Thiago Ribeiro Nunes

Construção de Extratores de Relações por Supervisionamento à Distância

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio.

Orientador: Prof. Daniel Schwabe

Rio de Janeiro

Agosto de 2012

Thiago Ribeiro Nunes

Construção de Extratores de Relações por Supervisionamento à Distância

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-graduação em Informática do Departamento de Informática do Centro Técnico e Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Daniel Schwabe
Orientador
Departamento de Informática – PUC-Rio

Prof. Ruy Luiz Milidiú
Departamento de Informática – PUC-Rio

Prof. Edward Hermann Haeusler
Departamento de Informática – PUC-Rio

Prof. José Eugenio Leal
Coordenador Setorial do Centro
Técnico Científico - PUC-Rio

Rio de Janeiro, 9 de agosto de 2012

Thiago Ribeiro Nunes

Possui graduação em Ciência da Computação pela Universidade Candido Mendes - Campos dos Goytacazes (2009). Tem experiência na área de Ciência da Computação, com ênfase em Sistemas de Informação, atuando principalmente nos seguintes temas: Desenvolvimento de aplicações Web, Gestão do Conhecimento e Inteligência Computacional. Possui interesse nas áreas de Web Semântica, Aprendizado de Máquina, Processamento de Linguagem Natural e Extração de Informação.

Ficha Catalográfica

Nunes, Thiago Ribeiro

Construção de Extratores de Relações por Supervisionamento à Distância / Thiago Ribeiro Nunes; orientador: Daniel Schwabe. – 2012.
112 f : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2011.
Inclui bibliografia

1. Informática – Teses. 2. Extração de Relações. 3. DBpedia. 4. Wikipedia. 5. Framework. I. Schwabe, Daniel. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD:004

Dedico este trabalho aos meus pais, Iva Rangel Ribeiro Nunes e Reginaldo Soares Nunes, por terem me ensinado a nunca desistir dos meus objetivos e por terem me incentivado de todas as formas possíveis.

Agradecimentos

Ao meu orientador, Professor Daniel Schwabe, por todo o apoio e orientação durante a realização deste trabalho, pela confiança e pelos inúmeros ensinamentos passados.

Aos professores Ruy Luiz Milidiú e Edward Hermann por terem aceitado prontamente participar da Comissão examinadora.

Aos meus colegas do laboratório TecWeb: Mairon Belchior e Guilherme Szundy por todo o apoio concedido durante o desenvolvimento desta dissertação. Aos colegas do labotarório LEARN: Carlos Crestana e Eraldo Fernandes pelas inúmeras conversas e valiosas dicas. A todos os amigos e colegas da PUC-Rio.

Ao laboratório Tecgraf pela disponibilização do *cluster* e pelo apoio técnico fornecido durante a realização dos experimentos.

A todos os professores e funcionários do Departamento de Informática da PUC-Rio pelo conhecimento passado e por todo apoio e atenção.

Ao CNPq e a PUC-Rio pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

Aos meus pais, Iva e Reginaldo, pelo amor, pelos incontáveis incentivos e por todo o apoio, sem os quais seria impossível a realização deste trabalho.

À minha namorada Camila Botelho pelo seu amor e por todo carinho e compreensão.

A Deus pela força e pela coragem nos momentos difíceis.

Resumo

Nunes, Thiago Ribeiro; Schwabe, Daniel. **Construção de Extratores de Relações por Supervisionamento à Distância.** Rio de Janeiro, 2012. 112p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Um problema conhecido no processo de construção de extratores de relações semânticas supervisionados em textos em linguagem natural é a disponibilidade de uma quantidade suficiente de exemplos positivos para um conjunto amplo de relações-alvo. Este trabalho apresenta uma abordagem supervisionada a distância para construção de extratores de relações a um baixo custo combinando duas das maiores fontes de informação estruturada e não estruturada disponíveis na Web, o DBpedia e a Wikipedia. O método implementado mapeia relações da ontologia do DBpedia de volta para os textos da Wikipedia para montar um conjunto amplo de exemplos contendo mais de 100.000 sentenças descrevendo mais de 90 relações do DBpedia para os idiomas Inglês e Português. Inicialmente, são extraídas sentenças dos artigos da Wikipedia candidatas a expressar relações do DBpedia. Após isso, esses dados são pré-processados e normalizados através da filtragem de sentenças irrelevantes. Finalmente, extraem-se características dos exemplos para treinamento e avaliação de extratores de relações utilizando SVM. Os experimentos realizados nos idiomas Inglês e Português, através de linhas de base, mostram as melhorias alcançadas quando combinados diferentes tipos de características léxicas, sintáticas e semânticas. Para o idioma Inglês, o extrator construído foi treinado em um corpus constituído de 90 relações com 42.471 exemplos de treinamento, atingindo 81.08% de medida F1 em um conjunto de testes contendo 28.773 instâncias. Para Português, o extrator foi treinado em um corpus de 50 relações com 200 exemplos por relação, resultando em um valor de 81.91% de medida F1 em um conjunto de testes contendo 18.333 instâncias. Um processo de Extração de Relações (ER) é constituído de várias etapas, que vão desde o pré-processamento dos textos até o treinamento e a avaliação de

detectores de relações supervisionados. Cada etapa pode admitir a implementação de uma ou várias técnicas distintas. Portanto, além da abordagem, este trabalho apresenta, também, detalhes da arquitetura de um framework para apoiar a implementação e a realização de experimentos em um processo de ER.

Palavras-chave

Extração de Relações; DBpedia; Wikipedia; Web Semântica; Framework.

Abstract

Nunes, Thiago Ribeiro; Schwabe, Daniel (Advisor). **Building Relation Extractors through Distant Supervision.** Rio de Janeiro, 2011. 112p. MSc. Dissertation – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A well known drawback in building machine learning semantic relation detectors for natural language is the availability of a large number of qualified training instances for the target relations. This work presents an automatic approach to build multilingual semantic relation detectors through distant supervision combining the two largest resources of structured and unstructured content available on the Web, the DBpedia and the Wikipedia resources. We map the DBpedia ontology back to the Wikipedia to extract more than 100.000 training instances for more than 90 DBpedia relations for English and Portuguese without human intervention. First, we mine the Wikipedia articles to find candidate instances for relations described at DBpedia ontology. Second, we preprocess and normalize the data filtering out irrelevant instances. Finally, we use the normalized data to construct SVM detectors. The experiments performed on the English and Portuguese baselines shows that the lexical and syntactic features extracted from Wikipedia texts combined with the semantic features extracted from DBpedia can significantly improve the performance of relation detectors. For English language, the SVM detector was trained in a corpus formed by 90 DBpedia relations and 42.471 training instances, achieving 81.08% of F-Measure when applied to a test set formed by 28.773 instances. The Portuguese detector was trained with 50 DBpedia relations and 200 examples by relation, being evaluated in 81.91% of F-Measure in a test set containing 18.333 instances. A Relation Extraction (RE) process has many distinct steps that usually begins with text pre-processing and finish with the training and the evaluation of relation detectors. Therefore, this works not only presents an RE approach but also an architecture of a framework that supports the implementation and the experiments of a RE process.

Keywords

Relation Extraction; DBpedia; Wikipedia; Semantic Web; Framework

Sumário

Lista de figuras

Lista de quadros

Lista de tabelas

1
Introdução

Desde a sua criação, uma grande quantidade de conteúdo não estruturado vem sendo publicada na Web. Como exemplo, podem-se citar os *blogs*, *emails*, wikis, documentos governamentais e sites de notícias. Essa grande massa de informação é bem compreendida por seres humanos, porém, seu significado é desconhecido pelas máquinas. A falta de estrutura semântica nos documentos publicados na Web, por exemplo, restringe as operações de busca de informação, que consideram, basicamente, a presença de palavras-chave, gerando como resultado uma grande quantidade de resultados irrelevantes. Além disso, os navegadores são limitados apenas a apresentar o conteúdo de acordo com as marcações de formatação presentes nos documentos. Diante deste cenário, surge a Web Semântica [Berners-lee, Hendler e Lassila, 2001], onde um dos idealizadores é o próprio criador da Web, Tim Berners-Lee.

A Web Semântica se propõe a fornecer a estrutura necessária para descrever o significado de conteúdos publicados na Web, através de modelos formais de meta-dados processáveis por agentes de software.

Desde a publicação do artigo visionário da Web Semântica, em 2001, uma vasta quantidade de conjuntos de dados representados em RDF[1] vem sendo publicados e interligados segundo os padrões fornecidos pela iniciativa *Linked Data* [Bizer, Heath e Berners-lee, 2009]. Dentre esses conjuntos, o que possui maior destaque é o DBpedia[2] [Bizer et al., 2009], construído pelo projeto de mesmo nome, cujo objetivo é extrair informações estruturadas da Wikipedia[3] e disponibilizá-las para consultas no estilo de um banco de dados.

Apesar da grande quantidade de dados publicada nos padrões *Linked Data*, o problema de como estruturar semanticamente a grande massa de conteúdo que já se encontrava disponível na Web, e que ainda vem sendo publicada, persiste. Essa questão tem motivado avanços significativos nas áreas de Processamento de Linguagem Natural (PLN) e Extração de Informação (EI),

[1] http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[2] http://dbpedia.org/About

[3] http://www.wikipedia.org/

onde, um dos principais objetivos é descobrir a semântica por trás de conteúdos não estruturados. Nesse contexto, uma sub-área que tem ganhado cada vez mais atenção nos últimos anos é a de Extração de Relações (ER), que tem como objetivo reconhecer relações entre entidades a partir de textos em linguagem natural. Técnicas de ER são úteis em tarefas de recuperação de informação e resolução de questões, onde, a estrutura semântica de interesse são as relações presentes nos textos.

Um problema comum em ER é a cobertura dos detectores de relações que, na maioria dos casos, é limitada a um conjunto pequeno de relações. O problema de cobertura pode ser minimizado pela construção de um corpus extenso para treinar detectores através de técnicas de aprendizado de máquina, porém, o esforço humano requerido para essa tarefa é bastante alto. Algumas abordagens ([Nguyen, Matsuo and Ishizuka, 2007], [Wang et al., 2011], [Yan et al., 2009]) tem sido projetadas para superar as limitações de cobertura e o alto custo para construção manual de um corpus de tamanho considerável. Essas abordagens, normalmente, obtêm automaticamente um conjunto amplo de instâncias de relações mineradas das caixas de informação da Wikipedia, e encontram sentenças nos artigos que possuem maior probabilidade de expressar essas relações.

A abordagem proposta neste trabalho tira vantagem da qualidade dos textos da Wikipedia e das informações semânticas disponibilizadas pelo DBpedia, que descrevem aproximadamente 1.84 milhões de recursos, para construir automaticamente um conjunto de exemplos de tamanho considerável para treinar e avaliar detectores capazes de reconhecer uma grande quantidade de relações. Este trabalho apresenta um método capaz de extrair conhecimento caracterizando mais de 90 relações do DBpedia com mais de 161.000 instâncias.

Para construir o conjunto de exemplos, inicialmente são consultadas todas as triplas RDF do DBpedia, no formato <*sujeito, predicado, objeto*>, que possuem uma determinada relação-alvo como predicado. Então, obtêm-se o artigo da Wikipedia que descreve o sujeito e selecionam-se as sentenças que possuem uma referência ao objeto. Após isso, realiza-se uma filtragem das sentenças irrelevantes, mantendo apenas àquelas que possuem maior probabilidade de expressar a relação-alvo. Juntamente com as sentenças, são obtidos os tipos na ontologia do DBpedia e no esquema de classificação YAGO [Suchanek, Kasneci e Weikum, 2007] das entidades envolvidas.

A vantagem de utilizar a Wikipedia e o DBpedia como fontes de conhecimento é que ambos são multilíngue. Portanto, a mesma abordagem pode ser utilizada para construir corpora para múltiplos idiomas. Procurando explorar essa característica, a presente abordagem é aplicada também as versões em Português desses conjuntos de dados, com o objetivo de treinar detectores capazes de reconhecer relações em textos escritos na língua portuguesa.

Os experimentos realizados com a ténica SVM (*Support Vector Machines*) nos idiomas Inglês e Português, através de linhas de base, mostram as melhorias alcançadas quando combinados diferentes tipos de características léxicas, sintáticas e semânticas. Para o idioma Inglês, o extrator construído foi treinado em um corpus constituído de 90 relações com 42.471 exemplos de treinamento. Em um conjunto de testes contendo 28.773 instâncias, o extrator construído atinge 81.08% de medida F1 para o idioma Inglês. Para Português, o extrator foi treinado em um corpus de 50 relações com 200 exemplos por relação, resultando em um valor de 81.91% de medida F1 em um conjunto de testes contendo 18.333 instâncias.

A abordagem proposta neste trabalho pode ser classificada na categoria de ER baseada em “supervisionamento distante” [Mintz et al., 2009], onde bases de conhecimento externas são combinadas de forma a contornar as limitações e melhorar o desempenho de detectores de relações.

Apesar de a estratégia adotada no presente projeto seguir a idéia de supervisionamento a distância utilizando fontes externas para adquirir conhecimento a respeito das relações-alvo do aprendizado, a presente abordagem difere das demais principalmente no que diz respeito a extração de características. As características utilizadas neste projeto são extraídas das palavras das sentenças, do menor caminho entre as entidades na árvore de dependência e das informações semânticas das entidades envolvidas. Além disso, este trabalho adota estratégias de resolução de correferências específicas para artigos da Wikipedia. Essas estratégias foram avaliadas em 97.25% de precisão quando utilizadas em conjunto.

Outro problema encontrado na área de ER é a falta de uma ferramenta que apoie o desenvolvimento das várias etapas que constituem o processo. Portanto, além da abordagem, esse trabalho apresenta um *framework* para apoiar a implementação e a realização de experimentos que consideram as várias etapas de um processo de ER.

2
Fundamentos

2.1.
Extração de Relações

Uma forma de resolver o problema de falta de estrutura dos conteúdos em linguagem natural é anotar semanticamente o conteúdo, porém, dada a sua grande quantidade e diversidade, essa tarefa se torna impraticável para seres humanos. Com isso, é desejável que as máquinas realizem uma anotação automática desses dados com a estrutura de interesse ou auxiliem os seres humanos nessa tarefa.

Neste trabalho, a estrutura de interesse é a relação existente entre entidades descritas no DBpedia, como *artista-album, músico-intrumento ou pessoa-localDeNascimento.*

Extração de Relações é definida como a tarefa de reconhecimento da presença de uma relação particular entre duas ou mais entidades em textos escritos em linguagem natural [Aasman, 2008]. Como exemplo, considere a sentença

"Tom Jobim nasceu na cidade do Rio de Janeiro."

Essa sentença possui duas entidades: "Tom Jobim" e "Rio de Janeiro" Nesse caso, a aplicação de uma técnica de ER poderá extrair a relação de "nascido_em" entre o músico brasileiro Tom Jobim e a cidade Rio de Janeiro.

O problema de ER é conhecido na literatura como parte de um problema mais geral, chamado de Extração de Informações, que tem o objetivo, não só de extrair relações, como também de reconhecer entidades mencionadas e realizar análises de correferências e anáforas em texto não estruturado. Técnicas de EI e ER são úteis para realização de tarefas de recuperação de informação e resolução de questões.

Um processo de extração de relações supervisionado é normalmente abordado na literatura como um problema de classificação de textos não estruturados que contém duas ou mais menções de entidades. Com isso, as classes a serem descobertas são as possíveis relações que podem ocorrer entre

essas entidades e os exemplos positivos são trechos de textos em linguagem natural que as descrevem. Como exemplo, o trecho de texto *"Tom Jobim nasceu na cidade do Rio de Janeiro"* é um exemplo positivo que descreve a relação "nascido_em".

As abordagens para ER encontradas na literatura podem ser agrupadas de acordo com as técnicas utilizadas. Os métodos baseados em regras consistem em montar de forma manual, semi-automática ou automática um conjunto de regras linguísticas que capturam a semântica das relações de interesse. Em seguida, as regras são aplicadas a conjuntos de textos não estruturados afim de se obter novas relações. Métodos baseados em características consistem em extrair uma grande quantidade de características léxicas, sintáticas e/ou semânticas dos exemplos e utilizá-las para treinar um modelo de aprendizado de máquina a fim de predizer a relação mais provável para um exemplo não visto. Métodos que utilizam *kernel* consistem em projetar uma função *kernel* capaz de calcular a similaridade entre os exemplos, utilizada juntamente com um método de *kernel* para realizar a classificação, onde, um dos mais conhecidos e amplamente utilizados é o SVM (*Support Vector Machines*[4])

2.1.1. Extração de Relações Utilizando Kernels

Em abordagens baseadas em *kernel* para ER, são projetadas novas funções *kernel* capazes de calcular a similaridade entre instâncias de relações e realizar a classificação juntamente com métodos de *kernel.*

Funções *kernel* são utilizadas para mapear um espaço de entrada em um espaço de características implicitamente. Dessa forma, uma função *kernel* é um modelo matemático que captura a noção de similaridade entre as instâncias de um espaço de entrada, formando o nosso conhecimento a priori. A vantagem da utilização de uma função *kernel* ao invés de uma representação explícita através de vetores de características é a possibilidade de análise de um espaço muito mais amplo de características a um custo computacional razoável [Culotta e Sorensen, 2004]. Dado um conjunto de instâncias rotuladas, métodos de *kernel,* como o SVM, são capazes de determinar rótulos para novas instâncias utilizando uma função *kernel*.

O método SVM realiza uma busca pelo hiperplano que melhor separa as instâncias de duas classes distintas, maximizando a margem que separa as

[4] http://www.support-vector-machines.org/

instâncias mais próximas do hiperplano em ambos os lados. Mais detalhes sobre o método SVM podem ser encontrados em [Cristianini e Shawe-Taylor, 2000].

Formalmente, uma função kernel K é definida como um mapeamento: $K : \mathbf{X} \times \mathbf{X} \rightarrow [0, \infty]$, onde X é um espaço de vetores de instâncias que são mapeados em uma medida de similaridade. Em sua forma mais simples, K pode ser o produto interno entre vetores que caracterizam as instâncias: $K(x, y) = \sum_i \phi_i(x)\phi_i(y)$, onde $\phi_i(x)$ é uma função de característica do vetor x.

Como exemplo, considere o problema de classificação de documentos onde cada documento é representado por um vetor binário. As posições do vetor indicam a presença ou a ausência de uma determinada palavra. Nesse caso, $\phi_i(x)$= 1 se a palavra i está presente no documento x. A função kernel $K(x,y)$, então, realiza uma contagem das palavras em comum entre os documentos x e y.

A medida que as instâncias tornam-se mais estruturadas, os kernels tendem a ser mais complexos. Em ER, é comum a extração de árvores de dependência gramatical para descreverem as instâncias. Com isso, o trabalho em [Haussler, 1999] apresenta uma categoria de *kernels* mais complexos, chamados *kernels* de convolução. Esses *kernels* calculam a similaridade entre duas estruturas de árvore somando as similaridades de suas sub-árvores. Outro exemplo são *kernels* que calculam a similaridade entre strings calculando a quantidade de substrings sobrepostas.

Considerando os exemplos citados, nota-se que, a utilização de *kernels* permite que um espaço de características mais amplo seja analisado sem a necessidade de uma declaração explícita de todas as possíveis substrings ou sub-árvores como características, pois, o custo computacional para isso seria proibitivo.

2.1.2. Métricas Utilizadas

Normalmente, métodos de ER são avaliados em termos da precisão, do *recall* e da medida F1, que fornece uma noção geral do desempenho do método.

A precisão é o número de resultados corretos *rc* dividido pelo número de resultados obtidos *r:*

rc / r

O *recall* fornece a noção de cobertura do método, onde, é medida a fração de resultados corretos dentre todos os possíveis resultados corretos *tc* no conjunto*:*

rc / tc

A medida F1 considera tanto a precisão quanto o *recall* fornecendo uma média harmônica entre as duas medidas:

*2 * ((precisão * recall) / (precisão + recall))*

2.2. Web Semântica

Considere o seguinte cenário que pode ser implementado na Web Semântica:

Michael havia acabado de sofrer um leve acidente de carro e estava sentindo dores no pescoço. Seu médico de plantão sugeriu uma série de sessões de fisioterapia. Michael solicitou ao seu agente da Web Semântica para levantar algumas possibilidades.

O agente recuperou detalhes a respeito do tratamento, diretamente do agente do seu médico, e procurou a lista de fisioterapeutas que atendem pelo seu plano de saúde. O agente procurou por aqueles que atendiam num raio de 10 quilômetros do escritório ou da casa de Michael, e que possuíam uma boa reputação de prestação de serviços. Então, o agente tentou agendar os horários disponíveis no calendário de Michael. Em poucos minutos o agente retornou com duas propostas. Infelizmente, Michael não estava satisfeito com nenhuma delas. Um dos fisioterapeutas ofereceu horários para iniciar o tratamento somente em duas semanas; para o outro fisioterapeuta, Michael teria que dirigir na hora do rush. Portanto, Michael decidiu colocar restrições de tempo e horário e solicitou uma nova tentativa ao seu agente.

