Aporte da Descoberta de Conhecimento em Banco de Dados em Aplicações de Linguística Computacional: Estudo de Caso de Rotulação de Papéis Semânticos

Osmar Souza Sexto Alexandre

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Orientador: Prof. Tarcísio de Souza Lima

Juiz de Fora, Minas Gerais
Julho de 2009

Aporte da Descoberta de Conhecimento em Banco de Dados em Aplicações de Linguística Computacional: Estudo de Caso de Rotulação de Papéis Semânticos

Osmar Souza Sexto Alexandre

Monografia submetida ao corpo docente do Departamento de Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora, como parte integrante dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Aprovada pela banca constituída pelos seguintes professores:

_____

Profº. Tarcísio de Souza Lima – orientador
MSc. em Informática, PUC-RJ, 1988

_____

Profª. Tatiane Ornelas Martins Alves
Especialista em Computação, UFV, 2006

_____

Profº. Raul Fonseca Neto
DSc. em Engenharia de Sistemas e Computação, Coppe/UFRJ, 1990

Juiz de Fora, Minas Gerais
Julho de 2009

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Reduções

ARFF	*Attribute-Relation File Format*
DM	*Data Mining*
EI	Extração de Informação
FRED	*Frame Desambiguation*
HEX	*Hexidecimal*
IA	Inteligência Artificial
ISO	*International Organization for Standardization*
KDD	*Knowledge Discovery in Databases*
POS	*Part-of-Speech*
POST	*Part-of-Speech Tagging*
ROSY	*Role Assignment System*
SRL	*Semantic Role Labeling*
TXT	*Text File*
UTF-8	*8-bit Unicode Transformation Format*
XML	*Extensible Markup Language*

Resumo

A evolução da linguística é crescente e acompanha, em ritmo mais acelerado ainda, a evolução da tecnologia e das técnicas computacionais. Neste cenário, busca-se a automatização de muitas das técnicas linguísticas, com o suporte inerente dos algoritmos computacionais e de todo poderio tecnológico. Há décadas, pesquisas sobre a análise e rotulação sintática vêm sendo feitas. Mais recentemente, o advento da rotulação de classes e papéis semânticos vem ganhando notoriedade e passa a ocupar uma grande fatia do estudo sobre linguística computacional. Várias bases de dados linguísticas vêm sendo criadas e mantidas, armazenando unidades lexicais e sentenças. Combinando essas bases de dados aos algoritmos, classificadores e técnicas de programação, chega-se ao processo de rotulação sintática e semântica sobre sentenças e/ou textos. Além de toda a parte teórica envolvida nesse cenário de rotulação semântica, este trabalho também apresenta um experimento computacional, onde são feitas as rotulações sintática e semântica sobre um conjunto de sentenças da base de dados da *FrameNet*. O resultado desse estudo de caso é demonstrado graficamente.

Palavras-chave:

Knowledge Discovery, *Database*, *Semantic Role Labeling*, *Naïve Bayes*, *FrameNet*, *Shalmaneser*

Capítulo 1
Introdução

A evolução da tecnologia da informação traz consigo o rápido e inevitável aumento no volume de dados. Nas empresas, uma solução passa pela construção de *Data Warehouse*s, a fim de prover as informações de maneira organizada, qualificada e selecionada. Na Web, a tentativa é de estruturar um pouco da grande massa de dados semi-estruturados ou desestruturados. Os bancos de dados, a cada dia, passam a ser parte essencial e integrante no gerenciamento de negócios e no armazenamento de informações. Dentre os mais variados tipos, encontram-se os bancos de dados linguísticos, alvo de estudo neste trabalho.

O avanço, em termos linguísticos, é notável e bem definido. O desafio passa a ser o de tornar a linguística possível, computacionalmente. Dessa maneira, tem-se a intenção de automatizar, via algoritmos da linguística computacional, o processo de formação, transformação e identificação de sentenças. Tanto a análise sintática quanto a análise semântica são consideradas tarefas complexas, uma vez que as sentenças podem apresentar as mais variadas estruturas, contextos distintos, regras gramaticais, entre outros.

Nas últimas décadas, houve um imenso avanço na análise sintática de textos. Atualmente, esse mesmo avanço está ganhando força e notoriedade na área semântica, no que diz respeito à interação de predicado-argumento, que modela o relacionamento entre predicados (verbos, por exemplo) e seus argumentos semânticos (ou papéis semânticos). Primeiramente, um predicado é atribuído a um sentido ou classe semântica. Essa classe semântica está ligada a diversos possíveis papéis semânticos (*semantic roles)*.

Todo esse contexto cria a necessidade de criar algoritmos e classificadores que, juntos, rotulam automaticamente um determinado texto, sintática e semanticamente. Para tal tarefa, é de extrema importância um banco de dados bem robusto e populoso, com o maior número possível de sentenças e unidades lexicais. É nesse cenário que foi escolhida a base de dados da *FrameNet*[1] (FILLMORE, RUPPENHOFER e BAKER, 2004). Quanto maior for a base de dados, maior será a chance dos resultados serem mais satisfatórios, uma vez que mais situações estarão sendo catalogadas, analisadas e comparadas.

[1] Repositório on-line para a língua inglesa, cujo maior produto é uma base de dados linguística com mais de 10.000 unidades lexicais e 135.000 sentenças completamente anotadas. Disponível em http://framenet.icsi.berkeley.edu.

Os algoritmos e classificadores linguístico-computacionais continuam evoluindo, se adequando aos diversos idiomas e atingindo resultados cada vez mais satisfatórios. O uso de ferramentas específicas também traz vantagens, já que facilita o processamento, transformação e rotulação das sentenças, além da visualização dos resultados gerados.

1.1 Objetivos da Monografia

Espera-se, com este trabalho, relevar a importância da integração de duas grandes áreas de conhecimento: o suporte da descoberta de conhecimento em banco de dados à linguística computacional. Primeiramente, descreve-se o contexto dos bancos de dados, ressaltando-se o processo de *Knowledge Discovery in Databases* (*KDD*, em português Descoberta de Conhecimento em Base de Dados) e a técnica de *Data Mining* (*DM*, em português Mineração de Dados). A base de dados, conforme já dito, é a base de dados linguística da *FrameNet*.

Com o uso de software específico e de softwares de apoio e suporte, visa-se colocar em prática o processo de rotulação sintática e semântica de sentenças de um determinado texto. Objetiva-se que essas rotulações sejam eficazes e fiéis e que o resultado possa ser mostrado ao usuário, através de um ambiente gráfico.

1.2 Metodologia Utilizada

Primeiramente, foi feito um estudo sobre a base teórica de todos os assuntos que envolvem o tema. São abordados aspectos do *KDD*, com enfoque na etapa principal – a tarefa de mineração de dados. Em seguida, é feito um detalhamento sobre a base de dados *FrameNet* e sua relação envolvendo a linguística computacional. Ainda na parte teórica, é realizado um estudo sobre a Rotulação de Papéis Semânticos (SRL, do inglês *Semantic Role Labeling*) (GILDEA e JURAFSKY, 2002; SURDEANU *et al*., 2002; XUE e PALMER, 2004; PRADHAN *et al*., 2005), seu histórico e sua importância, em paralelo com a rotulação sintática.

Após o levantamento dos subsídios teóricos, foi realizado um experimento computacional envolvendo os temas analisados. Nesse experimento, foi simulada a implementação da rotulação sintática e dos papéis semânticos de sentenças através do uso da ferramenta *Shalmaneser* e das ferramentas auxiliares: *Collins*[2] (COLLINS, 1997),

[2] Collins é um *parser*, responsável pela análise sintática das sentenças. O classificador para a *FrameNet1.3* junto com o Collins *parser* está disponível para download em http://www.coli.uni-saarland.de

Mallet[3] (MCCALLUM, 2002) e *TreeTagger*[4] (SCHMID, 1996). O resultado é demonstrado graficamente através do *software Salto*[5]. Como a base de dados linguística escolhida foi a *FrameNet*, há a necessidade de que o arquivo texto de entrada possua frases no idioma inglês. Após o processamento desse arquivo (pelos componentes do *Shalmaneser* e pelas ferramentas auxiliares), um arquivo *XML* (*Extensible Markup Language*) é gerado e pode ser interpretado graficamente.

1.3 Estrutura da Monografia

Este trabalho diz respeito a como a descoberta de conhecimento em banco de dados pode dar suporte à tarefa de rotulação de papéis semânticos, no caso específico do estudo de caso na base de dados da *FrameNet*, com suporte da ferramenta *Shalmaneser*[6] (ERK e PADO, 2007). Assim, os capítulos foram divididos de acordo com os temas pertinentes à realização do trabalho, sendo no total, além desta visão geral do trabalho, cinco capítulos distribuídos como se segue.

O segundo capítulo aborda algumas definições e conceitos relacionados ao processo de descoberta de conhecimento em base de dados (*KDD*) e uma revisão teórica sobre sua principal etapa, a mineração de dados (*data mining*). O estudo foi realizado sobre uma base de dados linguística. Neste cenário, o apoio da descoberta do conhecimento é essencial, uma vez que se tem a intenção de descobrir resultados e relações compreensíveis e potencialmente úteis. Através da análise e exploração – como feitos no processo do *KDD* – da base de dados linguística da *FrameNet*, busca-se reconhecer padrões existentes e que dêem suporte à rotulação semântica de sentenças.

No terceiro capítulo, são apresentados os temas e definições referentes ao conceito de *SRL* e da tarefa de *POST* (do inglês, *Part-of-Speech Tagging*). Além disso, o capítulo trata da *FrameNet*, a base de dados linguística que será usada no experimento computacional deste trabalho.

O quarto capítulo introduz a ferramenta computacional *Shalmaneser*, usada na tarefa da rotulação de papéis semânticos. Além disso, são apresentadas breves descrições

[3] *Mallet* é uma ferramenta de aprendizado de máquina responsável pelo processamento de textos. Disponível para download em http://mallet.cs.umass.edu/
[4] *TreeTagger* é uma ferramenta responsável pelas tarefas de lematização e *Part-of-Speech Tagging*. Disponível para download em http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/
[5] *Salto* é um *software* de interface gráfica para anotações de papéis semânticos. Disponível em http://www.coli.uni-saarland.de/projects/salsa/page.php?id=software
[6] *Shalmaneser* está disponível para download em http://www.coli.uni-saarland.de/projects/salsa/page.php?id=software

sobre cada uma das ferramentas que auxiliam e dão suporte ao *Shalmaneser*: *Collins*, *TreeTagger*, *Mallet* e *Salto*. Ainda neste capítulo, é tratado mais especificamente o algoritmo de classificação *Naïve Bayes*.

O quinto capítulo apresenta efetivamente o estudo de caso, onde toda a teoria é colocada em prática e a rotulação dos papéis semânticos de determinado texto ou sentença é feita através do uso do *Shalmaneser*. O resultado gerado é exibido no ambiente gráfico *Salto*.

O sexto e último capítulo conclui o presente trabalho, relatando o conhecimento adquirido, limitações e dificuldades enfrentadas e sinaliza possíveis trabalhos futuros.

Por fim, são registradas as referências bibliográficas utilizadas para o desenvolvimento do trabalho.

Capítulo 2
Descoberta de Conhecimento em Banco de Dados

No passado, as limitações tecnológicas sempre foram obstáculos para o armazenamento de dados. A restrição de *hardware* impedia que grandes volumes de dados pudessem ser armazenados para posterior avaliação. Desta maneira, simples consultas aos bancos de dados já eram consideradas suficientes para análise dos dados e para a extração do conhecimento desejado. No presente, com o avanço tecnológico – tanto em relação aos *hardwares* quanto aos *softwares* – a capacidade de armazenamento de dados tornou-se imensa, deixando de ser um gargalo para a construção de grandes bases de dados. Os bancos de dados passaram a representar depósitos de conhecimento em potencial, ou seja, ao serem explorados, descobrem-se relações, padrões e regras. A análise manual dos dados tornou-se impraticável e ineficaz, passando a ser feita através do auxílio de um computador, que extrai as informações e os padrões existentes nos dados. É exatamente nesse contexto de grande crescimento do volume de dados em tamanho e dimensionalidade que surge o *KDD* (*Knowledge Discovery in Databases*), responsável por todo o processo de extração do conhecimento em um determinado banco de dados.

O *KDD* é um processo não trivial que envolve várias etapas distintas e iterativas, até que se atinja a meta final de extração de conhecimento (ROSINI, 2003), buscando encontrar uma maneira automatizada de explorar as bases de dados e reconhecer os padrões existentes. É responsável por identificar padrões previamente desconhecidos, implícitos, válidos, compreensíveis e potencialmente úteis (FAYYAD, PIATETSKY-SHAPIRO e SMYTH, 1996). O *KDD* pode ser considerado um processo multidisciplinar, envolvendo áreas relativas à matemática, estatística, aprendizado de máquina, reconhecimento de padrões, base de dados, aquisição de conhecimento para sistemas especialistas e visualização de dados. Para que se inicie o processo de *KDD* é necessário o entendimento do domínio da aplicação e dos objetivos finais a serem atingidos. Logo após essa etapa, deve-se fazer um agrupamento organizado de uma massa de dados, que serão alvo da análise. Com os dados organizados, dá-se início à etapa de limpeza (*data cleaning)* e pré-processamento dos dados. Os dados pré-processados passam agora pela etapa de transformação, que armazenará os dados de uma forma adequada ao uso das técnicas de mineração de dados. Seguindo no processo de *KDD*, chega-se à etapa de *Data Mining*

(*DM*) especificamente, que começa com a escolha dos algoritmos a serem aplicados. A maioria dos métodos de *DM* é baseada em conceitos de aprendizagem de máquina, reconhecimento de padrões, classificação e clusterização. Essa etapa é responsável, de fato, pela descoberta de padrões nos dados e consequente geração do relatório das descobertas. A etapa seguinte é justamente a interpretação desse relatório, que contém o modelo descoberto e há de ser interpretado por um analista de mineração. Somente após a interpretação das informações obtidas chega-se ao conhecimento. A utilização do conhecimento obtido inclui sua incorporação no desempenho do sistema, tomando ações baseadas no conhecimento, ou simplesmente documentando e reportando para grupos interessados. (FAYYAD *et al.*, 1996)

2.1 Etapas do *KDD*

O processo de *KDD* é modularizado e dividido em cinco etapas distintas: seleção de dados, limpeza e pré-processamento, transformação, mineração de dados e interpretação. A seguir, cada uma dessas etapas está descrita detalhadamente.

2.1.1 Seleção de Dados

Após a definição do domínio sobre o qual se deseja executar o processo de descoberta, deve-se selecionar e coletar o conjunto de dados ou as variáveis necessárias. Caso haja diferentes bases de dados, será exigido um trabalho de compatibilidade, para que haja a alimentação de um único banco de dados, através de várias fontes distintas.

2.1.2 Limpeza e Pré-processamento

É a etapa responsável pela adequação dos dados aos algoritmos usados posteriormente. Uma série de tratamentos e adequações é feita, incluindo tratamento de dados inconsistentes, eliminações de registros repetidos, tratamento de incompletude dos dados, problemas de tipagem e demais ruídos.

2.1.3 Transformação

É a etapa onde os dados sofrem uma determinada transformação, de acordo com os dados alvos adequados. É uma das etapas mais importantes do processo de *KDD* no que se refere aos dados, uma vez que a geração do conhecimento através do *Data Mining* dependerá desses dados transformados.

2.1.4 Mineração de Dados (*Data Mining)*

É nesta etapa onde a descoberta de conhecimento acontece de fato. Esta fase inicia-se com a escolha dos algoritmos que serão aplicados, de acordo com o objetivo do processo de *KDD* (classificação, clusterização, regras associativas etc.). De modo geral, é nessa fase em que as ferramentas especializadas buscam por padrões nos dados. Esse processo de busca é costumeiramente iterativo, ou seja, os analistas revêem os resultados, formam um novo conjunto de questões para refinar a busca e realimentam o sistema com novos parâmetros. No final do processo de *DM*, é gerado um relatório de descobertas, que será interpretado pelos analistas. Somente após essa interpretação é que se tem o conhecimento desejado. Essa etapa de *Data Mining*, por ser considerada a principal, está descrita detalhadamente mais à frente.

