Recent Advances in Phonotactic Language Recognition Using
Binary-Decision Trees

Jiří Navrátil
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
e-mail: jiri@us.ibm.com

ABSTRACT

Binary decision trees are an effective model structure in language recognition. This paper presents several related
algorithmic steps to address data sparseness issues and computational complexity. In particular, a tree adaptation step,
a recursive bottom-up smoothing step, and two variants of the Flip-Flop approximation algorithm are introduced to
language detection and studied in the context of the NIST Language Recognition Evaluation task.

1. INTRODUCTION

Let \( A = \{a_1, \ldots, a_K\} \) denote the set of symbols representing the phonetic vocabulary of speech (covering
one or several languages), for example a multilingual phone set. Furthermore, let \( A = a_1, \ldots, a_T \), \( a \in A \),
denote a set of random variables corresponding to an utterance of length \( T \). The principle of the phonotactic
modeling relies on statistical constraints intrinsically governing such sequences, specifically it aims at estimating
the probability of a language \( L \in \{L_1, \ldots, L_M\} \) given \( A \), or equivalently the probability of \( A \) given
the hypothesized language \( L \):

\[
P(A|L) = P(a_1, \ldots, a_T|L) = \prod_{t=2}^{T} P(a_t|a_{t-1}, \ldots, a_1, L)
\]

The wide-spread use of \( N \)-grams in phonotactics involves the following approximation to (1):

\[
P(a_t|a_{t-1}, \ldots, a_1, L) \approx P(a_t|a_{t-1}, \ldots, a_{t-N+1}, L), \quad (2)
\]

i.e., a unit at time \( t \) is modeled as dependent on \( N - 1 \) units immediately preceding it. The \( N \)-gram models for \( N = 2 \)
and \( N = 3 \) are referred to as “bigrams” and “trigrams”, respectively. Naturally, the approximation accuracy grows
with the model order but is associated with an exponential increase of \( O(K^N) \) in model complexity. The latter causes
robustness problems in the estimates and hence most practical \( N \)-gram systems today restrict themselves to just bi-
and trigrams [1].

Given the modeling above, to identify a language \( L \in \{L_1, \ldots, L_M\} \) the Bayes classifier makes a hypothesis based
on the maximum-likelihood rule:

\[
L^* = \arg \max_{L} P(A|L_i)
\]

The task of detecting a hypothesized language \( L^* \) in \( A \) is typically performed using the likelihood ratio test:

\[
P(A|L) \neq L^*) \geq \theta
\]

subject to a decision threshold \( \theta \).

The binary tree (BT) language models belong to a class of
approaches aiming at reducing the model complexity via context clustering. Here, the probability of a current obser-
vation \( a_t \) (token) is conditioned on a set of token histories (a “cluster of histories”). Since the clusters may include histories
of arbitrary lengths, information from longer contexts can be modeled while the model complexity is only deter-
ned by the number of such clusters and may be chosen appropriately. Obviously, a sensible choice of the clustering
function is essential. The application of BTs for this purpose proved effective in our previous study in Language
Recognition [2], in Speaker Recognition [3], and is the central subject of this paper as well. The Section 2. describes
an efficient search algorithm for tree building proposed originally by Nádas et al. [4]. An adaptation and a smoothing
step coping with sparse samples are presented in Section 3., followed by experimental results obtained on the 2003

2. BUILDING BT LANGUAGE MODELS

Consider a sufficiently large training set \( A = \{a_1, \ldots, a_T\} \) representing the decoded speech and define the distribution
\( Y_A = \{p(a_j|A)\}_{1 \leq j \leq K} \) with the proportions of symbols \( a_j \in A \) observed in \( A \). The basic step in the BT building
process is to find two disjoint subsets \( A_1 \cup A_2 = A \) using which two descendant nodes are created. To as-

\[
H(Y_A) = -\sum_{j=1}^{K} p(a_j|A) \log_2 p(a_j|A)
\]