Alguns minutos mais tarde, o agente retornou com uma alternativa: um fisioterapeuta com boa reputação que possuía disponibilidade para iniciar o tratamento em dois dias. Contudo, havia alguns problemas menores. Alguns compromissos de trabalho menos importantes deveriam ser remarcados. O agente se ofereceu para arrumar os horários, caso essa solução fosse adotada. Além disso, o fisioterapeuta não constava na lista fornecida pelo site do plano de saúde porque cobrava um valor acima do limite máximo do plano na sua cobertura total. O agente encontrou o seu nome em uma lista independente de fisioterapeutas particulares e já havia checado que Michael

tinha direito a cobertura total, de acordo com as políticas do plano. O agente de Michael havia negociado um desconto especial com o agente do fisioterapeuta, pois o fisioterapeuta havia começado a cobrar um valor acima da média recentemente e estava interessado em atender novos pacientes.

Michael estava feliz com a recomendação porque teria que pagar apenas poucos dólares a mais [...].

[Antoniou e Van Harmelen, 2008]

Esse cenário não é possível graças à Web atual, onde os documentos são elaborados apenas para consumo humano. Mesmo em casos onde as páginas Web são geradas automaticamente a partir de um banco de dados, a estrutura do banco geralmente não está presente no conteúdo, dificultando o seu entendimento pelas máquinas.

O diferencial da Web Semântica está em trazer estrutura para as páginas Web atuais, criando um ambiente onde agentes de *software* possam navegar de página em página realizando tarefas complexas para os usuários. Dessa forma, a Web Semântica não se caracteriza como uma Web a parte e isolada, e sim, como uma extensão da Web atual, onde o conteúdo é publicado juntamente com seu significado, possibilitando uma melhor cooperação entre máquinas e humanos [Berners-lee, Hendler e Lassila, 2001]. Isso permite com que máquinas, não só sejam capazes de apresentar páginas, como também de operar de forma mais significativa sobre o conteúdo, como mostra o exemplo descrito acima.

Como resultados dos esforços para o desenvolvimento da Web Semântica foram criadas diversas tecnologias, como RDF, RDF *Schema* e OWL, que serão explicadas nas próximas seções.

2.3. RDF

Atualmente, as páginas Web, codificadas em HTML[5], fornecem os dados e a sua formatação, onde os programas apenas lêem e apresentam esses dados de forma adequada, sem nenhum conhecimento dos meta dados que os descrevem. O *Resource Description Framework* (RDF) é uma infraestrutura que permite a codificação, troca e reúso de meta dados estruturados [Miller, 1998]. A linguagem RDF é um padrão criado e mantido pela World *Wide Web*

[5] http://www.w3.org/TR/html4/

Consortium[6] (W3C) para descrição de recursos e suas relações na Web, como título, autor ou data de publicação de uma página de conteúdo. Esse padrão possibilita que agentes de *software* acessem, não só a informação e a sua formatação, como também os seus meta-dados.

Uma das idéias trazidas pelo RDF é a de utilização de um identificador único (*Uniform Resource Identifiers* URI[7]) para cada recurso na Web. Um recurso é qualquer coisa que possa ser descrita na Web, podendo ser algo diretamente recuperável, como é o caso de um documento eletrônico, ou não, como itens de uma loja de eletroeletrônicos.

RDF possui um esquema de declarações simples para descrição de recursos. Cada declaração é representada através de uma tripla na forma:

<sujeito, predicado, objeto>

O sujeito é a URI do recurso que se deseja descrever, o predicado é o identificador de uma propriedade associada ao recurso, como nome ou endereço, e o objeto é o valor dessa propriedade, podendo ser um literal ou uma outra URI. A figura 1 mostra um exemplo de descrição de uma pessoa chamada Eric Miller e identificada pela URI *http://www.w3.org/People/EM/contact#me*.

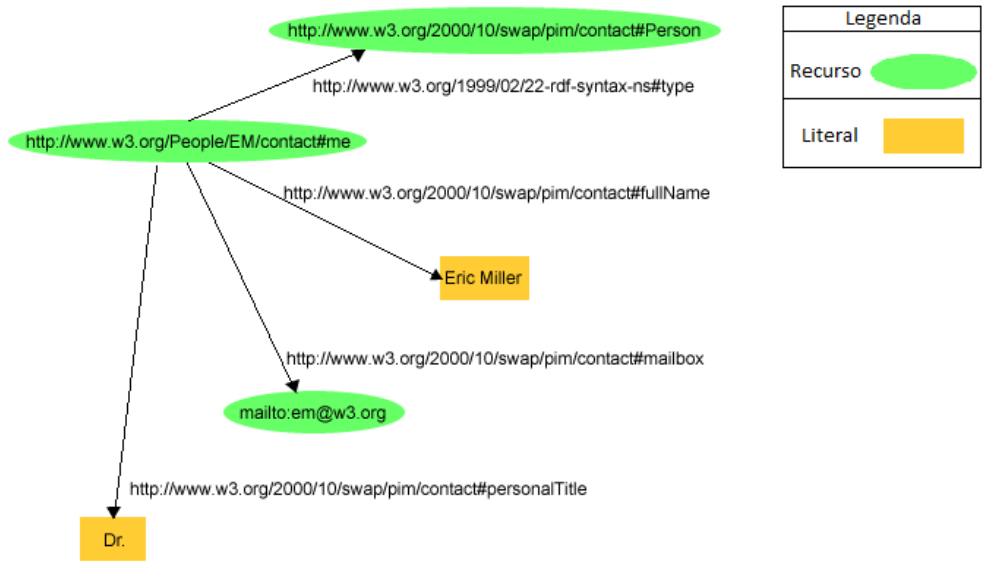

Figura 1 – RDF do recurso Eric Miller (fonte: http://www.w3.org/TR/rdf-primer/).

[6] http://www.w3.org/

[7] http://www.ietf.org/rfc/rfc3986.txt

Na figura acima, temos a representação de um grafo RDF para os seguintes fatos: O recurso identificado pela URI *http://www.w3.org/People/EM/contact#me* é do tipo *http://www.w3.org/2000/10/swap/pim/contact#Person;* possui nome completo *Eric Miller*; caixa de email "*wm@w3.org"* e atende pelo título de "*Dr."*.

O RDF utiliza XML[8] (*eXtensible Markup Language*) como uma sintaxe comum pare representação dos grafos. O quadro 1 mostra a representação em RDF/XML[9] do grafo apresentado na figura 1.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">

  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>

</rdf:RDF>
```

Quadro 1 – RDF/XML do grafo que descreve o recurso *Eric Miller* (fonte: http://www.w3.org/TR/rdf-primer/).

Além da sintaxe RDF/XML, há outras sintaxes disponíveis para codificação de um grafo RDF. Dentre as mais usadas, têm-se a N3[10], Ntriples[11] e Turtle[12].

Quando o objeto de uma tripla é a URI do sujeito de uma outra tripla, o conjunto de triplas assume mais acentuadamente a forma de um grafo conectado, como apresentado na figura 2.

[8] http://www.w3.org/standards/xml/

[9] http://www.w3.org/TR/REC-rdf-syntax/

[10] http://www.w3.org/TeamSubmission/n3/

[11] http://www.w3.org/2001/sw/RDFCore/ntriples/

[12] http://www.w3.org/TR/turtle/

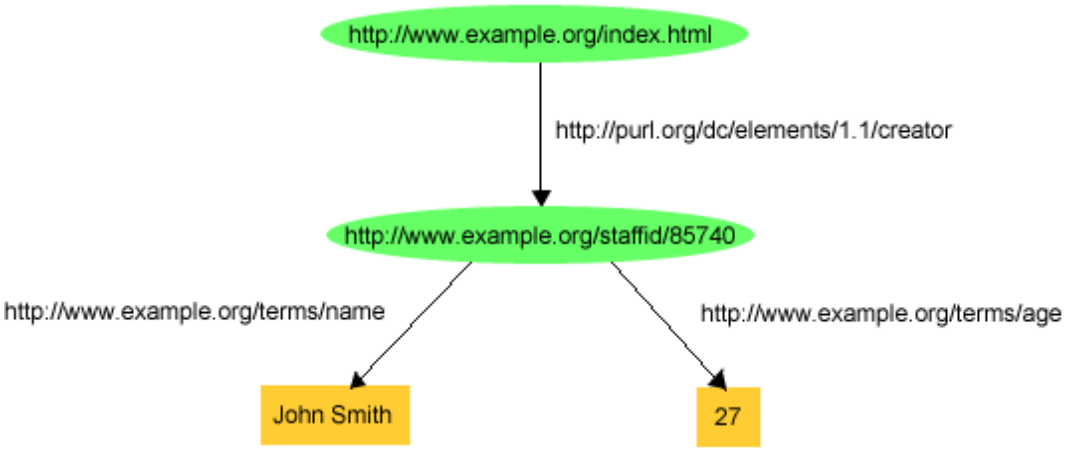

Figura 2 - Grafo RDF descrevendo o autor da página *http://www.example.org/index.html* (fonte: http://www.w3.org/TR/rdf-primer/).

Um dos problemas relacionados a atribuição de URIs a recursos na Web está em garantir a sua unicidade. Esse problema já era previsto na criação da linguagem XML, onde foi criada uma maneira de definir espaços de nomes (*namespaces*) para identificar grupos de recursos. Dessa forma, os *namespaces* são utilizados para particionar e gerenciar grupos de identificadores através de prefixos para as URIs. Logo, se definirmos o prefixo "*contact*" para o namespace "*http://www.w3.org/People/EM/contact#", podemos reescrever a URI "http://www.w3.org/People/EM/contact#me"* concatenando o prefixo do namespace seguido de ":" ao nome local do recurso, resultando na URI abreviada "*contact:me"*.

2.4. RDF Schema

Juntamente com a necessidade de se representar fatos sobre recursos através do uso de triplas RDF contendo propriedades e valores, houve a necessidade de se descrever vocabulários, contendo classes, propriedades e restrições. Esses vocabulários seriam utilizados para definir grupos de recursos e suas relações em um domínio específico.

O RDF não possui meios para definição de classes e propriedades restritas a um domínio. Para isso, foi criada uma extensão da linguagem RDF para definição de vocabulários, chamada RDF Schema[13] (RDFS). RDFS adiciona ao RDF a semântica para definição de classes em um sistema de

tipagem similar ao encontrado em linguagens orientadas a objetos. Com isso, é possível definir recursos como sendo instâncias de determinadas classes. Além disso, as classes podem ser organizadas em hierarquias através da definição de subclasses. A figura 3 mostra um exemplo de hierarquia de classes representada com RDFS.

RDFS não só permite a definição de hierarquias de classes, como também a definição de propriedades. Uma propriedade é, basicamente, definida através de restrições de domínio e contradomínio.

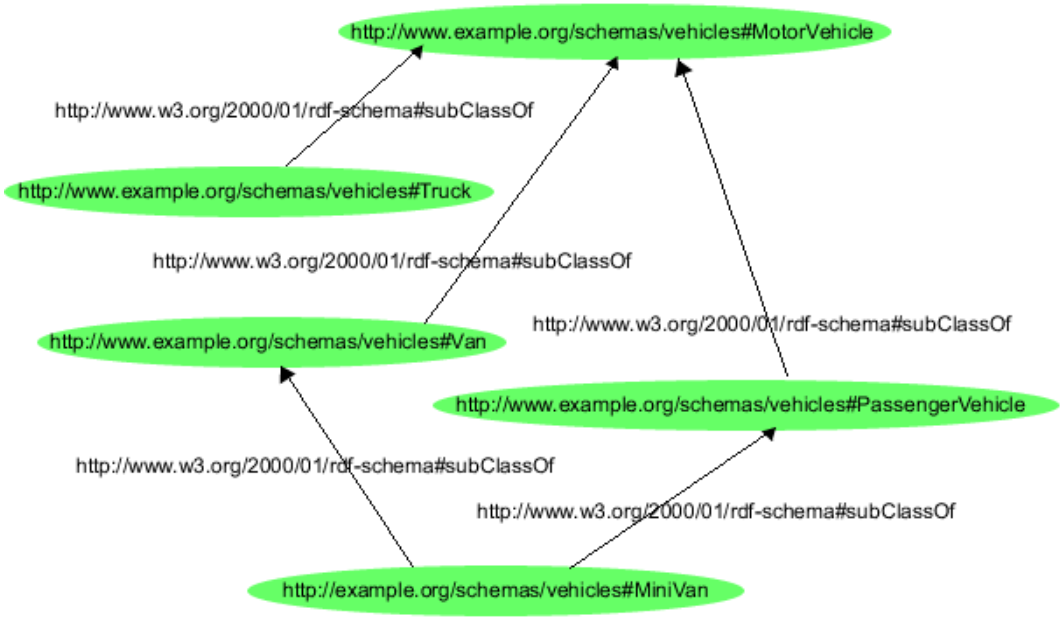

Figura 3 - Hierarquia de veículos representada com RDFS.

O domínio de uma propriedade define a classe ou o conjunto de instâncias ao qual a propriedade se aplica. A tripla <*P rdfs:domain C*> indica que P é uma propriedade que se aplica somente a instâncias da classe C, ou seja, todo sujeito de uma tripla contendo P como predicado, é uma instância da classe C.

O contradomínio de uma propriedade define a classe ou o conjunto de instâncias que ela pode assumir como valor. A tripla <*P rdfs:range C*> indica que P é uma propriedade que pode assumir como valor qualquer instância da classe C, ou seja, todo objeto de uma tripla contendo P como predicado, é uma instância da classe C.

[13] http://www.w3.org/TR/rdf-schema/

2.5. OWL

OWL[14] (*Web Ontology Language*) é uma linguagem para descrever ontologias. No contexto da Web semântica, uma das definições mais utilizadas para ontologia é a seguitne:

> "*ontologia é uma especificação formal e explícita de uma conceitualização compartilhada*" [Studer, Benjamins e Fensel, 1998].

A definição de ontologia acima traz as principais características de uma ontologia: um grupo de termos e relações do mundo real, compartilhado por grupos de pessoas ou organizações e especificado de forma explícita através de uma linguagem formal.

A linguagem OWL foi criada para dar mais expressividade a definição de vocabulários. A RDFS também permite a definição de classes e propriedades mas sua semântica é restrita a definição de hierarquias. Com isso, OWL adiciona semântica à RDFS nos seguintes aspectos:

- Permite a definição de tipos de relacionamentos, como transitividade, simetria, relação funcional e inversa;
- Possibilita a expressão de restrições de cardinalidade e presença de valores para propriedades, onde a RDFS permite apenas cardinalidade de *0..1*.;
- Definição de classes derivadas a partir de operações de conjuntos, como união, interseção e complemento;
- Definição de classes enumeradas e disjuntas;
- Definição de equivalência entre classes e propriedades;
- Definição de relação de igualdade entre URIs.

O quadro 2 mostra um exemplo de definição de interseção de classes em OWL.

[14] http://www.w3.org/TR/owl-features/

```
<owl:Class rdf:ID="WhiteWine">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor" />
      <owl:hasValue rdf:resource="#White" />
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Quadro 2 – Definição da classe "*VinhoBranco"* através de interseção.

No quadro 2, tem-se a definição dos membros da classe "*WhiteWine*" (VinhoBranco) através do uso do construto *"owl:intersectionOf",* que define um conjunto de recursos como sendo a interseção entre recursos da classe "*Wine*" (Vinho) com todos os recursos que possuem o valor "*White*" (Branco) para a propriedade *"hasColor"* (possuiCor).

2.6. Linked Open Data

Linked Open Data (LOD) [Bizer, Heath e Berners-lee, 2009] é uma iniciativa que tem o objetivo de padronizar a forma com que dados são publicados e interligados na Web Semântica, fornecendo um conjunto de princípios para este fim. Dessa forma, a LOD fornece um conjunto de práticas e diretrizes que auxiliam na migração dos dados da Web atual para a Web Semântica, onde conjuntos de dados com significado processável por agentes de *software* são predominantes. Essa iniciativa tem ganhado força nos últimos anos com a publicação de um número significativo de conjuntos de dados de escala considerável [Hausenblas e Karnstedt, 2010]. Juntos, Esses conjuntos de dados formam a Web de Dados e podem ser visualizados como uma nuvem de conjuntos interligados. A figura 4 mostra a nuvem de conjuntos mais atual.

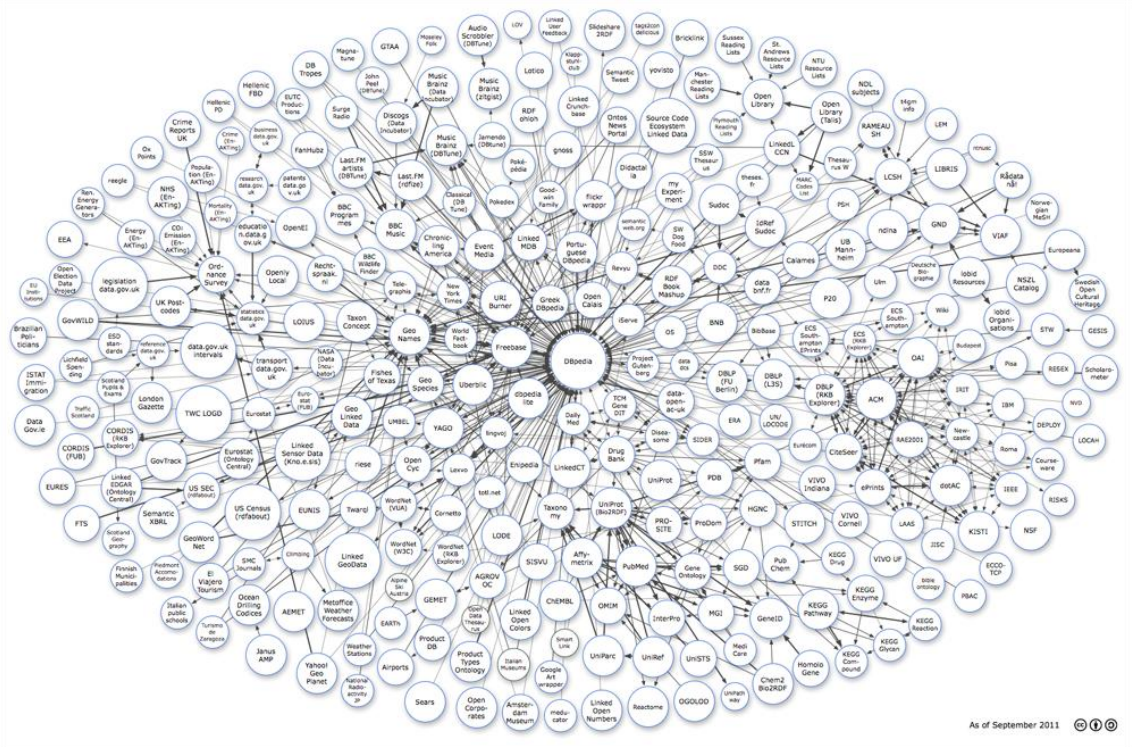

Figura 4 – nuvem da LOD em Setembro de 2011.

Os princípios fornecidos pela *Linked Open Data* são os seguintes:

- Use de URIs para identificação de qualquer tipo de recurso que possa ser descrito na Web. Esses recursos podem variar desde conceitos concretos, como pessoas, animais ou objetos, a conceitos abstratos, como amor, amizade, compaixão, etc;
- Toda URI deve ser derreferenciável, ou seja, quando acessadas através da Web via protocolo HTTP, deve ser possível obter o recurso ou alguma descrição dele.
- As informações obtidas pelo acesso a uma URI devem estar num dos formatos padrões da Web Semântica.
- Conjuntos de recursos devem estar interligados à URIs de recursos pertencentes a outros conjuntos publicados na LOD, permitindo assim, a descoberta de um número maior de informações através da exploração desses elos.

2.7. Wikipedia

A Wikipedia é um projeto internacional que tem como objetivo criar uma enciclopédia livre e multilíngue. Para isso, a Wikipedia utiliza *software* Wiki,

possibilitando uma edição colaborativa por milhares de pessoas. Wiki[15] é um tipo de sistema Web colaborativo, que permite a edição de um conjunto de documentos hipertexto por qualquer pessoa.

Wikis possuem uma linguagem de marcação própria, mais simples do que a linguagem de marcação de hipertextos HTML. Com isso, a publicação de hipertexto em Wikis se tornou mais rápida e mais simples do que em documentos HTML comuns. De fato, *Wiki* é um termo havaiano que significa *quick* (rápido em inglês). Além da facilidade na publicação de conteúdo, Wikis fornecem controle de versão, possibilitando que os usuários visualizem o histórico de revisões dos documentos. As figuras 5 e 6 mostram um exemplo de texto wiki extraído da Wikipédia, com seu resultado em HTML.

