BMRS-Net: Learning BMRS Predicates as Decision Trees

Yifan Hu[*]
University College London
yifanhu@umass.edu

Abstract

This paper explores two applications of learning Boolean Monadic Recursive Scheme (BMRS) feature predicates, leveraging their analogy to binary Decision Trees. Through two case studies, the paper demonstrates how these applications can successfully fit some datasets and analyze new phonological transformations in a decision-based approach, while retaining high transparency and interpretability.

1 Introduction

Phonology has traditionally been guided by frameworks such as the Sound Pattern of English (SPE) and Optimality Theory (OT) to understanding transformations and constraint satisfaction (Chomsky and Halle, 1968; Prince and Smolensky, 2002). However, there is an increasing interest in more computationally oriented models that can handle large datasets and adapt dynamically to new linguistic context. One such model is the Boolean Monadic Recursive Scheme (BMRS), a decision-based approach that utilizes recursive functions and Boolean logic, making it particularly compatible for extensive phonological analysis (Bhaskar et al., 2020; Chandlee and Jardine, 2021).

BMRS is structured around "if-then-else" expressions, which resemble the nodes of a binary Decision Tree where each decision leads to further branches and conditions. This decision-based structure associates it closely with computational models used in data science and machine learning (Quinlan, 1986). While BMRS predicates were typically defined manually (e.g., Hua et al., 2021; Oakden, 2021; Zhu, 2023; Jardine and Oakden, 2023), recent work demonstrates that decision tree learning algorithms can classify and stratify contrastive phonological features accordingly (Chandlee, 2023), suggesting a potential for these algorithms to automate the learning of BMRS feature predicates.

This paper employs the Classification and Regression Trees (CART) algorithm as a tool for automating the generation of BMRS feature predicates (Breiman et al., 1984; Pedregosa et al., 2011; Geron, 2019). We conceptualize a type of binary decision trees, termed *BMRS-Trees*, where the root and each intermediate node utilize only one Boolean attribute. Additionally, by connecting multiple BMRS-Trees in parallel, we can output a comprehensive phonological feature matrix – this network-like structure is termed the *BMRS-Net*.

2 Preliminaries

2.1 Boolean Monadic Recursive Schemes (BMRS)

BMRS (Bhaskar et al., 2020; Chandlee and Jardine, 2021) can best be conceptualized as an **index-by-index UR-to-SF** (Underlying Representation-to-Surface Form) **transducer**. It processes each index individually, starting from index 1 and iterating rightwards. To illustrate, in then mapping from the input string $x_1x_2 \ldots x_N$ to the output string $y_1y_2 \ldots y_N$, index 1 is the first to be assessed by BMRS' feature predicate, returning a Boolean value $True$ (denoted by $\top$ in this paper, or numeric 1 in vectors and matrices) or $False$ ($\bot$ or 0) that determines the output y_1, then index 2, index 3, until N. Each output character y_i is produced based primarily on its corresponding input character x_i, and the whole input string also provides contextual information, as well as all the y_i's predecessors in the output string. Given its index-by-index nature, BMRS requires its input and output be of the same length for error-free index iteration.[1]

[*] Research conducted while the author was affiliated with University College London. The author will begin a PhD program at University of Massachusetts Amherst in September 2025. This paper is a revised version of his MA dissertation.

[1] Readers unfamiliar with BMRS transduction may refer to Appendix A for a running example after finishing 2.1.

BMRS utilizes two position functions to navigate and manipulate string indices: the **predecessor** p and **successor** s, defined for any index i in a string of length N as:

$$p(x_i) = \begin{cases} x_{i-1} & \text{if } i > 1 \\ \_ & \text{if } i = 1 \end{cases}$$

$$s(x_i) = \begin{cases} x_{i+1} & \text{if } i < N \\ \_ & \text{if } i = N \end{cases}$$

The underscore _ serves as a boundary symbol at both ends.[2]

Recursively nesting position functions allows access to any preceding and succeeding characters of any given index, indicated by superscripts, e.g.:

$$p^2(x_i) = p(p(x_i))$$

$$s^3(x_i) = s(s(s(x_i)))$$

A superscripted asterisk ($*$) indicates an arbitrary number of nestings, e.g.: [3]

$$p^*(x_i) = p(x_i), p(p(x_i)) \text{ or } p(p(p(\ldots(x_i))))$$

Σ denotes the **Symbol Set** or **Alphabet**, encompassing all characters all possible characters in both input and output strings; the modified $\Sigma_\_$ incorporates the boundary symbol _. Feature predicates for each symbol σ in Σ are defined as:

$$\sigma(x) = \begin{cases} \top & \text{if } x \vDash \sigma \\ \bot & \text{if } x \nvDash \sigma \end{cases}$$

These feature predicates assess whether the character x at the current index "satisfies" or "models" the symbol σ, returning either $\top$ or $\bot$.[4] When applied to output strings, they are subscripted with an o to differentiate their application context, e.g.:[5]

$$\sigma_o(x_i) = \begin{cases} \top & \text{if } y_i \vDash \sigma \\ \bot & \text{if } y_i \nvDash \sigma \end{cases}$$

A well-formed BMRS predicate might include:

1. **Symbolic feature predicates**, which are simple checks like $\sigma(x)$ that directly assess the "match" of a symbol at the current index;
2. **Position-embedded feature predicates**, more complex predicates like $\sigma(p(x))$, $\sigma(s^2(x))$ or $\sigma(p^*(x))$[6] that evaluate the "match" of symbols at positions relative to the current index; and
3. **Conditional logic**, which refers to construction of "if-then-else" statements upon symbolic feature predicates, position-embedding feature predicates, and $\top/\bot$, e.g., if $\sigma(x)$ then $\top$ else $\sigma(p(x))$.

2.2 Decision Tree

...is a supervised learning model is used for classification and regression tasks (Quinlan, 1986; Breiman et al., 1984). It recursively splits the dataset based on input attributes, forming a tree where each node represents a decision, and each branch corresponds to a possible outcome. The leaf nodes return the predicted output.

This paper focuses on **binary classification decision trees**, where all attributes (including the target attribute) are Boolean. The training data is typically in a table, with each row representing a data instance and each column an attribute for splitting. The last column is by convention the target attribute, which the Decision Tree aims to predict. An example table is presented in Table 1:

Attribute 1	Attribute 2	Attribute 3	Target
$\top$	$\top$	$\bot$	$\bot$
$\top$	$\top$	$\top$	$\bot$
$\top$	$\bot$	$\bot$	$\bot$
$\top$	$\bot$	$\top$	$\bot$
$\bot$	$\top$	$\bot$	$\top$
$\bot$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$
$\bot$	$\bot$	$\top$	$\top$

Table 1: Example Attribute Table

Our implementation uses the scikit-learn library (Pedregosa et al., 2011), which defaults to the Classification and Regression Tree (CART) algorithm for growing Decision Trees (Breiman et al., 1984). When applied to the dataset in Table 1, CART generates the Decision Tree shown in Figure 1:

[2] The standard implementation uses a left edge ($\rtimes$) and right edge marker ($\ltimes$) instead (Bhaskar et al., 2020; Chandlee and Jardine, 2021), but in this paper use of _ is equivalent.

[3] The asterisk notation is an ad hoc reader-friendly simplification. Precise definition will be give in Footnote 6.

[4] Σ can denote beyond mere "symbols": when Σ denotes a set of phonological features (e.g., [front], as in Section 3.2) and x denotes some segment (e.g., [i]), then saying "[i] satisfies (or $\vDash$) [front]" makes more sense.

[5] $\sigma_o(x_i)$ is a target feature predicate, see Section 3.1.

[6] Technically, $p^*(x)$ is undefined and not a formal term used in BMRS. Below, we will first provide a precise definition of the functions p^* and s^*:

$p^*(\sigma, x_i) = \text{if } \sigma(x_i) \text{ then } \top \text{ else (if } \_(x_i) \text{ then } \bot \text{ else } p^*(\sigma, p(x))\text{)}$
$s^*(\sigma, x_i) = \text{if } \sigma(x_i) \text{ then } \top \text{ else (if } \_(x_i) \text{ then } \bot \text{ else } s^*(\sigma, s(x))\text{)}$

For simplicity, we will write both functions as $\sigma(p^*(x))$ and $\sigma(s^*(x))$ in the rest of the paper.

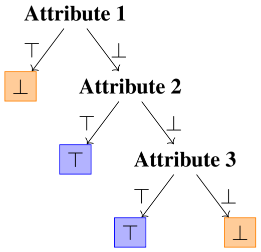


Figure 1: Example Decision Tree

While the Tree-growing algorithm is well-established (see Appendix B for a detailed explanation of CART), our focus will primarily revolve around extracting robust *attributes* (*feature predicates* in the context of BMRS, see Section 3.1).

