MAC5701 - Tópicos em Ciência da Computação

Modelos Probabilísticos de Análise Sintática

Fábio Natanael Kepler

Orientador: Marcelo Finger

Sumário

CAPíTULO 1

Introdução

Um dos requisitos fundamentais para o bom funcionamento de sistemas de processamento de linguagem, especialmente quando é necessária a interpretação (como em sistemas de Extração de Informação, Tradução Automática, ou Reconhecimento de Fala), é a eficiência na recuperação da estrutura sintática das sentenças. Em sua forma mais simples, o problema da análise sintática envolve a definição de um algoritmo que associa a uma senteça de entrada sua correspondente estrutura de árvore sintática [1].

Esta análise sintática é realizada pelo *analisador sintático*[2], que tem por função recuperá-la com a ajuda de uma gramática, ou através de aprendizagem automática utilizando um banco de sentenças, também chamado de córpus ou *treebank*. Um córpus é uma coleção (grande) de sentenças autênticas anotadas manualmente com informações sintáticas e morfossintáticas (veja seção 3). Exemplos de córpus são o *Penn Treebank* [**22**], para o inglês, e o *Tycho Brahe* [**15**], para o português do Brasil.

No caso da análise sintática como um problema de aprendizado de máquina, uma função *sentença* $\longrightarrow$ *árvore* é induzida a partir de um conjunto de treinamento obtido do córpus. A avaliação da precisão do modelo é feita usando-se um conjunto de teste contendo sentenças não participantes do treinamento. No caso dos métodos estatísticos, o aprendizado se torna uma tarefa de estimativa de valores de parâmetros a partir de dados de treinamento. A combinação dos melhores parâmetros forma a árvore mais provável para uma dada sentença.

1. Um Grande Problema: Ambigüidade

Ambigüidade é um grande problema na análise sintática, e uma das principais motivações para a escolha dos métodos estatísticos. Alguns poucos casos de ambigüidade sintática, dentre os muitos existentes, são os seguintes:

[1] Assumimos uma definição do problema da análise sintática onde cada sentença deve ser associada a uma única árvore sintática, mesmo que isto necessite de desambiguação usando fontes de conhecimento além da gramática. Isto contrasta com outra definição comum do problema, onde a tarefa é recuperar todas árvores bem-formadas para a sentença, sem nenhuma desambiguação.

[2] *Parser*, em inglês.

- Ambigüidade de categoria sintática. Por exemplo, a palavra banco pode ser tanto um verbo quanto um substantivo;
- Ambigüidade na ligação de sintagmas[3] preposicionais. A sentença "A menina viu o menino de binóculo" tem no mínimo duas árvores sintáticas: uma com o sintagma de binóculo modificando menino (ou seja, o menino é que estava de binóculo), e outra modificando menina (ou seja, a menina estava de binóculo e viu o menino).
- Ambigüidade em coordenação. Na sentença "um programa para promover segurança em caminhões e peruas", a palavra peruas pode ser coordenada com caminhões, segurança ou programa, gerando várias árvores sintáticas possíveis.

A ambigüidade sintática gera uma explosão exponencial de análises para uma sentença. Segundo Collins (1999) **[10]**, o problema com estes e outros tipos de ambigüidade é ainda agravado quando se trata de sentenças longas, tornando necessária uma gramática bastante ampla. Estes fatos motivaram as pesquisas por modelos estatísticos, que usam probabilidades observadas de um conjunto de treinamento para ajudar a tomar decisões de ligações entre as palavras.

2. Abordagens

2.1. Abordagens Baseadas em Regras. Uma abordagem padrão ao problema da análise sintática **[1]** é construir uma gramática manualmente, com algum tipo de formalismo, ou com uma grande quantidade de informação específica lexicalizada. A ambigüidade é resolvida através de *restrições seletivas* (por exemplo, um léxico pode (1) especificar que *comer* deve tomar um objeto com a característica *+comida*, e (2) especificar quais nomes no léxico possuem a característica *+comida*). Apesar de viáveis em um domínio limitado, como tarefas de consulta a bancos de dados, restrições seletivas enfrentam vários problemas quando aplicas a tarefas mais gerais. Primeiro, o tamanho do vocabulário se torna grande demais. Segundo, sabe-se que do ponto de vista da lingüística elas apresentam problemas teóricos. Enquanto estes problemas podem não aparecer em um domínio restrito, eles são freqüentemente encontrados em um domínio mais amplo. Por último, isto implica que restrições seletivas devem ser criadas como preferências leves ao invés de restrições duras.

2.2. Métodos Estatísticos. Em resposta a estas dificuldades, pesquisadores começaram a investigar abordagens baseadas em aprendizado de máquina, principalmente através de métodos estatísticos, mas com algumas notáveis exceções, como os métodos de aprendizado baseados em regras de Brill (1993) **[6]** ou Hermjakob and Mooney (1997) **[13]**. O aprendizado estatístico se insere

[3]Sintagmas são *categorias sintáticas*. No inglês são chamados *phrases*.

num contexto cuja linha de pesquisa é chamada de empiricista, uma vez que se baseia em exemplos já prontos (um córpus anotado) e aprende como lidar com aqueles ainda não vistos.

De acordo com Manning e Schütze **[24]**, a linha empiricista, que entre as décadas de 60 e 80 ficou nas sombras de crenças racionalistas encabeçadas por Chomsky (1965), cujo reflexo dentro da Inteligência Articial caracterizava-se pela criação de sistemas inteligentes com grande quantidade de conhecimento inicial codificado à mão, ressurgiu na década de 90, com a idéia de que o conhecimento pode ser induzido a partir de algumas operações básicas de associação e generalização (Mitchell 1980 **[25]**). Assim, segundo o empiricismo, uma máquina poderia aprender a estrutura de uma linguagem apenas observando uma grande quantidade de exemplos, usando procedimentos estatísticos gerais e métodos de associação e generalização indutiva, como aprendizado indutivo de regras (Brill 1993 **[6]**).

3. Objetivo

Embora vários modelos usados no Inglês tenham sido bastante descritos por Sousa **[11]** e Bonfante **[5]**, nosso objetivo final é mais específico: queremos levantar métodos estatísticos de análise sintática tendo em vista a construção de um analisador sintático para o Português. Não iremos propriamente construir um analisador, mas desejamos obter informação e conhecimento suficientes para poder decidir mais claramente qual modelo de análise sintática poderia ser melhor adaptado para o Português, considerando o estado-da-arte e os atuais recursos disponíveis (principalmente quanto a córpus anotado sintaticamente).

4. Estrutura desta Monografia

No próximo capítulo descrevemos os fundamentos da análise sintática, separando os passos necessários para a modelagem de um analisador sintático e mostrando as decisões a serem tomadas que diferenciam os analisadores sintáticos existentes. No capítulo 3 mostramos e descrevemos os principais modelos de analisadores sintáticos existentes, e no capítulo 4 comparamos os resultados de todos eles e analisamos potenciais adaptações para o Português.

CAPíTULO 2

Fundamentos da Análise Sintática Estatística

O processo de análise sintática estatística é separado em dois componentes:

- **o *modelo*:** que é a função que define o espaço de eventos e os parâmetros a serem associados a cada evento;
- **e o *analisador sintático*:** que propriamente dito, é o algoritmo que implementa a busca pela melhor árvore sintática para cada sentença.

Nesse cenário, se o espaço de eventos possíveis for associado aos mapeamentos entre as sentenças (S) e suas respectivas árvores sintáticas (A), o aprendizado passa a ser a indução de uma função probabilística

$$\mathit{Pontuação} : A \times S \rightarrow [0, 1],$$

em que $\mathit{Pontuação}(A, S)$ pode ser tanto uma probabilidade conjunta $P(A, S)$, quanto uma probabilidade condicional $P(A|S)$. A árvore mais provável para uma sentença de entrada S é então definida como

$$A_{melhor}(S) = \arg\max_{A} \mathit{Pontuação}(A, S) \tag{1}$$

Deste modo, os dois componentes da análise sintática podem ser definidos como:

- **o modelo:** é a função que define a probabilidade $\mathit{Pontuação}(A, S)$ para cada par (A, S);
- **o analisador sintático:** é o algoritmo que implementa a busca por A_{melhor} para qualquer sentença de entrada S.

