *Machine Learning Proceedings 1995*, pages 115–123. Elsevier.

[Goodman and Flaxman, 2017] Goodman, B. and Flaxman, S. (2017). European union regulations on algorithmic decision making and a "right to explanation". *AI Magazine*, 38(3):50–57.

[Guyomard et al., 2023] Guyomard, V., Fessant, F., Guyet, T., Bouadi, T., and Termier, A. (2023). Generating Robust Counterfactual Explanations. volume 14171 of *Lecture Notes in Computer Science*, pages 394–409. Springer Nature Switzerland, Cham.

[Hühn and Hüllermeier, 2009] Hühn, J. and Hüllermeier, E. (2009). FURIA: an algorithm for unordered fuzzy rule induction. *Data Mining and Knowledge Discovery*, 19(3):293–319.

[J.M. van de Loo, 2013] J.M. van de Loo (2013). Retail Risk Index. *Unpublished, copy received directly from author*.

[Kelly et al.,] Kelly, M., Longjohn, R., and Nottingham, K. The UCI Machine Learning Repository.

[Lakkaraju et al., 2016] Lakkaraju, H., Bach, S. H., and Leskovec, J. (2016). Interpretable Decision Sets. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 13-17-August-2016, pages 1675–1684, New York, NY, USA. ACM.

[Lundberg and Lee, 2017] Lundberg, S. and Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions.

[Mohseni et al., 2021] Mohseni, S., Zarei, N., and Ragan, E. D. (2021). A Multidisciplinary Survey and Framework for Design and Evaluation of Explainable AI Systems. *ACM Transactions on Interactive Intelligent Systems*, 11(3-4):1–45.

[Pedregosa et al., 2011] Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python. *JMLR*, 12:2825–2830.

[Proença et al., 2022] Proença, H. M., Grünwald, P., Bäck, T., and van Leeuwen, M. (2022). Robust subgroup discovery. *Data Mining and Knowledge Discovery*, 36(5):1885–1970.

[Proença and van Leeuwen, 2020] Proença, H. M. and van Leeuwen, M. (2020). Interpretable

multiclass classification by MDL-based rule lists. *Information Sciences*, 512:1372–1393.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of Decision Trees. Technical report.

[Ribeiro et al., 2016] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why Should I Trust You?" Explaining the Predictions of Any Classifier. Technical report.

[Rissanen, 1998] Rissanen, J. (1998). *Stochastic Complexity in Statistical Inquiry*. WORLD SCIENTIFIC.

[Salzberg, 1994] Salzberg, S. L. (1994). C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. *Machine Learning*, 16(3):235–240.

[Virtanen et al., 2020] Virtanen, P., Gommers, R., Oliphant, T. E., Ingold, G.-L., and Allen, G. E. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272.

[Witten et al., 1998] Witten, I., Frank, E., and Witten, I. H. (1998). Generating Accurate Rule Sets Without Global Optimization. Technical report.

[Yang et al., 2016] Yang, H., Rudin, C., and Seltzer, M. (2016). Scalable Bayesian Rule Lists.

[Yang and van Leeuwen, 2022] Yang, L. and van Leeuwen, M. (2022). Truly Unordered Probabilistic Rule Sets for Multi-class Classification.

[Zhang and Gionis, 2020] Zhang, G. and Gionis, A. (2020). Diverse Rule Sets. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1532–1541.

Appendix A

List of included features

The retail database provided by Locatus contains 64 features in total, of which 16 were used as features in this thesis. Table A.1 lists the features that were used, as well as a number of features that were excluded for a specific reason.
Beyond the features listed in Table A.1, more features were excluded from this thesis. These include:

- The Retail Risk Index and its four categories (5)
- The RRI and its indices, binned to categories (5)
- A number of ID's that connect the Locatus database to other databases, such as the BAG [1]. (8)
- ID's corresponding to other features in this database. (4)
- Features that bin a different feature into classes. (3)

[1] *Basisregistratie Adressen en Gebouwen*, a Dutch governmental database to register addresses.