2.1.5 Interpretação

Os resultados do processo de *KDD* podem ser mostrados de diversas formas, que devem ser criteriosas o suficiente para que se consiga identificar a necessidade de retornar a qualquer uma das etapas anteriores do processo de *KDD*, uma vez que este é interativo e iterativo.

A figura 2.1 ilustra as etapas do *KDD* aqui enunciadas.

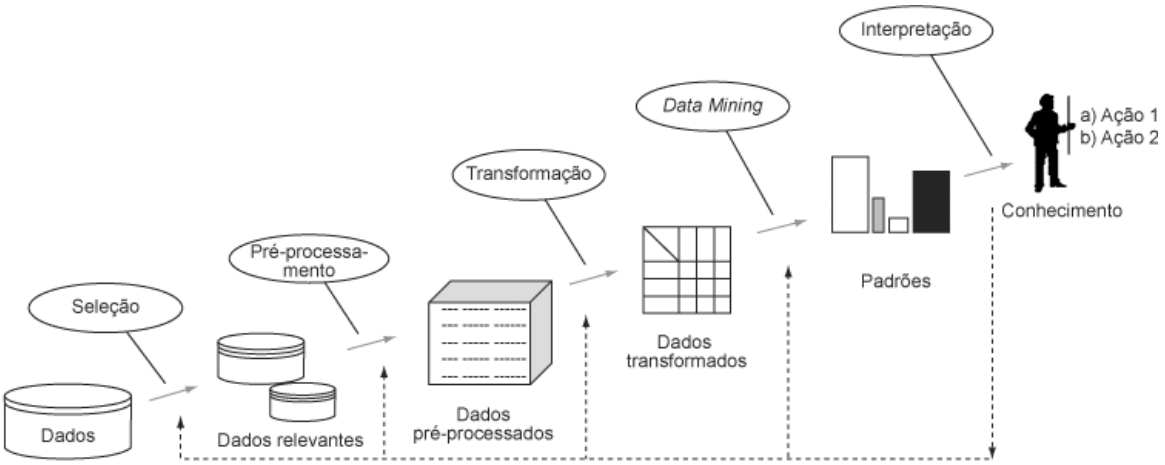

Figura 2.1 – Etapas do processo de *KDD* (adaptada de FAYADD *et al.*, 1996)

2.2 Mineração de Dados (*Data Mining*)

Com o passar dos anos, a tecnologia vem tornando cada vez mais pertinente o acúmulo acelerado de dados, gerando bancos de dados enormes. Entretanto, a análise dessa grande quantidade de dados ainda é precária, dispendiosa, demorada, ineficaz, ineficiente e pouco

automatizada. A automatização dos processos de análises de dados tornou-se uma necessidade e é nesse contexto que aparecem as técnicas do processo de mineração de dados.

O *Data Mining* é o processo de análise de volumosos conjuntos de dados que tem como objetivo a descoberta de padrões interessantes, a fim de representar informações úteis. Essa busca por padrões consistentes e válidos detecta relacionamentos sistemáticos entre variáveis, podendo formar novos subconjuntos de dados.

O processo de *DM* é uma das etapas mais importantes e vitais do processo de *KDD*, uma vez que é ao final dela que o conhecimento útil será retido de fato. Nesse momento, os dados já passaram por todas as etapas de adequação: seleção, limpeza, pré-processamento e transformações; e se encontram prontos para os algoritmos de mineração. Esses algoritmos são os responsáveis pela descoberta e extração de padrões. Esses padrões são nada mais do que unidades de informação que se repetem ou sequencias de informações que dispõe de uma estrutura que se repete e podem ser divididos em preditivos ou descritivos. Os padrões preditivos são justamente as descobertas que prevêem o valor futuro de um atributo a partir do valor conhecido dos demais atributos. Já os padrões descritivos têm caráter informativo e encontram padrões interessantes que descrevem os dados de forma compreensível para o homem.

A mineração de dados é uma área multidisciplinar, pois está relacionada com banco de dados, estatística, computação de alto desempenho, visualização, matemática e inteligência artificial. Dentro da inteligência artificial, tem-se que a mineração de dados está intimamente ligada com o aprendizado de maquina, um sub-campo da IA (Inteligência Artificial). O aprendizado de máquina se dedica ao desenvolvimento de técnicas e algoritmos que permitem que o computador aprenda e aperfeiçoe seu desempenho em uma determinada tarefa. Em se tratando de *data mining*, é necessário que os algoritmos sejam eficientes e de alta escalabilidade. O aprendizado divide-se em: supervisionado (método que requer análise dos dados para identificar um campo alvo) e não supervisionado (método que permite que os algoritmos encontrem padrões nos dados, independente de um alvo pré-estabelecido). O aprendizado supervisionado conta com as tarefas de classificação e análise de desvios, enquanto o aprendizado não supervisionado conta com as tarefas de associação e agrupamento. Essa divisão e as respectivas técnicas utilizadas podem ser melhor visualizadas na tabela 2.1.

Tabela 2.1: Divisão, Tarefas e Técnicas do Aprendizado de Máquina

Aprendizado	**Tarefa**	**Técnica**
Não Supervisionado	Associação	Regras de Associação Padrões Sequenciais
	Agrupamento	Clusterização
Supervisionado	Classificação	Regras de Indução Árvores de Decisão MBR – *Memory Based Reasoning* Redes Neurais
	Análise de Desvios	Árvores de Decisão Redes Neurais

2.3 Extração de Informações

Com o avanço da tecnologia e o advento da Internet e da Web, mais e mais informações estão eletronicamente disponíveis, seja por meio de discos, seja através da Web. O grande problema é que grande parte dessa informação é não-estruturada. Uma maneira de se encontrar automaticamente informações realmente interessantes e potencialmente úteis nesse imenso bando de dados é empregar técnicas de Extração de Informação – EI (do inglês *Information Extraction*).

A EI em textos possui, tipicamente, três etapas. Na primeira etapa, a informação necessária é abstraída e expressada por um grupo estruturado de categorias que se relacionam entre si. Este grupo estruturado recebe o nome de *template*, enquanto as categorias recebem o nome de *slots*. Por exemplo, caso se queira extrair informações onde o domínio seja viagens de avião, há interesse sobre o local de origem, o local de destino, a trajetória, o tempo de percurso e as condições climáticas. Então se deve gerar um *template* cujas categorias são LOCAL_ORIGEM, LOCAL_DESTINO, TRAJETORIA, TEMPO_PERCURSO e CONDICOES_CLIMATICAS. Na segunda etapa, uma vez que o *template* de extração já é conhecido, há de se identificar trechos de textos que contêm informações que completam os *slots* (categorias) do *template*. O reconhecimento de informação textual de interesse é um resultado entre o casamento de padrões com as regras de extrações. Por exemplo, caso se deseje extrair informações sobre viagens de avião,

precisamos reconhecer tanto os tipos de vôos, locais de origem, locais de destino, tempo de percurso quanto todas as alternativas sintáticas de expressões que induzem a vôos de avião. Na terceira e última etapa da extração de informação, é necessário mapear as informações de interesse identificadas em seus respectivos *slots*. Todas essas etapas da extração são dependentes do conhecimento dos eventos, estados e entidades pertencentes ao domínio do conhecimento. Um exemplo análogo, no contexto da *SRL*, seria a unidade lexical "querer" invocando dois possíveis *frames:* o de "desejo" e o de "ordem". Na frase "Eu quero laranja", o verbo "querer" tem sentido de "desejo", "vontade". Já na frase "Eu quero o relatório pronto amanhã", o sentido do verbo "querer" passa a ser o de "ordenação". Nesta situação, o *template* seria o verbo "querer", enquanto os *slots* (categorias) seriam "desejo" e "ordem".

É visível a importância do processo de *KDD* no que diz respeito à extração de conhecimento compreensível e útil das mais variadas áreas. É por conta disso que o *KDD* é tido como um processo multidisciplinar. No tocante à linguística computacional, não é diferente. O *KDD* dá suporte à descoberta de padrões linguísticos e, através da preparação, análise e exploração da base de dados (neste trabalho, a *FrameNet*), ele viabiliza a execução de tarefas, tal como a rotulação semântica, alvo desta monografia.

Capítulo 3

Rotulação de Papéis Semânticos

O papel semântico em uma linguagem é a relação de uma unidade lexical da sentença com um determinado predicado. Os argumentos semânticos mais comuns incluem agente, paciente, instrumento e também os que indicam local, noção de tempo, modo, causa, entre outros aspectos. A rotulação de papéis semânticos (*SRL*) produz uma representação superficial do texto, que indica as propriedades básicas e as relações entre as entidades da sentença.

A principal tarefa da rotulação semântica é justamente esta: a de indicar exatamente quais as relações semânticas existentes entre um predicado e suas associações, tanto com os participantes quanto com as propriedades. Essa é uma tarefa chave e crucial para responder perguntas como: "Quem?", "O que?", "Quando?", "Onde?" e "Por que?". Tem-se então, que a análise semântica de um texto está diretamente ligada com o acontecimento de eventos, determinando "quem" fez "o que" para "quem", além de detectar "onde", "quando" e "como". O predicado de uma sentença (geralmente um verbo) estabelece o que acontece, enquanto os outros constituintes da sentença expressam as entidades participantes. A função primária da *SRL* é indicar exatamente quais são as relações semânticas existentes do predicado com seus participantes e propriedades associadas. Essas relações estão pré-especificadas em uma lista de possíveis papéis semânticos do predicado em questão.

As respostas dadas após a análise semântica podem ser úteis em diversas áreas, incluindo a extração de informações, repostas automáticas de perguntas, sumarização, retirada de ambiguidades e, em geral, problemas de linguagem natural que necessitam de alguma interpretação semântica. Um exemplo de rotulação semântica pode ser visto a seguir:

[AGENTE João] golpeou [PACIENTE Maria].

A rotulação semântica nos permite responder às seguintes perguntas:

- Quem golpeou? (Quem é o agente?)
- Quem foi golpeado? (Quem é o paciente?)

Um exemplo de retirada de ambiguidade, através da rotulação semântica:

1) João deixou a sala.
2) João deixou sua irmã jogar bola à noite.

Na primeira sentença, através da rotulação semântica, tem-se o verbo "deixar" no sentido de "partir". Vê-se que "João" é o elemento que está partindo; e "sala" é o local de onde João partiu. Já na segunda sentença, o verbo "deixar" está no sentido de "permitir". Tem-se, então, que "João" é o elemento que permite, "irmã" é o elemento beneficiado e "jogar bola à noite" é o elemento que representa a noção de "permissão".

Um dos grandes alicerces da rotulação semântica é a identificação do verbo. A partir daí, rotula-se os demais elementos da sentença. Os papéis semânticos típicos são rotulados como "agente", "paciente", "instrumento", entre outros. Cada caso deve ser analisado de maneira particular, visto que não há uma regra geral. Um potencial problema, por exemplo, é que nem todos os sujeitos de sentenças são agentes. Como exemplo, tem-se abaixo a palavra "porta" aparecendo em lugares distintos da sentença, mas mantendo seu valor semântico:

1) [AGENTE O estudante] abriu [PACIENTE a porta].
2) [PACIENTE A porta] foi aberta.

Na primeira sentença, "o estudante" é o sujeito e "a porta" faz papel de objeto direto. Em termos semânticos, diz-se que "a porta" é o elemento paciente. Já na segunda sentença, "a porta" passa a ser o sujeito, mas mantém seu valor semântico, ou seja, de elemento paciente.

3.1 Histórico Computacional

Por décadas, as pesquisas em unidades lexicais, gramáticas e demais artifícios semânticos (HIRST, 1987; PUSTEJOVSKY, 1995; COPESTAKE e FLICKINGER, 2000) evoluíram como suporte a uma profunda análise semântica, a partir de uma determinada linguagem de entrada. Na década de 1990, houve um crescimento notável no desenvolvimento dos métodos de aprendizado de máquina através do campo da linguística computacional, permitindo que os sistemas aprendessem conhecimento linguístico complexo em detrimento dos métodos manuais. Esses métodos desenvolvidos mostraram-se eficientes em adquirir o conhecimento necessário para interpretações semânticas, assim como para

extrair as propriedades dos predicados e suas relações. Como exemplos, podem ser citados o aprendizado de subcategorização de *frames* (BRISCOE e CARROLL, 2002) e a classificação de verbos de acordo com as estruturas dos argumentos (MERLO e STEVENSON, 2001).

Recentemente, *corpora* médios e grandes têm sido manualmente anotados com papéis semânticos na *FrameNet* (FILLMORE, RUPPENHOFER e BAKER, 2004), *PropBank*[7] e *NomBank*[8] permitindo o desenvolvimento de avanços, especialmente para a rotulação de papéis semânticos. Com a evolução, a *SRL* se tornou uma tarefa bem definida com uma sólida área de trabalho e avaliações comparativas. A identificação de *frames*[9] de eventos pode, potencialmente, beneficiar as aplicações de processamento de linguagens naturais, assim como a extração de informação[10], respostas de questionários[11], sumarização[12] e tradução de máquina[13]. Trabalhos relacionados de classificação dos relacionamentos semânticos em frases nominais também têm sido abordados para os processos de linguagem natural.

Embora o uso de sistemas de rotulação semântica nas aplicações do mundo real tem sido limitado, a promessa é de que se estenda esse tipo de análise para muitas aplicações que requeiram certo nível de interpretação semântica. A *SRL* representa um *framework* excelente para realizar pesquisas em técnicas computacionais, visando adquirir e explorar relações semânticas entre os diferentes componentes do texto. Logo no início, tarefas mais simples eram realizadas através da rotulação semântica. Como exemplo, podemos citar: respostas sobre horários de chegada dos vôos, indicação de direção, relatórios de caixa de bancos ou as demais transações mais simples. Mais recentemente, com o avanço dos

---

[7] PALMER, M.; GILDEA D.; KINGSBURY P. **The Proposition Bank: An annotated corpus of semantic roles**. Computational Linguistics. [s.l.]. 2005. p.71-105.

[8] MEYERS, A.; REEVES R.; MACLEOD, C.; SZEKELY, R.; ZIELINSKA, V.; YOUNG, B.; GRISHMAN, R. **The NomBank Project: An interim report**. In: *Proceedings of the HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*. Boston, MA, 2004. p24-31.

[9] Veja definição de *frame* no item 3.3.

[10] SURDEANU, M.; HARABAGIU, J.; WILLIAMS, J.; AARSETH, P. **Using predicate-argument structures for information extraction**. In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Sapporo, Japan, 2003. p.8-15.

[11] NARAYANAN, S.; HARABAGIU, S. **Question Answering based on semantic structures**. In: *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*. Geneva, Switzerland, 2003. p.693-701.

[12] MELLI, G.; WANG, Y.; LIU, Y.; KASHANI, M.; SHI, Z.; GU, B.; SARKAR, A.; POPOWICH, F. **Description of SQUASH, the SFU question answering summary handler for the DUC-2005 Summarization Task**. In: *Proceedings of the HLT/EMNLP Document Understanding Workshop (DUC)*. Vancouver, Canada, 2005.

[13] BOAS, H. **Bilingual framenet dictionaries for machine transalation**. In: *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC)*. Las Palmas de Gran Canaria, Espanha, 2002. p.1364-1371.

estudos e o aperfeiçoamento dos algoritmos, outras tarefas podem ser destacadas, tais como: o resumo de artigos, perguntas-respostas automáticas e reconhecimento de estruturas semânticas mais complexas em diálogos.

3.2 *Part-of-Speech Tagging*

A tarefa de *Part-of-Speech Tagging* (*POS Tagging* ou *POST*), também chamada de marcação gramatical ou desambiguação de categorias de palavras, é o processo de detectar e classificar cada palavra de uma determinada frase, de acordo com seu significado sintático.

Uma forma simplificada e didática do processo de *POST* é a de identificar as palavras como "substantivos", "verbos", "adjetivos", "advérbios", "conjunções", etc. Porém, uma marcação mais avançada pode ser feita, visando aprimorar essa técnica. Essa marcação indica que se deve atentar não somente para a definição das classes gramaticais, mas também para o contexto em que cada palavra está inserida. As grandes classes gramaticais existentes e didaticamente ensinadas são: substantivo, verbo, adjetivo, preposição, pronome, advérbio, conjunção e interjeição. Entretanto, existem muito mais categorias e subcategorias. A seguir, podemos ver alguns exemplos de classes gramaticais:

1) Maria saiu com João.
2) Jonas chegou em casa repentinamente.
3) José comprou um tênis macio.