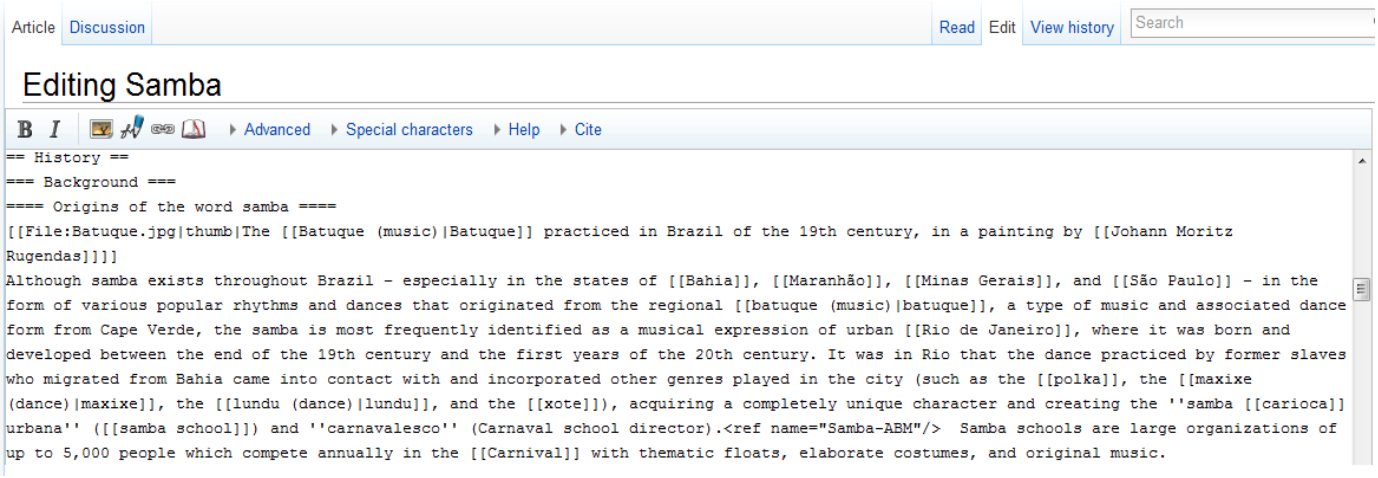

Figura 5 – Parte do texto wiki do recurso Samba na Wikipédia.

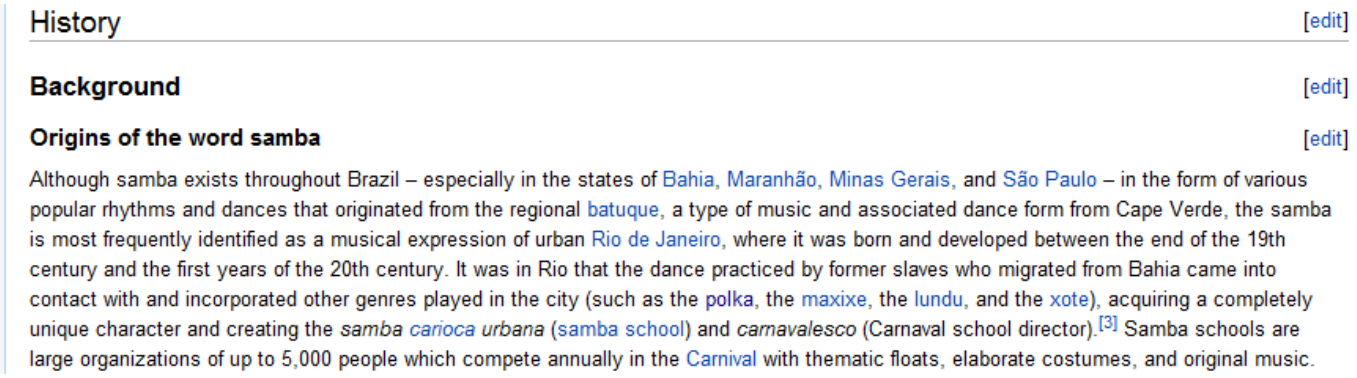

Figura 6 – Resultado em HTML do texto wiki apresentado na figura 5.

[15] http://en.wikipedia.org/wiki/History_of_wikis

Desde a sua fundação, em 2001, o número de artigos publicados na Wikipédia vem aumentando significativamente, atualmente totalizando mais de 11 milhões de artigos em 269 línguas [Zachte, 2012], cobrindo diversos temas, como Artes, História, Geografia, Ciências, Esportes e Jogos Eletrônicos. À medida que a Wikipédia vem se tornando uma fonte cada vez maior de conhecimento humano, ela também se torna uma fonte valiosa para mineração de informação, fazendo com que o termo *Wikipedia-Mining* (Mineração da Wikipedia) se torne cada vez mais frequente na literatura.

Além da abrangência de temas e da grande quantidade de artigos, algumas características contribuem para o aumento do interesse na Wikipédia como corpus para mineração de informação. De acordo com estudos estatísticos realizados em 2005, a Wikipédia é apontada como sendo tão precisa quanto a Enciclopédia Britânica na cobertura de temas científicos [Giles, 2005]. Outras características interessantes são a densa estrutura de links entre artigos, a desambiguação de entidades e as sentenças bem estruturadas [Nakayama et al., 2008].

2.8. DBpedia

Com o objetivo de extrair informações estruturadas da Wikipédia e disponibilizá-las na Web para consultas no estilo de um banco de dados, foi montado o projeto DBpedia. Atualmente, o DBpedia descreve mais de 3,64 milhões de recursos, onde aproximadamente 50% destes são classificados em uma ontologia consistente.

Dos recursos classificados em sua ontologia, o DBpedia possui 416.000 pessoas, 526.000 lugares, 106.000 álbuns musicais, 60.000 filmes, 17.500 jogos eletrônicos, 169.000 organizações, 183.000 espécies e 5.400 doenças [Bizer, 2011].

O projeto DBpedia apresenta um *framework* para converter o conteúdo da Wikipédia em uma base de conhecimento. Esse *framework* mapeia as caixas de informação dos artigos em uma ontologia e mantém a base atualizada através dos *feeds* de alteração. A figura 7 mostra a caixa de informação do artista musical Tom Jobim.

```
{{Infobox musical artist
| name                = Tom Jobim
| image                 = Tom cigar.jpg
| birth_name          = Antônio Carlos Brasileiro de Almeida Jobim
| birth_date                = {{birth date|1927|1|25|mf=y}}
| death_date                = {{death date and age|1994|12|8|1927|1|25}}
| origin              = [[Rio de Janeiro (state)|Rio de Janeiro]], Brazil
| instrument          = Piano, guitar, Flute
| genre               = [[Bossa Nova]]<br />[[Música Popular Brasileira|MPB]]
| occupation          = Musician, composer, songwriter, singer.
| years_active        = 1956–1994
```

Figura 7 – Caixa de informação Tom Jobim.

Foram utilizadas duas estratégias para conversão das caixas de informação em triplas RDF: Extração Genérica e Extração Baseada em Mapeamento.

2.8.1. Extração Genérica

O algoritmo de extração genérica processa todas as caixas de informação presentes nos artigos e cria triplas diretamente. O sujeito da tripla é a URI do DBpedia equivalente a entidade do título do artigo da Wikipédia, que consiste do *namespace* **http://dbpedia.org/resource/** concatenado ao rótulo do artigo, tendo os espaços substituídos por sublinhado. Dessa forma, a URI equivalente ao artigo Tom Jobim é:

http://dbpedia.org/resource/Tom_Jobim.

O predicado da tripla é mapeado de forma similar, concatenando o *namespace* **http://dbpedia.org/property/** ao nome do atributo. Logo, a URI do atributo *name,* presente na figura 7, ficaria:

http://dbpedia.org/property/name.

O objeto da tripla pode ser mapeado em uma outra URI ou em um literal. Caso o valor do atributo seja um elo para um outro artigo, o valor é mapeado na URI do artigo referenciado. Caso contrário, o valor é mapeado em um literal, onde é aplicada uma heurística para detectar o seu tipo. No exemplo do atributo *name*, o objeto é um literal do tipo *string*: "*Tom Jobim".* A figura 8 mostra o

resultado da tripla equivalente ao atributo *name* da caixa de informação “Tom_Jobim” em formato N3.

```
1  http://dbpedia.org/resource/Tom_Jobim
2      http://dbpedia.org/property/name "Tom_Jobim"
3      .
```

Figura 8 – Sujeito, predicado e objeto mapeados a partir do atributo *name* da caixa de informação “Tom_Jobim”.

Uma vantagem dessa estratégia é a possibilidade de cobertura completa de todas as caixas de informação e seus respectivos atributos. Porém, nem sempre atributos com mesmo significado são escritos da mesma forma. Por exemplo, o nome do atributo *origin,* presente na caixa de informação “Tom_Jobim” (figura 7), pode ser encontrado como *place_of_origin* ou *origin_place* em outras caixas de informação, dificultando consultas a respeito do local de origem de um artista. Outro problema é a alta taxa de erro da heurística que determina os tipos dos literais [Bizer et al., 2009]. A fim de resolver esses problemas, foi elaborada a estratégia de extração baseada em mapeamento, explicada na próxima subseção.

2.8.2. Extração Baseada em Mapeamento

Esse método consiste no mapeamento manual das caixas de informação e atributos mais frequentes em classes e propriedades de uma ontologia. Essa ontologia foi montada a partir dos 350 padrões de caixas de informação mais utilizados, resultando em 170 classes, organizadas de forma hierárquica, e 720 propriedades.

O problema com essa abordagem é a quantidade de padrões de caixas de informação utilizados, que é de apenas 350 padrões. Com isso, são geradas descrições para, aproximadamente, apenas 58% das entidades cobertas pelo método de extração genérica. Uma possível solução, discutida em [Bizer et al., 2009], é de integrar a ontologia de volta a Wikipedia, permitindo que as pessoas modifiquem tanto o conteúdo dos artigos, quanto o mapeamento dos padrões das caixas de informação na ontologia, evoluindo-a de forma colaborativa.

2.9. YAGO

YAGO é uma ontologia extensa, de alta precisão e cobertura, derivada automaticamente da Wikipedia e do Wordnet[16] [Suchanek, Kasneci e Weikum, 2007]. YAGO interliga páginas de categorias e fatos extraídos das caixas de informação da Wikipedia com elementos da base léxica para o idioma Inglês Wordnet. O método automático implementado para derivação da YAGO foi avaliado em 95% de precisão. Dessa forma, essa ontologia tira vantagem da vasta quantidade de indivíduos descritos na Wikipedia e da alta precisão das relações taxonômicas do Wordnet.

O sistema de classificação dos indivíduos da ontologia YAGO é derivado das categorias mais específicas do sistema de categorias da Wikipedia. Essas categorias são automaticamente relacionadas aos termos do Wordnet através de *subClassOf,* que é equivalente ao construto para relacionamento de subclasses fornecido pelo RDFS. O algoritmo para unifação da Wikipedia e do Wordnet é detalhado em [Suchanek, Kasneci e Weikum, 2007].

Na área de extração de relações, os tipos fornecidos pela ontologia YAGO podem ser utilizados como características semânticas das entidades relacionadas, como mostra o trabalho em [Wang et al., 2011]. Uma vez que os recursos do DBpedia são relacionados aos tipos YAGO através da propriedade *rdf:type*, é possível recuperar os tipos de uma entidade através de uma consulta SPARQL ao *endpoint*.

[16] http://wordnet.princeton.edu/

3
Abordagem para Extração de Relações

Para construir detectores de relações que possuam uma cobertura razoável, no que diz respeito a quantidade de relações, é necessária uma etapa de coleta e normalização de um conjunto de exemplos de tamanho considerável. Isso, normalmente, requer uso estensivo de trabalho manual. Procurando amenizar esse problema, a presente abordagem tem o objetivo de coletar instâncias de relações do DBpedia e sentenças que descrevem essas relações da Wikipedia, combinando dois dos maiores conjuntos de dados estruturados e não estruturados da Web.

3.1.
Geração do Conjunto de Exemplos

O primeiro passo é coletar instâncias de relações entre entidades, através de consultas ao DBpedia SPARQL endpoint[17], para obter triplas no formato <sujeito, predicado, objeto>, onde, o predicado é a relação desejada.. Após isso, obtem-se os artigos da Wikipedia que descrevem os sujeitos das triplas. Os artigos, por sua vez, são pré-processados para remover estruturas irrelevantes e marcações Wiki desnecessárias, como *templates*, tabelas e marcações de cabeçalho, por exemplo. São mantidas apenas estruturas que auxiliem o processo de alguma forma. No caso da Wikipedia, são mantidas as âncoras e as aspas. As âncoras são necessárias para identificar referências e menções ao objeto da relação. As aspas são úteis para encontrar menções ao sujeito descrito pelo artigo. Além da remoção de partes irrelevantes, a etapa de pré-processamento divide o artigo em sentenças.

Após a etapa de pré-processamento, inicia-se um processo de filtragem das sentenças irrelevantes do artigo. A heurística aplicada consiste em manter apenas a primeira sentença do artigo que contém menções para ambas as entidades sujeito e objeto da relação. Essa heurística é baseada na hipótese de que a primeira sentença que menciona as duas entidades tem maior chance de

[17] http://dbpedia.org/sparql

expressar a relação entre elas. Como exemplo, considere a relação do DBpedia, em notação N3, que representa o fato de o artista Eric Clapton ter nascido na localidade de Ripley, localizada na região de Surrey, Inglaterra:

```
@prefix dbpedia_ont: <http://dbpedia.org/ontology/>.
@prefix dbpedia_resource: <http://dbpedia.org/resource/>

dbpedia_resource:Eric_Clapton
   dbpedia_ont:birthPlace  dbpedia_resource:Ripley,_Surrey .
```

Quadro 3 – Relação "*birthPlace*" entre Eric Clapton e Ripley, Surrey em N3.

A primeira sentença que menciona Eric Clapton e possui uma referência para o artigo que descreve a localidade de Ripley está na seção "*Early Life*":

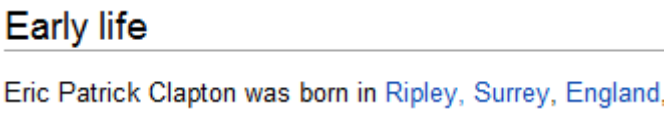

Figura 9 – Seção *Early Life* do artigo do Eric Clapton.

Essa sentença descreve com precisão a relação de "nascido-em" entre as entidades Eric Clapton e localidade de Ripley. Com o objetivo de obter mais evidências sobre as relações, são recuperados os tipos das entidades, nos sistemas de classificação da ontologia do DBpedia e YAGO, através de consultas ao *endpoint* do DBpedia. No exemplo acima, alguns tipos para as entidades são *dbpedia_ont:MusicalArtist* e *yago:EnglishRockMusicians* para Eric Clapton, e *dbpedia_ont:PopulatedPlace* e *yago:CivilParishesInSurrey* para a localidade de Ripley.

Uma vez que uma quantidade razoável de relações e exemplos por relação é registrada em um banco de dados, torna-se possível a aplicação de extratores de características para gerar o conjunto de dados que será utilizado para treinar e avaliar os classificadores.

Em resumo, o algoritmo para obtenção de um exemplo positivo para uma determinada relação R utilizando a Wikipedia como corpus e o DBpedia como modelo de dados é listado a seguir:

1. Consulta-se o DBpedia para obtenção de todos os pares de sujeitos e objetos de triplas que possuem R como predicado.
2. Para cada par <sujeito, objeto>, obtém-se o artigo do sujeito na Wikipédia.

2.1. Realiza-se um pré-processamento do texto wiki do artigo para remover todos os *templates* de caixas de informação, tabelas e listas, pois estas, não caracterizam sentenças.
2.2. Obtêm-se a primeira sentença no artigo que possui um elo para o artigo do objeto e uma menção para o sujeito.
2.3. Armazena-se a sentença, em formato de texto wiki, em um banco de dados local como exemplo positivo para a relação R.
2.4. Armazenam-se as classes do sujeito e do objeto, obtidas através de consultas ao DBpedia SPARQL *endpoint*.

O conjunto inicial de exemplos obtido para Inglês, sem a restrição de que as sentenças devem possuir uma menção ao sujeito, é composto de 161.829 exemplos para 101 relações do DBpedia. Quando adicionamos a condição de que as sentenças devem possuir uma menção ao sujeito da relação, a quantidade de exemplos cai para 125.351 exemplos para 92 relações. O Apêndice A mostra a tabela de frequências de exemplos positivos por relação.

Para o idioma Português, o conjunto montado é constituído de 147.589 exemplos para 55 relações. Devido as dimensões menores tanto da Wikipedia como do DBpedia para Português, as sentenças foram mineradas considerando apenas a condição de possuir uma menção ao objeto da relação. O Apêndice B mostra a tabela com as relações e suas respectivas quantidades de exemplos positivos para Português.

3.2. Estratégias de Análise de Correferências

Um desafio comum em NLP consiste em como identificar o termo ou o conjunto de termos que mencionam alguma entidade no texto. O estado-da-arte nessa tarefa é atingido no trabalho em [Fernandes, Dos Santos e Milidiú, 2012] na *CoNLL-2012 Shared Task*[18]. No presente trabalho, o problema se restringe a como identificar na sentença os termos que mencionam o sujeito da relação. Para isso, foram utilizadas três heurísticas baseadas no algoritmo para extração de menções ao sujeito da relação apresentado em [Nguyen, Matsuo e Ishizuka, 2007]. As heurísticas são: análise de termos entre aspas, pronomes mais frequentes e análise do sujeito das sentenças.

A pimeira heurística aplicada consiste em extrair os conjuntos de termos entre aspas e verificar se todos os *tokens* presentes no título do artigo estão

[18] http://conll.cemantix.org/2012/

incuídos nesse conjunto. Como exemplo, considere o artigo que descreve o artista musical Eric Clapton. O título do artigo é “Eric Clapton”. Como as aspas são preservadas na fase de pré-processamento, as marcações em negrito da Wikipedia também são. Portanto, na primeira sentença do artigo tem-se o seguinte texto wiki pré-processado:

'''Eric Patrick Clapton''', [[Order of the British Empire|CBE]], (born 1945) is an English guitarist and singer-songwriter.

Quadro 4 – Texto wiki da primeira sentença do artigo que descreve o artista Eric Clapton.

Como resultado da aplicação da primeira heurística, extrai-se “Eric Patrick Clapton” como uma menção da entidade “Eric Clapton” porque todos os *tokens* do título, “Eric” e “Clapton”, estão incluídos no conjunto entre aspas “Eric Patrick Clapton”.

No caso de a primeira heurística não identificar a menção ao sujeito, aplica-se a segunda heurística, que considera os pronomes mais frequentes. Trabalhos anteriores ([Nakayama et al., 2008]) mostram que na Wikipedia, o pronome mais frequente no texto do artigo é uma menção para a entidade sendo descrita, na maioria dos casos. No artigo do Eric Clapton, o pronome mais frequente é “*he*”, logo, extrai-se “*he*” como sendo uma co-referência para Eric Clapton.

A terceira heurística aplicada consiste em utilizar um analisador de dependências. Após a extração da árvore de dependência gramatical, seleciona-se o sujeito da sentença e verifica-se se ele está incluído no título do artigo. Como exemplo, considere a sentença no quadro 5.

Clapton was influenced by the [[Blues music|blues]] from an early age, and practised long hours to learn the [[chord (music)|chords]] of blues music by playing along to the records.

Quadro 5 – Exemplo de sentença onde o sujeito referencia o artista Eric Clapton.

Após a extração da árvore de dependências, o termo “Clapton” é identificado como o sujeito da passiva da cláusula “*Clapton was influenced by the [[Blues music|blues]]…*”. Então, como o termo “Clapton” está incluído no título, ele é reconhecido como uma menção a Eric Clapton.

A avaliação das estratégias de análise de correferências adotadas nesse trabalho foi realizada manualmente em uma amostra arbitrária de 400 sentenças

do conjunto de 161.829 sentenças da Wikipedia, resultando em um valor de 97,25% de precisão.

Os erros estão mais concentrados na estratégia de obtenção do pronome mais frequente, onde, a entidade do título nem sempre é mencionada pelo pronome mais frequente encontrado no artigo. Esse problema foi previamente confirmado em [Nakayama et. al, 2008]. Como exemplo, o pronome mais frequente no artigo do Eric Clapton é "*he*", porém, na sentença "*His third European tour with Steve Winwood began on {{j|18 May}} and ended {{j|13 June}}, including [[Tom Norris (musician)|Tom Norris]] as opening act.*" Eric Clapton é mencionado pelo pronome "*his*".

Uma vez que as respectivas menções das entidades são identificadas, podem-se extrair características sintáticas do caminho que liga essas duas entidades na árvore de dependência. Caso a menção ao sujeito da relação não possa ser identificada pelas três heurísticas apresentadas nessa seção, a sentença é considerada um ruído e é descartada.

3.3. Extração de Características

Na etapa de extração de características, são extraídas informações das sentenças e das entidades envolvidas que capturem a essência das relações as quais deseja-se aprender. A presente abordagem combina características léxicas e sintáticas, explorando a qualidade comprovada dos textos da Wikipedia [Giles, 2005], com características semânticas extraídas através de consultas ao DBpedia.

As caraterísticas léxicas são extraídas a partir de cada *token* da sentença, seguindo uma abordagem de saco-de-palavras unigrama. Como exemplo, considere a sentença

"Paulo Coelho was born in Rio de Janeiro, Brazil".

A tabela 1 mostra os pares de atributos e valores gerados como características léxicas:

Atributos	Valores
Paulo	1
Coelho	1
Was	1

Born	1
In	1
Rio	1
De	1
Janeiro	1

Tabela 1 – Dicionário de características léxicas obtidas da sentença *"Paulo Coelho was born in Rio de Janeiro, Brazil".*

Um problema comum que surge quando utilizamos *tokens* como unidades léxicas, é a necessidade de redução da variação de *tokens* relacionados. Como exemplo, os *tokens* "observe", "observador", "observatório" possuem significado similar e é desejável que sejam tratados como um único *token*, "observ". Para reduzir a variação dos *tokens* foi utilizado o algoritmo para extração de radicais de Porter[19].