3 BMRS-Tree

3.1 Implementation

As BMRS calculates output feature predicates index-by-index, extracting features for the CART attribute table also requires index-by-index processing. Given Σ and a UR $x_1x_2 \dots x_N$ to SF $y_1y_2 \dots y_N$ mapping, we propose that **at each index** i the following categories of feature predicates be aggregated:

Symbolic Feature Predicates: These represent whether an input character x_i matches each symbol σ in Σ, denoted as:

$$A_{symbolic} = \{\sigma(x_i) \mid \sigma \in \Sigma\}$$

Local Feature Predicates: To capture phonological dependencies from adjacent symbols, we define local feature predicates within a length of scanning window L(cf. Hua et al., 2021), with $L = 2$ by default:

$$\begin{aligned} A_{local} = &\{\sigma(p^k(x_i)) \mid \sigma \in \Sigma\_, 1 \leq k \leq L\} \\ &\cup \{\sigma(s^k(x_i)) \mid \sigma \in \Sigma\_, 1 \leq k \leq L\} \end{aligned}$$

Here, _ helps BMRS capture the absolute distance from the boundary, such as $\_(p(x_i))$ denoting whether x_1 is the first character, or $\_(s^2(x_i))$ denoting whether x_i is penultimate.[7]

Global Feature Predicates: These capture long-distance dependencies by scanning bidirectionally through the input, without precise index positioning:

$$\begin{aligned} A_{global} = &\{\sigma(p^*(x_i)) \mid \sigma \in \Sigma\} \\ &\cup \{\sigma(s^*(x_i)) \mid \sigma \in \Sigma\} \end{aligned}$$

Output-Dependent Feature Predicates: Based on Output Strictly-Local (OSL) transformations identified by (Chandlee, 2014) (see also Chandlee and Jardine, 2014; Chandlee et al., 2015, 2018), these predicates focus on the most recent output. We define two sets, local and global:

$$\begin{aligned} A_{localOutput} &= \{\sigma_o(p^k(x_i)) \mid \sigma \in \Sigma\_, 1 \leq k \leq L\} \\ A_{globalOutput} &= \{\sigma_o(p^*(x_i)) \mid \sigma \in \Sigma\} \end{aligned}$$

It's worth noting that output-dependent predicates $A_{localOutput}$ and $A_{globalOutput}$ differ from input-related A_{local} and A_{global} by including only one position function p, meaning they are restricted to left-subsequential. Unlike (Oakden, 2021), which used both left- and right-subsequential OSL functions, this paper prohibits right-subsequential OSL functions to avoid backtracking, in line with the no-backtracking mechanism of BMRS. Once an index returns an output, none of its predecessors can be reevaluated to adjust earlier outputs. Similar to A_{local} and A_{global}, $A_{localOutput}$ and $A_{globalOutput}$ embed an accurate memory of a length L scanning window and a vague memory of long-distance dependencies with respect to the output, essential for modeling phonological transformations where previous outputs influence the current index.

Target Feature Predicates: This category contains Boolean representations of the current index's output, with each feature predicate within it serving as the target attribute (the last column of an attribute table) and represented as a BMRS-Tree, denoted as:

$$A_{target} = \{\sigma_o(x_i) \mid \sigma \in \Sigma\}$$

A visual demonstration of the aggregation of feature predicates is presented in Figure 2, where each category is labeled with its number and set name, with superscripts p or s denoting left- or right-subsequential categories (cf. Oakden, 2021); application scopes are indicated by solid lines (accurate memory) or dashed lines (vague memory). All categories except A_{target} (1 to 4) constitute the set A of "Attributes" in Section 2.2:

$$A = A_{symbolic} \cup A_{local} \cup A_{global} \cup A_{localOutput} \cup A_{globalOutput}$$

[7] Strictly speaking, $\_(s^2(x_i))$ does not express "current index i being penultimate" with complete accuracy: supposing i already being final, then its successor of successor is still the boundary symbol _. Hence, in this paper, every position-embedded feature predicate with respect to _ inherently carries a second check that its predecessor/successor is not the boundary symbol _ (see below). But for simplicity, we still use $\_(s^2(x_i))$ to denote $penult(x_i)$ in the rest of the paper.

$$\text{penult}(x_i) = \text{if } \_(s^2(x_i)) \text{ then (if } \_(s(x_i)) \text{ then } \bot \text{ else } \top) \text{ else } \bot$$

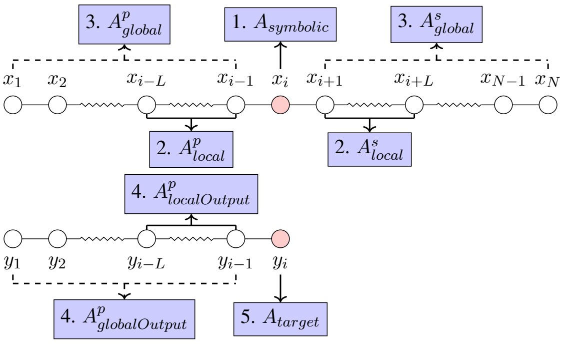


Figure 2: Aggregation of Feature Predicates at index i

The procedure of aggregating feature predicates is to arrange all the instances we extract into an attribute table, as displayed in Table 2, where the header row contains the names of each attribute $a \in A$, with each attribute a_i being an individual column. By convention the last header corresponds to one target attribute $a_{target} \in A_{target}$, the target BMRS-Tree to be learned from this attribute table.

	a_1	a_2	a_3	$\cdots$	$a_{\|A\|}$	a_{target}
Idx 1 of Data 1	$\top$	$\top$	$\bot$	$\cdots$	$\bot$	$\top$
Idx 2 of Data 1	$\top$	$\bot$	$\bot$	$\cdots$	$\top$	$\top$
$\vdots$				$\ddots$		$\vdots$
Idx N of Data 1	$\bot$	$\top$	$\top$	$\cdots$	$\bot$	$\bot$
Idx 1 of Data 2	$\top$	$\bot$	$\bot$	$\cdots$	$\bot$	$\top$
Idx 2 of Data 2	$\bot$	$\bot$	$\bot$	$\cdots$	$\top$	$\bot$
$\vdots$				$\ddots$		$\vdots$

Table 2: Example Attribute Table for BMRS-Tree

For the table content, **every row is filled in with an instance extracted from one certain index within a certain piece of data**, which represents a comprehensive snapshot of the phonological states around that index position in the string. The attribute table grows iteratively as we traverse through all possible indices across every piece of data.

Learning a target BMRS-Tree follows the same procedure as vanilla Decision Trees, using CART after arranging the attribute table. However, its evaluation differs significantly. Traditional Decision Tree evaluation focuses on cross-validation to prevent overfitting. In contrast, BMRS-Tree evaluation focuses on the purity of leaf nodes. To fit phonological data, CART minimizes Entropy in the leaf nodes (Shannon, 1948). **In the case of non-variable mappings, we propose that all leaf nodes in a well-fitted BMRS-Tree must achieve zero Entropy**, i.e. they are 100% pure. The reasons are as follows:[8]

1. BMRS-Tree learning aims to reconstruct deterministic phonological rules, rather than to generalize over unseen data (test set). In the two case studies discussed in Sections 3.2 and 4.2, all possible data are provided as the training set.[9] Thus, the training data should not be treated as samples from a larger distribution, but as a complete representation of the rule-governed system. The learning task then requires the model to fully capture and account for all observed patterns.

2. Non-variable mappings require each input to correspond to exactly one output, i.e., no free variation or probabilistic choice. If a leaf node contains multiple output classes, it introduces ambiguity, implying that a single context could trigger more than one realization. This contradicts the nature of non-variability and obstructs the derivation of a clear, well-defined rule. Zero entropy ensures that each decision path leads to a unique and unambiguous output—one that is interpretable, consistent, and faithful to the phonological data.

3. In traditional machine learning, overfitting refers to a model capturing too many exceptions or "outliers," reducing its ability to generalize. In contrast, exceptions in phonology are integral to the language and must be explicitly modeled; they are not noise to be ignored. Thus, requiring all leaf nodes to be 100% pure doesn't lead to overgeneration; rather, it helps prevent it. BMRS naturally handles exceptions through structured exception-filtering logic, represented using a series of embedded "if $exception_1(x)$ then $path_1$ else $path_2$" expressions.

In summary, the BMRS-Tree's uniqueness lies in its 100% accurate fit: its goal is to reconstruct the system rather than generalize from partial data. For interpretability, the BMRS-Tree can be validated against real phonological data, deriving rules and constraints from it (see Section 3.2 for a case study), which could be compared with already observed patterns.

3.2 Case Study 1: High Tone Shift in Kibondei

In our toy grammar, which is loosely based on the high tone shift patterns observed in Kibondei

[8] We will leave open the question of variability for the future.

[9] In Section 3.2, all training data have string lengths ranging from 1 to 8. However, we propose that the BMRS-Tree learned from the training set can also successfully generalize to strings longer than 9, due to the use of Global Feature Predicates, which are distance-insensitive.

(Merlevede, 1995; Lamont, 2024), elements can be **high-toned** (denoted by H), **low-toned** (L), or **unspecified** for tone (0). For simplicity, it is assumed that no more than one high-toned element is present in the input. The grammar operates under the following hypothetical rules:

Rule 1: L in the UR faithfully surfaces ($L \rightarrow L$; $L \nrightarrow H$, $L \nrightarrow 0$).

Rule 2: H shifts to the penultimate element if possible (e.g., $H000 \rightarrow 00H0$). It can only replace 0 and leaves the original position in 0.