Table A.1

Feature	**Original Dutch Name**	**Description / Reason for Exclusion**
Features included in the relevant feature set		
UnitID	UNITID	ID of a venue
Year	JAAR	
Population	INW	Population of the city
Shopping Area	WINKELGEBIED	
Shopping Area Main Type	WINKELGEBIEDS-HOOFDTYPE	Broad categorisation of shopping areas
Shopping Area Type	WINKELGEBIEDS-TYPERING	More specific categorisation of shopping areas
Latitude	YCOORD	
Longitude	XCOORD	
Inner City	BINNENSTAD	Whether the store is part of the inner city
Subcentre	SUBCENTRA	Whether the store is in a dense group of stores, such as a mall
City	WOONPLAATS	
Store Chain	FORMULE	E.g. Albert Heijn
Group	GROEP	Broad categorisation of branches (e.g. Hospitality Industry)
Main Branch	HOOFDBRANCHE	Branch, broad category (e.g. Food)
Branch	BRANCHE	Branch, specific (e.g. Lunchroom)
Store Surface Area	WVO	
Footfall	PASSANTENAANTAL	Number of passersby
Visit Motive Type	BEZOEKMOTIEFTYPE	Reason for visiting
Features excluded from the relevant feature set		
Postal Code	POSTCODE	Too many values
Postal Code Numbers	PC4	Too many values
Municipality	GEMEENTE	Used to construct the dataset
Province	PROVINCIE	Too broad
Neighbourhood	WIJKBUURT	Too many values
GOAD Plan	GOADPLAN	Very similar to Shopping Area
Street	STRAAT	Too specific
House Nr	HUISNR	Too specific
House Nr Addition	HUISNRTOEV	Too Specific
Region	REGIO	Too broad
Name	NAAM	Too specific
Store Chain NW	FORMULE_NW	Similar meaning to Store Chain
Organisation	ORGANISATIE	Similar meaning to Store Chain
Source for SSA	BRONWVO	Irrelevant
Check Date	CHECK_DAT	Irrelevant
Checkouts	KASSA	Too many missing values
Footfall Segment	SEGMENT	Incomparable between different cities.
Market Segment	MARKTSEGMENT	Too many missing values
Venue Quality	PANDKWALITEIT	Too many missing values
Grade	RAPPORTCIJFER	Too many missing values

Appendix B

Rulesets Indicating Non-Deterministic Behaviour

These are three rulesets produced by an unmodified TURS on the same configuration, showing non-deterministic behaviour.

Listing B.1: Possible output where two rules are found

```
01: Population >= 270175.0;
    Store Surface Area < 125.0;
    Group != Hospitality Services;
    Main Branch != Jewellery \& Optics, Cars \& Bicycles, Craft;
    Visit Motive Type != missing;
    Then: P(1) = 0.82, coverage: 2872


02: Main Branch == Private Services;
    Then: P(1) = 0.83, coverage: 1245

If none of above,
    Then: P(1) = 0.89, coverage: 16225
```

Listing B.2: Another possible output, which is almost identical to B.1, except the population feature has chosen a different cut point.

```
01: Population >= 261550.0;
    Store Surface Area < 125.0;
    Group != Hospitality Services;
    Main Branch != Jewellery & Optics, Cars & Bicycles, Craft;
    Visit Motive Type != missing;
    Then: P(1) = 0.82, coverage: 2872

02: Main Branch == Private Services;
    Then: P(1) = 0.83, coverage: 1245

If none of above,
    Then: P(1) = 0.89, coverage: 16225
```