As características sintáticas das sentenças são extraídas do menor caminho entre as entidades na árvore de dependências gramaticais. Para esse propósito foi utilizado o analisador de dependências de Stanford[20]. A figura 10 mostra um exemplo de árvore gramatical obtida da sentença "*Paulo Coelho was born in Rio de Janeiro, Brazil*".

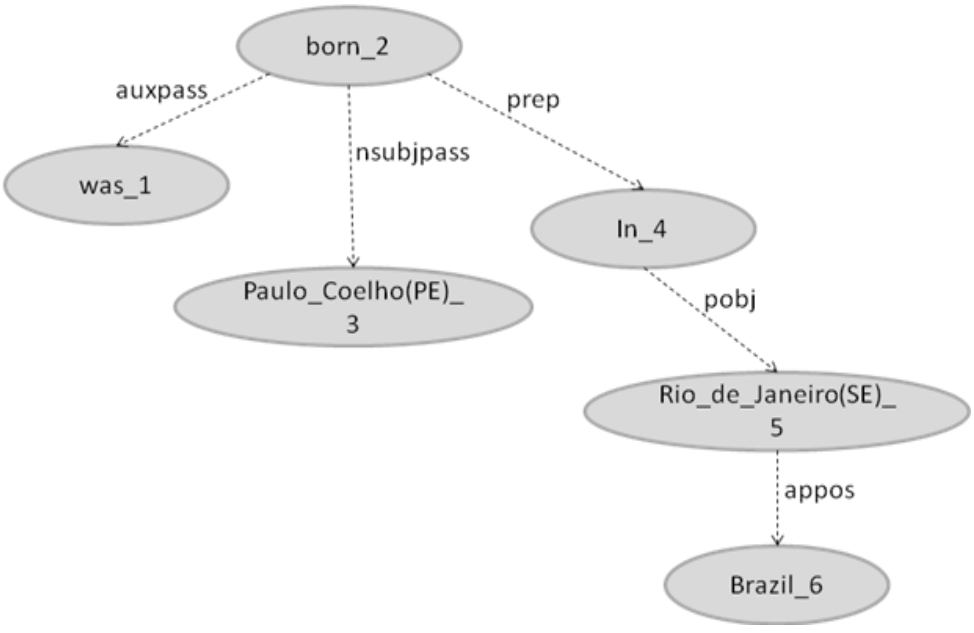

Figura 10 – Árvore de dependências da sentença *"Paulo Coelho was born in Rio de Janeiro, Brazil".*

[19] http://tartarus.org/~martin/PorterStemmer/

[20] http://nlp.stanford.edu/software/lex-parser.shtml

A árvore de dependências apresentada na figura 10 representa a relação de local de nascimento que ocorre entre o escritor brasileiro Paulo Coelho (Paulo_Coelho(PE)_3) e a cidade Rio de Janeiro (Rio_de_Janeiro(SE)_5). As arestas são as relações gramaticais entre os termos governante e dependente, seguindo o modelo de dependências de Stanford [De Marnee e Manning, 2011]. A partir dessa árvore, é extraído apenas o menor caminho não direcionado entre as duas entidades. Algumas abordagens em ER confirmam a hipótese de que o caminho mais curto tem maior chance de fornecer indícios que descrevem a relação-alvo [Wang et al., 2011], [Bunescu and Mooney, 2005], [Yan et al., 2009]. A figura 11 apresenta o caminho mais curto entre as entidades Paulo Coelho e Rio de Janeiro.

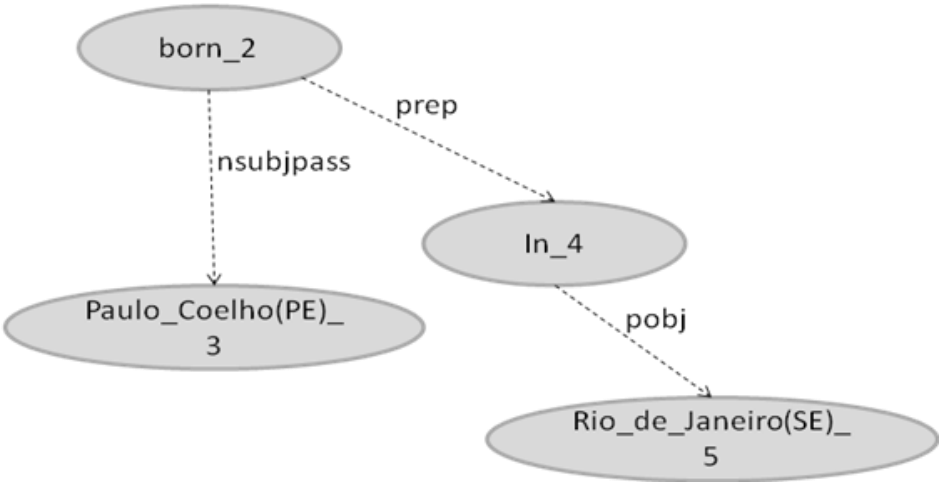

Figura 11 – Menor caminho entre as entidades Paulo Coelho e Rio de Janeiro.

O caminho apresentado na figura 11 captura a noção exata de que a cidade Rio de Janeiro é o local de nascimento do escritor Paulo Coelho. As características extraídas são os termos do menor caminho com suas direções na árvore de dependência. A tabela 2 mostra o conjunto de características extraídas do exemplo, onde, “PE” (*Principal Entity*) é o rótulo para o sujeito “Paulo Coelho” e “SE” (*Secondary Entity*) é o rótulo para o objeto da relação “Rio de Janeiro”.

Atributos	Valores
PE<-	1
born->	1
in->	1
SE<-	1

Tabela 2 – Dicionário de características sintáticas obtidas do exemplo da figura 11

As características semânticas são as classes, tanto do sujeito quanto do objeto da relação, fornecidas pelos esquemas de classificação YAGO e da ontologia DBpedia. As classes são obtidas através de consultas ao *SPARQL endpoint* do DBpedia. A tabela 3 mostra alguns exemplos de características semânticas obtidas para as entidades Paulo Coelho e Rio de Janeiro.

Atributos	Valores
yago:BrazilianWriters_PE	1
yago:WesternMystics_PE	1
yago:LivingPeople_PE	1
dbpedia_ont:Aritst_PE	1
dbpedia_ont:Writer_PE	1
yago:PopulatedPlacesEstablishedIn1565_SE	1
yago:PortCitiesAndTownsInBrazil_SE	1
yago:Locations_SE	1
dbpedia_ont:PopulatedPlace_SE	1
dbpedia_ont:Settlement_PE	1

Tabela 3 – Características semânticas para as entidades Paulo Coelho (terminação "_PE") e Rio de Janeiro (terminação "_SE").

3.4. Experimentos

O presente projeto utiliza uma estratégia de linha de base para avaliar como a performance de detectores de relações, construídos através da abordagem apresentada neste trabalho, pode ser melhorada de forma a atingir mais de 80% de medida F1. Essa linha de base é composta por um conjunto de 90 relações com 200 instâncias por relação, onde, são consideradas apenas características léxicas, atingindo 68.8% de F1. Foi utilizada a técnica SVM de aprendizado de máquinas, implementada pela biblioteca LibLINEAR[21]. Os experimentos mostram os resultados obtidos considerando cada tipo de característica separadamente, e as melhorias de performance alcançadas quando diferentes tipos de características léxicas, sintáticas e semânticas são combinadas.

[21] http://www.csie.ntu.edu.tw/~cjlin/liblinear/

A estratégia adotada para treinar o classificador SVM consiste em tomar as instâncias de uma relação como exemplos positivos e as instâncias das demais relações como exemplos negativos, seguindo uma abordagem um-contra-todos. Foram realizadas duas avaliações distintas: a primeira avaliação foi realizada através de validação cruzada com 10 partes e demonstra o impacto dos diferentes tipos de características léxicas, sintáticas e semânticas no desempenho do extrator de relações; A segunda avaliação fornece uma noção mais geral do desempenho do extrator utilizando um conjunto de testes contendo 28.773 instâncias. As tabelas 4 e 5 apresentam os resultados obtidos das avaliações.

Caracetrística/Medida	Precisão	Recall	F1
Léxico (L)	69%	69%	68.8%
Menor Caminho (MC)	63.5%	64.1%	63.6%
Tipos (T)	81%	80.8%	81%
L + MC	72.8%	73.1%	72.8%
L + T	85.5%	85.6%	85.5%
L+MC+T	87%	87%	86.9%

Tabela 4 – Resultados obtidos pela avaliação dos classificadores com validação cruzada 10 partes.

Os resultados mostram que a característica sintática de menor caminho não apresenta bons resultados quando utilizada separadamente, atingindo o valor mais baixo de *F1* com 63.3%. Considerando apenas os tipos das entidades, o valor sobe consideravelmente para 81% que ainda pode ser melhorado pela combinação dos outros tipos de características. O melhor resultado, então, ocorre quando combinamos características léxicas, sintáticas e semânticas, atingindo 86.9% de *F1*.

O trabalho em [Wang et al., 2011] detêm o estado da arte na área atingindo 81.18% de F1, através de validação cruzada 5 partes, em um corpus formado por 100 relações da Wikipedia com 200 instâncias para cada relação. Embora uma comparação mais precisa seja necessária, a abordagem proposta leva a construção de classificadores com uma acurácia equivalente, considerando a quantidade de relações e de exemplos por relação. Como a abordagem do presente trabalho tira vantagem do esforço realizado pela comunidade DBpedia para estabelecimento de mapeamentos, ela é mais simples e mais rápida de ser implementada.

Além da validação cruzada, foi montado um corpus de treinamento contendo 42.471 instâncias, onde, a distribuição da quantidade de exemplos por relação se dá da seguinte forma: para relações que possuem mais de 500 exemplos positivos, selecionamos 500 exemplos para treinamento e o restante para testes. Caso o número de instâncias seja abaixo de 500, selecionamos 80% dos exemplos para treinamento e 20% para testes. O conjunto de testes é formado por 28.773 sentenças da Wikipedia para as 90 relações, resultando em 82.1% de precisão, 80.1% de *recall* e 81.08% de F1.

A tabela 5 mostra o estado da arte dos valores de *F1* atingidos em três corpora distintos. O primeiro corpus foi construído utilizando a nossa abordagem, formado por 90 relações da ontologia do DBpedia com 200 instâncias por relação. Nesse corpus foi atingido o maior valor de 86.9%. O segundo corpus, com 81.18%, foi o construído a partir da Wikipedia e do sistema de classificação YAGO, em [Wang et al., 2011]. O terceiro corpus foi montado no presente trabalho, onde o extrator treinado atingiu 81.08% de *F1.* Finalmente, o quarto estado da arte foi atingido em [Zhou e Zhu., 2011], no corpus da ACE-2004, com um valor de 75.8%.

Corpus	Composição	Estratégia de Avaliação	F1
Wikipedia e DBpedia (presente trabalho)	90 relações e 200 exemplos por relação	Validação cruzada 10 partes	86.9%
Wikipedia e Yago [Wang et al., 2011]	100 relações e 200 exemplos por relação	Validação cruzada 5 partes	81.18%
Conjunto de testes da Wikipedia (presente trabalho)	42.471 instâncias de treinamento e 28.773 sentenças para testes	Conjunto de teste	81.08%
ACE-2004 [Zhou e Zhu, 2011]	30 relações, onde, 7 são gerais e 23 específicas	Conjunto de teste	75.8%

Tabela 5 – Resultados que representam o estado da arte dos valores de *F1* em quatro corpora distintos.

Embora seja de grande importância a realização de uma comparação mais precisa em algum corpus disponibilizado por uma iniciativa de avaliação conjunta, como a ACE, o foco deste trabalho está em demonstrar como a abordagem proposta pode ser útil na construção de reconhecedores de relações com uma boa cobertura na quantidade de relações.

3.4.1. Avaliação na Wikipedia e no DBpedia em Português

O trabalho em [Fernandes, 2012] apresenta um corpus construído a partir do *site* de notícias globo.com de tamanho satisfatório para aplicação de métodos de aprendizado de máquina para o problema de extração de citações. Contudo, a área de ER para Português começou a ser explorada somente recentemente e, por esse motivo, os métodos de ER para Português ainda carecem de um corpus com uma quantidade suficiente de relações e de exemplos por relação para realização de experimentos utilizando técnicas de aprendizado de máquina. As abordagens encontradas na literatura até o momento da realização deste trabalho são, em sua maioria, baseadas em regras [Santos, Mamede e Baptista, 2010] [Garcia e Pablo, 2011]. Observando esses fatos, selecionamos a Wikipedia e o DBpedia em Português para avaliar a presente abordagem em outro idioma.

O objetivo desse experimento é explorar a característica multilíngue, tanto da Wikipedia como do DBpedia, para fornecer uma abordagem para mineração de exemplos e construção de detectores de relações que possa ser aplicada em diversos idiomas.

Assim como no experimento para o idioma Inglês, utilizamos uma linha de base para avaliar a performance dos detectores em português. A linha de base é composta por um conjunto de 50 relações com 200 instâncias por relação, onde, novamente são consideradas apenas características léxicas, atingindo um valor de 76.5% de F1 com a técnica SVM. Análogamente as avaliações realizadas para o idioma Inglês, as avaliações para Português consistem de duas formas: validação cruzada 10 partes e um conjunto de testes formado por 18.333 instâncias. A tabela 6 apresenta os resultados obtidos através de validação cruzada no corpus em Português.

Diferentemente da aplicação no idioma Inglês, não foi possível utilizar a abordagem de caminho mínimo entre as entidades no grafo de dependência.

Isso se deve ao fato de que o analisador utilizado para Português, DepPattern, fornece apenas uma análise parcial de dependências, resultando em grafos desconexos na maioria dos casos. Portanto, utilizamos os termos governantes da sentença como características sintáticas, assim como foi feito em [Garcia e Pablo, 2011]. Experimentos futuros serão realizados com o analisador de dependências para Português do estado-da-arte, apresentado em [Fernandes e Milidiú, 2012].

Caracetrística/Medida	Precisão	Recall	F1
Léxico (L)	76.4%	76.8%	76.5%
Termos Governantes (TG)	21%	22.6%	21.1%
Tipos (T)	83.5%	81.9%	81.3%
L + TG	78.3%	78.5%	78.3%
L + T	89%	89%	88.98%
L+TG+T	89.73%	89.75%	89.73%

Tabela 6 - Resultados obtidos pela avaliação dos classificadores em Português.

Análogamente, o resultado mais baixo ocorre quando consideramos apenas as características sintáticas de termos governantes das dependências, com 21.1% de F1. O melhor resultado ocorre quando são combinadas as características léxicas, os termos governantes e os tipos das entidades, atingindo 89.73% de F1. Nota-se que a utilização dos termos governantes como características não influenciaram tanto nos resultados, houve pouco menos de 1% de melhora na F1 do conjunto L+T para o conjunto L+TG+T. Uma das razões pode ser a pouca informação fornecida pelos termos governantes a respeito da relação se comparada a informação obtida do menor caminho entre as entidades na árvore de dependências.

Além da validação cruzada, o extrator gerado foi avaliado em um conjunto de testes contento 18.333 sentenças da Wikipedia para as 50 relações, resultando em 82.70% de precisão, 81.90% de *recall* e 81.91% de F1.

4
Framework para Extração de Relações

Esse capítulo apresenta aspectos relacionados ao desenvolvimento do framework para ER proposto. As principais informações contidas nesse capítulo são:

1. Motivações;
2. Abordagem de mineração de exemplos com a Wikipedia e o DBpedia;
3. Especificação e detalhes da arquitetura modular do *framework*;
4. Decisões de implementação;
5. Avaliações realizadas, tanto no *framework* quanto nos classificadores gerados.

4.1.
Processo de Extração de Relações

Um processo de extração de relações supervisionado é, normalmente, constituído de várias etapas, tendo início no pré-processamento dos textos, passando por uma etapa de filtragem de sentenças irrelevantes, e finalizando com a aplicação de extratores de características, treinamento de classificadores e avaliação. As etapas e suas relações são apresentadas na figura 12.

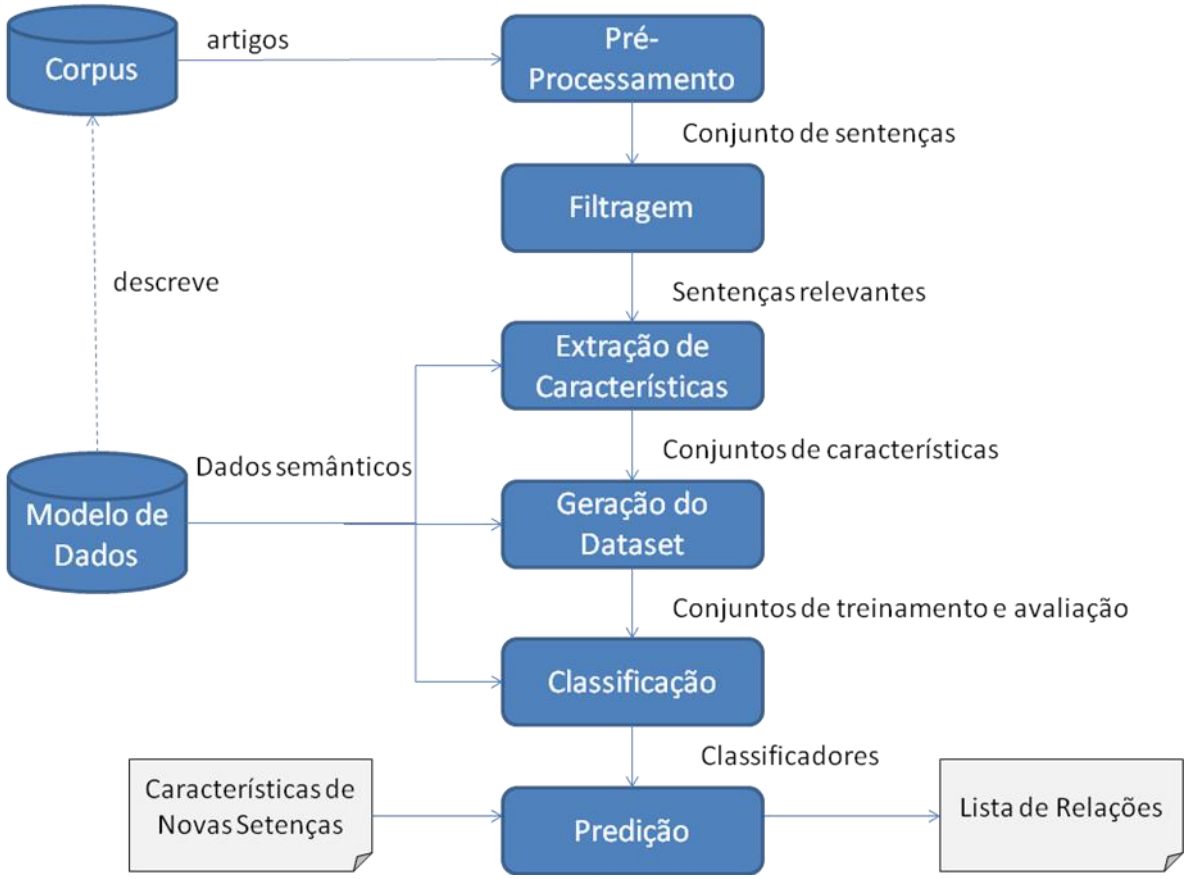

Figura 12 - Processo geral de extração de relações supervisionado.

Inicialmente, têm-se um **Corpus** que fornece os artigos em linguagem natural que descrevem as relações entre as entidades. Esse corpus é semanticamente descrito por um **Modelo de Dados**, podendo ser uma ontologia ou algum outro modelo formal.

Os artigos em linguagem natural fornecidos pelo corpus passam por um **Pré-Processamento**. Essa etapa divide os artigos em sentenças, que são pedaços de textos que contém pelo menos duas menções de entidades. Além disso, o pré-procesamento remove todas as partes do texto que são irrelevantes para extração de relações, como, por exemplo, tabelas e marcações de formatação. Após isso, realiza-se uma **Filtragem** do conjunto de sentenças com o objetivo de descartar aquelas que não fornecem indícios da existência de uma relação entre as entidades mencionadas. A filtragem pode ser, por exemplo, baseada na presença de palavras-chave na sentença. O conjunto de sentenças relevantes, obtido na fase anterior, serve como entrada para a fase de **Extração de Características**, onde são extraídas, entre outras, características léxicas, sintáticas e/ou semânticas das sentenças. Essa etapa gera conjuntos de características utilizados como exemplos de treinamento para relações previstas

no modelo de dados. Os conjuntos de características, por sua vez, servem como entrada para a fase de **Geração do *Dataset***, onde são gerados os conjuntos de treinamento e de avaliação, baseados no modelo de dados escolhido. Os conjuntos de treinamento e avaliação são usados na fase de **Classificação**, onde, um classificador é treinado e avaliado. A etapa de classificação pode gerar um classificador composto de dois ou mais classificadores, formando um modelo de assembléia. Por fim, os classificadores são disponibilizados para aplicação em novas sentenças em uma etapa de **Predição**. Essa etapa recebe como entrada representações de novas sentenças, em forma de conjuntos de características, e gera como saída uma lista de prováveis relações.

