UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Matematica “Tullio-Levi Civita”

UNIVERSITÉ PARIS-DAUPHINE PSL CEREMADE

Département de Mathématiques et Applications

Master Degree Course in Mathematics

Double Degree Course “MAPPA”

Project coordinator, Università di Padova: Francesco Rossi

GRAMMAR INFERENCE THROUGH HIDDEN MARKOV MODELS AND PROBABILISTIC CONTEXT-FREE GRAMMARS

Supervisor:
Prof. Robin Ryder

Co-supervisor:
Prof. Marco Formentin

Student:
Eva Shi

Student ID:
2020278 / 22100610

Academic year 2021/2022

Contents

Chapter 1

Introduction

Following the hierarchy proposed by Chomsky, grammars can be classified into several classes: moving from the simplest to the most complex we have: *regular, context-free, context-sensitive* and *recursively enumerable.* In this thesis we are interested in inferring the properties of an unknown grammar from a set of strings produced by that grammar. To do this we will use several models, first of all the Baum-Welch algorithm, an algorithm for finding the parameters of Hidden Markov Models. This algorithm is useful for us since we will see that because of their definition, regular grammars can be represented as Hidden Markov Models. We will use this algorithm together with the Bayesian Information Criterion [4],[5], where we choose the best fitted model based on a penalized likelihood probability. The other tool we will use to try to analyze grammars is a probabilistic context-free grammar for the construction of hypotheses, in particular we will exploit the C++ Fleet library developed by Yang and Piantadosi [1]. This library searches for the best program that generates the data it receives as input by combining primitive operations. This search is done by sampling through Markov Chain Monte Carlo method, in particular through a parallel tempering method, where the search is done using multiple chains at different temperatures. We find that, while these methods are not always very accurate, they are a good starting point for guessing what class the grammars we are interested in belong to.

In the following chapter we will introduce some preliminary notions of grammar, in particular the definition of formal grammar and probabilistic grammar, as well as the Chomsky hierarchy more in detail.

In Chapter 3 we will give the definition of Hidden Markov Models, which are useful to study regular grammars. We will also introduce the Baum-Welch algorithm and the Pumping lemma.

In Chapter 4 we will describe the library Fleet, explaining more in detail the model and the parallel tempering search. There will be also a short section explaining how to use the library and how to read the output.

In Chapter 5 we will finally infer grammars using the tools we have introduced in the previous chapters (Baum-Welch algorithm, Pumping lemma and Fleet). We start by studying simulated data, in particular, data generated by the grammar $a^n b^n$ and a simplified version of English grammar. We will obtain good result for the former, being a simple grammar, but not as good result for the latter being more complex. In the following we will also study some real data: a dataset of sentences from Harry Potter's book and a dataset of Campbell monkey calls.

Chapter 2

Preliminary notions of grammar

Before digging into more technical details, let us give a brief introduction about formal grammar and then probabilistic grammar, following the definitions given in [2] and [3].

2.1 Formal grammar

Most natural language processing systems are based on formal grammars which we will soon define. In the following we will introduce also probabilistic grammars, a natural extension of the first which allows us to use some mathematical tools like Hidden Markov Models and Bayesian statistics, therefore it will be the one we will work with.

Notation. First of all, let us introduce some notations: consider two sets X and Y, that could be set of symbols, characters or words. We will write:

- X^* to indicate the free monoid on X, i.e. the set of strings over X : an element of X^* is obtained by concatenating a finite number of elements of X, * is called the Kleene star;
- XY to denote the set of words obtained by concatenating an element of X and an element of Y.

For instance, if $X = \{a, b\}$ and $Y = \{c, d, e\}$ then $X^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, ...\}$ and $XY = \{ac, ad, ae, bc, bd.be\}$, with ϵ the empty string.

Definition 1. *A **formal grammar** is defined as $\mathcal{G} = (\mathcal{A}, \mathcal{B}, S, \mathcal{R})$, containing*

- *a finite set of terminal symbols $\mathcal{A} = \{a_1, ..., a_K\}$*
- *a finite set of nonterminal symbols $\mathcal{B} = \{B_0, ..., B_J\}$*
- *a distinguished non terminal symbol B_0, the start symbol, often denoted by S in the literature*
- *a finite set of rules $\mathcal{R}$, each of the form $\alpha \to \beta$ where $\alpha, \beta \in (\mathcal{A} \cup \mathcal{B})^*$.*

We can think of the set $\mathcal{A}$ as the set of all the words in a dictionary of the language and the set $\mathcal{B}$ as syntactic latent variables.

To produce a sentence with the grammar $\mathcal{G}$, one starts with the nonterminal S and applies the rules in $\mathcal{R}$ until there are no more nonterminal symbols left, which means there are no more rule that could be applied. It is possible to produce both finite and non finite sentences, depending on the rules that characterize the grammar. Let us show an example.
Example. Consider the grammar $\mathcal{G} = (\mathcal{A}, \mathcal{B}, S, \mathcal{R})$ by:

- $\mathcal{B} = \{B_0, B_1, B_2\}$
- $\mathcal{A} = \{a_1 = \text{“the dog”}, a_2 = \text{“the cat”}, a_3 = \text{“runs”}, a_4 = \text{“fast”}\}$

and the following rules

- $r_{0,1} : B_0 \to a_1 B_1 B_2$
- $r_{0,2} : B_0 \to a_2 B_1$
- $r_{1,1} : B_1 \to a_3$
- $r_{2,1} : B_2 \to a_4$

The grammar produces the following sentences:

1. $a_1 a_3 a_4$= “the dog runs fast” by applying rules $r_{0,1}, r_{1,1}, r_{2,1}$;
2. $a_2 a_3$ =“the cat runs” by applying rule $r_{0,2}, r_{1,1}$.

We can then classify grammars into different classes, depending on the form of the rules: the more complex the grammar, the more types of rule it contains. In this classification we will refer to the Chomsky hierarchy, represented in Figure 2.1. This containment hierarchy includes four different classes of grammar, the simplest ones are regular grammars, followed by context-free grammars. We have then context-sensitive grammars and finally recursively enumerable grammars, the most generic class of grammar, where there are no constraints on the form of the rules.

Definition 2. *Given a formal grammar $\mathcal{G} = (\mathcal{A}, \mathcal{B}, \mathcal{S}, \mathcal{R})$, we call it (right) regular if all the rules take one of the following form:*
-$B_i \to a_j B_k$
-$B_i \to a_j$
i.e. $\forall(\alpha \to \beta) \in \mathcal{R}, \alpha \in \mathcal{B}, \beta \in \mathcal{A} \cup (\mathcal{AB})$.

Definition 3. *A context-free grammar is one in which all the rules are of the form $B_i \to \beta$, i.e. $\forall(\alpha \to \beta) \in \mathcal{R}, \alpha \in \mathcal{B}$ and β string of terminal and nonterminal symbols, it could also be empty.*

Definition 4. *A context-sensitive grammar is one in which all rules are of the form $\alpha B_i \beta \to \alpha\gamma\beta$ with $\alpha, \beta, \gamma \in (\mathcal{A} \cup \mathcal{B})^*$.*

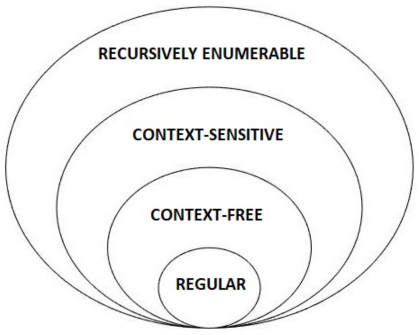


Figure 2.1: Chomsky hierarchy

Example of regular grammar. A simple example of regular grammar could is a^n, more explicitly with the following rules

– $r_{0,1} : B_0 \to aB_0$

– $r_{0,2} : B_0 \to a$

Example of context-free grammar. An example of context-free grammar is one that generates palindromic sentences, in its easiest version:

– $r_{0,1} : B_0 \to aB_0a$

– $r_{0,1} : B_0 \to bB_0b$

– $r_{0,2} : B_0 \to a$

– $r_{0,2} : B_0 \to b$

Example of context-sensitive grammar. An example of context-sensitive grammar is $a^nb^nc^n$ generated by the rules

- $r_1 : B_0 \to aB_1B_2$	- $r_6 : aB_1 \to ab$
- $r_2 : B_0 \to aB_0B_1B_2$	- $r_7 : bB_1 \to bb$
- $r_3 : B_2B_1 \to B_2T_1$	- $r_8 : bB_2 \to bc$
- $r_4 : B_2T_1 \to B_1T_1$	- $r_9 : cB_2 \to cc$
- $r_5 : B_1T_1 \to B_1B_2$	

The rules from r_3 to r_5 are needed to obtain $B_2B_1 \to B_1B_2$ which does not respect the form $\alpha B_i \beta \to \alpha\gamma\beta$.

2.2 Probabilistic grammars

In order to study grammars using statistical and probabilistic tools, we can define probabilistic regular and context-free grammars. This can be done by simply assign probability distributions to all the possible rules that can be applied to a nonterminal symbol. More formally,

Definition 5. *A **probabilistic grammar** is defined as $\mathcal{G} = (\mathcal{A}, \mathcal{B}, B_0, \mathcal{P}, \mathcal{R})$ where $(\mathcal{A}, \mathcal{B}, B_0, \mathcal{R})$ is a formal grammar and $\mathcal{P} = (P_0, ..., P_J)$, with $J+1 = |\mathcal{B}|$, where each P_j for $j = 0, ..., J$ is a probability distribution on $\mathcal{R}_j$, with $\mathcal{R}_j$ the set of rules which can be applied to the nonterminal B_j, i.e. $\mathcal{R}_j = \mathcal{R} \cap \{(B_j \to \beta) : \beta \in \mathcal{AB}^*\}$ (so that $\sum_{r \in R_j} P(r) = 1$).*

2.3 Data representation and inferring

We will study both known grammar from simulated data and unknown grammar with real data. For some languages, when working with real data, it would be too difficult to study the given dataset as it is, with every sentence containing a large number of different elements. Instead we will use a simplified version of the data, where each word is represented by its speech tag, in this way the set of terminal symbols is finite, that could be a set of characters or numbers for example.

In the following chapters we will see some methods to study grammars. At first we will focus on regular grammar, which as we will see, can be represented as Hidden Markov Models. Knowing this we will use methods to infer Hidden Markov Models, in particular using the

Baum-Welch algorithm. Later on we study grammars more in general, trying to infer the rules that generates it through a probabilistic context-free grammar for rule generation. We will then classify the resulting rule, and when possible we will compare it with the result from the Baum-Welch algorithm.

Chapter 3

Hidden Markov Models and Baum-Welch Algorithm

3.1 Hidden Markov Models

To study the properties of a regular grammar we can represent it as a Hidden Markov Model (HMM). A HMM is one of the main statistical tool to model discrete time series. It can be used in many application, for example speech recognition, computational biology, computer vision and econometrics. Let us start with its definition in the discrete version.

Definition 6. *A **Hidden Markov Model** is specified by the following components:*

- *a set of N **hidden states** $\mathcal{B} = \{B_1, B_2, ..., B_N\}$;*
- *a Markov Chain X_n taking values in $\mathcal{B}$, whose transition is not direclty observable, with **transition probability matrix** $P = (p_{ij})_{i,j \in \{1,..,N\}}$, $p_{ij} \in [0,1] \;\; \forall i, j$ known;*
- *a sequence of T **observation** $O_1 O_2 ... O_T$, each one drawn from a vocabulary of **emission symbols** $\mathcal{A} = \{a_1, ..., a_M\}$;*
- *an **emission probability matrix** $E = (e_i(a_j))_{ij}$, each component expressing the probability of observing a_j being generated from a state i, $e_i(a_i) \in [0,1] \;\; \forall i, j$;*
- *an **initial state distribution** (i.e. when t=1) $\pi_i = P(X_1 = B_i)$.*

In the formal grammar framework, hidden states correspond to nonterminal symbols, while the emission symbols are terminal symbols. We will always work with finite set of hidden states and emission symbols.