Listing B.3: Another possible ruleset, where the search process diverges and discovers two additional rules.

```
01: 295445.0 <= Population < 299395.0;
    Sh. Area Main Type == Spread out shopping;
```

```
    Then: P(1) = 0.7, coverage: 246

02: Population >= 261550.0;
    Store Surface Area < 125.0;
    Group != Hospitality Services;
    Main Branch != Jewellery & Optics, Cars & Bicycles, Craft;
    Visit Motive Type != missing;
    Then: P(1) = 0.82, coverage: 2872

03: Main Branch == Private Services;
    Then: P(1) = 0.83, coverage: 1245

04: 282740.0 <= Population < 288120.0;
    Sh. Area Main Type == Spread out shopping;
    Then: P(1) = 0.74, coverage: 269

If none of above,
    Then: P(1) = 0.89, coverage: 15743
```

Appendix C

Rulesets

Probability Threshold

Listing C.1: Short-Term Target

```
01: Inner City == Miscellaneous Nederland;
    Then: P(1) = 0.91, coverage: 6239

02: Store Chain == Albert Heijn;
    Then: P(1) = 0.98, coverage: 332

03: Store Chain == EAR&EYEMUSIC;
    Then: P(1) = 0.0, coverage: 5

04: Population < 261540.0;
    Then: P(1) = 0.89, coverage: 23098

If none of above,
    Then: P(1) = 0.86, coverage: 27839 (49%)
```

Listing C.2: Long-Term Target

```
01: Subcentre == Groeneweg/Laan van Nieuw Guinea;
    Then: P(1) = 0.0, coverage: 16

02: Subcentre == Nachtegaalstraat/
        Burgemeester Reigerstraat;
    Then: P(1) = 0.0, coverage: 15

03: Subcentre == Balijelaan/Rijnlaan;
    Then: P(1) = 0.0, coverage: 12

04: Visit Motive Type == missing;
    Sh. Area Main Type == Central, Miscellaneous;
    Then: P(1) = 0.0, coverage: 35

05: Subcentre == GWC Kanaleneiland (s);
    Then: P(1) = 0.0, coverage: 9

06: Population < 239425.0;
    Visit Motive Type == Comparison - XS;
    Then: P(1) = 0.2, coverage: 99

07: 25735.0 <= Population < 239425.0;
    Visit Motive Type != Convenience - L, missing;
    Subcentre != none;
    Then: P(1) = 0.21, coverage: 196

If none of above,
    Then: P(1) = 0.77, coverage: 8962 (96%)
```