Para que o modelo tenha um número tratável de parâmetros, um par (A, S) precisa ser quebrado em um conjunto de eventos $\langle \mathit{Evento}_1 \ldots \mathit{Evento}_n \rangle$. $\mathit{Pontuação}(A, S)$ é então calculada como um produto de termos, cada *Evento* tendo sua probabilidade correspondente:

$$\mathit{Pontuação}(A, S) = \prod_{i=1 \ldots n} \mathit{Pontuação}(\mathit{Evento}_i) \tag{2}$$

A escolha da *parametrização* é a escolha de como quebrar uma árvore. Em outras palavras, é a escolha de para quais eventos associar parâmetros. Existem muitas maneiras possíveis de quebrar uma árvore. Os métodos estatísticos na literatura diferem pelo modo como fazem isto, ou seja, como determinam

quais objetos lingüísticos são considerados para estimar os parâmetros de seu modelo. Veremos isto no capítulo 3.

A escolha da parametrização é central para o sucesso de um modelo de análise sintática. Uma boa parametrização deve representar concisamente os dados. Segundo Collins **[10]**, dois critérios são importantes:

Poder Discriminativo: os parâmetros devem incluir informação contextual necessária para decisões de desambiguação. Os modelos mais simples falham nesse sentido, por serem insensíveis demais a informações lexicais e preferências estruturais (por exemplo: as preferências estruturais de ligações entre os sintagmas dependem de informações lexicais (como as palavras que estão sendo ligadas) para serem realizadas);

Poder de compactação: o modelo deve ter o menor número possível de parâmetros. O número de parâmetros do modelo determina a quantidade de dados de treinamento necessária para treiná-lo. Isso significa que o poder de compactação do modelo determinará o quão perto ele chega do seu potencial máximo, dada a quantidade (quase certamente) limitada de dados para treinamento.

Estes dois critérios evoluem em sentidos opostos. Quanto mais parâmetros um modelo possui, maior é, em teoria, seu poder discriminativo, e menor seu poder de compactação. Em um nível mais extremo, por exemplo, a geração de uma árvore poderia ser composta por um único passo, o da própria geração de toda a árvore. Este caso pecaria pela compactação, já que o número de parâmetros a ser estimado seria do tamanho do domínio de sentenças possíveis da linguagem, ou seja, extremamente grande. A quantidade de dados de treinamento precisaria ser igualmente extensa (e portanto inviável). Deste modo, um bom modelo deve ser obtido pelo equilíbrio dos dois critérios, sempre visando eliminar parâmetros redundantes (que diferenciam as mesmas ambigüidades).

Para isto, um modelo parametrizável possui um argumento adicional na função *Pontuação*, dado por um vetor de parâmetros Θ. A função é então escrita como $Pontuação(A, S|\Theta)$, e o problema do aprendizado consiste em se ajustar a estimativa de Θ a partir do conjunto de exemplos de treinamento. Isto pode ser feito de duas maneiras: de modo supervisionado ou não-supervisionado. Estes dois métodos são explicados na seção 4.4.

Portanto, a tarefa de modelagem da análise sintática é dividida em três etapas:

(1) Definição do modelo de estrutura: a função $Pontuação(A, S|\Theta)$;
(2) Definição do modelo parametrizável;

(3) Definição do algoritmo de busca para encontrar $A_{melhor}(S) = \arg\max_{A} Pontuação(A, S|\Theta)$.

1. Estruturas e Representações

A estrutura geralmente usada para codificar os vários níveis de informação sintática de uma sentença é chamada de *árvore sintática*. Um exemplo de árvore sintática pode ser visto na figura 1. Nesta figura, as folhas da árvore são

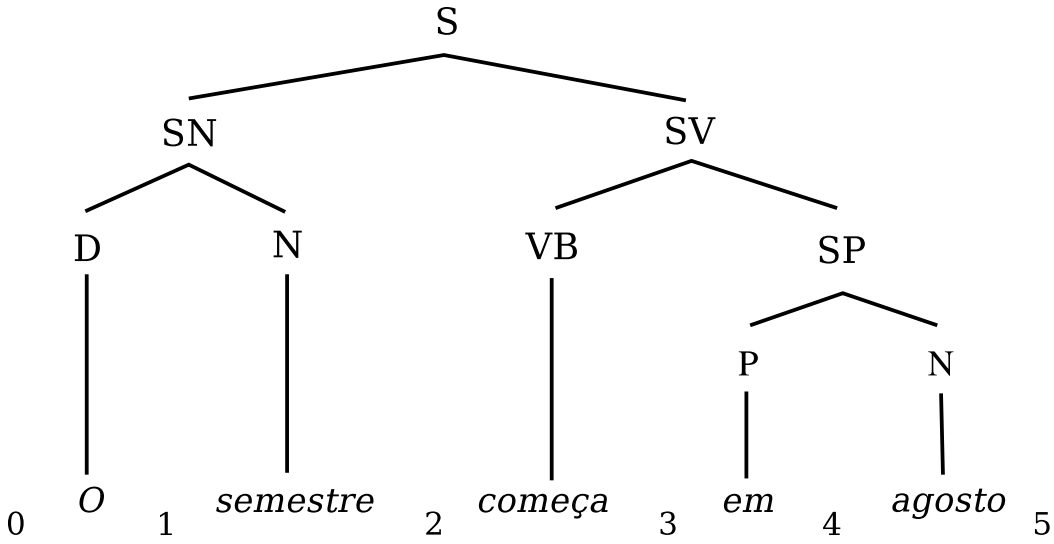

FIGURA 1. Um exemplo de árvore sintática.

as palavras de uma sentença, e os nós imediatamente acima de cada palavra são suas categorias morfossintáticas (veja **[17]** para uma descrição detalhada sobre a atribuição de categorias morfossintáticas a palavras). Assim, as palavras '*semestre*' e '*agosto*' são *nomes* (**N**), '*O*' é um *determinante* (**D**), '*começa*' é um *verbo* (**VB**) e '*em*' é uma *preposição* (**P**). Também temos a descrição sintática de agrupamentos de palavras, como '*O semestre*', que é um *sintagma nominal* (**SN**), '*em agosto*', que é um *sintagma preposicional* (**SP**), e mesmo **S**, a sentença.

2. Dependência e Constituintes

Uma idéia fundamental é que certos agrupamentos de palavras se comportam como *constituintes*. Constituintes podem ser detectados por serem capazes de ocorrer em várias posições, e mostrarem possibilidades sintáticas uniformes de expansão. Por exemplo, um constituinte que agrupa nomes e

seus modificadores, como "o semestre". Todos elementos que podem ser substituídos um pelo outro em uma certa posição sintática são membros de um *paradigma*. Duas palavras possuem uma *relação sintagmática* se podem formar um *sintagma*, como *o semestre* (um sintagma nominal).

Outra importante noção é o conceito de *dependência*. Em uma sentença como:

O semestre começa em agosto.

semestre e *agosto* são dependentes de um evento de começo. Vamos dizer que eles são dois argumentos do verbo *começar*. Normalmente, sintagmas nominais são argumentos de verbos. Um ilustração desta noção de dependência é dada na figura 2.

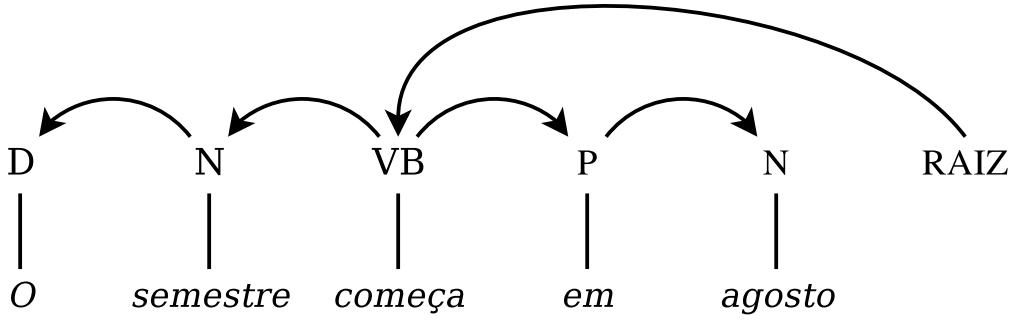

FIGURA 2. Estrutura clássica de dependência.

3. Recursos Lingüísticos

Um conjunto de textos é chamado de córpus[1]. Um córpus pode ser anotado manualmente com informações lingüísticas. As mais comuns usadas são informações referentes a categorias (etiquetas) morfossintáticas das palavras, e a estruturas sintáticas das senteças.