The data split is based on a set of predictors associated with each element of \( A \) and a binary question \( Q \) – in our case
the predictors are drawn from the history \( \{a_{t-1}, a_{t-2}, \ldots\} \) of \( a_t \). In general, \( Q \) may be composite, however, in practice,
simple expressions of the type \( ^*X \in S'' \) are used, whereby \( X \) is a selected predictor variable, say \( X = a_{t-2} \), and \( S \subset A \) is a
specific subset of the symbol (e.g. phone) vocabulary. The splitting criterion then aims at finding \( Q^* \) to maximize
the reduction in the average entropy after the split. A recursive algorithm to build the tree can be summarized in the
following steps [6, 2]:

1. Let \( n \) be the current node of the tree. Initially \( n \) is the root.
2. For each predictor variable $X_i$ ($i = 1, ..., N$) find the subset $S_{i,j}$, i.e. the question "$Q_j : X_i \in S_{i,j}$?", which minimizes the average conditional entropy of the symbol distribution $Y$ at node $n$:

$$H_i(Y) = p(Q_i)H(Y|Q_i) + p(\overline{Q}_i)H(Y|\overline{Q}_i)$$

3. Determine which of the $N$ questions derived in Step 2 leads to the lowest entropy. Let this be question $k$, i.e.,

$$k = \arg \min_{1 \leq i \leq N} H_i(Y)$$

4. The reduction in entropy at node $n$ due to question $k$ is

$$R_n(k) = H(Y) - H_k(Y)$$

If this reduction is "significant," store question $k$, create two descendant nodes, $n_1$ and $n_2$, pass the data corresponding to the conditions $X_i \in S_{i,j}$ and $X_i \notin S_{i,j}$, and repeat Steps 2-4 for each of the new nodes separately.

Note that $R_n(k)$ is deemed significant based on a preset threshold.

Minimizing the overall average entropy is intuitive, as good language models are expected on average to predict tokens $a_i$ from their context $a_{i-1}$, ..., with minimum uncertainty. It is easy to show that minimizing the prediction entropy is equivalent to maximizing the training data likelihood [3]. Another interpretation by means of mutual information is also possible. By rearranging the Eq. (5):

$$R = H(Y) - \sum_{q \in \{Q\}} \sum_{a \in A} p(a,q) \log_2 \frac{p(a,q)}{p(a)p(q)} = I(Q,Y)$$

it is clear that Step 3 maximizes the mutual information between the split distribution and the node question.

The remaining task of finding the subset $S^*$ is the main source of computational complexity. An exhaustive search involves $2^{K-1}$ entropy evaluations and is unsuitable for most practical vocabulary sizes $K$. Bahl et al. [6] described an iterative greedy search algorithm adopted in our previous work [2, 3]. In this paper we apply the "Flip-Flop" algorithm introduced in [4] and compare the performance to the greedy baseline.

2.1. Determining $S^*$ using the Flip-Flop Algorithm

The idea of the Flip-Flop (FF) algorithm [4] revolves around a fact discovered by Breiman that the $2^{K-1}-1$ possible valid $S^*$ subsets (splits) can be replaced by searching only $K-1$ selected subsets, provided only two classes (two different symbols) are to be predicted [5].

To explain the process, let the data statistics of $A$ be represented in a $K \times 2$ table whose row indices correspond to the different values of a given $X_i \in A$, and column indices to the two predicted classes, say 1 and 2. We seek the best subset (split) of rows, $S_X$. As shown in [5], the best subset is among the sequence of subsets obtained by first ordering the row indices $x$ according to an increasing value of the class-conditional probability, $p(x) = p(x,1) + p(x,2)$ (the class choice is arbitrary) and then forming a sequence of sets: starting with an empty set adding one more index $i$ at a time in this order, i.e. $\{i_1\}, \{i_1, i_2\}, ..., \{i_1, ..., i_{K-1}\}$. The latter is referred to as the Two-Optimal Sequence [4]. Summing all rows $x \in S_X$ to form a first row of a new $2 \times 2$ table, and, similarly all $x \notin S_X$ to form the second row results in a $2 \times 2$ table with a certain mutual information. Selecting the $S_X$ with the maximum in the Two-Optimal Sequence is referred to as the Two-Optimal Selection.