Rerun Else Rule

Listing C.3: Short-Term Target

```
01: Population >= 261550.0;
    Store Chain == Independent;
    Subcentre == Hoog Catharijne (s);
    Main Branch != Food;
    Then: P(1) = 0.51, coverage: 168

02: Population >= 295445.0;
    Sh. Area Main Type ==
        Spread out shopping;
    Longitude >= 5.06;
    Inner City != Miscellaneous Nederland;
    Group != Culture & Relaxation;
    Then: P(1) = 0.65, coverage: 591

03: Store Chain == EAR&EYEMUSIC;
    Then: P(1) = 0.0, coverage: 5

04: 258690.0 <= Population < 295445.0;
    Subcentre == Miscellaneous Utrecht;
    Then: P(1) = 0.99, coverage: 717

05: Branch == Misc foodstuffs;
    Then: P(1) = 0.57, coverage: 49

06: 282740.0 <= Population < 288120.0;
    Sh. Area Main Type ==
        Spread out shopping;
    Then: P(1) = 0.73, coverage: 709

07: Store Chain == Albert Heijn;
    Then: P(1) = 0.98, coverage: 332

08: Visit Motive Type == Convenience - XL
;
    Then: P(1) = 0.76, coverage: 243

09: Subcentre == Miscellaneous Utrecht;
    Then: P(1) = 0.96, coverage: 745

10: 270175.0 <= Population < 299395.0;
    Branch == Employment Agency;
    Then: P(1) = 0.69, coverage: 207

11: Population >= 299395.0;
    Subcentre == Miscellaneous Utrecht;
    Then: P(1) = 0.97, coverage: 726

12: Store Chain == Kruidvat;
    Then: P(1) = 0.97, coverage: 272

13: Longitude < 5.13;
    Inner City == City with Inner City;
    Store Chain == Independent;
    Main Branch !=
        Jewellery & Optics, Fastservice;
    Branch != Hairdressers;
    Visit Motive Type != missing;
    Then: P(1) = 0.84, coverage: 8303

14: 273130.0 <= Population < 295445.0;
    Visit Motive Type == Comparison - XL;
    Subcentre != none;
    Then: P(1) = 0.66, coverage: 369

15: Population >= 261550.0;
    5.11 <= Longitude < 5.12;
    Store Surface Area < 100.0;
    Visit Motive Type == Comparison - XL;
    Subcentre == none;
    Main Branch != Drinks;
    Branch != Restaurant, Hairdresser;
    Then: P(1) = 0.81, coverage: 2797

16: 239425.0 <= Population < 244095.0;
    Sh. Area Main Type == Spread out shopping;
    Store Surface Area < 174.0;
    Then: P(1) = 0.72, coverage: 597

17: 270175.0 <= Population < 273130.0;
    Sh. Area Main Type == Spread out shopping;
    Subcentre == none;
    Then: P(1) = 0.82, coverage: 1393

18: 235745.0 <= Population < 270175.0;
    Main Branch == Private Services;
    Shopping Area != Miscellaneous Utrecht;
    Then: P(1) = 0.82, coverage: 1343

19: Population >= 282740.0;
    52.09 <= Latitude < 52.09;
    Subcentre == Hoog Catharijne (s);
    Then: P(1) = 0.31, coverage: 51

20: Subcentre == Miscellaneous Utrecht;
    Then: P(1) = 0.99, coverage: 475

21: Store Chain == Kruidvat;
    Then: P(1) = 0.99, coverage: 164

22: Branch == Employment Agency;
    Then: P(1) = 0.81, coverage: 750

23: Population >= 261550.0;
    Sh. Area Main Type == Central;
    Longitude < 5.11;
    Then: P(1) = 0.7, coverage: 276
```

```
24: Population >= 261550.0;
    Store Chain == Independent;
    Group == Fashion & Luxury;
    Subcentre != none;
    Then: P(1) = 0.55, coverage: 130

25: 282740.0 <= Population < 288120.0;
    Sh. Area Main Type ==
        Spread out shopping;
    Then: P(1) = 0.72, coverage: 446

26: Main Branch == Private Services;
    Branch != Brokerage, Laundromat;
    Then: P(1) = 0.8, coverage: 1620

If none of above,
    Then: P(1) = 0.89, coverage: 35125 (62%)
```

Listing C.4: Long-Term Target

```
1: Subcentre == Nachtegaalstraat
       /Burgemeester Reigerstraat;
   Then: P(1) = 0.0, coverage: 15

2: Subcentre == Balijelaan/Rijnlaan;
   Then: P(1) = 0.0, coverage: 12

3: Visit Motive Type == missing;
   Sh. Area Main Type !=
       Spread out shopping;
   Then: P(1) = 0.14, coverage: 69

4: Subcentre == GWC Kanaleneiland (s);
   Then: P(1) = 0.0, coverage: 9

5: Population < 239425.0;
   Visit Motive Type == Comparison - XS;
   Then: P(1) = 0.2, coverage: 99

6: Population < 239425.0;
   Sh. Area Type == Inner City;
   Then: P(1) = 0.32, coverage: 357

7: 282740.0 <= Population < 299395.0;
   Inner City == Inner City;
   Subcentre != none;
   Then: P(1) = 0.27, coverage: 81

8: Subcentre == MiscellaneousUtrecht;
   Then: P(1) = 0.99, coverage: 196

9: Branch == 45.203.270-Garagebedrijf;
   Then: P(1) = 0.94, coverage: 232

10: Population < 239425.0;
   Sh. Area Main Type == Ondersteunend;
   Longitude >= 5.11;
   Visit Motive Type != Convenience - M;
   Then: P(1) = 0.23, coverage: 123

11: Subcentre ==
       Groeneweg/Laan van Nieuw Guinea;
   Then: P(1) = 0.0, coverage: 17

12: Subcentre == Nachtegaalstraat
       /Burgemeester Reigerstraat;
   Then: P(1) = 0.0, coverage: 16

13: Subcentre == Balijelaan/Rijnlaan;
   Then: P(1) = 0.0, coverage: 14

14: Sh. Area Main Type == Centraal;
   Visit Motive Type == missing;
   Then: P(1) = 0.0, coverage: 21

15: Subcentre == GWC Kanaleneiland (s);
   Then: P(1) = 0.0, coverage: 9

16: Population < 239425.0;
   Visit Motive Type == Comparison - XS;
   Longitude >= 5.1;
   Then: P(1) = 0.14, coverage: 83

17: Population < 239425.0;
   Inner City == Inner City;
   Longitude < 5.12;
   Then: P(1) = 0.29, coverage: 267

18: Population < 239425.0;
   Inner City == Inner City;
   Then: P(1) = 0.34, coverage: 401

If none of above,
   Then:
   Probability of False is 0.21
   Probability of True is 0.79
   Coverage of the else rule: 8124 (87%)
```