Um córpus que contém anotação sintática também é chamado de *banco de árvores*[2]. Atualmente não existe um grande número de córpus anotados, e muitas línguas nem dispõem de um. Isto se deve ao imenso esforço humano necessário para se anotar um córpus. Mesmo assim, existem alguns disponíveis, e que são bastante utilizados em ligüística computacional. Para o Inglês, um bom exemplo é o *Penn Treebank* [**22**], que contém textos anotados do *Wall Street Journal* (WSJ). Para o Português, existem poucos córpus anotados sintaticamente. Em particular, o córpus *Tycho Brahe* [**15**] possui anotação morfossintática, mas ainda possui pouca anotação sintática. Um dos

[1]Preferimos unicamente este termo ao invés do singular *corpus* e plural *corpora*

[2]Do inglês *treebank*.

objetivos deste trabalho é investigar modelos existentes de análise sintática visando a anotação do *Tycho Brahe*.

4. Métodos e Técnicas Utilizados

Os modelos de análise sintática se diferenciam em pelo menos uma técnica ou método empregado. Quando disponível e claramente mostrado nas fontes de referência, vamos salientar cada modelo quanto à *lexicalização* das árvores, o método de *geração* empregado, a utilização da técnica de *aglomeração*, e a forma de *aprendizado* usada.

4.1. Lexicalização. Em uma gramática *lexicalizada*, cada nó da árvore que representa um sintagma é marcado com seu item léxico núcleo. Assim, a árvore da figura 1 é lexicalizada como a árvore em 3. Isto fornece muito

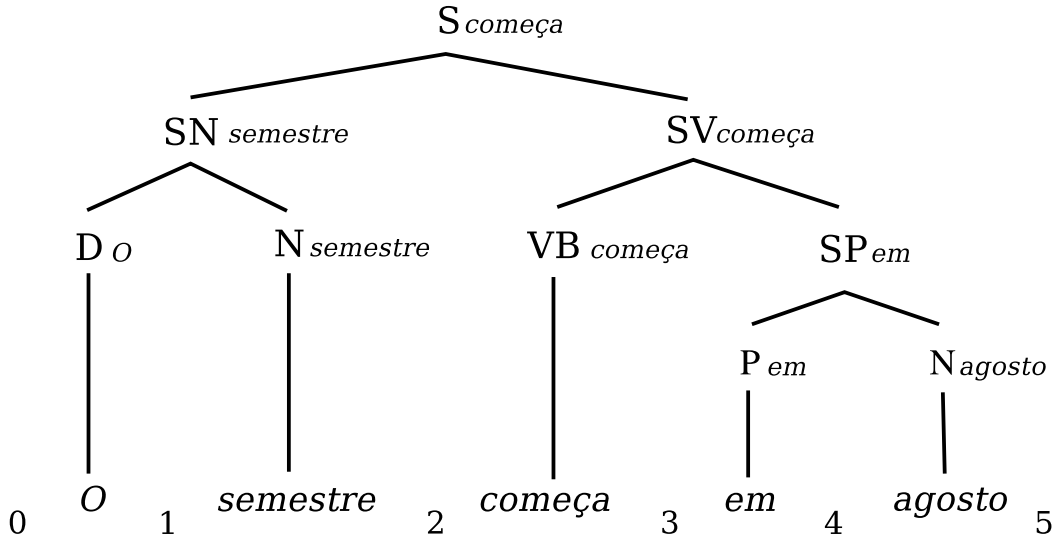

FIGURA 3. Um exemplo de árvore sintática lexicalizada.

mais informação ao analisador sintático sobre o que são realmente as palavras na sentença no momento de decidir sobre a estrutura da árvore sintática. Por outro lado, esta informação a mais degrada a eficiência computacional do analisador, que também enfrenta problemas de dados esparsos.

Para um analisador sintático não-lexicalizado, a sentença de entrada a ser analisada é na verdade somente uma lista de etiquetas morfossintáticas. Uma vantagem deste tipo de analisador é que o pequeno alfabeto de terminais o torna fácil de construir. Não é preciso se preocupar muito com eficiência computacional ou questões de suavização de dados esparsos. Entretanto, isto claramente fornece muito menos informação para ser usada do que uma sentença com palavras reais.

4.2. Geração. Um analisador sintático pode gerar a estrutura de uma árvore de baixo para cima, partindo dos itens léxicos da sentença até a raiz, reconhecendo subestruturas, ou ao contrário, partindo da raiz e derivando subestruturas, até os itens léxicos. Este segundo método de geração de uma árvore é chamado de *gerativo*, e basicamente não apresenta vantagens ou desvantagens em relação ao primeiro, *não-gerativo*. A diferença está, na verdade, na forma como o modelo do analisador cria as distribuições de probabilidade. Um modelo *gerativo* cria distribuições de probabilidade *conjunta* ($P(A, S)$), e um modelo *não-gerativo* cria distribuições de probabilidade *condicional* ($P(A|S)$).

4.3. Aglomeração. A técnica de *aglomeração*[3] consiste em classificar as palavras de um córpus em categorias de semelhança. Deste modo, palavras de algum tipo escolhido são referenciadas como um único grupo. Isto é usado principalmente por modelos não-lexicalizados de análise sintática, ou que utilizam aprendizado não-supervisionado. Diferentes modos de decidir quais palavras são de alguma forma semelhantes, e aglomerá-las junto, são melhor descritos em Schütze (1995) [**29**].

4.4. Supervisão do Aprendizado. Um dos itens principais em que modelos de análise sintática podem se diferenciar é quanto ao tipo de aprendizado de máquina que utilizam para ajustar o parâmetro Θ, definido na seção 2, da melhor forma possível. A tarefa do aprendizado também é chamada de *treinamento* do modelo.

O *aprendizado supervisionado* utiliza córpus pré-anotado, e estima as probabilidades das árvores sintáticas através da Máxima-Verossimilhança (MV)[4] (Manning e Schütze, 1999 [**24**]). Num modelo simples, a probabilidade $Pontuação(A, S|\Theta)$ é igual à probabilidade de A dado S e Θ, $P(A|S, \Theta)$, na qual o parâmetro Θ do modelo lista uma probabilidade para cada membro do conjunto $S \times A$. Estes parâmetros, então, é que são estimados usando a estimativa MV, que dá a estimativa de Θ:

$$Pontuação(A, S|\Theta) = P(A|S, \Theta) = \frac{Cont(A, S)}{Cont(S)}.$$

$Cont(S)$ é a freqüência com a qual a sentença S é vista no treinamento, e $Cont(A, S)$ é a freqüência com a qual a sentença S é vista no treinamento associada à árvore A. Neste modelo, encontrar A_{melhor} para uma entrada S consiste em procurar sua árvore mais freqüente:

$$A_{melhor}(S) = \arg\max_{A} Pontuação(A, S|\Theta).$$

[3] Do inglês *clustering*.

[4] No inglês é chamada de *Maximum-Likelihood*, ML.

O problema deste método de aprendizado, então, é que se pressupõe que uma entrada S vista no conjunto de teste terá sido vista pelo menos uma vez nos dados de treinamento. Isto claramente não é verdade, e por isso modelos que usam este tipo de aprendizado têm que se preocupar com questões de suavização de probabilidades.

Por este último motivo o aprendizado *não-supervisionado* é preferível, na medida em que remove a necessidade do alto custo de anotação manual de um córpus. Isto porque o aprendizado não-supervisionado usa texto não-analisado como dado de treinamento, procurando identificar estruturas e subestruturas comuns. Entretanto, é geralmente reconhecido que este é um problema muito mais difícil, pelo menos para o objetivo de aprender estruturas lingüísticas plausíveis.

Estruturas de modelo que definem distribuições de probabilidade conjunta $P(A, S|\Theta)$ podem ser treinados de maneira não-supervisionada usando o algoritmo de *Maximização de Esperança* (ME)[5] **[12]**. No treinamento não-supervisionado os dados de treinamento são uma seqüência de eventos $X = x_1 \ldots x_n$ tirados de $\mathcal{X}$ $((X) = \langle w_1, \ldots, w_n \rangle$, $w_{1,n}$ as palavras de uma sentenças de tamanho $n)$. No apêndice A descrevemos o algoritmo de treinamento *interno-externo*, que é uma variação do algoritmo ME.

[5]Do inglês *Expectation-Maximization (EM) algorithm.*

CAPíTULO 3

Modelos Estatísticos de Análise Sintática

Vamos examinar como algumas idéias têm sido combinadas em analisadores sintáticos estatísticos, e mostrar alguns dos modelos que estão sendo usados e o atual estado da arte.