No exemplo 1, tanto a palavra "Maria" quanto "João" são classificadas como substantivos. "Saiu" é uma conjugação do verbo "sair" e é identificado como o verbo da oração. No exemplo 2, "Jonas" é o substantivo, "chegou" é o verbo (conjugação do verbo "chegar") da oração, "em casa" é o adjunto adverbial de lugar e "repentinamente" é o advérbio de modo. Por último, no exemplo 3, "José" e "tênis" são substantivos, "comprou" é o verbo da oração (conjugação do verbo "comprar") e "macio" é o adjetivo, que caracteriza um substantivo (no caso, o "tênis").

É comum que a marcação gramatical seja feita manualmente. Porém, com o avançado da tecnologia e com o aprimoramento das técnicas utilizadas, essa marcação já pode ser feita por algoritmos computacionais linguísticos. A entrada de um algoritmo de *POST* é um texto com palavras em linguagem natural e uma série de possíveis classes

gramaticais. A saída do algoritmo é a possível classe gramatical para cada unidade do texto de entrada.

Diferentemente da linguagem computacional, as palavras em uma linguagem natural nem sempre têm um sentido definido. Muito pelo contrário, elas são extremamente dependentes do contexto e do sentido com que aparecem nos textos. O grande desafio da técnica de *POST* é exatamente este: o de que identificar as palavras que podem ser vítimas da ambiguidade e classificar, corretamente, sua classe gramatical. A seguir, podemos ver casos com a presença da ambiguidade, que podem dificultar a técnica de *POST*.

4) O andar daquela senhora é elegante.
5) Amanhã, vou andar até cansar.
6) Esse suco está estranho.
7) Um estranho me abordou ontem à noite.
8) Planta precisa de água e luz.
9) O homem planta a árvore.

No exemplo 4, a palavra "andar" está presente no texto como substantivo e é caracterizado pelo adjetivo "elegante". Em contrapartida, no exemplo 5, a palavra "andar" é o verbo e faz parte da locução verbal "vou andar". No exemplo 6, a palavra "estranho" faz papel de adjetivo da frase, caracterizando o substantivo "suco". Já no exemplo 7, "estranho" é um substantivo da frase, responsável pela ação representada pelo verbo "abordar" (no caso, a conjugação "abordou"). No exemplo 8, a palavra "planta" é classificada como substantivo, enquanto no exemplo 9, "planta" é o verbo da sentença.

3.3 *FrameNet*

A *FrameNet* é um projeto de lexicografia computacional (ATKINS, FILLMORE e JOHNSON, 2003; BAKER, FILLMORE e LOWE, 1998) que, através de processos de anotação automática e manual de sentenças, extrai informação das propriedades semântico-sintáticas de unidades lexicais de extensas base de dados eletrônicas. A grande meta do projeto *FrameNet* é produzir descrições das palavras baseadas nos *frames* semânticos. Segundo FILLMORE (1982), *frame* é qualquer sistema de conceitos relacionados de forma que para entender qualquer conceito é necessário entender toda a estrutura em que ele se encontra. Assim, ele pode ser entendido como uma estrutura conceitual que estabelece determinados relacionamentos com outras estruturas do mesmo tipo. O resultado disso é a

formação de uma rede de estruturas conceituais, a *FrameNet*. Ainda segundo FILLMORE (1982), os *frames* semânticos são representações esquemáticas de situações envolvendo vários participantes, propriedades e funções, nas quais uma palavra deve ser tipicamente usada.

A metodologia da construção da *FrameNet* é relativamente simples. O procedimento segue, basicamente, quatro passos. Primeiramente, define-se, descobre-se e descreve-se os *frames* que farão parte da base de dados. A seguir, são instituídos os participantes de cada *frame*, ou seja, os papéis semânticos que estão diretamente ligados ao *frame* em questão. Esses participantes recebem o nome de *frame elements* (*FE*). Prosseguindo na construção da *FrameNet*, lista-se as unidades lexicais que invocam o *frame*. Geralmente, essas unidades são os verbos e remetem subitamente ao *frame* relacionado. Por último, são encontrados e anotados exemplos de sentenças no *British National Corpus* (*BNC*).

As anotações disponíveis da *FrameNet* permitem que seja gerado um procedimento de rotulação que extraia as informações necessárias de qualquer domínio. Essas anotações são construídas de formar a mostrar a realização de cada elemento do *frame*, do tipo de sintagma e da função gramatical deste sintagma (RUPPENHOFER *et al.*, 2006). Segundo RUPPENHOFER *et al.* (2006), o maior produto desse projeto é o banco de dados léxico produzido que atualmente possui mais de 10.000 unidades lexicais, dentre as quais mais de 6.100 estão completamente anotadas, contidas em aproximadamente 825 frames e exemplificadas em mais de 135.000 sentenças anotadas.

Os *frames* são definidos em termos dos participantes que os integram (elementos do *frame* ou *frame elements*). Por exemplo, o *frame* Chegada é composto por quatro elementos: um TEMA, uma ORIGEM, um CAMINHO e um DESTINO, de forma que o TEMA se move de uma ORIGEM ao longo de um CAMINHO até um determinado DESTINO. A *FrameNet* então, busca pelos usos linguísticos de unidades lexicais como chegar, vir, retornar, entrar, visitar que realizam este *frame* e descreve as dependências sintáticas e semânticas destes itens.

Na grande maioria das vezes, a unidade lexical que evoca o *frame* é um verbo, mas há casos em que ela também pode ser um substantivo ou um adjetivo. A figura 3.1 mostra três exemplos dos diferentes tipos de unidades lexicais – fritou (verbo), retorno (substantivo) e entediados (adjetivo).

[$_{\text{cozinheiro}}$ Maria Fernanda] fritou [$_{\text{comida}}$ batatas]

O retorno [$_{\text{Tema}}$ dela] custou caro para a empresa

[$_{\text{entediado}}$ Eles] ficaram entediados [$_{\text{duração}}$ por horas]

Figura 3.1 – Exemplos de unidades lexicais

3.3.1 Relações Semânticas na *FrameNet*

Os *frames*, os elementos do *frame* (*frame elements*, papéis semânticos) e as unidades lexicais associadas a eles, estão situados num espaço semântico ligados por meio de relações *frame*-a-*frame* (RUPPENHOFER *et al*.,2006 *apud* FERNANDES, 2008). Ainda segundo RUPPENHOFER, essas relações aumentam a compreensão dos *frames*, uma vez que um *frame* mais complexo pode ser melhor interpretado se analisado através da relação com outros *frames* mais simples. Além disso, traz robustez à base de dados, já que embora os pesquisadores possam fazer uma divisão diferente dos *frames*, eles podem ser associados através de suas relações semânticas. Cada relação na *FrameNet* é uma relação entre dois *frames*, onde um *frame* (mais abstrato/menos dependente) pode ser chamado de *super_frame* e outro de *sub_frame* (menos abstrato/mais dependente).

As relações semânticas entre os *frames* estabelecidas pela *FrameNet* podem ser divididas em **herança**, **uso**, ***subframe*** e **perspectiva**.

A **herança** é a relação mais forte entre os *frames*. Ela remete à ideia de que um *frame* é a extensão de outro, acrescido de características específicas. Para essa relação, todo fato que é estritamente real para o *frame-pai* deve ser igualmente verdadeiro ou mais específico para o *frame-filho* (RUPPENHOFER *et al*.,2006 *apud* FERNANDES, 2008). Um exemplo seria o *frame "*partida", que herda elementos do *frame "*transporte". A figura 3.2 a seguir mostra outro exemplo da relação de **herança**.

Figura 3.2: Exemplo da relação de herança, onde Assistência herda de Ato_intencional

A relação de **uso** indica uma situação mais abstrata. Essa relação é usada nos casos em que a cena evocada pelo *frame-filho* referencia ao *frame-pai*. Nessa relação, é possível que um *frame* use um ou mais *frames*. Um exemplo seria o *frame* "velocidade", que usa (pressupõe) o *frame* "movimento". Outro exemplo pode ser visto na figura 3.3, onde o *frame* "Comunicação_de_julgamento" usa os *frames* "Julgamento" e "Sentença".

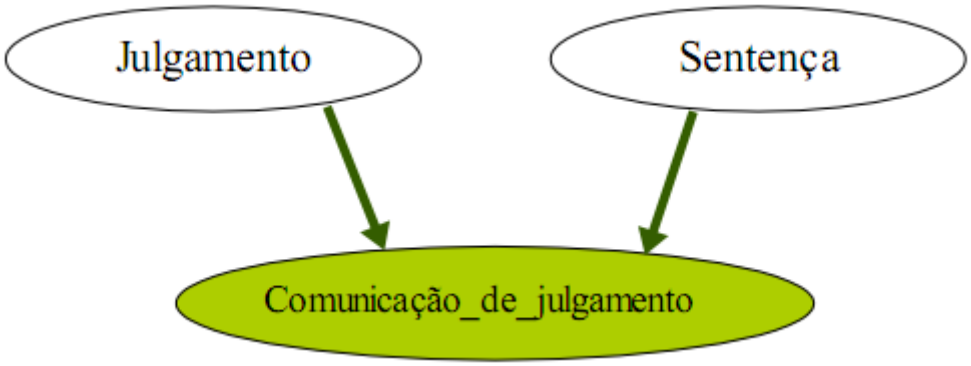

Figura 3.3: Exemplo da relação de uso, onde Comunicação_de_julgamento usa os *frames* Julgamento e Sentença

A relação de **subframe** ocorre quando uma sequência de estados e transições que compõem um *frame* complexo pode ser dividida em *frames*. Os *frames* separados são chamados de *subframes* (RUPPENHOFER *et al*.,2006 *apud* FERNANDES, 2008). Um exemplo seria o *frame* "Processo_criminal" onde para cada etapa do processo existe um *frame* correspondente, como mostrado na figura 3.4.

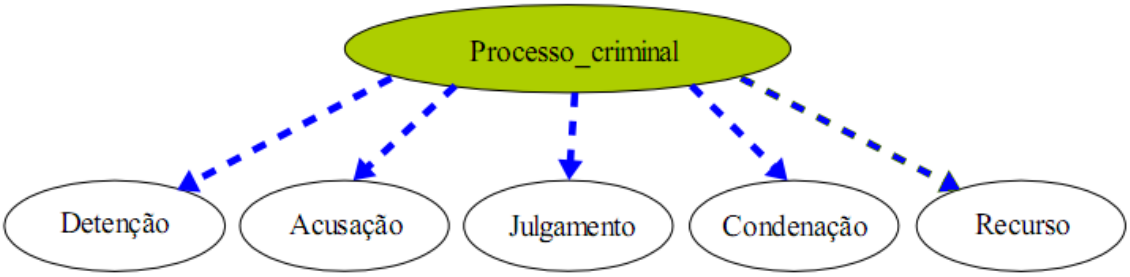

Figura 3.4: Exemplo da relação de *subframes*, que mostra as etapas do processo_criminal

Finalmente, a última relação é a de **perspectiva** e ocorre quando é identificada a presença de pelo menos dois pontos de vista a partir de um *frame* neutro (RUPPENHOFER *et al*.,2006 *apud* FERNANDES, 2008). Um exemplo pode ser visto na figua 3.5, que mostra a perspectiva do "Comprador" e do "Vendedor", ligadas ao *frame* "Comércio_transferência-de-mercadorias".

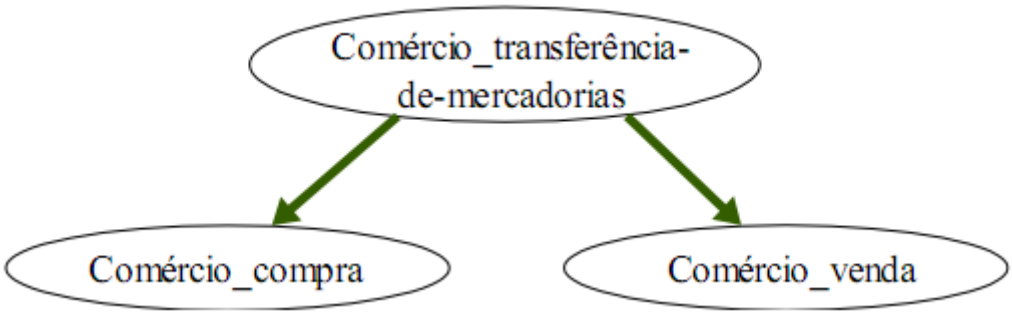

Figura 3.5: Exemplo da relação de perspectiva

Uma vez feito todo o estudo teórico que envolve a descoberta de conhecimento em banco de dados (*KDD*), a aplicação do processo na linguística computacional, a rotulação de papéis semânticos e a base de dados da *FrameNet*, um experimento computacional foi simulado, buscando comprovar a possibilidade de se descobrir padrões na base de dados, aplicar algoritmos de classificação de aprendizado de máquina e executar tarefas, como a rotulação de papéis semânticos em sentenças.

Para auxiliar nesta tarefa de rotulação, muitas ferramentas foram usadas, cada qual realizando seu papel durante a análise e processamento das sentenças. No próximo capítulo, as ferramentas estão descritas detalhadamente, além do classificador *Naïve Bayes* – um algoritmo de aprendizado de máquina.

Capítulo 4

Uso da Ferramenta Computacional

A ferramenta utilizada foi o ***Shalmaneser***[14] (ERK e PADO, 2007), responsável pela rotulação semântica superficial, ou seja, a atribuição de classes e papéis semânticos para cada membro de uma determinada sentença. O ***Shalmaneser*** é uma ferramenta modularizada, ou seja, é composta por módulos independentes, que se comunicam através de um arquivo de formato XML. A saída da ferramenta pode ser interpretada graficamente. A ferramenta foi desenvolvida de modo a ser facilmente adaptada a novas linguagens, integração de *parsers* sintáticos adicionais e novos sistemas de aprendizado de máquina.

Os componentes da arquitetura do ***Shalmaneser*** são: (a) *FRPREP*, um módulo de pré-processamento para analisar o texto puro de entrada, responsável pela lematização, *Part-of-Speech Tagging* e *parsing*, (b) *FRED* (*Frame Disambiguation*), um componente que cuida da desambiguação e (c) *ROSY* (*Role Assignment System*), responsável pela identificação e rotulação dos papéis semânticos no contexto linguístico de cada predicado que introduz um *frame*.

4.1 Arquitetura e Componentes

A arquitetura de processamento do ***Shalmaneser*** pode ser vista na figura 4.1.

O *software* inclui um componente de pré-processamento chamado *FRPREP*, que combina programas externos – responsáveis pelo *parsing*, lematização e a tarefa de *POS Tagging* – a fim de produzir o arquivo *XML* de saída. Visando manter a flexibilidade e a extensibilidade, o *FRPREP* interage com todos os componentes de análise sintática através de uma interface abstrata comum.

[14] Ferramenta livre escrita em Ruby (linguagem de *scripts* orientada a objetos). Download disponível a partir do website http://www.coli.uni-saarland.de/projects/salsa/page.php?id=software

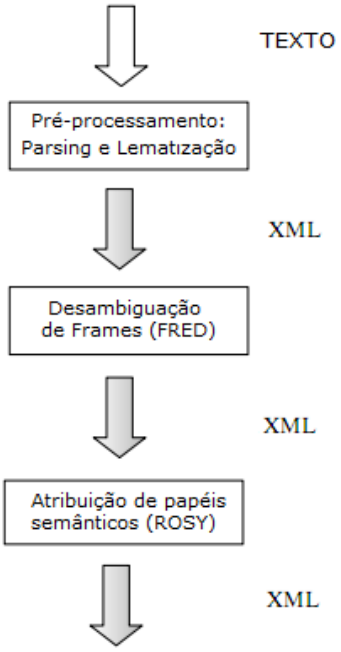

Figura 4.1 – Visão Geral da Arquitetura (adaptada de ERK e PADO, 2007)

O componente *FRED* (*FRamE Disambiguation)* é um sistema para desambiguação do sentido das palavras. Ele analisa cada palavra através de um conjunto de inúmeras palavras e sentenças, de acordo com sua função gramatical. Quando verbos, ele analisa seus alvos e sua voz (ativa, passiva, reflexiva). O *FRED* faz uso de um classificador – o *Naïve Bayes*[15] – que estima probabilidades através das probabilidades já conhecidas. Como o componente de aprendizado de máquina é todo encapsulado, existe ainda facilidade de alternar o algoritmo classificador.