Force Else Rule Probability

Listing C.5: Long-Term Target

```
0: Store Chain == Albert Heijn;
    Then: P(1) = 0.98, coverage: 332

1: Store Chain == Kruidvat;
    Then: P(1) = 0.97, coverage: 272

2: Subcentre == Miscellaneous De Meern;
    Then: P(1) = 1.0, coverage: 57

3: Store Chain == PLUS Slijter;
    Then: P(1) = 1.0, coverage: 54

4: Population >= 258690.0;
    Subcentre == Miscellaneous Utrecht;
    Inner City != City with Inner City;
    Then: P(1) = 0.97, coverage: 740

5: 258690.0 <= Population < 261540.0;
    Visit Motive Type == missing;
    Then: P(1) = 0.97, coverage: 698

6: Store Chain == Bakker Bart;
    Then: P(1) = 1.0, coverage: 48

7: Branch == Swimming Pool;
    Then: P(1) = 1.0, coverage: 51

8: Store Chain == Gall & Gall;
    Then: P(1) = 0.97, coverage: 167

9: Subcentre == Miscellaneous Utrecht;
    Then: P(1) = 0.96, coverage: 745

10: 288120.0 <= Population < 295445.0;
    Subcentre == Miscellaneous Utrecht;
    Then: P(1) = 0.99, coverage: 717

11: Store Chain == SNS Bank;
    Then: P(1) = 0.98, coverage: 40

12: Branch == Home Improvement;
    Then: P(1) = 0.98, coverage: 93

13: Store Chain == Specsavers;
    Then: P(1) = 1.0, coverage: 33

14: Store Chain == Bagels&Beans;
    Then: P(1) = 1.0, coverage: 32

15: Store Chain == Esprit;
    Then: P(1) = 1.0, coverage: 32

16: Store Chain == Sissy-Boy;
    Then: P(1) = 1.0, coverage: 32

17: Store Chain == Boni;
    Then: P(1) = 1.0, coverage: 31

18: Store Chain == Carpetright;
    Then: P(1) = 1.0, coverage: 29

19: Store Chain == America Toda;
    Then: P(1) = 1.0, coverage: 27

20: Store Chain == PLUS;
    Then: P(1) = 0.98, coverage: 58

21: Store Chain == Banierhuis;
    Then: P(1) = 0.98, coverage: 56

22: Store Chain == Jones&Jones;
    Then: P(1) = 1.0, coverage: 25

23: Store Chain == Emmaus;
    Then: P(1) = 0.98, coverage: 54

24: Store Chain == DirckIII;
    Then: P(1) = 1.0, coverage: 23

25: Store Chain == Fit For Free;
    Then: P(1) = 0.97, coverage: 31

26: Store Chain == Praxis;
    Then: P(1) = 1.0, coverage: 27

27: Store Chain == Scapino;
    Then: P(1) = 1.0, coverage: 22

28: Branch == Kitchens & Bathrooms;
    Then: P(1) = 0.98, coverage: 56

29: 273130.0 <= Population < 282740.0;
    Sh. Area Main Type ==
        Spread out shopping;
    Store Surface Area >= 48.0;
    Then: P(1) = 0.98, coverage: 635

30: Store Chain == Hoogvliet;
    Then: P(1) = 0.95, coverage: 22

31: Store Chain == MediaMarkt;
    Then: P(1) = 1.0, coverage: 21

32: Branch == Nail Studio;
    Then: P(1) = 1.0, coverage: 21

33: Subcentre == Miscellaneous
```