Without loss of generality the sets of hidden states and emission symbols can be considered as sets of integers of the same cardinality, we can therefore describe a hidden Markov chain only by the triplet $\theta = (\mathcal{A}, E, \pi)$. Given observed data we aim to infer the parameters that characterize the transition and emission matrices of a specific HMM. To do so we use the Baum-Welch algorithm.

3.2 Pumping Lemma for regular language

As already mentioned, a regular grammar can be represented as a HMM where non terminals are hidden states and terminals are emission symbols. With this representation, it is easy to prove the Pumping lemma , which is useful in the classification of grammars, we refer to [7]. The Pumping lemma is usually used to prove that a grammar is not regular.

Lemma 3.2.1. *Let $\mathcal{G}$ be a regular grammar. Then there exists an integer $p \geq 1$ depending only on L such that every string w in L of length at least p (p is called the "pumping length") can be written as $w = xyz$ (i.e., w can be divided into three substrings), satisfying the following conditions:*

- $|y| \geq 1$
- $|xy| \leq p$
- $(\forall n \geq 0)(xy^n z \in \mathcal{L})$

y is the substring that can be pumped (removed or repeated any number of times, and the resulting string is always in L).

Proof. Remember that a regular language is characterised by a HMM, which has finite number of hidden states. Let p be the number of hidden states, then for any string of length greater than p, there must be a hidden state that has been visited more than once, let's call this state A, the substring created from the first encounter of S to the second encounter of S is what is defined as y in the statement of the lemma. It is clear that the sequence of transitions generating y can be repeated any number of times by the HMM or not appearing at all, so the conditions of the lemma are satisfied. □

3.3 Baum-Welch algorithm

The Baum–Welch algorithm [5] is a powerful tool to learn the probabilities of a HMM given a set of observed feature vectors, using the idea of the expectation- maximization algorithm to find the maximum likelihood estimates of said parameters, by estimating the expected value of the likelihood function and then find the value of the parameters that maximize it.

The algorithm is divided into different steps.

1. **Initial step.** In this first step, we first initialize the transition and emission matrices, as well as the initial probabilities, simply by giving guesses of them. It can be done randomly if there is no information about them.

2. **Forward probabilities.** Here we apply a forward procedure to compute the probabilities $\alpha_t(i) = P(O_1, .., O_t, X_t = i|\theta)$, i.e. the probability of seeing the observation $O_{1:t} = O_1, O_2, ..., O_t$ and being in state i at time t. To compute this we can write

$$\begin{aligned}\alpha_t(i) &= P_\theta(O_{1:t}, X_t = i)\\ &= \sum_{k=1}^{N} P_\theta(X_t = i, X_{t-1} = k, O_{1:t})\\ &= \sum_k P_\theta(O_t|X_t = i, X_{t-1} = k, O_{1:t-1})P_\theta(X_t = i|X_{t-1} = k, O_{1:k-1})P_\theta(X_{t-1} = k, O_{1:t-1})\end{aligned}$$

We observe that thanks to the Markov property we have $P_\theta(O_t|X_t = i, X_{t-1} = k, O_{1:t-1}) = P_\theta(O_t|X_t = i) = e_i(O_t)$ and $P_\theta(X_t|X_{t-1}, O) = P_\theta(X_t = i|X_{t-1} = k) = p_{ki}$. Moreover notice that $P_\theta(X_{t-1} = k, O_{1:t-1}) = \alpha_{t-1}(k)$. Then the probabilities α are recursively given by:

$$\begin{cases}\alpha_1(j) = \pi_j e_j(a_1) 1 \leq j \leq n\\ \alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) p_{ij} e_j(O_t); \quad 1 \leq j \leq N, 1 \leq t \leq T\end{cases}$$

Notice that summing up all the forward probabilities in T we obtain the probability of seeing the sequence of observation $O = O_1 O_2 ... O_T$:

$$P(O_{1:T}|\theta) = \sum_{i=1}^{N} \alpha_T(i)$$

3. **Backward phase.** In this phase we compute the probabilities $\beta_t(i) = P(O_{t+1:T}, X_t = B_i|\theta)$ of seeing the observations from time $t+1$ to the end, and being in state i at time t. Similarly to the forward procedure, we deduce these probabilities recursively:

$$\begin{cases} \beta_T(i) = 1 \quad 1 \leq i \leq N \\ \beta_t(i) = \sum_{j=1}^{N} p_{ij} e_j(O_{t+1}) \beta_{t+1}(j) \;\; 1 \leq i \leq N, 1 \leq t < T \end{cases}$$

As for the forward probabilities, by summing up all the backward probabilities at the initial time we obtain the probability to see the observations $O = O_1 O_2 ... O_T$:

$$P(O_{1:T}|\theta) = \sum_{j=1}^{N} \pi_j e_j(O_1) \beta_1(j)$$

4. **Probabilities γ and ξ.** Using the forward and backward probabilities $\gamma_t(j)$ we can now compute the probability of being in state j at time t given the observations in $O = O_1 ... O_T$:

$$\gamma_t(j) = P(X_t = j|O, \theta)$$

We can compute this thanks to the Bayes theorem:

$$\gamma_t(j) = \frac{P(X_t = j, O|\theta)}{P(O|\theta)} = \frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}$$

Another useful probability we can compute is the probability of being in the state i and state j at time t and $t+1$ respectively, given the observation sequence and the model:

$$\xi_t(i,j) = P(X_t = i, X_{t+1} = j|O, \theta) = \frac{\alpha_t(i) p_{ij} e_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}$$

5. **Estimation.** Using all the probabilities computed in the previous steps we can now compute the transition probability p_{ij} and observation probability $b_i(a_j)$ from an observation sequence, even if we don't know the hidden states sequence.

Let's begin by seeing how to estimate

$$\hat{p}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

The idea behind the computation of the numerator is to sum over all times t the probability to have the transition ij at a specific time t.

The expected number of transitions from state i to state j is then the sum over all t of ξ. Then for the transition probabilities estimate we have:

$$\hat{p}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)}$$

Now we would like to estimate the emission probabilities:

$$\hat{e}_j(a_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } a_k}{\text{expected number of times in state } j}$$

For this, we will need to use the probabilities $\gamma_t(j)$, we have:

$$\hat{e}_j(\nu_k) = \frac{\sum_{t=1}^{T} \mathbb{1}_{O_t=\nu_k}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)}$$

We repeat iteratively these steps, starting each time with the transition and emission matrices just estimated, until we reach some convergence conditions, which could be for example that the change in norm of the transition and emission matrices are less than a tolerance value.

Algorithm 1 Baum-Wech algorithm

```
**Input**: guess for transition matrix $P = (p_{ij})_{ij}$, guess for emission matrix $E = (e_{ij})_ij$, initial probability vector, string of observation $a = a_1a_2...a_T$ .
**while** Conditions > tolerance

Compute needed probilities
  • Forward probabilities
      $\begin{cases} \alpha_1(j) = \pi_j e_j(a_1) 1 \leq j \leq n \\ \alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) p_{ij} e_j(O_t); \quad 1 \leq j \leq N, 1 \leq t \leq T \end{cases}$
  • Backward probabilities
      $\begin{cases} \beta_T(i) = 1 \quad 1 \leq i \leq N \\ \beta_t(i) = \sum_{j=1}^{N} p_{ij} e_j(O_{t+1}) \beta_{t+1}(j) \;\; 1 \leq i \leq N, 1 \leq t < T \end{cases}$
  • Compute $\gamma$ and $\xi$ probabilities
      $\begin{cases} \gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)} \quad 1 \leq j \leq N, 1 \leq t \leq T \\ \xi_t(i,j) = \frac{\alpha_t(i) p_{ij} e_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)} \end{cases}$

Estimation
  • $\hat{p}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{k=1}^{N}\xi_t(i,k)}$
  • $\hat{e}_j(\nu_k) = \frac{\sum_{t=1}^{T}\mathbb{1}_{O_t=\nu_k}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)}$

Compute convergence condition parameter, could be one or more of the following:
  • Conditions=$\|P - \hat{P}\|$
  • Conditions = $\|E - \hat{E}\|$
  • Conditions = log likelihood of the string with estimated matrices
```

```
Update
    • $P \leftarrow \hat{P} = (\hat{p}_{ij})$
    • $E \leftarrow \hat{E} = (\hat{e}_{ij})$
end while
return P,E
```

Original transition	Estimated transition	Original emission	Estimated emission
$\begin{bmatrix} 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix}$	$\begin{bmatrix} 0.68 & 0.32 \\ 0.22 & 0.78 \end{bmatrix}$	$\begin{bmatrix} 0.1 & 0.2 & 0.7 \\ 0.7 & 0.2 & 0.1 \end{bmatrix}$	$\begin{bmatrix} 0.12 & 0.20 & 0.69 \\ 0.71 & 0.19 & 0.10 \end{bmatrix}$
$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0.2 & 0.8 \\ 0.8 & 0 & 0.2 \end{bmatrix}$	$\begin{bmatrix} 0.04 & 0.95 & 0 \\ 0 & 0.17 & 0.83 \\ 0.77 & 0.05 & 0.19 \end{bmatrix}$	$\begin{bmatrix} 0.2 & 0.2 & 0.6 \\ 0.6 & 0.2 & 0.2 \\ 0.4 & 0.4 & 0.2 \end{bmatrix}$	$\begin{bmatrix} 0.24 & 0.18 & 0.58 \\ 0.58 & 0.22 & 0.20 \\ 0.40 & 0.41 & 0.20 \end{bmatrix}$
$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.2 & 0.8 & 0 \\ 0 & 0 & 0.2 & 0.8 \\ 0.8 & 0 & 0 & 0.2 \end{bmatrix}$	$\begin{bmatrix} 0.02 & 0.98 & 0 & 0 \\ 0 & 0.19 & 0.81 & 0 \\ 0 & 0.01 & 0.24 & 0.75 \\ 0.83 & 0 & 0 & 0.17 \end{bmatrix}$	$\begin{bmatrix} 0.1 & 0.22 & 0.7 \\ 0.7 & 0.2 & 0.1 \\ 0.4 & 0.5 & 0.1 \\ 0.1 & 0.5 & 0.4 \end{bmatrix}$	$\begin{bmatrix} 0.12 & 0.22 & 0.66 \\ 0.68 & 0.20 & 0.12 \\ 0.40 & 0.51 & 0.09 \\ 0.11 & 0.48 & 0.41 \end{bmatrix}$

Table 3.1: Estimated probabilities obtained by training sequences of simulated data of length $k = 5000$.

3.4 Implementation

We want to use the algorithm to infer grammars from dataset of sentences. In order to test the algorithm and apply it to our data set, we will use the already implemented MATLAB function `hmmtrain` which, given a sequence of observations, a guess for the transition matrix and a guess for the emission matrix, returns the estimated transition and emission matrices and the loglikelihood of the observed sequence, using the Baum-Welch algorithm. However, we have to be careful while using the Baum-Welch algorithm, because it can have some convergence problems due to the existence of possible locally optimal results. Indeed, depending on the starting guessing matrices the algorithm could not converge to the global optimal solution but to the locally optimal one. To divert this problem we ran the algorithm starting from different random initial guesses and took the results with the best loglikelihood.