Rule 3: H cannot shift across L. If an L intervenes between the H and the penultimate element, then H shifts only up to the L (e.g., $H000L000 \rightarrow 000HL000$; $H000L000 \nrightarrow 0000L0H0$).

Rule 4: H cannot surface on the final element. Underlyingly final H shifts to the penultimate position if possible (e.g., $000H \rightarrow 00H0$), and deletes if the penultimate position is occupied by an L (e.g., $00LH \rightarrow 00L0$; $00LH \nrightarrow 0HL0$).

To demonstrate the learning results of BMRS-Trees, we generated a dataset of UR-SF pairs, with each string having a length between 1 and 8, sufficient to capture potential long-distance dependencies in the high tone shift. The algorithm used to generate the dataset is provided in pseudocode in Appendix C. Ten representative data samples are presented in Table 3:

Data	UR	SF	Data	UR	SF
1	H00L	00HL	6	000H00L	00000HL
2	LH000	L00H0	7	L0000HL	L0000HL
3	000L0H	000LH0	8	00H0000L	000000HL
4	LH0L00	L0HL00	9	L0H000L0	L0000HL0
5	LH000L0	L000HL0	10	L0H0000L	L00000HL

Table 3: Data Samples of Kibondei High Tone Shift

The first step in attribute aggregation is to enumerate each symbol $\sigma \in \Sigma$: $\Sigma = \{H, L, 0\}$.

Next, by aggregating feature predicates from each index within each data (see Section 3.1), we obtain the attribute table for learning the BMRS-Tree of the target feature predicate $H_o(x)$. Running CART on this table (see Section 2.2) with scikit-learn generates the Tree diagram of $H_o(x)$, displayed in Figure 3.

The BMRS-Tree begins at the root node $L(x)$, which checks for the presence of L at the current index. If $L(x) = \top$, Rule 1 ensures that H cannot occur at the same index, returning $\bot$ for $H_o(x)$.

Continuing down, the BMRS-Tree evaluates whether the current index is valid to receive H shift: Rule 3 ensures that an L at the succeeding index ($L(s(x)) = \top$) blocks this shift, allowing the current index to be a valid alternative; Rule 2 prefers high tone shifts to the penultimate position, which is evaluated by $\_(s^2(x))$.[10]

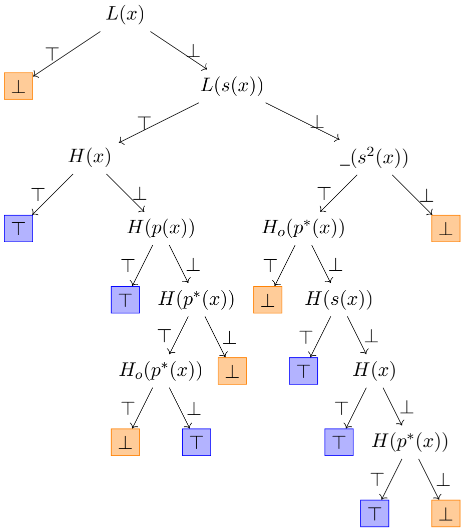


Figure 3: BMRS-Tree $H_o(x)$

Rule 3 also implicitly ascertains that the closest H can successfully shift to the current index without encountering an intervening L:

When the immediate successor is L ($L(s(x)) = \top$), then the BMRS-Tree returns $\top$ if an H exists either at the current index ($H(x) = \top$) or the immediately preceding index ($H(p(x)) = \top$). The challenge arises when locating H among all predecessors, evaluated by $H(p^*(x))$. If $H(p^*(x)) = \top$, it confirms an H at some index to the left but doesn't verify if it's blocked by an L. A common solution is to recursively test two competing elements H and L to decide which one appears earlier when looking ahead backward, using a manually-formulated function defined as:

$$\begin{aligned} Hprec(x) = &\text{ if } H(p(x)) \text{ then } \top \text{ else} \\ &\text{ if } L(p(x)) \text{ then } \bot \text{ else } Hprec(p(x)) \end{aligned}$$

This function finds the closest H or L backward from the current index, successfully indicating whether an H can shift without interruption.

By comparison, learned from real dataset, the BMRS-Tree introduces a refined method by using an output-dependent feature predicate $H_o(p^*(x))$, which checks if an H has been output among all predecessors. If $H(p^*(x)) = \top$ and $H_o(p^*(x)) = \bot$, it confirms that this H can shift to the current index, simplifying the decision-making process without recursive backtracking.

[10] See Footnote 7 for discussion.

When the current position is penultimate ($\_(s^2(x)) = \top$), the BMRS-Tree also ascertains that $H_o(p^*(x)) = \top$ for an uninterrupted H shift, validating subsequent paths: $H(s(x))$ follows Rule 4 for final H shifts, and $H(x)$ and $H(p^*(x))$ check for H at the current index or among predecessors.

In general, the BMRS-Tree intricately captures interactions of H and L in an artificial dataset by integrating output-dependent feature predicates, simplifying and optimizing the process of locating valid H shifts. This approach enhances its capability of handling wider range of phonological transformations (Oakden, 2021).

4 BMRS-Net

4.1 Implementation

Using phonological features as embeddings allows parallel processing of multiple BMRS-Trees, forming a complex network-like structure, referred to as BMRS-Net. This paper proposes a method of vectorization that treats phonological features as Boolean values (cf. Prickett, 2021).

Central to this method is the redefinition of each symbol $\sigma \in \Sigma$. Traditionally seen as mere symbols (characters) in strings, in the BMRS-Net symbols in Σ are understood as underlying components (i.e. phonological features) of each segment. Thus, Σ can be defined as:

$$\Sigma = \{[F_1], [F_2], \ldots, [F_m]\}$$

where each $[F_i]$ represents a Boolean phonological feature that returns either $\top$ (1) or $\perp$ (0); m equals $|\Sigma|$, the size of the Symbol Set, which signifies the total count of unique features.

Each segment ω from the vocabulary V (also referred to as the phoneme inventory) is then represented as an $(m + 1)$-dimensional vector. This vector is constructed by assessing each phonological feature $[F_i]$ for ω, plus an extra 0 as the final element, which represents an additional phonological feature specifically for the boundary symbol _. This boundary feature returns $\top$ only when evaluated on the boundary symbol _. Formally, a character ω in V can be represented by the vector:

$$\vec{v} = ([F_1](\omega), [F_2](\omega), \ldots, [F_m](\omega), 0)$$

We introduce the embedding matrix E, a one-to-one mapping from each segment to its vector representation, expressed as:

$$E : \omega \to \vec{v}$$

The notation $E(\omega) = \vec{v}$ denotes the vector associated with a segment ω, and its inverse function $E^{-1}(\vec{v}) = \omega$ denotes the retrieval of the original segment from its vector representation.

By definition, BMRS-Net is the parallel connection of $m + 1$ BMRS-Trees, where $m = |\Sigma|$.

The embedding matrix E facilitates transformation of phonological data into a vector format, essential for BMRS-Net processing. It computes the output vector $\vec{v}_o$ for a given input segment ω. The input vector $\vec{v}$ with respect to the input segment ω and the corresponding output vector $\vec{v}_o$ are respectively defined as:

$$\vec{v} = E(\omega)$$

$$\vec{v}_o = ([F_1]_o(\omega), [F_2]_o(\omega), \ldots, [F_m]_o(\omega), 0)$$

As can be noticed, $\vec{v}_o$ includes the same features as $\vec{v}$, with an additional zero for the boundary symbol _. Once $\vec{v}_o$ is computed, the inverse function of E is employed to retrieve the corresponding segment for further analysis or processing. The retrieved segment, denoted as ω_o, is obtained through:

$$\omega_o = E^{-1}(\vec{v}_o)$$

Figure 4 visualizes the BMRS-Net transformation of a given index i (each grey block denotes an individual target feature predicate):

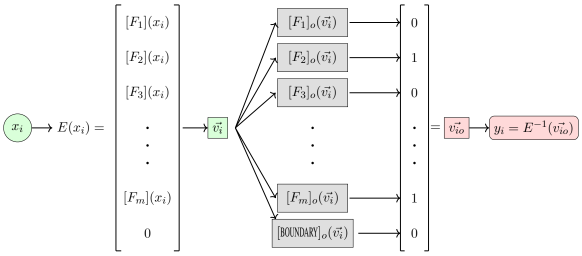


Figure 4: BMRS-Net[11]

4.2 Case Study 2: Rhotacization in Mandarin

This phenomenon refers to the transformation of a non-rhotic sound into a rhotic one, typically resembling a [ɹ]-like sound (Chao, 1968; Lu, 1995; Eckert, 2018). It generally occurs at the syllabic level, and adding the suffix -ɚ induces alternations within the rhyme.

The dataset, summarized in Table 4, draws from research by Lin (1989), Duanmu (2007), and Zhu

[11] For simplification, $[F]_o(\vec{v})$ denotes the same as $[F]_o(E^{-1}(\vec{v}))$.

(2023). This training set includes only the rhyme components (nucleus + coda) of stems plus the suffix -ə˞. Glide components in the onset parts of URs are also included if they trigger Mid Vowel Alternation (discussed later in this section); rhotacization can also alter some segments into glides in SFs.