1. Medidas de Eficiência

Para podermos comparar diferentes modelos de análise sintática, precisamos de alguma medida, diretamente comparável. A maioria dos trabalhos sobre algum modelo utiliza as medidas de **precisão**, **cobertura**, e **medida-f** como forma de mensurar sua eficiência. Considerando-se as árvores geradas por algum algoritmo de busca e as respectivas árvores originais obtidas do conjunto de teste, estas medidas são calculadas da seguinte maneira:

$$\begin{aligned} \text{precisão} &= \frac{\text{número de nós corretos da árvore gerada}}{\text{número total de nós na árvore gerada}} \\ \text{cobertura} &= \frac{\text{número de nós corretos da árvore gerada}}{\text{número total de nós na árvore original}} \\ \text{medida-f} &= 2 \times \frac{\text{precisão} \times \text{cobertura}}{\text{precisão} + \text{cobertura}} \end{aligned}$$

Existem outras medidas de eficiência, mas nem todas publicações utilizam as mesmas. Assim, vamos comparar diferentes trabalhos basicamente através das medidas acima.

2. Gramáticas Livres de Contexto (GLCs)

Uma Gramática Livre de Contexto (GLC) é geralmente definida como uma quádrupla $G = (N, \Sigma, R, I)$, onde:

- N é um conjunto finito de não-terminais;
- Σ é um conjunto finito de símbolos terminais, isto é, é o alfabeto que define as cadeias da linguagem;
- R é um conjunto finito de produções da forma $\alpha \rightarrow \beta$, onde $\alpha \in N$ e $\beta \in (\Sigma \cup N)^*$;
- $I \in N$ é o símbolo inicial.

A linguagem definida por G é o conjunto de cadeias de terminais que podem ser derivadas a partir de um seqüência de operações de reescrita com

o símbolo inicial I. As operações de reescrita são definidas pelas produções da forma $\alpha \rightarrow \beta$, onde α é substituído por β.

As GLCs são de grande utilidade prática. Basicamente todas as linguagens de programação são definidas por GLCs[1]. Elas são reconhecidas por *autômatos de pilha*, de eficiente implementação no computador.

Entretanto, nestes casos, o tamanho da gramática é relativamente pequeno. Já no caso de linguagens naturais, o tamanho da gramática é muito maior, o que degrada sensivelmente a eficiência dos algoritmos tradicionais. Além disso, um grande problema do uso de GLCs na análise sintática de liguagens naturais é o fato do modelo ser muito pouco sensível ao contexto da sentença, o que acaba gerando problemas na representação de determinadas relações de concordância. Outro problema é o fato de que o treinamento automático de uma GLC precisa ser feito com a utilização de exemplos negativos, para que a GLC gerada não seja permissiva, isto é, aceite muitas sentenças que não são gramaticalmente corretas.

3. Gramáticas Livres de Contexto Probabilísticas (GLCPs)

Uma Gramática Livre de Contexto Probabilística (GLCP) é semelhante a uma GLC normal, mas contém, adicionalmente, uma probabilidade associada a cada regra da gramática. Escrevemos a probabilidade associada à regra $\alpha \rightarrow \beta$ como $P(\alpha \rightarrow \beta|\alpha)$. Isto é interpretado como a probabilidade condicional de escolher a regra $\alpha \rightarrow \beta$, dado que α é o não-terminal sendo reescrito em uma derivação. Se D é um função que atribui uma probabilidade a cada membro de R, uma GLCP é um quíntupla $G = (N, \Sigma, R, I, D)$.

Dada uma GLCP, a probabilidade para qualquer árvore livre de contexto em uma linguagem é o produto das probabilidades das regras que ela contém. Isto é, se A é uma derivação livre de contexto que envolve n regras da forma $\alpha_i \rightarrow \beta_i$,

$$P(A) = \prod_{i=1...n} P(\alpha_i \rightarrow \beta_i|\alpha_i) \tag{3}$$

Uma GLCP também define uma distribuição de probabilidade sobre sentenças. Se $\mathcal{A}(S)$ é o conjunto de árvores que possuem como superfície a sentença S, então

$$P(S) = \sum_{A \in \mathcal{A}(S)} P(A) \tag{4}$$

[1]Os *tipos* das variáveis são a única condição de contexto utilizada em linguagens de programação, em geral.

E principalmente, dado que estamos tentando definir modelos de análise sintática, uma GLCP também define a árvore mais provável para cada sentença S, $A_{melhor}(S)$ (veja equação 1):

$$A_{melhor}(S) = \arg\max_{A \in \mathcal{A}(S)} P(A) \tag{5}$$

Um dos métodos mais simples que implementa esta função é uma extensão do algoritmo CKY (Hopcroft e Ullman, 1979) [**14**] – originalmente desenvolvido para gramáticas livres de contexto – para GLCPs. Outro método bastante usado é o algoritmo de Earley [**9**], que descrevemos no apêndice B.

A principal vantagem das GLCPs sobre o modelo anterior é que pode-se distinguir entre árvores candidatas de maneira direta. Outra vantagem é que não é necessário o uso de exemplos negativos para o treinamento.

As principais desvantagens das GLCPs são a falta de sensibilidade a dependências léxicas e a falta de sensibilidade a preferências estruturais. A primeira deficiência significa que a decisão entre duas estruturas depende apenas das probabilidades das regras utilizadas, e não tem nenhuma relação com os itens lexicais em questão, pois as GLCPs usam árvores **não-lexicalizadas**. A segunda deficiência pode ser verificada quando existem duas possíveis formas de ramificação, à direita ou à esquerda. Elas possuem as mesmas regras gramaticais, e portanto uma GLCP não sabe distinguir a correta.

Alguns resultados publicados sobre o uso de GLCPs para análise sintática são relativamente ruins. Charniak (1997) [**8**] treinou e testou uma GLCP usando o banco de árvores WSJ, e obteve uma precisão de 70,6% e uma cobertura de 74,8% (uma medida-f de 72,6%).

4. Modelo Proposto por Charniak (1997)

O modelo proposto por Charniak [**8**] limita a geração de árvores a apenas utilizar as regras gramaticais vistas nos dados de treinamento do algoritmo. As regras são **lexicalizadas**, e sua aplicação é dividida em duas partes. A primeira atribui uma probabilidade para a utilização da regra em si, e a segunda avalia a probabilidade dos itens lexicais que preenchem as folhas.

Charniak enfoca o aprendizado automático de gramáticas livres de contexto a partir de dados de treinamento não anotados manualmente, ou seja, propõe um modelo de aprendizado **não-supervisionado**. Em seu trabalho, ele apresenta técnicas de **aglomeração** de palavras, para diminuir o problema dos dados esparsos, e também apresenta técnicas para decidir o sentido de uma palavra de múltiplos sentidos, através do contexto ou de informações externas.

Com este modelo, Charniak alcança 87,4% de precisão e 87,5% de cobertura (e 87,4% de medida-f) no córpus *WSJ*.

Entretanto, as hipóteses de independência de Charniak, de que a expansão de um símbolo não depende de nenhum símbolo acima e nenhum símbolo dos lados, são fortes demais. Alguns modelos mais adiante neste trabalho vão afirmar isto, e argumentar que é necessário considerar o contexto dos símbolos.

5. Modelos Baseados em História

Apesar de produzirem bons resultados, os modelos descritos nas seções anteriores não conseguem capturar informações suficientes para lidar muito bem com a ambigüidade. As GLCPs tradicionais contam com uma certa noção de contexto, mas ainda restrita a características locais. Na análise sintática, supõe-se que a informação lexical de longa distância é crucial para desambiguação, e neste caso modelos locais como as GLCPs são insuficientes.

Um Modelo Baseado em História[2] (MBH) é um modelo **gerativo** probabilístico capaz de considerar informação contextual em qualquer posição da história do discurso. Foi proposto a primeira vez por Black et al. (1992) **[4]** para o processamento de linguagem natural. Trabalhos posteriores como **[16, 23, 27, 26]** o usaram para problemas de análise sintática e etiquetagem morfossintática.

A idéia é definir um mapeamento um-pra-um que associa cada membro do espaço de entrada e saída $S \times A$ (sentença e árvore) a uma seqüência de decisões $\langle d_1, d_2, \ldots, d_n \rangle$. A probabilidade conjunta de um membro $(s, a) \in S \times A$ (s uma sentença e a uma árvore) é dada por

$$P(s,a) = P(\langle d_1, d_2, \ldots, d_n \rangle) = \prod_{i=1\ldots n} P(d_i | d_1, \ldots, d_{i-1}). \tag{6}$$

O contexto condicionante para cada d_i, $\langle d_1, d_2, \ldots, d_{i-1} \rangle$, é referido como a "história", e é equivalente a alguma estrutura parcialmente construída.