We test the correctness of the algorithm simulating data using the MATLAB function `hmmtrain`. We can observe that the implemented algorithm is quite accurate for different number of hidden states as shown in Table 3.1

As we can see the algorithm gives quite accurate estimates for different cases with different number of hidden states.

3.5 Deducing the number of hidden states

Given observed data, we usually don't know how many hidden states characterize the HMM, so the next step is to infer the actual number of hidden states given the observed sequence. To do so we run the Baum-Welch algorithm over different number of hidden states and compute for each of them the Bayesian Information Criterion (BIC)

$$BIC = -2 \log \mathbf{L} + p_n \cdot \log k$$

where:

- **L** is the likelihood function
- k is the number of the observed data
- p_n is the number of free parameters to be estimated, in our case for n hidden states and e emission symbols $p_n = n(n + e - 1)$

The BIC is a criterion that allow us to avoid the overfitting problem that we would have by increasing the number of parameters, it does so by introducing a penalty term given by $p_n \cdot \log k$. Hence we will choose as the best model the one that has the lowest BIC.

Again, we test the correctness of the criterion using simulated data to check if the method infer correctly the number of hidden states given a sequence. The tests are made using the same HMM simulated in the previous section (see Table 3.1). The results are shown in Figure 3.1, where we plotted the BIC value computed with respect to the number of hidden state guessed. As we can see the minimum BIC value is always reached at the right number of hidden states.

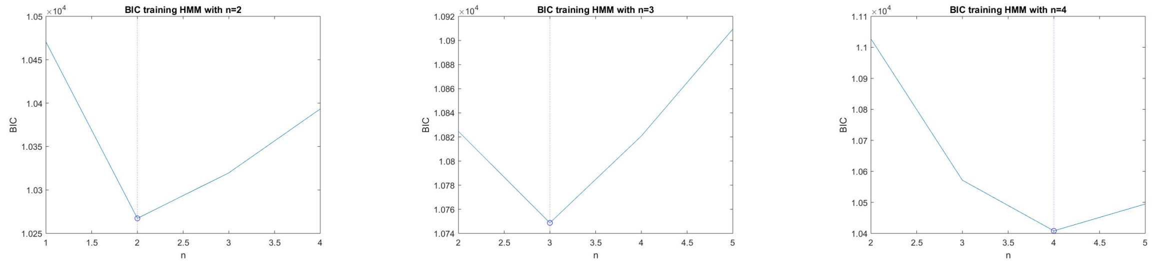


Figure 3.1: From left to right, the graph of the BIC value tested on HMM with number of hidden states 2, 3 and 4 respectively.

As we can observe, by minimizing the Bayesian Information Criterion we can correctly infer the number of hidden states.

In chapter 6 we will see some applications of the method to some unknown grammar.

Chapter 4

Fleet Library

In the previous chapter we introduced a method to study regular grammars, we wish now to study grammars more in general. For this analysis we will use the approach introduced by Yang and Piantadosi in [1].
In particular we will use the library *Fleet* (https://github.com/piantado/Fleet) they implemented for some data we have. But let us explain the method.

4.1 The model

Fleet is a C++ library for the construction of rules for grammars. In particular this is done by specifying a grammar of primitive operations which can be composed to form complex hypotheses, following the idea in [8]. These hypotheses are functions that could be composed of multiple factors recalling themselves. With the library Fleet we observe the data and infer the most likely hypothesis that could have generate such data.

The hypothesis are generated by the grammar with primitive elements displayed in table 4.1. The hypothesis are explored through MCMC techniques. In particular we will use the parallel tempering search described in section 4.2.2. After the search Fleet returns the hypotheses with the highest posterior probability.

4.2 Sampling through MCMC

The sampling of the hypotheses is done using Markov Chain Monte Carlo (MCMC) methods, in particular using the Metropolis Hasting algorithm [6] together with parallel tempering method [9] we describe in the following. The sampling is done from the posterior distribution.

4.2.1 Metropolis-Hasting Algorithm

The Metropolis–Hastings algorithm is a MCMC method to sample elements from an unknown distribution, of which we only know a function proportional to it.

Suppose then that we want to obtain random samples $(Z_n)_n$ from a distribution π, but we only know $\hat{\pi}$, which is proportional to it. To do so we generate a Markov Chain such that its stationary distribution is our distribution target π.

We start with any Z_0 as the first observation and find iteratively Z_{n+1} given Z_n. The algorithm is defined as follows: first of all we choose an arbitrary distribution from which sample a candidate Y_n, the usual distribution is the gaussian distribution centered at Z_n, but it could be any:

Primitive	**Description**
Functions on lists (strings)	
pair(L,C)	Concatenates character C onto list L
head(L)	Return the first character of L
tail(L)	Return everything except the first character of L
insert(X,Y)	Insert list X into the middle of Y
append(X,Y)	Append list X and Y
Logical functions	
flip(p)	Returns true with probability p
X==Y	True if string X is the same string as Y
empty(X)	True if string X if empty; otherwise, false
if(B,X,Y)	Return X if B else return Y
	(X and Y may be lists, sets or probabilities)
and, or, not	Standard Boolean connectives
Set functions	
Σ	The set of alphabet symbols
{s}	A set consisting of a single string
set∪set	Union of two sets
set\s	Remove a string from a set
sample(set)	Sample from a set
Strings and characters	
∅	empty symbol
x	the argument to the function
'a','b','c',...	Alphabet characters (language specific)
Function calls	
Fi(z)	Calls factor Fi with argument z

Table 4.1: Table of the primitives elements from which compose hypotheses

$$Y_n = Z_n + \epsilon, \quad \text{with } \epsilon \sim \mathcal{N}(0, \Sigma).$$

We then define the following ratio

$$\alpha = 1 \wedge \frac{\pi(Y_n)}{\pi(Z_n)} = 1 \wedge \frac{\hat{\pi}(Y_n)}{\hat{\pi}(Z_n)}$$

Notice that in the ratio we can consider the known function $\hat{\pi}$ obtaining the same ratio as if considering π, because the two are proportional. Then with probability α we accept the candidate Y_n, otherwise we keep the previous sample Z_n, more specifically:

$$Z_{n+1} = \begin{cases} Y_n & \text{with probability } \alpha, \\ Z_n & \text{with probability } 1 - \alpha, \end{cases}$$

It can be proved that a Markov Chain defined like this has stationary distribution π [6]. Indeed we recall that for a Markov chain X_t of stationary distribution $P(X_i|X_j)$, π is its stationary distribution if and only if it satisfies the Detail Balance Equation:

$$P(X_i|X_j)\pi(X_j) = P(X_j|X_i)\pi(X_i)$$

Let's verify that such equation is satisfied. The transition distribution of the Markov chain we have built is $P(Y_n|Z_n) = \alpha_{ZY} = 1 \wedge \frac{\pi(Y_n)}{\pi(Z_n)}$. Suppose that we have $\pi(Y_n) > \pi(Z_n)$:

$$\begin{aligned}
P(Y_n|Z_n)\pi(Z_n) &= \left(1 \wedge \frac{\pi(Y_n)}{\pi(Z_n)}\right)\pi(Z_n)\\
&= \pi(Z_n)\\
&= \frac{\pi(Z_n)}{\pi(Y_n)}\pi(Y_n)\\
&= \left(1 \wedge \frac{\pi(Z_n)}{\pi(Y_n)}\right)\pi(Y_n)\\
&= P(Z_n|Y_n)\pi(Y_n)
\end{aligned}$$

Similarly we can verify the Detail Balance Equation also in the case $\pi(Y_n) < \pi(Z_n)$ and this prove that π is the stationary distribution for the Markov chain we have constructed.

4.2.2 Parallel Tempering

There might be problems to explore the distribution when π is multimodal because the chain could get stuck in one mode. In this case it is useful to use the parallel tempering technique: we can flatter the distribution π by using a parameter $T > 1$ (that we call temperature) writing $\pi^{1/T}$. The idea is to take $T_1, T_2, ..., T_n$ and at random steps to swap from a temperature to another, this swaps happen with probability

$$A_{i,j} = \min\left\{\left(\frac{\pi(\theta_i)}{\pi(\theta_j)}\right)^{\beta_j-\beta_i}, 1\right\}$$

where θ_i is the current position in the parameter space of the i^{th} chain and $\beta_i = 1/T_i$ is the inverse temperature of this chain. When a swap is accepted, the chain exchange their position in the parameter space, so that chain i is at θ_j and chain j is at θ_i. In this way the chain manage to explore the entire distribution.

4.2.3 Adaptive Temperature Ladders

Parallel tempering is a very efficient method for sampling that overcomes the problem of sampling multi-modal probability distribution. However a difficult task is finding the ladder of temperature that better explore such distribution. We have that at high temperatures it is easier to switch from one mode to another, while at low temperatures the single mode is more correctly explored.

We wish for a good ladder that satisfies the following features:

- T_{max} large enough so that at this temperature it is easy to access to all the modes of the distribution;
- the difference $\beta_i - \beta_j$ to be small enough so that A_i, j is not too small, allowing neighbouring chain to swap.

These requirements depend on the shape of the distribution, which we do not know. In the literature [10], [11] [12] [13] it has been suggested that a good ladder is one where the acceptance ratios $A_{i,j}$ are uniform for all pairs (i, j) of adjacent chains.

Assuming that such a ladder gives good results, Vousden, Farr and Mandel suggest in [9] an adaptive temperature ladder, which is an algorithm that adapts the temperature ladder at every swap having uniform rate of exchange between chains. Let us explain such algorithm. We define the following term:

$$S_i := \log(T_i - T_{i-1})$$

We will have that the correct ordering of temperatures ($T_1 < ... < T_N$) will always be preserved. In order to have T_{max} large enough we simply impose β_N. In order to obtain the same acceptance rate for all chains we impose:

$$\frac{dS_i}{dt} = k(t)[A_i(t) - A_{i+1}(t)], \quad \text{for } 1 < i < N \tag{4.1}$$

where k is a positive constant that we will discuss later, t indicates the iteration and $A_i := A_{i,i-1}$.

With this scheme we have that if swaps between itself and adjacent chains are accepted too often then the gap between the temperature is increased, while, on the contrary, if they are not accepted very often then the gap is decreased. Discretising 4.1 we have:

$$S_i(t+1) - S_i(t) = k(t)[A_i(t) - A_{i+1}(t)]$$

We have to remark, however, that the scheme proposed and every adaptive sampling scheme do not satisfy the condition for detailed balance that guarantee an MCMC sampler to converge. However, from [14] we know that in order for an adaptive sampler to converge it is sufficient for it to be ergodic in the target distribution. They determined that we can obtain this by diminishing the amplitude of adaptations at each iteration. For this purpose we use the parameter $k(t)$ and we define it as follows:

$$k(t) = \frac{1}{\nu}\frac{t_0}{t + t_0}$$

where further considerations mande in [9] suggests to take as default parameter values $\nu = 10^2/n_c$ and $t_0 = 10^3/n_c$, with n_c the number of chains.

4.3 Likelihood probability

The model is such that it allows errors or noise so that almost correct strings can have non zero likelihood. Such likelihood is computed as follows: starting from the hypothesis found we generate the most likely strings that can be generated. Then we define p_{del} the probability of eliminating the last element from the string and p_{add} the probability of adding one element. Consider a string x of length S_x that is generated by the hypothesis found with probability P_x, we compute the probability that this string can be equivalent to a string y of length S_y of the dataset we possess by deleting and adding elements from this string.