UR	**SF**	**UR**	**SF**	**UR**	**SF**	**UR**	**SF**
i-ə˞	jə˞	u-ə˞	u˞	ə-ə˞	ɤ˞	əi-ə˞	ə˞
in-ə˞	jə˞	un-ə˞	u˞	jə-ə˞	je˞	ai-ə˞	a˞
iŋ-ə˞	jə̃˞	uŋ-ə˞	ũ˞	ɥə-ə˞	ɥe˞	əu-ə˞	ou˞
y-ə˞	ɥə˞	a-ə˞	a˞	wə-ə˞	wo˞	au-ə˞	au˞
yn-ə˞	ɥə˞	an-ə˞	a˞	ən-ə˞	ə˞		
yŋ-ə˞	ɥə̃˞	aŋ-ə˞	ã˞	əŋ-ə˞	ə̃˞	ɨ-ə˞[12]	ə˞

Table 4: Mandarin Rhotacization Dataset

Observing the dataset, we can make several generalizations, some consistent with Zhu (2023):

1. **Alveolar nasal coda [n]** does not nasalize the surrounding vowel, while **velar nasal [ŋ]** does. Both nasal codas are deleted in SFs.

2. The **segment undergoing rhotacization** in the SF varies significantly depending on the **nuclei of the stems** in URs. When the stem nucleus is:

- **High front vowels [i]/[y]** (Column 1 Table 4): [i] and [y] reduce to glides [j] and [ɥ], with [ə˞] becoming the nucleus in SF; the suffix vowel [ə˞] becomes the nucleus in the SF.

- **Back or low vowel [u]/[a]** (Column 2 Table 4): [u] and [a] remain as the nucleus and undergo rhotacization; the suffix vowel [ə˞] then deletes.

- **Mid vowel [ə]** (Column 3 Table 4): [ə] firstly undergoes Mid Vowel Alternation, summarized in Table 5, then the altered vowel becomes the nucleus and undergoes rhotacization; the suffix vowel [ə˞] deletes.

- **Diphthong** (Column 4 Table 4): The coda vowel [i] deletes, and the "real" nucleus undergoes rhotacization. The coda vowel [u] undergoes rhotacization while the preceding vowel remains unchanged or undergoes Mid Vowel Alternation (ə→ o / __ u). In both scenarios, the suffix vowel [ə˞] deletes.

- **High central vowel [ɨ]** (the last line of Column 4 Table 4): [ɨ] is assumed to undergo rhotacization but surfaces as [ə˞], with the suffix vowel [ə˞] being deleted.

Description	**Rule**
Undergoes [+front] assimilation	ə → e / {j, ɥ} __
Undergoes [+back] assimilation	ə → o / {w __, __ u}
Surfaces as [ɤ] in open syllable stems	ə → ɤ / __ $]_\sigma$
Remains unchanged with nasal coda	ə → ə / __ {n, ŋ}

Table 5: Mid Vowel Alternations

Given that some segments are deleted in the training set (Table 4), we propose inserting the symbol 0, representing a zero vector where every output feature predicate returns $\perp$, to indicate deleted elements in the output.[13] This alignment ensures that each UR-SF pair is of the same length, consistent with BMRS' index-by-index nature. The aligned training set is presented in Table 6:

UR	**SF**	**UR**	**SF**	**UR**	**SF**	**UR**	**SF**
i ə˞	j ə˞	u ə˞	u˞ 0	ə ə˞	ɤ˞ 0	ə i ə˞	ə˞ 0 0
i n ə˞	j 0 ə˞	u n ə˞	u˞ 0 0	j ə ə˞	j e˞ 0	a i ə˞	a˞ 0 0
i ŋ ə˞	j 0 ə̃˞	u ŋ ə˞	ũ˞ 0 0	ɥ ə ə˞	ɥ e˞ 0	ə u ə˞	o u˞ 0
y ə˞	ɥ ə˞	a ə˞	a˞ 0	w ə ə˞	w o˞ 0	a u ə˞	a u˞ 0
y n ə˞	ɥ 0 ə˞	a n ə˞	a˞ 0 0	ə n ə˞	ə˞ 0 0		
y ŋ ə˞	ɥ 0 ə̃˞	a ŋ ə˞	ã˞ 0 0	ə ŋ ə˞	ə̃˞ 0 0	ɨ ə˞	ə˞ 0

Table 6: Mandarin Rhotacization Data (after alignment)

For Σ, we refer to the Feature Charts from Hayes (2009) to select relevant phonological features. For vowels, we include attributes like **[high]**, **[low]**, **[front]**, **[back]**, and **[round]**. For the three glides observed ([j], [ɥ] and [w]), we use **[cons]** and **[syll]**. The **[nasal]** feature covers nasalized vowels and two nasal codas ([n] and [ŋ]), and additional features **[COR]** and **[DOR]** help distinguish them. **[rhotic]** is specifically used for rhotacized vowels.[14]

Σ contains all the features above plus the [BOUNDARY] feature; and the Embedding Matrix E is outlined using this subpart of the Feature Chart (see Appendix D).

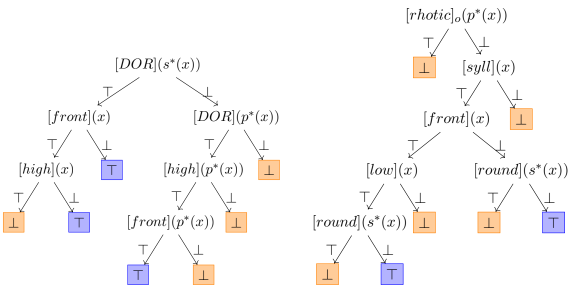
Following the established procedures from Sections 3.1 and 4.1, we can learn all the target feature predicates on Σ, provided in Appendix E. Some representative Tree diagrams will be reproduced in the following discussion for illustration:

1. **Nasal Assimilation** is controlled by $[nasal]_o(x)$ (Figure 5a), and is only triggered by a surrounding [ŋ], represented by [+DOR].

[12] Two syllabic fricatives ([z̩] and [ʐ̩]), also called apical or fricative vowels, or syllabic approximants (cf. Lee-Kim, 2014) are merged into the high central unrounded vowel [ɨ], according to (Cheng, 1973) and by convention.

[13] Similar to early OT methods where unparsed segments were considered deleted.

[14] The Hayes (2009) feature for rhotacization is [+COR, +anterior, +distributed, –strident]. Here for simplicity, we use the informal feature [rhotic].



(a) $[nasal]_o(x)$ (b) $[rhotic]_o(x)$

Figure 5

When $[DOR](s^*(x)) = \top$ ([ŋ] appears among successors), then all the vowels except [+high, +front] will be nasalized in the SF, reflected in the data: [uŋ-ɚ] → [ũ˞], [aŋ-ɚ] → [ã˞], [əŋ-ɚ] → [ə̃˞].

When $[DOR](p^*(x)) = \top$ ([ŋ] precedes the current index), the output segment will be nasalized only if there is [+high, +front] among its predecessors, reflected in: [iŋ-ɚ] → [jə̃˞], [yŋ-ɚ] → [ɥə̃˞].

2. **Rhotacization** is controlled by $[rhotic]_o(x)$ (Figure 5b), which decides whether the current segment can undergo rhotacization in the SF (i.e. to receive the [rhotic] feature).

The root node $[rhotic]_o(p^*(x))$ checks whether [+rhotic] has surfaced before the current segment. As observed from Table 4, [+rhotic] must be aligned to the final segment and surface at the final position, saying that $[rhotic]_o(p^*(x))$ actually checks whether the output string has reached the end: if it has ($[rhotic]_o(p^*(x))$ returns $\top$), then every segment from the current position will delete.

In the rest of Figure 5b, all leaf nodes returning $\top$ appear when $[round](s^*(x))$ returns $\bot$, which imposes a constraint-like condition on that a vowel cannot receive [rhotic] if it's followed by a [+round] element (in Mandarin, [u]).[15] This is also coherent to the dataset: if a vowel is succeeded by a [u], then [u] is always the one to receive [rhotic].

There are only two leaf nodes returning $\top$ (the blue nodes), which denote respectively:

- **[+front, -low]**: **[a]** (the bottom-left $\top$);
- **[-front]**: **[ɨ]**, **[ə]** and **[u]** (the bottom-right $\top$).

3. **Glide Formation** ([j], [ɥ]) in SFs offers an explanation for why [+high, +front] cannot be rhotacized. This glide formation is controlled by $[syll]_o(x)$ (Figure 6a). It also starts with $[rhotic]_o(p^*(x))$, restricting that the output string (SF) has not yet reached the end. And $[syll](x)$ asserts that [-syll] segments won't surface as [+syll]. The rest of the two intermediate nodes $[front](x)$ and $[high](x)$ denotes respectively two categories of vowels that remain [+syll] in the SF:

- **[-front]**: [ə] and [u];
- **[+front, -high]**: [a].

The bottom-left $\bot$ leaf node (in orange) denotes exactly the category that will possibly be altered to glides (or even deleted): **[+front, +high]**, consistent with the data in Column 1 Table 4).

4. **Mid Vowel Alternation** is applied to ə in the stem before deciding whether it receives [rhotic] or not. It is controlled by three predicates: $[front]_o(x)$, $[back]_o(x)$, and $[round]_o(x)$ (refer to Table 9). After being applied to the underlying ə, they are reproduced in Figures 6b, 6c and 6d.