Diversas técnicas podem ser utilizadas separadamente ou em conjunto em cada etapa apresentada na figura 12. Por exemplo, na etapa de pré-processamento, podem ser utilizados analisadores que removem estruturas irrelevantes no texto, como tabelas e caixas de informação. Além disso, é necessária a utilização de uma estratégia para dividir o texto em sentenças. A etapa de filtragem pode selecionar sentenças com base no grau *TF-IDF*[22], que mede a especifidade dos seus termos, como mostra [Nguyen, Matsuo e Ishizuka, 2007], ou no grau de entropia das palavras, como em [Yan et al., 2009]. Na etapa de extração de relações, podem ser utilizados em conjunto, analisadores de dependência gramatical e analisadores de partes do discurso para obter evidências da presença de uma relação ([Bunescu e Mooney, 2005]). A geração do *dataset* pode considerar atributos numéricos e/ou nominais, bem como diversas técnicas e formatos de serialização do conjunto de dados. A etapa de classificação pode ser realizada por diferentes técnicas de aprendizado de máquina e/ou considerar a utilização de vários classificadores em conjunto.

Portanto, o *framework* proposto neste trabalho pretende permitir a implementação e a combinação dessas técnicas e algoritmos de forma flexível, garantindo a independência na implementação de cada fase do processo e maior agilidade na realização de experimentos.

4.2. Especificação

Os requisitos para o desenvolvimento do *framework* para extração de relações são listados a seguir:

[22] http://tfidf.com/

1. Permitir a variação e a implementação de técnicas distintas para os seguintes pontos de variação:
    a. Pré-Processamento;
    b. Filtragem de Exemplos;
    c. Extração de Características;
    d. Geração do *Dataset*;
    e. Classificação: o *framework* deve permitir o uso de diversas técnicas de aprendizado supervisionado para classificação;
    f. Validação: o *framework* deve permitir o uso de várias técnicas de avaliação dos classificadores
2. O *framework* deve permitir a instanciação através de configuração;
3. O *framework* deve gerar *logs* de execução para cada fase;
4. Os classificadores treinados devem ser registrados em uma base e disponibilizados para utilização.

A arquitetura modular proposta pretende acomodar cada um dos requisitos fornecendo pontos de variação para cada uma das etapas do processo de ER, configurados através de arquivos.

4.3. Arquitetura

A arquitetura modular apresentada nesta seção permite a implementação de diversas técnicas para as principais etapas de um processo de ER de forma flexível. A figura 13 apresenta uma visão geral dos módulos presentes no *framework*.

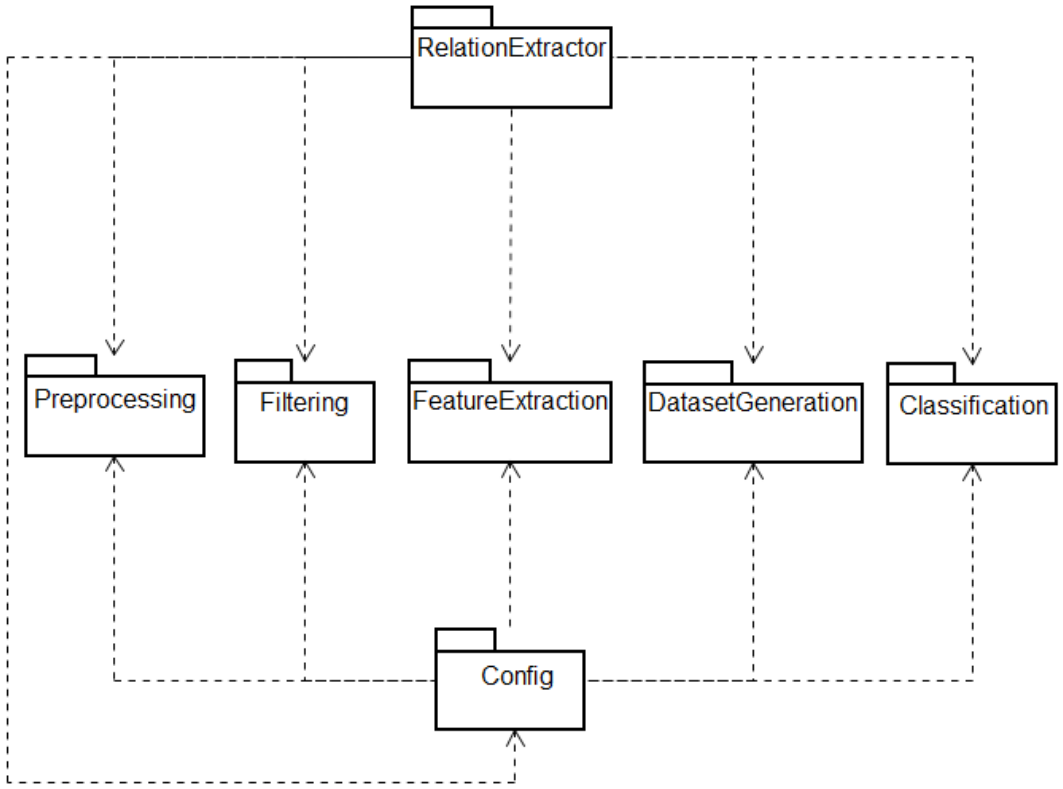

Figura 13 – Arquitetura de módulos do *framework* proposto.

A figura 13 apresenta os módulos principais, correspondentes as etapas da figura 12, e suas relações de dependências. As funções de cada módulo principal são as mesmas descritas na seção 5.1.

O módulo principal, *RelationExtractor,* é responsável por realizar chamadas aos objetos das classes pertencentes aos demais módulos para atingir um resultado. A principal idéia por trás da implementação do módulo *RelationExtractor* é a de mediação, onde, ele é responsável por implementar os fluxos de execução servindo como mediador entre as comunicações dos demais módulos. Isso os torna independentes uns dos outros. Todos os módulos principais são instanciados e configurados pelo módulo *Config,* de acordo com as configurações presentes nos arquivos de configuração.

Apesar de independentes uns dos outros, os módulos principais apresentados na figura 13 dependem de outros módulos que provêem funcionalidades de suporte as suas execuções. Os módulos de suporte e suas relações de dependência são apresentados na figura 14.

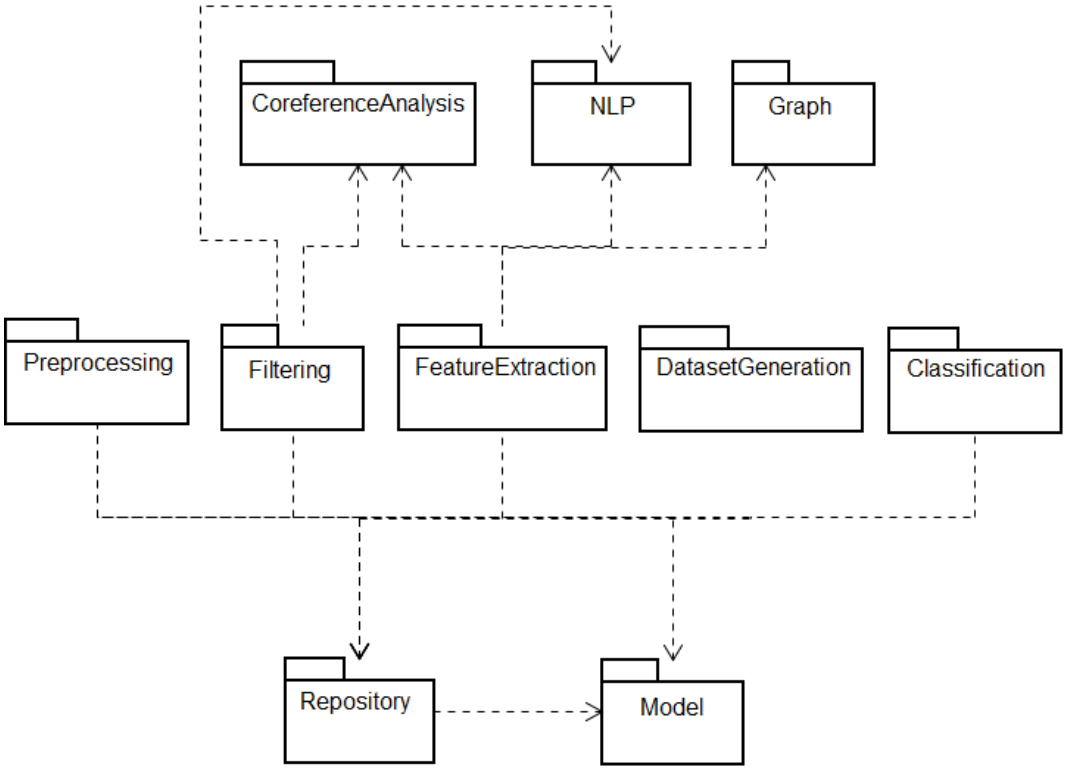

Figura 14 – Relações de dependência entre os módulos principais e os módulos de suporte.

Todos os módulos principais, com excessão do *DatasetGeneration,* dependem especificamente de dois módulos: *Repository* e *Model*. *Model* é responsável por fornecer classes de modelo de um processo de ER. Isso inclui, entre outras, classes para representar setenças e entidades. Cada módulo necessita realizar consultas a um repositório de sentenças, ou a um repositório de dados, e salvar o resultado de suas operações. Para isso, o módulo *Repository* fornece classes para acesso a repositórios de dados e de sentenças.

Algumas técnicas de filtragem de sentenças irrelevantes estudadas baseiam-se na presença de no mínimo duas menções de entidades. Para isso, é necessário realizar uma análise de possíveis correferências existentes no sujeito e no objeto das sentenças. Por esse motivo, o módulo *Filtering* é dependente dos módulos *NLP* e *CoreferenceAnalysis*.

Algumas estratégias de extração de características implementadas no módulo *FeatureExtraction* envolvem análise de dependêcia gramatical e de partes do discurso, análise de co-referências e aplicação de algoritmos de grafos. Portanto, o módulo *FeatureExtraction* depende dos módulos de *NLP*, *CoreferenceAnalysis* e *Graph,* que fornecem essas funcionalidades.

4.3.1.
Módulo de Modelo

Esse módulo fornece todas as classes relacionadas ao modelo de um processo de extração de relações. Isso inclui classes que representam sentenças, entidades, correferências, relações, dependências gramaticais e artigos. O diagrama de classes do módulo de modelo é apresentado na figura 15.

As classes que representam as relações entre as entidades são a *Relation* e a *Entity*. Uma instância de *Relation* possui um identificador e um rótulo, e está relacionada a duas entidades, uma é o sujeito e outra é o objeto da relação. Essas relações são extraídas do modelo de dados.

As sentenças são representadas pela classe *Sentence* e formam a parte textual que descreve em linguagem natural as instâncias das relações. Um exemplo para uma relação é, normalmente, constituído de informações obtidas a partir do texto e das entidades presentes. Com isso, uma instância de *Sentence* possui um atributo texto (*text*) e uma relação para classe *Article*. Além disso, uma sentença também possui uma lista de *tokens*, extraídos na fase de pré-processamento.

Durante a fase de filtragem ou extração de características, podem ser extraídas dependências gramaticais entre os termos com a utilização de um analisador de dependências. Uma dependência possui um nó governante, um nó dependente e uma relação gramatical. Com isso, as classes *TypedDependency* e *Node* são utilizadas para representar dependências e nós. A classe *Node* possui um índice e um *token* da sentença. A classe *TypedDependency* possui a relação gramatical entre os nós governante e dependente. Além disso, uma sentença pode estar relacionada a várias dependências entre termos.

Alguns filtros e extratores de características identificam menções do sujeito da relação na sentença, essas menções são representadas por instâncias da classe *Coreference*. Uma co-referência menciona uma entidade em uma sentença específica. Além disso, podem existir várias menções de uma entidade em uma sentença. Com isso, várias instâncias de *Coreference* podem estar relacionadas a uma instância de *Entity* e a uma instância de *Sentence*.

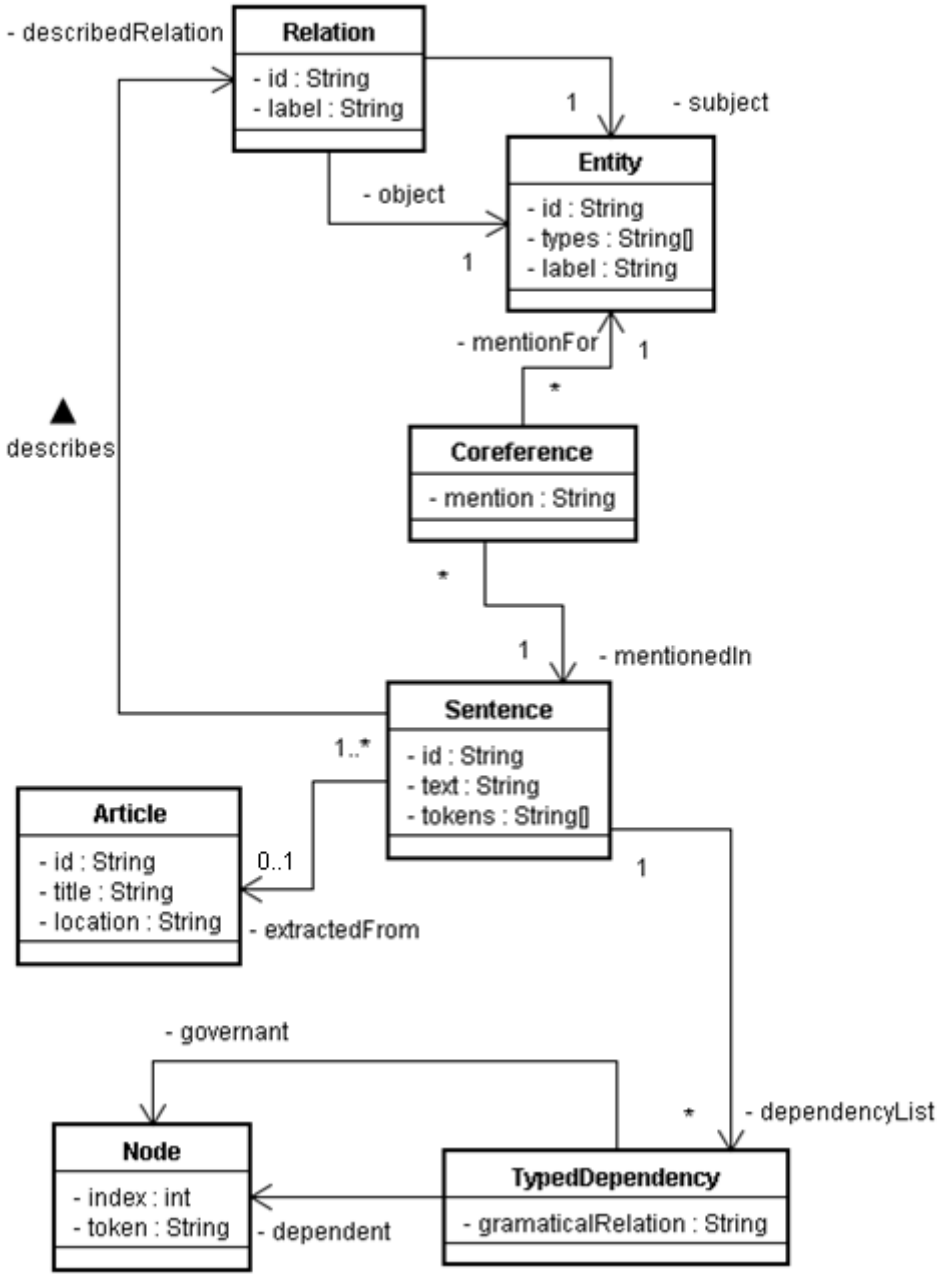

Figura 15 – Diagrama de classes de modelo do *framework*.

4.3.2. Módulo de Pré-processamento

O módulo *de* pré-processamento fornece classes e interfaces utilizadas para remover informações irrelevantes dos artigos, dividir os artigos em sentenças, sentenças em tokens e reduzir a variação de *tokens* nas sentenças. A figura 16 mostra a classe principal *Preprocessor* e os pontos de extensão do módulo através das interfaces que englobam os diversos aspectos dessa etapa.

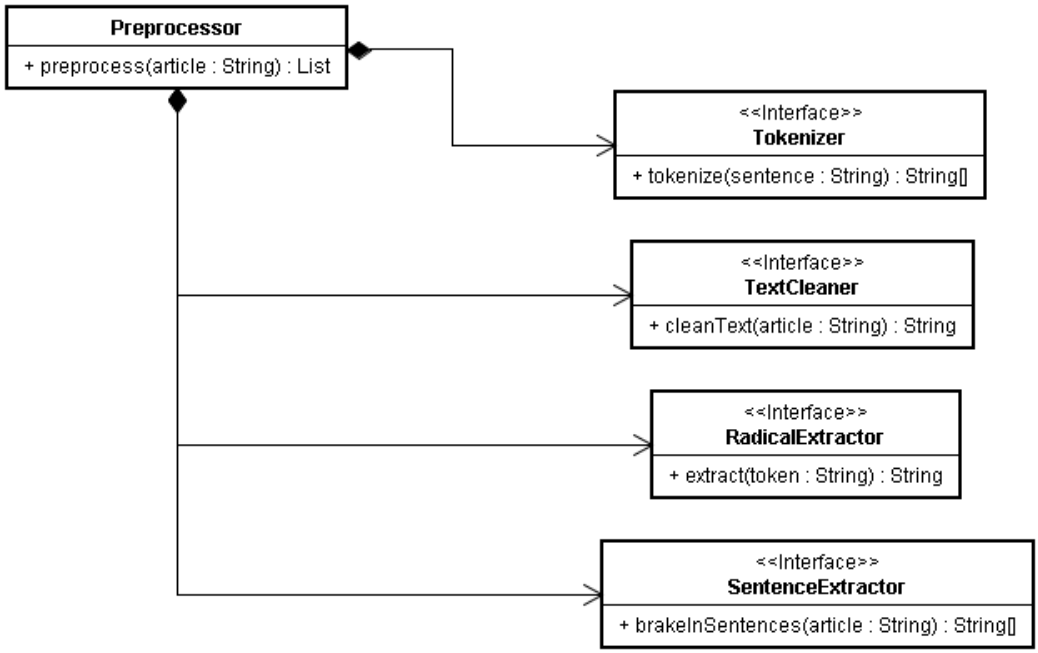

Figura 16 – Classe *Preprocessor* e interfaces do módulo de pré-processamento.

O pré-processador, implementado pela classe *Preprocessor,* é configurado com várias estratégias que fornecem as operações necessárias a sua execução. A operação de remoção de estruturas e marcações irrelevantes afim de normalizar o texto é capturada pela interface *TextCleaner*. Utilizando o padrão Estratégia [Gamma et al., 1995], é possível estender essa interface para realizar a normalização de vários tipos de texto, como texto wiki em várias linguagens de marcação (markdown, mediawiki, etc.) ou texto de notícias, por exemplo.

A operação de divisão do texto do artigo em sentenças é fornecida pela interface *SentenceExtractor*. Essa operação pode ser impementada por uma simples heurística de reconhecimento de sentenças utilizando a pontuação como delimitador, ou podem ser utilizadas técnicas mais elaboradas com um modelo de aprendizado treinado para reconhecer sentenças em uma lingua específica, como é o caso da ferramenta OpenNLP[23].

A divisão da sentença em *tokens* é realizada pela operação fornecida pela interface *Tokenizer*. Assim como a divisão do texto em sentenças, a divisão em *tokens* pode ser realizada por heurísticas mais simples, usando espaços em branco como delimitadores, ou por técnicas mais elaboradas.

Um problema comum que surge quando utilizamos *tokens* como unidades léxicas, é a necessidade de redução da variação de *tokens* relacionados. Como exemplo, os *tokens* “conectar”, “conectado” e “conectando” possuem significado

[23] http://opennlp.apache.org/

similar e é desejável que sejam tratados como um único *token*, “conect”. A interface responsável pela operação de extração de radicais dos *tokens* é a *RadicalExtractor*. Existem várias estratégias para essa operação, como os algoritmos de Porter e Savoy[24].

A figura 17 mostra alguns exemplos de estratégias fornecidas pelo *framework*.