The probability to delete exactly m elements from the tail of the string is

$$p_{del}^{(S_x - m)} \cdot (1 - p_{del})$$

While the probability to add the exact elements to convert the truncated string x into y is

$$\left(p_{add} \cdot \frac{1}{N}\right)^{(S_y - m)} \cdot (1 - p_{add})$$

Then summing up over different m, the overall probability to convert x into y is

$$\sum_{x_{1:m} = y_{1:m}} p_{del}^{(S_x - m)} \cdot (1 - p_{del}) \cdot \left(p_{add} \cdot \frac{1}{N}\right)^{(S_y - m)} \cdot (1 - p_{add})$$

where $x_{1:m}$ denotes the string composed of the first m element of string x.

We then sum this probability to the probability of generating x and compute this probability for every string generated by the hypothesis, whose set we denote with X and every string in the dataset Y:

$$\sum_{x\in X}\left(P_x+\sum_{y\in Y}\sum_{x_{1:m}=y_{1:m}} p_{del}^{(S_x-m)}\cdot(1-p_{del})\cdot\left(p_{add}\cdot\frac{1}{N}\right)^{(S_y-m)}\cdot(1-p_{add})\right)$$

This is the likelihood of a hypothesis.

4.4 Prior

As already mentioned, hypotheses are constructed through a grammar composed of primitive operations that are combined to form more complex hypotheses. Based on how these hypotheses are composed, we assign a probability. This prior is calculated by assigning a probability to each rule that characterises the grammar that constructs the hypotheses, in this way the more complex the hypothesis the lower the prior will be. Let us specify the grammar $\mathcal{G}$ for the construction of such hypotheses: we have 6 non terminal symbols and the following rules

L → pair(L,C)	L → if(B,L,L)
L → head(L)	L → sample(S)
L → tail(L)	L → ∅
L → insert(L,L)	L → x
L → append(L,L)	L → Fi(L)
S → Σ	B → flip(N)
S → {L}	B → L==L
S → S∪S	B → empty(L)
S → S \L	B →and(B)
S → if(B,S,S)	B → or(B)
	B → not(B)
N → 1/24, 1/12, ..., 1/2	C → ‘a’,‘b’,‘c’,...
N → if(B,N,N)	

To define the prior we follow the work in of Goodman, Tenenbaum, Feldman and Griffiths [8]. Let F be a hypothesis generated by the grammar G which has distribution τ and let $Deriv_F$ be the set of all possible ways to obtain F through this grammar, then the prior probability is given by

$$P(F|\mathcal{G},\tau)=\prod_{s\in Deriv_F}\tau(s),$$

However,we don't want to fix the distribution of the production probabilities, thus we write

$$\begin{aligned}P(F|\mathcal{G}) &= \int P(F,\tau|\mathcal{G})d\tau\\ &= \int P(\tau)P(F|\tau,\mathcal{G})d\tau\\ &= \int P(\tau)\left(\prod_{s\in Deriv_F}\tau(s)\right)d\tau\end{aligned}$$

Where $P(\tau)$ is the prior probability for a given set of production probabilities. Because we have no a priori reason to prefer one set of values for τ to another, we select the least informative prior: $P(\tau) = 1$. The probability of a formula becomes:

$$P(F|\mathcal{G}) = \int \left(\prod_{s \in Deriv_F} \tau(s) \right) d\tau$$

After some simplification done in [8] by recognizing the integral as a Multinomial-Dirichlet form we obtain:

$$P(F|\mathcal{G}) = \prod_{Y \in \text{non-terminals of } \mathcal{G}} \frac{\beta(\mathbf{C}_Y(F) + \mathbf{1})}{\beta(\mathbf{1})}$$

where the term $\mathbf{C}_Y(F)$ indicates the vector of counts of the productions for non-terminal symbol Y in $Deriv_F$ (and $\mathbf{1}$ is the vector of ones of the same length). The function $\beta(c_1, c_2, ..., c_n)$ is the multinomial beta function for the partial counts $(c_1, c_2, ..., c_n)$, which can be written in terms of the Gamma function:

$$\beta(c_1, c_2, ..., c_n) = \frac{\prod_{i=1}^{n} \Gamma(c_i)}{\Gamma(_{i=1}^{n} c_i)}$$

4.5 LogSumExp trick

During the implementation of the method, it is easy to come across underflow or overflow problems given the nature of the data, especially since we usually work with small probabilities. To overcome this problem all probabilities are computed as log probabilities so that the multiplication of two small numbers does not give underflow problems, but it becomes a sum. Following this idea another trick is to use the LogSumExp operation, for example when we want to sum small probabilities, this is how it works: suppose we have $\hat{p}, \hat{q}$ two small probailities, we want to sum the two knowing their logprobabilities p and q respectively, then we would do $\hat{p} + \hat{q} = \exp(p) + \exp(q)$ but since this could give underflow we define $m := \max(p, q)$ and write $\hat{p} + \hat{q} = \exp(p) + \exp(q) = m + \log(\exp(a - m) + \exp(b - m))$. This is the LogSumExp trick, more in general, given $p_1, p_2, ..., p_n$ logprobabilities:

$$LSE(p_1, p_2, ..., p_n) = m + \log \left(\sum_{i=1}^{n} \exp(p_i - m) \right)$$

4.6 How to use Fleet

Fleet is a C++ library implemented by Piantadosi.
Please read the web page https://codedocs.xyz/piantado/Fleet/index.html for a brief introduction. It uses libraries that only work in Linux, so it is recommended to install Linux before using it or work on a Linux virtual machine. The easiest way to use Fleet is to start from the examples in the `/Fleet-master/Models` folder. Various examples of different kinds are presented here, the one we will use to study grammar is FormalLanguageTheory-Complex.

In the example folder there is, among other files, the main file `Main.cpp` , a `MyGrammar.h` file that contains the grammar defining the construction of the hypotheses and a `MyHypothesis.h` file that allows the hypotheses to be generated from the grammar. There is also a `Makefile` that debugs and runs the code, to do this, simply go to the example folder via the terminal and enter the command `make`, this will create a main file which will be sufficient to run the algorithm. To make it run then just enter the command `./main` followed by the parameters

Command	Description
`--input=`	text file containig the input data (do not write the suffix ".txt")
`--alphabet=`	string containing the terminal symbols in the data
`--time=`	running time, need to write a unit of measurement: seconds(s), minutes(m), hours(h), days(d)
`--steps=`	number of steps
`--top=`	number of top hypotheses to show
`--nfactors=`	number of factors composing the hypothesis
`--chains=`	number of chains on which to run the MCMC search

Table 4.2: Table of parameters for Fleet

you are interested in. The most relevant parameters we used are those shown in the Table 4.2.

To be able to use your own data, you must first make some changes to the main file. First make sure that the vector `data_amounts` declared on line 32 has size 1. then comment out the current definition of the string `data_path` on line 113 and redefine it as

```
S data_path = FleetArgs::input_path + ".txt";
```

You can now safely use your data file. Be careful to declare it in the folder in which it is located.

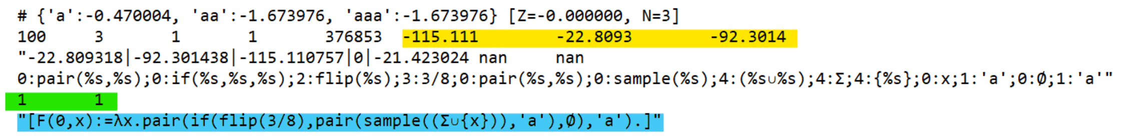
4.6.1 Example of output

Let us give a simple example of what kind of output we could expect. This is the hypothesis we obtain by giving as input data from the grammar AAA,which generates the strings in the set {a,aa,aaa}:

```
F(0,x):=.pair(if(flip(3/8),pair(sample((Σ∪{x})),'a'),),'a').
```

The default argument when calling F is an empty string. Hence the hypothesis generates 'a' with probability 5/8, 'aa' with probability $1/2 \cdot 3/8$ as well as for 'aaa'.

After running the code, certain information are printed for each hypothesis, which we can see illustrated in below.



```
# {'a':-0.470004, 'aa':-1.673976, 'aaa':-1.673976} [Z=-0.000000, N=3]
100     3       1       1       376853  -115.111        -22.8093        -92.3014
"-22.809318|-92.301438|-115.110757|0|-21.423024 nan     nan
0:pair(%s,%s);0:if(%s,%s,%s);2:flip(%s);3:3/8;0:pair(%s,%s);0:sample(%s);4:(%s∪%s);4:Σ;4:{%s};0:x;1:'a';0:∅;1:'a'"
1       1
"[F(0,x):=λx.pair(if(flip(3/8),pair(sample((Σ∪{x})),'a'),∅),'a').]"
```

The hypotheses are printed in order of decreasing posterior probability. For each hypothesis, the strings that are generated with the highest probability are shown with their respective probability. Highlighted in yellow we have in order logPosterior, logPrior and logLikelihood. Highlighted in green we have the precision and recall values. Finally in light blue we have the hypothesis that has been sampled. Notice that the code prints also the primitives used to build the hypotheses, preceded by the terminal that called them.

Chapter 5

Grammar inferring

Let us now analyse grammars using the tools we have presented so far, that is, the Baum Welch algorithm and the Fleet library. We will first analyse the a^nb^n grammar which generates strings composed of 'a's successive to an equal number of 'b's. We will then analyse a simplified version of the English grammar, which we will define specifically later. To analyse such grammars, we will generate data that we will use as input in the various algorithms. We will then move on to analysing real data, in particular we will analyse sentences from Harry Potter's book and a dataset of Campbell monkey calls. Our goal is to be able to classify the grammars that characterise them.

5.1 $\mathbf{a^nb^n}$

In the study of the grammar a^nb^n we will use the data generated in Yang and Piantadosi's study [1]. But first let us show through the Pumping lemma that the grammar is non regular:

Claim. *The grammar a^nb^n is not a regular grammar.*

Proof. To prove the claim we just need to prove that the grammar does not satisfy the Pumping lemma conditions. Suppose by contradiction that $\exists p \geq 1$ s.t. the Pumping lemma is satisfied, then take $n = \lceil \frac{p}{2} \rceil$ and the string $w = a^nb^n$, where w has length greater0 or equal to p. We want to find the decomposition $w = xyz$ that satisfies the statement of the lemma i.e. such that $\forall k \;\; xy^kz$ is a string in a^nb^n.

Suppose y is made of only 'a's. For simplicity suppose also that z is only made of 'b's. Then we have $x = a^q$, $y = a^{n-q}$, $z = b^n$. We have then $|y| \geq 1, |xy| = n = \lceil \frac{p}{2} \rceil \leq p$ which are the hypothesis of the Pumping lemma. But taking for example $k = 2$ we have that $xy^2z = a^qa^{2n-2q}b^n = a^{2n-q}b^n$ wich is not a string that can be generated by a^nb^n.

With similar reasoning, it is easy to show that the lemma is not satisfied even in the case where y is of the form b^q or a^jb^q. □

Now we use the Baum-Welch algorithm together with the BIC criterion to study which HMM, hence regular grammar, best approximates it. As we can see from the Figure 5.1, the best approximation is an HMM with three hidden states.