However, in our case $K > 2$ symbols of $A$ need to be predicted, so we have a $K \times K$ table with the probabilities $p(X_i, a_i)$ at hand.

The FF algorithm copes with this problem by iteratively employing a Twisting step, in which, first, the columns $y$ are merged (summed) according to some chosen subset $S_y$, such that $K \times K$ becomes $K \times 2$, whereby the first and the second column contain sums of all $y \in S_y$, and $y \notin S_y$, respectively. The FF algorithm has two alternative variants, as follows.

2.1.1. Variant FF1

Having applied the Twisting step, this faster version determines $S_y$ via the Two-Optimal Selection. Then, the roles of $x$ and $y$ in the original table are interchanged ("Flip"); the current $S_X$ is used to form a $2 \times K$ table, followed by a search for the best $S_y$ via the Two-Optimal Sequence and the Two-Optimal Selection. The roles are then again interchanged ("Flop") and the process is repeated iteratively.

Nadas et al. proved the convergence of this variant, however, pointed out that the criterion considered in the Two-Optimal Selection, i.e. the mutual information in the $2 \times 2$ table, is not necessarily the desired one [4].

2.1.2. Variant FF2

The Two-Optimal Selection step is replaced by a K-Optimal Selection in which the mutual information in the $K \times 2$ (or $2 \times K$) is evaluated and maximized, rather than the one in the collapsed $2 \times 2$ table as before. While there is no convergence proof for this variant, it uses a direct objective for $S_X$ and hence for the Step 3 in the tree building algorithm. The sequence of steps in the FF2 algorithm can be summarized as follows [4]:

1. Start with an initial $S_X$
2. Sum rows of the $K \times K$ table using $S_X$ to form a $2 \times K$ table
3. Form sequence of candidate column splits via the Two-Optimal Sequence
4. Choose $S_Y$ via the K-Optimal Selection
5. Sum columns of the $K \times K$ table using $S_Y$ to form a $K \times 2$ table
6. Form sequence of candidate row splits via the Two-Optimal Sequence
7. Choose $S_X$ via the K-Optimal Selection
8. Go to Step 2, or quit

We used the relative change in the $2 \times K$ mutual information as a termination criterion.

The optimum solution $S_X$ is used as the $S^*$ in the Eq. (3) of the recursive tree building algorithm (Step 3).

3. HANDLING DATA SPARSENESS

Robustness issues and over-training present undoubtedly a challenge. In the previous work [2] a leaf minimum occupancy constraint was applied to prevent underpopulated leaves. This constraint causes the BT models to grow adaptively to the data set size, which, however, may become a
problem with small training data amounts resulting in a re-
atively few leaf nodes and hence too coarse models. In the
case of speaker verification [3], two mitigating techniques
were successfully applied, namely a tree adaptation and a
recursive bottom-up smoothing, bringing about consider-
able improvements in robustness. We now briefly outline
these two steps and apply them to language recognition.

3.1. Leaf Adaptation
In case of limited training data for a set of languages, a
language-independent (LI) BT model can be built from
pooled data to provide a robust tree structure as a basis
to create the language-specific BT model by adaptation. In
order to do so, the training set is partitioned according to
the fixed LI structure first, followed by an update of the leaf
distributions. Let $Y_l = \{P_l(\alpha_j)\}_{\alpha_j \in A}$ denote a symbol dis-
tribution at a leaf $l$ of the LI model, $\#(\alpha_j, l)$ the language-
specific count of $\alpha_j$ observations at leaf $l$, and $||l||$ the overall
count of the language-specific data in leaf $l$. The updated leaf
distribution $Y_l = \{\hat{P}_l(\alpha_j)\}_{\alpha_j \in A}$ is then calculated as a
linear interpolation

$$\hat{P}_l(\alpha_j) = \left[ b_j \frac{\#(\alpha_j, l)}{||l||} \left( 1 - b_j \right) \right] / D \quad (6)$$

with

$$b_j = \frac{\#(\alpha_j, l)}{\#(\alpha_j) + r} \quad (7)$$

where $D$ normalizes the adapted values to satisfy
$\sum_l \hat{P}_l(\alpha_j) = 1$, and $r$ is an empirical value controlling
the extent of the update. The adapted BT model retains
the context resolution of the SI model, while capturing the
language-specific leaf statistics.