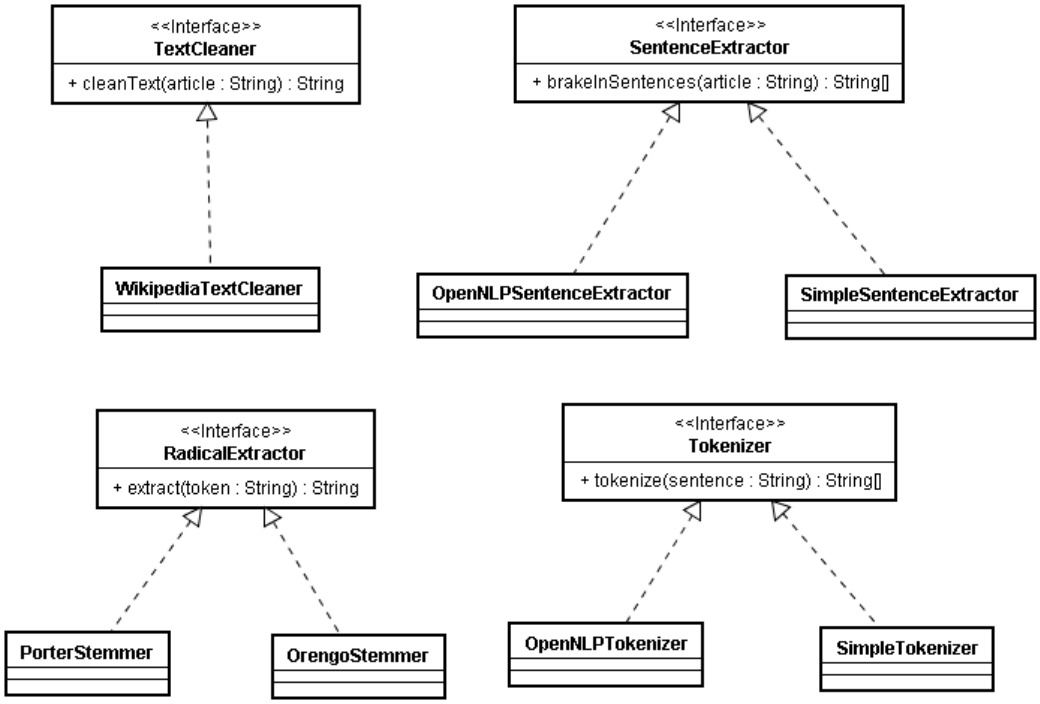

Figura 17 – Estratégias fornecidas pelo *framework* para cada ponto de extensão.

A classe *Preprocessor* implementa o fluxo de execução do módulo através do método *preprocess*. Esse método recebe o artigo a ser pré-processado e realiza uma sequência de chamadas as interfaces que compõem o módulo para obter como resultado uma lista de sentenças, representadas por instâncias da classe *Sentence* do modelo. O diagrama de sequências da figura 18 mostra a sequência de chamadas as interfaces implementada na classe *Preprocessor.*

[24] http://members.unine.ch/jacques.savoy/clef/index.html

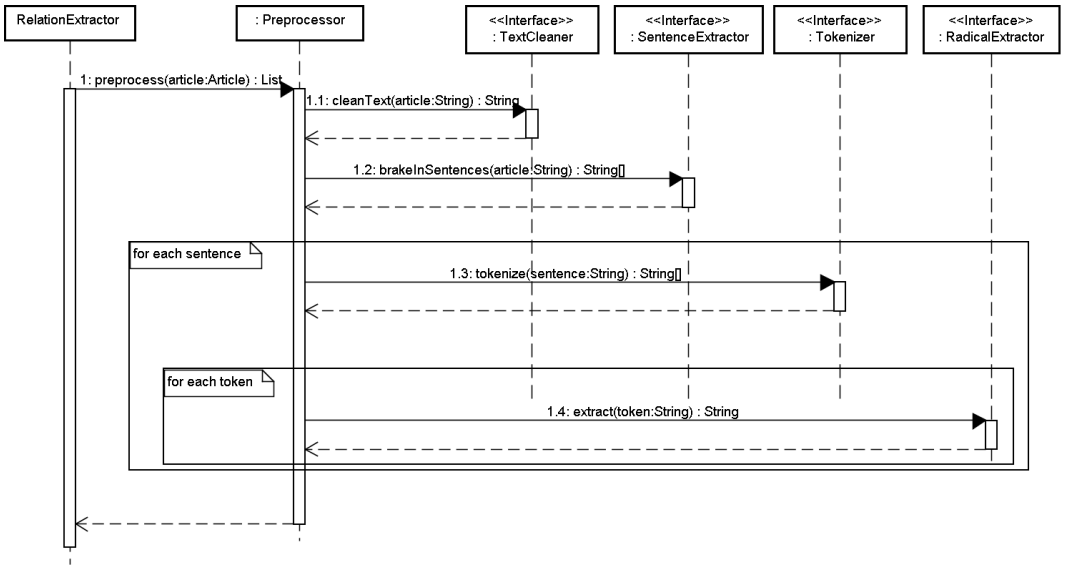

Figura 18 – Diagrama de sequências da execução do método *preprocess* da classe *Preprocessor*

Após a configuração correta do módulo, o pré-processamento tem início com uma chamada ao método *preprocess* em uma instância de *Preprocessor,* passando uma instância do artigo a ser processado como parâmetro. Primeiro, o pré-processador remove estruturas e marcações irrelevantes do texto através da chamada *cleanText* a uma instância de *TextCleaner*. O texto limpo é, então, dividido em sentenças através de uma chamada ao método *brakeInSentences* de *SentenceExtractor*, que retorna uma lista de textos do tipo *string*. Após isso, incia-se um laço para dividir cada sentença em uma lista de *tokens* através de chamadas ao método *tokenize* da classe *Tokenizer.* Para cada *token* obtido, o pré-processador extrai o seu radical através de chamadas ao método *extract* da classe *RadicalExtractor*. Por último, o pré-processador cria e retorna uma lista de instâncias da classe *Sentence*, com os seus respectivos textos e radicais.

4.3.3. Módulo de Filtragem

Esse módulo é responsável por fornecer heurísticas de filtragem de sentenças irrelevantes na descrição das relações. Cada heurística aplicada é considerada um filtro, onde, os filtros são aninhados em cadeia e aplicados em sequência a uma lista de sentenças. A figura 19 mostra o diagrama de classes do módulo de filtragem.

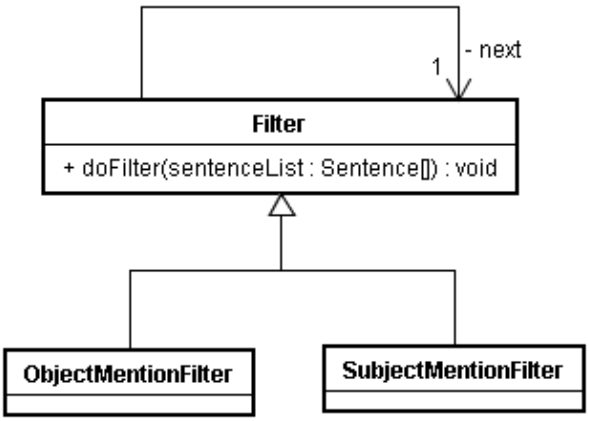

Figura 19 – Diagrama de classes do módulo de filtragem.

Os diferentes tipos de filtros são extensões concretas da classe *Filter*. Cada extensão deve fornecer uma estratégia de filtragem que pode ser utilizada isoladamente ou em conjunto. A figura 19 mostra dois tipos de filtros concretos, o *ObjectMentionFilter* e o *SubjectMentionFilter*. Esses dois filtros são responsáveis, respectivamente, por descartar todas as sentenças que não possuem uma referência explícita ao objeto e ao sujeito da relação a qual descrevem.

Para formar uma cadeia de responsabilidades [Gamma et al., 1995], a classe *Filter* possui uma referência para o próximo filtro a ser aplicado. Dessa forma, o método *doFilter* aplica a estratégia de filtragem à lista de sentenças e, em seguida, realiza uma chamada ao método *doFilter* do seu sucessor passando a lista processada. Essa sequência de chamadas se estende até o final da cadeia, passando filtro a filtro. Por exemplo, se quisermos manter somente as sentenças que mencionam tanto o sujeito quanto o objeto das relações, é necessário instanciar as classes *SubjectMentionFilter* e *ObjectMentionFilter*, relacionar as instância através do papel *next* e invocar o método *doFilter* na primeira instância passando a lista de sentenças.

Uma vez que as classes concretas *ObjectMentionFilter* e *SubjectMentionFilter* necessitam reconhecer menções de entidades que participam das relações, elas dependem do módulo de análise de co-referências que será explicado nas próximas seções.

De forma a acomodar novos tipos de filtros, como por exemplo, filtros baseados em palavras-chave compartilhadas entre as sentenças ([Nguyen, Matsuo e Ishizuka, 2007]), a classe *Filter* oferece um ponto de extensão para o módulo.

4.3.4.
Módulo de Extração de Características

O módulo de extração de características fornece classes responsáveis por extrair informações das sentenças que são relevantes para caracterizar uma relação. Cada classe desse módulo é responsável por extrair um determinado tipo de característica, podendo ser léxica, sintática ou semântica. A figura 20 mostra as principais classes e relacionamentos desse módulo.

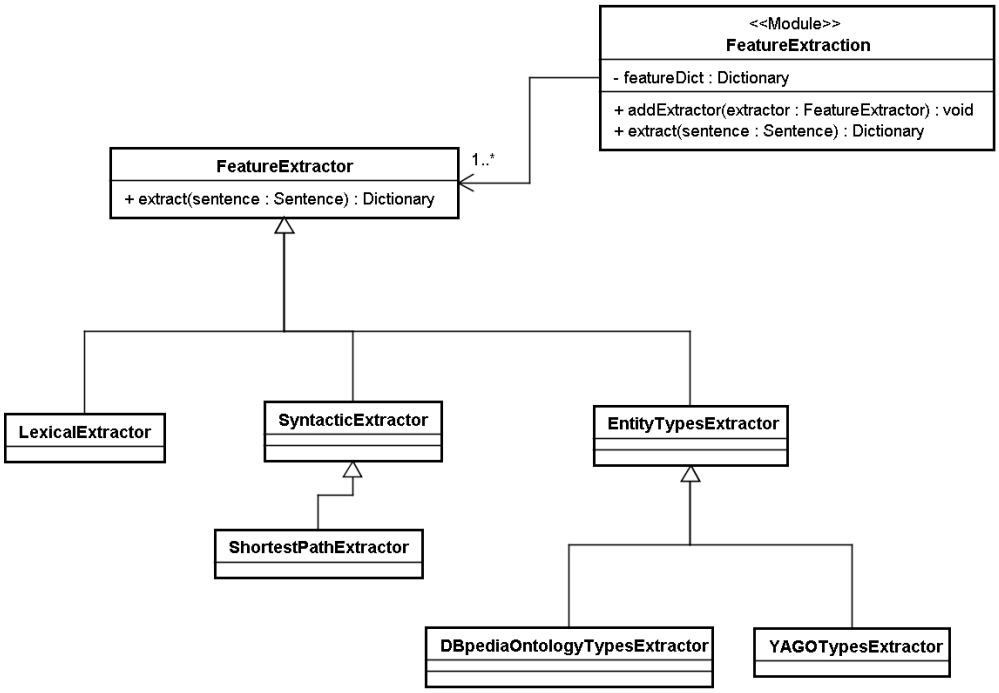

Figura 20 – Diagrama de classes do módulo de extração de características.

Todos os extratores de características devem estender a classe *FeatureExtractor* que possui o método *extract*. O método *extract* recebe uma sentença e extrai um dicionário de atributos e valores. Cada par <*atributo, valor*> representa uma característica da sentença. A representação em dicionário do conjunto de características foi adotada por questões de otimização de consumo de memória e por facilitar a geração de representações esparsas na fase de geração do conjunto de dados.

As classes concretas que estendem *FeatureExtractor* devem prover diferentes implementações do método *extract,* considerando diferentes tipos de características.

O módulo *FeatureExtraction* fornece métodos para adicionar extratores (*addExtractor*) e extrair características (*extract*) utilizando os extratores registrados. Os extratores de características são selecionados através de

configuração, onde, o módulo de configuração é responsável por criar as instâncias corretas e adicioná-las ao módulo *FeatureExtraction*.

Após a configuração do módulo *FeatureExtraction* com os extratores de interesse, a extração deve ser efetuada pela chamada ao método *extract* do módulo. Esse método executa em sequência os extratores adicionados e combina os dicionários retornados. Dessa forma, é possível utilizar várias combinações de extratores de características onde, ao final, todos os resultados individuais são combinados.

Alguns extratores concretos são fornecidos pelo *framework*. A classe *LexicalExtractor* considera cada *token* da sentença como sendo uma característica, seguindo uma abordagem de saco-de-palavras unigrama.

Além de extratores léxicos, o *framework* fornece a classe *ShortestPathExtractor* para extração de características sintáticas das sentenças. Essa classe é responsável por extrair o menor caminho entre as entidades na árvore de dependência.

Normalmente, extratores de características sintáticos utilizam de dois tipos de analisadores, um analisador de partes do discurso e um analisador de dependências gramaticais. Com isso, as classes dos extratores sintáticos devem estar relacionadas aos analisadores fornecidos pelo módulo de processamento de linguagem natural NLP. Esses relacionamentos são estabelecidos na super-classe *SyntaticExtractor*, como mostra o diagrama da figura 21.

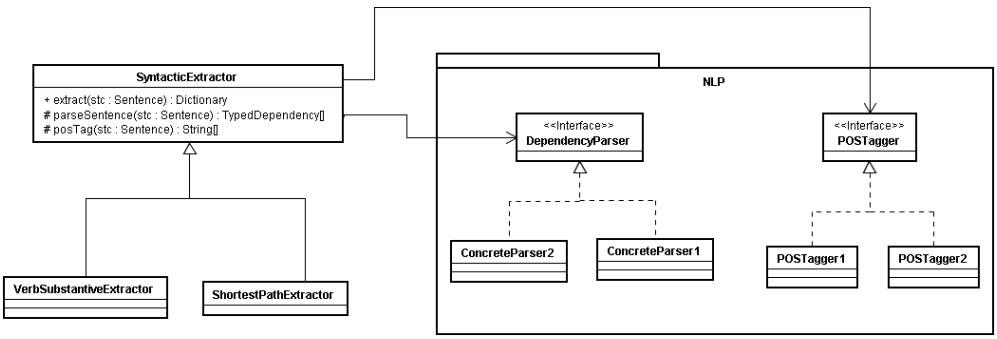

Figura 21 – Dependências do módulo de extração de características.

A classe *SyntaticExtractor* está relacionada aos analisadores através das interfaces *DependencyParser* e *POSTagger* do módulo NLP. Dessa forma, é possível variar os tipos de analisadores sem causar nenhum impacto na implementação dos extratores de características. Isso permite, por exemplo, a utilização de analisadores construídos para diversos idiomas, possibilitando o desenvolvimento de reconhecedores de relações multilingue. A implementação

dessa parte do *framework* está de acordo com o padrão de projetos *Bridge* [Gamma et al., 1995].

As operações de análise são fornecidas pela própria classe *SyntaticExtractor* através dos métodos *parseSentence* e *posTag*. Esses métodos, por sua vez, delegam as chamadas para os analisadores concretos selecionados através de configuração. Esses dois métodos não são necessariamente utilizados em conjunto na mesma classe filha de *SyntaticExtractor*. Por exemplo, a classe filha *VerbsSubstantiveExtractor* somente utiliza o método *posTag* para obter as partes do discurso dos termos e selecionar somente verbos e substantivos como características.

Os extratores semânticos são implementados nas classes *EntityTypesExtractor*, *YAGOTypesExtractor* e *DBpediaOntologyTypesExtractor*. A superclasse *EntityTypesExtractor* extrai todos os tipos das entidades sujeito e objeto da relação. Suas subclasses selecionam um esquema de classificação específico. Por exemplo, A subclasse *YAGOTypesExtractor* é responsável por extrair somente os tipos do esquema de classificação YAGO. Da mesma forma, a classe DBpediaOntologyTypesExtractor seleciona somente as classes do esquema de classificação da ontologia do DBpedia, identificadas através do namespace "*http://dbpedia.org/ontology/*".

4.3.5. Módulo de Geração do Dataset

Uma vez que os dicionários de características já se encontram gerados, é necessário convertê-los para um formato de dados que possa ser aplicado para treinar e avaliar classificadores. Esses formatos podem variar de acordo com as necessidades dos usuários. O diagrama da figura 22 apresenta as principais classes desse módulo.

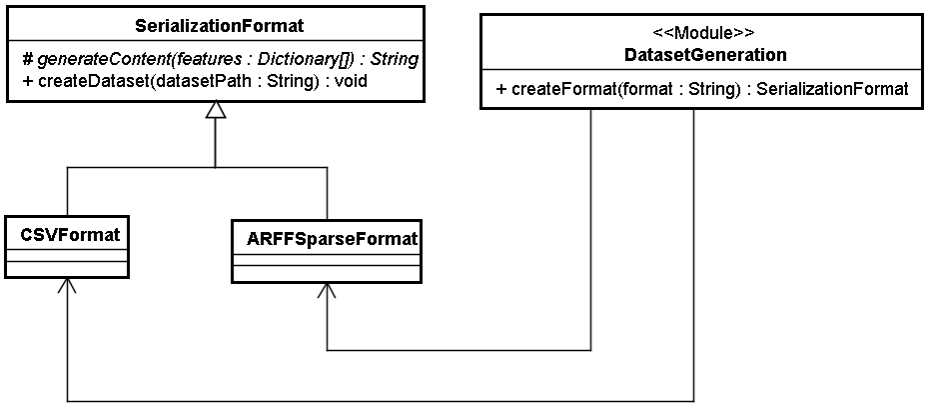

Figura 22 – Diagrama de classes do módulo de geração do *dataset*.

O módulo *DatasetGeneration* possui o método *createFormat* que é responsável por instanciar a classe do formato correto de acordo com o valor do parâmetro *format*. Caso o valor seja “CSV”, será criada uma instância da classe *CSVFormat*, caso seja “sparseARFF”, será retornada uma instância de *ARFFSparseFormat*. Cada classe de formato deve fornecer sua própria implementação do método *generateContent*, fornecido pela super-classe *SerializationFormat,* e retornar uma representação em string do dataset para ser salva em arquivo. O caminho do arquivo é especificado através do parâmetro *datasetPath* no método *save*.

A classe *CSVFormat* gera o dataset como uma matriz em formato csv, onde cada linha representa um exemplo para uma relação. A primeira linha é o cabeçalho da matrix, formado por todos os atributos. Cada célula armazena um valor para o atributo da coluna. As últimas colunas representam a classe dos exemplos, que em ER são equivalentes as relações sobre as quais se deseja aprender. A tabela 7 mostra um exemplo em forma de uma matriz binária para a relação *birthPlace*, onde, o valor 1 indica a presença da palavra no texto da sentença que descreve a relação. Caso contrário, o valor é 0.

	...	15	16	17	18	19	20	21	...	27.280
1	...	Birth	Born	City	Grow	In	Location	Place	...	Relation
2	...	0	1	1	0	1	0	0	...	birthPlace

Tabela 7 – Exemplo de matriz gerada pela classe *CSVFormat* para ser salva em formato csv.

A classe *ARFFSparseFormat* gera o conteúdo em formato ARFF de forma esparsa, ou seja, ignorando os atributos que possuem valor 0. Isso é feito

indicando o índice do atributo, no cabeçalho, e o valor. Com isso, o exemplo da tabela 7 seria reduzido para:

16 1, 17 1, 19 1, 27.280 birthPlace

onde, os pares "*atributo valor*" são separados por vírgula e a última posição é dedicada a classe do exemplo.

A representação ARFF esparsa requer menos espaço de armazenamento e possui um desempenho melhor. Caso haja a necessidade de novos formatos de serialização, basta criar uma nova classe filha de *SerializationFormat* e fornecer uma implementação diferente para o método *generateContent* e/ou *save.*

4.3.6. Módulo de Classificação

O módulo de classificação fornece classes para construção de classificadores e avaliadores. Todos os classificadores implementados no *framework* são *wrappers* para os classificadores fornecidos pela biblioteca *Weka*[25], implementada em Java. A figura 23 mostra o diagrama de classes desse módulo.

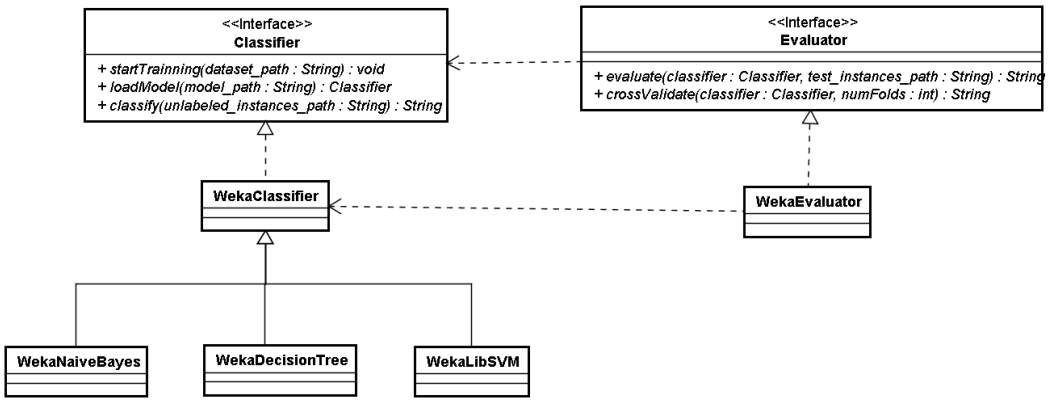

Figura 23 – Diagrama de classes do módulo de classificação.

Os classificadores são implementações da interface *Classifier* e devem prover implementações para três métodos: *startTrainning*, *loadModel* e *classify.* Após a geração do arquivo contendo o conjunto de instâncias de treinamento,

[25] http://www.cs.waikato.ac.nz/ml/weka/

um classificador é instanciado e treinado através da chamada ao método *startTrainning*, onde, o caminho do conjunto de treinamento é passado por parâmetro. Após finalizado o treinamento, o modelo gerado é salvo em um diretório específico do *framework*, obtido do arquivo de configuração.