All three BMRS-Trees start with $[syll](p^*(ə))$, checking whether a [+syll] segment (i.e. vowel) precedes ə. This presents a restriction that ə alternates only if it's the stem's nucleus or the so-called "real" nucleus of a diphthong; the ə in the suffix -ɚ or as the coda vowel doesn't alternate (though it never appears as the coda in Mandarin).

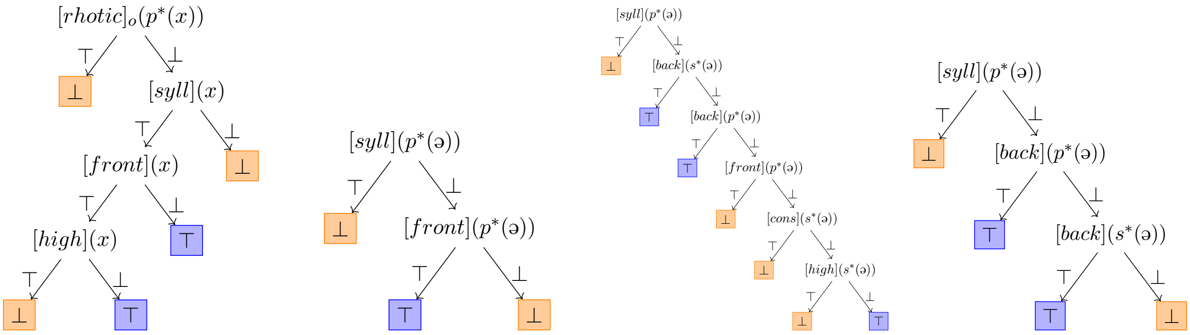
Ascertaining that ə appears as the stem's nucleus ($[syll](p^*(ə)) = \bot$, this continues as a prerequisite in the following discussion), Figure 6b then successfully models **[+front] assimilation**: a [+front] segment preceding the ə assimilates it into a front vowel [e] ($[front](p^*(ə)) = \top$).

The two upper $\top$ nodes in Figure 6c model **[+back] assimilation**: ə is [+back] assimilated when there exists a [+back] segment before or after it. Continuing down, $[front](p^*(ə))$ filters out two front glides ([j] and [ɥ]) that license the [+front] assimilation; $[cons](s^*(ə))$ filters out two cases where ə is followed by nasal codas [n] and [ŋ] – the only two consonants existent in our dataset ([ən-ɚ] → [ɚ], [əŋ-ɚ] → [ə̃˞]).

$[high](s^*(ə))$ models another possibility: **pseudo-[+back] assimilation**, when followed by a [+high] segment. This is consistent with the piece of data: [ə-ɚ] → [ɤ˞], in comparison with [əi-ɚ] → [ɚ] (the ə followed by [i] doesn't alternate). In fact, this is also consistent with Line 3 Table 5 that ə surfaces as ɤ in the open syllable stem (cf. Duanmu, 2007).

Figure 6d is almost identical to the upper part of 6c, both seeking a [+back] environment. To generalize, ə automatically receives [+round] when it re-

[15] In Mandarin, [y] never appears in complex nuclei or diphthongs.



(a) $[syll]_o(x)$

(b) $[front]_o(ə)$

(c) $[back]_o(ə)$

(d) $[round]_o(ə)$

Figure 6

ceives [+back] from its surrounding context, which is to say, **[+back] triggered by "real" [+back] assimilation innately carries [+round]**.

All the discussion above serves as an illustration of complete transparency and interpretability of BMRS-Trees learned via CART. Collectively, BMRS-Net successfully fitted the dataset and is capable of efficiently performing complex string (vector) transformations.

5 Future Research Directions

First, regarding the class of string functions, the High Tone Shift in Kibondei can be modeled by a Subsequential function (Heinz and Lai, 2013; Heinz, 2018), while the Rhotacization in Mandarin could be considered Output-Strictly local (Chandlee et al., 2015), at least in this paper, due to its dependence on the previous output to determine transformations. However, in our implementation, all categories of feature predicates (including symbolic, local, global, and output-dependent) were aggregated to form the attributes used in Decision Trees for phonological analysis (Section 3.1). Therefore, it would be beneficial to systematically analyze which categories of feature predicates are sufficient to model different string function classes.

Second, the unequal string lengths for Mandarin Rhotacization (Section 4.2) is handled a little unusually in this paper, and the use of zero vector (0) to indicate deleted segments is obviously not scalable to inserted one. Thus how to extend the current implementation to handle both deletion and epenthesis remains open for further exploration. According to a suggestion from an anonymous reviewer, the use of licensing functions and copy sets, as discussed in the work of Courcelle and Engelfriet (2012), offers a promising direction. Besides, the integration of order-preserving functions (as explored by Lindell and Chandlee, 2016) could also enable deriving both deletion and epenthesis.

If the successful categorization of feature predicates for different string function classes is achievable, and handling epenthesis becomes feasible, then our BMRS implementation could serve as a versatile tool for analyzing various classes of string functions and a broader range of phonological transformations, with enhanced flexibility and expressivity.

6 Conclusion

This paper presents the implementation of *BMRS-Trees* and *BMRS-Net* as an automated BMRS predicate learner, requiring only minimal human input, i.e., symbol (or feature) selection. Their successful application to two non-trivial (yet still limited) phonological phenomena substantiates their potential as an automation tool for researching phonological transductions, from segmental alternation, deletion to long-distance shifts (with epenthesis left for future exploration). The results offer a promising alternative to traditional rule- or constraint-based approaches, advancing the integration of machine learning in computational phonology.

Acknowledgments

This work is based on the author's MA dissertation at University College London. I am extremely grateful to Andrew Lamont for introducing BMRS to me, suggesting possible research directions and reviewing my draft, and to Sebastian Schuster for discussing and inspiring me about correlations between BMRS and Decision Trees and reviewing my draft. I also thank the anonymous reviewers for their valuable comments and suggestions, which greatly improved the quality of this paper. All remaining errors are my own.

References

Siddharth Bhaskar, Jane Chandlee, Adam Jardine, and Christopher Oakden. 2020. Boolean monadic recursive schemes as a logical characterization of the subsequential functions. In *Language and Automata Theory and Applications - LATA 2020*, Lecture Notes in Computer Science, pages 157–169. Springer.

Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. 1984. *Classification and Regression Trees*. Wadsworth International Group.

Jeroen Breteler. 2017. Deriving bounded tone with layered feet in harmonic serialism: The case of saghala. *Glossa: a journal of general linguistics*, 2(1).

Jane Chandlee. 2014. *Strictly Local Phonological Processes*. Ph.d. thesis, University of Delaware.

Jane Chandlee. 2023. Decision trees, entropy, and the contrastive feature hierarchy. *Proceedings of the Linguistic Society of America*, 8(1):5465.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2015. Output strictly local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pages 112–125, Chicago, USA. Association for Computational Linguistics.

Jane Chandlee, Jeffrey Heinz, and Adam Jardine. 2018. Input Strictly Local opaque maps. *Phonology*, 35:1–35.

Jane Chandlee and Adam Jardine. 2014. Learning phonological mappings by learning Strictly Local functions. In *Proceedings of the 2013 Meeting on Phonology (UMass Amherst)*, Proceedings of the Annual Meetings on Phonology. LSA.

Jane Chandlee and Adam Jardine. 2021. Computational universals in linguistic theory: Using recursive programs for phonological analysis. *Language*, 93:485–519.

Yuanren Chao. 1968. *A Grammar of Spoken Chinese*. University of California Press, Berkeley, California.

Chin-Chuan Cheng. 1973. *A Synchronic Phonology of Mandarin Chinese*. De Gruyter Mouton, Berlin, New York.

Noam Chomsky and Morris Halle. 1968. The sound pattern of english.

Bruno Courcelle and Joost Engelfriet. 2012. *Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach*. Cambridge University Press.

San Duanmu. 2007. *The Phonology of Standard Chinese*, 2nd edition. Oxford University Press Inc., New York.

Penelope Eckert. 2018. *Meaning and Linguistic Variation: The Third Wave in Sociolinguistics*. Cambridge University Press.

Aurelien Geron. 2019. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd edition. O'Reilly Media, Inc.

Bruce Hayes. 2009. *Introductory Phonology*. Wiley-Blackwell Publication, UK.

Jeffrey Heinz. 2018. The computational nature of phonological generalizations. In Larry Hyman and Frans Plank, editors, *Phonological Typology*, Phonetics and Phonology, chapter 5, pages 126–195. De Gruyter Mouton.

Jeffrey Heinz and Regine Lai. 2013. Vowel harmony and subsequentiality. In *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, pages 52–63, Sofia, Bulgaria.

Wenyue Hua, Huteng Dai, and Adam Jardine. 2021. Learning underlying representations and input-strictly-local functions. In *Proceedings of the 37th West Coast Conference on Formal Linguistics*, pages 143–151. Cascadilla Proceedings Project.

Adam Jardine and Christopher Oakden. 2023. Computing Process-Specific Constraints. *Linguistic Inquiry*, pages 1–9.