Haarzuilens;
Then: P(1) = 1.0, coverage: 21

34: Store Chain == Intersport;
Then: P(1) = 1.0, coverage: 20

35: Store Chain == Score;
Then: P(1) = 1.0, coverage: 20

36: Store Chain == Basic-Fit;
Then: P(1) = 1.0, coverage: 29

37: Store Chain == Montel;
Then: P(1) = 1.0, coverage: 19

38: Store Chain == WesternUnion;
Then: P(1) = 1.0, coverage: 19

39: Store Chain == Haan;
Then: P(1) = 1.0, coverage: 21

40: Store Chain == Schaap en Ci;
Then: P(1) = 1.0, coverage: 18

41: Store Chain == Boels;
Then: P(1) = 0.96, coverage: 24

42: Store Chain == Foot Locker;
Then: P(1) = 1.0, coverage: 17

43: Store Chain == Grapedistric;
Then: P(1) = 1.0, coverage: 17

44: Store Chain == Huis & Hypot;
Then: P(1) = 1.0, coverage: 27

45: Store Chain == Pathe;
Then: P(1) = 1.0, coverage: 19

46: Store Chain == Schoonenberg;
Then: P(1) = 1.0, coverage: 17

47: Store Chain == Vanilia;
Then: P(1) = 1.0, coverage: 17

48: Store Chain == FEBO;
Then: P(1) = 1.0, coverage: 16

49: Store Chain == Gauchos;
Then: P(1) = 1.0, coverage: 16

50: Store Chain == Lederland;
Then: P(1) = 1.0, coverage: 16

51: Store Chain == Sani-Dump;
Then: P(1) = 1.0, coverage: 16

52: Store Chain == Boni Slijter;
Then: P(1) = 1.0, coverage: 15

53: Store Chain == KILROY;
Then: P(1) = 1.0, coverage: 15

54: Store Chain == Keuken Kampi;
Then: P(1) = 1.0, coverage: 15

55: Store Chain == Pauw;
Then: P(1) = 1.0, coverage: 15

56: Store Chain == Shoeby;
Then: P(1) = 1.0, coverage: 15

57: Store Chain == Superkeukens;
Then: P(1) = 1.0, coverage: 15

58: Store Chain == Swarovski;
Then: P(1) = 1.0, coverage: 15

59: Store Chain == Amazing Orie;
Then: P(1) = 1.0, coverage: 14

60: Store Chain == De Bijenkorf;
Then: P(1) = 1.0, coverage: 14

61: Store Chain == Haco;
Then: P(1) = 1.0, coverage: 14

62: Store Chain == Le Ballon;
Then: P(1) = 1.0, coverage: 14

63: Store Chain == Run2Day;
Then: P(1) = 1.0, coverage: 14

64: Store Chain == Suitsupply;
Then: P(1) = 1.0, coverage: 15

65: Store Chain == Vlaamsch Bro;
Then: P(1) = 1.0, coverage: 14

66: Store Chain == Blokker Splg;
Then: P(1) = 1.0, coverage: 13

67: Store Chain == Fair Play;
Then: P(1) = 1.0, coverage: 13

68: Store Chain == Marlies Dekk;
Then: P(1) = 1.0, coverage: 13

69: Store Chain == Profijt Meub;
Then: P(1) = 1.0, coverage: 13

70: Store Chain == Wolky;
Then: P(1) = 1.0, coverage: 13

```
71: Store Chain == mockamore;
    Then: P(1) = 1.0, coverage: 13

72: Store Chain == CarltonHotel;
    Then: P(1) = 1.0, coverage: 16

73: Store Chain == Cavallaro Na;
    Then: P(1) = 1.0, coverage: 12

74: Store Chain == De Pizzabakk;
