



Universidade de São Paulo

Dissertação de Mestrado

UM ETIQUETADOR MORFO-SINTÁTICO  
BASEADO EM CADEIAS DE MARKOV DE  
TAMANHO VARIÁVEL

---

Fábio Natanael Kepler

Programa de Pós-Graduação em Ciência da Computação

São Paulo, SP, Brasil

2005

UM ETIQUETADOR MORFO-SINTÁTICO  
BASEADO EM CADEIAS DE MARKOV DE  
TAMANHO VARIÁVEL

---

**Fábio Natanael Kepler**

Dissertação apresentada  
ao  
Instituto de Matemática e Estatística  
da  
Universidade de São Paulo  
para  
obtenção do grau  
de  
**Mestre em Ciência da Computação.**

Área de Concentração: **Ciência da Computação**  
Orientador: **Prof. Dr. Marcelo Finger**

São Paulo, SP, Brasil

2005

**UM ETIQUETADOR MORFO-SINTÁTICO  
BASEADO EM CADEIAS DE MARKOV DE  
TAMANHO VARIÁVEL**

Este exemplar corresponde à redação  
final da dissertação devidamente  
corrigida e defendida por  
**Fábio Natanael Kepler**  
e aprovada pela comissão julgadora.

São Paulo, abril de 2005.

**COMISSÃO EXAMINADORA:**

- **Prof. Dr. Marcelo Finger** (Orientador) - IME-USP
- **Profa. Dra. Cláudia Monteiro Peixoto** - IME-USP
- **Profa. Dra. Maria das Graças Volpe Nunes** - ICMC-USP

It was Winston Churchill who said,  
*'Out of intense complexities, intense  
simplicities emerge'*.

(Lex Luthor, in *Smallville*)

*Dedicado às pessoas da próxima página*

# Agradecimentos

Não porque é bonito ou emocionalmente comovente, mas porque realmente sou grato, por todo amor e incrível graça que Deus demonstra na minha vida. Obrigado, Senhor.

Com a família que tenho, não tem como não ser agradecido. Obrigado pai, mãe, Diogo e Marcos. Tirando todo enfraquecimento que essa expressão sofreu, pai e mãe, vocês são, no sentido literal, os melhores pais do mundo. E Diogo e Marcos, mesmo que eu não chame vocês de “manos” aqui, tô pra ver três irmãos como nós. ;-)

Minha namorada, Michelle. Obrigado pela companhia maravilhosa que você é pra mim. Dos dois anos que já estou em São Paulo, estou há mais tempo namorando você do que o tempo que fiquei sem você. E isto não atrasou este trabalho. ;-) Só motivou.

Marcelo, meu orientador. Obrigado por sempre me dar atenção. Com ela vinham força e motivação pra fazer um trabalho cada vez melhor. E muito obrigado pela dedicação e interesse, em mim e nesse trabalho. É por isso que uso a primeira pessoa do plural em todo texto. ;-)

Tenho que agradecer a todas pessoas que me receberam e fizeram com que eu me sentisse em casa em São Paulo. Obrigado a todos meus pais adotivos da igreja alemã, pelos abraços, almoços, e roupas lavadas. Obrigado à galera que me recebeu, me tornou amigo e me convidou pros jogos de futebol no sábado. Obrigado aos meus colegas-amigos da USP, que também vieram de longe, e iam juntos almoçar no bandejão. E obrigado a todo pessoal, que está quase todo lá no sul, que continuou sendo meu amigo mesmo estando longe.

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Objetivo . . . . .	1
1.2	Estrutura . . . . .	2
<b>2</b>	<b>Análise Morfo-sintática</b>	<b>3</b>
2.1	O Problema da Etiquetagem . . . . .	3
2.2	Métodos e Modelos . . . . .	4
<b>3</b>	<b>Fundamentação Teórica</b>	<b>5</b>
3.1	Noções Básicas . . . . .	5
3.2	Cadeias de Markov . . . . .	6
<b>4</b>	<b>Modelos de Markov Ocultos</b>	<b>9</b>
4.1	Encontrando a Probabilidade de uma Observação . . . . .	10
4.1.1	Algoritmo Progressivo . . . . .	11
4.1.2	Algoritmo Regressivo . . . . .	11
4.2	Encontrando a Melhor Sequência de Estados . . . . .	12
4.2.1	Algoritmo de Viterbi . . . . .	12
4.3	Estimando os Parâmetros . . . . .	13
<b>5</b>	<b>Cadeias de Markov de Tamanho Variável</b>	<b>15</b>
5.1	Representação do Espaço de Estados . . . . .	16
5.2	Algoritmo de Contexto . . . . .	16
<b>6</b>	<b>Etiquetadores Morfo-Sintáticos</b>	<b>20</b>
6.1	Modelos Internos . . . . .	20
6.1.1	O Etiquetador de Ordem Dois (HMM) . . . . .	20
6.1.1.1	Representação dos Estados . . . . .	21
6.1.1.2	Tratamento de Palavras Desconhecidas . . . . .	22
6.1.2	O Etiquetador de Tamanho Variável (VLMC) . . . . .	23
6.1.2.1	Valor de Corte da Árvore de Contexto . . . . .	23
6.2	Arquiteturas dos Etiquetadores . . . . .	24
6.2.1	Arquitetura Básica . . . . .	24
6.2.2	Arquitetura do <b>HMM Tagger</b> . . . . .	26
6.2.3	Arquitetura do <b>VLMC Tagger</b> . . . . .	26
6.3	Implementação dos Etiquetadores . . . . .	26
6.3.1	Classe Automaton . . . . .	27
6.3.2	Classe AlgoViterbi . . . . .	28
6.3.3	Classe AffixTree . . . . .	29

6.3.4	Classe AlphaBeta . . . . .	29
6.3.5	Classe AlgoTraining . . . . .	29
6.3.6	Classe ContextTree . . . . .	30
<b>7</b>	<b>Testes e Resultados</b>	<b>32</b>
7.1	Testes . . . . .	32
7.2	Resultados . . . . .	33
7.2.1	Precisão . . . . .	33
7.2.2	Tempo . . . . .	38
7.2.3	Outras Medidas . . . . .	41
7.2.4	Resultados Finais e Comparações . . . . .	45
7.2.5	Outros Experimentos . . . . .	45
<b>8</b>	<b>Conclusões</b>	<b>47</b>
	<b>Referências</b>	<b>49</b>
<b>A</b>	<b>O Sistema de Anotação do Córpus Tycho Brahe</b>	<b>52</b>
<b>B</b>	<b>Recursos do Etiquetador VLMC Tagger</b>	<b>56</b>

# Lista de Tabelas

3.1	Notação utilizada para etiquetagem . . . . .	5
7.1	Média de acertos e desvio padrão dos testes com o <b>VLMC Tagger</b> . . . . .	35
7.2	Média de acertos e desvio padrão dos testes com o <b>HMM Tagger</b> . . . . .	37
7.3	Palavras desconhecidas nos testes com o <b>VLMC Tagger</b> . . . . .	38
7.4	Taxa de acertos e tempo de execução dos dois etiquetadores. . . . .	45
7.5	Precisão e tempos de aprendizado e etiquetagem dos etiquetadores VLMC Tagger e MXPOST. . . . .	45
A.1	Etiquetas do cópuz Tycho Brahe. . . . .	52
B.1	Parâmetros do <b>VLMC Tagger</b> . . . . .	58

# Lista de Figuras

3.1	Um modelo de Markov. . . . .	7
3.2	Um modelo de Markov de ordem 2. . . . .	8
5.1	Definição de uma <b>Árvore de Contexto</b> . . . . .	17
5.2	Exemplificação de uma árvore de contexto. . . . .	18
6.1	Modelo interno do etiquetador baseado em HMM. . . . .	22
6.2	Arquitetura do <b>HMM Tagger</b> . . . . .	25
6.3	Arquitetura do <b>VLMC Tagger</b> . . . . .	25
6.4	Classes implementadas do <b>HMM Tagger</b> . . . . .	27
6.5	Classes implementadas do <b>VLMC Tagger</b> . . . . .	28
6.6	Exemplo de uma <b>Árvore de Contexto</b> . . . . .	31
7.1	Eficiência do <b>HMM Tagger</b> e do <b>VLMC Tagger</b> . . . . .	34
7.2	Distribuição das taxas de acerto dos testes com o <b>VLMC Tagger</b> . . . . .	35
7.3	Distribuição das taxas de acerto dos testes com o <b>HMM Tagger</b> . . . . .	36
7.4	Média das taxas de acerto do <b>VLMC Tagger</b> para palavras conhecidas e desconhecidas. . . . .	37
7.5	Média das taxas de acerto do <b>HMM Tagger</b> para palavras conhecidas e desconhecidas. . . . .	39
7.6	Média do tempo de execução do <b>HMM Tagger</b> e do <b>VLMC Tagger</b> . . . . .	39
7.7	Distribuição dos tempos de execução dos testes com o <b>VLMC Tagger</b> . . . . .	40
7.8	Distribuição dos tempos de execução dos testes com o <b>HMM Tagger</b> . . . . .	41
7.9	Distribuição do número de etiquetas reconhecidas pelo <b>VLMC Tagger</b> . . . . .	42
7.10	Distribuição do número de estados de duas etiquetados obtidos pelo <b>HMM Tagger</b> . . . . .	43
7.11	Crescimento do número de galhos da árvore de contexto do <b>VLMC Tagger</b> . . . . .	43
7.12	Crescimento do número de nós da árvore de contexto do <b>VLMC Tagger</b> . . . . .	44
7.13	Proporção de galhos da árvore de contexto do <b>VLMC Tagger</b> com tamanhos diferentes. . . . .	44

## RESUMO

Dissertação de Mestrado  
Ciência da Computação  
Universidade de São Paulo

# UM ETIQUETADOR MORFO-SINTÁTICO BASEADO EM CADEIAS DE MARKOV DE TAMANHO VARIÁVEL

AUTOR: FÁBIO NATANAEL KEPLER

ORIENTADOR: PROF. DR. MARCELO FINGER

Local e data da defesa: São Paulo, abril de 2005.

Dado um texto, queremos atribuir a cada palavra, de acordo com seu contexto, uma categoria morfo-sintática. Para isto, implementamos dois etiquetadores morfo-sintáticos baseados em cadeias de Markov. Primeiro, utilizando uma abordagem bastante conhecida, construímos um etiquetador que usa cadeias de Markov de ordem fixa igual a dois. Então, propomos e implementamos outro etiquetador utilizando uma abordagem recente, baseada em cadeias de Markov de tamanho variável. Depois de mostrar a teoria estatística dos dois modelos e os problemas e desafios mais comuns a serem resolvidos, explicamos o funcionamento dos etiquetadores e expomos os resultados obtidos. Com estes resultados, obtemos uma comparação mais precisa da eficiência destes dois modelos aplicados à etiquetagem morfo-sintática, identificando pontos fortes e fracos de cada um. Por uma combinação de fatores, consideramos o etiquetador com cadeias de Markov de tamanho variável melhor do que o de ordem fixa, e alcançamos um dos melhores resultados em etiquetagem morfo-sintática do português atualmente: 95,51% de precisão, obtida em um tempo total de execução, incluindo o aprendizado e etiquetagem de mais de um milhão de palavras, de menos de três minutos. Contribuímos, assim, com estado da arte da área, e além disto, fornecemos resultados que nos permitem observar limitações e vantagens da aplicação de modelos estatísticos, em geral, ao problema focado, que podem ajudar a comunidade a identificar pontos críticos sobre os quais as pesquisas nesta área deverão procurar se concentrar.

**Palavras-chave:** análise morfo-sintática, processamento de linguagem natural, lingüística computacional, etiquetador morfo-sintático estatístico, cadeias de Markov, cadeias de Markov de tamanho variável.

## ABSTRACT

Dissertação de Mestrado  
Ciência da Computação  
Universidade de São Paulo

# UM ETIQUETADOR MORFO-SINTÁTICO BASEADO EM CADEIAS DE MARKOV DE TAMANHO VARIÁVEL

(A Part-of-Speech Tagger based on Variable Length Markov Chains)

AUTOR: FÁBIO NATANAEL KEPLER

ORIENTADOR: PROF. DR. MARCELO FINGER

Local e data da defesa: São Paulo, abril de 2005.

The problem of Part-of-Speech (POS) tagging consists in assigning to each word in context a POS tag. To solve this problem, we implement two POS-taggers based on Markov chains. First, using a well-known approach, we build a tagger that uses fixed Markov chains of order 2. Then, we propose and implement another tagger using a more recent approach, based on variable length Markov chains. After showing the statistical theory of the two models and the common problems and challenges to be solved, we describe the taggers' functionality and show the obtained results. With these results, we get a more precise comparison of the efficiency of these models applied to the POS-tagging, identifying strong and weak points of each one. By a combination of factors, we consider the tagger based on variable length Markov chains better than the one with fixed order, and we reach one of the best results in POS tagging of portuguese nowadays: a precision of 95.51%, obtained in a total time of execution, including the learning and tagging of more than a million words, of less than three minutes. We contribute, thus, with the state-of-the art of the area, and moreover, we show results that give us the possibility of seeing limitations and advantages of the application of statistical models in general to the focused problem, which can help the community in identifying critic points over which the researches in this area should be concentrated.

**Keywords:** part-of-speech analysis, natural language processing, computational linguistics, statistical part-of-speech tagger, Markov chains, variable-length Markov chains.

## Introdução

A Linguística procura caracterizar e explicar os fenômenos lingüísticos que nos cercam. Uma parte desta tarefa está relacionada ao lado cognitivo de como os humanos adquirem, produzem e entendem a linguagem; outra parte está relacionada a entender a relação entre expressões lingüísticas e o mundo; e uma terceira parte está relacionada a entender as estruturas lingüísticas pelas quais a linguagem realiza a comunicação. O tratamento destas questões através do computador é chamado de Linguística Computacional, e estas questões estão relacionadas ao Processamento de Linguagem Natural (PLN).

Para procurar realizar esta última tarefa, ou seja, entender as estruturas lingüísticas que a linguagem possui, muitos propuseram criar um conjunto de regras que descrevesse a relação e o significado das palavras em uma linguagem. Esta abordagem tem um longo histórico que se estende a pelo menos 2000 anos atrás. Entretanto, ainda que atualmente seja possível adquirir, armazenar e consultar uma grande quantidade de dados, a caracterização exata e completa de uma linguagem natural tem se mostrado impossível. As pessoas estão sempre criando e adaptando essas regras de acordo com suas necessidades de comunicação.

Desse modo, nosso trabalho explora uma abordagem que trata da questão de outra forma: buscando padrões comuns que ocorrem no uso da linguagem. Para identificar estes padrões usamos a teoria probabilística, que aplicamos sobre conjuntos de textos, chamados *córpus*<sup>1</sup>. Esta abordagem empírica aplicada ao PLN sugere que podemos aprender a complicada e extensiva estrutura da linguagem especificando um modelo geral apropriado, e então, induzindo os valores dos parâmetros aplicando métodos estatísticos, de reconhecimento de padrões, e de aprendizado de máquina, a uma grande quantidade de usos da linguagem.

Nosso trabalho concentra-se na tarefa intermediária de classificar as palavras de um texto em categorias gramaticais dentro dos seus contextos. Esta tarefa é chamada de *análise morfo-sintática*, e uma vez que a tenhamos cumprido, possivelmente poderemos alavancar o desenvolvimento das tarefas dos níveis mais altos de processamento de linguagem, como as análises sintática, semântica e pragmática.

### 1.1 Objetivo

Nosso objetivo é comparar lado a lado a eficiência de dois modelos estatísticos aplicados ao problema da etiquetagem morfo-sintática. Implementando paralelamente os dois modelos, queremos realçar a contribuição da parte teórica de cada modelo na eficiência da solução do problema.

---

<sup>1</sup>Um conjunto de textos é chamado de *corpus* (do Latim), e quando temos várias dessas coleções de textos temos *corpora*. Para soar melhor no português, vamos utilizar o termo *córpus* para qualquer um dos casos.

---

Com esta comparação mais precisa, também esperamos enxergar limitações e vantagens da aplicação de quaisquer modelos estatísticos a este problema.

## 1.2 Estrutura

Esta dissertação está organizada da seguinte maneira: no Capítulo 2 explicamos o que é a análise morfo-sintática e o problema da etiquetagem, e mostramos quais são as principais abordagens existentes e quais delas iremos utilizar no nosso trabalho; em função desta abordagem escolhida, no Capítulo 3 damos algumas noções básicas da teoria empregada, e mostramos em quais modelos específicos iremos nos concentrar. Nos Capítulos 4 e 5 detalhamos os aspectos teóricos dos dois modelos em questão, o de cadeias de Markov de ordem fixa, no Capítulo 4, e o de cadeias de Markov de tamanho variável, no 5. Depois, no Capítulo 6, explicamos a construção de dois etiquetadores baseados nestes modelos, e mostramos suas arquiteturas e detalhes das implementações; então, no Capítulo 7, apresentamos os testes feitos com estes etiquetadores e discutimos os resultados obtidos; e, finalmente, colocamos no Capítulo 8 as conclusões e experiências obtidas com este trabalho.

# Análise Morfo-sintática

Os lingüistas agrupam as palavras de uma linguagem em classes que mostram comportamento sintático semelhante. Estas classes de palavras também são chamadas de categorias morfo-sintáticas ou gramaticais. A análise morfo-sintática consiste em classificar gramaticalmente cada palavra de uma frase dentro do seu contexto. Isso é útil para outros estudos da lingüística, como tradução automática e extração de informações de textos [19].

## 2.1 O Problema da Etiquetagem

O problema a ser resolvido não é trivial, pois muitas palavras possuem várias categorias gramaticais. Para estas palavras, dadas fora de contexto, existe *ambigüidade* em relação a como elas devem ser classificadas. A tarefa de tirar a ambigüidade é determinar qual função sintática de uma palavra ambígua é invocada em um uso particular dela. Isto é feito olhando-se para o contexto desta palavra.

O problema da ambigüidade é de clara importância em muitas aplicações de PLN. Sempre que as ações de um sistema dependam do significado do texto sendo processado, tirar a ambigüidade é necessário. Um programa de computador que procura resolver este problema é chamado de etiquetador morfo-sintático. Seu papel é associar a cada palavra de um texto uma única etiqueta que expresse sua classificação gramatical no contexto das outras palavras. Por exemplo, considere a sentença

“Eu canto na escola.”.

A palavra *canto* é ambígua, podendo ser etiquetada como substantivo (de “o *canto da mesa*”), ou como verbo (conjugação de *cantar*). Entretanto, na sentença acima, o contexto permite verificarmos que é o segundo caso que deve ser aplicado. Assim, teríamos a seguinte seqüência de etiquetas:

“Eu/PRO canto/VB na/P+D escola/N ./.”,

onde PRO representa a classe gramatical dos pronomes, VB dos verbos, P+D das preposições com determinantes, N dos nomes (substantivos), e . dos símbolos de pontuação final.

Entretanto, as questões inerentes a esta tarefa são complexas e desafiadoras, não se conhecendo nenhum método analítico para resolvê-la de forma exata. Assim, várias abordagens teóricas e práticas têm sido pesquisadas pela comunidade de PLN.

## 2.2 Métodos e Modelos

A construção de um etiquetador envolve a escolha de um modelo teórico de etiquetagem e a escolha do método usado pelos algoritmos para alimentar o modelo.

Primeiro, podemos classificar os algoritmos quanto ao método de aprendizado usado: supervisionado ou não-supervisionado. No aprendizado supervisionado sabemos o estado atual para cada dado sobre o qual treinamos nosso sistema. Já no aprendizado não-supervisionado não sabemos a classificação dos dados na amostra de treinamento. Assim, o aprendizado não-supervisionado pode geralmente ser visto como uma tarefa de aglomeração (agrupamento), enquanto o supervisionado pode geralmente ser visto como uma tarefa de classificação. Baseados nas informações agrupadas ou classificadas, pode-se calcular informações probabilísticas ou extrair um conjunto de regras, o que irá depender do modelo usado.

Segundo, os modelos usados em etiquetadores podem geralmente ser classificados em quatro grupos:

- Neurais;
- Simbólicos;
- Estatísticos;
- Híbridos.

Os modelos neurais utilizam Redes Neurais [21], que procuram imitar o comportamento de uma rede de neurônios, onde, simplificada, cada neurônio realiza um processamento específico e transmite seu resultado adiante para outros neurônios. Entretanto, dependendo do problema a ser resolvido, são lentas e exigem uma grande quantidade de dados.

Os modelos simbólicos englobam as técnicas que usam lógicas, árvores de decisão e regras de transformação. São chamados de simbólicos por se basearem na manipulação de estruturas simbólicas de conceitos. Por esta razão também costumam ser mais lentos que modelos numéricos. Em particular, os baseados em regras de transformação analisam os erros de etiquetagem e procuram induzir regras que os transformem em acertos. Dentre estes, o mais conhecido atualmente é o etiquetador de Brill [6], que obtém uma precisão em torno de 95,4% no cópuz em inglês Wall Street Journal [20]. Para o português, Chacur e Finger [2] propuseram e implementaram uma variante do método de Brill, e obtiveram resultados razoáveis. Finger [15] depois utilizou algumas técnicas de otimização e obteve resultados melhores, em torno de 95,43% para o cópuz Tycho Brahe [17].

Os modelos estatísticos baseiam-se em resultados de probabilidade, estatística e teoria da informação, buscando identificar padrões no uso da linguagem. Dentre eles, destacam-se os modelos baseados em Cadeias de Markov Ocultas (conhecidos na literatura por HMMs, do inglês *Hidden Markov Models*), os baseados no método da Máxima Verossimilhança, e os baseados no método da Máxima Entropia. Os dois últimos são bastante parecidos, e são descritos e utilizados por Alves [13] na etiquetagem do português. Os HMMs são descritos no próximo capítulo. Para a etiquetagem do inglês, Ratnaparkhi [24] implementa um etiquetador baseado na Máxima Entropia, e obtém precisão de 96,6% no cópuz Wall Street Journal. Já Brants [5] implementa um etiquetador baseado em HMM's, e obtém uma precisão de 96,7%. Recentemente, Toutanova [30] apresentou um etiquetador baseado em Redes de Dependência Cíclica, que obtém 97,24% de precisão, e afirma que é o estado-da-arte para o inglês. Para o português, além do etiquetador citado no parágrafo anterior, Aires [1] adapta diversos etiquetadores do inglês e mostra seus resultados para o português. O melhor deles obtém 90,25% de precisão, e é o obtido pela adaptação do etiquetador baseado na Máxima Entropia de Ratnaparkhi [24].

Por fim, os modelos híbridos apresentam uma mistura dos modelos acima, normalmente para suprir as deficiências de um com as qualidades de outro.

## Fundamentação Teórica

Como dissemos na introdução, vamos usar modelos estatísticos de etiquetagem [9, 19]. Mais especificamente, os modelos baseados em cadeias de Markov. Além disso, estamos particularmente interessados no método de aprendizado supervisionado. Por estas razões, estaremos usando um *córpus* pré-etiquetado chamado *Tycho Brahe* [17], mantido pelo IME-USP. O conjunto de etiquetas utilizadas no *córpus* Tycho Brahe está listado no Apêndice A.

Nas seções seguintes daremos algumas noções de Teoria da Probabilidade e explicaremos a teoria básica das cadeias de Markov. Para isto, quando relevante, usaremos uma notação normalmente utilizada em textos sobre etiquetagem estatística, mostrada na Tabela 3.1. O conjunto

$w_i$	a palavra na posição $i$ no <i>córpus</i> ou na sentença
$l_i$	a etiqueta de $w_i$
$w_{i,i+m}$	a seqüência de palavras de $w_i$ até $w_{i+m}$
$l_{i,i+m}$	as etiquetas de $w_{i,i+m}$
$w^k$	a $k$ -ésima ocorrência de palavra no léxico
$l^j$	a $j$ -ésima etiqueta no conjunto de etiquetas
$n$	tamanho da sentença
$T$	tamanho da seqüência de estados
$L$	cardinal do conjunto de etiquetas
$N$	número de estados no espaço de estados

**Tabela 3.1:** Notação utilizada para etiquetagem

formado pelas palavras (obtidas do *córpus*) e por suas categorias morfo-sintáticas é chamado de *léxico*. Quando nos referirmos a “palavras de um *córpus*”, estaremos nos referindo a todos os símbolos léxicos deste *córpus*, como palavras, números e sinais de pontuação, por exemplo.