We also obtain that it approximates it with likelihood equal to -170.547 and its transition and emission matrices are

$$\mathbf{P} = \begin{pmatrix} 0.50 & 0.00 & 0.50 \\ 0 & 1 & 0 \\ 0.06 & 0.73 & 0.21 \end{pmatrix}$$

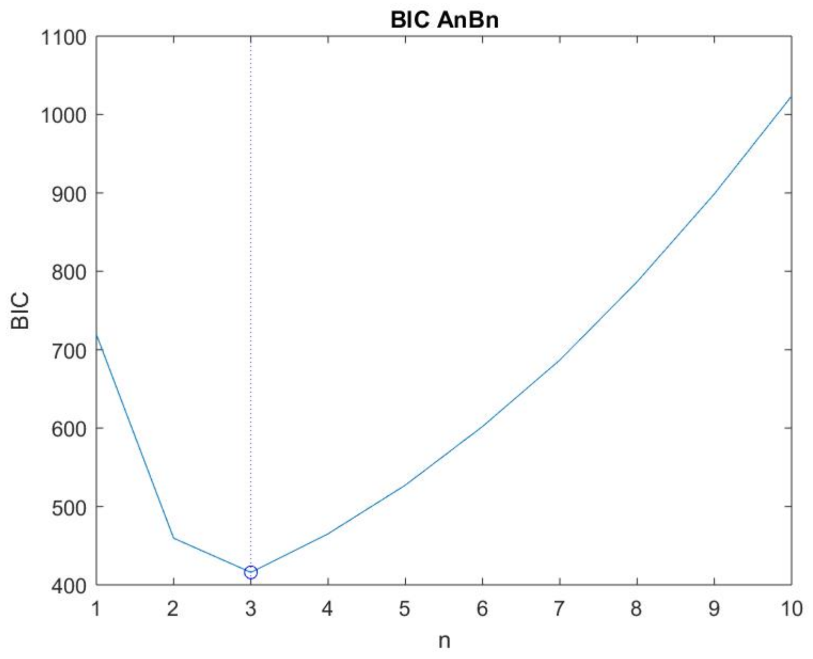
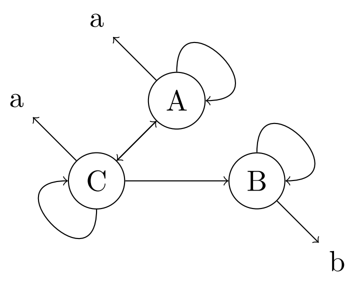


Figure 5.1: BIC from data generated from grammar $a^n b^n$, the lowest BIC is reached with $n = 3$ hidden states

$$\mathbf{E} = \begin{pmatrix} 1 & 0.00 \\ 0 & 1 \\ 1 & 0.00 \end{pmatrix}$$



By naming the hidden states A, B and C, we can observe that, starting from A, the HMM initially oscillates between states A and C emitting only ‘a’ and then arrives at state B and remains there emitting only ‘b’. We can therefore observe that the HMM recreates similar data from the $a^n b^n$ grammar, however it is unable to replicate exactly the rule that the number of ‘a’s equals the number of ‘b’s as it can not count the number of times they are emitted.

Let us now try to analyse the data using Fleet. In our work we use 50 numbers for `flip()` probabilities and do the search for 10m. The top hypothesis we find is

```
F(0,x):=x.pair(append(pair(x,‘a’),if(flip(31/100),∅,F0(x))),‘b’)
```

with likelihood probability of -40.1515. The value of precision and recall are 1 and 1.

We therefore have a higher likelihood, so if we did not know the true grammar behind the data we have, we would accept Fleet's result. Indeed, it is easy to observe that the hypothesis found adds a 'b' each time an 'a' is generated. We therefore have that in this case Fleet recognises the correct grammar which we know to be a non-regular grammar.

5.2 Simplified English

We would like to study now simplified English grammar, a very simplified version of English grammar that can be found in introductory textbooks:

$$\begin{aligned} S &\to NP\,VP \\ NP &\to n \mid d\ n \mid d\ AP\ n \mid NP\ PP \\ AP &\to a \mid a\ AP \\ VP &\to v \mid v\ NP \mid v\ t\ S \mid VP\ PP \\ PP &\to p\ NP \end{aligned}$$

It is not so easy to verify which class this grammar belongs to. In this case we cannot use the Pumping lemma as at first sight this grammar would seem to satisfy it. Let us then use the Baum-Welch algorithm as before to find the best fitting HMM and see if such HMM replicates the same behaviour of the grammar defined above . We perform a search for the best parameters using again the BIC criterion and obtain that the best estimate has 6 hidden states and a logLikelihood of -592.52.

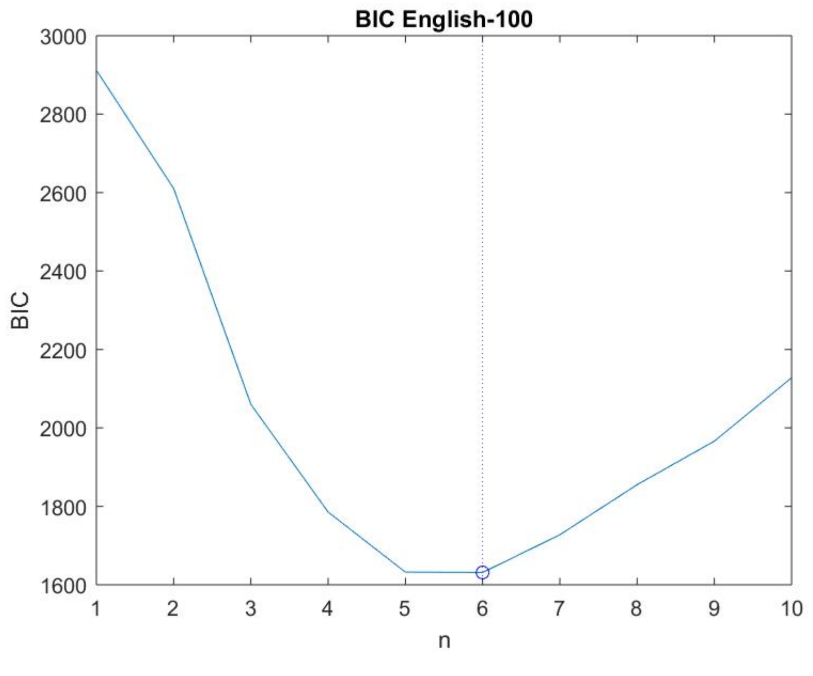
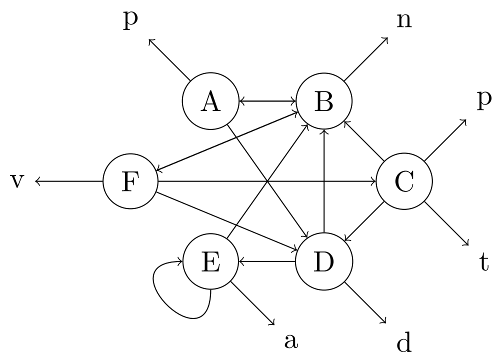


Figure 5.2: Bayesian Information Criterion applied to English dataset, the lowest BIC value is reached in $n = 6$ and has value 1631.31 and the log likelihood has value -592.52.

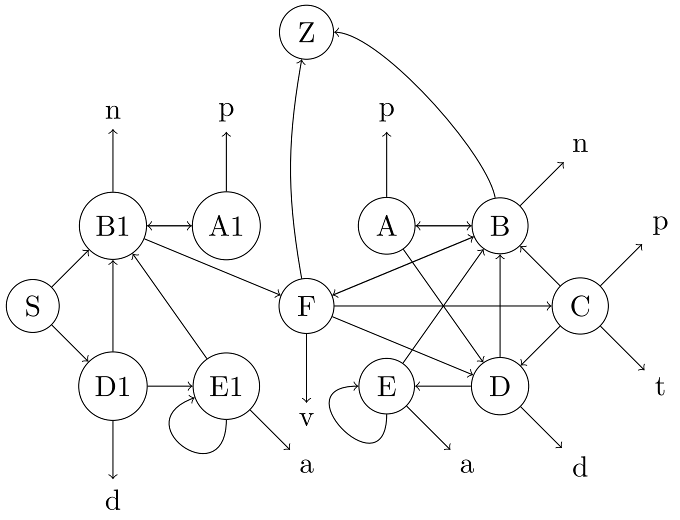
The best HMM has transition and emission matrices respectively of:

$$\mathbf{P} = \begin{pmatrix} 0 & 0.3026 & 0 & 0.6974 & 0 & 0 \\ 0.2464 & 0 & 0 & 0 & 0 & 0.7536 \\ 0 & 0.3714 & 0 & 0.6286 & 0 & 0 \\ 0 & 0.4565 & 0 & 0 & 0.5435 & 0 \\ 0 & 0.7812 & 0 & 0 & 0.2188 & 0 \\ 0 & 0.1261 & 0.5882 & 0.2857 & 0 & 0 \end{pmatrix}$$

$$\mathbf{E} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.8429 & 0.1571 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$



In the graph the circle nodes are the hidden states, while the other nodes are the emission. We can see that the HMM above describes the behaviour of the grammar quite correctly. However, by not imposing the option of a start and an end state in the definition of the algorithm, it does not respect certain characteristics of the grammar, in particular the fact that each generated sentence must contain at least an ‘n’ and a ‘v’. To overcome this issue we can easily modify the HMM by adding a ”starting HMM” and other hidden states as follows.

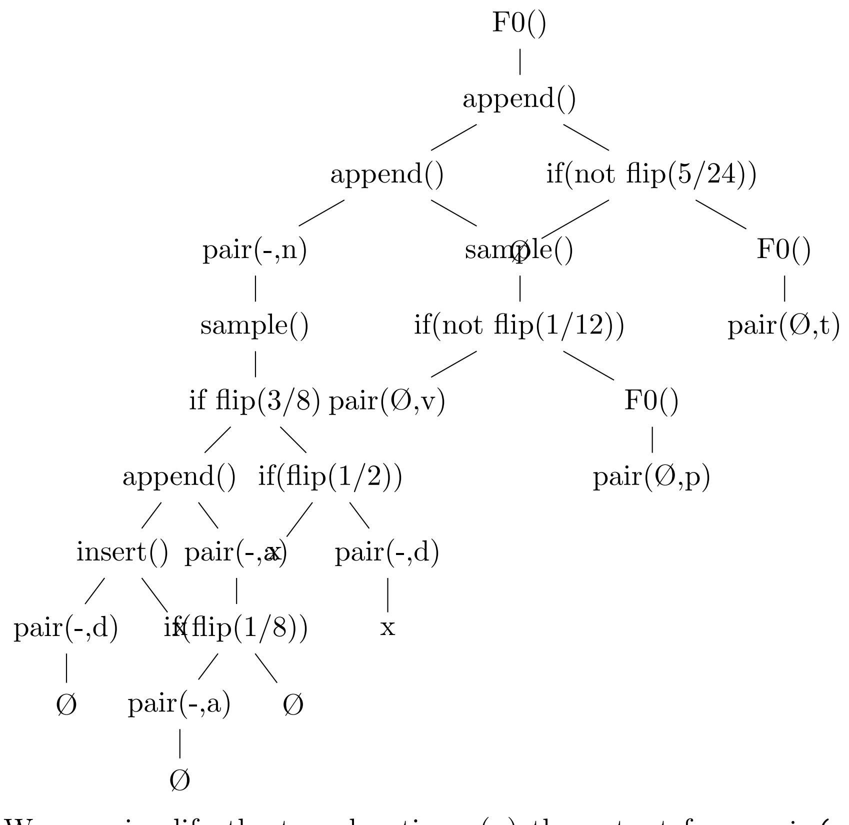


This HMM replicates well the behaviour given by the rules of the grammar. If we consider a sentence to start from the node S and finish in node Z, we observe that now in order to reach the end node we have to pass through state B1 that emits ‘n’ and state F that emits a ‘v’. We can conclude that the simplified English grammar is regular as it is possible to represent it with a HMM.