O componente *ROSY* (*ROle assignment SYstem)* é responsável por atribuir papéis semânticos para o contexto linguístico do predicado, baseado nas classes semânticas já atribuídas para ele. *ROSY* atribui papéis semânticos para os constituintes da estrutura sintática. De todos esses constituintes, apenas alguns podem realmente ser rotulados com um papel semântico. Então, *ROSY* faz uso de um *parser* baseado em uma estrutura de árvore para descobrir, de fato, quais constituintes podem suportar um papel semântico.

4.2 O uso de Ferramentas de Suporte

O *software* ***Shalmaneser*** possui componentes próprios, como foi descrito na seção anterior. São eles: *FRPREP*, *FRED* e *ROSY*. Basicamente, o componente *FRPREP* faz a

[15] Veja informações do classificador *Naïve Bayes* no item 4.2.4

combinação dos *softwares* externos utilizados pelo ***Shalmaneser*** e é responsável pelo pré-processamento do texto de entrada. Esses programas externos atuam nas tarefas de lematização, *parsing* e *Part-of-Speech*. O resultado final, após o processamento do texto por todos os programas externo, é a geração de um arquivo em formato *XML*. Como *parser*, está sendo usado o *Collins*, um classificador feito para a base de dados FrameNet 1.3. Esse classificador faz uso do algoritmo de aprendizado de máquina *Naive Bayes* e realiza a rotulação sintática. O programa externo *TreeTagger* é usado na tarefa de lematização e *Part-of-Speech*, ou seja, ele processa o texto de entrada de forma a deixá-lo pronto para que as análises sintáticas e semânticas sejam feitas. O programa externo *Mallet* (versão 0.4) é um sistema de aprendizado de máquina que serve de suporte para o componente *ROSY* (do ***Shalmaneser***) e é responsável por auxiliar na rotulação semântica das sentenças. Por último, o programa *Salto* – escrito em Java – é um programa de interface gráfica para anotação de papéis semânticos. Ele tem como entrada exatamente o arquivo *XML* gerado após o processamento e como saída um gráfico que demonstra o resultado das rotulações sintáticas e semânticas de cada sentença.

4.2.1 *Mallet*

O *software Mallet* é um pacote baseado na linguagem Java para processamento estatístico de linguagem natural, classificação de documentos, clusterização, modelagem de tópicos, extração de informações, dentre outras aplicações de aprendizado de máquina para textos e sentenças. O programa inclui poderosas ferramentas de conversão de textos, além de fazer uso de diversos algoritmos (*Naïve Bayes*, algoritmo da máxima entropia e árvores de decisão, entre outros). Além disso, existe uma rotina de transformação dos documentos de texto em representações numéricas que são processadas eficientemente.

4.2.2 *TreeTagger*

O *software TreeTagger* é responsável pelo processamento de textos em linguagem natural, executando as tarefas de lematização e *Part-of-Speech Tagging*. A ferramenta estima probabilidades de transição através de uma árvore de decisão binária. A probabilidade de um dado problema é determinada seguindo a trajetória correspondente na árvore, até que se atinja uma folha da estrutura da árvore. Atualmente, o *TreeTagger* é usado com sucesso para as seguintes línguas: inglês, alemão, francês, italiano, holandês, espanhol, búlgaro, russo, grego, português e chinês. A tabela seguinte demonstra o processamento que o

TreeTagger faz sobre uma determinada sentença. No caso, a sentença é "The *TreeTagger* is easy to use".

Tabela 4.1: Processamento das palavras pelo *TreeTagger* (adaptada de SCHMID, 1996)

Palavra	**Part-of-Speech**	**Lematização**
The	DT (determinante)	the
TreeTagger	NP (proper noun, nome próprio)	TreeTagger
is	VBZ (verbo "ser", 3ª pessoa singular)	be
easy	JJ (adjetivo)	easy
to	TO (verb "to" – verbo "para")	to
use	VB (verbo)	use

4.2.3 *Salto*

O *Salto* é uma ferramenta escrita em Java com o intuito de exibir uma anotação de uma segunda camada estrutural, sobre uma camada de estrutura sintática já existente. Ou seja, ele representa a rotulação semântica, interligando-a com a rotulação sintática. A entrada do programa é o arquivo *XML* gerado após o processamento feito pelos componentes *FRED*, *ROSY* e pelos *softwares TreeTagger* e *Mallet*. Então, resumidamente, o programa exibe as classes semânticas (*frames* semânticos), os papéis semânticos (de acordo com cada *frame* semântico identificado) e as classes gramaticais a que cada termo da sentença se refere. Através desse *XML* gerado, o *SALTO* oferece uma interface gráfica amigável, intuitiva e customizável, como segue na figura 4.2.

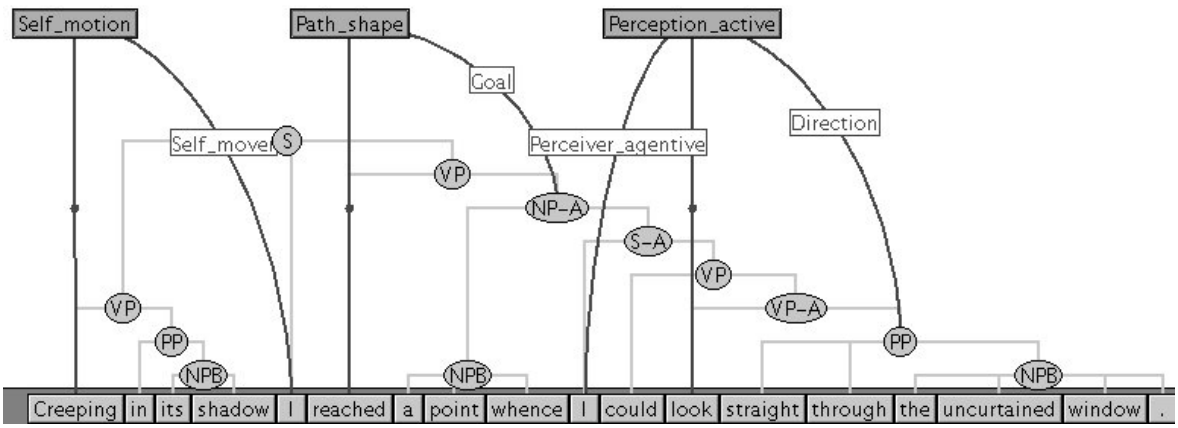

Figura 4.2 – Gráfico gerado após processamento do *Shalmaneser* (ERK e PADÓ, 1997)

4.2.4 *Collins Parser* e o Classificador *Naïve Bayes*

A análise sintática das sentenças é feita pelo *Collins Parser*[16]. Esse *parser* pode ser entendido como uma sequencia de decisões baseadas em fatos, abstraindo-se para uma complexa árvore. Dois importantes quesitos para que o *parser* dê um bom resultado é um alto poder de escolha e boa compactação. Em conjunto com o *Collins Parser*, um classificador para a base de dados da *FrameNet 1.3* foi desenvolvido, baseado no algoritmo classificador *Naïve Bayes*, conforme consta no Anexo A.

O classificador *Naïve Bayes*[17] é provavelmente um dos classificadores mais utilizados em Aprendizado de Máquina. Ele é denominado ingênuo (do inglês *naive*) por assumir que os atributos são condicionalmente independentes, ou seja, o acontecimento de um evento não é informativo e dependente sobre nenhum outro. Na maioria das tarefas, essa suposição de que existe independência entre as palavras de um texto é falsa. Porém, já foi demonstrado teoricamente que essa suposição, mesmo que falsa, não prejudica a eficiência do classificador na maioria dos casos. Apesar dessa visão ingênua, *Naïve Bayes* reporta o melhor desempenho em várias tarefas de classificação. Outra vantagem é o fato de que o classificador requer poucos dados de treinamento para estimar os parâmetros necessários para a tarefa de classificação, uma vez que as variáveis são consideradas independentes. Dessa maneira, só as variações das variáveis de cada classe precisam ser determinadas. Em comparação com os outros algoritmos de *DM*, o classificador *Naïve Bayes* é computacionalmente menos robusto, possibilitando que modelos de mineração sejam gerados rapidamente, viabilizando a descoberta das relações entre os atributos e classes.

A ideia por detrás desse classificador é pensar no inverso de um problema – "*backward probability"*. Como exemplo, pode-se citar uma urna com bolas brancas e pretas. Nesse cenário, a questão é qual a probabilidade de sortear uma bola branca, uma bola preta ou alguma combinação entre elas. O classificador *Naïve Bayes* interpreta exatamente ao contrário, ou seja, dado que uma ou mais bolas foram sorteadas, o que pode ser dito sobre o número de bolas brancas e pretas na urna? Analogamente para a linguística computacional, a questão passa a ser: dado que se tenham exemplos de texto de cada classe, o que se pode inferir sobre o processo gerador destes textos?

---

[16] Desenvolvido em 1999, pelo professor Michael Collins (MIT – Massachussets Institute of Technology)
[17] O classificador recebeu esse nome em homenagem ao ministro inglês Thomas Bayes, que foi o primeiro a formalizar uma teoria responsável pela sua origem

O classificador *Naïve Bayes* se baseia na aplicação do Teorema de *Bayes* para o cálculo das probabilidades necessárias para a classificação. O Teorema de *Bayes* é mostrado abaixo já no contexto de Aprendizado de Máquina, isto é, dada uma instância $A=a_1,a_2,...,a_n$, deseja-se predizer sua classe:

$$P(classe|\ A) = \frac{P(A\,|\,classe)\times P(classe)}{P(A)}$$

Como $A=a_1,a_2,...,\ a_n$, tem-se:

$$P(classe|\ a_1...a_n) = \frac{P(a_1...a_n\,|\,classe)\times P(classe)}{P(a_1...a_n)}$$

Para calcular a classe mais provável da nova instância, calcula-se a probabilidade de todas as possíveis classes e, no fim, escolhe-se a classe com a maior probabilidade como rótulo da nova instância. Em termos estatísticos, isso é equivalente a maximizar a $P(classe|a_1...a_n)$. Para tanto, deve-se maximizar o valor do numerador $P(a_1...a_n|classe)\times P(classe)$ e minimizar o valor do denominador $P(a_1...a_n)$. Como o denominador $P(a_1...a_n)$ é uma constante, pois não depende da variável classe que se está procurando, pode-se anulá-lo no *Teorema de Bayes*, resultando na fórmula abaixo, na qual se procura a classe que maximize o valor do termo $P(classe|\ a_1...a_n) = P(a_1...a_n|classe)\times P(classe)$:

$$\arg\max P(classe\,|\,a_1...a_n) = \arg\max P(a_1...a_n\ |classe)\times P(classe)$$

A suposição "ingênua" que o classificador *Naïve Bayes* faz é que todos os atributos $a_1...a_n$ da instância que se quer classificar são independentes. Sendo assim, o complexo cálculo do valor do termo $P(a_1...a_n|classe)$ reduz-se ao simples cálculo $P(a_1|classe)\times... \times P(a_n|classe)$. Assim, a fórmula final utilizada pelo classificador é:

$$\arg\max P(classe\,|\,a_1...a_n) = \arg\max \prod_i P(a_i\ |\ classe)\times P(classe)$$

Sabe-se, entretanto, que, na maioria dos casos, a suposição de independência dos atributos de uma instância é falsa. Mesmo assim, o classificador *Naïve Bayes* produz resultados bastante satisfatórios. Quando os atributos são realmente independentes, o classificador fornece a solução ótima.

Como dito anteriormente, o cálculo da classe de uma nova instância consiste no cálculo da probabilidade de todas as possíveis classes, escolhendo-se, a seguir, a classe

com maior probabilidade. De acordo com a fórmula acima, devem-se calcular os termos $P(a_i|classe)$ e P(classe). P(classe) é simplesmente o número de casos pertencente à classe em questão sobre o número total de casos. $P(a_i|classe)$, por sua vez, é o número de casos pertencente à classe em questão com o atributo *i* com valor a_i sobre o número total de casos.

Após apresentadas e descritas todas as ferramentas que auxiliaram na execução do experimento computacional, deu-se início ao estudo de caso. Primeiramente, as ferramentas foram instaladas e configuradas. Logo em seguida, foi feito o experimento de fato, executando-se a tarefa da rotulação dos papéis semânticos. Tudo isto está explicitado detalhadamente no capítulo seguinte.

Capítulo 5
Experimento Computacional

Foi realizado um experimento computacional com o intuito de verificar a rotulação automática de sentenças. A base de dados (*corpus*) adotada foi a *FrameNet* 1.3. Vários programas foram utilizados como suporte para que o experimento atingisse os objetivos propostos. A ferramenta principal é o *Shalmaneser* (*freeware*) que, integrado com os demais *softwares*, faz a rotulação sintática e semântica a partir da entrada de sentenças em texto puro. Como a base de dados utilizada foi a FrameNet, o experimento foi baseado em sentenças da língua inglesa.

A grande motivação em utilizar a ferramenta *Shalmaneser* foi a inexistência, até então, de uma ferramenta de rotulação que fosse robusta e, ao mesmo tempo, *freeware* e *open-source*. Essa foi, inclusive, a motivação dos desenvolvedores da ferramenta. Os dados de entrada passam por várias etapas de processamento e dão origem, ao final de todo o processo, a um arquivo com extensão *XML*. O resultado da rotulação pode ser visto e analisado graficamente, através de um programa suporte, o *Salto*.

5.1 Motivação

A motivação para se fazer o experimento computacional veio do fato de que a rotulação semântica é considerada uma tarefa recente e tem sido explorada e desvendada gradativamente. Nas últimas décadas, muito foi desenvolvido no que diz respeito à rotulação sintática e à tarefa de *Part-of-Speech*. A importância da rotulação semântica tem sido descoberta ao passar do tempo e muitas bases de dados, algoritmos e classificadores que auxiliam na tarefa de rotular têm sido criados e aprimorados. Além disso, o fato da inexistência, até então, de um programa *freeware* e robusto que cumprisse bem a tarefa de rotulação sintático-semântica, serviu de motivação para o uso (e teste) do programa *Shalmaneser*.

5.2 Preparação

Para dar início ao experimento computacional, foi necessária a instalação do sistema operacional *Linux*, mais especificamente a distribuição *Ubuntu 9.04*. Essa necessidade deve-se ao fato de que o *Collins Parser* (um dos programas que auxiliam o *Shalmaneser*)

não roda sob a plataforma *Windows*. Após a instalação, foram baixados vários pacotes necessários ao bom funcionamento das ferramentas necessárias. O banco de dados usado foi o *MySQL Server 5.0*. Junto com ele, foram baixados os pacotes do *PHP 5.0* e o *phpMyAdmin*, para gerenciar de maneira mais intuitiva (via *browser*) o banco de dados criado. Foi necessário também, a instalação do *Ruby* (linguagem de *script* orientada a objetos, na versão 1.8), uma vez que o programa *Shalmaneser* é escrito nessa linguagem. Para conectar o *Ruby 1.8* ao banco de dados *MySQL*, foi obtido o pacote *libmysql-ruby1.8*. Vale ressaltar também que os pacotes do *apache*, *gcc* e *make* devem estar instalados na máquina.

Com esse cenário já pronto, o próximo passo é a configuração dos *softwares* que vão auxiliar o *Shalmaneser* na tarefa da rotulação. É aconselhável que cada programa seja extraído para uma pasta particular e fiquem dentro do mesmo diretório principal, visando facilitar a configuração do *Shalmaneser*.

Após as extrações, deve-se configurar o *Shalmaneser*. Em seu diretório, pelo terminal (console), deve-se executá-lo, através do comando *ruby setup.rb*. Como já falado anteriormente, esse comando executa o aplicativo, escrito em ruby (extensão *.rb*). O comando de chamada do *setup* pode ser visto a seguir, na figura 5.1.