### 3.1 Noções Básicas

Quando queremos prever ou medir as chances que algo possui de acontecer, utilizamos a *Teoria da Probabilidade*. Podemos, além disto, condicionar estas chances a algo que já aconteceu:  $P(Y|X)$  representa a probabilidade de  $Y$  ocorrer dado que  $X$  ocorreu. Deste modo, no contexto da etiquetagem,  $P(l_i|w_i)$  é a probabilidade da palavra  $w_i$  ter a etiqueta  $l_i$ , por exemplo.

Para uma introdução sobre probabilidade e estatística veja [32, 27].

## 3.2 Cadeias de Markov

Cadeias de Markov [32, 27, 26] foram primeiramente desenvolvidas por Andrei A. Markov para modelar as seqüências de letras em trabalhos de literatura Russa, e desde então se desenvolveram como uma ferramenta estatística geral.

Suponha  $X = (X_1, \dots, X_T)$  uma seqüência de variáveis aleatórias com valores num conjunto finito  $S = (s_1, \dots, s_N)$ , o espaço de estados. Então  $X$  é dita ser uma cadeia de Markov se tiver a seguinte propriedade de Markov:

### Horizonte Limitado

$$P(X_{t+1} = s_k | X_1, \dots, X_t) = P(X_{t+1} = s_k | X_t)$$

Podemos descrever uma cadeia de Markov por uma matriz de probabilidades de transição  $A$ :

$$a_{ij} = P(X_{t+1} = s_j | X_t = s_i)$$

Aqui,  $a_{ij} \geq 0, \forall i, j$  e  $\sum_{j=1}^N a_{ij} = 1, \forall i$ .

Além disso, temos que especificar as probabilidades de diferentes estados iniciais para a cadeia de Markov, dadas no vetor  $\Pi$ :

$$\pi_i = P(X_1 = s_i)$$

Aqui,  $\sum_{i=1}^N \pi_i = 1$ . Podemos evitar o uso deste vetor se especificarmos que a cadeia de Markov sempre começa em um certo estado especial,  $s_0$ , e então usarmos transições a partir deste estado contido na matriz  $A$  para especificar as probabilidades que tínhamos em  $\Pi$ .

Dadas as definições acima, podemos descrever um modelo de Markov como uma tripla  $\mu = (S, A, \Pi)$ , onde, como enunciado,  $S$  é o espaço de estados,  $A$ , a matriz de transição, e  $\Pi$ , o vetor de probabilidades iniciais.

Para fixar a idéia, considere um modelo de Markov de 3 estados do clima. Assumimos que uma vez por dia o tempo climático é observado como sendo um desses:

Estado 1: chuvoso

Estado 2: nublado

Estado 3: ensolarado

Postulamos que o clima no dia  $t$  é caracterizado por um único dos três estados acima, e que a matriz  $A$  de probabilidades de transição é dada por

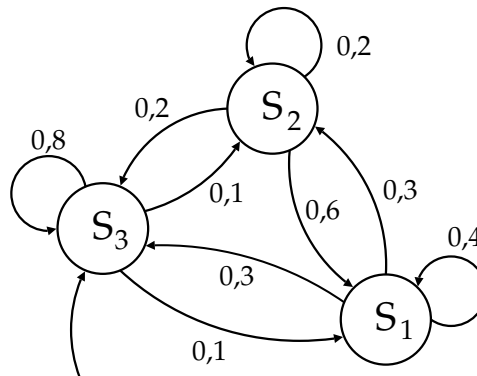
$$A = \{a_{ij}\} = \begin{bmatrix} 0,4 & 0,3 & 0,3 \\ 0,2 & 0,6 & 0,2 \\ 0,1 & 0,1 & 0,8 \end{bmatrix}$$

Dado que o tempo no dia 1 ( $t = 1$ ) está ensolarado (estado 3), podemos perguntar: Qual é a probabilidade (de acordo com o modelo) de que o tempo para os próximos 7 dias seja "ensolarado-ensolarado-chuvoso-chuvoso-ensolarado-nublado-ensolarado"? Definindo esta seqüência de estados como  $O$ ,  $O = \{S_3, S_3, S_3, S_1, S_1, S_3, S_2, S_3\}$ , queremos determinar a probabilidade de  $O$ , dado que

o estado inicial é  $S_3$ . Esta probabilidade pode ser expressa como

$$\begin{aligned}
 P(O|Modelo) &= P[S_3, S_3, S_3, S_1, S_1, S_3, S_2, S_3|Modelo] \\
 &= P[S_3] \cdot P[S_3|S_3] \cdot P[S_3|S_3] \cdot P[S_1|S_3] \cdot P[S_1|S_1] \cdot P[S_3|S_1] \cdot P[S_2|S_3] \cdot P[S_3|S_2] \\
 &= 1 \cdot a_{33} \cdot a_{33} \cdot a_{31} \cdot a_{11} \cdot a_{13} \cdot a_{32} \cdot a_{23} \\
 &= 1 \cdot (0,8)(0,8)(0,1)(0,4)(0,3)(0,1)(0,2) \\
 &= 1,535 \times 10^{-4}
 \end{aligned}$$

Podemos, também, representar uma cadeia de Markov por um diagrama de estados, como na Figura 3.1. Os estados são mostrados como círculos ao redor de seu nome, e o estado inicial



**Figura 3.1:** Um modelo de Markov.

é indicado por uma seta de entrada. As transições possíveis são mostradas por arcos dirigidos conectando estados, e estes arcos são rotulados com a probabilidade desta transição ser tomada. Transições com probabilidade zero são omitidas. As probabilidades dos arcos de saída de cada estado somam 1. A partir desta representação, podemos perceber que um modelo de Markov pode ser visto como um autômato de estados finitos não-determinístico com probabilidades em cada arco.

Na etiquetagem com modelos de Markov, olhamos para a seqüência de etiquetas em um texto como uma cadeia de Markov. Pela propriedade de Markov, assumimos que a etiqueta de uma palavra só depende da etiqueta anterior (horizonte limitado). Usando a notação da Tabela 3.1, podemos escrever a propriedade do Horizonte Limitado assim:

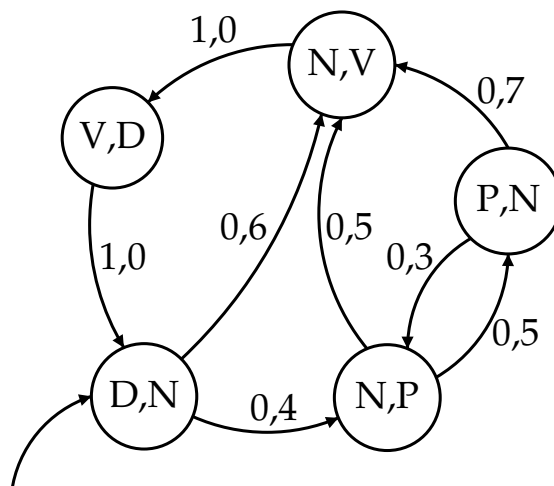
$$P(l_{i+1}|l_{1,i}) = P(l_{i+1}|l_i)$$

Apesar da propriedade do Horizonte Limitado, podemos considerar que um estado é composto por duas etiquetas. Assim estaremos olhando duas etiquetas para trás. Uma cadeia de Markov dessa forma é dita ser de ordem dois. Então, o que a propriedade do Horizonte Limitado quer dizer é que a ordem da cadeia é fixa e finita.

Por exemplo, considerando  $N, P, D, V$  como as etiquetas, o diagrama da Figura 3.2 representa uma cadeia de Markov de ordem 2.

Além do uso de modelos de Markov na etiquetagem morfo-sintática, eles também têm sido usados em PLN para modelar seqüências fonéticas válidas em reconhecimento de voz [18], e em biologia para modelar problemas como o do alinhamento de seqüências de DNA [7].

Em um Modelo de Markov Visível (VMM, *Visible Markov Model*), sabemos por quais estados o modelo está passando, e assim a seqüência de estados ou alguma função determinística dela



**Figura 3.2:** Um modelo de Markov de ordem 2.

pode ser considerada como a saída, e sua probabilidade pode ser facilmente calculada. Entretanto, este modelo é restritivo demais para que seja aplicável a vários problemas de interesse. No próximo capítulo estendemos o conceito de modelos de Markov para incluir o caso onde a saída não corresponde à seqüência de estados.

---

## Modelos de Markov Ocultos

Nos Modelos de Markov Ocultos (HMM, *Hidden Markov Models*), não sabemos qual é a seqüência de estados que o modelo percorre, mas apenas alguma função probabilística dela. Podemos pensar em eventos inferiores gerando probabilisticamente eventos de superfície, como é o caso da etiquetagem morfo-sintática. Para entender melhor, considere o exemplo seguinte.

*O Modelo das Bolas e Urnas:* Assuma que existem  $N$  grandes urnas de vidro em uma sala. Dentro de cada urna existe um grande número de bolas coloridas. Assuma que existam  $M$  cores distintas de bolas. O processo físico para se obter observações é o seguinte: um gênio está na sala, e de acordo com algum processo aleatório, ele escolhe uma urna inicial. Desta urna, uma bola é escolhida aleatoriamente, e sua cor é gravada como a observação. A bola então é recolocada na urna de onde ela foi escolhida. Uma nova urna é então selecionada de acordo com o processo de seleção aleatória associado com a urna atual, e o processo de seleção de bola é repetido. Este processo inteiro gera uma seqüência de observações finitas de cores, as quais gostaríamos de modelar como a saída observável de um HMM.

O HMM mais simples que corresponde ao processo da urna e da bola é aquele em que cada estado corresponde a uma urna específica, e para o qual uma probabilidade de cor (de bola) é definida para cada estado. A escolha das urnas é ditada pela matriz de transição de estados do HMM.

O exemplo acima nos dá uma boa idéia do que é um HMM e como ele pode ser aplicado a casos simples. Vamos agora definir formalmente alguns aspectos destes modelos. (Para uma descrição mais detalhada um bom tutorial sobre *HMMs* é o de Rabiner [23]).

Um HMM é definido como uma quintupla  $(S, K, \Pi, A, B)$ , onde  $S$  e  $K$  são o conjunto de estados e o alfabeto de saída, e  $\Pi$ ,  $A$  e  $B$  são as probabilidades para o estado inicial, transições entre estados, e emissões de símbolos, respectivamente. Desse modo,  $a_{ij} \in A$  denota a probabilidade de ir do estado  $s_i$  ao estado  $s_j$ , e  $b_{ijk} \in B$  a probabilidade de ir do estado  $s_i$  ao estado  $s_j$  emitindo o símbolo  $o_k$ .

Cada vez que o HMM deixa um estado, um símbolo é emitido. Um HMM deste tipo é chamado de arco-emissor. Em nosso contexto, os estados do modelo são formados por etiquetas, e palavras são emitidas toda vez que se deixa um estado.

Existem três questões fundamentais que queremos saber sobre um HMM:

1. Dado um modelo  $\mu = (A, B, \Pi)$ , como calculamos eficientemente a probabilidade de uma certa observação, isto é,  $P(O|\mu)$ ?
2. Dada a seqüência de observações  $O$  e um modelo  $\mu$ , como escolhemos uma seqüência de estados  $(X_1, \dots, X_{T+1})$  que melhor explica  $O$ ?

3. Dada uma seqüência de observações  $O$ , e um espaço de modelos possíveis encontrados pela variação dos parâmetros  $\mu = (A, B, \Pi)$ , como encontramos o modelo que melhor explica os dados observados?

A questão 1 nos permite dizer com que probabilidade um dado modelo gerou as observações, isto é, quão bem este modelo explica as observações. Usamos isto para escolher entre vários modelos qual o melhor. A segunda questão nos permite supor qual caminho provavelmente foi seguido pela cadeia de Markov. Na prática, é onde etiquetamos as palavras de um texto, pois procuramos a seqüência de etiquetas que melhor explica a seqüência de palavras dadas. Na terceira questão, não sabemos os parâmetros do modelo, e por isso precisamos estimá-los a partir dos dados. Este é o processo de treino supervisionado, citado na Seção 2.2.

Podemos traduzir estas questões para o contexto da etiquetagem. Seja  $w_{1,n}$  uma seqüência de  $n$  palavras observadas, e  $l_i$  a etiqueta de  $w_i$ . Então,

1. Como calculamos  $P(w_{1,n}|\mu)$ , ou seja, qual a probabilidade da seqüência  $w_{1,n}$  ter sido gerada por  $\mu$ ?
2. Como achar  $l_{1,n}$  tal que  $P(l_{1,n}|w_{1,n})$  é a melhor possível, isto é, qual é a melhor seqüência de etiquetas para as palavras dadas?
3. Como treinar  $\mu$  de modo que  $P(w_{1,n}|\mu)$  seja maximizada?

A seguir descrevemos cada uma destas questões.

## 4.1 Encontrando a Probabilidade de uma Observação

Digamos que temos uma sentença  $w_{1,n}$  com etiquetas  $l_{1,n}$ , e que nosso etiquetador usa um modelo  $\mu$ . Então, com qual probabilidade o etiquetador atribuiria a  $w_{1,n}$  as etiquetas  $l_{1,n}$ ? Em outras palavras, qual a probabilidade de que  $l_{1,n}$  sejam as etiquetas de  $w_{1,n}$ ?

Para qualquer seqüência de estados  $X = (X_1, \dots, X_{T+1})$ ,

$$\begin{aligned} P(O|X, \mu) &= \prod_{t=1}^T P(o_t|X_t, X_{t+1}, \mu) \\ &= b_{X_1 X_2 o_1} b_{X_2 X_3 o_2} \cdots b_{X_T X_{T+1} o_T} \end{aligned}$$

e,

$$P(X|\mu) = \pi_{X_1} a_{X_1 X_2} a_{X_2 X_3} \cdots a_{X_T X_{T+1}}.$$

Agora, pela regra de Bayes,

$$P(O, X|\mu) = P(O|X, \mu)P(X|\mu).$$

Assim,

$$\begin{aligned} P(O|\mu) &= \sum_X P(O|X, \mu)P(X|\mu) \\ &= \sum_{X_1 \cdots X_{T+1}} \pi_{X_1} \prod_{t=1}^T a_{X_t X_{t+1}} b_{X_t X_{t+1} o_t} \end{aligned}$$

Nesta derivação, nós simplesmente somamos a probabilidade da observação ocorrer de acordo com cada seqüência possível de estados. Infelizmente a avaliação direta da expressão é invariavelmente ineficiente. Para o caso geral (onde podemos começar em qualquer estado, e ir para qualquer outro a cada passo), o cálculo requer  $(2T + 1) \cdot N^{T+1}$  multiplicações.

Para evitar esta complexidade utilizamos a técnica de programação dinâmica, onde lembramos resultados parciais ao invés de recalculá-los, através de um algoritmo chamado de Algoritmo PROGRESSIVO [23, 19].

### 4.1.1 Algoritmo Progressivo

Seja  $\alpha_i(t)$  a probabilidade observar  $w_{1,t-1}$  e chegar ao estado  $l_i$  no tempo  $t$ .

$$\alpha_i(t) = P(w_{1,t-1}, X_t = l_i | \mu)$$

Calculamos as variáveis progressivas usando o seguinte procedimento:

1. Inicialização

$$\alpha_i(1) = \pi_i, \quad 1 \leq i \leq N$$

2. Indução

$$\alpha_j(t+1) = \sum_{i=1}^N \alpha_i(t) a_{ij} b_{ij o_t}, \quad 1 \leq t \leq T, \quad 1 \leq j \leq N$$

3. Total

$$P(O|\mu) = \sum_{i=1}^N \alpha_i(T+1)$$

Este é um algoritmo muito mais eficiente que requer apenas  $2N^2T$  multiplicações.

Só precisamos deste algoritmo para resolver a questão 1. Mas para ajudar a resolver a questão 3, iremos introduzir outro algoritmo, bastante semelhante.

### 4.1.2 Algoritmo Regressivo

O Algoritmo REGRESSIVO calcula variáveis regressivas, que são a probabilidade total de vermos o resto da seqüência da observação dado que estávamos no estado  $l_i$  no tempo  $t$ . A razão de incluímos este cálculo menos intuitivo é porque o uso combinado das probabilidades progressiva e regressiva é essencial para resolvermos a terceira questão da estimação de parâmetros.

Defina variáveis regressivas como

$$\beta_i(t) = P(w_t \cdots w_T | X_t = i, \mu)$$

Então podemos calcular variáveis regressivas da direita para a esquerda como segue:

1. Inicialização

$$\beta_i(T+1) = 1, \quad 1 \leq i \leq N$$

2. Indução

$$\beta_i(t) = \sum_{j=1}^N a_{ij} b_{ij o_t} \beta_j(t+1), \quad 1 \leq t \leq T, \quad 1 \leq i \leq N$$

3. Total

$$P(O|\mu) = \sum_{i=1}^N \pi_i \beta_i(1)$$

Combinando as probabilidades progressiva e regressiva, também obtemos a probabilidade da seqüência de observações.

$$P(O|\mu) = \sum_{i=1}^N \alpha_i(t) \beta_i(t), \quad 1 \leq t \leq T+1$$

## 4.2 Encontrando a Melhor Seqüência de Estados

A segunda questão pergunta qual a seqüência de estados que melhor explica as observações. O método normalmente usado é o algoritmo de Viterbi [31], que computa eficientemente a melhor seqüência de estados.

### 4.2.1 Algoritmo de Viterbi

Queremos encontrar o melhor caminho completo, isto é,

$$\arg \max_X P(X|O, \mu)$$

Como

$$P(X|O, \mu) = \frac{P(X, O|\mu)}{P(O|\mu)},$$

e  $O$  é fixo, é suficiente maximizar:

$$\arg \max_X P(X, O|\mu)$$

Defina:

$$\delta_j(t) = \max_{X_1 \cdots X_{t-1}} P(X_1 \cdots X_{t-1}, o_1 \cdots o_{t-1}, X_t = j|\mu)$$

Ou seja,  $\delta_j(t)$  é a probabilidade de um melhor caminho em  $\mu$  que produz as observações  $o_1 \cdots o_t$  e que tem  $j$  como último estado. Em outras palavras, é a probabilidade do caminho mais provável que leva a este estado. Para saber qual é este caminho, usamos a variável correspondente  $\psi_j(t)$ , que guarda o estado do arco de entrada que leva a este caminho mais provável. Usando programação dinâmica, calculamos o caminho mais provável como segue:

1. Inicialização

$$\delta_j(1) = \pi_j, \quad 1 \leq j \leq N$$

2. Indução

$$\delta_j(t+1) = \max_{1 \leq i \leq N} \delta_i(t) a_{ij} b_{ijw_t}, \quad 1 \leq j \leq N$$

Guarda rastro

$$\psi_j(t+1) = \arg \max_{1 \leq i \leq N} \delta_i(t) a_{ij} b_{ijw_t}, \quad 1 \leq j \leq N$$

3. Finalização e restauração do caminho. A melhor seqüência de estados é obtida de trás para a frente:

$$\hat{X}_{T+1} = \arg \max_{1 \leq i \leq N} \delta_i(T+1)$$

$$\hat{X}_t = \psi_{\hat{X}_{t+1}}(t+1)$$

$$P(\hat{X}) = \max_{1 \leq i \leq N} \delta_i(T+1)$$

A complexidade de tempo do algoritmo de Viterbi é  $O(TN^2)$ .

### 4.3 Estimando os Parâmetros

A questão de estimar os parâmetros de um HMM também é conhecida como o problema do treinamento, que é um método de aprendizado supervisionado. Dada uma certa seqüência de observações  $O$  sobre um alfabeto  $K$  e estados  $S$ , queremos encontrar os valores dos parâmetros do modelo  $\mu = (A, B, \Pi)$  que melhor explicam o que observamos. Isto significa que queremos encontrar os valores que maximizam  $P(O|\mu)$ :

$$\arg \max_{\mu} P(O|\mu).$$

Entretanto, não existem métodos analíticos conhecidos para resolver o problema de maneira exata, isto é, escolher  $\mu$  que maximiza  $P(O|\mu)$ . Por isto geralmente usam-se métodos iterativos, que garantem apenas encontrar máximos locais. Um destes métodos mais conhecido é o *algoritmo de Baum-Welch* [3] ou *algoritmo Progressivo-Regressivo*, que é um caso especial do método de Maximização de Esperança (do inglês, *Expectation Maximization method*). Neste algoritmo, começamos com um modelo inicial (pré-selecionado, ou escolhido aleatoriamente) e calculamos a probabilidade da seqüência de observações usando este modelo. Olhando para este cálculo, podemos ver quais transições de estado foram mais usadas, e então aumentar a probabilidade delas (ainda respeitando as restrições estocásticas). Com isto, podemos escolher um modelo revisado que fornece uma probabilidade mais alta para a seqüência de observações. Repetimos então este processo, esperando convergir em valores ótimos para os parâmetros do modelo  $\mu$ . Este processo de maximização é geralmente chamado de *treinamento* do modelo, e é feito usando-se *dados de treinamento*. Por isto, e para dar mais representatividade ao aprendizado, é importante a existência de um cópulus pré-etiquetado e de tamanho razoável.

Seja  $p_t(i, j)$ ,  $1 \leq t \leq T$ ,  $1 \leq i, j \leq N$ , a probabilidade de atravessar o arco do estado  $i$  para o  $j$  no tempo  $t$  dada a seqüência de observações  $O$ .

$$\begin{aligned} p_t(i, j) &= P(X_t = i, X_{t+1} = j | O, \mu) \\ &= \frac{P(X_t = i, X_{t+1}, O | \mu)}{P(O | \mu)} \\ &= \frac{\alpha_i(t) a_{ij} b_{j o_t} \beta_j(t+1)}{\sum_{m=1}^N \alpha_m(t) \beta_m(t)} \end{aligned}$$

Usamos as seguintes fórmulas para re-estimarmos as probabilidades do modelo:

$$\begin{aligned} \hat{\pi}_i &= \text{freqüência esperada no estado } i \text{ no tempo } t = 1 \\ &= \sum_{j=1}^N p_1(i, j) \\ \hat{a}_{ij} &= \frac{\text{freqüência de transições do estado } i \text{ para o } j}{\text{freqüência de transições saindo do estado } i} \\ &= \frac{\sum_{t=1}^T p_t(i, j)}{\sum_{t=1}^T \sum_{j=1}^N p_t(i, j)} \\ \hat{b}_{ijk} &= \frac{\text{freqüência de transições de } i \text{ para } j \text{ com } k \text{ observado}}{\text{freqüência de transições de } i \text{ para } j} \\ &= \frac{\sum_{\{t: o_t=k, 1 \leq t \leq T\}} p_t(i, j)}{\sum_{t=1}^T p_t(i, j)} \end{aligned}$$

Assim, de  $\mu = (A, B, \Pi)$ , derivamos  $\hat{\mu} = (\hat{A}, \hat{B}, \hat{\Pi})$ . E como provado por Baum, temos  $P(O|\hat{\mu}) \geq P(O|\mu)$ . Portanto, repetindo esta re-estimação de parâmetros, esperamos melhorar

nosso modelo, até que não tenhamos mais resultados significativamente melhores. Entretanto, como dissemos antes, não temos garantia de que encontraremos o melhor modelo, porque o processo pode travar em um máximo local. Ainda assim, geralmente o algoritmo de Baum-Welch é eficiente para HMMs.

# Cadeias de Markov de Tamanho Variável

O número de parâmetros de uma cadeia de Markov cresce exponencialmente com sua ordem. Por isto, no contexto da etiquetagem, geralmente são usadas ordens pequenas. Por exemplo, assumindo que existam 383 etiquetas diferentes, como no corpus Tycho Brahe, existem  $383^3 = 56.181.887$  diferentes combinações de ordem 3 das etiquetas (claro que nem todas elas irão realmente ocorrer). Assim, modelos que utilizam cadeias de Markov de ordem alta são difíceis de estimar. Entretanto, existem palavras que dependem de um contexto maior para serem etiquetadas corretamente. Então gostaríamos de condicionar o tamanho do contexto a ser analisado à palavra sendo analisada.