    Then: P(1) = 1.0, coverage: 14

75: Store Chain == Decorette;
    Then: P(1) = 1.0, coverage: 17

76: Store Chain == Dille&Kamill;
    Then: P(1) = 1.0, coverage: 12

77: Store Chain == Europcar;
    Then: P(1) = 1.0, coverage: 12

78: Store Chain == KAV;
    Then: P(1) = 1.0, coverage: 17

79: Store Chain == Nedgame;
    Then: P(1) = 1.0, coverage: 12

80: Store Chain == Scheer&Foppe;
    Then: P(1) = 1.0, coverage: 12

81: Store Chain == Vitaminstore;
    Then: P(1) = 1.0, coverage: 12

82: Store Chain == WE Store;
    Then: P(1) = 1.0, coverage: 12

83: Branch == Hospital Store;
    Then: P(1) = 1.0, coverage: 17

84: Store Chain == Brokking;
    Then: P(1) = 1.0, coverage: 11

85: Store Chain == Carglass;
    Then: P(1) = 1.0, coverage: 15

86: Store Chain == Lush;
    Then: P(1) = 1.0, coverage: 11

87: Store Chain == SportCity;
    Then: P(1) = 1.0, coverage: 12

88: Store Chain == Suitable;
    Then: P(1) = 1.0, coverage: 11

89: Store Chain == ibis;
    Then: P(1) = 1.0, coverage: 16

90: Branch == Go Kart Track;
    Then: P(1) = 1.0, coverage: 14

91: Store Chain == EAR&EYEMUSIC;
    Then: P(1) = 0.0, coverage: 5

92: Store Chain == Avis;
    Then: P(1) = 0.92, coverage: 13

93: Store Chain == Bastion;
    Then: P(1) = 1.0, coverage: 15

94: Store Chain == AH PUP;
    Then: P(1) = 0.0, coverage: 1

95: Store Chain == AKTIE SLAPEN;
    Then: P(1) = 0.0, coverage: 1

96: Population < 270175.0;
    Store Chain == Blokker;
    Then: P(1) = 0.99, coverage: 84

97: Population < 261550.0;
    Main Branch ==
        Misc. Hospitality Services;
    Then: P(1) = 0.97, coverage: 175

98: Population < 261540.0;
    Branch == Shoes;
    Then: P(1) = 0.95, coverage: 439

99: Population < 273130.0;
    Branch == Textile Super;
    Then: P(1) = 0.96, coverage: 142

100: Sh. Area Main Type ==
        Spread out shopping;
    Inner City == Miscellaneous Nederland;
    Then: P(1) = 0.97, coverage: 1644

101: Subcentre ==
        Groeneweg/Laan van Nieuw Guinea;
    Then: P(1) = 0.0, coverage: 16

If none of above,
    Then: P(1) = 0.5, coverage: 49919 (88%)
```

Listing C.6: Long-Term Target

```
1: Subcentre == Miscellaneous Utrecht;
    Then: P(1) = 0.99, coverage: 196

2: Branch == Garage;
    Then: P(1) = 0.94, coverage: 232

3: Subcentre ==
        Miscellaneous Utrecht;
    Then: P(1) = 0.91, coverage: 247

4: Shopping Area ==
        Miscellaneous De Meern;
    Then: P(1) = 0.94, coverage: 123

5: Sh. Area Main Type ==
        Spread out shopping;
    Then: P(1) = 0.9, coverage: 2199

If none of above,
    Then: P(1) = 0.5, coverage: 7081 (76%)
```