Associar um parâmetro $P(d_i | d_1, \ldots, d_{i-1})$ a cada possível prefixo $\langle d_1, d_2, \ldots, d_i \rangle$ levaria a um número muito grande de parâmetros. Assim, usa-se uma função Φ para agrupar histórias em classes de equivalência, o que nos dá

$$P(s,a) = P(\langle d_1, d_2, \ldots, d_n \rangle) = \prod_{i=1\ldots n} P(d_i | \Phi(d_1, \ldots, d_{i-1})). \tag{7}$$

A função Φ pode ser definida tanto manualmente quanto automaticamente através de técnicas de aprendizado computacional.

O mapeamento entre eventos em $S \times A$ e seqüências de decisão é alcançado definindo-se um programa estocástico que gera eventos em $S \times A$. Um programa estocástico é um algoritmo que em certos pontos faz uma escolha aleatória entre decisões alternativas, de acordo com alguma distribuição de probabilidade. O traço do programa pode ser representado como a seqüência

[2]Do inglês *History-Based Models*

de decisões feita, e a probabilidade desta seqüência é o produto das probabilidades das diferentes decisões. O programa, então, define uma distribuição sobre seqüências de decisão, e uma distribuição sobre $S \times A$.

Se $\Phi(d_1, \ldots, d_{i-1}) = \alpha$, onde α é o não-terminal mais à esquerda na árvore parcial definida por $\langle d_1, \ldots, d_{i-1} \rangle$, então o programa estocástico é equivalente a uma GLCP, no sentido em que ele gera árvores com a distribuição definida pela GLCP com parâmetros $P(\alpha \rightarrow \beta | \alpha)$. Por outro lado, Φ pode ser estendida para incluir contextos arbitrários adicionais na árvore parcial definida por $d_1, \ldots, d_{i-1}$ (por exemplo, o pai de α na árvore, o símbolo diretamente à esquerda de α na árvore, e assim por diante). Assim, quebra-se as fortes hipóteses de independência consideradas nas GLPCs. Black et al. **[4]** descreve um método que usa árvores de decisão para buscar valores de Φ que incluam contexto adicional. Deste modo, enquanto este modelo baseado em história inclui GLCPs, ele também é poderoso o suficiente para estendê-los de várias maneiras.

Resumindo, a distinção com GLCPs ocorre no sentido de que cada estrutura de sintagma depende não somente da entrada, mas também da história inteira naquele ponto da sentença. A história é interpretada como qualquer elemento da árvore que já foi determinado, incluindo palavras anteriores, não-terminais, estruturas de sintagmas, e qualquer outra informação lingüística que é gerada como parte da estrutura da análise.

5.1. Modelos Baseados em História Condicionais. Modelos baseados em história também podem ser usados para definir distribuições condicionais $P(a|s)$ para $a \in A$ e $s \in S$. Trabalhos como **[27, 23, 16]** descrevem modelos condicionais para análise sintática; **[26]** descreve um modelo assim para etiquetagem morfossintática. Em modelos condicionais, o par s, a é representado novamente como uma seqüência de decisões, mas a entrada s é uma variável condicionante:

$$P(a|s) = P(\langle d_1, d_2, \ldots, d_n \rangle | s) = \prod_{i=1\ldots n} P(d_i | d_1, \ldots, d_{i-1}, s). \tag{8}$$

5.1.1. *O Analisador Sintático SPATTER.* Magerman (1995) **[23]** desenvolveu um analisador sintático chamado *SPATTER* (uma extensão do trabalho descrito em **[16]**), que utiliza um modelo de probabilidade condicional. O analisador foi completamente obtido a partir de um banco de árvores, sem a necessidade de uma gramática escrita à mão. É, portanto, de aprendizado **supervisionado**. A escolha da função Φ para definir os contextos de decisão (seção 5) foi implementada através de árvores de decisão para procura de hipóteses de independência. O analisador trabalha de baixo para cima até que uma árvore mínima seja construída, e a partir daí seleciona a melhor.

Considerando as três etapas para o projeto de um analisador sintático (veja página 11), Magerman usou um especialista em lingüística para definir uma

representação (etapa 1) que contivesse todos os elementos de árvore-sentença que pudessem ser úteis para a resolução de ambigüidades; e então escolheu usar árvores de decisão para identificar quais características seriam realmente significativas como critério de decisão (etapa 3).

O problema desta estratégia é a pouca importância dada à etapa 2 (de parametrização, ou decomposição) do modelo. O SPATTER escolhe parâmetros que algumas vezes perdem informações de distinção importantes, e outras vezes fragmentam desnecessariamente os dados de treinamento nas ramificações das árvores de decisão.

Na seção 5.2 apresentamos o trabalho de Collins **[10]**, que baseia o relativo sucesso de seu analisador sintático à importância dada à etapa 2, em que é tomado especial cuidado na escolha dos parâmetros que quebram e classificam as árvores.

O treinamento do SPATTER é feito através do algoritmo *Progressivo-Regressivo*[3] **[3, 7]**, que refina o modelo aumentando o peso das decisões que contribuem mais, convergindo em um modelo que melhor representa os dados. Este algoritmo é uma variação do algoritmo de Maximização de Esperança (ME) **[12]**, e parecido com o algoritmo Interno-Externo, descrito no apêndice A.

Este trabalho apresentado por Magerman representou uma maturação das técnicas de análise sintática probabilística. Ele representou um avanço na escala das tarefas feitas por analisadores sintático. Todas sentenças com até 40 palavras foram analisadas, em um domínio mais amplo (*WSJ*) do que testes anteriores. O analisador obteve resultados de 84,6% de cobertura e 84,9% de precisão (e 84,7% de medida-f). Provavelmente muito da melhora obtida em relação aos métodos baseados em GLCPs (uma GLCP **não-lexicalizada** alcança em média 72% de precisão/cobertura, conforme seção 3) se deve ao fato do modelo usar parâmetros que se baseiam fortemente em informações léxicas.

5.1.2. *Modelo Baseado em Máxima Entropia.* Ratnaparkhi propõe um modelo de analisador sintático baseado em uma técnica de aprendizado de máquina chamada de Máxima Entropia **[28]**. O modelo é treinado com um banco de árvores anotado manualmente, e portanto é **supervisionado**.

A construção da árvore sintática é feita com ações semelhantes às de um analisador de empilhar-reduzir. A seqüência de ações $a_1, \ldots, a_n$ para construir uma árvore completa é chamada de derivação de A. O analisador parte de uma derivação $d = a_1, \ldots, a_n$ e prediz alguma ação a_{n+1} para criar uma nova derivação, $d' = a_1, \ldots, a_{n+1}$. Não há uma gramática que dita quais ações são possíveis.

[3]Do inglês *Forward-Backward*.

Todas as ações que levam a árvores bem-formadas são permitidas e o método de máxima entropia é usado para pontuá-las. A pontuação de cada ação na derivação é usada para computar a pontuação da derivação inteira. O analisador usa um **modelo condicional baseado na história**, na qual uma probabilidade $p_x(a|d)$ é usada como a pontuação de uma ação a, dependendo da derivação parcial d (também chamada de contexto ou história) que está disponível no momento da decisão. O modelo condicional p_x é estimado usando-se máxima entropia, que pode usar diversas informações no contexto d para computar a probabilidade de uma ação a.

Para analisar uma sentença, o analisador usa uma rotina de busca que explora o espaço de todas as árvores possíveis e tenta encontrar a de maior probabilidade.

Com esse modelo, Ratnaparkhi alcança resultados de 87,5%/86,3% de precisão/cobertura (e 86,9% de medida-f) no WSJ.

5.2. Modelo Orientado ao Núcleo Léxico. O trabalho desenvolvido por Collins [**10**] primeiro partiu de um modelo puramente baseado em dependências léxicas, que havia sido desenvolvido em 1995. Com este modelo já obtia-se uma precisão em torno de 85% no WSJ. A partir daí, novos elementos lingüísticos foram inseridos no modelo, na tentativo de torná-lo mais exato (mas com isso tornando-o também mais complexo).