A idéia, então, é permitir que a memória da cadeia de Markov utilizada no modelo do etiquetador tenha um tamanho variável, dependendo dos valores passados observados. Daí o nome Cadeias de Markov de Tamanho Variável (referenciadas na literatura como VLMC, *Variable Length Markov Chains*, do inglês). Neste Capítulo iremos explicar o conceito de VLMC's considerando o contexto do problema da etiquetagem morfo-sintática. Assim, a notação usada será a mais semelhante possível à notação que usamos na seção referente às cadeias de Markov ocultas (Seção 4). Para uma descrição formal e com as provas matemáticas veja [8, 22].

Para começar, considere uma cadeia de Markov de ordem finita  $k$ . Então,

$$P(l_i | l_{i-\infty, i-1}) = P(l_i | l_{i-k, i-1}), \forall l_{i-\infty, i}.$$

A idéia de uma memória de tamanho variável pode ser vista como um corte de estados irrelevantes do histórico  $l_{i-k, i-1}$ . Em outras palavras, o histórico de  $l_i$  possui apenas alguns estados que precisam ser considerados. Ao conjunto destes estados damos o nome de *contexto* de  $l_i$ .

Formalizando, seja  $X$  um processo estacionário (assim como as cadeias de Markov de ordem fixa, página 6) com valores  $l_i \in \mathcal{L}$ ,  $|\mathcal{L}| < \infty$ ,  $\mathcal{L}$  o conjunto de etiquetas possíveis. Seja  $c : \mathcal{L} \rightarrow \mathcal{L}$  uma função que mapeia

$$\begin{aligned} c : l_{i-\infty, i-1} &\mapsto l_{i-h, i-1}, \text{ onde } h \text{ é definido por} \\ h = h(l_{i-\infty, i-1}) &= \min\{k | P(l_i | l_{i-\infty, i-1}) = P(l_i | l_{i-k, i-1}) \forall l_i \in \mathcal{L}\} \\ \text{onde } h \equiv 0 &\text{ corresponde à independência.} \end{aligned}$$

Então,  $c(\cdot)$  é chamada de uma função contexto e para qualquer  $i < |\mathcal{L}|$ ,  $c(l_{i-\infty, i-1})$  é chamada de função contexto para a variável  $l_i$ . O nome *contexto* se refere à porção do passado que influencia a próxima saída.

Seja  $0 \leq k \leq \infty$  o menor inteiro tal que

$$|c(l_{i-\infty, i-1})| = h(l_{i-\infty, i-1}) \leq k, \forall l_{i-\infty, i-1} \in \mathcal{L}. \quad (5.1)$$

Então  $c(\cdot)$  é dita uma função contexto de ordem  $k$ , e se  $k < \infty$ ,  $X$  é dita uma cadeia de Markov de tamanho variável de ordem  $k$ .

Geralmente, o espaço da função contexto  $c(\cdot)$  não é o espaço completo  $\mathcal{L}^k$ . Neste caso a VLMC de ordem  $k$  é uma cadeia de Markov de ordem  $k$ , mas que tem *memória de tamanho variável*  $h$ . Do contrário, a VLMC é uma cadeia de Markov completa de ordem  $k$ .

## 5.1 Representação do Espaço de Estados

Os estados que determinam as probabilidades de transição da VLMC são dados pelos valores da função contexto  $c(\cdot)$ . Representamos estes estados — o espaço de estados minimal da VLMC — como uma árvore.

Uma **árvore de contexto** é uma árvore com um nó raiz no topo, de onde descem ramificações, tal que cada nó interno da árvore possui no máximo  $|\mathcal{L}|$  filhos. Veja a Figura 5.1. Cada valor de uma função contexto  $c(\cdot)$  pode ser representado como um galho (i.e., um caminho que vai da raiz até uma folha) de tal árvore. O contexto  $c(l_{i-\infty, i-1})$  é representado por um galho, cujo sub-galho no topo é determinado por  $l_{i-1}$ , o próximo sub-galho por  $l_{i-2}$  e assim por diante, até a folha, determinada por  $l_{i-h(l_{i-\infty, i-1})}$ .

Um exemplo de uma árvore de contexto contendo etiquetas morfo-sintáticas é dado na Figura 5.2. A partir da raiz, estão as etiquetas mais recentes de um contexto, descendo pelos nós para as mais antigas, até as últimas consideradas em um contexto. Cada nó fornece a probabilidade de uma etiqueta  $l$  dado que o contexto ocorrido é a seqüência formada por este nó mais os nós acima dele. Assim, se tivermos a frase

A criança correu rapidamente para casa quando começou a chover. ,

e já tivermos etiquetado até *casa*,

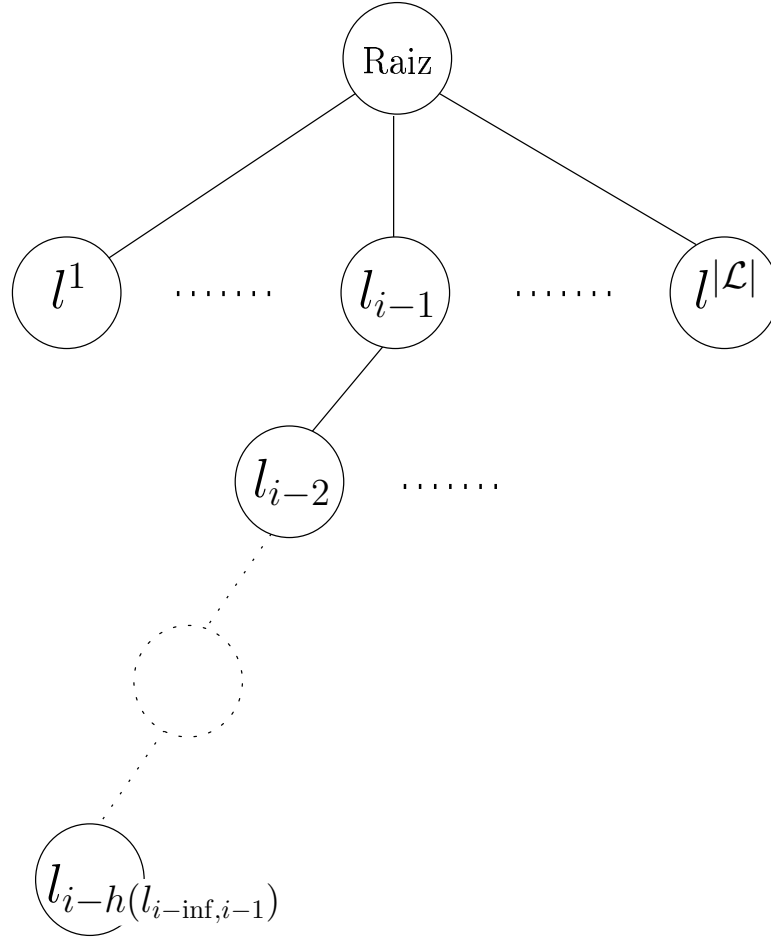
A/D criança/N correu/VB rapidamente/ADV para/P casa/N quando começou a chover. ,

a probabilidade de que a etiqueta  $l$  venha em seguida é dada pelo nó **VB** mais abaixo na árvore:  $P(l|VB, ADV, P, N)$  (para efeitos de exemplo não estamos considerando a palavra a ser etiquetada, *quando*). O restante do histórico,  $D, N$ , não é considerado relevante, por isso não está na árvore.

Na próxima seção introduzimos um algoritmo de contexto que constrói uma destas árvores para uma VLMC. Por isto, é conveniente darmos a noção de uma árvore de contexto terminal. Seja  $\tau$  uma árvore de contexto como definida no parágrafo anterior. Então uma árvore de contexto terminal  $\tau^T$  é uma árvore que contém somente nós terminais (folhas) de  $\tau$ .

## 5.2 Algoritmo de Contexto

Dada uma VLMC, o objetivo é encontrar a função contexto  $c(\cdot)$  subjacente e suas probabilidades. Para resolver esta questão usamos uma versão do algoritmo de contexto de Rissanen [25]. Primeiro, construímos uma grande árvore de contexto, utilizando para isso o córpus de treinamento. Depois, o algoritmo utiliza uma função reversa de poda de árvore considerando um critério de decisão local.



**Figura 5.1:** Definição de uma **Árvore de Contexto**.

Convencionamos que quantidades envolvendo índices de tempo fora de  $\{1, \dots, n\}$  são iguais a zero (ou irrelevantes). Seja  $C(v)$  o número de ocorrências da etiqueta  $v$  na seqüência  $l_{1,n}$ . Ainda, seja

$$\hat{P}(v) = C(v)/n, \quad \hat{P}(u|v) = \frac{C(uv)}{C(v)}, \quad v, u \in \mathcal{L}^+, \quad (5.2)$$

$\mathcal{L}^+$  o conjunto de todas seqüências não nulas de etiquetas de  $\mathcal{L}$ .

**Algoritmo 5.1 (Algoritmo de Contexto)**

**Passo 1** Dadas as etiquetas  $l_1, \dots, l_n$  em  $\mathcal{L}$ , busque a função contexto  $c_{max}(\cdot)$  com árvore de contexto terminal  $\tau_{max}^T$ , onde  $\tau_{max}^T$  é a maior árvore tal que toda folha em  $\tau_{max}^T$  tenha sido observada pelo menos duas vezes em  $l_1, \dots, l_n$ . Faça  $\tau_i^T = \tau_{max}^T$ ,  $i$  o número da iteração sendo executada.

**Passo 2** Examine cada elemento (folha) de  $\tau_i^T$  como segue. Seja  $c(\cdot)$  a função contexto correspondente de  $\tau_i^T$  e seja

$$vu = l_{i-h+1,i} = c(l_{i-\text{inf},i}), \quad u = l_{i-h+1}, \quad v = l_{i-h+2,i},$$

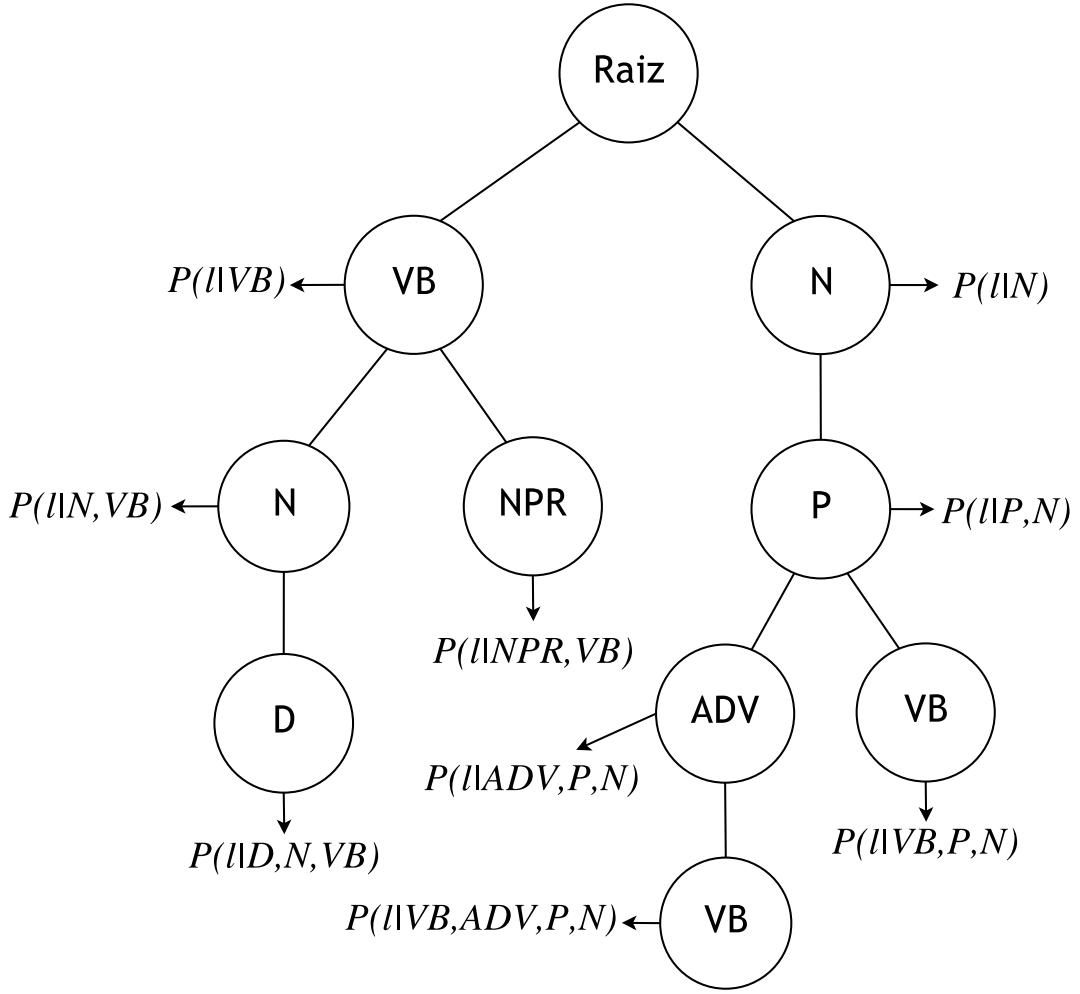


Figura 5.2: Exemplificação de uma árvore de contexto.

onde  $vu$  é um elemento (folha) de  $\tau_i^T$ , o qual comparamos com sua versão podada  $v = l_{i-h+2,i}$ . Se  $h = 1$ , a versão podada é o ramo vazio, isto é, a raiz.

Faça a poda de  $vu = l_{i-h+1,i}$  para  $v = l_{i-h+2,i}$  se

$$\Delta_{vu} = \sum_{l \in \mathcal{L}} P(l|vu) \log \left( \frac{P(l|vu)}{P(l|v)} \right) C(vu) < K, \quad (5.3)$$

onde  $K$  é o valor de corte, definido como

$$K = K_n \sim C \log(n), \quad C > 2|\mathcal{L}| + 4, \quad (5.4)$$

e  $P(\cdot|\cdot)$  é como em 5.2. A decisão sobre podar para cada elemento leva possivelmente a uma árvore menor  $\tau_{i+1}^T$ . Construa tal árvore de contexto terminal.

**Passo 3** Repita o **Passo 2** para  $i = 1, 2, \dots$ , até que mais nenhuma poda seja possível. Denote esta árvore de contexto podada maximal por  $\hat{\tau}$ , e sua função contexto correspondente por  $\hat{c}(\cdot)$ .

**Passo 4** Estime as probabilidades de transição  $P(l_i|c(l_{i-\infty,i-1}))$  por  $\hat{P}(l_i|\hat{c}(l_{i-\infty,i-1}))$ .

---

Para obter o contexto de uma palavra, primeiro construímos uma árvore com todo passado possível, e depois vamos efetuando podas nesta árvore. Fazemos isto comparando a probabilidade de um contexto com sua probabilidade sem a última etiqueta. Se a diferença não é significativa, podemos do contexto esta última etiqueta. A relevância desta diferença nas probabilidades é dada pelo valor de corte  $K$ , determinado por uma constante  $\mathcal{C}$ , obtida empiricamente.

# Etiquetadores Morfo-Sintáticos

Implementamos dois etiquetadores morfo-sintáticos baseados nos modelos de Markov descritos anteriormente. Primeiro, implementamos um etiquetador supervisionado baseado em modelos de Markov ocultos (HMM) de ordem dois, ou seja, onde a etiqueta de uma palavra depende da própria palavra e das duas etiquetas anteriores. Depois, implementamos um etiquetador baseado em cadeias de Markov de tamanho variável (VLMC), ou seja, onde a etiqueta de uma palavra é determinada pela própria palavra e por uma seqüência variável de etiquetas anteriores.

Nesta seção vamos descrever os passos tomados na criação destes etiquetadores morfo-sintáticos. Para isto, vamos dividir este Capítulo em três seções, explicando de forma progressiva os esforços aplicados nesta criação. Primeiro, na Seção 6.1, vamos mostrar um ajuste das teorias dos Capítulos anteriores, feito para tornar viável a implementação computacional das mesmas. Como veremos, muitas equações não são trivialmente implementáveis. Depois, na Seção 6.2, iremos apresentar a estrutura funcional dos etiquetadores, e então, na seção seguinte (6.3), vamos mostrar os detalhes internos destas estruturas.

## 6.1 Modelos Internos

Para implementar eficientemente os etiquetadores, utilizamos algumas técnicas tradicionais da área [12]. A simples implementação trivial das equações descritas anteriormente não resolve alguns problemas computacionais. Assim, precisamos fazer alguns ajustes e tomar alguns cuidados para evitar estes problemas. Nas duas seções seguintes mostramos estes ajustes, primeiro os do etiquetador de ordem dois baseado em HMMs e depois os do baseado em VLMCs.

### 6.1.1 O Etiquetador de Ordem Dois (HMM)

Este nosso etiquetador morfo-sintático usa modelos de Markov de segunda ordem, onde os estados representam as etiquetas, e as observações representam as palavras. As probabilidades de transição dependem dos estados, neste caso pares de etiquetas, e as probabilidades de saída dependem do estado destino. Formalmente, queremos calcular

$$\arg \max_{l_1 \dots l_T} \left[ \prod_{i=1}^T P(l_i | l_{i-1}, l_{i-2}) P(w_i | l_i) \right] \quad (6.1)$$

para uma dada seqüência de palavras  $w_1 \dots w_T$  de comprimento  $T$ , onde  $l_1 \dots l_T$  são elementos do conjunto de etiquetas, e  $l_0$  e  $l_{-1}$  são marcadores de início de sentença.

As probabilidades de transição e saída são estimadas a partir de um corpus etiquetado. Inicialmente, usamos as probabilidades de máxima verossimilhança  $\hat{P}$ , que são derivadas das freqüências

relativas:

$$\begin{aligned} \text{Léxica:} \quad \hat{P}(w_3|l_3) &= \frac{C(w_3,l_3)}{C(l_3)} \\ \text{Bigrama:} \quad \hat{P}(l_3|l_2) &= \frac{C(l_2,l_3)}{C(l_2)} \\ \text{Trigrama:} \quad \hat{P}(l_3|l_1, l_2) &= \frac{C(l_1,l_2,l_3)}{C(l_1,l_2)} \end{aligned}$$

para todo  $l_1, l_2, l_3$  no conjunto de etiquetas e  $w_3$  no léxico, onde  $C(X)$  é o número de vezes em que  $X$  ocorre no cópuz em questão. Definimos uma probabilidade de máxima verossimilhança como zero se os numeradores e denominadores correspondentes são zero.

Na Equação 6.1 ficamos em dúvida se usávamos  $P(w_i|l_i)$  ou  $P(l_i|w_i)$ . Durante as primeiras fases da implementação deste etiquetador, onde usávamos um cópuz de treinamento pequeno, o uso de  $P(l_i|w_i)$  dava melhores resultados. Já nas fases finais, quando passamos a utilizar um cópuz de treinamento maior, os resultados melhores foram obtidos com  $P(w_i|l_i)$ . A influência do tamanho do cópuz é exatamente a explicação dada por Charniak [9]. Ele diz que Church [11] e DeRose [14] usaram cada um um dos termos acima, e então Boggess et al. [4] testaram ambos e reportaram que  $P(l_i|w_i)$  era melhor. Mas Boggess et al. usaram um cópuz muito pequeno, e então Charniak et al. [10] testaram os dois termos com um cópuz grande, de aproximadamente um milhão de palavras, e mostraram que  $P(w_i|l_i)$  dava resultados significativamente melhores.

### 6.1.1.1 Representação dos Estados

Por causa da esparsidade dos dados de treinamento, é provável que ocorrências raras de palavras e de seqüências de etiquetas tenham probabilidade zero no modelo treinado. Assim, quando o modelo for usado para etiquetar um texto em que uma destas palavras ou seqüência de etiquetas ocorrer, por causa das multiplicações das probabilidades a probabilidade de toda sentença será zero, o que não é um efeito desejado porque torna impossível classificar diferentes sentenças que contêm probabilidade zero. Para evitar que isto aconteça, não fazemos uso da equação 6.1. Ao invés disto, utilizamos sempre o método de interpolação linear, definido como:

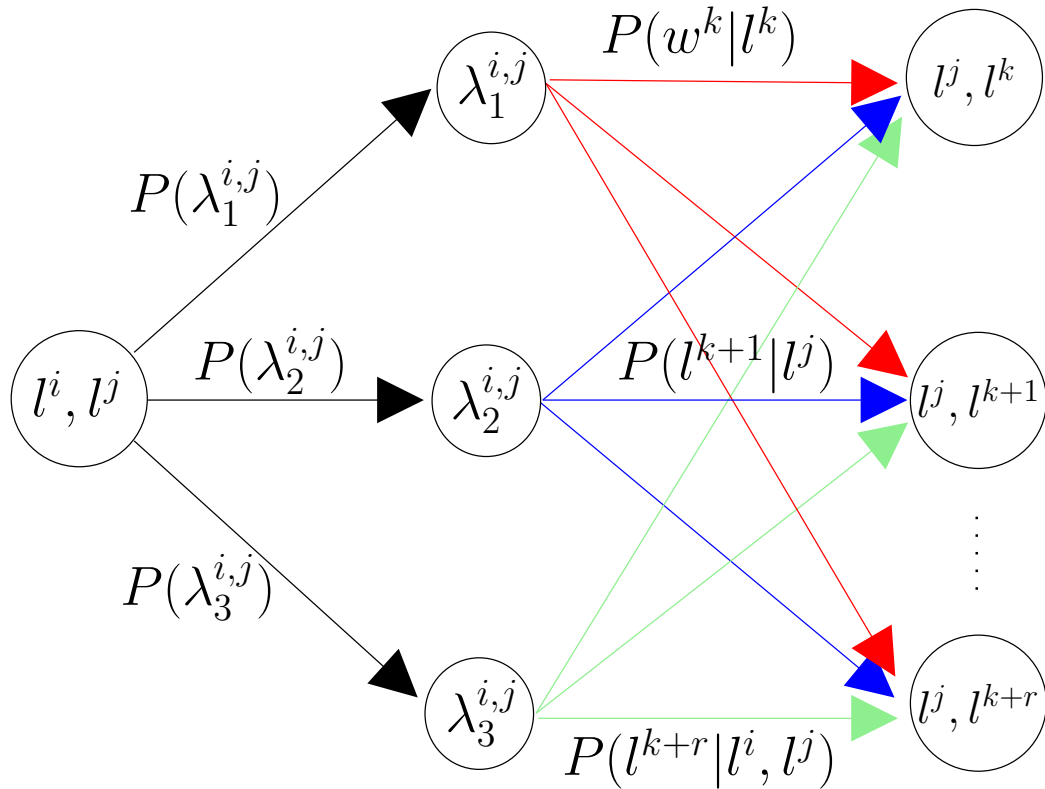
$$P(l_i|l_{1,i-1}, w_i) = \lambda_1 P_1(w_i|l_i) + \lambda_2 P_2(l_i|l_{i-1}) + \lambda_3 P_3(l_i|l_{i-1}, l_{i-2})$$

Atribuimos valores aos  $\lambda$ 's de maneira que  $\lambda_1 + \lambda_2 + \lambda_3 = 1$ , assim  $P$  de novo representa distribuições de probabilidade.

A questão, então, é como definir estes valores. Podemos fazer isto manualmente, ou tentar encontrar a combinação de pesos que funciona melhor, estimando-os automaticamente através de treinamento, considerando ou não o contexto. Em seu etiquetador TnT, Brants [5] usa a variante independente de contexto da interpolação linear, explicando que com os valores dos  $\lambda$ 's não dependendo de um trigrama em particular, os resultados são melhores do que com  $\lambda$ 's estimados para cada trigrama ou grupo de trigramas. Manning e Schütze [19] falam que a aplicação de um algoritmo de Maximização de Esperança (EM, do inglês *Expectation Maximization*) para obter valores dos  $\lambda$ 's automaticamente funciona bem. Entretanto, até onde estamos cientes, ainda não foram publicados resultados comparando as diferenças entre valores definidos manualmente e obtidos automaticamente, com contexto ou não.