Andrew Lamont. 2024. Shift is derived. *Journal of Linguistics*.

Sang-Im Lee-Kim. 2014. Revisiting mandarin 'apical vowels': An articulatory and acoustic study. *Journal of the International Phonetic Association*, 44(3):261–282.

Yen-Hwei Lin. 1989. *Autosegmental Treatment of Segmental Processes in Chinese Phonology*. Phd dissertation, The University of Texas at Austin, Austin.

Steven Lindell and Jane Chandlee. 2016. A logical characterization of input strictly local functions. Presented at the Fourth Workshop on Natural Language and Computer Science, in affiliation with LICS 2016.

Yunzhong Lu. 1995. *Putonghua de Qingsheng he Erhua [Neutral Tones and Rhotacization in Mandarin]*. Shangwuyinshuguan, Beijing, China.

Andrea Merlevede. 1995. Een schets van de fonologie en morfologie van het bondei. Ma thesis, Leiden University, Leiden. Written in Dutch.

Christopher Donal Oakden. 2021. *Modeling Phonological Interactions Using Recursive Schemes*. Ph.d. dissertation, Rutgers University, School of Graduate Studies, New Jersey. Degree granted in October 2021.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

Brandon Prickett. 2021. *Learning Phonology With Sequence-To-Sequence Neural Networks*. Ph.d. dissertation, University of Massachusetts Amherst.

Alan S. Prince and Paul Smolensky. 2002. Optimality theory: Constraint interaction in generative grammar. Technical Documentation 991031549929204646, Rutgers University, Rutgers University. Essentially identical to the Tech Report (July 1993), with new pagination but the same footnote and example numbering; corrections of typos, oversights, and outright errors; improved typography; and occasional small-scale clarificatory rewordings.

J. Ross Quinlan. 1986. Induction of decision trees. *Machine Learning*, 1:81–106.

C. E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423.

Ziling Zhu. 2023. Modeling mandarin rhotacization with recursive schemes. *Proceedings of the Linguistic Society of America*, 8(1):5510.

A Modeling Tonal Shift and Spread with BMRS

Below is a brief example of BMRS transduction on a tonal system.

Saghala (Breteler, 2017) has a tone system that contrasts only high-toned elements (denoted by H) with unspecified ones (0). An underlying H shifts to the next position and then spreads one position further to the right (e.g., $00 \rightarrow 00$, $H00 \rightarrow 0HH$, $H000 \rightarrow 0HH0$, $0H00 \rightarrow 00HH$).

Assuming $\Sigma = H, 0$, we can define $H_o(x)$ to check whether the current index outputs H:

$$H_o(x) = \text{if } H(x) \text{ then } \bot \text{ else (if } H(p(x)) \text{ then } \top \text{ else } H(p^2(x)) \text{)}$$

This captures the rightward shift-and-spread behavior of H. The tree diagram in Figure 7 also visualizes this same behavior:

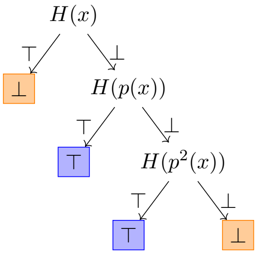


Figure 7: $H_o(x)$

Table 7 illustrates the index-by-index transductions on two input-output mappings:

	0	1	2	3	4	0	1	2	3	4
input	_	H	0	0	0	_	0	H	0	0
$H(x)$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$H(p(x))$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$
$H(p^2(x))$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$
$H_o(x)$	$\bot$	$\bot$	$\top$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$
output	_	0	H	H	0	_	0	0	H	H

Table 7: $H000 \rightarrow 0HH0$, $0H00 \rightarrow 00HH$

B Classification and Regression Tree (CART) Algorithm

CART (Breiman et al., 1984) builds binary trees. When used for classification, CART aims to split the data into subsets that are as "homogeneous" (pure) as possible with respect to the target attribute.

Entropy, borrowed from Information Theory (Shannon, 1948), is a common metric to quantify the degree of homogeneity or impurity in a dataset, and is employed as the split criterion in this paper. For a binary classification task that returns Boolean values, Entropy H of a dataset D is defined as:

$$H(D) = -p_0 \log_2(p_0) - p_1 \log_2(p_1)$$

where p_0 and p_1 refer respectively to the proportions of instances returning $\bot$ in the dataset and to that of instances returning $\top$. Entropy $H(D)$ reaches its maximum when $\top$ instances and $\bot$ instances are equally distributed (the dataset D being the most "heterogeneous" or impure) and its minimum (zero) when the dataset contains only one class (completely pure).

CART grows a Decision Tree in these steps:

1. **Calculate Initial Entropy**: The algorithm begins by calculating the Entropy of the entire dataset $H(D)$, which gives a baseline measure of impurity.

2. **Evaluate Each Attribute and Choose the Best Split**: For each attribute, CART firstly considers its split and calculates the Entropy of two resulting subsets. It then computes the **Information Gain**, which is the reduction in Entropy from the initial dataset to the combination of its two subset. The Information Gain IG from splitting dataset D on attribute A is defined as:

$$IG(D, A) = H(D) - \left(\frac{|D_0|}{|D|}H(D_0) + \frac{|D_1|}{|D|}H(D_1)\right)$$

where D_0 and D_1 denote the subsets formed by the split on attribute A, and $|D_0|$, $|D_1|$

and $|D|$ denote respectively the number of instances in the corresponding set.

The attribute that yields the largest Information Gain is selected for that split.

3. **Split the Subsets Recursively**: The process of splitting based on Information Gain continues recursively for each subset, creating decision nodes and branches, until all instances in a subset belong to the same class, or no further information gain can be achieved (because all the attributes are used up).

4. **Assign Leaf Nodes**: When no further splits are possible or necessary, the remaining data in each terminal node is assigned a label based on the majority class within that subset, forming a leaf node.

Focusing on reducing uncertainty at each step, CART constructs Decision Trees that classify the dataset as accurately as possible, while being relatively easy to interpret and to visualize using the scikit-learn library.

C Algorithm to Generate the Dataset for High Tone Shift in Kibondei

Algorithm 1 Generate Input Strings

```
Require: $min\_len$, $max\_len$
Ensure: A list of input strings consisting of $H$ (at most 1), $L$, and 0
$inputs \leftarrow [\,]$
for $length \leftarrow min\_len$ to $max\_len$ do
    $strings \leftarrow$ all combinations of $L$ and 0 of $length$
    for all $s \in strings$ do
        Append $s$ to $inputs$
        for $i \leftarrow 0$ to $length(s) - 1$ do
            $modified \leftarrow s$ with character at position $i$ replaced by $H$
            Append $modified$ to $inputs$
        end for
    end for
end for
return $inputs$
```

For reference, when $min\_len = 1$ and $max\_len = 8$, Algorithm 1 returns a list of length 4096, i.e., containing 4096 possible inputs.

Algorithm 2 Map Input to Output

```
Require: A string $input$
Ensure: The output string after applying the BMRS transduction
if $input$ ends with $0H$ then
    Replace the suffix $0H$ with $H0$
end if
if $input$ ends with $LH$ then
    Replace the suffix $LH$ with $L0$
end if
Replace every substring matching pattern H(0*)L with:
    same number of 0's as in the match, followed by $HL$
if $input$ ends with a substring matching pattern H0+ then
    Replace it with:
        one fewer 0 followed by $H0$
end if
return $input$
```

D Mandarin Feature Chart

Phonological features selected for Section 4.2 are presented in Table 8.[16]

	[high]	[low]	[front]	[back]	[round]	[cons]	[syll]	[nasal]	[COR]	[DOR]	[rhotic]	[BOUNDARY]
i	1	0	1	0	0	0	1	0	0	0	0	0
y	1	0	1	0	1	0	1	0	0	0	0	0
ɨ	1	0	0	0	0	0	1	0	0	0	0	0
u	1	0	0	1	1	0	1	0	0	0	0	0
u˞	1	0	0	1	1	0	1	0	0	0	1	0
ũ˞	1	0	0	1	1	0	1	1	0	0	1	0
e	0	0	1	0	0	0	1	0	0	0	0	0
e˞	0	0	1	0	0	0	1	0	0	0	1	0
ə	0	0	0	0	0	0	1	0	0	0	0	0
ə˞	0	0	0	0	0	0	1	0	0	0	1	0
ə̃˞	0	0	0	0	0	0	1	1	0	0	1	0
ɤ	0	0	0	1	0	0	1	0	0	0	0	0
ɤ˞	0	0	0	1	0	0	1	0	0	0	1	0
o	0	0	0	1	1	0	1	0	0	0	0	0
o˞	0	0	0	1	1	0	1	0	0	0	1	0
a	0	1	1	0	0	0	1	0	0	0	0	0
a˞	0	1	1	0	0	0	1	0	0	0	1	0
ã˞	0	1	1	0	0	0	1	1	0	0	1	0
j	1	0	1	0	0	0	0	0	0	0	0	0
ɥ	1	0	1	0	1	0	0	0	0	0	0	0
w	1	0	0	1	1	0	0	0	0	0	0	0
n	0	0	0	0	0	1	0	1	1	0	0	0
ŋ	0	0	0	0	0	1	0	1	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
_	0	0	0	0	0	0	0	0	0	0	0	1