Collins argumenta que deve-se dar muita importância à parametrização do modelo (a etapa 2 do projeto de um modelo de análise sintática, que descrevemos na página 11). Por isto ele considera dois pontos críticos:

(1) Como as árvores sintáticas podem ser quebradas em fragmentos menores?
(2) Como esta escolha pode ser feita para se obter um modelo estatístico?

Esta ênfase na parametrização é explicada por Collins como devida à propriedade da localidade que as palavras em uma sentença possuem, que define o domínio no espaço sobre o qual a palavra núcleo influencia as demais palavras da sentença. Assim, o propósito é escolher uma parametrização que reflita melhor a influência local dos itens lexicais núcleos. Portanto, o item lexical núcleo deve ser gerado antes da estrutura que depende dele, numa derivação chamada de *dirigida ao núcleo*[4]. Com esta decomposição, aplica-se hipóteses de independência sobre relacionamentos que estão fora do domínio de influência (localidade).

Assim, neste modelo proposto por Collins, uma árvore sintática é representada como uma seqüência de decisões em uma ordem de cima para baixo, com

[4]Do inglês *Head-driven*.

uma derivação centrada no núcleo, em que cada decisão tem uma probabilidade associada. O modelo, então, chamado de Modelo Orientado ao Núcleo Léxico, é **gerativo**, dirigido ao núcleo, e probabilístico.

Nos testes feitos na seção 23 do WSJ, o modelo proposto por Collins conseguiu uma taxa de 88,3%/88,0% de cobertura/precisão (e 88,1% de medida-f) na recuperação dos sintagmas. Isso representa uma taxa relativa de redução de erro em torno de 25% sobre os resultados do SPATTER, quando treinados e testados com o mesmo conjunto de dados.

5.2.1. *Modelos Adaptados para o Português.* O trabalho realizado por Collins naturalmente foi aplicado ao Inglês, mas Sousa [**11**] e Bonfante [**5**] apresentam analisadores sintáticos para o Português baseados no trabalho de Collins.

Entretanto, os resultados obtidos pelos dois são inferiores aos obtidos para o Inglês. Ambos trabalhos chegam a conclusões semelhantes quando apresentam algumas razões para que isto tenha acontecido: o fato principal de terem contado com poucos dados de treinamento (às vezes 10 vezes menores que para o Inglês); e o fato de existir uma pequena gama de conhecimento coletado sobre o problema da análise sintática do Português.

Os dois trabalhos explicam a adaptação do método de Collins para o Português, e contribuem para o conhecimento da área quando relacionam as dificuldades e os problemas encontrados no desenvolvimento de analisadores sintáticos para o Português.

6. GLCPs Não-Lexicalizadas

Klein e Manning (2003) [**20**] apresentam um analisador sintático baseado em GLCPs **não-lexicalizadas**. As suposições de liberdades de contexto feitas por GLCPs são muito fortes, e enfraquecê-las neste sentido melhora bastante o modelo. Collins (1999) [**10**] (seção 5.2) faz isto, ao usar várias subcategorizações cuidadosamente projetas à mão e motivadas lingüisticamente para quebrar as má suposições de liberdade de contexto. Klein e Manning utilizam uma *markovização* das regras, considerando o contexto vertical e horizontal de um nó.

Esta GLCP *não-lexicalizada* supera a GLCP *lexicalizada* de Magerman (1995) [**23**], usada no analisador sintático SPATTER (apresentado na seção 5.1.1), mas não supera modelos mais recentes, como Charniak (1997) [**8**] (seção 4) e Collins (1999) [**10**]. Algumas vantagens de uma GLCP não-lexicalizada são: é mais fácil de interpretar e melhorar do que modelos lexicalizados mais complexos; a representação da gramática é muito mais compacta, não necessitando mais de grandes estruturas para armazenar probabilidades lexicalizadas;

os algoritmos de análise têm complexidade assintótica mais baixa[5] e têm constantes de gramática muito menores; e uma GLCP não-lexicalizada é mais simples de construir e otimizar.

O modelo usa aprendizado **supervisionado**, através de estimações de Máxima-Verossimilhança. Para a análise, o analisador usa o algoritmo CKY [**14**], semelhante ao algoritmo de Earley descrito no apêndice B. Usando o WSJ com sentenças de até 40 palavras, o modelo obteve 86,9% de precisão, 85,7% de cobertura e 86,3% de medida-f. Para sentenças com até 100 palavras, a precisão foi de 86,3%, a cobertura 85,1% e a medida-f 85,7%.

7. Modelo de Contexto de Constituintes

Klein e Manning (2002) [**19**] apresentam um modelo distribucional **gerativo** para indução **não-supervisionada** de sintaxe de linguagem natural. A busca de parâmetros é feita utilizando Maximização de Entropia. Na maioria dos experimentos, seqüências de etiquetas morfossintáticas são tomadas como entrada, mas também testa-se como entrada etiquetas induzidas distribuidamente.

A busca de parâmetros é local, e por isto parâmetros que são ótimos localmente podem ser ruins globalmente. Mas ela tem a vantagem potencial de, por agregar somente estruturas sintáticas válidas e completas de cada sentença, incorporar naturalmente a restrição de que constituintes não podem se cruzar.

Para explorar este e outros benefícios da busca de parâmetros, Klein e Manning usam um modelo novo que é projetado especificamente para permitir um espaço de busca mais apropriado. A suposição fundamental é que constituintes aparecem em contextos de constituintes. Um fenômeno ligüístico particular que o sistema explora é que constituintes longos geralmente têm equivalentes curtos e comuns, que aparecem em contextos similares e cujos "corpos de constituintes"[6] são facilmente descobertos. O modelo é projetado para transferir o corpo de constituintes de uma seqüência diretamente para o contexto que o contém, que então tenta pressionar novas seqüências que ocorrem nesse contexto a serem analisadas como constituintes na rodada seguinte. O modelo também é projetado para explorar as vantagens da "aglomeração distribuída"[7], e pode igualmente ser visto como executar aglomeração distribuída na presença de restrições de não-sobreposição. O algoritmo de indução combina os benefícios da busca de parâmetros baseada em Máxima Entropia e dos métodos de aglomeração distribuída.

[5] $O(n^3)$ vs. $O(n^5)$ (ou vs. $O(n^4)$ para uma implementação melhor).
[6] Do inglês *constituency*, de difícel tradução.
[7] Do inglês *distributional clustering*.

Os experimentos foram feitos utilizando as sentenças do WSJ que possuíam não mais que 10 palavras depois da remoção de pontuação e elementos nulos (WSJ-10). Constituintes que não podiam ser errados (como uma só palavra e sentenças inteiras) foram descartados. Utilizando etiquetas morfossintáticas como entrada, o modelo alcançou 63,8% de precisão, 80,2% de cobertura e 71,1% de medida-f. Isto reduz a esparsidade dos dados, e torna mais fácil enxergar um esboço da gramática, mas também limita o máximo que este modelo pode alcançar. Mesmo assim, algumas etiquetas morfossintáticas dão dicas de conhecimento posterior da estrutura, isto é, carregam distinções que somente poderiam ser feitas sintaticamente. Klein e Manning então fazem os experimentos utilizando como entrada etiquetas induzidas distribuidamente. Com isto, a precisão vai para 56,8%, a cobertura para 71,1% e a medida-f para 63,2%.

8. Modelo de Dependência e Corpo de Constituintes

Klein e Manning (2004) **[21]** apresentam um modelo **gerativo** para o aprendizado **não-supervisionado** de estruturas de dependência. O modelo é chamado de Modelo de Dependência com Valência (MDV), e utiliza as idéias principais que tornaram os modelos de dependência supervisionada eficazes na análise sintática estatística. O treinamento é feito com o algoritmo *interno-externo* **[2]** (veja apêndice A).

Este modelo obtém 46,6% de precisão, 59,2% de cobertura e 52,1% de medida-f. Estes valores são baixos, mas são os primeiros resultados publicados a superar a heurística da palavra adjacente (de 33,6% neste córpus).

Klein e Manning (2004) e Klein (2005) **[18]** então demonstram como este modelo pode ser combinado com o (melhor) modelo de indução de constituintes existente **[19]**, explicado na seção 7, para produzir uma combinação que supera substancialmente qualquer um dos modelos individuais, em qualquer medida. As vantagens de cada modelo são complementares. O modelo de constituintes (MCC) é melhor na recuperação de corpos de constituintes, e o modelo de dependência (MDV) é melhor na recuperação de estruturas de dependência. No modelo combinado, a pontuação (avaliação) de cada árvore é dada pelo produto das probabilidades de cada modelo acima.