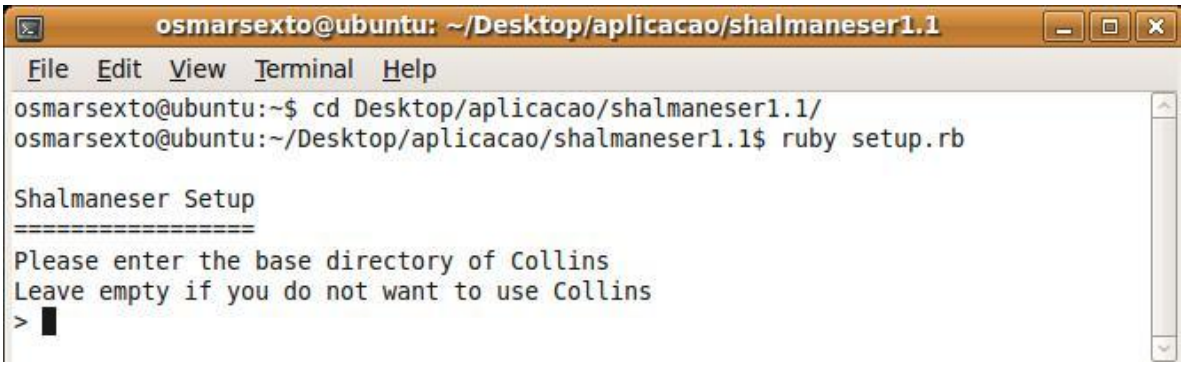

Figura 5.1: Comando para chamada do *setup* do *Shalmaneser*

Assim que o *setup* iniciar, deve-se indicar o caminho (endereço) de cada uma das pastas dos programas externos, que serão usados pelo *Shalmaneser* e realizarão a tarefa cooperativamente, cada um exercendo sua função dentro do processo de rotulação. Será necessário indicar o caminho dos programas *Collins (parser), TreeTagger* e *Mallet*. Caso o usuário não deseje utilizar um dos programas, basta deixar a linha em branco e seguir em frente. Vale lembrar que alguns programas (como o *Sleepy* e o *Minipar*) não precisam ser usados, uma vez que ou executam tarefas já realizadas por outros ou dizem respeito ao

processamento do texto em outra língua (alemão, no caso do *Sleepy*). Essa configuração dos programas de suporte ao *Shalmaneser* pode ser vista a seguir, na figura 5.2.

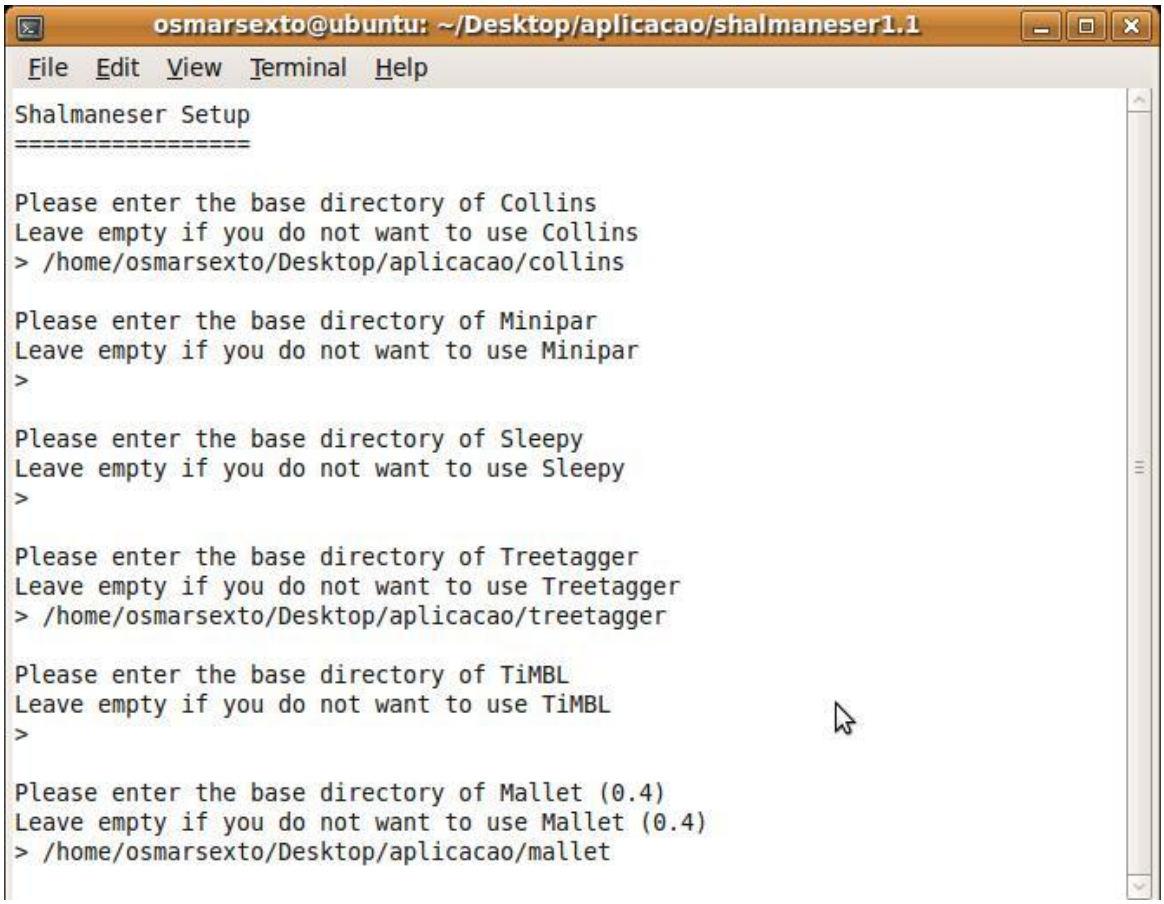

Figura 5.2: Configuração dos programas auxiliares ao *Shalmaneser*

A próxima etapa, ainda no *setup* do *Shalmaneser* é indicar o banco de dados (que já deve ter sido criado pelo *phpMyAdmin*). Basta que o usuário indique o *host*, *username*, *password* e o *database name*. A tela da configuração do banco de dados pode ser vista a seguir, na figura 5.3.

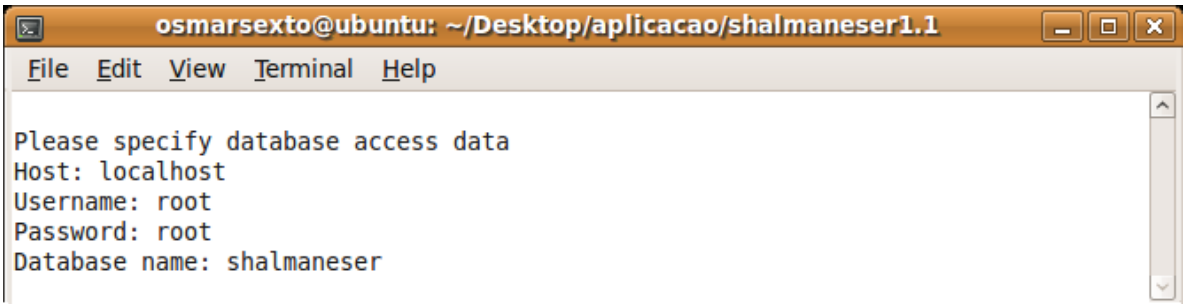

Figura 5.3: Configuração do banco de dados

Prosseguindo na configuração, aparecerá um alerta questionando se o usuário deseja instalar um *patch* de atualização do *Mallet*. Isso se deve ao fato do programa apresentar um comportamento atípico, considerado um *bug* pelos próprios desenvolvedores do *Shalmaneser*. O *bug* acontece porque o *Mallet* não consegue lidar com os rótulos (*labels*) das classes nos dados de testes ainda não vistos nos dados de treinamento. Portanto, é extremamente aconselhável que esse *patch* seja instalado para o correto funcionamento de todos os componentes do *Shalmaneser*. A tela que mostra o *warning* e confirmação da instalação do *patch* pode ser vista a seguir, na figura 5.4.

Figura 5.4: Instalação do *patch* de correção do *Mallet*

Após a instalação do *patch* de correção do *Mallet*, a configuração de todos os programas estará completa.

5.3 Experimento Computacional

Já com todas as ferramentas configuradas, dá-se início ao experimento computacional. Além da rotulação sintática, esse experimento também é responsável por executar a tarefa da rotulação dos papéis semânticos. Neste momento, todos os programas agem cooperativamente, cada qual sobre uma tarefa específica. Os componentes interagem entre si através de um arquivo comum, em formato *XML*. Ou seja, um componente executa sua tarefa e gera um arquivo *XML* que será interpretado pelo próximo componente. Dessa maneira, chega-se a um arquivo *XML* final, após o processamento de todos os *softwares*.

A rotulação sintático-semântica é feita sobre a base de dados da *FrameNet* 1.3, uma base linguística da língua inglesa. Portanto, nessas condições, o arquivo de entrada para o *Shalmaneser* deverá conter sentenças em língua inglesa. Então, basta criar um arquivo texto (*text file*, *TXT*) com as frases que se deseja rotular. É importante certificar que nesse arquivo texto, cada frase ocupe uma linha. Tal procedimento é necessário para que o *Collins parser* efetue corretamente seu processamento.

Inicialmente, duas pastas devem ser criadas. A pasta que receberá o nome de *data-for-analysis* conterá o arquivo *txt* criado, com suas respectivas sentenças que serão

processadas e rotuladas. A outra pasta receberá o nome de *analysed-data* e será o destino do arquivo *XML* gerado após o processamento do arquivo *txt* que está na pasta *data-for-analysis*. É aconselhável que ambas as pastas estejam no mesmo diretório, como mostra a figura 5.5 a seguir.

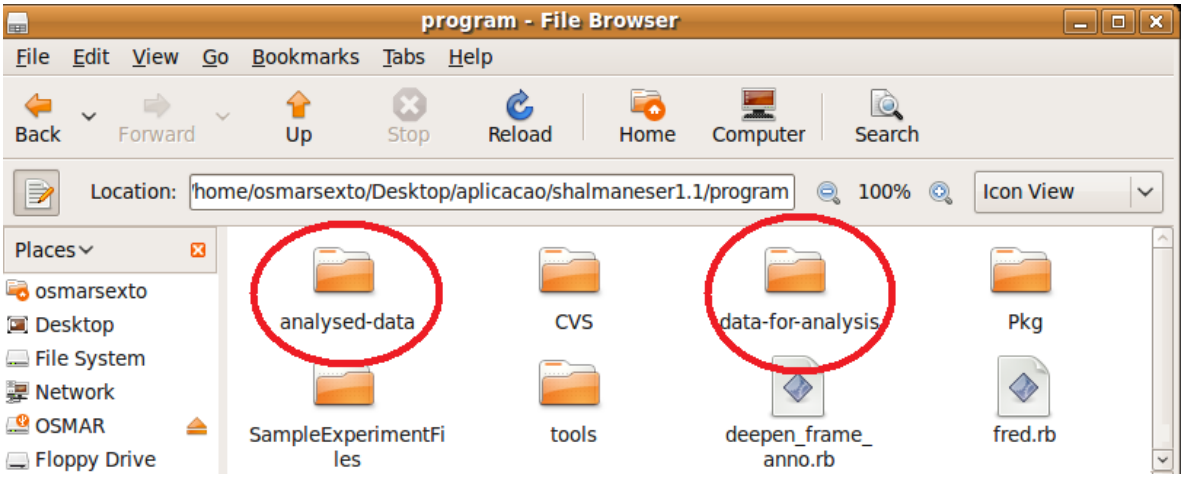

Figura 5.5: Criação das pastas *analysed-data* (destino) e *data-for-analysis (*origem)

A próxima etapa é executar o *Shalmaneser*, de fato. Para tal, basta um comando através do terminal. Nesse comando, os parâmetros desejáveis para o processamento devem ser passados. Existem cinco tipos de parâmetros:

- ***-i***: do inglês *input*, o parâmetro *-i* indica o caminho da pasta de origem, ou seja, deve-se passar o endereço da pasta *data-for-analysis* (de acordo com o estudo de caso em questão), que possui o arquivo *txt* que deverá ser processado.
- ***-o***: do inglês *output*, o parâmetro *-o* indica o caminho da pasta de destino, ou seja, deve-se passar o endereço da pasta *analysed-data* (de acordo com o estudo de caso em questão). Essa pasta conterá o arquivo *XML* gerado após o processamento do arquivo *txt*.
- ***-e***: do inglês *encoding*, o parâmetro *-e* indica o formato de codificação do arquivo. As possibilidades são "ISO", "UTF-8" e "HEX". Por padrão, é adotado o formato "ISO".
- ***-l***: do inglês *language*, o parâmetro *-l* indica o idioma dos arquivos de entrada. As possibilidades são "EN" (inglês) e "DE" (alemão). Por padrão, o idioma alemão é adotado.

- ***-p***: do inglês *parser*, o padrâmetro *-p* indica qual será o *parser* (e seu respectivo classificador) adotado no processamento. As possibilidades são “collins”, “sleepy” e “minipar”. Por padrão, é adotado o *parser* “sleepy”.

Uma vez que se tenha decidido os parâmetros que serão adotados, o *Shalmaneser* está pronto para ser executado e iniciar o processamento do texto e as tarefas de rotulação sintática e semântica. Escolhendo-se as pastas de entrada (origem) e saída (destino), o idioma inglês e o *parser* “collins” e mantendo-se a codificação padrão (UTF-8), a chamada para execução do *Shalmaneser* seria como a representada na figura 5.6:

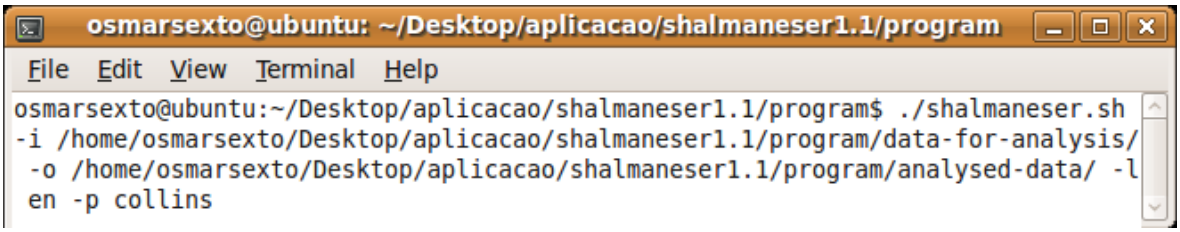

Figura 5.6: Procedimento de chamada e execução do *Shalmaneser*

A partir deste momento, tanto o processamento quanto a rotulação serão feitos automaticamente. Primeiramente, o componente *FRPREP* do *Shalmaneser* assume o controle e realiza suas tarefas. Ele lê o arquivo de entrada e faz sua transformação para a codificação *UTF-8*, caso esta seja “*ISO*” ou “*HEX*”. Esperando uma sentença por linha, ele transforma o texto plano em uma espécie de tabela formatada. Em seguida, ele transforma esse texto tabulado em um arquivo *XML* que será o arquivo de interseção entre as etapas, ou seja, um componente processa o arquivo *XML* e o transmite para o próximo componente realizar sua tarefa. Caso haja mais de um arquivo na pasta *data-for-analysis*, ele lê cada um separadamente e armazena todos em um mesmo arquivo final. Agora, junto com o programa *TreeTagger*, o *FRPREP* coordena as tarefas de lematização e dá início – junto com o *Collins Parser* – à tarefa de *parsing*. Juntos, eles realizam a rotulação sintática, a partir do número de sentenças presentes no arquivo *txt* analisado. Todo esse processo conjunto do componente *FRPREP* e dos programas *TreeTagger* e *Collins Parser* está descrito a seguir, através da figura 5.7. Os processos e tarefas mais importantes estão destacados.