Table 8: Embedding Matrix

E Mandarin Rhotacization BMRS-Trees

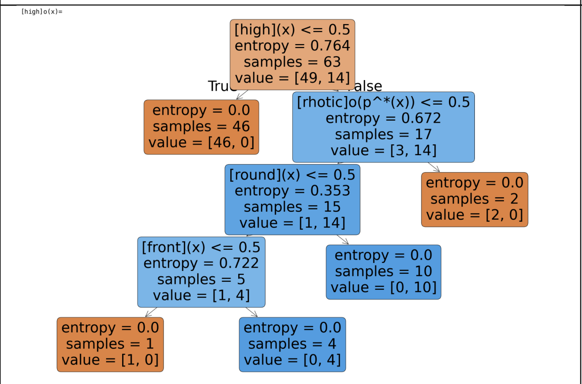
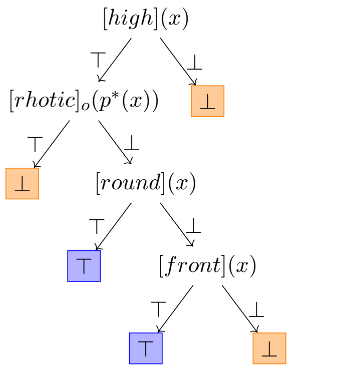
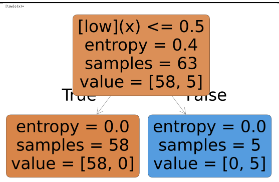
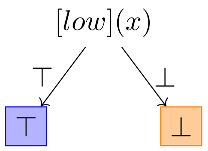
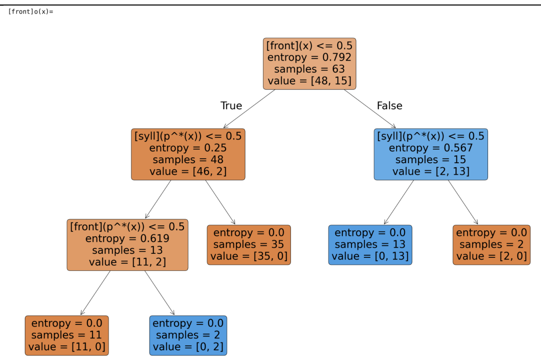
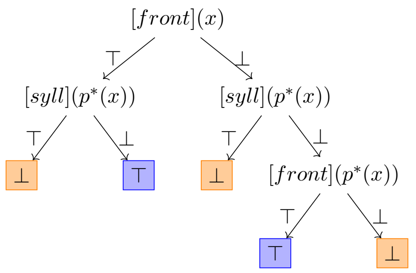
In this appendix, both original Decision Tree diagrams generated directly by scikit-learn and their simplified (remade) versions are presented in Table 9.[17]

[16] The feature [BOUNDARY] and the default boundary symbol _ are also included in E, for the sake of completeness.

[17] As can be noticed from Table 4, all [+cons] segments (i.e., [n] and [ŋ]) are deleted and don't surface in the output.

The original BMRS-Tree diagrams generated by scikit-learn can appear perplexing due to its exceedingly detailed node information, and somewhat counter-intuitive as each node tests whether a feature predicate returns “False”: the node checks whether the truth value is $<= 0.5$. Therefore, original and remade versions look like horizontal mirror images of each other. And given the low readability, a remade version is reproduced below in Table 9 and used in the main body of this paper for better visualization.

One prominently essential parameter exclusively existing here in Column “Original”, Table 9 is **Entropy**: all terminal leaf nodes’ Entropy equals zero, which is a key indication of 100% accurate fit.

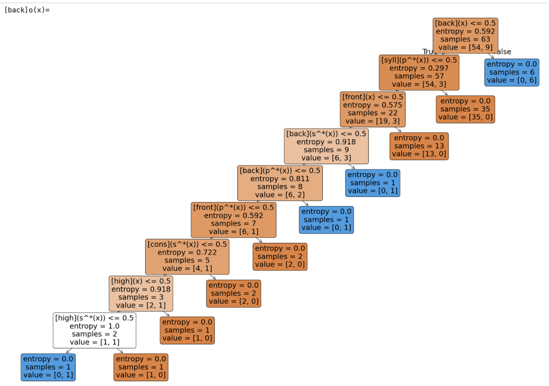
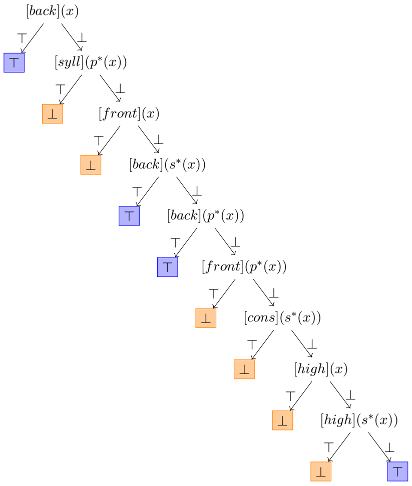
BMRS-Tree	Original	Reproduced
$[high]_o(x)$		
$[low]_o(x)$		
$[front]_o(x)$		

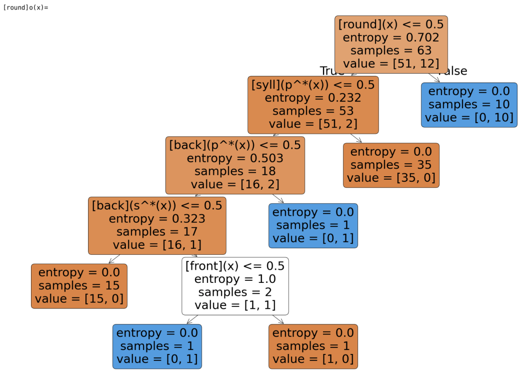
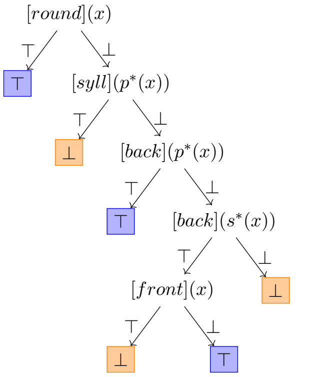
Continued on the next page

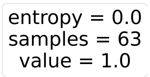
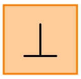
Therefore, $[cons]_o(x)$, $[COR]_o(x)$ and $[DOR]_o(x)$ always return $\perp$ – this is a side effect of only including the rhymes in the dataset. [+cons] segments will still surface in the onset.

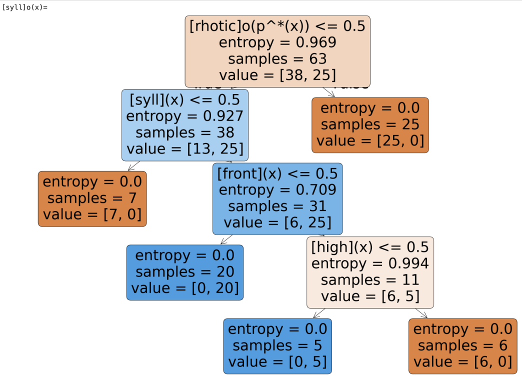
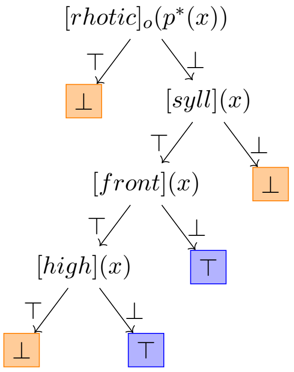
Table 9, Continued

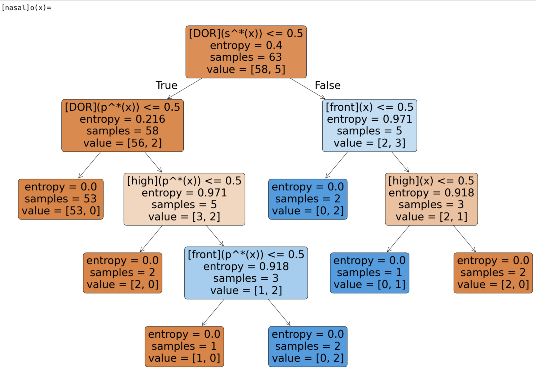
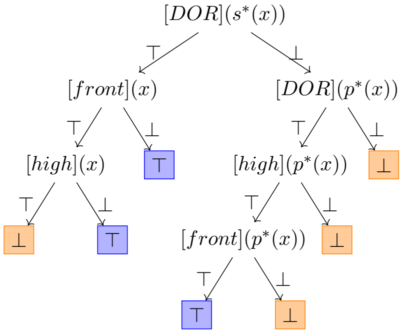
BMRS-Tree	Original	Reproduced