Considerando etiquetas morfossintáticas como entrada, o modelo combinado (MDV+MCC) obtém 69,3% de precisão, 88,0% de cobertura e 77,6% de medida-f, no córpus WSJ-10. O modelo também foi testado com classes de palavras induzidas automaticamente, usando o método de aglomeração distribuída mais simples de Schütze (1995) **[29]**. Com estas classes como entrada, o modelo apresentou uma degradação, obtendo 65,2% de precisão, 82,8% de cobertura e 72,9% de medida-f. Entretanto, é interessante notar que estes números são totalmente não-supervisionados.

O modelo combinado também é testado para o Alemão (no córpus NEGRA-10) e para o Chinês (CTB-10), e também obtém bons resultados. Com o córpus NEGRA-10, a medida-f obtida é de 63,9%. Com o CTB-10, ela vale 43,3%. Neste caso, o modelo de dependências (MDV) separado apresentou melhor resultado: 46,7% de medida-f.

Uma razão principal para estes modelos serem capazes de recuperar estruturas mais precisamente que trabalhos anteriores é que eles minimizam a quantidade de estruturas ocultas que precisam ser induzidas. Os resultados obtidos demonstram que a ampla estrutura de constituintes e dependências de uma linguagem pode ser recuperada com bastante sucesso a partir de uma quantidade muito modesta de dados de treinamento.

CAPíTULO 4

Discussão

Na tabela 1 são mostrados os resultados alcançados pelos analisadores que apresentamos. As entradas em itálico são os modelos não-supervisionados. A coluna **P** indica a precisão, e a **C** a cobertura. Os valores na coluna $\leq$ indicam para sentenças de até quantas palavras os modelos foram testados. A anotação *PoS* indica que modelo usa como entrada etiquetas morfossintáticas, e a anotação *Distr.* indica que o modelo induz (distribuidamente) aglomerações de palavras. Mesmo que os modelos que usam aprendizado não-supervisionado

Modelo	**% P**	**% C**	$\leq$	Córpus
GLCP	70,6	74,8		
Magerman (1995) SPATTER	84,9	84,6	40	WSJ
Charniak (1997) melhor	87,4	87,5	40	WSJ
Ratnaparkhi (1999) EM	87,5	86,3		WSJ
Collins (1999) MONL	88,7	88,6	40	WSJ
KleinManning (2002) MCC (PoS)	*63,8*	*80,2*	*10*	*WSJ*
KleinManning (2002) MCC (Distr.)	*56,8*	*71,1*	*10*	*WSJ*
KleinManning (2003) GLCP não-lex.	86,9	85,7	40	WSJ
KleinManning (2003) GLCP não-lex.	86,3	85,1	100	WSJ
KleinManning (2004) MDV	*46,6*	*59,2*	*10*	*WSJ*
KleinManning (2005) MDV+MCC (PoS)	*69,3*	*88*	*10*	*WSJ*
KleinManning (2005) MDV+MCC (Distr.)	*65,2*	*82,8*	*10*	*WSJ*
KleinManning (2005) MDV+MCC	*49,6*	*89,7*	*10*	*NEGRA*
KleinManning (2005) MDV+MCC	*33,3*	*62,0*	*10*	*CTB*

TABELA 1. Resultados dos vários modelos de análise sintática mostrados.

possuam precisão bastante abaixo dos modelos supervisionados, eles são os melhores resultados publicados até hoje, e mostram que o potencial desse tipo de aprendizado é real e viável. Isto incentiva pesquisas nesta área para o Português, que atualmente conta com muitos poucos córpus anotados sintaticamente para conseguir obter um desempenho satisfatório na análise sintática supervisionada.

Pelos modelos de análise sintática apresentados, vimos que o estado-da-arte para o Inglês está progredindo. Já para o Português, temos poucos modelos

(bem) adaptados, quanto mais criados especificamente. Os poucos recursos lingüísticos disponíveis talvez estejam desencorajando mais pesquisas nesta área. Este trabalho serve como guia para futuras pesquisas, mostrando o avanço dos modelos de análise sintática, principalmente quanto à utilização de modelos não-supervisionados para o aprendizado lingüístico. Assim, para que mais esforço de pesquisa nesta área seja empregado para o Português, estes modelos podem ser a chave inicial. Considerando os resultados obtidos, o melhor modelo não-supervisionado apresentado parece ser capaz de fornecer tal chave. Os trabalhos futuros são, portanto, analisar a fundo este modelo e adaptá-lo para o Português. Abrimos mão de analisar a utilização de cadeias de Markov de alcance variável diretamente em algum modelo, ao contrário do que havíamos proposto no Plano de Estudos desta disciplina, porque isto significaria que precisaríamos utilizar modelos de aprendizado supervisionado, o que não parece ser a melhor idéia para o Português hoje. Entretanto, mesmo que muito esforço seja necessário para adaptar um modelo não-supervisionado — por causa das diferenças entre linguagens —, uma vez que se alcance resultados satisfatórios a geração de córpus anotado sintaticamente para o Português ficará bastante simplificada e rápida. E com isto, analisadores sintáticos mais eficientes poderão ser criados.

Referências Bibliográficas

[1] J. Allen. *Natural Language Understanding*. Benjamin/Cummings Publishing, 1995.

[2] James K. Baker. Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pages 547–550, 1979. *apud:* [**19**].

[3] L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of a markov process. *Inequalities*, 3:1 – 8, 1972. *apud:* [**24**].

[4] Ezra Black, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, pages 134–139, Harriman, NY, February 1992. *apud*: [**24, 10**].

[5] Andréia Gentil Bonfante. *Parsing Probabilístico para o Português do Brasil*. Tese de doutorado, Programa de Pós-Graduação em Ciência da Computação, Instituto de Ciências Matemáticas e Computação, Universidade de São Paulo, Junho 2003. `http://www.nilc.icmc.usp.br/nilc/download/TeseFinalBonfante.zip`.

[6] Eric Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, 1993. *apud*: [**10**].

[7] Eugene Charniak. *Statistical Language Learning*. The MIT Press, 1993.

[8] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *AAAI/IAAI*, pages 598–603, 1997. `citeseer.ist.psu.edu/charniak97statistical.html`.

[9] Eugene Charniak, Curtis Hendrickson, Neil Jacobson, and Mike Perkowitz. Equations for part-of-speech tagging. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 784–789, Menlo Park, 1993. AAAI Press/MIT Press. *apud:* [**7**]. `citeseer.nj.nec.com/34373.html`.

[10] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. Tese de doutorado, Department of Computer and Information Science, University of Pennsylvania, 1999. `http://people.csail.mit.edu/u/m/mcollins/public_html/`.

[11] Fabiano de Carvalho e Sousa. Analisador sintático estatístico orientado ao núcleo-léxico para a língua portuguesa. Dissertação de mestrado, Programa de Pós-Graduação em Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, Outubro 2003.

[12] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39:1—38, 1977. *apud*: [**10, 24**].

[13] U. Hermjakob and R. J. Mooney. Learning parse and translation decisions from examples with rich context. In *Proceedings of the 35th Annual Meeting of the Association*

for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics, pages 482—489, Madrid, Spain, 1997. ACL. *apud:* [**10**].

[14] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979. *apud*: [**24, 10**].

[15] IEL-UNICAMP and IME-USP. *Corpus Anotado do Português Histórico Tycho Brahe.* `http://www.ime.usp.br/~tycho/corpus`, acessado em 2005. `http://www.ime.usp.br/~tycho/corpus`.

[16] Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, Adwait Ratnaparkhi, and Salim Roukos. Decision tree parsing using a hidden derivation model. In *Proceedings of the 1994 Human Language Technology Workshop*, pages 272—277, 1994. *apud*: [**24, 10**].

[17] Fábio Natanael Kepler. Um etiquetador morfo-sintático baseado em cadeias de markov de tamanho variável. Dissertação de mestrado, Programa de Pós-Graduação em Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, Abril 2005. `http://www.ime.usp.br/~kepler/`.

[18] Dan Klein. *The Unsupervised Learning of Natural Language Structure.* Ph.d. thesis, Stanford University, 2005. `http://www.cs.berkeley.edu/~klein/`.

[19] Dan Klein and Christopher D. Manning. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting of the ACL*, 2001.

[20] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, 2003.

[21] Dan Klein and Christopher D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting of the ACL*, 2004.