Em nossa implementação, utilizamos uma modelagem na qual cada estado possui seus próprios  $\lambda$ 's. Em outras palavras, acrescentamos três estados  $\lambda$ 's a cada estado original (que contém um par de etiquetas). Para visualizar como é esta modelagem, veja a Figura 6.1.

Desta maneira, decidimos realizar a fase de treinamento sobre os  $\lambda$ 's, e não mais sobre as probabilidades obtidas do cópuz. Em outras palavras, a probabilidade de uma palavra ter uma etiqueta, ou de uma etiqueta vir depois de outras duas, não é alterada. Ao invés disto, treinamos



**Figura 6.1:** Modelo interno do etiquetador baseado em HMM.

o etiquetador sobre os valores dos  $\lambda$ 's de cada estado, tentando, assim, realçar as transições mais relevantes entre dois estados particulares.

Os valores iniciais para os três lambdas acima foram definidos empiricamente como:

$$\lambda_1 = 0,4$$

$$\lambda_2 = 0,3$$

$$\lambda_3 = 0,3$$

### 6.1.1.2 Tratamento de Palavras Desconhecidas

Quando etiquetamos um texto que não está contido no cópulo que usamos para treinar nosso etiquetador, geralmente encontramos palavras desconhecidas ao nosso modelo. Assim, precisamos de um método que trate estas palavras e tente etiquetá-las corretamente.

No etiquetador implementado, utilizamos dois métodos complementares. O primeiro consiste em restringir as possíveis etiquetas de uma palavra desconhecida, isto é, não consideramos as etiquetas que são fechadas. Por exemplo, não encontraremos uma palavra desconhecida que seja um artigo ou uma conjunção, a não ser que nosso etiquetador tenha sido treinado com um cópulo pequeno demais (como apenas 1000 palavras e etiquetas). É mais provável que uma palavra desconhecida seja um verbo ou um nome. Estas etiquetas que têm grandes chances de serem atribuídas a palavras desconhecidas são chamadas de etiquetas abertas.

No segundo método, analisamos a morfologia da palavra. Isto pode ser feito em duas partes: uma é analisar o sufixo da palavra, e outra, a raiz da palavra. No último, usa-se a técnica

de *stemming* para extrair a raiz. Em nosso etiquetador, utilizamos uma análise de sufixos simplificada<sup>1</sup>: construímos uma árvore com os sufixos de todas as palavras do córpus de treino que possuem uma etiqueta aberta; normalizamos esta árvore, criando distribuições de probabilidade; e então para uma palavra desconhecida procuramos seu maior sufixo existente na árvore, retornando as probabilidades das etiquetas possíveis. Consideramos como sufixo de uma palavra usada para construir a árvore a última metade da palavra. Além disto, verificamos se a primeira letra da palavra é maiúscula e, se sim, caso esta palavra não seja a primeira da sentença, atribuímos uma probabilidade alta de que ela seja um nome próprio.

Como estamos interessados em medir o desempenho dos modelos internos de nossos dois etiquetadores, não implementamos métodos avançados de tratamento de palavras desconhecidas. Futuramente, poderemos voltar a esta questão e analisar se ela acrescenta um ganho maior nos resultados de apenas um modelo ou não.

### 6.1.2 O Etiketador de Tamanho Variável (VLMC)

Este etiquetador utiliza cadeias de Markov de tamanho variável, onde as seqüências de etiquetas obtidas do córpus de treino são usadas para alimentar a árvore de contexto necessária ao modelo. Depois que todo o córpus é lido, a árvore é podada, para que sejam consideradas apenas seqüências relevantes de etiquetas. Os galhos restantes que formam a árvore definem a função contexto desta árvore. Assim, para uma dada seqüência de etiquetas  $l_{i-\infty, i-1}$ , a probabilidade de que a próxima etiqueta seja  $l_i$  é dada por  $P(l_i|c(l_{i-\infty, i-1}))$ .

Mas além desta probabilidade do contexto, o etiquetador também precisa considerar a palavra que vai receber a etiqueta, isto é, a probabilidade de que a palavra  $w_i$  tenha a etiqueta  $l_i$ . Esta probabilidade é dada por  $P(w_i|l_i)$ <sup>2</sup>, e está armazenada numa matriz de transições, não contida na árvore de contexto.

Desta maneira, para etiquetar uma palavra, o etiquetador precisa considerar estas duas probabilidades. Uma maneira simples de se fazer isto é usar o método de interpolação, mostrado na Seção 6.1.1.1, e usar dois lambdas. Assim, a probabilidade final da próxima etiqueta ser  $l_i$  é dada por

$$P(l_i|w_i, c(l_{i-\infty, i-1})) = \lambda_1 P(w_i|l_i) + \lambda_2 P(l_i|c(l_{i-\infty, i-1})). \quad (6.2)$$

Entretanto, durante os testes, não estávamos conseguindo encontrar a melhor combinação de valores para estes lambdas. Experimentamos, então, simplesmente multiplicar as probabilidades da palavra e do contexto:

$$P(l_i|w_i, c(l_{i-\infty, i-1})) = P(w_i|l_i) * P(l_i|c(l_{i-\infty, i-1})). \quad (6.3)$$

E esta equação deu melhores resultados do que qualquer combinação testada de lambdas da Equação 6.2. A justificativa para isto é que existe uma hipótese de quase independência entre os fatores da equação acima, além de que ela se assemelha à equação 6.1 feita para HMM's. Portanto, todos os resultados deste etiquetador da Seção 7.2 foram obtidos usando a Equação 6.3.

#### 6.1.2.1 Valor de Corte da Árvore de Contexto

Durante o processo de poda da árvore de contexto, descrito no algoritmo 5.1, a decisão sobre o corte ou não de uma folha é determinada por um valor  $K$  (Equação 5.3), chamado *valor de corte* da árvore de contexto. Na teoria, como definido em [8], este valor de corte é determinado pela

<sup>1</sup>O termo sufixo usado significa “seqüência final de caracteres de uma palavra”, o que não é necessariamente um sufixo significativo linguisticamente.

<sup>2</sup>Utilizamos este termo e não o  $P(l_i|w_i)$  pelos mesmos motivos explicados na Seção 6.1.1 acima.

Equação 5.4, repetida abaixo.

$$K = K_n \sim \mathcal{C} \log(n), \quad \mathcal{C} > 2|\mathcal{L}| + 4,$$

Neste caso,  $n$  é o número de palavras usadas no treino do etiquetador e  $\mathcal{L}$  é o conjunto das etiquetas reconhecidas.

Assim, como veremos nos resultados mostrados na Seção 7.2, um treinamento com 775602 palavras faz o etiquetador reconhecer 262 etiquetas distintas. Aplicando estes valores na equação acima, e fazendo  $\mathcal{C} = 2|\mathcal{L}| + 5$ , o valor de  $K$  é igual a aproximadamente 10350. Durante a construção e os testes com o etiquetador, este valor se mostrou excessivamente grande, fazendo com que quase toda a árvore fosse podada. Por isto efetuamos vários testes utilizando diferentes valores para  $K$ .

Como era esperado, observamos que valores de  $K$  próximos a zero faziam com que muito pouco da árvore fosse podado. O resultado obtido com isto não foi bom, nos levando a concluir que a árvore possuía muitos galhos pouco significativos, que atrapalhavam a etiquetagem. Por outro lado, valores muito altos de  $K$ , perto de 500, faziam com que a árvore fosse podada demais, tirando fora muita informação do contexto, embora o resultado não fosse muito ruim.

Por fim, decidimos utilizar o valor de  $K$  dado pela equação

$$K = \frac{\log(n)}{\log(|\mathcal{L}|)} \cdot \frac{n}{|\mathcal{S}|}, \quad (6.4)$$

onde  $|\mathcal{S}|$  é o número de sentenças usadas no treinamento,  $n$  é o número de palavras (e assim  $n/|\mathcal{S}|$  é o número médio de palavras por sentença), e  $|\mathcal{L}|$  é o número de etiquetas distintas. Para o córpus de treino inteiro (775602 palavras),  $K$  ficou valendo 54,6615 (mas este valor pode ser definido manualmente no momento da execução do etiquetador, como consta no Apêndice B). Os resultados mostrados na Seção 7.2 foram obtidos com a utilização desta equação.

## 6.2 Arquiteturas dos Etiketadores

O primeiro etiquetador implementado foi o baseado em cadeias de Markov de ordem 2. Vamos chamá-lo de **HMM Tagger**. O segundo foi o etiquetador baseado em cadeias de Markov de tamanho variável, o **VLMC Tagger**. Para alcançar nosso objetivo de comparar lado a lado as duas teorias, buscamos tornar a implementação dos dois etiquetadores a mais semelhante possível. Por isto, implementamos o **VLMC Tagger** a partir da implementação do **HMM Tagger**, deste modo fazendo com que os dois possuam várias estruturas em comum. Isto pode ser visto nas duas figuras a seguir: a Figura 6.2 mostra a arquitetura do **HMM Tagger**, e a Figura 6.3 mostra a arquitetura do **VLMC Tagger**.

Nas seções a seguir, vamos primeiro descrever as estruturas comuns aos dois etiquetadores, e então vamos mostrar as estruturas exclusivas do **HMM Tagger** e depois do **VLMC Tagger**.

### 6.2.1 Arquitetura Básica

As estruturas de entrada e de saída dos dois etiquetadores são as mesmas. Como entrada fornecemos o córpus de treino e o córpus de teste, e como saída obtemos o córpus de teste etiquetado pelo etiquetador, junto com as medidas de eficiência, como tempo total de execução e taxa de palavras etiquetadas corretamente.

A estrutura que funciona como núcleo para os dois etiquetadores é a classe **Automaton**, mas ela é implementada de maneiras diferentes para cada um deles. Esta classe é a que armazena as diversas associações entre palavras e etiquetas, e suas respectivas probabilidades. Ela obtém estas associações e probabilidades a partir do córpus de treino, recebido como entrada, e as fornece às

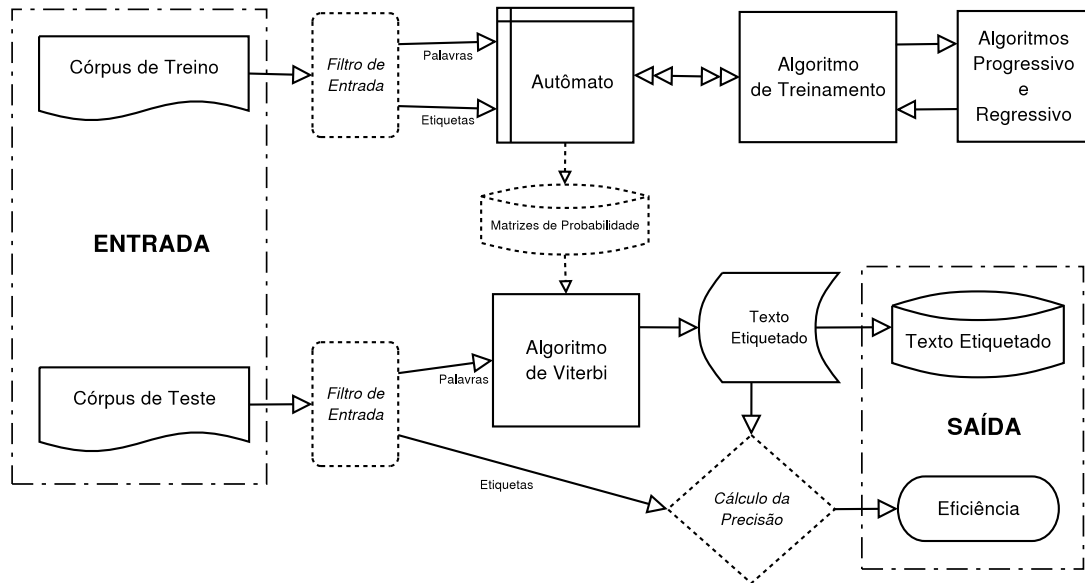


Figura 6.2: Arquitetura do HMM Tagger.

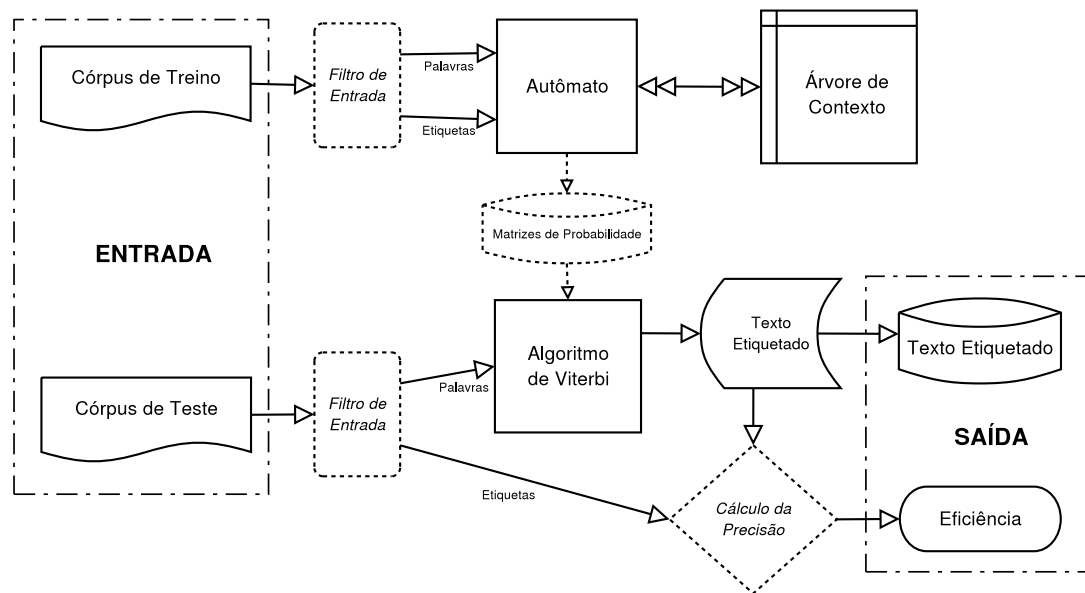


Figura 6.3: Arquitetura do VLMC Tagger.

demais classes do programa. A maneira como ela faz isto depende do modelo de Markov utilizado, por isto vamos explicar as diferenças nas duas próximas seções. A implementação desta classe será explicada na Seção 6.3.1.

Para etiquetar um texto, os dois etiquetadores utilizam a mesma classe chamada `AlgoViterbi`, que contém a implementação do algoritmo de Viterbi, descrito na Seção 4.2.1. As probabilidades das palavras e etiquetas que `AlgoViterbi` precisa são obtidas da classe `Automaton`, ou então a partir de um arquivo contendo as probabilidades já calculadas, para o caso de se desejar executar apenas a fase da etiquetagem. Em qualquer um dos casos, a maneira como estas probabilidades

são calculadas é transparente à classe `AlgoViterbi`. Assim, esta última classe é praticamente a mesma para os dois etiquetadores. Vamos descrever sua implementação na Seção 6.3.2.

Por último, o tratamento das palavras desconhecidas é o mesmo nos dois etiquetadores. Uma implementação diferente para cada um deles poderia prejudicar nosso objetivo de avaliar os modelos de Markov utilizados. A classe que implementa o tratamento das palavras desconhecidas é chamada de `AffixTree`, e será explicada na Seção 6.3.3.

## 6.2.2 Arquitetura do HMM Tagger

O etiquetador com cadeias de Markov de ordem 2 possui duas classes exclusivas, não existentes no etiquetador com cadeias de Markov de tamanho variável. A primeira é a classe que executa o treinamento das probabilidades, que chamamos de `AlgoTraining`. Ela implementa o algoritmo de treinamento de *Baum-Welch*, descrito na Seção 4.3, que modifica os valores  $\lambda$ 's armazenados em `Automaton`.

A segunda classe, chamada `AlphaBeta`, é a que implementa os algoritmos progressivo e regressivo, descritos na Seção 4.1. Ela é utilizada pela classe de treinamento para obter a probabilidade de uma determinada seqüência de etiquetas estar associada a uma determinada sentença (seqüência de palavras). As respectivas implementações destas duas classes serão mostradas nas Seções 6.3.5 e 6.3.4.

Além destas duas classes exclusivas, na classe `Automaton` do **HMM Tagger**, as matrizes de probabilidade e também outras estruturas são implementadas de forma diferente do que no **VLMC Tagger**. Estes detalhes serão mostrados quando explicarmos a implementação de cada um dos etiquetadores.

## 6.2.3 Arquitetura do VLMC Tagger

Conforme vimos na Seção 5.1, o etiquetador baseado em cadeias de Markov de tamanho variável utiliza uma árvore de contexto para representar as probabilidades de transição da cadeia. Para implementar esta árvore, utilizamos a classe chamada `ContextTree`, que é criada e acessada por `Automaton` do **VLMC Tagger**. É esta classe que implementa o algoritmo de contexto descrito na Seção 5.2.

Esta classe substitui as classes `AlgoTraining` e `AlphaBeta` do etiquetador de ordem 2, já que o modelo baseado em cadeias de Markov de tamanho variável não precisa passar por nenhum treinamento semelhante ao feito no modelo de ordem 2. Podemos considerar o processo de poda da árvore de contexto no modelo variável como um processo análogo ao do treinamento no modelo de ordem fixa 2.

Assim, esta é única classe exclusiva ao **VLMC Tagger**. Seus detalhes de implementação serão mostrados na Seção 6.3.6.

## 6.3 Implementação dos Etiquetadores

Para implementar os dois etiquetadores utilizamos a linguagem de programação C++. Na implementação das estruturas de dados e dos algoritmos sobre elas, fizemos um extenso uso da biblioteca padrão de C++, chamada STL (do inglês, *Standard Template Library*) [28, 29]. A STL é uma biblioteca de classes recipientes, algoritmos, e iteradores, e fornece muitos dos algoritmos e estruturas de dados básicas da ciência da computação. Algumas classes da STL foram bastante utilizadas para implementar nossas estruturas de dados, por isso as descrevemos brevemente a seguir. Para uma descrição mais detalhada de cada uma delas consulte [28].

**vector** Esta classe implementa um vetor de elementos de algum tipo qualquer (por exemplo, números inteiros). Utilizamos ela quando tivemos que acessar aleatoriamente algum elemento seu, ou quando a ordem em que os elementos eram inseridos era importante. Neste caso, esta classe ainda possui a vantagem de o tempo de inserção de um elemento no final do vetor ser constante.

**set** Esta classe implementa um conjunto de elementos de um tipo qualquer. Uma característica principal é que os elementos de um conjunto estão sempre ordenados, e os elementos não se repetem, se a ordenação for estritamente crescente, por exemplo. Assim, a busca por um elemento específico de um conjunto é feita muito mais rapidamente do que em um vetor. Outra vantagem é que a STL implementa as principais operações sobre conjuntos, como união e intersecção.

**map** Esta classe implementa um mapa, ou seja, um conjunto de pares: uma chave e um dado associado a ela. Os elementos (pares) do mapa são ordenados pelas chaves, e portanto podem ser acessados rapidamente. Combinando dois mapas, utilizamos esta classe principalmente para implementar as matrizes de probabilidade.

A seguir explicamos como foram implementadas as classes utilizadas pelos etiquetadores. As Figuras 6.4 e 6.5 mostram os módulos correspondentes das arquiteturas apresentadas que estas classes implementam.

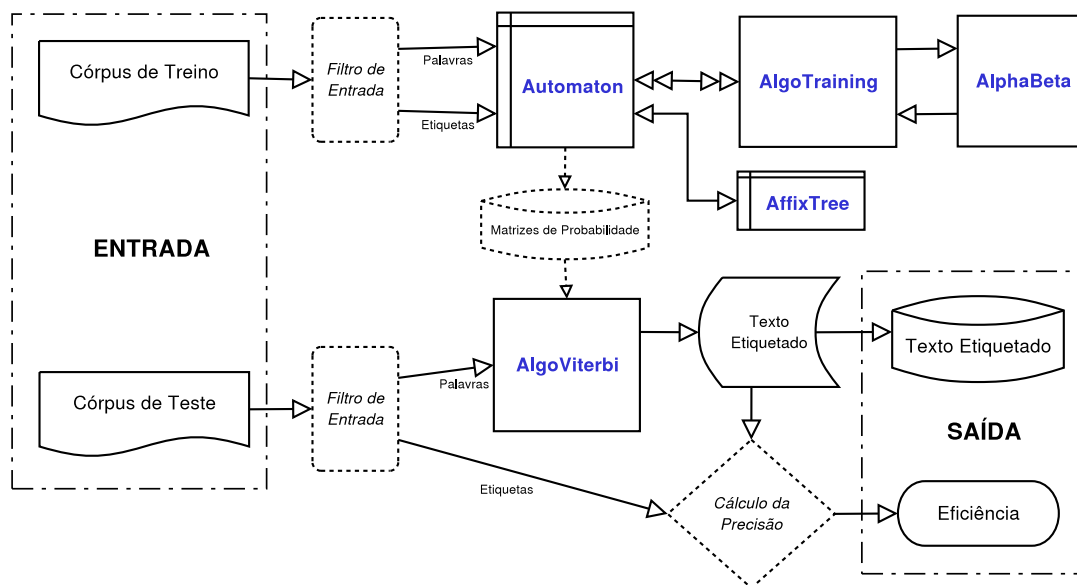


Figura 6.4: Classes implementadas do HMM Tagger.

### 6.3.1 Classe Automaton

A classe Automaton lê o cópus de treino etiquetado e cria várias matrizes de probabilidade. Para isto, primeiro ela preenche estas matrizes com as frequências totais obtidas do cópus. Depois, ela normaliza estas frequências, obtendo as distribuições de probabilidade.

Uma destas matrizes, comum aos dois etiquetadores, é a que contém as probabilidades entre etiquetas e palavras, ou seja,  $P(w^j|l^i)$ . Ela é implementada utilizando mapas.

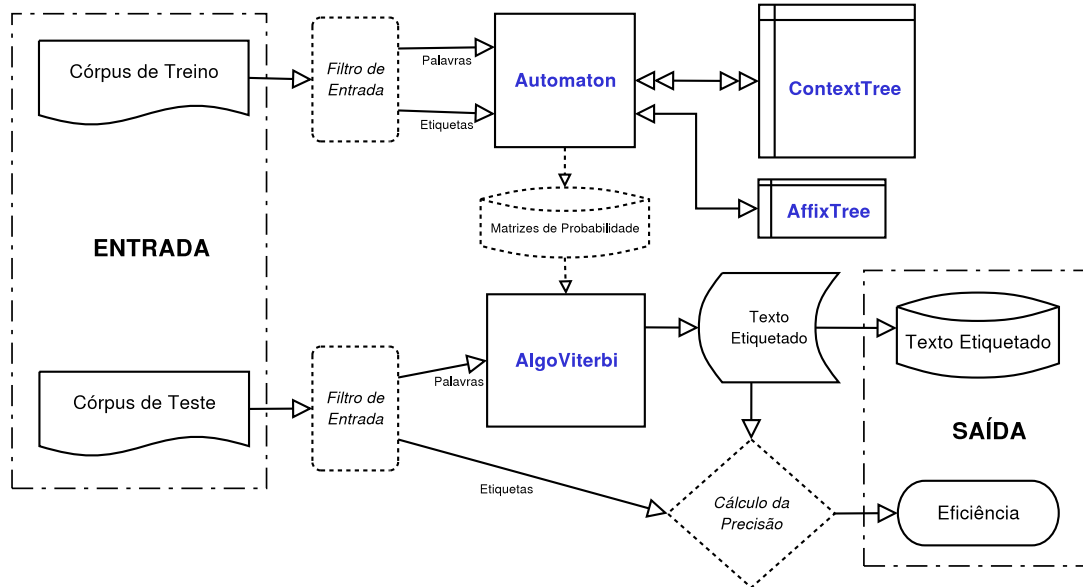


Figura 6.5: Classes implementadas do VLMC Tagger.