Let us now try to analyse it using Fleet. We run Fleet for 6 hours using 15 chains for the parallel tempering. The best hypothesis we obtain is :

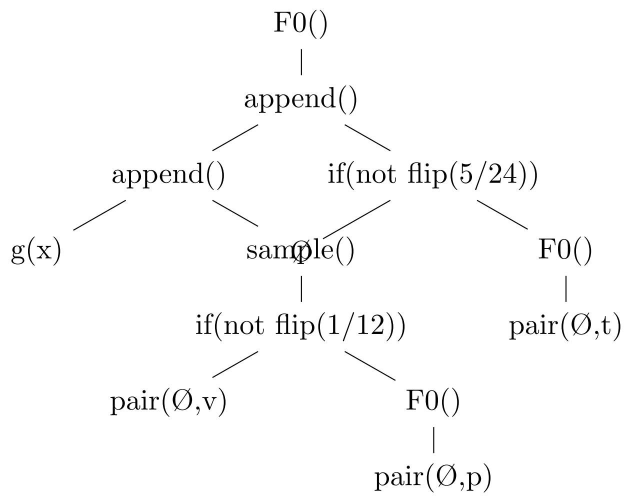
```
F(0,x):=λx.append(append(pair(sample(if(flip(3/8),{append(insert(pair(∅,‘d’),x),
 pair(if(flip(1/8),pair(∅,‘a’),∅),‘a’))},if(flip(1/2),x,pair(x,‘d’)))),‘n’),
        sample(if(not(flip(1/12)),{pair(∅,‘v’)},{F0(pair(∅,‘p’))}))),
                    if(not(flip(5/24)),∅,F0(pair(∅,‘t’)))).
```

with likelihood and posterior probability respectively of -2544.41 and -2661.33, much lower values compared to the one we obtained previously. However we notice that the value of the precision and recall parameters for this hypothesis are 0.343 and 0.178, thus the hypothesis we found is not really accurate for our grammar. Indeed there are many strings in our dataset that can not be generated from this hypothesis. In [1] Yang and Piantadosi manage to obtain good values of precision and recall for this amount of data, by running it for 7 days as it is more complex compared to $a^n b^n$. Unfortunately, we have neither the time nor the technical resources to perform the same computation, thus our results will not be very precise. Therefore, we will not consider Fleet’s results as correct but will only use them as a guideline to deduce ideas and conclusions. So let us analyse anyway what we have obtained. To have a better understanding of the hypothesis we found let us represent it in a tree.

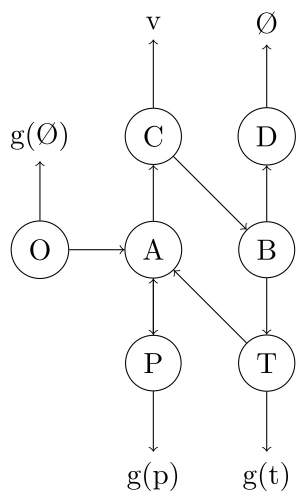


We can simplify the tree denoting $g(x)$ the output from `pair(sample(if(flip(3/8), {append(insert(pair(∅,‘d’),x), pair(if(flip(1/8),pair(∅,‘a’),∅),‘a’))}, {if (flip(1/2),x,pair(x,‘d’))})),‘n’)` where we can see that F0 is never called, so this piece of the function generates a finite number of sentences, which is therefore easily represented by HMMs (trivially, for example by defining each possible sentence as a terminal, the simplest

HMM consists of a single hidden state that can emit each of the terminals with the appropriate probability).



Observing the tree we can notice then that the grammar described by the hypothesis can be represented through the following schematic HMM:



Hence, even if the model is not entirely correct, as we notice the HMM inferred is not really similar to the one we found before, we have that the best fitted grammar that Fleet can find within our resources is still regular.

5.3 Harry Potter data

Let us now analyse some real data, in particular we have a dataset consisting of 107 sentences from Harry Potter's book. The book is written in English so it will also be interesting to analyse it by comparing it with the results for simplified English.

We begin our analysis with the Baum-Welch algorithm. As always we are interested in minimizing the BIC tested for different number of hidden states. We obtain the graph in Figure 5.3

This means that the best HMM that fits our dataset is one with $n = 6$ hidden states, moreover we can also compute the estimated transition and emission matrices, which are:

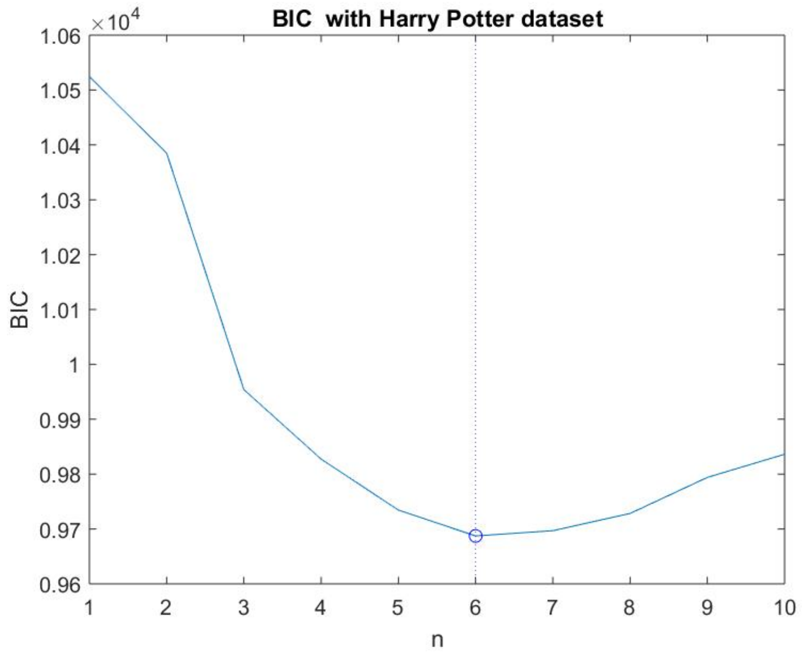


Figure 5.3: Bayesian Information Criterion applied to Harry Potter data, the lowest BIC value is reached in $n = 6$ and has value $9.6873e03$ and the log likelihood has value $-4.3841e03$.

$$\mathbf{P} = \begin{pmatrix} 0.0168 & 0.2657 & 0.4567 & 0.0000 & 0.0821 & 0.1786 \\ 0.0907 & 0.0449 & 0.2649 & 0.0080 & 0.1682 & 0.4232 \\ 0.3600 & 0.1913 & 0.0792 & 0.3454 & 0.0000 & 0.0241 \\ 0.5596 & 0.0000 & 0.0582 & 0.2188 & 0.1017 & 0.0617 \\ 0.0616 & 0.5670 & 0.0959 & 0.0760 & 0.1995 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 1.0000 & 0.000 \end{pmatrix}$$

$$\mathbf{E} = \begin{pmatrix} 0.05 & 0.00 & 0.10 & 0.02 & 0.04 & 0.00 & 0 & 0.00 & 0.00 & 0.01 & 0.00 & 0.00 & 0 & 0.02 & 0.76 \\ 0.00 & 0.42 & 0.00 & 0.00 & 0.07 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.46 & 0.06 & 0.00 \\ 0.00 & 0.00 & 0.20 & 0.00 & 0.05 & 0.00 & 0.01 & 0.00 & 0.00 & 0.07 & 0.33 & 0.33 & 0.00 & 0.01 & 0.00 \\ 0.00 & 0.00 & 0.06 & 0.61 & 0.00 & 0.00 & 0 & 0.00 & 0 & 0.17 & 0.00 & 0.00 & 0.17 & 0.00 & 0.00 \\ 0.19 & 0.00 & 0.01 & 0.00 & 0.00 & 0 & 0 & 0.74 & 0.01 & 0.00 & 0.00 & 0.05 & 0.00 & 0.00 & 0.00 \\ 0.09 & 0.00 & 0.00 & 0.00 & 0.01 & 0.63 & 0 & 0.00 & 0.01 & 0.00 & 0.25 & 0.00 & 0.00 & 0.00 & 0.01 \end{pmatrix}$$

Remark. When written 0.00, we consider the value to be smaller than $5 \cdot 10^{-3}$.

In order to understand better the matrices in our results we have in Table 5.1 the representation of each emission symbol.

Looking at the results we can observe that there are a lot of transitions and emissions that have a low probability to happen, we can then try to apply once again the Baum-Welch algorithm starting from guessing matrices obtained by approximating the estimated matrices. The approximation is done by setting low probabilities (smaller than 0.005) to zero and then normalizing the matrices. In this way we obtain a slightly better result, as the new estimations has a higher log likelihood (-4.38613e03), and more interesting, we have matrices with more

Table of symbols					
1	ADJ	adjective	9	NUM	number
2	ADP	preposition	10	PART	particle
3	ADV	adverb	11	PRON	pronoun
4	AUX	auxiliar verb	12	PROPN	proper noun
5	CCONJ	conjunction	13	PUNCT	punctuation
6	DET	article	14	SCONJ	sub conjuction
7	INTJ	interjection	15	VERB	verb
8	NOUN	noun			

Table 5.1: Table of the speech tag used in the data set

zero entries:

$$\mathbf{P} = \begin{pmatrix} 0.0167 & 0.2657 & 0.4567 & 0 & 0.0821 & 0.1787 \\ 0.0907 & 0.0449 & 0.2649 & 0.0080 & 0.1682 & 0.4232 \\ 0.3600 & 0.1913 & 0.0792 & 0.3454 & 0 & 0.0241 \\ 0.5596 & 0 & 0.0582 & 0.2188 & 0.1017 & 0.0617 \\ 0.0616 & 0.5670 & 0.0959 & 0.0760 & 0.1995 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\mathbf{E} = \begin{pmatrix} 0.06 & 0 & 0.11 & 0.02 & 0.04 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 & 0 & 0.01 & 0.75 \\ 0 & 0.42 & 0 & 0 & 0.07 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.45 & 0.06 & 0 \\ 0 & 0 & 0.21 & 0 & 0.05 & 0 & 0.01 & 0 & 0 & 0.07 & 0.33 & 0.33 & 0 & 0.01 & 0 \\ 0 & 0 & 0.06 & 0.60 & 0 & 0 & 0 & 0 & 0 & 0.17 & 0 & 0 & 0.17 & 0 & 0 \\ 0.20 & 0 & 0.01 & 0 & 0 & 0 & 0 & 0.74 & 0.01 & 0 & 0 & 0.05 & 0 & 0 & 0 \\ 0.10 & 0 & 0 & 0 & 0.01 & 0.63 & 0 & 0 & 0.01 & 0 & 0.25 & 0 & 0 & 0 & 0.01 \end{pmatrix}$$

Observations. We can notice that when being in state 6 the model goes to state 5 almost certainly.We notice also that state 6 is the only state that emits symbol 6 which corresponds to the article particle, while state 5 mainly emits symbol 8 (nouns) with probability 0.74 and symbol 1(adjective) with prob 0.18. This reflects how in English, articles precede only nouns and adjectives.

Moreover we observe that state 6 emits, other than articles (with probability 0.63), also symbol 11, pronouns, with probability 0.26. This could be thought of the rule for the use of possessive pronouns ("my", "yours", "his", "her"..), which in English has to be followed by a noun ("his wife") or an adjective ("his pale [aunt]") (examples taken from the dataset).