3.2. Bottom-Up Recursive Smoothing
Despite sufficient token counts in a leaf overall, individual
symbols may still occur sparsely in some leaves. The hier-
archical BT framework offers a convenient way to identify
more robust estimates for smoothing, namely by backing-off
to the parent distribution of a leaf. Each parent distribu-
tion is a pool of both child distributions and therefore is
more likely to contain more observations of a given symbol.
The following recursive smoothing algorithm for calculat-
ing the probability of a symbol $a_i = \alpha_j$ given its context
$\{a_{i-1}, a_{i-2}, \ldots\}$ proved successful [3]:

1. Determine the leaf $l$ using $X$. Set a node variable $n = l$.
2. Calculate symbol probability

$$\hat{P}_\text{smooth}(\alpha_j) = b_j \hat{P}_n(\alpha_j) + (1 - b_j) \hat{P}_\text{par}(n)(\alpha_j)$$

where $b_j$ is as in (7) and $\hat{P}_\text{par}(n)(\alpha_j)$ is obtained by
repeating Step 2 with $n := \text{par}(n)$ recursively until
$n = \text{root}$.

Again, a linear interpolation scheme is used, whereby $\text{par}(n)$ denotes the parent node of $n$, and $r$ is defined by
(7).

4. EXPERIMENTAL RESULTS
The binary-tree (BT) system was evaluated as a com-
ponent within the combined MIT Lincoln Laboratory (MIT-
LL) language detection system developed for the 2005 NIST
Language Recognition Evaluation (LRE) [7], described in

<table>
<thead>
<tr>
<th>Configuration</th>
<th>% EER</th>
<th>Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>6dec, BT, S0/A0</td>
<td>11.7</td>
<td>D1</td>
</tr>
<tr>
<td>6dec, BT, S1/A0</td>
<td>8.6</td>
<td>D1</td>
</tr>
<tr>
<td>6dec, BT, S0/A1</td>
<td>7.4</td>
<td>D1</td>
</tr>
<tr>
<td>6dec, BT, S1/A1</td>
<td>7.4</td>
<td>D1</td>
</tr>
<tr>
<td>6dec, BT, S1/A1, BE</td>
<td>5.8</td>
<td>D1</td>
</tr>
<tr>
<td>6dec, BT, S1/A1</td>
<td>4.0</td>
<td>D2</td>
</tr>
<tr>
<td>12dec, BT, S1/A1</td>
<td>2.9</td>
<td>D2</td>
</tr>
<tr>
<td>12dec, BT, S1/A1, BE</td>
<td>2.1</td>
<td>D2</td>
</tr>
</tbody>
</table>

Table 1. Equal-Error Rate performance overview. S0/1=Smoothing off/on, A0/1=Adaptation off/on, BE=Back-end normalization

the related paper [1]. This section details on the various
BT configurations, while using data consistent with [1].

4.1. Development Corpus
Two development data sets were used:

- D1: training using the NIST LRE 1996 set and testing
  on the NIST LRE 2003 evaluation comprising of 12
  languages (CallFriend corpus). The 30-sec utterances
  were taken for both the training and the test.
- D2: extended training and evaluation data drawn from
  the CallFriend, the Fisher, and the Mixer corpora.
The training dataset is referred to as
train.xcorpus.balanced and the testing set as
test_13lang in [1] and comprises of 13 languages.

4.2. NIST LRE 2005 Data
The primary LRE05 task was the detection of a hypoth-
esized language in 30-sec long telephone utterances [7].
The speech material for this primary task was drawn from
the OHSU corpus with 7 a-priori known target languages.
Other conditions involving 13 languages, 3-sec, and 10-sec
utterances were also defined [7].

4.3. Phonetic Decoders
Decoding speech utterances into token sequences was per-
formed in three configurations using:

- 1. dec: a single phone decoder with acoustic models
  trained in the framework of the IBM large vocabulary
  speech recognition

4.4. Language Detection Performance
The detection task was performed by means of the log-
likelihood ratio test as outlined by Eq. (4) and as described
in detail in [1].