Outra estrutura comum é a lista das **etiquetas fechadas**, ou seja, das etiquetas que não aceitam ou dificilmente vão estar associadas a palavras desconhecidas (veja a Seção 6.1.1.2). Por exemplo, a etiqueta DET (Determinante). Existe um número limitado e conhecido de determinantes. Então a menos que o cópulo de treinamento seja muito pequeno, nenhuma palavra desconhecida será um determinante. Esta lista é criada dinamicamente depois da leitura do cópulo de treino: as etiquetas que estão associadas a uma quantidade de palavras distintas menor que um determinado número  $\mathcal{M}$  são consideradas fechadas. Este número  $\mathcal{M}$  é determinado empiricamente, e neste momento está definido como 10. As etiquetas pertencentes ao conjunto de etiquetas que não estão presentes na lista de etiquetas fechadas são consideradas **etiquetas abertas**. Consideramos apenas as etiquetas abertas como possíveis de serem atribuídas a palavras desconhecidas. Estas duas listas são implementadas como vetores.

As demais estruturas da classe, como as matrizes de probabilidade entre etiquetas, são diferentes para os dois etiquetadores. No caso do **HMM Tagger**, existem três matrizes diferentes: a de ordem um, que fornece a probabilidade de uma etiqueta seguir outra ( $P(l^j|l^i)$ ); a de ordem dois, que fornece a probabilidade das seqüências de três etiquetas ( $P(l^k|l^i, l^j)$ ); e a matriz de interpolação, que contém os valores dos  $\lambda$ 's para os conjuntos de etiquetas, conforme explicado na Seção 6.1.1.1.

No caso do **VLMC Tagger**, a classe `Automaton` transmite as freqüências das seqüências de etiquetas para a classe `ContextTree`, que implementa a árvore de contexto (Seção 5.1), normalizando estas freqüências.

### 6.3.2 Classe `AlgoViterbi`

A classe `AlgoViterbi` recebe uma seqüência de palavras e, consultando a classe `Automaton`, devolve a seqüência de etiquetas com maior probabilidade, conforme explicado na Seção 4.2.1. O algoritmo é o mesmo para os dois etiquetadores, e sua implementação também é quase idêntica, já que quem trata os detalhes do modelo de cada etiquetador é a classe `Automaton`.

A `AlgoViterbi` possui uma estrutura formada por uma seqüência de etiquetas e a pro-

babilidade desta seqüência. Com um conjunto de estruturas deste tipo, a classe armazena as sub-seqüências de etiquetas mais prováveis até a  $t$ -ésima palavra. Quando alcança a última palavra da seqüência, a classe retorna a seqüência de etiquetas com maior probabilidade dentre as contidas neste conjunto.

Ao longo deste processo, algumas medidas como as mostradas na Seção 4.2.1 são tomadas, para que o número de sub-seqüências não seja exponencial. Além disto, a seqüência de palavras recebida cada vez é constituída por uma sentença do córpus de teste, ou seja, termina sempre com um símbolo de pontuação final.

### 6.3.3 Classe AffixTree

Esta classe implementa o tratamento das palavras desconhecidas, descrito na Seção 6.1.1.2. Ela armazena em uma árvore os sufixos das palavras do córpus de treino que possuem uma etiqueta aberta, junto com a probabilidade de cada sufixo. Cada nó representa um caractere, e os sufixos são armazenados em ordem inversa, da raiz da árvore até um nó interno. Assim, para o sufixo *ável*, partindo da raiz, iríamos primeiro para o nó filho com o caractere *l*, deste para o nó com *e*, depois *v* e então *á*. A cada nó também é associada uma lista de etiquetas abertas com suas respectivas probabilidades, o que representa as probabilidades de que as etiquetas da lista de um determinado nó sejam atribuídas a uma palavra que tenha o sufixo formado pelo caminho deste nó até a raiz.

Para escolher as etiquetas possíveis para uma palavra desconhecida, considera-se sempre o maior sufixo da palavra que é reconhecido pela árvore. Assim, para a suposta palavra desconhecida *possível*, se, seguindo os nós como no exemplo acima, chegamos ao nó *v* e não encontramos nenhum nó filho *í*, consideramos o sufixo de *possível* como *vel*, e analisamos as etiquetas e probabilidades contidas neste nó *v*.

Construímos esta árvore a partir das palavras do córpus de treino que possuem etiquetas abertas. Entretanto, precisamos definir quais são os sufixos destas palavras. Empiricamente, escolhemos utilizar a última metade das palavras com etiqueta aberta como sufixo, mas este parâmetro pode ser definido no momento da execução do etiquetador (veja o Apêndice B).

### 6.3.4 Classe AlphaBeta

Esta classe implementa o algoritmo progressivo (alpha) e o algoritmo regressivo (beta), como descrito na Seção 4.1. Ela recebe uma seqüência de palavras e uma de etiquetas, consulta a classe `Automaton` sobre suas probabilidades, e calcula com que probabilidade a seqüência de etiquetas se associa com a de palavras. Esta probabilidade é usado pelo algoritmo de treinamento.

### 6.3.5 Classe AlgoTraining

A classe `AlgoTraining` implementa o algoritmo de treinamento do autômato do etiquetador de ordem 2, ou seja, o algoritmo de Baum-Welch da Seção 4.3. Mas como explicado na Seção 6.1.1.1, decidimos fazer o treinamento dos valores lambdas usados para interpolação, deixando com que as probabilidades de etiquetas e palavras obtidas do córpus de treino não sejam modificadas.

`AlgoTraining` utiliza principalmente mapas para implementar suas estruturas de dados, já que elas são, na maioria das vezes, matrizes de probabilidade. Ela executa uma iteração do treinamento sobre uma cópia do autômato inicial, reestimando as probabilidades dos lambdas nesta cópia. Se, pelo resultado do algoritmo progressivo, a probabilidade total do córpus de treino for maior utilizando o autômato recém treinado do que utilizando o inicial, o autômato inicial é atualizado com as probabilidades reestimadas. Este processo é repetido enquanto a melhora obtida entre os dois autômatos é maior do que um valor pré-definido. Definimos empiricamente este valor como 50%, ou seja, se a diferença entre a probabilidade total do córpus dada pelo autômato recém

treinado e a dada pelo autômato inicial for menor do que 50% da dada pelo autômato inicial, o treinamento pára.

### 6.3.6 Classe ContextTree

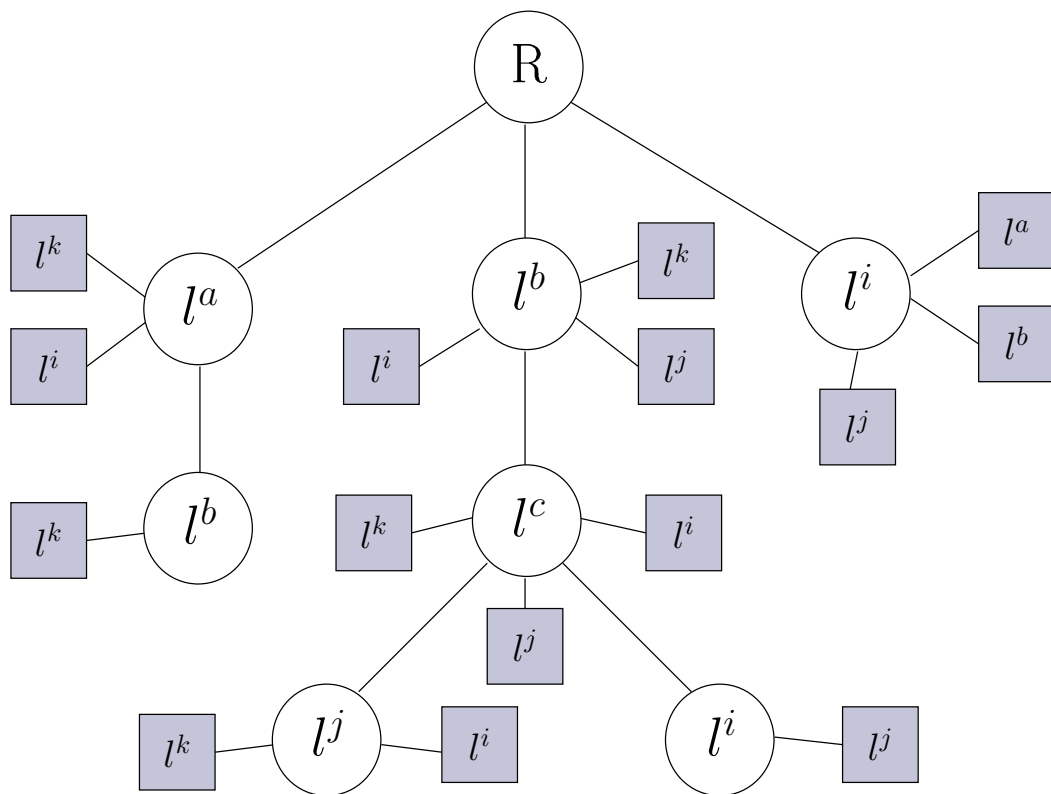
Através desta classe implementamos a árvore de contexto do **VLMC Tagger**, representando os estados da VLMC (Seção 5.1). Cada nó da árvore representa uma etiqueta, e a árvore é construída a partir da classe **Automaton**, que fornece as seqüências de etiquetas encontradas no cópuz de treinamento. **ContextTree**, então, organiza estas seqüências, construindo a árvore de cima para baixo, da etiqueta mais recente de cada seqüência até a mais antiga. Ou seja, se a seqüência de etiquetas é  $l_{t-3}, l_{t-2}, l_{t-1}, l_{t-3}$  sendo a etiqueta mais antiga, e  $l_{t-1}$  a mais recente, **ContextTree** adiciona  $l_{t-1}$  como nó filho da raiz, depois  $l_{t-2}$  como nó filho de  $l_{t-1}$ , e então  $l_{t-3}$  como nó filho de  $l_{t-2}$ .

Definimos o tamanho máximo das seqüências acima como 10 (mas outro valor pode ser especificado no momento de execução do etiquetador; veja detalhes no apêndice B). Deste modo, conforme a Equação 5.1, o **VLMC Tagger** é um etiquetador baseado em cadeia de Markov de tamanho variável de ordem 10. Este valor foi definido empiricamente.

Cada nó da árvore também possui em sua estrutura, da mesma maneira que a classe **AffixTree**, uma lista de etiquetas com probabilidades, que representam com que probabilidade cada etiqueta da lista de um determinado nó possui como contexto a seqüência de etiquetas formada pelo caminho da raiz até este nó. Chamamos as etiquetas desta lista de *brotos*, para mantermos uma nomenclatura associada a árvores. Assim, seguindo o exemplo do parágrafo anterior, considerando que a próxima etiqueta da seqüência é  $l_t$ , o nó  $l_{t-3}$  possuirá como broto a etiqueta  $l_t$ , associada a alguma probabilidade. Além disso,  $l_{t-2}, l_{t-1}$  e  $l_{t-1}$  também são contextos de  $l_t$ , e portanto os nós acima  $l_{t-2}$  e  $l_{t-1}$  também devem incluir como broto a etiqueta  $l_t$ .

Para fixar melhor esta idéia, veja um exemplo de uma árvore de contexto na Figura 6.6. Os brotos são representados pelos quadrados, e os círculos são os nós. A raiz da árvore é um nó especial, representada por **R**. Para visualizar o exemplo do parágrafo anterior, considere que a etiqueta em  $l_{t-3}$  é a etiqueta  $l^i$ , a em  $l_{t-2}$  é  $l^c$ , em  $l_{t-1}$  é  $l^b$ , e em  $l_t$  é  $l^j$ . Então, a seqüência dada anteriormente por  $l_{t-3}, l_{t-2}, l_{t-1}, l_t$  é a seqüência  $l^i, l^c, l^b, l^j$ . Colocada esta seqüência na árvore, podemos encontrá-la saindo da raiz em **R** e indo para o nó  $l^b$ , depois para o nó  $l^c$  e então para o nó  $l^i$ , que finalmente contém o broto  $l^j$ .

Para calcular as probabilidades dos brotos de cada nó, **ContextTree** primeiro constrói a árvore somando a freqüência das seqüências de etiquetas recebidas de **Automaton**. Depois, os nós que possuem freqüência somente 1 são excluídos da árvore. Desta maneira, somente seqüências de etiquetas que ocorrem pelo menos duas vezes no cópuz de treino são consideradas, conforme estabelecemos no **Passo 1** do Algoritmo 5.1. As freqüências do restante da árvore são, então, normalizadas, criando distribuições de probabilidade.



**Figura 6.6:** Exemplo de uma Árvore de Contexto.

---

## Testes e Resultados

Com os etiquetadores implementados, selecionamos os mesmos córpus de treino e de teste para executar sobre eles. Utilizamos vários parâmetros diferentes para testá-los, os quais descrevemos na Seção 7.1. Na Seção 7.2 comparamos e analisamos os resultados obtidos.

### 7.1 Testes

Para treinar e testar nossos etiquetadores, utilizamos o córpus *Tycho Brahe* [17], que contém vários textos do português histórico etiquetados manualmente no formato  $\langle \textit{palavra} \rangle / \langle \textit{etiqueta} \rangle$ . O conjunto de etiquetas utilizadas no córpus está listado no Apêndice A. Destes textos disponíveis, selecionamos alguns específicos para compor o córpus de treino e de teste do nosso trabalho. Não selecionamos todos porque alguns deles estão na ortografia original, muito diferente da encontrada na maioria dos outros textos. Se os selecionássemos, teríamos alguns “ruídos” nas distribuições de probabilidade dos etiquetadores, provavelmente denegrindo seu desempenho.

Citamos abaixo os autores dos textos selecionados para compor nossos córpus, em ordem cronológica.

- (1517-1584) FRANCISCO DE HOLANDA – *Da Pintura Antiga* (Número de palavras: 55691)
- (1542-1606) DIOGO DO COUTO – *Décadas* (Número de palavras: 53916)
- (1556-1632) LUIS DE SOUSA – *A vida de Frei Bertolameu dos Mártires* (Número de palavras: 59158)
- (1579-1621) FRANCISCO RODRIGUES LOBO – *Côrte na Aldeia e Noites de Inverno* (Número de palavras: 59686)
- (1601-1667) MANUEL DA COSTA – *A Arte de Furtar* (Número de palavras: 58400)
- (1608-1697) ANTONIO VIEIRA – *Cartas* (Número de palavras: 57823)  
– *Sermões* (Número de palavras: 62387)
- (1608-1666) FRANCISCO MANUEL DE MELO – *Cartas Familiares* (Número de palavras: 58118)
- (1631-1682) ANTONIO DAS CHAGAS – *Cartas Espirituais* (Número de palavras: 57429)
- (1644-1710) MANUEL BERNARDES – *Nova Floresta* (Número de palavras: 59116)
- (1651-1735) JOSÉ DA CUNHA BROCHADO – *Cartas* (Número de palavras: 34743)

- (1675-1754) ANDRÉ DE BARROS – *A Vida do Padre António Vieira* (Número de palavras: 50207)
- (1695-? ) ALEXANDRE DE GUSMÃO – *Cartas* (Número de palavras: 31235)
- (1702-1783) CAVALEIRO DE OLIVEIRA – *Cartas* (Número de palavras: 53672)
- (1705-1763) MATIAS AIRES – *Reflexão sobre a Vaidade dos Homens e Carta sobre a Fortuna* (Número de palavras: 66722)
- (1713-1792) LUIZ ANTONIO VERNEY – *Verdadeiro Método de Estudar* (Número de palavras: 54952)
- (1724-1772) CORREIA GARCAO – *Obras Completas* (Número de palavras: 26845)
- (1750-1839) MARQUESA D'ALORNA – *Cartas e outros Escritos* (Número de palavras: 49881)
- (1799-1854) ALMEIDA GARRETT – *Viagens na minha terra* (Número de palavras: 51474)
- (1836-1915) RAMALHO ORTIGAO – *Cartas a Emília* (Número de palavras: 34138)

Juntando todos os textos acima, obtivemos um total de 1035593 palavras e etiquetas. Para gerar os corpú de treino e de teste, separamos aleatoriamente, de cada texto, três quartos das sentenças para o corpú de treino e o quarto restante para o corpú de teste, ou seja, 75% das sentenças dos texto acima formaram o corpú de treino, e os 25% restante formaram o corpú de teste. Com isto, os resultados obtidos são mais consistentes, pois avaliamos melhor a eficiência dos etiquetadores não testando-os com algo que foi usado para treiná-los. Obtivemos, assim, um corpú de treino contendo 775602 palavras/etiquetas, e um corpú de teste contendo 259991 palavras/etiquetas.

Testamos os etiquetadores com estes corpú, gerando diversas informações durante a execução de cada um, afim de compará-los melhor. Para obter mais dados sobre o comportamento dos etiquetadores, executamos conjuntos de testes variando o tamanho do corpú de treino, mas utilizando sempre o corpú de teste sem alterações. Assim escolhemos aleatoriamente 5% do corpú de treino e treinamos um etiquetador, testando ele com o corpú de teste e guardando o resultado obtido. Depois repetimos este processo 9 vezes, sempre selecionando aleatoriamente 5% do corpú de treino. Então também fizemos 10 iterações para 10%, 15%, 20%, 25%, ..., 85%, 90% e 95% do corpú de treino. E por último executamos uma vez utilizando todo o corpú de treino. Fizemos estes testes nos dois etiquetadores, mas a cada execução cada etiquetador escolhia aleatoriamente a porcentagem correspondente do corpú de treino.

As execuções foram feitas numa máquina com processador Intel Pentium 4 de 3 GHz e com 1 GB de memória RAM. Os etiquetadores foram compilados com o compilador g++ versão 3.3.4.

## 7.2 Resultados

Com os resultados obtidos a partir das execuções descritas na seção anterior, geramos alguns gráficos e tabelas para mostrar melhor o que obtivemos. Nas próximas seções vamos mostrar estes gráficos e tabelas e explicá-los um a um.

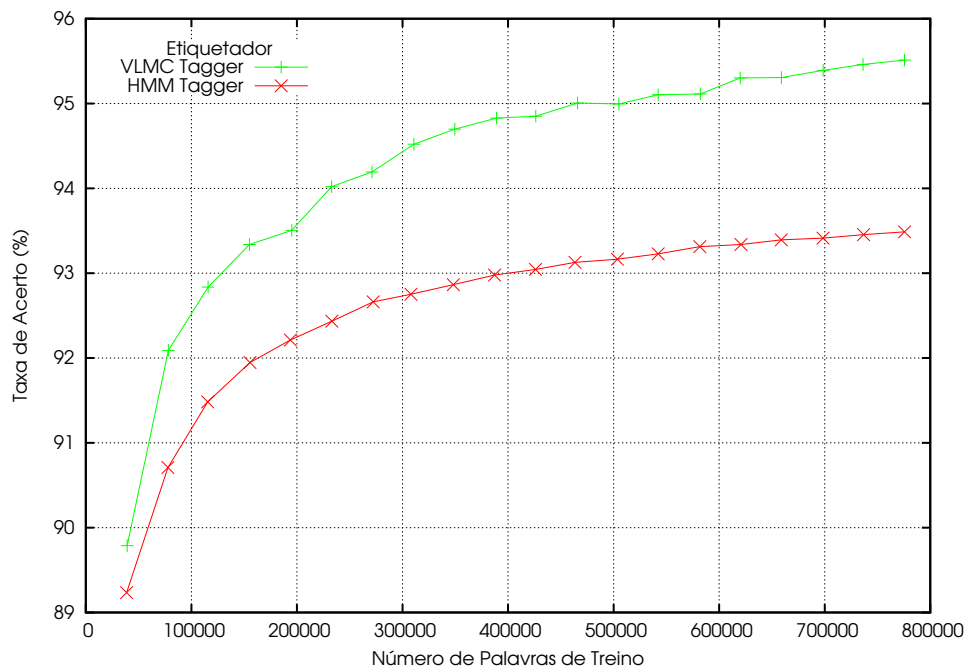
### 7.2.1 Precisão

Vamos começar com a taxa de acerto de cada etiquetador. Colocamos o acerto dos etiquetadores em um mesmo gráfico, mostrado na Figura 7.1. Por taxa de acerto ou precisão queremos

dizer a proporção de palavras do corp us de teste  s quais o etiquetador atribuiu a etiqueta correta, ou seja,

$$\text{Precis o} = \text{Taxa de acerto} = \frac{\text{N mero de etiquetas coincidentes}}{\text{N mero total de etiquetas}}$$

A primeira conclus o  bvia   esta: o **VLMC Tagger** acerta mais do que o **HMM Tagger**.



**Figura 7.1:** Efici ncia do **HMM Tagger** e do **VLMC Tagger**.

Al m disto, ele faz isto com uma boa taxa de acerto: 95,5129% (contra 93,4875%). Este   um bom resultado, que n o era poss vel prever no in cio do projeto. Mas ainda temos que analisar outros resultados antes de podermos afirmar que o etiquetador de ordem vari vel (**VLMC Tagger**)   melhor que o de ordem fixa (**HMM Tagger**), como por exemplo, o tempo de execu o gasto para alcan ar esta taxa de acerto. Mesmo assim, podemos enxergar mais um dado interessante no gr fico da Figura 7.1: a curva do **VLMC Tagger** cresce mais r pido do que a do **HMM Tagger**. Baseados nesta informa o, podemos dizer que, mesmo aumentando o n mero de palavras do corp us de treino, o etiquetador de ordem fixa n o alcan ar  a taxa de acerto do de ordem vari vel.

Os gr ficos das Figuras 7.2 e 7.3 mostram os acertos de cada itera o de teste descritas na se o anterior. Na Figura 7.2 temos a distribui o das taxas de acerto dos testes com o **VLMC Tagger**. Cada ponto ao redor da linha indica a taxa de acerto de uma itera o de teste, e a linha em si passa pela m dia da taxa de acerto de cada conjunto de itera es.

Vemos que quanto maior o n mero de palavras do corp us de treino, menor a diferen a de resultado entre itera es de teste de um mesmo conjunto. Em outras palavras, o desvio padr o tende a diminuir. Isto pode ser visto na Tabela 7.1, que mostra a m dia de acertos e o desvio padr o obtidos nos conjuntos de testes.

O gr fico da Figura 7.3 mostra a distribui o das taxas de acerto obtidas nos testes com o **HMM Tagger**. Como no caso do **VLMC Tagger**, os pontos ao redor da linha indicam as taxas de acerto das itera es de teste, e a linha indica a m dia de cada conjunto de itera es.

Com o **HMM Tagger**, podemos ver que o desvio padr o dos acertos de um conjunto de itera es de teste foi menor do que para o **VLMC Tagger**. Compare a Tabela 7.2 com a Tabela 7.1

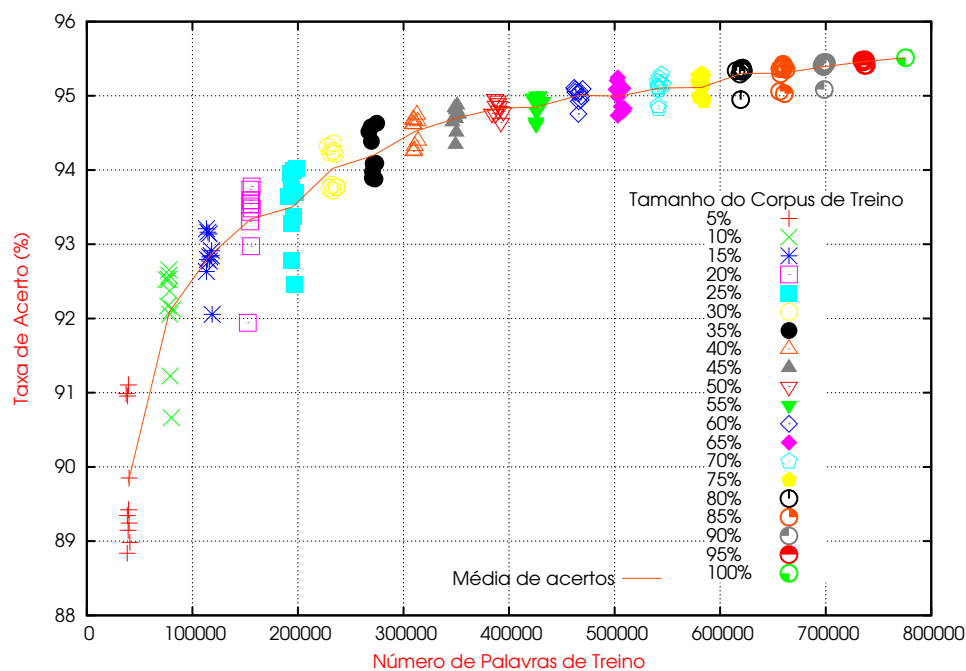


Figura 7.2: Distribuição das taxas de acerto dos testes com o VLMC Tagger.