Another observation we can make is also notice that symbol 4 (auxiliar verb) is emitted mainly by state 4 with prob 0.60. From state 4 there are two major possible transition: go to state 1 ($p_{41} = 0.56$) which mainly emits verb (symbol 16), that is grammatically the main use of the auxiliar verb, or stay in 4 and emit with higher prob another auxiliar verb or an adjective, reflecting correctly the use of the auxiliar in the English grammar.

However, being such HMM quite complex (as there are many non zero entries in the transition matrix) it is difficult for us to look into further details. We try now to study it with Fleet.

Before using Fleet on the data we convert its format in a way such that its output is a bit easier to read. The conversion is shown in table 5.2

As with the Baum-Welch algorithm, we would like to compute the BIC to find the best number of parameters that gives us the best hypothesis. The parameters varies based on the number of primitives we construct the hypotheses from. We obtain the graph in Figure 5.4

Table of symbols					
1	ADJ	a	9	NUM	b
2	ADP	f	10	PART	p
3	ADV	s	11	PRON	o
4	AUX	x	12	PROPN	r
5	CCONJ	e	13	PUNCT	q
6	DET	d	14	SCONJ	t
7	INTJ	i	15	VERB	v
8	NOUN	n			

Table 5.2: Table of conversion symbols for the Harry Potter data

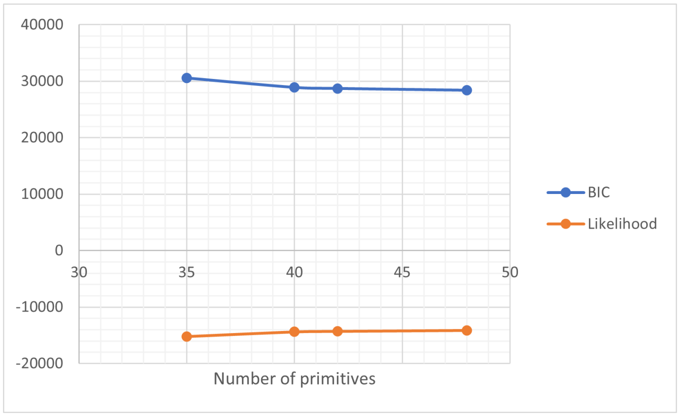


Figure 5.4: Bayesian Information Criterion applied to Harry Potter data using Fleet

We have that the best hypothesis is obtained when doing the sampling with all the primitives. So we do our search with all the primitives. Having a large number of terminals, we decide to run the code for 12 hours with 15 chains, the result we get is the following:

```
F(0,x):=λx.if(not(flip(if(flip(5/12),11/24,1/2))),∅,pair(∅,‘v’)).
F(1,x):=λx.insert(sample(Σ),if(flip(11/24),insert(head(pair(F0(∅),‘x’)),
        if(flip(5/12),pair(x,‘o’),pair(x,‘r’))),sample(Σ))).
```

with loglikelihood and logposterior probability respectively of -14040.1 and -14120.

It is immediate to note that the set of sentences generated by this hypothesis is finite, in fact the factor F0 does not recall any other factors, so as there is no recursion it only produces a finite number of sentences. On the other hand, factor F1, only recalls factor F0 which, as just said, is finite, so the grammar generated by this hypothesis is finite and therefore regular. However we notice that the loglikelihood has a much lower value with respect to the one obtained from the Baum-Welch algorithm. If the hypothesis found was accurate we would have obtained at least similiar value of loglikelihood or higher, because if it was regular the best hypothesis should be similar to the approximation as an HMM, while if it was not regular we would have obtained a more fitting model. However we have to remark that the value of precision and recall for this hypothesis are both 0, which makes us think that this hypothesis is not very accurate. One of the reasons why we have not been able to get a good result is possibly because the data we have is generated by a grammar that consists of a large number of rules, and the data we have are not sufficient to deduce it properly.

Let us then try to manipulate the data to see if perhaps reducing the number of terminals gives a better result. We then merge a few terminals that have a similar syntactic function, this merge is shown in the Table 5.3.

n°of terminals	**Alphabet**	**Merging**
15	nvsadtpiqeforxb	No merging
14	nvsadtpiqeforx	b=d
13	nvsadtpiqefor	b=d, x=v
12	nvsadtpiqefo	b=d, x=v, r=n
11	nvsadtpiqef	b=d, x=v, r=n, o=n
10	nvsadtpiqe	b=d, x=v, r=n, o=n, f=s
9	nvsadtpiq	b=d, x=v, r=n, o=n, f=s, e=t
8	nvsadtpi	b=d, x=v, r=n, o=n, f=s, e=q=t
7	nvsadtp	b=d, x=v, r=o=n, f=i=s, e=q=t
6	nvsadt	b=d, x=v, r=o=n, f=i=p=s, e=q=t
5	nvsad	b=d, x=v, r=o=n, f=i=p=e=q=t=s
4	nvsa	x=v, r=o=n, f=i=p=e=q=t=b=d=s
3	nvs	x=v, r=o=n, f=i=p=e=q=t=d=a=s

Table 5.3: Table of the merging of terminals.

By doing this merging we obtain, as shown in Figure 5.5 that as we merge more terminals the likelihood increases as expected since we are decreasing the number of terminals.

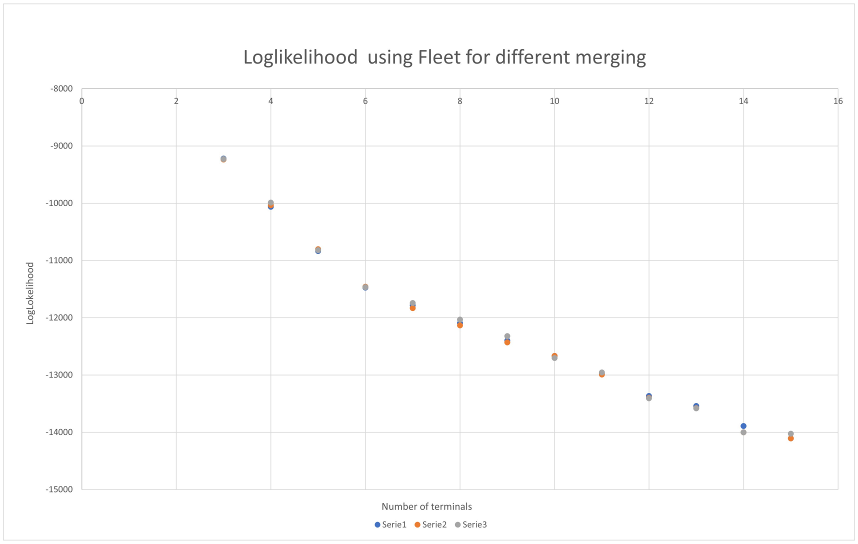


Figure 5.5: Result using Fleet for Harry Potter data with different merging. The analysis has been done for each merge with different number of factors.

In particular, we observe that almost all the hypotheses we obtain, generate a finite grammar and have precision and recall both close to zero. We have exceptions in the merge that gives us 7 terminals, which is about the number of terminals of the simplified English grammar. We therefore decide to manipulate the original data again in a slightly different way: our aim is to make them as close as possible to that of the simplified English grammar we studied earlier. As a first step, we break up the sentences using punctuation and coordinating conjunctions. We then merge by grouping terminals with similar grammatical function together to obtain the same terminals. The merging is

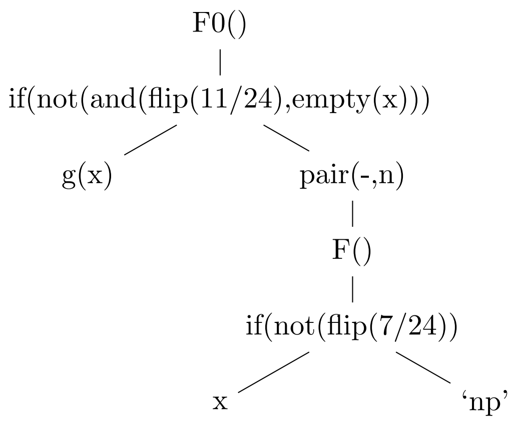
- n indicates noun (n), proper noun (r), pronoun (o), interjections (i);
- v indicates verbs(n), auxiliary verbs (x);
- a indicates adjective (a)
- d indicates article (d) and number(b)
- t indicates subordinating conjunctions (t)
- p indicates particles (p) and prepositions (f)

Finally, as a further simplification we eliminate adverbs, since their presence is not so relevant in the sentence and because they do not properly belong to any of the terminal classes. The best hypothesis we find is

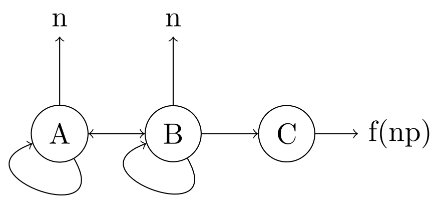
```
F(0,x):=λx.if(not(and(flip(11/24),empty(x))),append(sample((if(flip(1/3),Σ,
        {pair(if(flip(3/8),pair(pair(if(flip(1/2),x,∅),‘n’),‘v’),pair(
    sample(if(flip(11/24), Σ,{∅})),‘n’)),‘v’)})∅)),append(sample(Σ),x)),
        pair(F0(if(not(flip(7/24)), x,pair(pair(∅,‘n’),‘p’))),‘n’)).
```

with posterior and likelihood probability respectively of -7528.14 and -7413.76, and precision and recall value of 0.235294 and 0.25.

We can observe that the hypothesis can be simplified as in the tree below, where as before $g(x)$ indicates the part of the hypothesis that, having no recursion, generates a finite number of strings.



Notice that in order to go to g(x), and hence finish the sentence, it is necessary for x to be empty, although it is not sufficient. We can then represent the hypothesis as an HMM:



The hypothesis we found is then regular. We remark that this grammar is different from that of simplified English, it contains many sentences that could never be generated by simplified English, however we can see that in both cases the best guess Fleet finds is a regular grammar.

Now we want to add back the adverbs we removed earlier, increasing the number of terminals to see how Fleet infers the grammar even with this added element, which adds complexity to the grammar. We obtain after running for 9 hours:

```
F(0,x):=λx.append(sample(if(flip(1/2),{append(pair(pair(if(flip(5/24),
sample(Σ),x),‘n’),‘v’),if(flip(1/6),pair(∅,‘v’),if(flip(1/4),sample(Σ),x)))},
({sample(Σ)}∪{∅}))),if(flip(11/24),pair(if(flip(1/6),pair(pair(∅,‘p’),‘d’),
sample(Σ)),‘n’),sample(Σ)))
```

and a posterior and likelihood probability of -8742.3 and -8640.56, and precision and recall value of 0.207143 and 0.22. We can see that even with this added element the inferred grammar is finite, hence regular. The reason why we get a finite grammar may be because by adding adverbs we add complexity and Fleet cannot find a recurring pattern.

In the end, we cannot deduce with certainty whether the grammar of the data we possess belongs to the regular class or not. Although Fleet probably lacks sufficient data, time and hardware resources to work with, we can observe that Fleet's best guess under these conditions is in our case a regular grammar.

5.4 Campbell monkey calls

We now want to study other real data. Sumir Keenan collected 224 sequences of monkey calls consisting on 7 terminals. We would like to study this grammar as we did before. We start by looking for the best HMM that approximates them. In this analysis we have to remark that we are not actually considering the data in their original form: as we notice that one terminal appears only in the two first elements of the string in almost everywhere we decide to ignore the first two characters in every string, allowing to reduce the number of terminals to 6.