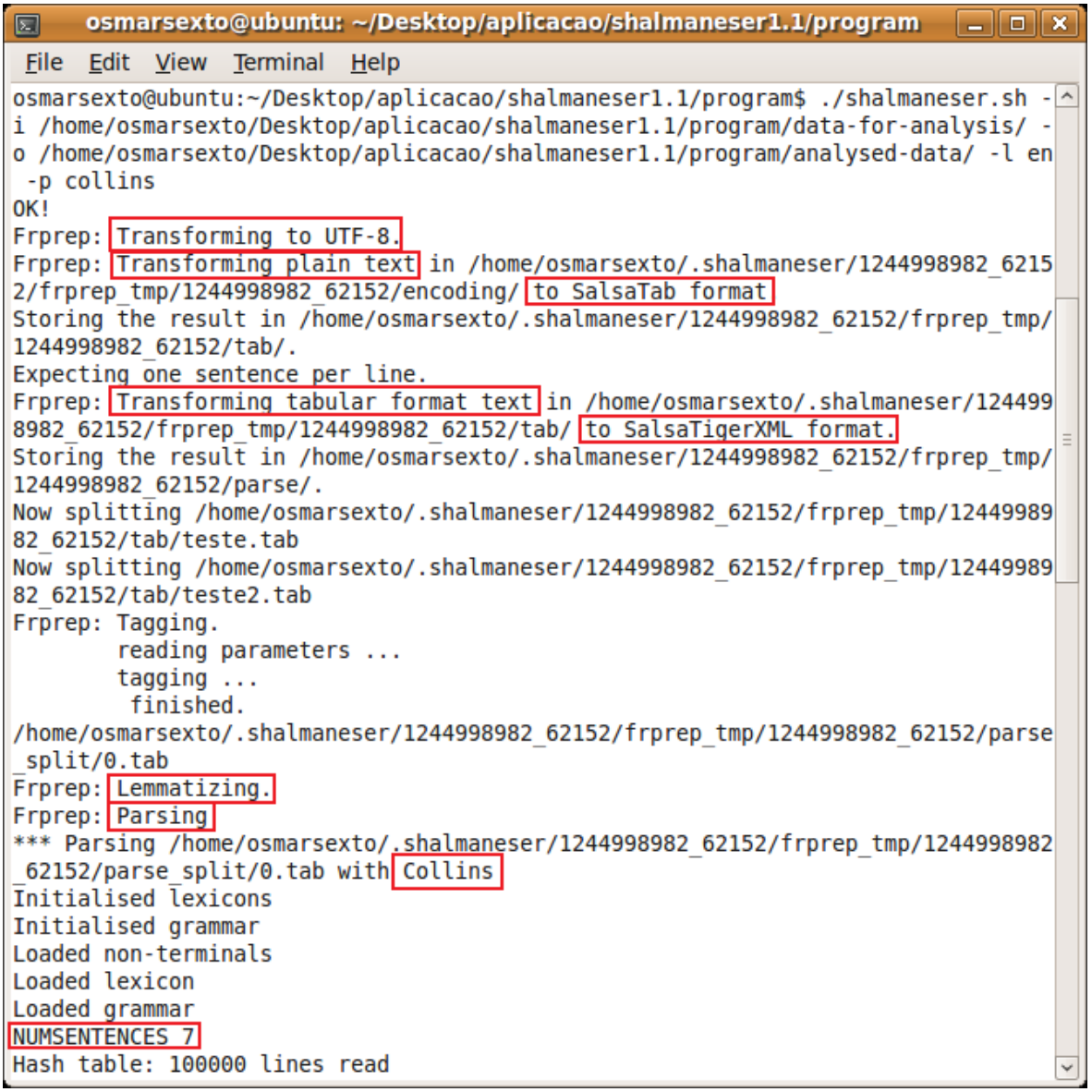

Figura 5.7: Processos e tarefas executados pelo *FRPREP*, *TreeTagger* e *Collins Parser*

Após o processo de *parsing*, o *FRPREP* faz o pós-processamento no arquivo *XML* gerado, que é gravado com o nome *0.XML*, na pasta do próprio *FRPREP*. Neste momento, entra em ação o outro componente do *Shalmaneser* – o *FRED* (*Frame Disambiguation).* A primeira ação é eliminar os dados antigos e desnecessários para a continuidade do processamento. Em seguida, o componente carrega o arquivo *0.XML* presente na pasta do *FRPREP* e aplica os classificadores com o objetivo de eliminar as possíveis ambiguidades. Em seguida, ele grava o arquivo de saída (também em formato *XML*) em sua respectiva pasta (*fred_out).* Todo o processamento do componente *FRED* pode ser visto através da figura 5.8. As partes mais importantes estão destacadas.

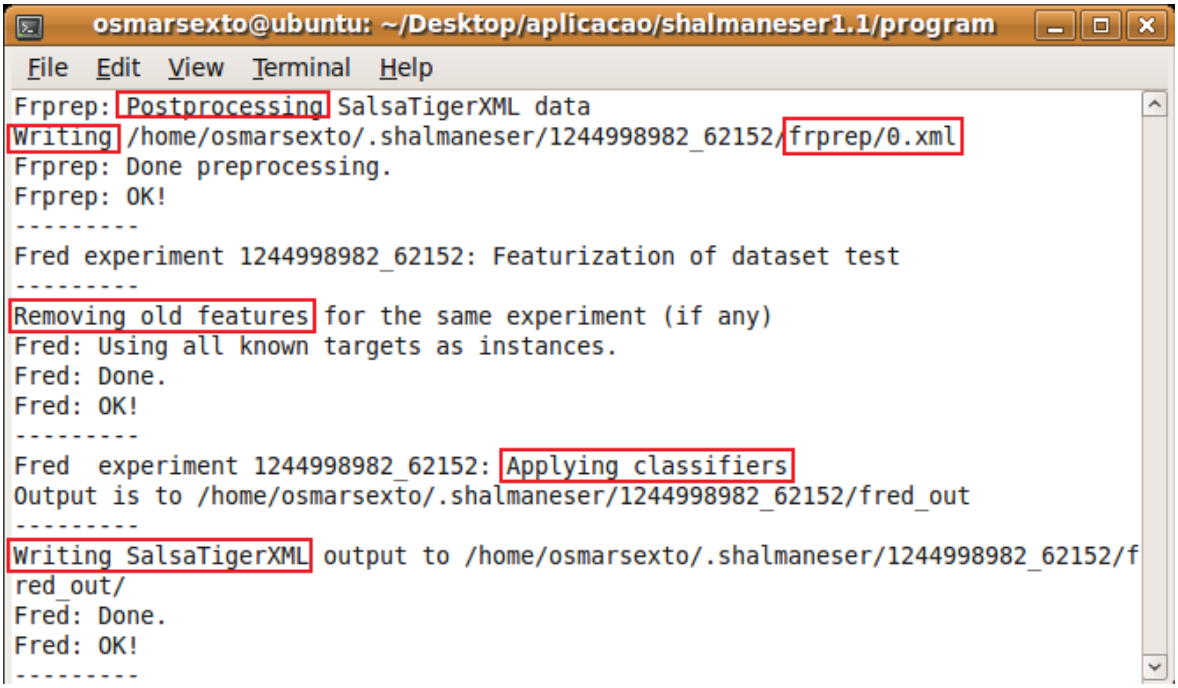

Figura 5.8: Processamento do componente *FRED*

Após o processamento do *FRED*, o componente *ROSY* começa a executar sua tarefa. Ela carrega o arquivo *XML* gerado como saída do *FRED* e inicia a atribuição de papéis semânticos para os constituintes da sentença. De todos os constituintes da estrutura sintática da sentença, apenas alguns podem realmente ser rotulados com um papel semântico. Então, o componente *ROSY*, através do uso de um *parser* com uma estrutura em árvore, descobre quais constituintes podem suportar um papel semântico de fato.

Cada um dos *frames* é identificado e o sub-componente *argrec* aplica um algoritmo classificador (com o auxílio da ferramenta *Mallet*) para cada um deles. Como resultado, o algoritmo retorna o grau de precisão de cada um dos *frames* analisados. Após o processamento dos *frames*, o componente *ROSY* se encarrega de remover do banco de dados, a tabela com os dados de teste.

Todo o processamento do componente *ROSY* pode ser visto através da figura 5.9. As partes mais importantes estão destacadas.

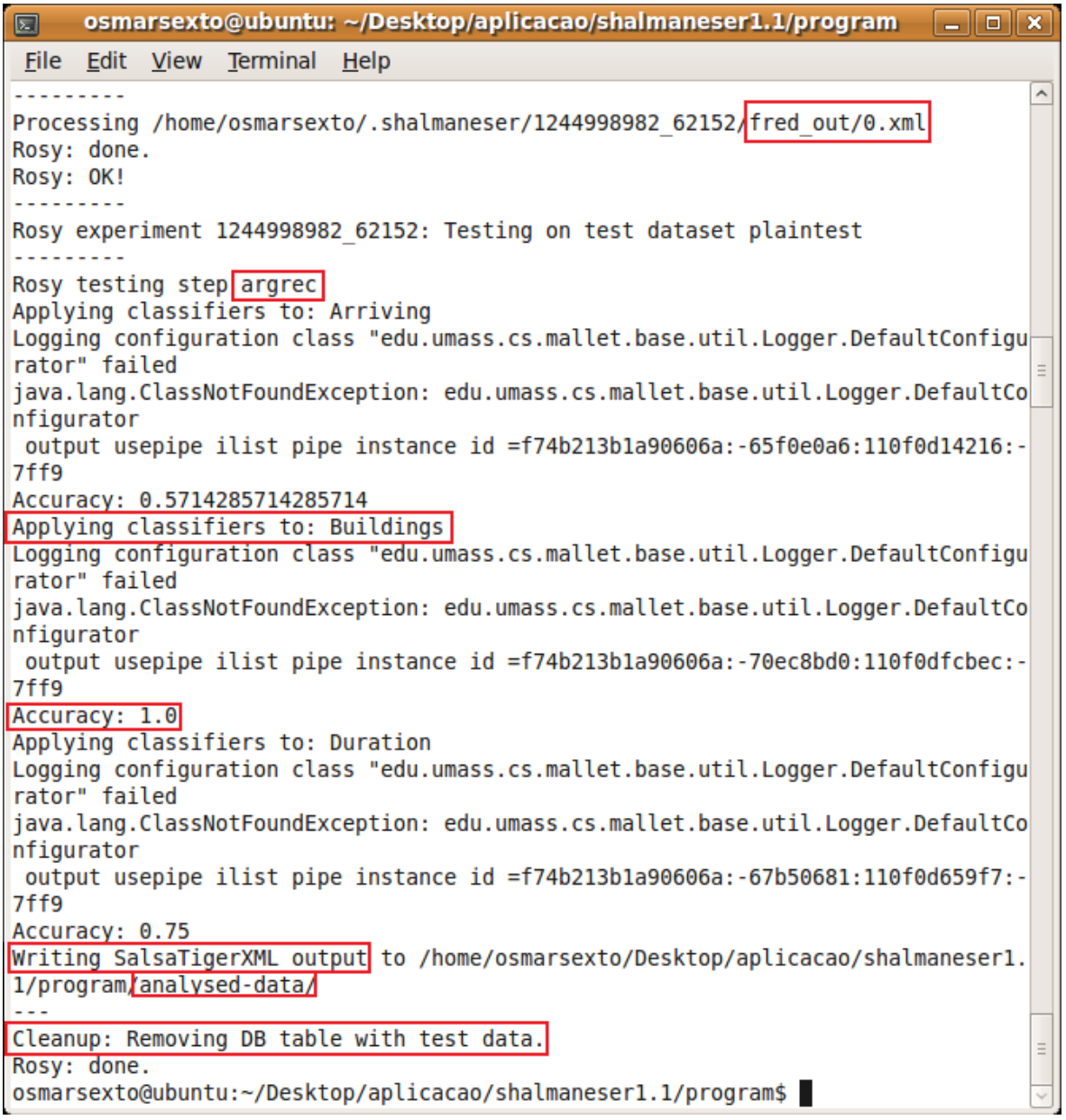

Figura 5.9: Processamento do componente ROSY

Após todo o processamento feito pelo *FRPREP*, *FRED* e *ROSY*, o arquivo *XML* final foi gerado e gravado na pasta destino – no caso, na pasta *analysed-data*. Neste momento, ele já pode ser carregado pela ferramenta *Salto* e seus resultados já podem ser analisados.

5.4 Demonstração dos Resultados

Os resultados gerados após o processamento e a rotulação são escritos num arquivo *XML* que é gravado na pasta escolhida como destino – no caso deste experimento computacional, na pasta *analysed-data*.

Todo o processamento acontece sobre um arquivo *txt* de entrada. Esse arquivo de texto deve conter as sentenças das quais se deseja extrair a rotulação dos papéis semânticos. É necessário que o arquivo texto contenha uma frase por linha, para que a análise seja bem sucedida. O ponto final indica o final da sentença e garante que as frases serão rotuladas com a maior precisão possível. Além disso, as sentenças devem estar na língua inglesa, uma vez que a base de dados *linguística* escolhida foi a *FrameNet*, versão 1.3.

Um exemplo do arquivo texto com as sentenças que passarão pela tarefa de rotulação pode ser visto a seguir, na figura 5.10.

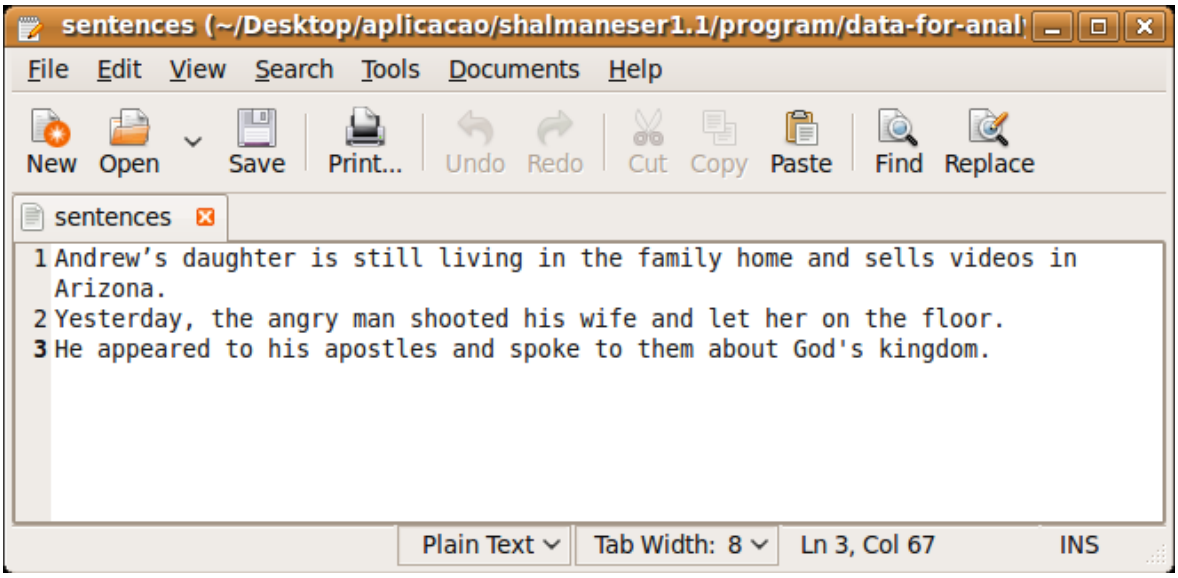

Figura 5.10: Arquivo *TXT* com as sentenças que serão rotuladas

Este foi o arquivo *txt* que passou por todo o processamento: lematização, *Part-of-Speech Tagging*, rotulação sintática e rotulação semântica. Após todos esses processos, os resultados foram gravados em um arquivo *XML* que será aberto e analisado através da ferramenta *Salto – software* responsável por representar a rotulação semântica, interligando-a com a rotulação sintática. O programa exibe as classes semânticas (*frames* semânticos), os papéis semânticos (*semantic roles*) e as classes gramaticais a que cada termo da sentença se refere.

Para executar a ferramenta *Salto*, deve-se abrir o terminal e invocá-la através de um comando, como mostra a figura 5.11.