Tamanho do Córpus de Treino (%)	Taxa de Acerto (%)	
	Média	Desvio Padrão
5	89,7874	0,8904
10	92,0899	0,6506
15	92,8358	0,3333
20	93,3384	0,5413
25	93,5042	0,5324
30	94,0211	0,2733
35	94,1937	0,3047
40	94,5203	0,1918
45	94,6978	0,1651
50	94,8293	0,0975
55	94,8504	0,1592
60	95,0061	0,1068
65	94,9946	0,1804
70	95,1044	0,1486
75	95,1126	0,1432
80	95,3013	0,1256
85	95,3065	0,1426
90	95,3922	0,1093
95	95,4613	0,0275

Tabela 7.1: Média de acertos e desvio padrão dos testes com o VLMC Tagger.

já mostrada. Apenas com 95% do córpus de treino é que o desvio padrão dos dois etiquetadores fica próximo. Esta observação nos leva a concluir que o etiquetador de tamanho variável é mais sensível ao córpus de treino do que o de tamanho fixo. Embora a média de acertos do variável seja maior do que a do fixo para todos os tamanhos de córpus de treino utilizados, determinadas

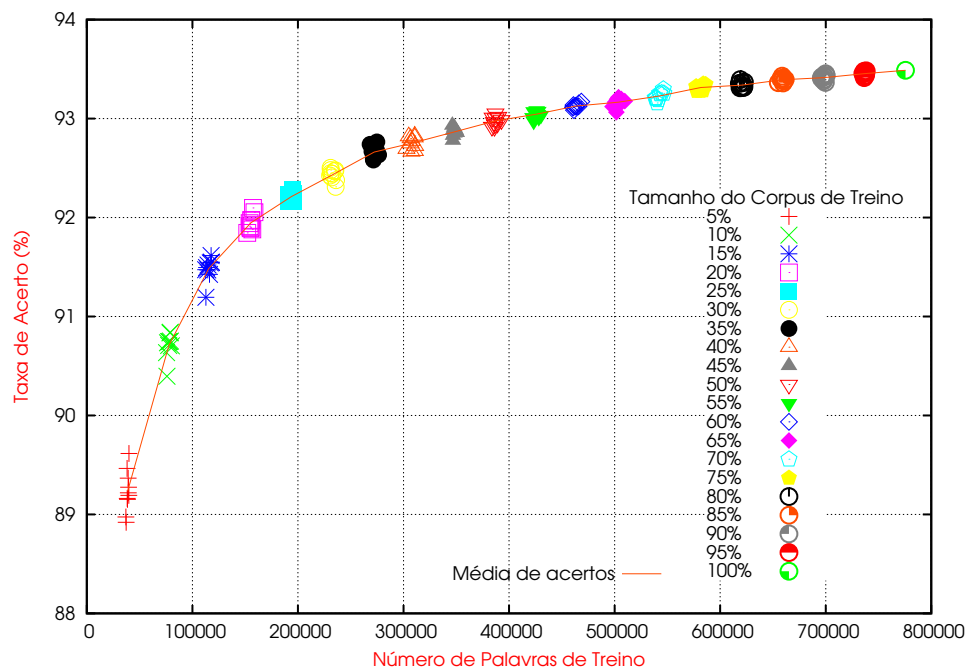


Figura 7.3: Distribuição das taxas de acerto dos testes com o HMM Tagger.

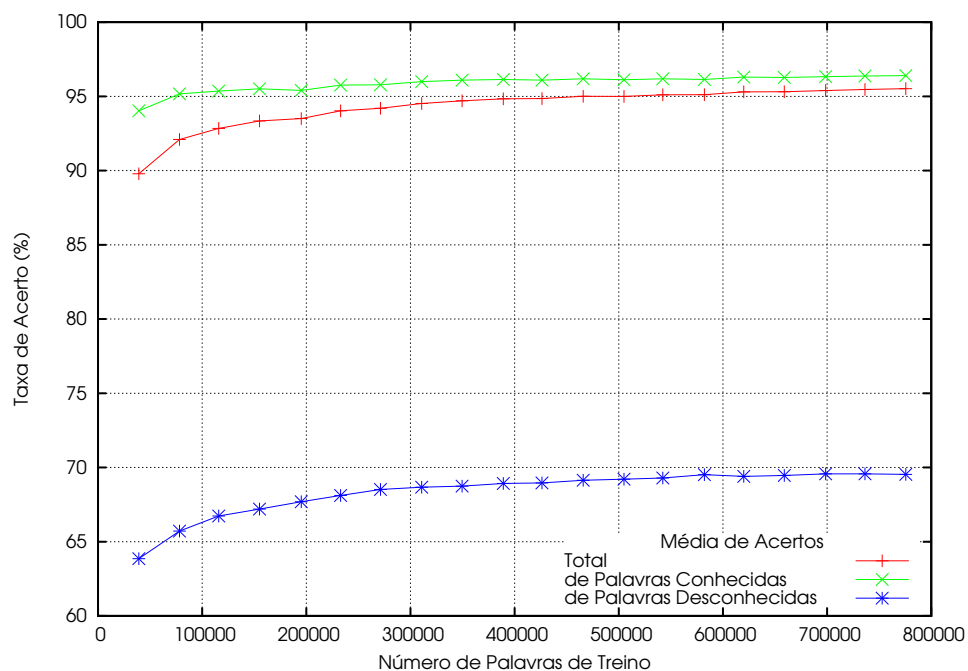
iterações de teste no variável obtiveram resultados bastante melhores ou bastante piores do que a média, indicando a existência de sentenças que favorecem o aprendizado do etiquetador ou então que prejudicam. Em qualquer caso, o etiquetador de tamanho fixo parece ser pouco afetado por combinações de diferentes sentenças, o que pode ser explicado pelo fato dele considerar contextos de apenas três palavras. Com contextos maiores as possíveis seqüências de palavras aumentam bastante, assim ocorrências de determinadas seqüências que influenciam o resultado positiva ou negativamente podem aparecer com freqüências significantes ou insignificantes, de acordo com a composição do córpus de treino. Isto explica o comportamento observado no etiquetador variável.

Agora veja a Figura 7.4. No gráfico que ela contém são exibidas três curvas, que representam a média de acertos total do **VLMC Tagger**, a média de acertos de palavras conhecidas, e a média de acertos de palavras desconhecidas. Como explicamos na Seção 6.1.1.2, o córpus de teste geralmente contém palavras que não existem no córpus de treino, e estas palavras são chamadas de “desconhecidas”. Tentamos etiquetá-las corretamente utilizando, além da informação do contexto, a árvore de sufixos, como explicado na mesma Seção 6.1.1.2. Fazendo desta forma, etiquetamos corretamente 69,5351% das palavras desconhecidas quando usado todo o córpus de treino. Já com apenas 5% do córpus de treino, a média de acertos das palavras desconhecidas foi de apenas 63,97208%, uma diferença de mais de 5%. Considerando o fato de que o número de palavras desconhecidas é maior quando usado um córpus de treino menor, esta diferença representa aproximadamente 4% do córpus de teste, o que significa mais de 10650 erros de etiquetagem. Veja na Tabela 7.3 a quantidade de palavras desconhecidas em relação ao tamanho do córpus de treino. As quantidades de palavras possuem uma parte fracionária porque são valores médios dos testes executados.

Em relação às palavras conhecidas, a média de acertos com 5% do córpus de treino foi de 94,20155%, e com o córpus inteiro foi de 96,3929%. Esta diferença equivale a aproximadamente 3872 palavras, mais ou menos 1,49% do córpus de teste.

Tamanho do Córpus de Treino (%)	Taxa de Acerto (%)	
	Média	Desvio Padrão
05	89,2342	0,2103
10	90,7098	0,1256
15	91,4833	0,1145
20	91,9473	0,0791
25	92,2121	0,0408
30	92,4335	0,0573
35	92,6616	0,0586
40	92,7514	0,0623
45	92,8653	0,0422
50	92,9776	0,0456
55	93,0440	0,0345
60	93,1272	0,0233
65	93,1653	0,0473
70	93,2282	0,0416
75	93,3133	0,0266
80	93,3382	0,0291
85	93,3925	0,0207
90	93,4136	0,0269
95	93,4547	0,0218

**Tabela 7.2:** Média de acertos e desvio padrão dos testes com o **HMM Tagger**.



**Figura 7.4:** Média das taxas de acerto do **VLMC Tagger** para palavras conhecidas e desconhecidas.

No gráfico da Figura 7.5 vemos as curvas de acerto do **HMM Tagger**: total, de palavras conhecidas, e de palavras desconhecidas. A porcentagem de palavras desconhecidas etiquetadas corretamente varia de 62,37534%, com 5% do córpus de treino, a 68,8109%, com o córpus todo. Uma diferença de seis e meio por cento (6,5%), que no **VLMC Tagger** era de 5%. Isto de novo

Tamanho Córpus Treino	Número de Palavras Desconhecidas				
	Durante Etiquetagem		Etiquetadas Certo		Etiquetadas Errado
5%	14,15%	36789,61	63,97%	23535,07	13254,53
10%	10,59%	27539,57	65,47%	18030,91	9508,66
15%	8,73%	22704,50	66,40%	15078,00	7626,49
20%	7,64%	19877,89	67,29%	13377,25	6500,64
25%	6,83%	17775,60	67,73%	12039,42	5736,17
30%	6,28%	16328,10	68,07%	11115,12	5212,97
35%	5,80%	15101,60	68,20%	10300,64	4800,95
40%	5,41%	14082,00	68,39%	9631,29	4450,70
45%	5,10%	13275,49	68,73%	9125,17	4150,31
50%	4,81%	12514,19	68,98%	8632,38	3881,81
55%	4,57%	11885,59	68,84%	8182,56	3703,02
60%	4,35%	11328,29	69,19%	7839,04	3489,25
65%	4,17%	10866,90	69,18%	7518,33	3348,56
70%	4,00%	10418,29	69,42%	7233,21	3185,08
75%	3,85%	10035,19	69,36%	6961,34	3073,85
80%	3,71%	9669,10	69,43%	6713,37	2955,72
85%	3,60%	9360,00	69,46%	6501,97	2858,02
90%	3,47%	9044,50	69,60%	6295,30	2749,19
95%	3,37%	8772,30	69,63%	6108,19	2664,10
100%	3,27%	8518	69,53%	5923	2595

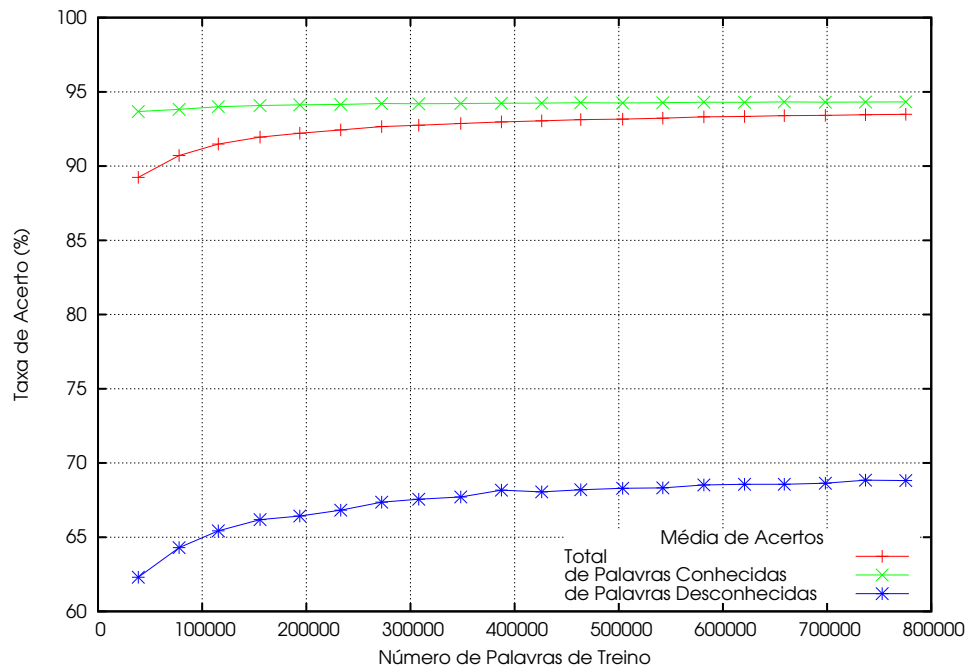
**Tabela 7.3:** Palavras desconhecidas nos testes com o **VLMC Tagger**.

pode ser explicado pelo fato do etiquetador de ordem fixa considerar um contexto muito pequeno, e portanto acabar expondo a escolha da melhor etiqueta de uma palavra desconhecida somente à árvore de sufixos. Por isto o aumento do número de palavras de treino melhora bastante o acerto das palavras desconhecidas, como visto no gráfico.

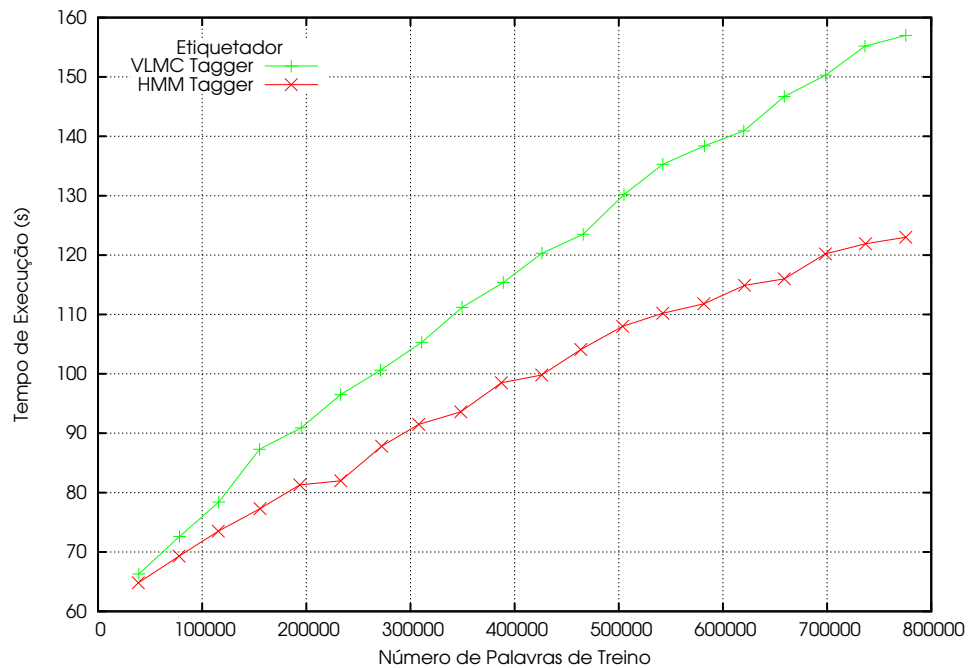
O fato do etiquetador de ordem fixa considerar um contexto bastante pequeno também explica o interessante resultado obtido com as palavras conhecidas, visto na curva mais alta do gráfico da Figura 7.5. A variação na porcentagem de acertos das palavras conhecidas conforme o tamanho do córpus de treino foi pequena: 93,61883% com 5% do córpus de treino, e 94,3234% com o córpus todo. Apenas 0,7% de diferença, contra 2,19% do etiquetador de ordem variável. Ou seja, pelo **HMM Tagger** considerar um contexto pequeno, a etiquetagem das palavras conhecidas é pouco influenciada pela das palavras desconhecidas. Isto nos mostra que, ao mesmo tempo em que o **VLMC Tagger** trata melhor as palavras desconhecidas, pois usa um contexto maior como auxílio, ele também é mais sensível a eventuais erros de etiquetagem, podendo errar a etiqueta de uma palavra por influência do contexto.

### 7.2.2 Tempo

Vamos agora ver os resultados obtidos relativos a tempo. A Figura 7.6 mostra o gráfico com os tempos médios gastos pelos dois etiquetadores em relação ao número de palavras com que foram treinados. Vemos que o tempo total gasto pelo **HMM Tagger** aumenta em relação ao número de palavras de treino numa proporção menor do para o **VLMC Tagger**. Entretanto, se nos basearmos nas curvas do gráfico, podemos dizer que os dois etiquetadores possuem complexidade linear em relação ao tempo. De fato, efetuando o cálculo da correlação entre o número de palavras usadas no treino e o tempo total de execução, encontramos para o **HMM Tagger** uma correlação aproximada de 0,9956, e para o **VLMC Tagger** uma correlação de aproximadamente 0,9969. Ou seja, baseados nos testes feitos, o tempo de execução dos dois etiquetadores em função do número



**Figura 7.5:** Média das taxas de acerto do **HMM Tagger** para palavras conhecidas e desconhecidas.

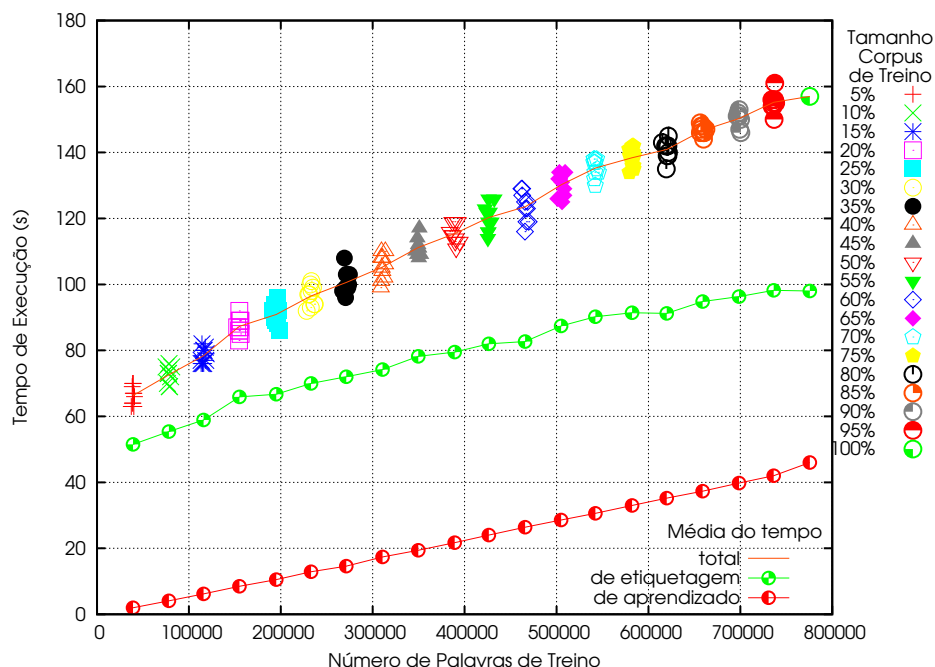


**Figura 7.6:** Média do tempo de execução do **HMM Tagger** e do **VLMC Tagger**.

de palavras usadas para treino é quase exatamente linear. E este é um resultado muito bom. Se o **VLMC Tagger** tivesse complexidade exponencial ou mesmo quadrática de tempo, então provavelmente poderíamos dizer que a melhora da taxa de acerto obtida em relação ao **HMM**

**Tagger** não compensaria o tempo necessário para isto. Entretanto, como não parece ser isto o que ocorre, o **VLMC Tagger** parece ser realmente melhor que o **HMM Tagger**.

Vamos olhar os gráficos de tempo de cada um dos etiquetadores. A Figura 7.7 mostra o gráfico com as curvas dos tempos de execução total, de aprendizado e de etiquetagem gastos pelo **VLMC Tagger**<sup>1</sup>. Vemos que as três curvas apresentam comportamento linear.



**Figura 7.7:** Distribuição dos tempos de execução dos testes com o **VLMC Tagger**.

Compare com a Figura 7.8, que mostra as curvas de tempo de execução, aprendizado e etiquetagem do **HMM Tagger**. Note como o tempo de aprendizado é particularmente baixo. A explicação para isto é que não realizamos nenhuma iteração de treinamento do **HMM Tagger** conforme descrito na Seção 6.1.1.1. Ao invés disto, aprendemos as probabilidades através do cópulo de treino e utilizamos os valores dos lambdas obtidos empiricamente (Seção 6.1.1.1). Fizemos isto porque a precisão do etiquetador caía fazendo o treinamento. E, além disto, o tempo gasto para uma iteração de treino era muito grande. Para exemplificar, executamos duas vezes o **HMM Tagger** com os mesmos dados e parâmetros, apenas alterando para que em uma execução fossem feitas duas iterações de treino e em outra execução, nenhuma. A precisão da etiquetagem sem o treinamento foi, como já vimos, de 93,4875%, e com o treinamento foi de 92,677%, 0,81% menor do que sem treino. Ainda, o tempo total de execução com o treinamento foi de 4622 segundos, 37 vezes maior do que sem treino. Então, mesmo se a precisão aumentasse com o treinamento, o tempo gasto para isto não compensaria. Para complementar, os trabalhos de Church [11] e DeRose [14] também coletam informações estatísticas de um cópulo etiquetado ao invés de utilizarem o processo de treinamento para HMMs.

Desta forma, portanto, as curvas de tempo do **HMM Tagger** também apresentam comportamento linear, e mostram que o tempo de execução é maior no **VLMC Tagger** devido principalmente ao maior tempo gasto no seu aprendizado. Isto porque, além da contagem e normalização

<sup>1</sup>Note que o tempo de execução total é maior do que a soma dos tempos de aprendizado e etiquetagem, porque inclui o tempo tomado por outras operações, como a leitura dos arquivos do cópulo e o cálculo da precisão.

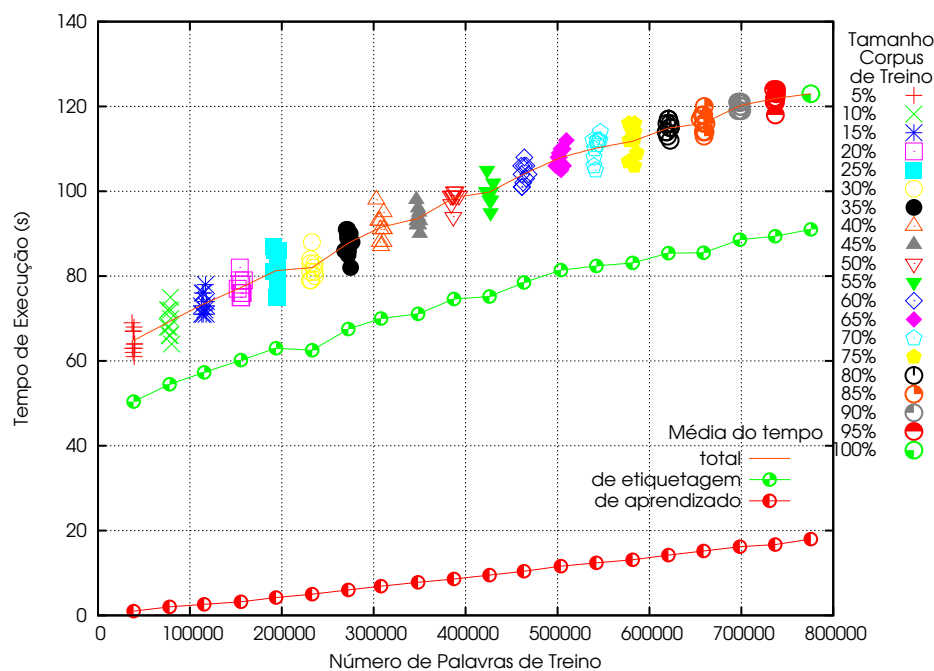


Figura 7.8: Distribuição dos tempos de execução dos testes com o **HMM Tagger**.