We find that the best HMM is one with 5 hidden states with BIC of value 3590.91 and logLikelihood of -1579.31.

The best HMM has transition and emission matrices respectively of:

$$\mathbf{P} = \begin{pmatrix} 0.4993 & 0.3708 & 0.0562 & 0.0267 & 0.0470 \\ 0 & 0.9857 & 0 & 0 & 0.0143 \\ 0 & 0 & 0.8950 & 0.1050 & 0 \\ 0 & 0.0421 & 0.0333 & 0.9247 & 0 \\ 0 & 0.0444 & 0 & 0 & 0.9556 \end{pmatrix}$$

$$\mathbf{E} = \begin{pmatrix} 0 & 0 & 0.6500 & 0 & 0.0082 & 0.0106 & 0.3312 \\ 0 & 0 & 0 & 0.0080 & 0.8889 & 0 & 0.1031 \\ 0.0058 & 0.7153 & 0.0729 & 0 & 0.0087 & 0.1625 & 0.0347 \\ 0 & 0.0405 & 0.0575 & 0 & 0.4984 & 0.4036 & 0 \\ 0 & 0 & 0 & 0.9327 & 0.0673 & 0 & 0 \end{pmatrix}$$

From the transition matrix we can see that the hidden chain is hardly connected and the most likely transitions are loops to the same node. We might think then that it is possible that the data we have are generated by distinct grammars.

We study now our grammar through Fleet. The best hypothesis we obtain is

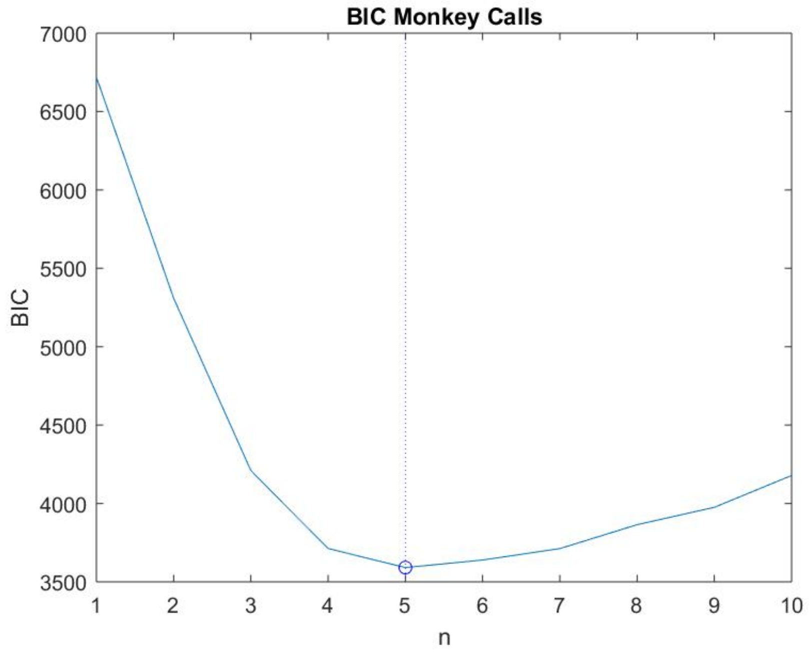
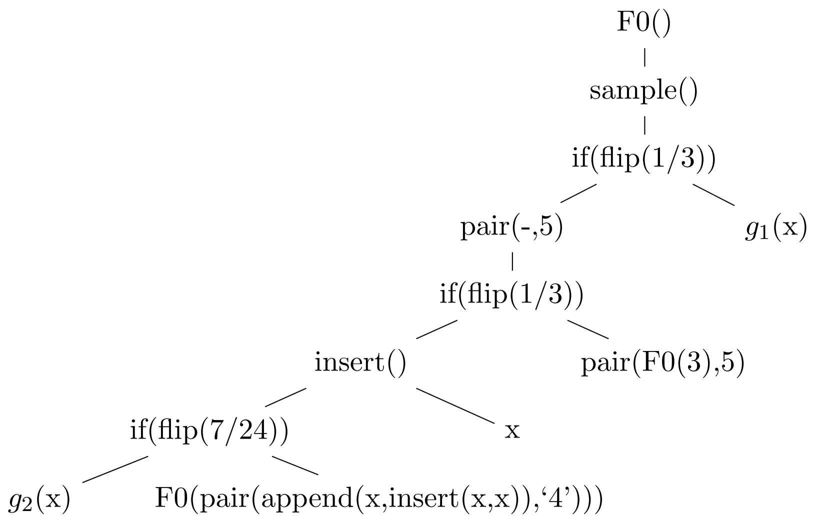


Figure 5.6: Bayesian Information Criterion applied to Monkey calls data, the lowest BIC value is reached in $n = 5$.

```
F(0,x):=λx.sample(if(not(flip(1/3)), {pair(if(flip(1/3),insert(if(flip(7/24),
append(if(flip(11/24), x, ∅) ,x), F0( pair(append (x, insert(x,x)),‘4’))),x),
pair (F0(pair(∅, ‘3’)),‘5’)),‘5’)}, if(flip(1/3), {insert(insert(head(pair(x,
              ‘2’)), insert(head(pair(x, ‘2’)), x)), x)},{∅})))
```

with posterior and likelihood probabilities respectively of -8612.82 and -8491.28, and precision and recall value of 0.377358 and 0.4.

The likelihood obtained with Fleet remains lower than with the representation at HMM, however we observe how the recall value is not very high in this case either, so we cannot consider this model as correct, however we can still analyze this hypothesis, to see, in these resource constraints, to which class the hypothesis found belongs. Let us visualize the hypothesis as a tree:



As usual the functions $g_1(x)$ and $g_2(x)$ represent a part of the hypothesis where there is no recursion. We can see that the hypothesis we obtain is of a non regular grammar as it is not possible to represent it with an HMM. In fact we can see that every time there is recursion in the sentence a 5 is added to the end, furthermore when the recursion `insert(F0(pair(append(x, insert(x,x)),'4'),x)` is called the internal part of the sentence is modified, therefore since there is no way for an HMM to store the number of times there has been recursion the grammar is non-regular.

Unfortunately, this characterization is not entirely true for all strings in the original data, however, having a recall value of 0.4 , we have that this is true for 40% of the sentences in our dataset. So from the observation resulting from the analysis as an HMM it is possible that this hypothesis found characterizes the grammar of some of the data we have. We can therefore think about the possibility that the grammar underlying the Campbell monkey call is non regular.

Chapter 6

Conclusions

In this thesis, we studied methods for deducing grammars, i.e. finding the best HMM that approximates the data through the Baum-Welch algorithm and through the Fleet library, trying to find the program that best generates the input data we receive. We obtained that Baum-Welch's algorithm gives a good approximation of HMMs for regular grammars, despite not being implemented to with a start and end state for the sentence. In fact, we noticed that for example in the simplified English grammar the HMM found reproduced the rules of the grammar well except for the requirement of the presence of at least one 'n' terminal and one 'v' terminal in each sentence. This was not respected because the sentence could end before passing through certain states. With slight modifications to the HMM, in particular by appropriately adding a start and an end state, along with some other states, we found an HMM that reflects the rules of the grammar well, thus proving that the grammar is regular. The Baum-Welch algorithm is therefore a good starting point to check whether a grammar is regular or not.

The Fleet library is a potentially very powerful tool when used with the appropriate technological resources. However, it is not always very easy to analyze the hypotheses we find. The easiest way to visualize the hypotheses we have found is to represent it as a tree, but even then we have found a few times complex hypotheses that are very difficult to analyze but not impossible.

In our analyses on simulated data it appears that Fleet deduces the correct class grammar, although it does not deduce exactly the correct grammar. We have not had a chance to perform this test on a lot of data so we cannot be sure that this always happens but we can think that analyzing data with Fleet might be a good starting point for hypothesizing what class the grammar belongs to.

By retrieving more data and the right tools, as well as time, it would probably be possible to reach more accurate results that can give more certainty about which class a grammar belongs. Ideally, in the case in which we obtain that Fleet is able to obtain almost perfectly accurately the grammar that generates the input data, we could classify the grammar by comparing the value of the likelihood obtained with Fleet with that obtained with the Baum-Welch algorithm: in the case in which the former is quite better we would be led to think that the grammar is non-regular, without needing to analyze the hypothesis obtained. In fact, if it were regular, the hypothesis obtained by Fleet would be regular and therefore representable with an HMM, consequently we would have obtained a result similar to that obtained with Baum-Welch. If this did not happen, it means that the grammar that best approximates it is non-regular. While if the two likelihood have similar value then we would classify the grammar as regular.

Then having good result in Fleet would lead to a simple way to classify grammar into regular and non regular by simply comparing the likelihood result with the Baum-Welch algorithm's output. Then an interesting research could be to improve the search of hypotheses in Fleet to reduce the computational time, as we assumed that one of the reason why we did not have accurate result was because we needed to search for a longer time. It would be also interesting to add more primitives to Fleet to reach the optimal BIC value, which we have seen for the Harry Potter data to be better using all the currently defined primitives.

Bibliography

[1] Y. Yang, S. T. Piantadosi, *One model for the learning of language.* Proceedings of the National Academy of Sciences (2022).

[2] R. J. Ryder, *Confidence in Bayesian Computational Statistics and Applications to Linguistics*, HDR thesis, University Paris-Dauphine (2022).

[3] Geuvers, H., Rot, J., *Applications, Chomsky hierarchy, and Recap*, Lecture notes, Radboud University Nijmegen (2017).

[4] D. Jurafsky, J. H. Martin, *Speech and Language Processing*, Chapter A (2021).

[5] L. R. Welch, *The Shannon Lecture: Hidden Markov Models and the Baum-Welch Algorithm* , IEEE Information Theory Society Newsletter, Vol.53, No.4 (2023).

[6] P. Brémaud, *Markov Chains, Gibbs Fields, Monte Carlo Simulation, and Queues*, Springer (2021)

[7] M. Sipser .*Introduction to the Theory of Computation*, PWS Publishing.(1997)

[8] N. D. Goodman, J. B. Tenenbaum, J. Feldman, T. L. Griffiths, *A rational analysis of rule-based concept learning*, Cogn. Sci. 32, 108-154 (2008).

[9] W. D. Vousden, W. M. Farr, I. Mandel, *Dynamic temperature selection for parallel-tempering in Markov Chain Monte Carlo simulation.* Mon. Not. R. Astron. Soc. 455, 1919-1937 (2015)

[10] D.J. Earl, M. W. Deem, *Parallel tempering: Theory, applications, and new perspectives*, Physical Chemistry Chemical Physics, Issue 23 (2005).

[11] Y. Sugita, Y. Okamoto, *Replica-exchange molecular dynamics method for protein folding*, Chemical Physics Letters, Issue 1-2 (1999)

[12] D. A. Kofke, *On the acceptance probability of replica-exchange Monte Carlo trials*, The Journal of Chemical Physics, Volume 117, Issue 15 (2002).

[13] D. A. Kofke, *Erratum: "On the acceptance probability of replica-exchange Monte Carlo trials" [J. Chem. Phys. 117, 6911 (2002)]*, The Journal of Chemical Physics, Volume 120, Issue 22 (2004)

[14] G. O. Roberts, J. S. Rosenthal, *Coupling and Ergodicity of Adaptive Markov Chain Monte Carlo Algorithms*, Journal of Applied Probability, Volume 44, Issue 2 (2007)