The various combinations of BT adaptation and smooth-
ing in terms of their Equal-Error Rates (EER) are summa-
ized in Table 1.

The trends allow us to conclude that both the adapta-
tion and the smoothing steps are highly beneficial noting
that smoothing helps in non-adapted trees, while adapted
trees outperform the smoothing in isolation. The use of the
nonlinear back-end classifier (as described [1], Section 2.7) is
shown to fuse the language hypotheses more effectively than the uniform averaging. Furthermore, as may be expected, increasing the amount of training data (switching to \( D_2 \), although the test sets are different) as well as increasing the number of decoders give a gain in accuracy. The latter indicates that errors made during phonetic decoding are likely a major adverse factor in phonotactic language recognition, and can be mitigated by fusion.

The comparative results for the BT component with adaptation and smoothing along with its comparable baseline (smoothed trigrams) of the MIT-LL system obtained on the NIST LRE 2005 primary task are shown as a DET plot in the Figure 1 (from \[1\]).

Table 2 shows EER and computational expense results obtained using the Flip-Flop (FF) algorithms. Note that, in this experiment, the 1dec decoder was used. The computational complexity is measured as the average number of entropy evaluations made during the search for the optimum node question per predictor. While the \( FF_2 \) variant reduced the complexity by a half compared to the baseline search, the fast \( FF_1 \) variant required considerably less computation due to the fact that such entropy evaluation is performed for a \( 2 \times 2 \) table, as opposed to \( K \times 2 \). Across the various conditions, the performance seems roughly comparable for all three search algorithms. Although for \( K \) ranging between 30 and 40 with the phonetic decoders used, the tree building took on the order of seconds to complete using standard hardware, for tasks with larger search space, i.e., larger \( K \), the Flip-Flop algorithms may therefore present a very attractive choice to reduce the tree building time.

### 5. CONCLUSIONS

As measured on the NIST LRE datasets 2005 the BT's perform very well, favorably comparing to N-grams and other approaches to language recognition (see Figure 1). It should be pointed out that the BT approach is not viewed as a competing replacement of the standard N-grams but rather its effective counterpart in fusion. For a detailed analysis of fusion experiments including the presented BT component, the reader is referred to \[1\].

<table>
<thead>
<tr>
<th>% EER</th>
<th>Greedy</th>
<th>FF1</th>
<th>FF2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Pred.</td>
<td>6.5</td>
<td>7.1</td>
<td>6.7</td>
</tr>
<tr>
<td>3 Pred.</td>
<td>7.1</td>
<td>6.8</td>
<td>6.9</td>
</tr>
<tr>
<td>4 Pred.</td>
<td>7.2</td>
<td>7.3</td>
<td>8.0</td>
</tr>
<tr>
<td>5 Pred.</td>
<td>6.7</td>
<td>7.5</td>
<td>8.2</td>
</tr>
<tr>
<td>Avg. C</td>
<td>5%</td>
<td>5%</td>
<td>7%</td>
</tr>
</tbody>
</table>

Table 2. EER performance of the 1dec and the FF algorithms with varying number of predictors on the 30-sec 2003 NIST LRE task (D1 set). Avg. C stands for average number of entropy computations per predictor and node during the subset search.

Both the adaptation and the recursive smoothing were shown to be essential performance factors in the detection task, addressing the data sparseness and bringing about 40% reduction in the EER, relative to a baseline configuration of \[2\]. This confirms the findings made in the speaker verification task \[3\].

Both variants of the Flip-Flop algorithm resulted in an accuracy comparable to the baseline, however, brought a considerable reduction in computation, effectively speeding up the tree building process by an order of magnitude, thus offering an attractive alternative in larger training tasks.

### 6. ACKNOWLEDGMENTS

The author wishes to thank the MIT Lincoln Lab team for providing the decoded phone sequences used in this study, and for their openness to collaboration during the NIST LRE05 efforts. The author is thankful to David Nahamoo for insightful discussions on the decision trees and the Flip-Flop search algorithms.

### REFERENCES