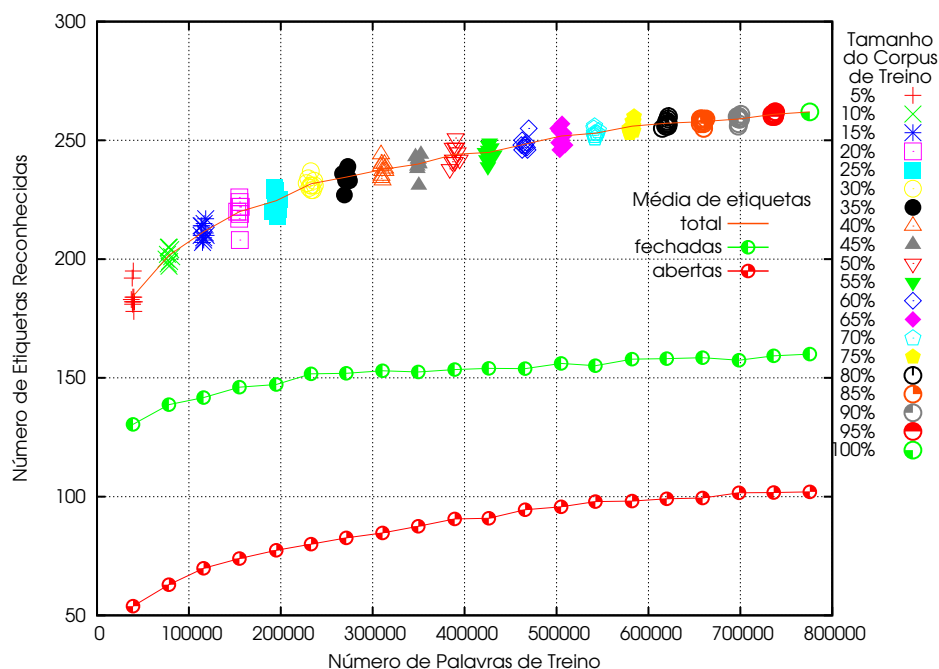
das palavras e etiquetas, que são feitas pelos dois etiquetadores, e da construção do espaço de estados pelo etiquetador fixo e da árvore de contexto pelo variável, o variável gasta tempo no corte da árvore de contexto, uma etapa não necessária para o fixo. Por esta razão é que, com um cópús de treino pequeno, o tempo de aprendizado dos dois etiquetadores é quase o mesmo, e com o cópús inteiro, o tempo de aprendizado do etiquetador variável é 85% maior. Por sua vez, o tempo requerido para a etiquetagem é quase o mesmo nos dois etiquetadores, indicando que o tempo de acesso à árvore de contexto do **VLMC Tagger** é semelhante ao tempo de acesso aos estados do **HMM Tagger**. Portanto, o maior tempo de execução final necessário ao etiquetador variável decorre principalmente do maior tempo necessário ao aprendizado.

O etiquetador baseado no método de Brill empregado por Chacur e Finger [2] para o português toma aproximadamente 5 horas para executar e obter uma taxa de acerto de 95,43% (depois das técnicas de precisão aplicadas por Finger [15]). O tempo gasto pelo **VLMC Tagger** para obter 95,51% de taxa de acerto é de cerca de 157 segundos, um tempo extremamente menor. Comparando, o **VLMC Tagger** toma um tempo mais de 100 vezes menor do que o do etiquetador baseado no método de Brill para o português, o que é um resultado muito bom. Outros resultados são comparados na Seção 7.2.4.

### 7.2.3 Outras Medidas

A Figura 7.9 mostra o gráfico do número de etiquetas reconhecidas pelo **VLMC Tagger**, isto é, a cardinalidade do conjunto  $\mathcal{L}$  de etiquetas (mostrado na tabela 3.1, página 5) ou ainda, o número de etiquetas distintas presentes na porção utilizada do cópús de treino, em relação ao número de palavras desta porção do cópús de treino. Além disto, o gráfico também mostra quantas destas etiquetas são abertas e quantas são fechadas. O que são etiquetas abertas e fechadas foi explicado na Seção 6.1.1.2, e a maneira como elas são separadas entre abertas e fechadas pelos etiquetadores foi mostrado na Seção 6.3.1. Como esta maneira de separar é a mesma para os dois

etiquetadores, e os dois utilizam o mesmo corp us de treino, a distribui o das etiquetas reconhecidas pelo **HMM Tagger**   semelhante   do **VLMC Tagger**. Podemos ver que, como era esperado, na medida em que o corp us de treino cresce, o n mero de etiquetas reconhecidas tamb m cresce. Mas um resultado interessante   que as etiquetas abertas crescem mais r pido que as fechadas: entre 5% do corp us de treino e o corp us inteiro, as etiquetas fechadas cresceram 20% (de 133 a 160 etiquetas), enquanto as abertas cresceram mais de 93% (de 52 a 102 etiquetas). Podemos concluir com isto que etiquetas fechadas ocorrem com bastante homogeneidade entre as senten as de um corp us, ou seja, est o presentes em praticamente qualquer subconjunto de senten as deste corp us.

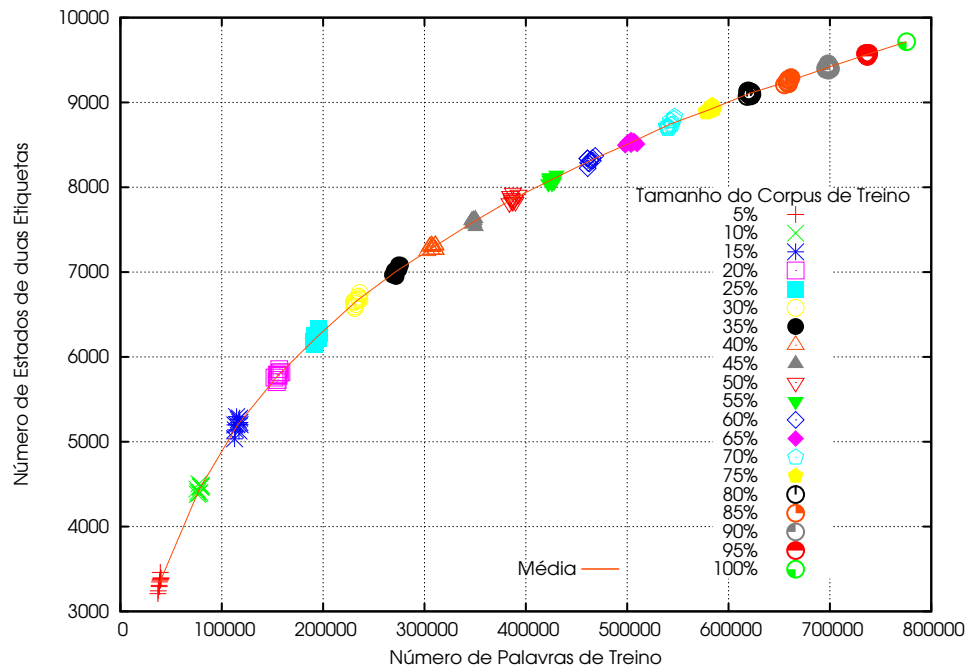


**Figura 7.9:** Distribui o do n mero de etiquetas reconhecidas pelo **VLMC Tagger**.

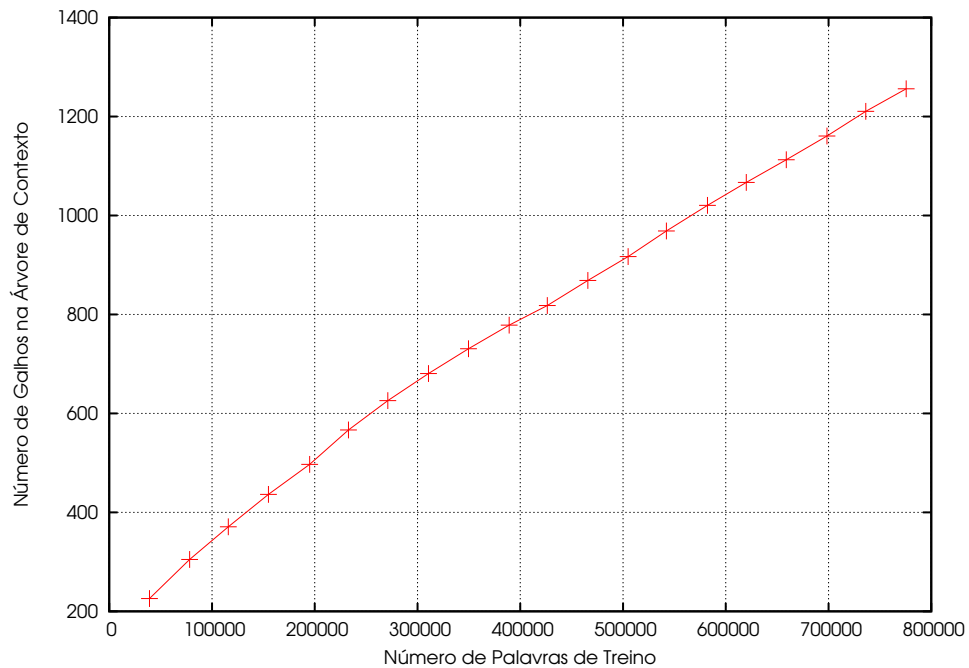
No gr fico da Figura 7.10 vemos a quantidade de estados de duas etiquetas constru da pelo **HMM Tagger** durante a fase de aprendizado. Em outras palavras, esta   a quantidade de diferentes seq ncias de duas etiquetas que ocorrem no corp us de treino.

J  na Figura 7.11 vemos o aumento do n mero de galhos da  rvore de contexto do **VLMC Tagger** em rela o ao n mero de palavras usadas na fase de aprendizado, em uma curva quase linear de correla o aproximada de 0,9975. Da mesma maneira, a Figura 7.12 mostra o crescimento do n mero de n s da  rvore de contexto. Este crescimento tamb m parece ser linear, e de fato apresenta uma correla o de aproximadamente 0,99888. Comparando o n mero de n s com o n mero de estados de duas etiquetas do **HMM Tagger** (Figura 7.10), vemos que o n mero de estados   muito maior do que o de n s, o que mostra que muitas seq ncias de duas etiquetas n o acrescentam informa o  til para o processo de etiquetagem.

Detalhando mais a Figura 7.11, mostramos na Figura 7.13, para os diversos tamanhos do corp us de treino, o n mero de galhos de tamanhos distintos contidos na  rvore de contexto. Nesta Figura vemos o fato interessante de que a maioria dos galhos da  rvore de contexto tem tamanho dois, mesmo que o n mero de galhos maiores tamb m aumente em propor o ao tamanho do corp us de treino. Considerando o corp us inteiro, a  rvore de contexto do **VLMC Tagger** armazena por

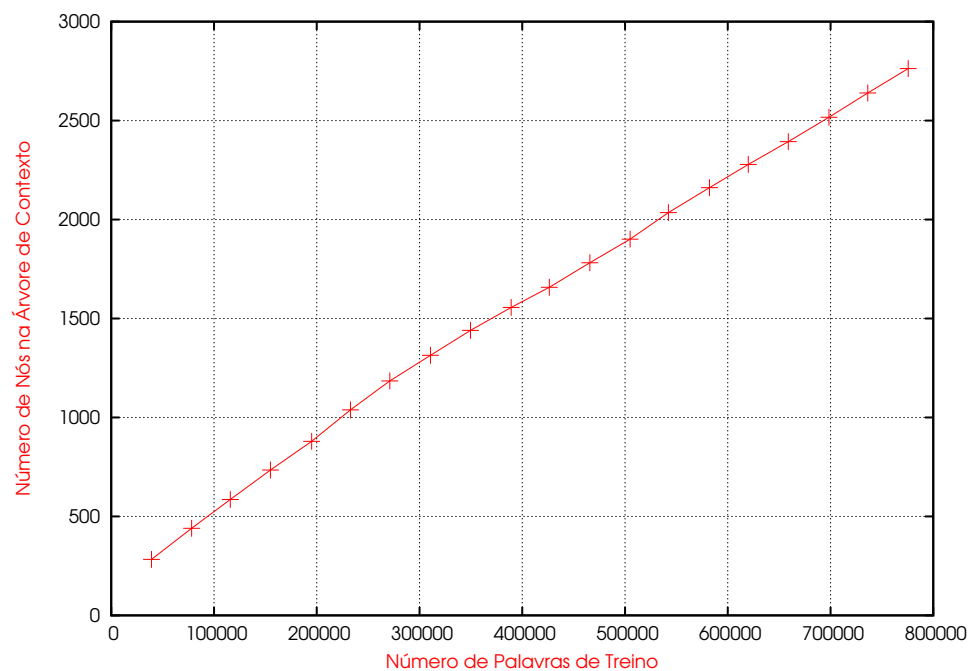


**Figura 7.10:** Distribuição do número de estados de duas etiquetas obtidos pelo HMM Tagger.

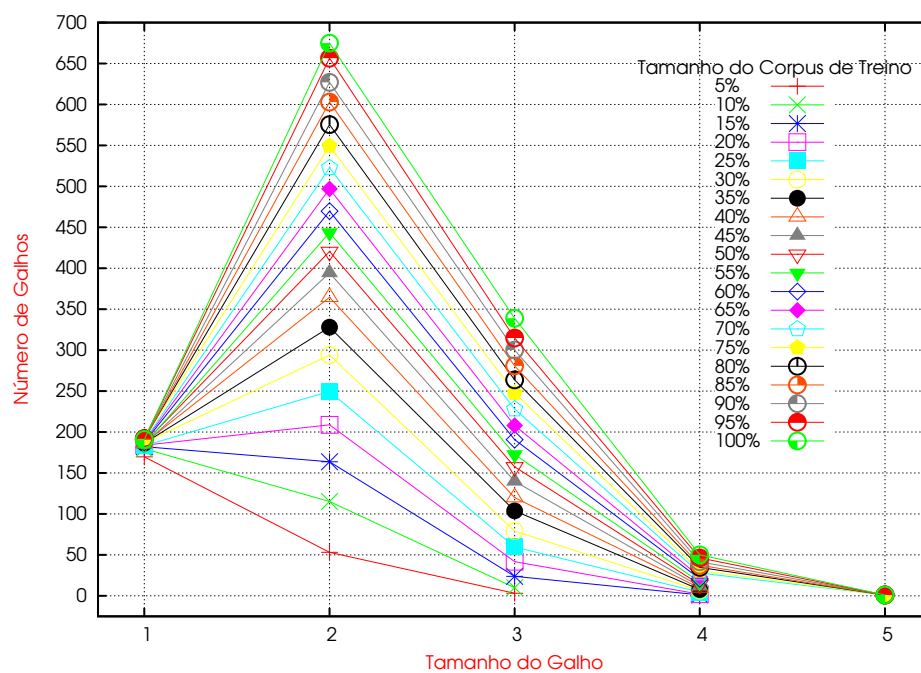


**Figura 7.11:** Crescimento do número de galhos da árvore de contexto do VLMC Tagger.

volta de 675 galhos de tamanho dois, enquanto o HMM Tagger (Figura 7.10) possui mais de 9700 estados de duas etiquetas. Assim, isto reforça o que dissemos no parágrafo anterior, de que muitas seqüências de duas etiquetas do corpus de treino são consequência de dados esparsos, e não



**Figura 7.12:** Crescimento do número de nós da árvore de contexto do VLMC Tagger.



**Figura 7.13:** Proporção de galhos da árvore de contexto do VLMC Tagger com tamanhos diferentes.

acrescentam muito conhecimento relevante para a etiquetagem.

## 7.2.4 Resultados Finais e Comparações

Por fim, como principais resultados, temos o que consta na Tabela 7.4. Antes do nosso

Etiquetador	Precisão	Tempo Aprend. + Etiq.
<b>HMM Tagger</b>	93,4875%	123s
<b>VLMC Tagger</b>	95,5129%	157s

**Tabela 7.4:** Taxa de acertos e tempo de execução dos dois etiquetadores.

trabalho, o melhor etiquetador que utilizava o córpus Tycho Brahe era o de Chacur-Finger [15], que obtinha 95,43% de precisão, tomando para isto em torno de cinco horas. Além disto, ele não utilizava todo conjunto de etiquetas (veja Apêndice A), mas apenas a parte básica delas, sem as inflexões. O **VLMC Tagger**, agora, obtém 95,51% de precisão em menos de três minutos, e utilizando todas etiquetas com as quais o córpus Tycho Brahe foi originalmente anotado.

Em relação a outros etiquetadores para o português, comparamos o **VLMC Tagger** com o MXPOST, originalmente criado por Ratnaparkhi [24] e adaptado para o português por Aires [1]. Aires utilizou no etiquetador o Córpus NILC [16], na época contendo quase 105 mil palavras, com 81 diferentes etiquetas. Para obter uma comparação mais precisa, executamos o **VLMC Tagger** com o mesmo córpus NILC utilizado no MXPOST, também dividindo-o em 90% para aprendizado e 10% para teste. Os resultados comparativos são mostrados na tabela 7.5 Vemos nesta comparação

Etiquetador	Taxa de Acerto	Tempo (Aprend. + Etiq.)	Córpus (tamanho)
MXPOST [1]	90,25%	5307s + 386s	NILC (104.963 palavras)
VLMC Tagger	90,3232%	4s + 4s	NILC (104.963 palavras)

**Tabela 7.5:** Precisão e tempos de aprendizado e etiquetagem dos etiquetadores VLMC Tagger e MXPOST.

que o **VLMC Tagger** possui uma precisão quase igual à do MXPOST. Entretanto, a principal diferença está no tempo tomado para obter esta precisão: enquanto o MXPOST leva 5693 segundos para realizar o aprendizado e a etiquetagem, o **VLMC Tagger** toma apenas 8 segundos para esta mesma tarefa, um tempo em torno de 40 vezes menor. Este resultado destaca a boa eficiência de tempo do **VLMC Tagger**.

Na próxima seção vamos falar de mais alguns resultados e outras conclusões obtidas com os testes dos etiquetadores que construímos, e também vamos expor mais algumas experiências obtidas e realizadas com eles.

## 7.2.5 Outros Experimentos

Durante as implementações e os testes dos etiquetadores realizamos alguns experimentos que achamos válido citar aqui. Primeiro, estávamos utilizando um córpus com 21 textos: os 20 citados na Seção 7.1 mais um texto de Jerônimo Contador de Argote (1976-1749), chamado *Regras da Língua Portuguesa, Espelho da Língua Latina*, de 49194 palavras. Este texto de Argote trata de questões gramaticais, e por causa disso possui uma etiquetagem peculiar, com letras e palavras comuns recebendo a etiqueta de nome próprio (NPR). Por exemplo:

Quando/CONJS o/D nome/N ,/, a/P que/WPRO dizem/VB-P ordem/N ,/, tem/TR-P  
antes/ADV de/P si/PRO o/D artigo/N ao/NPR ,/, a/NPR ,/, aos/NPR ,/, as/NPR  
,/, os/NPR ,/, assim/ADV como/CONJS ,/, Gritei/VB-D aos/P+D-P soldados/N-P  
./.

Note os artigos *ao*, *a*, *aos*, *as* e *os* recebendo a etiqueta NPR. Resolvemos, então, retirar este texto do nosso *cópus*, e observar que resultado era obtido com isto. Obtivemos, respectivamente com o **HMM Tagger** e o **VLMC Tagger**, 93,4875% e 95,5129%, como mostramos na seção anterior, enquanto que com o texto de Argote incluído obtínhamos apenas 93,068% e 95,1708%. Portanto, este texto em particular acrescentava *ruídos* ao *cópus* de treino, prejudicando a precisão dos etiquetadores. Podemos concluir com isto que os dois etiquetadores são sensíveis à qualidade do *cópus*.

Outro experimento que fizemos foi o de tentar melhorar o resultado obtido pelo **VLMC Tagger** utilizando um pós-corretor baseado em regras, ou seja, enviando o *cópus* etiquetado pelo etiquetador (possuindo 95,5129% de etiquetas corretas em relação ao *cópus* de teste) a este pós-corretor, que procuraria então corrigir os erros utilizando conhecimento lingüístico. O pós-corretor que utilizamos foi o implementado por Finger para o seu *Etiquetador Tycho Brahe*, descrito em [15]. Alimentamos este pós-corretor com o *cópus* de teste etiquetado pelo **VLMC Tagger**, e o executamos. Entretanto, ao contrário do que esperávamos, o resultado obtido foi pior: o pós-corretor “corrigiu” erroneamente etiquetas que já estavam corretas em maior quantidade do que etiquetas que estavam erradas e foram corrigidas corretamente. Isto é um indício de que etiquetadores são independentes um do outro, e por isso para melhorar o resultado teríamos que ter um pós-corretor que utilizasse regras específicas criadas com base nos erros do **VLMC Tagger**.

Com base no experimento anterior, alteramos o pós-corretor de Finger para tentar melhorar sua precisão para o *cópus* etiquetado pelo **VLMC Tagger**. As principais alterações que fizemos foram substituir etiquetas compostas por simples (por exemplo, “ao/P+D” por “a/P o/D”), e aglutinar etiquetas semelhantes em classes (N, N-P, NPR e NPR-P em NOME, por exemplo). Com estas alterações, o pós-corretor corrigiu perto de 35% dos erros, atingindo uma taxa de acerto em torno de 97,2%. Um resultado bastante bom, principalmente se não estivermos muito interessados nas flexões das etiquetas. De qualquer forma, o resultado oficial continua sendo o da seção anterior. Mas com esta experiência podemos mostrar que etiquetadores híbridos ou compostos implementados em conjunto possivelmente apresentem resultados melhores que etiquetadores baseados em apenas um modelo (veja [1] para um resultado semelhante).

---

## Conclusões

Construímos um etiquetador morfo-sintático baseado em cadeias de Markov de ordem 2, e testamos sua eficiência com um cópulo anotado do português [17]. Baseados em recentes pesquisas sobre um novo modelo estatístico teórico [8] chamado *Cadeias de Markov de Tamanho Variável* (VLMC, do inglês *Variable Length Markov Chains*), construímos outro etiquetador para utilizar este modelo. Testamos então este etiquetador com as mesmas informações e os parâmetros mais semelhantes possíveis aos do primeiro etiquetador, e comparamos os resultados dos dois.

Com isto, pudemos comparar diretamente os dois modelos estatísticos usados: o com cadeias de Markov de ordem fixa e o com cadeias de Markov de tamanho variável. Colocamos as teorias de cada modelo em prática no mesmo ambiente, implementando-as da mesma maneira e testando-as com os mesmos dados, e com isso pudemos avaliar de forma mais transparente e imparcial o essencial delas.

Vimos que o etiquetador que usa Cadeias de Markov de Tamanho Variável (o **VLMC Tagger**) apresentou precisão melhor do que o que usa cadeias de Markov de ordem fixa (o **HMM Tagger**), mesmo tomando um pouco mais de tempo para isto: 95,5129% contra 93,4875%, sobre 259991 palavras de teste. Ainda assim, o tempo necessário ao **VLMC Tagger** para ser treinado e testado foi bastante bom, considerada a quantidade de palavras e etiquetas utilizadas: 157 segundos usando um total de 1035593 palavras, 775602 para treinar e 259991 para testar. Desse modo, considerando o tempo gasto para testar, a taxa de etiquetagem foi de 2653 palavras por segundo.

O **HMM Tagger** levou um tempo total menor, de 123 segundos, com uma velocidade de 2857 palavras etiquetadas por segundo. Portanto, como vimos na Seção 7.2, os dois etiquetadores tomaram praticamente o mesmo tempo na etiquetagem. A diferença, então, foi no tempo gasto para o treinamento dos etiquetadores: 46 segundos para o **VLMC Tagger** contra 18 para o **HMM Tagger**. Entretanto, mostramos que não tínhamos realizado nenhuma iteração de treinamento do **HMM Tagger**, porque a precisão do etiquetador caía fazendo isto. E além disto, o tempo necessário para fazer uma iteração de treino é muito grande. Para comparar, executamos o **HMM Tagger** com os mesmos dados e parâmetros, apenas alterando para que fossem feitas duas iterações de treino. A precisão da etiquetagem foi de 92,677%, 0,81% menor do que sem treino, e o tempo total de execução foi de 4622 segundos, 37 vezes maior do que sem treino. Assim, mesmo que a precisão aumentasse com o treinamento, o tempo para isto dificilmente diminuiria, e portanto o ganho alcançado não compensaria o custo requerido, principalmente se comparado com o ganho e o custo do **VLMC Tagger**. Desta maneira, concluímos que o etiquetador baseado em cadeias de Markov de tamanho variável é melhor do que o baseado em cadeias de Markov de tamanho fixo, pois apresenta uma combinação melhor de resultados.