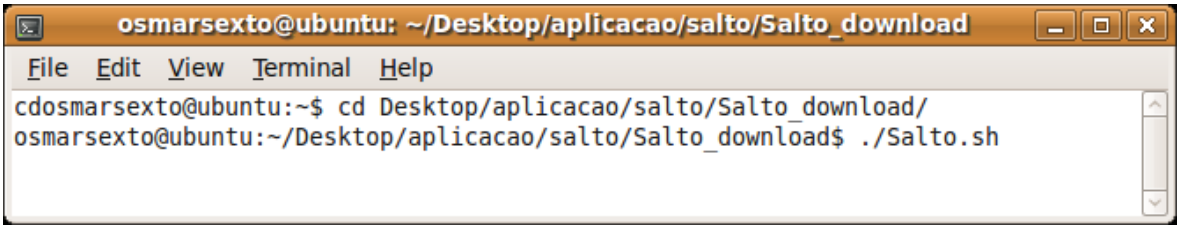

Figura 5.11: Execução da ferramenta *Salto*

Dentro da ferramenta *Salto*, deve-se abrir o arquivo *XML* que contém as sentenças e suas respectivas rotulações. O carregamento do arquivo dá-se pela simples ação: ***File*** => ***Open File***. Neste instante, abre-se a interface gráfica do *Salto*, onde é possível navegar por todas as sentenças, redimensionar os componentes gráficos e fazer a análise da rotulação sintática e da descrição dos papéis semânticos.

Em termos sintáticos, os termos são rotulados de acordo com sua função gramatical na sentença. A marcação é feita através de siglas, que são descritas na tabela 5.1 a seguir.

Tabela 5.1 – Siglas e descrições das classes gramaticais da rotulação sintática

Sigla	**Descrição**
S	Sentença
VP	Frase Verbal
PP	Frase Preposicionada
NPB	Frase Subjetiva
ADVP	Frase Adverbial
VP-A	Frase Verbal

Além da rotulação sintática, a ferramenta *Salto* também exibe os *frames* (destacados por retângulos preenchidos) referentes às unidades lexicais e seus respectivos papéis semânticos (destacados pelos retângulos vazados).

A primeira sentença rotulada foi: "*Andrew's daughter is still living in the family home and sells videos in Arizona*". O resultado está exibido pela figura 5.12, a seguir.

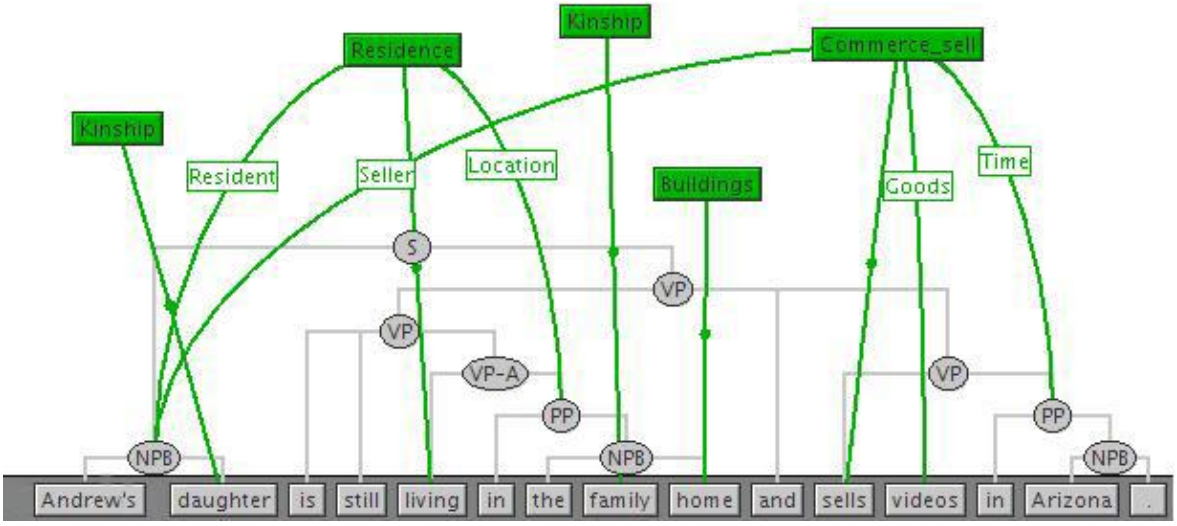

Figura 5.12: Rotulação sintática e semântica – Exemplo 1

Na figura 5.12, foram identificados os *frames* “Kinship”, “Residence”, “Buildings” e “Commerce_sell”, evocados por suas respectivas unidades lexicais. Foram identificados ainda, os papéis semânticos “Resident” e “Location” referentes ao *frame* “Residence” e os papéis semânticos “Seller”, “Goods” e “Time”, referentes ao *frame* “Commerce_sell”. Além disso, cada termo da sentença foi rotulado sintaticamente.

Neste exemplo, percebe-se que a rotulação não é perfeita. A origem dessa imperfeição vem da classificação dos *frames* (feita pelo componente *argrec* do ROSY), que nem sempre atinge o grau de precisão máximo. Uma evidência acontece no exemplo 1, onde “in Arizona” deveria ser rotulado como “location” e não como “time”.

A segunda sentença rotulada foi: “*Yesterday, the angry man shooted his wife and let her on the floor*”. O resultado está exibido pela figura 5.13, a seguir.

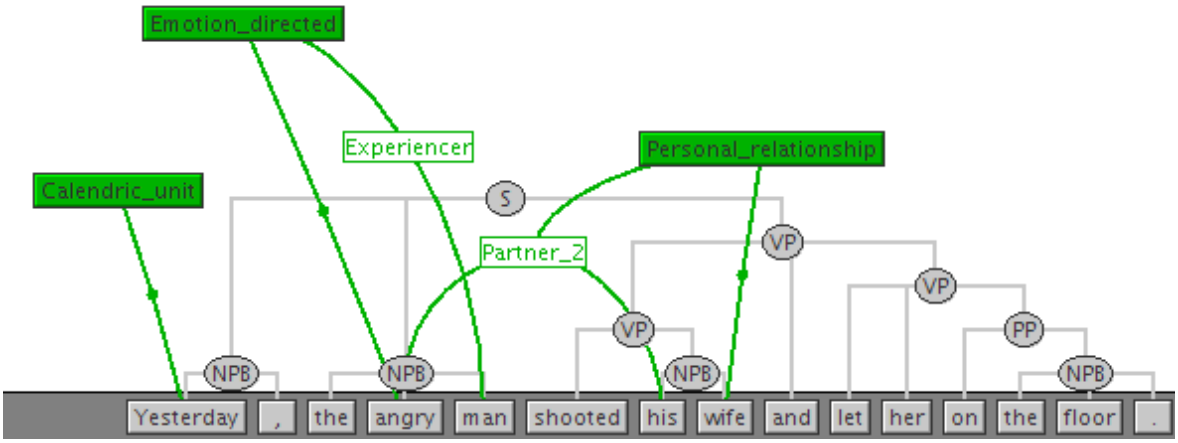

Figura 5.13: Rotulação sintática e semântica – Exemplo 2

Na figura 5.13, foram identificados os *frames* “Calendric_unit”, “Emotion_directed” e “Personal_relationship”, evocados por suas respectivas unidades lexicais. Foram identificados ainda, o papel semântico “Experiencer” referente ao *frame* “Emotion_directed” e o papel semântico “Partner_2”, referente ao *frame* “Personal_relationship”. Além disso, cada termo da sentença foi rotulado sintaticamente.

Por último, a terceira sentença rotulada foi: “*He appeared to his apostles and spoke to them about God’s kingdom*”. O resultado está exibido pela figura 5.14, a seguir.

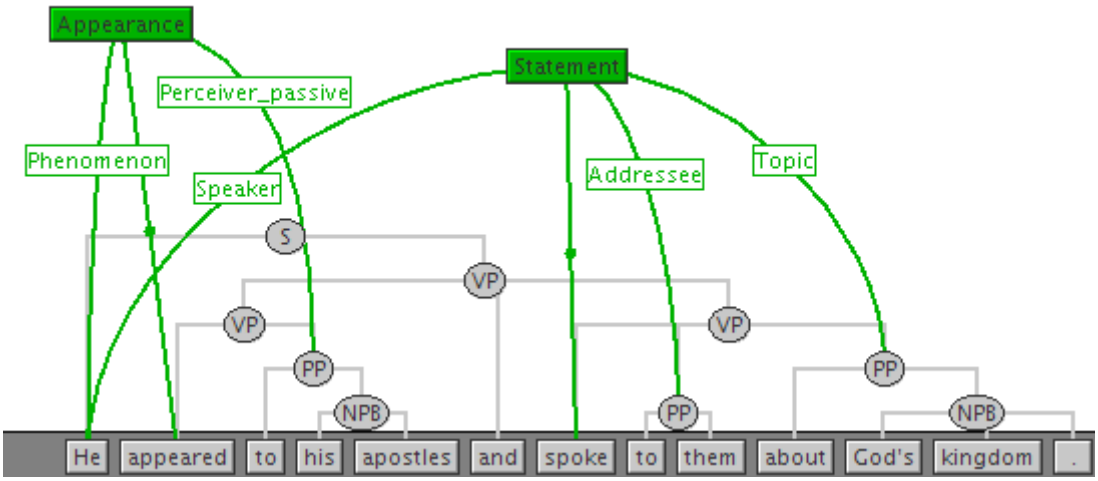

Figura 5.14: Rotulação sintática e semântica – Exemplo 3

Na figura 5.14, foram identificados os *frames* “Appearance” e “Statement”, evocados por suas respectivas unidades lexicais. Foram identificados ainda, os papéis semânticos “Phenpmenon” e “Parceiver_passive”, referentes ao *frame* “Appearance” e os papéis semânticos “Speaker”, “Address” e “Topic”, referentes ao *frame* “Statement”. Além disso, cada termo da sentença foi rotulado sintaticamente.

Capítulo 6

Considerações Finais e Trabalhos Futuros

A evolução da tecnologia de informação é acompanhada do inevitável acúmulo do volume de dados, tornando os bancos de dados instrumentos imprescindíveis. Além disso, a grande maioria desses dados está desestruturada, necessitando de um tratamento e pré-processamento. É nesse cenário que surge a importância do *KDD* (*Knowledge Discovery in Databases*) que, através de suas etapas, realiza a análise, tratamento e processamento dos dados. A partir desses processos – e aplicando uma técnica de mineração de dados – encontram-se padrões existentes nos dados, que levam ao conhecimento compreensível, válido e potencialmente útil. Todo o processo de *KDD* pode ser estendido às bases de dados linguísticas, dando suporte à descoberta de padrões e conhecimento e permitindo a execução de tarefas relacionadas a esse cenário, como a rotulação sintática e a rotulação de papéis semânticos.

Durante décadas, as pesquisas em unidades lexicais, gramáticas e demais artifícios semânticos evoluíram notavelmente. Na década de 1990, técnicas e algoritmos de aprendizado de máquina foram desenvolvidos e aperfeiçoados, visando o campo da linguística computacional. Estes métodos de IA (Inteligência Artificial) mostraram-se eficientes em adquirir o conhecimento necessário para interpretações semânticas, assim como para extrair as propriedades dos predicados e suas relações.

Este trabalho realizou um estudo teórico sobre todas as áreas que envolvem a rotulação automática de papéis semânticos. Dentre as áreas citadas, estão o processo de *KDD*, a técnica de *Data Mining*, os conceitos de rotulação sintática e semântica e a arquitetura da *FrameNet*. Depois, as ferramentas utilizadas no estudo de caso foram descritas e um experimento computacional foi realizado e exemplificado.

Dentre os ganhos obtidos com a realização desta monografia, podem ser ressaltados os seguintes:

- Revisão de conceitos envolvendo o processo de *KDD* e a mineração de dados.
- Estudo aprofundado em uma base de dados linguística – a *FrameNet*.
- Conhecimento adquirido no campo da linguística computacional, envolvendo a rotulação de papéis semânticos, a tarefa de *Part-of-Speech Tagging* e a conexão dessas tarefas com a descoberta de conhecimento em banco de dados.

- Utilização de outro sistema operacional (*Linux)*, já que algumas ferramentas não eram compatíveis.
- Estudo, instalação e configuração de ferramentas que auxiliam na tarefa da rotulação de papéis semânticos, tais como: *Shalmaneser, TreeTagger, Mallet* e *Salto.*
- Noções das áreas relacionadas ao estudo, como o Aprendizado de Máquina. Consequentemente, um estudo superficial sobre o algoritmo de classificação de *Naïve Bayes*.

Algumas das limitações que ocorreram no decorrer da elaboração desta monografia foram:

- Necessidade de alternar o sistema operacional que vinha sendo usado no decorrer do trabalho.
- Devido ao pouco conhecimento inicial sobre o assunto, foi gasto mais tempo com pesquisa, leitura e estudos teóricos do que o planejado inicialmente.
- Falta de disponibilidade de ferramentas para a realização do experimento computacional. Desta maneira, foi necessário o uso de várias ferramentas integradas para se atingir o objetivo final.

Por fim, são sugeridos como trabalhos futuros:

- Utilização da base de dados da *FrameNet* no *Weka* – ferramenta de mineração de dados rica em recursos e cheia de funcionalidades. Para tal, a base tem de ser transformada do formato *XML* para o formato *ARFF*.
- Utilização da ferramenta *RapidMiner*, que também conta com muitos recursos e funcionalidades, para se obter novas regras, padrões e resultados.
- Adaptação das ferramentas de rotulação semântica para uma base de dados linguística como a *FrameNet Brasil* – versão da *FrameNet* para o português usado no Brasil.
- Comparação entre as diferentes ferramentas de mineração de dados, buscando melhores resultados.
- Investigação de outros algoritmos com o objetivo de analisar seus níveis de desempenho em relação aos algoritmos usados.

Referências Bibliográficas

ATKINS, S.; FILLMORE, C. J.; JOHNSON, C. **Lexicography relevance: selecting information from corpus evidence**. International Lexicographic Journal, 2003. p.251-280.

BAKER, C. F.; FILLMORE, C. J.; LOWE, J. B. **The Berkeley FrameNet Project**. In: *Proceedings of COLING-ACL*. Montreal, 1998. p.86-90.

BRISCOE, T.; CARROLL, J. **Automatic extraction of subcategorization from corpora**. In: *Proceedings of the 5th ACL Conference on Applied Natural Language Processing (ANLP)*. Washington, DC, 2002. p.356-363.

COLLINS, M. **A New Statistical Parser Based on Bigram Lexical Dependencies**. In: *Proceedings of the 34th Annual Meeting of the ACL*. Santa Cruz. 1997.

COPESTAKE, A.; FLICKINGER, D. **An open-source grammar development environment and broad-coverage English grammar using HPSG**. In: *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC)*. Athens, Greece, 2000. p.591-600.

ERK, K.; PADO, S.; Shalmaneser – **A Toolchain for shallow semantic parsing**. Computational Linguistics. Saarbrücken, Alemanha. 2007.

FAYYAD, U.; SHAPIRO, P. G.; SMYTH, P. **Knowledge Discovery and Data Mining: Towards a Unifying Framework**. 1996.

FERNANDES, W. P. D. **Rotulação Semântica Automática de Sentenças para a Framenet**. Departamento de Ciência da Computação, UFJF (Monografia Final Curso), 2008.

FILLMORE, C. J. **Frame semantics**. In: *Linguistics in the morning calm.* Seoul, 1982. p.111-137.

FILLMORE, C. J.; BAKER, C. F. **Frame semantics for text understanding**. In: *Proceedings of WordNet and Other Lexical Resources Workshop*. Pittsburgh, 2001. p.59-64.

FILLMORE, C. J.; RUPPENHOFER, J.; BAKER, C. F. **Framenet and representing the link between semantic and syntactic relations**. In: *Churen Huang and Winfried Lenders. Frontiers in Linguistic volume I of Language and Linguistics Monograph Series B.* Institute of Linguistics, Academia Sinica, Taipei. 2004. p.19-59.

GILDEA, D.; JURASFSKY D. **Automatic labeling of semantic roles**. Computational Linguistics. [S.l: s.n.], 2002. p.245-288.

HIRST, G. **Semantic Interpretation and the Resolution of Ambiguity**. Cambridge University Press, Cambridge. 1987.

MCCALLUM, A. K. **Mallet: A Machine Learning for Language Toolkit**. Disponível em <http://mallet.cs.umass.edu>. 2002. Acesso em 22 jun. 2009.