$[back]_o(x)$	 [back]o(x)=
	[back](x) <= 0.5
	entropy = 0.592
	samples = 63
	value = [54, 9]
	True
	False
	[syll](p^*(x)) <= 0.5
	entropy = 0.297
	samples = 57
	value = [54, 3]
	entropy = 0.0
	samples = 6
	value = [0, 6]
	[front](x) <= 0.5
	entropy = 0.575
	samples = 22
	value = [19, 3]
	entropy = 0.0
	samples = 35
	value = [35, 0]
	[back](s^*(x)) <= 0.5
	entropy = 0.918
	samples = 9
	value = [6, 3]
	entropy = 0.0
	samples = 13
	value = [13, 0]
	[back](p^*(x)) <= 0.5
	entropy = 0.811
	samples = 8
	value = [6, 2]
	entropy = 0.0
	samples = 1
	value = [0, 1]
	[front](p^*(x)) <= 0.5
	entropy = 0.592
	samples = 7
	value = [6, 1]
	entropy = 0.0
	samples = 1
	value = [0, 1]
	[cons](s^*(x)) <= 0.5
	entropy = 0.722
	samples = 5
	value = [4, 1]
	entropy = 0.0
	samples = 2
	value = [2, 0]
	[high](x) <= 0.5
	entropy = 0.918
	samples = 3
	value = [2, 1]
	entropy = 0.0
	samples = 2
	value = [2, 0]
	[high](s^*(x)) <= 0.5
	entropy = 1.0
	samples = 2
	value = [1, 1]
	entropy = 0.0
	samples = 1
	value = [1, 0]
	entropy = 0.0
	samples = 1
	value = [0, 1]
	entropy = 0.0
	samples = 1
	value = [1, 0]
		 $[back](x)$
		$\top$
		$\perp$
		$\top$
		$[syll](p^*(x))$
		$\top$
		$\perp$
		$\perp$
		$[front](x)$
		$\top$
		$\perp$
		$\perp$
		$[back](s^*(x))$
		$\top$
		$\perp$
		$\top$
		$[back](p^*(x))$
		$\top$
		$\perp$
		$\top$
		$[front](p^*(x))$
		$\top$
		$\perp$
		$\perp$
		$[cons](s^*(x))$
		$\top$
		$\perp$
		$\perp$
		$[high](x)$
		$\top$
		$\perp$
		$\perp$
		$[high](s^*(x))$
		$\top$
		$\perp$
		$\perp$
		$\top$

$[round]_o(x)$	 [round]o(x)=
	[round](x) <= 0.5
	entropy = 0.702
	samples = 63
	value = [51, 12]
	True
	False
	[syll](p^*(x)) <= 0.5
	entropy = 0.232
	samples = 53
	value = [51, 2]
	entropy = 0.0
	samples = 10
	value = [0, 10]
	[back](p^*(x)) <= 0.5
	entropy = 0.503
	samples = 18
	value = [16, 2]
	entropy = 0.0
	samples = 35
	value = [35, 0]
	[back](s^*(x)) <= 0.5
	entropy = 0.323
	samples = 17
	value = [16, 1]
	entropy = 0.0
	samples = 1
	value = [0, 1]
	entropy = 0.0
	samples = 15
	value = [15, 0]
	[front](x) <= 0.5
	entropy = 1.0
	samples = 2
	value = [1, 1]
	entropy = 0.0
	samples = 1
	value = [0, 1]
	entropy = 0.0
	samples = 1
	value = [1, 0]
		 $[round](x)$
		$\top$
		$\perp$
		$\top$
		$[syll](p^*(x))$
		$\top$
		$\perp$
		$\perp$
		$[back](p^*(x))$
		$\top$
		$\perp$
		$\top$
		$[back](s^*(x))$
		$\top$
		$\perp$
		$[front](x)$
		$\perp$
		$\top$
		$\perp$
		$\perp$
		$\top$

$[cons]_o(x)$		

$[syll]_o(x)$	 [syll]o(x)=
	[rhotic]o(p^*(x)) <= 0.5
	entropy = 0.969
	samples = 63
	value = [38, 25]
	[syll](x) <= 0.5
	entropy = 0.927
	samples = 38
	value = [13, 25]
	entropy = 0.0
	samples = 25
	value = [25, 0]
	entropy = 0.0
	samples = 7
	value = [7, 0]
	[front](x) <= 0.5
	entropy = 0.709
	samples = 31
	value = [6, 25]
	entropy = 0.0
	samples = 20
	value = [0, 20]
	[high](x) <= 0.5
	entropy = 0.994
	samples = 11
	value = [6, 5]
	entropy = 0.0
	samples = 5
	value = [0, 5]
	entropy = 0.0
	samples = 6
	value = [6, 0]
		 $[rhotic]_o(p^*(x))$
		$\top$
		$\perp$
		$\perp$
		$[syll](x)$
		$\top$
		$\perp$
		$[front](x)$
		$\perp$
		$\top$
		$\perp$
		$[high](x)$
		$\top$
		$\top$
		$\perp$
		$\perp$
		$\top$

$[nasal]_o(x)$	 [nasal]o(x)=
	[DOR](s^*(x)) <= 0.5
	entropy = 0.4
	samples = 63
	value = [58, 5]
	True
	False
	[DOR](p^*(x)) <= 0.5
	entropy = 0.216
	samples = 58
	value = [56, 2]
	[front](x) <= 0.5
	entropy = 0.971
	samples = 5
	value = [2, 3]
	entropy = 0.0
	samples = 53
	value = [53, 0]
	[high](p^*(x)) <= 0.5
	entropy = 0.971
	samples = 5
	value = [3, 2]
	entropy = 0.0
	samples = 2
	value = [0, 2]
	[high](x) <= 0.5
	entropy = 0.918
	samples = 3
	value = [2, 1]
	entropy = 0.0
	samples = 2
	value = [2, 0]
	[front](p^*(x)) <= 0.5
	entropy = 0.918
	samples = 3
	value = [1, 2]
	entropy = 0.0
	samples = 1
	value = [0, 1]
	entropy = 0.0
	samples = 2
	value = [2, 0]
	entropy = 0.0
	samples = 1
	value = [1, 0]
	entropy = 0.0
	samples = 2
	value = [0, 2]
		 $[DOR](s^*(x))$
		$\top$
		$\perp$
		$[front](x)$
		$[DOR](p^*(x))$
		$\top$
		$\perp$
		$\top$
		$\perp$
		$[high](x)$
		$\top$
		$[high](p^*(x))$
		$\perp$
		$\top$
		$\perp$
		$\top$
		$\perp$
		$\perp$
		$\top$
		$[front](p^*(x))$
		$\perp$
		$\top$
		$\perp$
		$\top$
		$\perp$

Continued on the next page

Table 9, Continued

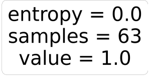
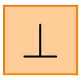
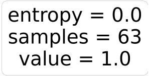
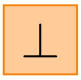
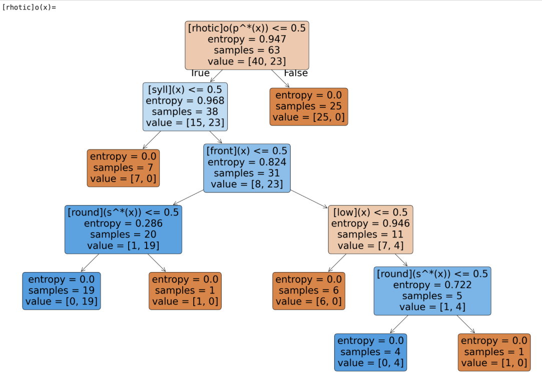
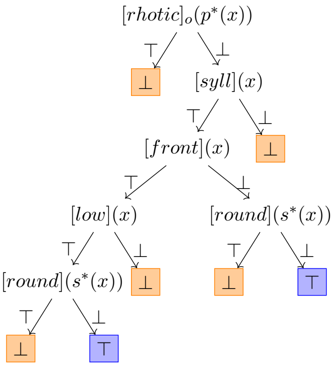
BMRS-Tree	**Original**	**Reproduced**
$[COR]_o(x)$		
$[DOR]_o(x)$		
$[rhotic]_o(x)$	 [rhotic]o(x)= [rhotic]o(p^*(x)) <= 0.5 entropy = 0.947 samples = 63 value = [40, 23] True False [syll](x) <= 0.5 entropy = 0.968 samples = 38 value = [15, 23] entropy = 0.0 samples = 25 value = [25, 0] entropy = 0.0 samples = 7 value = [7, 0] [front](x) <= 0.5 entropy = 0.824 samples = 31 value = [8, 23] [round](s^*(x)) <= 0.5 entropy = 0.286 samples = 20 value = [1, 19] [low](x) <= 0.5 entropy = 0.946 samples = 11 value = [7, 4] entropy = 0.0 samples = 19 value = [0, 19] entropy = 0.0 samples = 1 value = [1, 0] entropy = 0.0 samples = 6 value = [6, 0] [round](s^*(x)) <= 0.5 entropy = 0.722 samples = 5 value = [1, 4] entropy = 0.0 samples = 4 value = [0, 4] entropy = 0.0 samples = 1 value = [1, 0]	 $[rhotic]_o(p^*(x))$ ⊤ ⊥ ⊥ $[syll](x)$ ⊤ ⊥ $[front](x)$ ⊥ ⊤ ⊥ $[low](x)$ $[round](s^*(x))$ ⊤ ⊥ ⊤ ⊥ $[round](s^*(x))$ ⊥ ⊥ ⊤ ⊤ ⊥ ⊥ ⊤

Table 9: BMRS-Tree Diagrams in Mandarin Rhotacization