Em relação ao **VLMC Tagger** e o valor de corte  $K$  de sua árvore de contexto, vimos na

Seção 6.1.2.1 que valores pequenos de  $K$  (perto de zero) dão uma árvore de contexto grande, mas que obtêm um resultado pior na etiquetagem. Então a idéia que tínhamos de que se achariam dependências fortes de contexto não se confirmou. Talvez uma parte do contexto, que não seja a mais recente, seja a mais importante, mas então teríamos “regras”, pois precisaríamos desconsiderar as etiquetas mais recentes e procurar por algumas mais antigas que satisfizessem algum parâmetro. Por exemplo, se a palavra *que* ocorresse depois de uma seqüência formada por um ADV-R (advérbio de comparação) e quaisquer três outras etiquetas, uma regra diria que este *que* provavelmente deveria ter a etiqueta WPRO. Mas o fato de um valor grande de  $K$  (mais de 200) melhorar o resultado se deve à árvore ficar mais enxuta; neste caso o contexto recente então é o que ajuda. Portanto, possivelmente se inseríssemos algumas “regras” na árvore de contexto melhoraríamos a precisão do etiquetador. Mas como nosso etiquetador é estritamente probabilístico, não queremos fugir do nosso foco. Então deixamos esta questão para um trabalho futuro.

Uma grande vantagem do nosso etiquetador ser especificamente probabilístico é que ele não depende do conjunto de etiquetas e nem da linguagem do córpus utilizados. Sua implementação foi feita sem considerar nenhum conhecimento lingüístico específico, a menos dos sinais de pontuação final. Desta forma, ele pode ser utilizado com outros córpus anotados existentes sem precisar nem ao menos ser recompilado ou configurado. Embora seja claro que os resultados serão diferentes para córpus diferentes, o etiquetador em si não possui nenhum vínculo explícito com o córpus utilizado durante sua implementação, o Tycho Brahe. Assim, trabalhos futuros incluem a utilização de outros córpus tanto do português quanto de outras línguas no **VLMC Tagger**, o que possibilitará uma comparação ainda mais direta entre etiquetadores e entre diferentes córpus e idiomas.

Por fim, como era nosso objetivo, descobrimos que a recente teoria de *Cadeias Markov de Tamanho Variável* é melhor na aplicação à análise morfo-sintática do que a teoria classicamente aplicada de cadeias Markov de ordem fixa. Embora alguns aspectos desta nova teoria ainda estejam sendo pesquisados e estabelecidos, nosso trabalho mostrou que ela já fornece bons resultados no nosso contexto. Além disso, com nosso trabalho também pudemos identificar e assim reafirmar aos pesquisadores quais os principais pontos em aberto desta teoria, principalmente em relação ao valor de corte da árvore de contexto. Portanto, esta teoria aplicada à análise morfo-sintática possui um potencial ainda maior de eficiência. Nosso trabalho, por fim, mostra que esta abordagem permite dar-se um passo a mais em direção à uma melhor solução do problema da etiquetagem.

# Referências Bibliográficas

- [1] Rachel Virgínia Xavier Aires, *Implementação, adaptação, combinação e avaliação de etiquetadores para o português do brasil*, Dissertação de mestrado, Instituto de Ciências Matemáticas e Computação, Universidade de São Paulo - Campus São Carlos, Outubro 2000.
- [2] Carlos Daniel Chacur Alves and Marcelo Finger, *Etiquetagem do português clássico baseada em corpora*, Proceedings of IV Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR99) (Évora, Portugal), September 1999, pp. 21–22.
- [3] L. E. Baum, *An inequality and associated maximization technique in statistical estimation for probabilistic functions of a markov process*, Inequalities **3** (1972), 1 – 8, *apud*: [19].
- [4] L. Boggess, R. Agarwal, and R. Davis, *Disambiguation of prepositional phrases in automatically labeled technical text*, Proceedings of the Ninth National Conference on Artificial Intelligence (Menlo Park), AAAI Press/MIT Press, 1991, *apud*: [9], pp. 155–159.
- [5] Thorsten Brants, *Tnt – a statistical part-of-speech tagger*, Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000) (Seattle, WA), 2000.
- [6] Eric Brill, *Unsupervised learning of disambiguation rules for part of speech tagging*, Proceedings of the Third Workshop on Very Large Corpora (Somerset, New Jersey) (David Yarovsky and Kenneth Church, eds.), Association for Computational Linguistics, 1995, pp. 1–13.
- [7] Rogério Theodoro de Brito, *Alinhamento de sequências biológicas*, Dissertação de mestrado, Programa de Pós-Graduação em Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, Setembro 2003.
- [8] Peter Bühlmann and Abraham J. Wyner, *Variable length markov chains*, Annals of Statistics **27** (1999), no. 2, 480–513.
- [9] Eugene Charniak, *Statistical language learning*, The MIT Press, 1993.
- [10] Eugene Charniak, Curtis Hendrickson, Neil Jacobson, and Mike Perkowitz, *Equations for part-of-speech tagging*, Proceedings of the Eleventh National Conference on Artificial Intelligence (Menlo Park), AAAI Press/MIT Press, 1993, *apud*: [9], pp. 784–789.
- [11] Kenneth Ward Church, *A stochastic parts program and noun phrase parser for unrestricted text*, Proceedings of the second conference on Applied natural language processing (Austin, Texas), Association for Computational Linguistics, 1988, pp. 136–143.
- [12] Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun, *A practical part-of-speech tagger*, Tech. report, Xerox Palo Alto Research Center, 1991.

- 
- [13] Archias Alves de Almeida Filho, *Maximização de entropia em lingüística computacional para a língua portuguesa*, Dissertação de mestrado, Programa de Pós-Graduação em Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, Dezembro 2002.
- [14] Steven J. DeRose, *Grammatical category disambiguation by statistical optimization*, Computational Linguistics **14** (1988), 31–39, *apud*: [9].
- [15] Marcelo Finger, *Técnicas de otimização da precisão empregadas no etiquetador Tycho Brahe*, Proceedings of V Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR2000) (Atibaia, Brazil), November 2000, pp. 19–22.
- [16] ICMC-USP, *NILC's corpora*, Acessado em 2005.
- [17] IEL-UNICAMP and IME-USP, *Corpus Anotado do Português Histórico Tycho Brahe*, Acessado em 2005.
- [18] Daniel S. Jurafsky and James H. Martin, *Speech and language processing*, Prentice Hall, 2000.
- [19] Christopher D. Manning and Hinrich Schütze, *Foundations of statistical natural language processing*, The MIT Press, 1999.
- [20] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz, *Building a large annotated corpus of english: The penn treebank*, Computational Linguistics **19** (1994), no. 2, 313–330.
- [21] Nuno C. Marques and Gabriel Pereira Lopes, *Using neural nets for portuguese part-of-speech tagging*, Proceedings of the Fifth International Conference on Cognitive Science and Natural Language Processing (Dublin City University, Ireland), September 1996.
- [22] Martin Mächler and Peter Bühlmann, *Variable length markov chains: Methodology, computing and software*, Research Report 104, Eidgenössische Technische Hochschule (ETH), CH-8091 Zürich, Switzerland, March 2002, Seminar fur Statistik.
- [23] Lawrence R. Rabiner, *A tutorial on hidden markov models and selected applications in speech recognition*, Proceedings of the IEEE, no. 2, vol. 77, February 1989, pp. 257–285.
- [24] Adwait Ratnaparkhi, *A maximum entropy model for part-of-speech tagging*, Proceedings of the Empirical Methods in Natural Language Processing Conference (University of Pennsylvania), May 1996.
- [25] J. Rissanen, *A universal data compression system*, IEEE Trans. Inform. Theory **IT-29** (1983), 656 – 664.
- [26] Sheldon M. Ross, *Applied probability with optimization applications*, Holden-Day, Inc., 1970.
- [27] ———, *Introduction to probability and statistics for engineers and scientists*, John Wiley & Sons, Inc., 1987.
- [28] SGI, *Standard template library programmer's guide*, 2005, Acessado em Janeiro de 2005.
- [29] Bjarne Stroustrup, *The C++ Programming Language*, Addison-Wesley Longman Publishing Co., Inc., 2000.

- 
- [30] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer, *Feature-rich part-of-speech tagging with a cyclic dependency network*, Proceedings of HLT-NAACL 2003, 2003, pp. 252–259.
- [31] A. J. Viterbi, *Error bounds for convolutional codes and an asymptotically optimal decoding algorithm*, IEEE Transactions on Information Theory (1967), 260 – 269, *apud*: [19].
- [32] Erhan Çinlar, *Introduction to stochastic processes*, Prentice-Hall, Inc., 1975.

## O Sistema de Anotação do Córpus Tycho Brahe

A Tabela A.1 abaixo lista as etiquetas utilizadas no c3rpus Tycho Brahe, um total de 383 etiquetas. A maioria delas possui uma forma raiz que aceita flex3es. Entretanto, para simplificar, n3o vamos descrever o significado das etiquetas flexionadas. Para a descri33o completa e exemplificada de todas as etiquetas, consulte o manual de etiquetas na p3gina do Tycho Brahe [17], dispon3vel no endere3o <http://www.ime.usp.br/~tycho/corpus/manual/tags.html>.

Vamos explicar apenas as flex3es que mais se aplicam, que s3o as flex3es de g3nero e n3mero: a flex3o -F indica o g3nero feminino de uma etiqueta, e a flex3o -G indica o g3nero neutro. A aus3ncia destas duas flex3es indica o g3nero masculino. A flex3o -P indica o plural de uma etiqueta, e sua aus3ncia indica o singular.

Tabela A.1: Etiquetas do c3rpus Tycho Brahe.

ETIQUETA	TIPO	ETIQUETA	TIPO
.	Pontua33o final	,	Pontua33o n3o-final
(	Par3ntese	QT	Cita33es ( " ou " )
FW	Palavras Estrangeiras	XX	Palavras Desconhecidas
ADV ADV-NEG ADV-R ADV-S	Adv3rbios	NUM NUM-F	N3meros Cardinais
C	Complementizador	NEG	Part3culas de Nega33o
CL	Cl3ticos em geral	CL+CL	Contra33es de cl3ticos em geral
CONJ CONJ-NEG	Conjun33es Coordenativas	CONJS	Conjun33es Subordinativas
INTJ	Interjei33es	P	Preposi33es

Tabela A.1: continuação...

ETIQUETA	TIPO	ETIQUETA	TIPO
ADJ ADJ-F ADJ-F-P ADJ-G ADJ-G-P ADJ-P ADJ-R ADJ-R-F ADJ-R-F-P ADJ-R-G ADJ-R-G-P ADJ-R-P ADJ-S ADJ-S-F ADJ-S-F-P ADJ-S-P	Adjetivos	P+ADV P+CL P+D P+D-F P+D-F-P P+D-P P+D-UM P+D-UM-F P+D-UM-F-P P+DEM P+OUTRO P+OUTRO-F P+OUTRO-F-P P+PRO P+Q P+Q-F P+Q-F-P P+Q-P P+WADV P+WD P+WPRO	Contrações de Preposições
D D-F D-F-P D-G D-G-P D-P D+OUTRO D+OUTRO-F D+OUTRO-F-P D+OUTRO-P  D-UM D-UM-F D-UM-F-P D-UM-P	Determinantes Definidos  Determinantes Indefinidos	WPRO WPRO-F WPRO-F-P WPRO-P WPRO\$ WPRO\$-F WPRO\$-F-P WPRO\$-P  WD WD-F WD-F-P WD-P	Elementos Relativos ou Exclamativos / Interrogativos  Determinantes Exclamativos / Interrogativos
DEM	Outros demonstrativos	FP	Partículas de Foco
N N-P	Nomes	NPR NPR-P	Nomes Próprios
PRO	Pronomes	WQ	Elementos Exclamativos/Interrogativos
PRO\$ PRO\$-F PRO\$-F-P PRO\$-P	Pronomes possessivos	OUTRO OUTRO-F OUTRO-F-P OUTRO-P	<i>Outro</i>

Tabela A.1: continuação...

ETIQUETA	TIPO	ETIQUETA	TIPO
Q Q-P Q-F Q-F-P Q-G Q-G-P	Quantificadores	Q-NEG Q-NEG-F Q-NEG-F-P Q-NEG-P	Quantificadores de ne- gação
SE	Clíticos <i>Se</i>	SE+CL	<i>Se</i> + clítico dativo <i>the</i>
SENAO	Partículas de Negação	WADV	Elementos Relativos ou Exclamativos / Interro- gativos
VB VB-AN VB-AN-F VB-AN-F-P VB-AN-P VB-D+CL+CL VB-F+SE+CL	Verbos em geral	VB-G+SE VB-R!CL VB-R!CL+CL VB-R!SE VB-R!SE+CL VB- SP+SE+CL VB- SR+SE+CL	Verbos em geral
ET ET-F+SE+CL ET-G+SE ET-R!CL ET-R!SE ET-SD+SE ET- SD+SE+CL ET- SP+SE+CL ET- SR+SE+CL	Verbo ESTAR	HV HV-AN HV-AN-F HV-AN-F-P HV-AN-P HV-D+CL+CL HV-F+SE+CL HV-G+SE HV-P+P HV-P+P+CL HV- P+P+CL+CL HV-P+P+SE HV- P+P+SE+CL HV- SP+CL+SE HV- SR+CL+SE	Verbo HAVER

Tabela A.1: continuação...

ETIQUETA	TIPO	ETIQUETA	TIPO		
SR	Verbo SER	TR	Verbo TER		
SR-D+CL+CL		TR+G+SE			
SR-F+SE+CL		TR-AN			
SR-G+SE		TR-AN-F			
SR-R!CL		TR-AN-F-P			
SR-R!CL+CL		TR-AN-P			
SR-RA		TR-D+CL+CL			
SR-RA+CL		TR-R!CL			
SR-		TR-R!CL+CL			
RA+CL+CL		TR-R!SE			
SR-RA+SE		TR-R!SE+CL			
SR-		TR-			
RA+SE+CL		SP+SE+CL			
SR-		TR-			
SP+SE+CL		SR+SE+CL			
SR-					
SR+SE+CL					
+CL				-P	
+CL+CL				-P+CL	
+SE				-P+CL+CL	
+SE+CL		-P+SE			
-D		-P+SE+CL			
-D+CL		-RA			
-D+SE		-RA+CL			
-D+SE+CL		-RA+CL+CL			
-F		-RA+SE			
-F+CL	Flexões comuns a todos os verbos (VB, ER, HV, SR, TR)	-RA+SE+CL	Flexões comuns a todos os verbos (VB, ER, HV, SR, TR)		
-F+CL+CL		-SD			
-F+SE		-SD+CL			
-G		-SD+CL+CL			
-G+CL		-SP			
-G+CL+CL		-SP+CL			
-G+SE+CL		-SP+CL+CL			
-I		-SP+SE			
-I+CL		-SR			
-I+CL+CL		-SR+CL			
-PP		-SR+CL+CL			
-R		-SR+SE			

# Recursos do Etiquetador VLMC Tagger

O programa **VLMC Tagger** está disponível como *software livre* sob a GPL (Licença Pública Geral, do inglês *General Public License*). Os executáveis, o código-fonte do programa e o manual de uso são encontrados em <http://www.ime.usp.br/~kepler/>.

Por padrão, o nome do arquivo executável do etiquetador se chama `vlmmtagger` (no MS Windows<sup>©</sup> ele possui a extensão `.exe`). A linha de comando é esta:

```
vlmmtagger [-trn ARQUIVO [-tag ARQUIVO | -utag ARQUIVO] | -corpus ARQUIVO  
[-pcorpus NÚMERO]] [OPÇÕES]...
```

O parâmetro `-trn ARQUIVO` especifica qual o arquivo que contém o cópulus de treino. Assim,

```
vlmmtagger -trn corpus.train
```

diz ao etiquetador para ler o arquivo `corpus.train` e utilizá-lo no seu treinamento. O cópulus NÃO precisa ser o Tycho Brahe [17], mas por padrão o formato de `ARQUIVO` considerado é *palavra/etiqueta*. Para alterar o delimitador entre *palavra* e *etiqueta*, utilize a opção `-delim 'C'`, onde `C` é o caractere delimitador. Por enquanto o caractere `' '` (espaço) não é suportado (e nem outros caracteres especiais como `'\t'` e `'\n'`). O conjunto de etiquetas é automaticamente extraído do cópulus.

Note que os símbolos de pontuação devem estar **SEPARADOS** das palavras (por exemplo, `'no fim .'`, e não `'no fim.'`). Geralmente cópulus anotados já estão neste formato. Mas se for etiquetar um texto não etiquetado (veja no próximo parágrafo), os resultados podem ser indesejáveis se o arquivo não estiver neste formato.

Os parâmetros `-tag` e `-utag` são similares e especificam o arquivo a ser etiquetado. A diferença é que `-tag` diz ao etiquetador que o texto do arquivo especificado já está etiquetado, enquanto que `-utag` especifica um arquivo contendo texto não etiquetado. No caso do texto já estar etiquetado, o programa calcula ao final da execução a taxa de acertos obtida pela etiquetagem. Assim, o etiquetador executado com o comando

```
vlmmtagger -trn corpus.train -tag corpus.test
```

faz o treinamento com o cópulus em `"corpus.train"` e a etiquetagem com `"corpus.test"`, calculando a taxa de acertos.

Se não for utilizado nenhum dos parâmetros acima, pode-se utilizar `-corpus ARQUIVO`, para especificar um único arquivo contendo um cópulus etiquetado. Usa-se então a opção `-pcorpus`

NÚMERO para definir que  $(\text{NÚMERO} \times 100)\%$  das sentenças do cópús, escolhidas aleatoriamente, serão utilizadas no treino do etiketador. As sentenças restantes serão utilizadas na etiketagem. Se `-pcorpus` não for especificada, por padrão seu valor é igual a 0.75. Assim, a linha de comando

```
vlmmtagger -corpus corpus.all -pcorpus 0.9
```

executa o etiketador e o treina com 90% de “corpus.all” e etiketa os 10% restantes.

Junto com estes parâmetros pode-se usar a opção `-ptrn NÚMERO`, NÚMERO sendo entre 0 e 1. Esta opção instrui o etiketador a selecionar aleatoriamente  $(\text{NÚMERO} \times 100)\%$  das sentenças do cópús de treino, e efetuar o treinamento apenas com estas sentenças. Esta opção existe para facilitar testes com diferentes parâmetros.

Também para auxiliar nos testes existem as opções `-q` e `-v`. A primeira torna a execução do etiketador silenciosa, de modo que nenhuma mensagem é enviada à saída padrão. Ela é útil quando se quer executar o etiketador dentro de algum *script* ou programa, e quando se está interessado apenas no texto final etiketado. Já a segunda opção aumenta o nível de verbosidade do etiketador, gerando mensagens extras na saída. É mais útil quando se está testando os parâmetros do etiketador.

Por último, vamos explicar os parâmetros que influenciam as decisões algorítmicas do etiketador. Em relação à árvore de contexto, pode-se definir três atributos:

1. A constante de corte do algoritmo de contexto;
2. O número mínimo de ocorrências requeridas de uma seqüência de etiketas para que ela conste na árvore e;
3. A ordem máxima da cadeia de Markov de tamanho variável, ou seja, o tamanho máximo considerado de uma seqüência de etiketas.

O primeiro atributo é especificado com a opção `-K NÚMERO`. O segundo com `-density NÚMERO`. E o terceiro com a opção `-order NÚMERO`.

As últimas duas opções se referem a decisões gerais do etiketador, feitas durante a fase de aprendizado. A opção `-opentag NÚMERO` define o número mínimo de vezes que uma etiketa deve ocorrer com palavras distintas para que seja considerada uma etiketa aberta. E a opção `-suffix NÚMERO` especifica quanto de uma palavra deve ser considerado como seu sufixo. Por exemplo, `-suffix 0.5` diz que a última metade de uma palavra é seu sufixo, e `-suffix 0.7` diz que 70% da parte de trás de uma palavra é seu sufixo.

Para resumir, os parâmetros do `vlmmtagger` estão colocados na Tabela B.1, e podem ser obtidos pelo etiketador com o comando

```
vlmmtagger --help
```

Lembrando que a linha geral de comando é

```
vlmmtagger [-trn ARQUIVO [-tag ARQUIVO | -utag ARQUIVO] | -corpus ARQUIVO  
[-pcorpus NÚMERO]] [OPÇÕES]...
```

<b>Opções do Etiketador Morfo-Sintático baseado em VLMCs</b>	
<b>Parâmetro</b>	<b>Descrição</b>
-trn ARQUIVO	usa ARQUIVO como cópulo de treino (ARQUIVO deve estar etiquetado (padrão: "corpus.train"));
-tag ARQUIVO	etiqueta o texto em ARQUIVO e salva o texto etiquetado resultante em "ARQUIVO.tagged" (ARQUIVO deve estar etiquetado) (padrão: não etiqueta nada);
-utag ARQUIVO	etiqueta o texto em ARQUIVO e salva o texto etiquetado resultante em "ARQUIVO.tagged" (ARQUIVO NÃO deve estar etiquetado) (padrão: não etiqueta nada);
-corpus ARQUIVO	especifica o ARQUIVO de cópulo a ser usado; os cópulos de treino e teste não obtidos a partir deste de acordo com a opção '-pcorpus'; salva o texto etiquetado resultante em "ARQUIVO.test.tagged" (ARQUIVO deve estar etiquetado);
-pcorpus NÚMERO	divide aleatoriamente (NÚMERO × 100)% do cópulo para treinamento e o restante para teste (NÚMERO deve estar no intervalo [0,1] (padrão: 0.75)) (somente aplicável se a opção '-corpus' for especificada);
<b>As [OPÇÕES] podem ser as seguintes:</b>	
-s ARQUIVO	salva o modelo treinado no ARQUIVO;
-l ARQUIVO	carrega o modelo treinado do ARQUIVO; estas opções podem ser usadas para evitar a execução do treinamento; (assim, se a opção '-trn' é especificada, ela é ignorada);
-sc ARQUIVO	salva o modelo acumulado no ARQUIVO;
-lc ARQUIVO	lê o modelo acumulado do ARQUIVO; estas opções podem ser usadas quando se quer adicionar novas informações a uma execução anterior do etiketador, geralmente quando se consegue um novo cópulo; (assim, o treinamento tem que ser executado);
-delim C	define C como o delimitador entre palavras e etiquetas no cópulo usado (C é um caractere, exceto espaço, tabulação e nova linha (padrão: /))
-ptrn NÚMERO	seleciona aleatoriamente somente (NÚMERO × 100)% do cópulo de treino para uso (NÚMERO deve estar no intervalo [0,1] (padrão: 1));
-q   -v	modo quieto/silencioso (-q) ou verboso (-v);
<b>[OPÇÕES] algorítmicas:</b>	
-K NÚMERO	especifica a constante de corte da Árvore de Contexto;
-density NÚMERO	especifica o NÚMERO mínimo de vezes que uma seqüência de etiquetas deve aparecer no cópulo para que ela conste na Árvore de Contexto;
-order NÚMERO	especifica a ordem máxima da cadeia de Markov de tamanho variável;
-opentag NÚMERO	especifica o NÚMERO mínimo de vezes que uma etiqueta deve ocorrer com palavras distintas para que seja considerada uma etiqueta aberta;
-suffix NÚMERO	especifica a parte de uma palavra a ser considerada como seu sufixo (NÚMERO deve estar no intervalo [0,1]) (padrão: 11/20);

**Tabela B.1:** Parâmetros do **VLMC Tagger**