MERLO, P.; STEVENSON, S.; **Automatic verb classification based on statistical distributions of argument structure**. Computational Linguistics. [s.l.]. 2001. p.373-408.

PRADHAN, S.; HACIOGLU, K; WARD, W; MARTIN, J.; JURAFSKY D**. Semantic role chunking combining complementary syntactic views**. In: *Proceedings of the Ninth Conference on Computational Nature Language Learning (CoNLL-2005).* Ann Arbor, MI. 2005. p.217-220.

PUSTEJOVSKY, J. **The Generative Lexicon**. MIT Press, Cambridge, MA. 1995.

ROSINI, A.; PALMISIANO, A. **Administração de Sistemas de Informação e a Gestão do Conhecimento**. [S.l.]. Cengage Learning Editores, 2003. 229p.

RUPPENHOFER, J.; ELLSWORTH, M.; PETRUCK, M. R. L.; SCHEFFCZYK, J. **FrameNet II: Extended Theory and Practise**. Disponível em <http://framenet.icsi.berkeley.edu/book/book.html>. Acesso em: 20 mai. 2009.

SCHMID, H. **Probabilistic Part-of-Speech Tagging using Decision Trees**. Stuttgart, Alemanha. 1996.

SURDEANU, M.; HARABAGIU, J.; WILLIAMS, J.; AARSETH, P. **Using predicate-argument structures for information extraction**. In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Sapporo, Japan, 2003. p.8-15.

XUE, N.; PALMER, M. **Calibrating features for semantic role labeling**. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Barcelona, Spain. 2004. p.88-94.

Anexo A

Código do Classificador *Naïve Bayes*

```
# NaiveBayes


# Katrin Erk April 05
#
# Naive Bayes classifier with Lidstone smoothing
# here: work with actual probabilities, not with log probabilities,
# and also calculate context probability,
# such that the weights produced by the classifier are
# probabilities and can be used further.
#
# The class NaiveBayesClassifier inherits from FredAbstractClassifier
# and provides methods for computing classifiers and for using classifiers.

require "FileZipped"
require "StandardPkgExtensions"

require "FredAbstractClassifier"
require "FredCountInstances"
require "ClassifierResult"

# The class NaiveBayesClassifier inherits from FredAbstractClassifier
# and provides methods for computing classifiers and for using classifiers.
class NaiveBayesClassifier < FredAbstractClassifier

  ###
  # classifier_name
  #
  # returns a string, the name prefix for classifier file names
  # and the name for the directory in which classifiers will be stored
  def NaiveBayesClassifier.classifier_name()
    "naive_bayes"
  end

  ###
  # start_writing_classifier
  #
  # prepare file(s) for writing a classifier
  def start_writing_classifier(lemma, # string: make classifier specific to this lemma
                               sense="") # string: make classifier specific to this sense
                                         # (may be an empty string if we are not doing binary
classifiers)

    # record current lemma
    @lemma = lemma

    # construct output files:
    begin
      # one for sense probabilities
      @sense_file   =   FileZipped.new(@directory   +   make_classifier_name("sense",   lemma,
sense), "w")
      # one for lemma/sense/feature probabilities
      @feature_file  =  FileZipped.new(@directory  +  make_classifier_name("feature",  lemma,
sense), "w")
    rescue
      raise "Couldn't write to classifier files"
    end
  end

  ###
  # start_using_classifier
  #
  # prepare file(s) for using a classifier
  def start_using_classifier(lemma, # string: make classifier specific to this lemma
                             sense="") # string: make classifier specific to this sense
```

```
                                    # (may be an empty string if we are not doing binary
classifiers)
    # record current lemma
    @lemma = lemma

    # record sense if we have sense-specific classifiers
    if sense.empty?
      @sense = nil
    else
      @sense = sense
    end


    # open input files:
    begin
      # one for sense probabilities
      sense_file = FileZipped.new(@directory + make_classifier_name("sense", lemma, sense))
      # one for lemma/sense/feature probabilities
      feature_file  =  FileZipped.new(@directory  +  make_classifier_name("feature",  lemma,
sense))
    rescue
      $stderr.puts "No classifier files for #{lemma}, not assigning anything."
      @sense_prob = @feature_prob = nil
      return
    end

    # and read probabilities
    # sense_prob: hash '#{lemma} #{sense}'(string) => probability(float)
    @sense_prob = read_prob(sense_file)
    # feature_prob: hash '#{lemma} #{sense} #{feature}'(string) => probability(float)
    @feature_prob = read_prob(feature_file)

    sense_file.close()
    feature_file.close()

    if @sense_prob.length() > 0
      # we have seen training data
      # assert sum prob == 1

      sum = @sense_prob.values.big_sum(0.0) { |p| p}
      if sum < 0.99 or sum > 1.00001
        $stderr.puts "WARNING: sum sense prob: #{sum}"
      end
    end
#    sum = Hash.new(0.0)
#    @feature_prob.each_pair { |lsf, p|
#      sense = lsf.split()[1]
#      sum[sense] = sum[sense] + p
#    }
#     sum.each_pair { |sense, ps|
#       if ps < 0.99 or ps > 1.00001
#       $stderr.puts "WARNING: feature prob sense #{sense}: sum #{ps}"
#       end
#     }
  end

  ###
  # close classifier
  def close_classifier()
    if @sense_file
      @sense_file.close()
      @sense_file = nil
    end
    if @feature_file
      @feature_file.close()
      @feature_file = nil
    end
  end

  ###
  # handle_training_instances
  #
  # use the given instances to train a classifier
  #
  # this method accepts as its parameter a reader object
```

```
# that has methods instances() and each_instance().
# instances() returns a list of instances,
# each_instance() yields each instance in turn.
#
# An instance is a hash
#  "sentid" => sentence ID(string),
#   "lemma" => lemma(string),
#   "sense" => sense(string),
#   "features" => array of FredFeature objects
def handle_training_instances(reader) # object with methods as described above

  ###
  # count frequencies of lemmas, senses, features
  counter = FredCountInstances.new(reader)
  ###
  # compute probabilities from the frequencies
  prob = NaiveBayesProbWithLidstoneSmoothing.new(counter.lemma_freq,
                                                 counter.sense_freq,
                                                 counter.feat_freq,
                                                 @exp)

  ###
  # print out sense probabilities
  counter.senses(@lemma).each { |sense|

    @sense_file.print @lemma, "\t", sense, "\t"
    @sense_file.print prob.prob_sense(@lemma, sense), "\n"

  }

  ###
  # print out probabilities of features given sense

  # collect list of all feature names, for later
  all_features = Array.new

  # for each sense of the lemma
  counter.senses(@lemma).each { |sense|

    # print probabilities for known features
    counter.features(@lemma, sense).each { |feature|

     all_features << feature

     @feature_file.print @lemma, "\t", sense, "\t", feature, "\t"
     @feature_file.print prob.prob_feature(@lemma, sense, feature), "\n"
    }

    # print probabilities for unseen features
    @feature_file.print @lemma, "\t", sense, "\t", "<UNSEEN>", "\t"
    @feature_file.print prob.prob_feature(@lemma, sense, "<UNSEEN>"), "\n"
  }


  @sense_file.flush()
  @feature_file.flush()
end

###
# handle_test_instances
#
# classify the given instances with the appropriate classifier
#
# this method accepts as its parameter a reader object
# that has methods instances() and each_instance().
# instances() returns a list of instances,
# each_instance() yields each instance in turn.
#
# An instance is a hash
#  "sentid" => sentence ID(string),
#   "lemma" => lemma(string),
#   "sense" => sense(string),
#   "features" => array of FredFeature objects
#
# returns: a ClassifierResult object
```

```
  def handle_test_instances(reader) # object with methods as described above

    # store classification results here, in a ClassificationResult object
    results = ClassifierResult.new(NaiveBayesClassifier.classifier_name(), @sense)

    # what are the senses for this lemma?
    # keys of @sense_prob: strings of the form '#{lemma} #{sense}'
    if @sense_prob
      senses = @sense_prob.keys.map { |lemma_and_sense|
        lemma_and_sense.split.last
      }
    else
      # no classifier for this lemma
      senses = []
    end

    # handle each instance that the reader has
    reader.each_instance { |inst|

      if senses.empty?
        # I don't know any senses for this lemma
        # record an empty result
        results.record_result(InstanceResult.new())

      elsif senses.length() == 1
        # there is exactly one sense for this lemma
        results.record_result(InstanceResult.new([[senses.first, 1.0]]))

      else
        # there are at least two senses for this lemma
        # sense to be assigned:
        # argmax_{sense} (logprob(sense) +
        #                 sum_{context_word} logprob(context_word | sense))

        # feature names for this instance: array of official feature names.
         # filter features that are unseen for all senses
        features = Array.new

         inst["features"].map { |feature_obj|
           feature_obj.official_feature_name()
         }.each { |feature|
           # select feature names such that there is at least one sense
           # for which @feature_prob[@lemma + " " + sense + " " + f] is defined
           senses.each { |sense|
             if @feature_prob[@lemma + " " + sense + " " + feature]
               features << feature
               break
             end
           }
         }

         unless @sense_prob
           raise "Shouldn't be here"
         end

        # compute all [sense, P(sense)P(context|sense)] pairs in this_result
        this_result_cxmissing = senses.map { |sense|
           [
             sense,
             @sense_prob[@lemma + " " + sense] * feature_prob_sum(features, sense)
           ]
         }

        # compute P(context) = sum_sense P(sense)P(context|sense)
        p_context = 0.0
        this_result_cxmissing.each { |sense, prob|
          p_context += prob
        }

#         # test output
#         features.each { |f|
#           $stderr.print f, " "
#           senses.each { |sense|
#             $stderr.print sense, " ", sprintf("%.2e", feature_prob_sum([f], sense)), " "
```

```
#             unless @feature_prob[@lemma + " " + sense + " " + f]
#               $stderr.print " UNSEEN "
#             end
#           }
#           $stderr.puts
#         }

        # P(sense|context) = entry_in_this_result_cxmissing / p_context
        this_result = this_result_cxmissing.map { |sense, prob|

          if p_context == 0.0
            val = @sense_prob[@lemma + " " + sense]
          else
            val = prob / p_context
          end
          if val > 1.0001
            $stderr.puts "ERROR: not a probability?"
            features.each { |f|
              $stderr.puts f + " " + sprintf("%.2e", feature_prob_sum([f], sense))
            }
            $stderr.print " sense=#{sense} sp=", sprintf("%.2f", @sense_prob[@lemma + " " +
sense])
            $stderr.print " fp=", sprintf("%.2e", feature_prob_sum(features, sense))
            $stderr.print " p=", sprintf("%.2f", val), "\n"
          end

#             $stderr.print "sense=#{sense} sp=", sprintf("%.2f", @sense_prob[@lemma + " " +
sense])
#           $stderr.print " fp=", sprintf("%.2e", feature_prob_sum(features, sense))
#           $stderr.print " cp=", sprintf("%.2e", p_context)
#           $stderr.print " p=", sprintf("%.2f", val), "\n"

          [sense, val]
        }
#         $stderr.puts "any key"
#         $stdin.gets()

        # assert sum_prob == 1
        sum = 0.0
        this_result.each { |sense, prob| sum += prob }
        if sum < 0.99 or sum > 1.00001
          $stderr.puts "Error: probabilities don't sum up to 1: "
          $stderr.puts this_result.map { |s, prob| "#{s}:#{prob}" }.join(", ")
        end

        # record result
        results.record_result InstanceResult.new(this_result)
      end
    }
    return results
  end

  ########################################
  private

  ###
  # returns a string, the name of a classifier file
  def make_classifier_name(sense_or_feature, # string: 'sense', 'feature', 'context': what
kind of frequencies?
                           lemma,  # string: lemma for which we are storing frequencies
                           sense) # string: sense, may be nil or empty

    # sense may be left empty, such that this method can make
    # both sense-specific and lemma-specific classifier filenames
    if sense.nil? or sense.empty?
      sense = "_"
    end

    # make filename (without directory)
    return NaiveBayesClassifier.classifier_name() + "." +
      lemma + "." +
      sense + "." +
      sense_or_feature
  end
```

```
  ########################
  # read_prob:
  #
  # given a file with probabilities,
  # read the info into a hash and return that.
  #
  # file format:
  # <some columns> [whitespace] log_probability
  #
  # hash that is returned:
  # keys are all columns but the last joined by " ",
  # values are log probabilities (floats)

  def read_prob(file) # stream to read from
    retv = Hash.new

    # for each line of the file:
    while (line = file.gets())
      # read line, split at whitespace
      line = line.chomp().split()

      # the probability is in the last column.
      # make it into a float
      value = line.pop().to_f
      # make all other columns into the key for the hash
      retv[line.join(" ")] = value
    end

    # return the hash
    return retv
  end

  #############
  # feature_prob_sum
  #
  # compute sum_{word in words} log_prob(word | sense)
  # take into account unseen senses
  def feature_prob_sum(features, # array: list of features(strings)
                       sense) # string: sense
    sum = 1.0
    features.each { |f|
#      $stderr.puts "feature prob " + sprintf("%.4e", prob_f_or_unseen(f, sense))
      sum *= prob_f_or_unseen(f, sense)
    }
    return sum
  end

  # prob_f_or_unseen(feature, sense)
  #
  #
  # if feature_prob["lemma sense feature"] is defined, return the value at
  # that point. it is log prob(feature | "lemma sense")
  #
  # otherwise, return the probability of "lemma sense" for an unseen context word:
  # f_prob["lemma sense <UNSEEN>"]

  def prob_f_or_unseen(f,  # string: feature name
                       sense) # string: sense

    unless @feature_prob
       $stderr.puts "NaiveBayes: Shouldn't be here, but continuing anyway"
      return 0.0
    end

    prob = @feature_prob[@lemma + " " + sense + " " + f]

    unless prob.nil?
      return prob
    end

    prob = @feature_prob[@lemma + " " + sense + " " + "<UNSEEN>"]
    unless prob.nil?
      return prob
    end
```

```
    $stderr.puts "Error: couldn't determine cond. probability for " +
      @lemma + " " + sense.to_s + " " + f
    return 0.0
  end
end

##########################33
# NaiveBayesProbWithLidstoneSmoothing
# computes the probabilities

class NaiveBayesProbWithLidstoneSmoothing
  def initialize(lemma_freq, # hash lemma(string) => frequency(float)
                 sense_freq, # hash lemma(string) => hash sense(string) => frequency(float)
                 feat_freq,  # hash lemma(string) => hash sense(string) =>
                             #   hash feature(string) => frequency(float)
                 exp_obj) # FredConfigData object

    # record given frequency hashes
    @lemma_freq = lemma_freq
    @sense_freq = sense_freq
    @feat_freq = feat_freq

    # determine smoothing factor
    @lambda = exp_obj.get("smoothing_lambda")
    if @lambda.nil?
      @lambda = 1.0
    end

    # hash: lemma -> num features(lemma)
    @num_features = Hash.new
    @feat_freq.each_key { |lemma|
      features = Array.new
      @feat_freq[lemma].each_value { |hash|
        features.concat hash.keys
      }
      @num_features[lemma] = features.uniq.length()
    }
  end

  # sense probability
  #
  #   P(sense) = f(sense) / f(lemma)
  def prob_sense(lemma, sense)
    return (@sense_freq[lemma][sense].to_f) /
      (@lemma_freq[lemma].to_f)
  end

  # feature probability
  #
  #   P(feat | sense) = f(feat, sense) / f(sense)
  # smoothing, simple solution: Lidstone's law
  #   P(feat | sense) = (f(feat, sense) + lambda) /
  #                     (f(sense) + lambda * (num_features_for_this_lemma + 1))
  def prob_feature(lemma, sense, feature)

    if @feat_freq[lemma].nil?
      return 0.0
    end

    if @feat_freq[lemma][sense].nil? or
       @feat_freq[lemma][sense][feature].nil?

      # unseen feature
      return @lambda /
       (@sense_freq[lemma][sense].to_f +
        @lambda * (@num_features[lemma] + 1).to_f)
    else

      # feature we've seen before
      return (@feat_freq[lemma][sense][feature].to_f + @lambda) /
       (@sense_freq[lemma][sense].to_f +
        @lambda * (@num_features[lemma] + 1).to_f)
    end
  end
end
```