[22] B. Santorini M. Marcus and M. Marcinkiewicz. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313 — 330, 1993.

[23] David M. Magerman. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276—283, 1995.

[24] Christopher D. Manning and Hinrich Schütze. *Foundations Of Statistical Natural Language Processing.* The MIT Press, 1999.

[25] Tom M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Departament of Computer Science, Rutgers University, 1980. *apud*: [**24**].

[26] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Conference on Empirical Methods in Natural Language Processing*, May 1996.

[27] Adwait Ratnaparkhi. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, Brown University, Providence, Rhode Island, 1997. *apud*: [**24, 10**].

[28] Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151 — 175, February 1999.

[29] Hinrich Schütze. Distributional part-of-speech tagging. *EACL*, 7:141—148, 1995. *apud*: [**21, 19, 24**].

APÊNDICE A

Algoritmo de Treinamento

Um algoritmo bastante usado para treinar uma GLCP é o chamado *Algoritmo Interno-Externo*[1] **[2]**, que é uma instância do algoritmo de Maximização de Esperança (ME) **[12]**. Ele utiliza um córpus de treinamento e as árvores sintáticas de treinamento para tentar aumentar a probabilidade das regras mais utilizadas destas árvores. Primeiro o algoritmo calcula as probabilidades interna e externa de uma sentença.

Seja $w_{1,n}$ uma sentença do corpus de treino, onde w_i denota a i-ésima palavra da sentença. O símbolo terminal i representa w_i.

A probabilidade interna $\beta_X(k,l)$ é a probabilidade do não-terminal X dominar os terminais de k a l ($[k,l]$), ou seja, é a probabilidade dos terminais "dentro" de $X_{k,l}$. Seja S um símbolo inicial da gramática. Então, quando tivermos calculado as probabilidades internas da sentença, teremos a probabilidade da sentença:

$$\beta_S(1,n) = P(w_{1,n})$$

Para calcular as probabilidades internas eficientemente, começamos com o caso base:

$$\beta_X(k,k) = P(X \to w_k)$$

E então, considerando o conjunto das regras usadas para criar $X_{k,l}$, iteramos por:

$$\beta_X(k,l) = \sum_{X_{k,l} \to Y_{k,\ldots} \cdots Y_{\ldots,l}} P(X_{k,l} \to Y_{k,\ldots} \cdots Y_{\ldots,l}) \prod_{Y_{p,q} \in Y_{k,\ldots} \cdots Y_{\ldots,l}} \beta_Y(p,q) \tag{9}$$

A probabilidade externa é definida por $\alpha_X(k,l) = P(w_{1,k_1}, X, w_{l+1,n})$, ou seja, é a probabilidade de produzir o material "fora" de X enquanto X domina o resto do material. A probabilidade externa é calculada do topo da árvore para baixo. Assim o caso base é:

$$\alpha_S(1,n) = P(S_{1,n}) = 1.0$$

Continuamos o processo por:

$$\alpha_X(k,l) = \sum_{Z_{a,b} \to [\cdots X_{k,l} \cdots]} \alpha_Z(a,b) \frac{P(Z_{a,b} \to [\cdots X_{k,l} \cdots]) \prod_{Y_{p,q} \in [\cdots X_{k,l} \cdots]} \beta_Y(p,q)}{\beta_X(k,l)}$$

[1] Do inglês *Inside-Outside Algorithm*.

Finalmente, para reestimarmos as probabilidades das regras, utilizamos a equação

$$P_e(X \to Y^1 \cdots Y^m) = \frac{C(X \to Y^1 \cdots Y^m)}{\sum C(X \to \cdots))}$$

onde $C(X \to Y^1 \cdots Y^m)$ = somatório da probabilidade de que X foi construído usando a regra $X \to Y^1 \cdots Y^m$ sobre todas as aplicações da regra, ou seja,

$$C(X \to Y^1 \cdots Y^m) = \frac{1}{P(w_{1,n})} \sum_{\text{ocorre } X \to Y^1 \cdots Y^m} \alpha_X(k,l) P(X \to Y^1 \cdots Y^m) \beta_{Y^1}(k, \cdots) \ldots \beta_{Y^m}(\cdots, l).$$

APÊNDICE B

Algoritmo de Earley

O algoritmo de Earley verifica se uma sentença é reconhecida pela gramática dada. Com alguma estrutura a mais, podemos fazê-lo encontrar as árvores sintáticas da sentença. E utilizando uma gramática probabilística, encontrar apenas a árvore sintática com maior probabilidade.

Primeiro, o algoritmo precisa de três estruturas de dados:

(1) Uma matriz de probabilidade contendo as regras instanciadas já fechadas e a probabilidade da árvore a partir de cada uma dessas regras;
(2) Uma pilha de entradas de símbolos instanciados a serem analisados;
(3) Um conjunto de regras instanciadas parcialmente.

A primeira estrutura é implementada utilizando `map`, e contém as regras instanciadas que já foram fechadas, junto com suas respectivas probabilidades. Ela é indexada por [índice superior − índice inferior][índice inferior]. Por exemplo, considerando o símbolo à esquerda da uma regra instanciada $NP_{1,2} \rightarrow D_{1,1}N_{2,2}$, a regra é armazenada na posição $[2-1][1]$. A terceira estrutura é fornecida pelo objeto da gramática passado ao algoritmo, e é implementada em parte pela classe de regra (`Rule`): cada regra instanciada possui a posição no lado direito em que atualmente se encontra o arco.

Um arco mostra até que ponto uma regra foi expandida, indicando a posição já reconhecida da sentença. Dessa forma, o arco da regra $SV_{1,j} \rightarrow VT_{1,2} \bullet SN_{2,j}$, por exemplo, mostra que os terminais entre 1 e 2 já foram reconhecidos.

Para encontrar a probabilidade de cada árvore sintática, calculamos a probabilidade interna do símbolo à esquerda de cada regra instanciada que se fecha. Utilizamos para isso uma variação da equação 9 da probabilidade interna, substituindo o cálculo do somatório pelo do máximo:

$$\beta_X(k,l) = \max_{X_{k,l} \rightarrow Y_{k,\ldots} \cdots Y_{\ldots,l}} P(X_{k,l} \rightarrow Y_{k,\ldots} \cdots Y_{\ldots,l}) \prod_{Y_{p,q} \in Y_{k,\ldots} \cdots Y_{\ldots,l}} \beta_Y(p,q) \tag{10}$$

O algoritmo realiza a análise sintática para cada sentença do corpus de teste, da seguinte maneira:

- Coloque na pilha, em ordem inversa, os símbolos terminais da sentença sendo analisada, instanciados por suas respectivas posições na sentença

- Enquanto houver símbolos na pilha:
  (1) Remova o símbolo no topo da pilha
  (2) Se ele já está na pilha, vá para a próxima iteração
  (3) Para toda regra da gramática, não-instanciada, que começa com esse símbolo, copie esta regra e a instancie, fazendo-a expandir seu arco uma posição para a direita, atualizando o índice inferior do símbolo à esquerda e do primeiro à direita com o índice inferior do símbolo sob o qual se expandiu o arco, ou seja, do símbolo retirado da pilha
  (4) Para toda regra instanciada da gramática, em que o símbolo à direita seguinte ao seu arco é igual ao símbolo retirado da pilha, com exceção do índice superior, copie essa regra e faça-a expandir seu arco uma posição para a direita, atualizando o índice superior do símbolo coberto pelo arco e o índice inferior do símbolo que ficou após o arco, se existir
  (5) Caso o arco de alguma regra seja terminado, feche a regra e
     - Atualize o índice superior do símbolo à esquerda com o índice superior do último símbolo à direita (o mesmo do símbolo retirado da pilha)
     - Calcule a probabilidade interna da regra conforme a equação 10. Se ela for maior que a probabilidade já encontrada na matriz de probabilidades, atualize a matriz com a regra e sua probabilidade
     - Adicione o símbolo à esquerda da regra à pilha
- Procure na matriz o símbolo inicial com maior probabilidade, e construa a árvore sintática de maior probabilidade a partir dele
- Armazene essa árvore em um conjunto, que depois será usado para verificar se as árvores encontradas são as corretas

Caso um símbolo terminal não seja conhecido pela gramática, isso é, caso não exista nenhuma regra derivando esse símbolo, a gramática retorna um conjunto de regras especiais. Essas regras são atualizadas para cada terminal

Depois que o algoritmo realiza estes passos para cada sentença, ele retorna ao analisador sintático o conjunto de árvores sintáticas encontradas.