Para realizar a avaliação dos classificadores, foi criada a interface *Evaluator* que apresenta duas formas de avaliação: validação cruzada e avaliação por conjuntos de teste. Para execução do método de validação cruzada, é necessário especificar o número de partes desejado como parâmetro. O método de avaliação por conjuntos de teste exige que seja passado o caminho para o arquivo onde se encontram as instâncias de teste. Os dois métodos avaliam uma instância de um classificador que implementa a interface *Classifier*.

A superclasse *WekaClassifier* é responsável por carregar as classes da biblioteca Weka necessárias a utilização dos classificadores. Cada classe concreta instancia um classificador Weka diferente. A classe *WekaNaiveBayes* utiliza o classificador NaiveBayes do weka. O mesmo ocorre para a técnica de árvores de decisão (*WekaDecisionTree*) e SVM da biblioteca LibSVM (*WekaLibSVM*), que é integrada ao Weka.

O módulo de classificação oferece um dos pontos de extensão do *framework*, onde, novos classificadores podem ser adicionados, bastando para isso, que eles implementem a interface *Classifier*. Assim como classificadores, novos avaliadores podem ser implementados de forma independente do restante do *framework*.

4.3.7. Módulo NLP

O módulo NLP é um dos módulos de suporte aos principais sendo responsável por fornecer os analisadores de dependência e de partes do discurso para textos em linguagem natural. A figura 24 apresenta as principais classes e interfaces desse módulo.

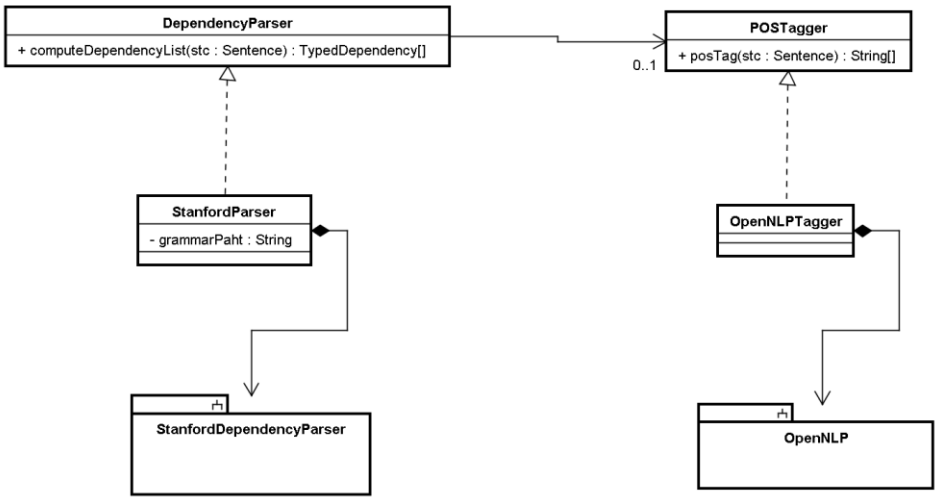

Figura 24 – Diagrama de classes do módulo NLP.

As principais classes desse módulo são a *DependencyParser* e a *POSTagger*. Essas classes fornecem métodos para calcular as dependências gramaticais e extrair as partes do discurso dos termos presentes nas sentenças passadas por parâmetro.

O método *computeDependencyList* de *DependencyParser* retorna uma lista de objetos do tipo *TypedDepedency* (módulo de modelo). Como explicado na seção 5.3.1, essa classe representa uma dependência gramatical entre dois termos e possui como atributo o rótulo da relação entre eles. As dependências são relacionadas a sentença que as originou e persistidas para futuras consultas.

A classe *POSTagger* fornece a operação *posTag* para extrair partes do discurso dos termos presentes na sentença passada por parâmetro. Esse método retorna uma lista de rótulos do mesmo tamanho da lista de *tokens* da sentença, onde, cada posição da lista armazena o rótulo para o *token* naquela mesma posição.

Analisadores de dependência podem estar relacionados a um analisador de partes do discurso, como é o caso do analisador parcial de dependências multilíngue DepPattern. Por esse motivo, a classe *DependencyParser* possui um relacionamento para a classe *POSTagger.*

As classes mais específicas implementam um determinado tipo de analisador. A classe *StanfordParser* encapsula chamadas ao analisador de dependências de Stanford. O analisador de Stanford pode ser utilizado com diversas gramáticas, para vários idiomas. Como exemplo, são disponibilizadas gamáticas para inglês, francês, alemão e chinês. Por esse motivo, a classe

StanfordParser possui um atributo que armazena o caminho da gramática a ser utilizada. A classe *OpenNLPTagger* encapsula chamadas ao analisador de partes do discurso implementado na biblioteca OpenNLP.

Além das classes de analisadores, o módulo NLP fornece duas operações auxiliares, como mostra a figura 25.

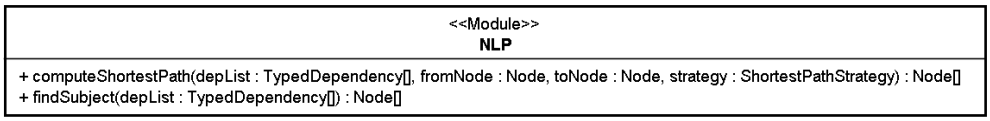

Figura 25 – Operações do módulo NLP.

A operação *computeShortestPath* tem a função de retornar uma lista de nós que compõem o menor caminho na árvore de dependência entre um nó de origem e um nó de destino. Para isso, ela recebe uma lista de dependências (*depList*), gera uma representação em grafo não direcionado dessa lista e aplica um algorimo de caminho mínimo, definido pelo parâmetro *strategy* entre os nós de origem (*fromNode*) e de destino (*toNode*). Essa operação, pressupõe que a lista de dependências forma um grafo conectado, como a árvore da figura 10 . A operação *findSubject* encontra os nós que compõem o sujeito da sentença na lista de dependências.

Caso haja necessidade de utilização de outros tipos de analisadores, basta criar uma nova implementação das classes *DependencyParser* e *POSTagger*.

4.3.8. Módulo de Grafos

Assim como os módulo NLP, esse módulo também é de suporte. Ele fornece classes e operações relacionadas a grafos. Atualmente, esse módulo fornece uma interface, uma implementação do algoritmo Dijkstra e uma operação auxiliar, como mostram as figuras 26 e 27.

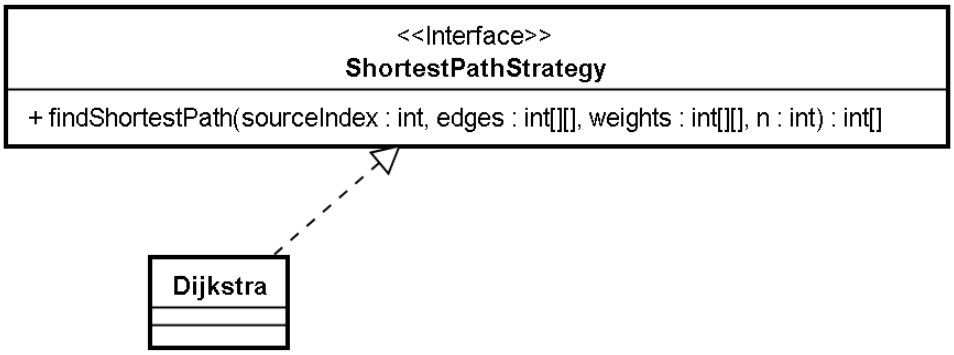

Figura 26 – Diagrama de classes do módulo de grafos.

A interface *ShortestPathStrategy* fornece a operação de cálculo de caminho mínimo de um nó de origem para todos os outros. Ela recebe como parâmetros, respectivamente, o índice do nó de origem, uma matrix de adjacências, uma matrix de pesos e a quantidade de nós. Após o cálculo, a operação retorna um vetor onde os índices equivalem aos nós e os valores equivalem aos seus respectivos predecessores no caminho mínimo.

Para converter uma lista de dependências em um grafo representado por uma matriz de adjacências e uma matriz de pesos, o módulo *Graph* fornece uma operação auxiliar, como mostra a figura 27.

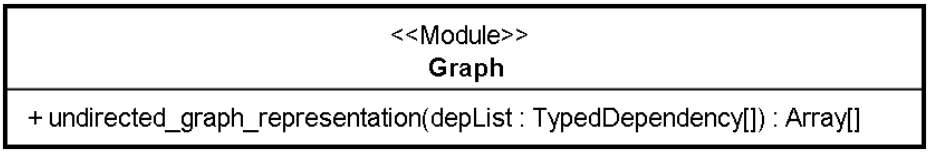

Figura 27 – Operação auxiliar do módulo *Graph*.

A operação *undirected_graph_representation* recebe como parâmetro uma lista de dependências do tipo *TypedDependency*. Baseado nas dependências dessa lista é montado um grafo não direcionado representando como uma matriz de adjacências e uma matriz de pesos. As matrizes são adicionadas a uma lista e retornados.

4.3.9. Módulo Repositório

Esse módulo encapsula o acesso aos repositórios e traduzem seus dados para os objetos de modelo que alimentam todo o processo de ER. O diagrama da figura 28 mostra as principais classes do módulo Repositório. Pelo diagrama,

percebe-se que o *framework* é capaz de trabalhar com três tipos diferentes de repositórios: um repositório *RDF*, um repositório relacional e um repositório em XML.

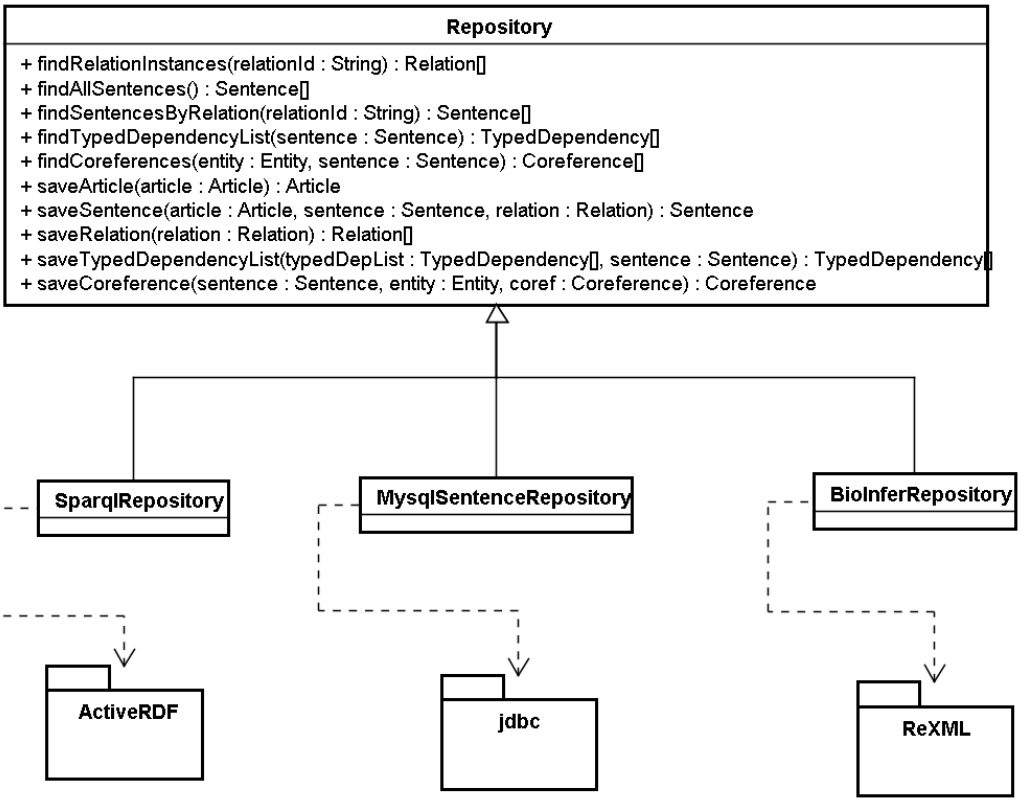

Figura 28 – Diagrama de classes dos repositórios fornecidos pelo *framework*

Atualmente, o repositório *RDF,* implementado pela classe *RDFRepository,* fornece o acesso a um determinado *SPARQL endpoint*. Dessa forma, o modelo de dados pode ser uma ontologia e um dos conjuntos de dados publicados através da iniciativa *Linked Data*. O principal método de busca implementado nessa classe é o *findRelationInstances* que, dada uma URI, ele busca por instâncias de relações registradas como triplas RDF. O acesso aos *endpoints* é encapsulado pelo *framework* para acesso a dados RDF ActiveRDF [Oren et al., 2010].

A classe *MysqlSentenceRepository* fornece o acesso ao banco que possui o esquema equivalente ao modelo apresentado na seção 5.3.1. Esse banco armazena relações, sentenças de artigos, entidades e tipos, relações de dependências entre termos e correferências. Essa classe é utilizada para armazenar informações obtidas dos exemplos à medida que eles são passados

através dos módulos. O acesso ao banco de dados Mysql é realizado através da biblioteca java JDBC[26].

A classe *BioInferRepository* foi implementada como exemplo de utilização de outros tipos de conjuntos de sentenças no *framework*. BioInfer[27] é um conjunto de dados para ER voltado para a área de bio-medicina, representado em XML. A classe *BioInferRepository* utiliza a biblioteca ReXML[28] para manipulação de dados em XML.

Além de viabilizar o acesso a repositórios de dados estruturados, o módulo repositório também fornece meios para acessar repositórios de dados não estruturados, que fornecem textos de artigos em linguagem natural. A figura 29 mostra uma interface e uma classe fornecida pelo *framework* para recuperar artigos de entidades da Wikipedia.

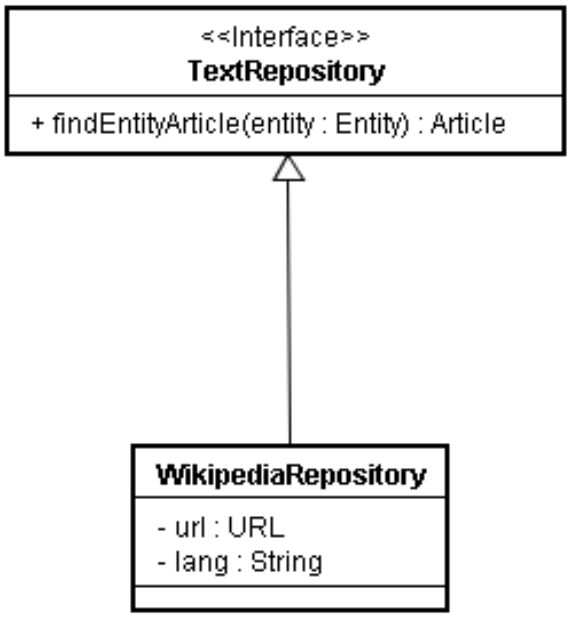

Figura 29 – Interface *TextRepository* e classe *WikipediaRepository*.

A interface *TextRepository* fornece o método *findEntityArticle* para recuperação de artigos que descrevem entidades. Esse método recebe uma instância de *Entity* como parâmetro. Uma implementação fornecida pelo *framework* é a classe *WikipediaRepository*. Essa classe possui dois atributos, a *URL* da Wikipédia, ou de uma versão dela, e a sua linguagem.

[26] http://www.oracle.com/technetwork/java/javase/jdbc/index.html

[27] http://mars.cs.utu.fi/BioInfer/

[28] http://www.germane-software.com/software/rexml/

4.3.10.
Módulo de Análise de Correferências

Um desafio comum em NLP consiste em como identificar o termo ou o conjunto de termos que mencionam alguma entidade no texto. Neste trabalho, o problema é como identificar na sentença os termos que mencionam o sujeito e o objeto da relação. Para isso, foram implementadas três heurísticas que tem mostrado bons resultados principalmente em artigos da Wikipedia: análise de termos entre aspas, pronomes mais frequentes e análise do sujeito das sentenças. As heurísticas podem ser aplicadas separadamente ou em conjunto. A figura 30 apresenta o diagrama de classes desse módulo.

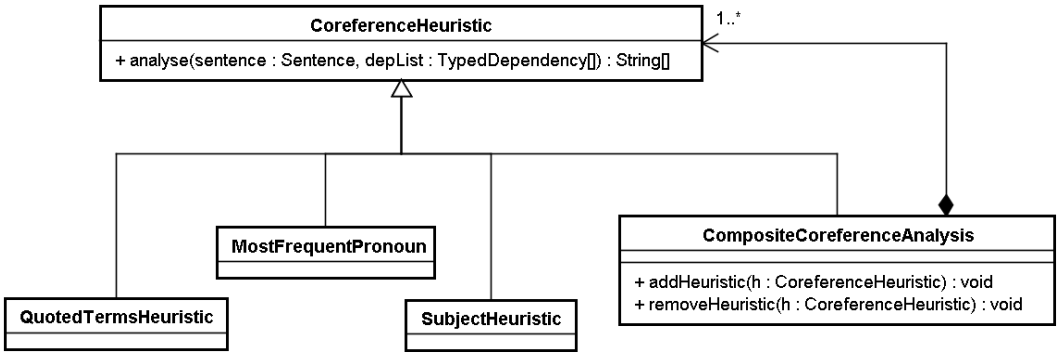

Figura 30 – Diagrama de classes do módulo de correferências.

A superclasse *CoreferenceHeuristic* fornece o método para análise de coreferências *analyse*. Esse método recebe como parâmetro a sentença a ser analisada e uma lista de dependências dessa sentença. Após a análise, o método retorna uma lista de correferências para o sujeito da relação descrita pela sentença. Uma heurística pode ser composta de várias outras mais simples, para isso, a classe *CompositeCoreferenceAnalysis* fornece métodos para adicionar e remover heurísticas.

4.4.
Implementação

Esta seção apresenta aspectos de implementação do *framework*. Isso inclui a linguagem de programação utilizada, as decisões de implementação tomadas e as principais dependências.

4.4.1. JRuby

Ruby é uma linguagem de *scripting*, que têm sido apontada como adequada ao desenvolvimento de aplicações para Web Semântica graças a características como tipagem dinâmica, herança múltipla e por não ser estritamente necessário que os objetos estejam em conformidade com as suas definições de classes [Synth]. Portanto, a linguagem Ruby foi selecionada para a implementação deste trabalho, pois, o *framework* foi projetado para operar não só com dados relacionais, como também com esquemas e dados RDF, tirando vantagem do crescimento contínuo da rede de conjuntos de dados estruturados publicados de acordo com os padrões *Linked Data*.

JRuby é uma implementação da linguagem Ruby cujo interpretador é escrito em linguagem Java. Com isso, a interoperabilidade entre Ruby e Java é uma característica marcante dessa tecnologia e, como a maioria das bibliotecas para processamento de linguagem natural é escrita em Java, Jruby se mostrou adequado ao desenvolvimento do *framework* proposto. Além disso, a utilização da máquina virtual Java garante um desempenho melhor do que a implementação em linguagem C.

4.4.2. OpenNLP

OpenNLP é uma biblioteca para processamento de linguagem natural baseada em aprendizado de máquina e desenvolvida pela fundação Apache[29]. Essa biblioteca dá suporte a realização de tarefas básicas de NLP, possibilitando o desenvolvimento de tarefas mais complexa. Dentre as funcionalidades fornecidas pela biblioteca OpenNLP estão:

1. *Tokenização*: processo de divisão de uma sequência de caracteres em unidades menores chamadas *tokens*, normalmente consistindo de palavras, números e pontuações;
2. Segmentação de sentenças: consiste em dividir textos em sentenças, normalmente delimitadas por pontuação;
3. Rotulação de partes do discurso: descoberta de rótulos referentes as partes do discurso dos termos de uma sentença;

[29] http://www.apache.org/

4. Reconhecimento de entidades nomeadas: detecta nomes presentes em sentenças ou grupos de *tokens*. O modelo depende dos tipos de entidades e do idioma para os quais é treinado;
5. Categorização de documentos: classifica documentos em categorias pré-definidas utilizando um modelo de entropia máxima.
6. Rotulação de partes do discurso: utiliza um modelo probabilístico para rotular as palavras das sentenças com as suas respectivas classes gramaticais.

Além da utilização dos modelos pré-treinados com vários corpus disponibilizados pela comunidade de NLP, o OpenNLP fornece uma API para treinamento dos seus modelos utilizando corpus definidos pelo usuário.

O OpenNLP é utilizado nos módulos de pré-processamento e de NLP do *framework*. No módulo de pré-processamento, as classes *OpenNLPTokenizer* e *OpenNLPSentenceExtractor* englobam as funcionalidades de *tokenização* e divisão do texto em sentenças do OpenNLP utilizando modelos treinados para o idioma inglês. No módulo NLP, a classe *OpenNLPTagger* engloba a funcionalidade de rotulação de partes do discurso fornecida pelo OpenNLP, também utilizando modelos treinados para inglês.

O OpenNLP foi selecionado por possuir uma comunidade ativa, organizada pela fundação Apache, e por reunir diversos aspectos básicos de um processo de ER, como divisão em *tokens*, reconhecimento de sentenças e análise de partes do discurso, em uma única ferramenta.

4.4.3. Analisador de Dependências de Stanford

O analisador de dependências de Stanford é um analisador probabilístico, implementado em Java, capaz de descobrir a estrutura gramatical das sentenças utilizando gramaticas livres de contexto. Esse tipo de analisador descobre a estrutura mais provável para novas sentenças baseado no conhecimento prévio adquirido a partir de sentenças manualmente anotadas.

Esse analisador foi escolhido para implementação da funcionalidade de análise de dependências, fornecida pelo módulo NLP do *framework*, porque possui uma comunidade ativa que vêm realizando trabalhos constantes relacionados principalmente ao desempenho e a internacionalização. Atualmente, além do idioma inglês, o analisador de Stanford vêm sendo

adaptado, entre outros, para os idiomas chinês, alemão, Italiano, Búlgaro e Árabe.

A classe *StanfordParser* é a responsável por englobar a funcionalidade de análise de dependências do analisador de Stanford. Essa classe é utilizada para o idioma inglês mas possui um atributo que especifica a gramática a ser utilizada, permitindo a aplicação do analisador para outros idiomas.

4.4.4. Biblioteca para Extração de Radicais

A extração de radicais é uma funcionalidade fornecida pelo módulo de pré-processamento, através da classe *RadicalExtractor* (Figura 16). Essa técnica permite a redução das variações de palavras relacionadas, como “observe”, “observador”, “observatório”, para o seu radical, “observ”.

Uma versão do algoritmo de Porter para realização dessa tarefa já se encontra implementada para linguagem Ruby na biblioteca *stemmify*[30], desenvolvida por Ray Pereda. Uma vez instalada, a funcionalidade é disponibilizada para todas as strings através do método *stem,* acrescentado a classe String da API do Ruby. O quadro 6 mostra alguns exemplos da utilização dessa biblioteca.

```
irb(main):001:0> require 'rubygems'
=> true
irb(main):002:0> require 'stemmer'
⇒True
irb(main):005:0> 'observer'.stem
=> "observ"
irb(main):006:0> 'observing'.stem
=> "observ"
irb(main):007:0> 'observe'.stem
=> "observ"
```

Quadro 6 – Exemplos de uso da biblioteca Stemmify.

A implementação padrão do método *extractRadical* delega para o método *stem,* adicionado a classe String do Ruby.

[30] https://github.com/raypereda/stemmify

4.4.5. ActiveRDF

ActiveRDF é um *framework* que realiza o mapeamento de um modelo de triplas RDF em um modelo orientado a objetos em linguagem Ruby, permitindo a manipulação dos dados RDF através de classes, objetos e atributos Ruby. O desenvolvimento desse *framework* foi incentivado pela grande quantidade de conjuntos de dados que vem sendo publicadas em RDF e pela necessidade de algo similar ao mapeamento objeto-relacional, oferecido por frameworks como Hibernate[31] e ADO.net[32], para manipulação de dados RDF.

ActiveRDF permite o acesso a classes, propriedades e recursos através de uma linguagem específica de domínio (DSL), baseada no modelo RDF sendo manipulado. O quadro 7 mostra um exemplo de uso da DSL fornecida pelo ActiveRDF para manipulação de um recurso identificado pela URI "*http://example.com/peter*".

```
peter = RDF::Resource.new("http://example.com/peter")

print peter.foaf::name
Peter

peter.foaf::age = 25

print peter.foaf::age
25
```

Quadro 7 – Exemplo de utilização da DSL fornecida pelo ActiveRDF.

Além de permitir a manipulação de recursos através de uma DSL, o ActiveRDF fornece meios para consultas em conjuntos RDF locais ou remotos, disponibilizados em arquivos locais ou *SPARQL endpoints*. Além de suporte a consultas comuns, o ActiveRDF fornece um esquema de federação, onde as consultas são espalhadas pelos conjuntos de dados registrados e os resultados são integrados e retornados. A função do ActiveRDF no *framework* proposto é fornecer meios para consultas aos *endpoints* registrados. O quadro 8 mostra um exemplo de consulta uttilizando a *Query engine* do ActiveRDF:

[31] http://www.hibernate.org/

[32] http://www.macoratti.net/ado_net1.htm

```
Query.new.distinct(:s, :o).where(:s, "http://dbpedia.org/ontology/parent", :o).execute
```

Quadro 8 – Exemplo de consulta por sujeitos e objetos para a relação *"http://dbpedia.org/ontology/parent"*.

Essa consulta retorna todos os pares de sujeitos e objetos relacionados pela propriedade "*parent*", presente na ontologia do DBpedia.

O ActiveRDF é utilizado pela classe *RDFRepository* e sua escolha vem principalmente dos casos de sucesso em trabalhos anteriores, como no Explorator[33] e no Synth [Bonfim, 2011].

4.5. Aplicação do *Framework* na Wikipedia e no DBpedia em Inglês

Inicialmente, foi instalada uma versão local da Wikipedia de 12/01/2012[34]. Somente as revisões mais atuais das páginas foram consideradas. Após isso, o *framework* foi configurado para atuar no modo de mineração de exemplos. A figura 31 mostra a parte do arquivo de configuração referente ao modo de mineração.

```
minning:
  enabled: 'yes'
  target_relations: [genre, award, instrument, associatedMusicalArtist, composer, album]
  namespace: "http://dbpedia.org/ontology/"
  preprocessing:
    tokenizer: OpenNLPTokenizer
    text_cleaner: WikipediaTextCleaner
    radical_extractor: DefaultRadicalExtractor
    sentence_extractor: OpenNLPSentenceExtractor
  filters: [SubjectMentionFilter, ObjectMentionFilter]
  sentence_db:
    db: mysql
    location: "jdbc:mysql://localhost:3306/sentence_db?user=root&password=db@dm348"
  relations_db:
    db: sparql
    location: "http://dbpedia.org/sparql"
  corpus:
    class: WikipediaRepository
    url: http://localhost/mediawiki/index.php
    lang: en
```

Figura 31 – Parte do arquivo de configuração referente ao modo de mineração.

A habilitação de um modo de operação é realizada pelo valor da chave "*enabled*". Em seguida, o atributo "*target_relations"* indica um vetor de relações para as quais se deseja encontrar exemplos de sentenças. Para fins de

[33] http://www.tecweb.inf.puc-rio.br/explorator/

[34] http://dumps.wikimedia.org/enwiki/20111201/

simplificação, a figura 31 mostra apenas alguns exemplos de relações utilizadas. A quantidade real é de 90 relações. Algumas dessas relações são descritas a seguir:

1. *genre:* essa propriedade relaciona um artista ou uma obra ao seu respectivo gênero.
2. *award:* relaciona um artista ou um trabalho a uma premiação.
3. *instrument:* relaciona um artista musical a um instrumento
4. *associatedMusicalArtist*: relaciona artistas musicais associados
5. *composer*: relaciona uma obra ao artista que a compôs.
6. *album*: relaciona uma música ao álbum que a contém.

Os módulos utilizados no modo de mineração são o de pré-processamento e o de filtragem, onde, os seus pontos de extensão são configurados nas chaves "*preprocessing*" e "*filters*".

O módulo de pré-processamento é configurado para utilizar o *tokenizador* fornecido pela biblioteca OpenNLP. A remoção das estruturas irrelevantes é realizada pela classe *WikipediaTextCleaner*, configurada na chave "*text_cleaner*". A extração de radicais e a divisão dos artigos em sentenças são realizadas pelas classes *OpenNLPSentenceExtractor* e *WikipediaTextCleaner*, configurados respectivamente nas chaves "*sentence_extractor*" e *"radical_extractor"*.

Dois filtros são aplicados ao conjunto de sentenças, um deles identifica a presença de uma menção ao sujeito do artigo (*SubjectMentionFilter*), e o outro identifica a presença de uma menção ao objeto da relação (*ObjectMentionFilter*). Os filtros são configurados na chave "*filters*".

São utilizados dois repositórios, um para armazenar as sentenças encontradas e um repositório que realiza consultas ao *SPARQL endpoint* do DBpedia. O repositório *SPARQL*, implementado pela classe *SparqlRepository* do módulo Repositório, fornece o acesso ao modelo de dados, onde as relações entre as entidades estão previamente estabelecidas em RDF e descritas em uma ontologia. Os dois repositórios são configurados nas chaves "*sentence_db*" e "*relations_db*". A última configuração corresponde ao corpus, que é um repositório de artigos em linguagem natural. Nesse caso, a classe responsável é a *WikipediaRepository* que recupera artigos utilizando a instalação local da Wikipedia inglesa, disponibilizada na url "*http://localhost/mediawiki/index.php*".

A execução do *framework* no modo de mineração utilizando a configuração descrita foi capaz de montar, automaticamente, um banco de sentenças

constituído de 90 relações descritas na ontologia do DBpedia, totalizando mais de 100.000 exemplos.

Comparando com o estado da arte atual na área de ER, no que diz respeito a quantidade de relações, a cobertura da abordagem de mineração porposta neste trabalho é superada apenas pelo método em [Wang et al., 2011], com um banco de 7.000 relações mineradas diretamente das caixas de informação da Wikipédia. O problema com a abordagem em [Wang et al., 2011] é justamente o mesmo problema enfrentado pelo método genérico de extração de informação do projeto DBpedia, onde, relações similares são registradas nas caixas de informação pelos editores sem uma consulta prévia.

Após a montagem de um conjunto de sentenças com boa cobertura no que diz respeito as quantidades de relações e de exemplos por relação, é possível aplicar os extratores de características e gerar os conjuntos de treinamento e avaliação dos classificadores. Nessa etapa, foram gerados seis conjuntos considerando diferentes combinações de tipos de características, onde, todos os conjuntos possuem um total de 90 relações com 200 instâncias por relação. A tabela 8 mostra as combinações de características de cada conjunto e as classes do módulo de extração de características utilizadas para gerá-los.

Conjuntos	Classes Utilizadas
Léxico (L)	LexicalExtractor
Menor Caminho (SP)	ShortestPathExtractor
Tipos (T)	DBpediaOntologyTypesExtractor
L + SP	LexicalExtractor + ShortestPathExtractor
L + T	LexicalExtractor + DBpediaOntologyTypesExtractor
L + SP + T	LexicalExtractor + DbpediaOntologyTypesExtractor + ShortestPathExtractor

Tabela 8 – Conjuntos de treinamento e classes que os geraram.

A figura 32 mostra o trecho correspondente ao modo de geração do conjunto de treinamento no arquivo de configuração para um dos casos apresentados na tabela 8.

```
dataset_generation:
  enabled: 'yes'
  target_relations: [riverMouth, award, musicalBand, college, birthPlace, country, isPartOf, w
  max_examples_quantity: 200
  namespace: "http://dbpedia.org/ontology/"
  filters: []
  feature_extractors: [LexicalExtractor, ShortestPathExtractor, DBpediaOntologyTypesExtractor]
  dataset_format: arff
  dataset_name: l_sp_t
  dataset_folder: ./trainning_data
  sentence_db:
    db: mysql
    location: "jdbc:mysql://localhost:3306/sentence_db?user=root&password=db@dm348"
```

Figura 32 – Trecho do arquivo de configuração referente ao modo de geração dos conjuntos de treinamento.

Algumas configurações apresentadas na figura 32 são similares às do modo de mineração. As chaves específicas são:

- *feature_extractors*: usada para especificar as classes dos extratores de características desejados
- *dataset_format:* usada para especificar o formato de serialização desejado. A classe correta do módulo *DatasetGeneration* é instanciada com base no valor dessa chave
- *dataset_name:* usada para especificar o nome do arquivo do conjunto de treinamento gerado.
- *dataset_folder:* usada para especificar o diretório que irá conter o arquivo do conjunto de treinamento. Nesse caso, é o diretório *trainning_data,* localizado na raiz do *framework.*

Os arquivos gerados e salvos no diretório *trainning_data* são utilizados na próxima etapa de treinamento e avaliação de classificadores, onde, a configuração é mostrada na figura 33.

```
trainning:
  enabled: 'yes'
  dataset_folder: ./trainning_data
  dataset_name: l_sp_t
  dataset_format: csv
  classifier_folder: ./ml_models
  classifier: WekaNaiveBayes
  cross_validation_folds: 10
  classifier_settings: ''
```

Figura 33 – Trecho do arquivo de configuração referente ao treinamento e a avaliação de classificadores.

Algumas chaves de configuração se repetem da etapa de geração do conjunto de exemplos, como a "*dataset_folder*" e "*dataset_name*". As chaves de configuração específicas desse modo de execução são as seguintes:

- *classifier_folder*: diretório onde será salvo o modelo após o treinamento
- *classifier*: classe do classificador a ser utilizado.
- *cross_validation_folds:* número de partes da validação cruzada
- *classifier_settings*: string de parâmetros do classificador.

Após a etapa de configuração, o framework pode ser executado através do módulo *RelationExtraction,* implementado no arquivo *relation_extraction.rb*, como mostra a figura 34.

```
C:\Users\tnunes\Documents\Dissertacao\re-framework>jruby -J-Xmx6g relation_extractor.rb
Loading parser from serialized file lib/stanford-parser-2012-03-09/englishPCFG.ser.gz ... done
http://dbpedia.org/ontology/country
http://dbpedia.org/ontology/isPartOf
http://dbpedia.org/ontology/city
http://dbpedia.org/ontology/location
http://dbpedia.org/ontology/starring
http://dbpedia.org/ontology/team
http://dbpedia.org/ontology/family
http://dbpedia.org/ontology/region
http://dbpedia.org/ontology/battle
http://dbpedia.org/ontology/writer
http://dbpedia.org/ontology/builder
http://dbpedia.org/ontology/formerBandMember
http://dbpedia.org/ontology/director
http://dbpedia.org/ontology/series
http://dbpedia.org/ontology/managerClub
```

Figura 34 – Execução do *framework* através do arquivo *relation_extraction.rb* com o comando "*jruby*".

4.6. Aplicação do *Framework* na Wikipedia e no DBpedia em Português

Além da aplicação para o idioma Inglês, selecionamos também a Wikipedia e o DBpedia em português para demonstrar a utilização do *framework* em outro idioma.

Para executar o *framework* no modo de mineração com os conjuntos de dados em português foi utilizada a configuração presente na figura 35. Os valores dos primeiros atributos de configuração são similares aos utilizados para os conjuntos em inglês, variando apenas as relações alvo no atributo "*target_relations*". Como *tokenizador,* foi utilizado o *SimpleTokenizer*, que divide as sentenças em *tokens* utilizando como delimitador espaços vazios " ". O extrator de radicais foi implementado especificamente para português e utiliza a biblioteca PTStemmer[35] em java. Essa biblioteca fornece, entre outras, uma implementação do algoritmo de Porter adaptada para português. As demais diferenças estão relacionadas as configurações dos repositórios e do corpus de

[35] http://code.google.com/p/ptstemmer/

artigos. A figura 35 apresenta o trecho do arquivo de configuração referente ao modo de mineração.

```
minning:
  enabled: 'yes'
  target_relations: [country, isPartOf, city, location, starring, team, family]
  namespace: "http://dbpedia.org/ontology/"
  preprocessing:
    tokenizer: DefaultTokenizer
    text_cleaner: WikipediaTextCleaner
    radical_extractor: PortugueseRadicalExtractor
    sentence_extractor: OpenNLPSentenceExtractor
  filters: [ObjectMentionFilter]
  sentence_db:
    db: mysql
    location: "jdbc:mysql://localhost:3306/pt_sentences_db?user=root&password=db@dm348"
  relations_db:
    db: sparql
    location: "http://pt.dbpedia.org/sparql"
  corpus:
    class: WikipediaRepository
    url: http://pt.wikipedia.org/wiki
    lang: pt
```

Figura 35 – Trecho do arquivo de configuração para mineração de exemplos em português.

Afim de armazenar sentenças em português, foi criado um novo banco de sentenças, o *pt_sentences_db*, configurado na chave "*location*" de *"sentence_db"*. O local do *SPARQL endpoint* do DBpedia foi alterado para utilizar a versão em português, como mostra o valor da chave "*location*" em "*relations_db*". Por último, o repositório de artigos foi configurado para utilizar a Wikipedia em português e a linguagem foi alterada de inglês (*en*) para português (pt) na chave "*lang*" da configuração do corpus. A execução do *framework* com essa configuração recuperou um total de 70 relações com mais de 110.000 exemplos.

De forma a acrescentar um extrator sintático para português, foram criadas duas classes, como mostra a figura 36.

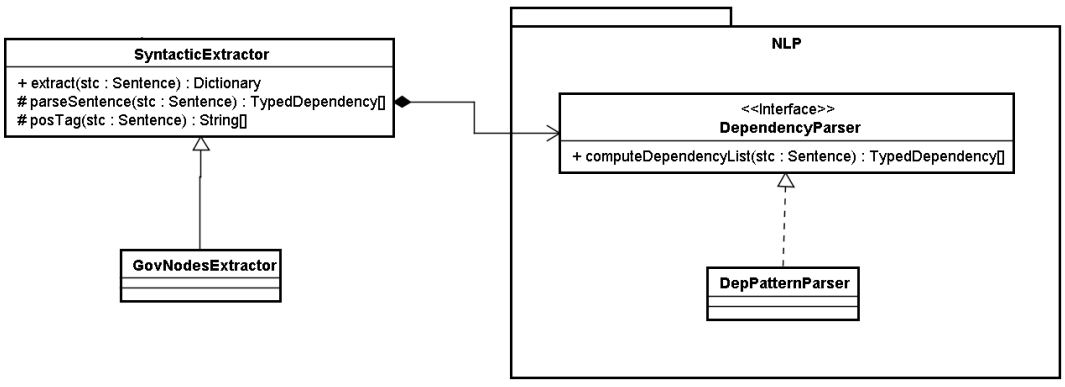

Figura 36 – Classes *GovNodeExtractor* e *DepPatternParser*.

A classe *GovNodesExtractor* é responsável por extrair somente os nós governantes das dependências como características sintáticas. A classe *DepPatternParser* engloba chamadas ao analisador multilingue DepPattern, que realiza a análise parcial de dependências para sentenças em português. Com isso, a geração do conjunto de exemplos possui uma configuração bem parecida com a demonstrada para inglês, com a diferença da classe *GovNodesExtractor*. As figuras 37 e 38 mostram as configurações do modo de geração do conjunto de exemplos e do módulo NLP.

```
22  dataset_generation:
23    enabled: 'yes'
24    target_relations: [country, isPartOf, city, location, starring, team, family]
25    max_examples_quantity: 500
26    namespace: "http://dbpedia.org/ontology/"
27    filters: []
28    feature_extractors: [LexicalExtractor, GovNodesExtractor, EntityTypesExtractor]
29    dataset_format: arff
30    dataset_name: l_sp_t
31    dataset_folder: ./trainning_data
32    sentence_db:
33      db: mysql
34      location: "jdbc:mysql://localhost:3306/pt_sentence_db?user=root&password=db@dm348"
35
```

Figura 37 – Trecho de configuração para geração do conjunto de exemplos em português.

```
37  nlp:
38    parser: DepPatternParser
39    grammar: ./lib/DepPattern-2.1/grammars/grammar-devel-pt.dp
```

Figura 38 – Trecho de configuração do módulo NLP para utilização da classe *DepPatternParser* e uma gramática para português.

As configurações para a etapa de treinamento e avaliação de classificadores para português e a execução do *framework* ocorre da mesma forma como foi apresentada para inglês (seção 5.5).

5
Trabalhos Relacionados

Este capítulo apresenta uma discussão sobre os trabalhos relacionados na área de extração de relações e *frameworks* propostos. Basicamente, os métodos de extração de relações estudados podem ser agrupados em três tipos: métodos baseados em regras [Nakayama et al., 2008], [Garcia e Gamallo, 2011]; métodos baseados em *kernel* [Bunescu e Mooney, 2005], [Nguyen, Matsuo e Ishizuka, 2007], [Wang et al., 2011] e métodos baseados em características [Kambhatla, 2004], [Yan et al., 2009]. Além disso, uma categoria de métodos para ER que vem sendo explorada em trabalhos mais recentes é a de "supervisionamento distante" [Mintz et al., 2009] [Chan e Roth, 2010] [Wang et al., 2011], que procura contornar limitações dos métodos de ER integrando conhecimento obtido de diferentes bases.

5.1.
Métodos Baseados em Regras

Os métodos baseados em regras aplicam uma série de regras linguísticas ao texto para capturar padrões de relações. Em [Nakayama et al., 2008] é apresentada uma abordagem para extração de relações semânticas através da mineração de textos de artigos da Wikipedia. Nesse trabalho, os artigos da Wikipedia são vistos como um conjunto de sentenças indicando relações entre conceitos. Com isso, é proposto um método para extrair e analisar essas sentenças em três etapas: pré-processamento, análise de partes do discurso e extração de termos principais.

A etapa de pré-processamento é responsável pela remoção de marcações wiki do texto, divisão em sentenças por períodos separados por (".") e por uma rotulação parcial dos termos. Nessa etapa, são preservadas as marcações de elos, pois estas são utilizadas para identificar relações entre páginas.

A sub etapa de rotulação parcial adiciona rótulos de *substantivo* para termos entre aspas e termos que aparecem como nomes de elos. Esse processo tem o objetivo de otimizar o tempo requerido pelo analisador de partes do discurso.

Na etapa de análise de partes do discurso o analisador NLP de Stanford é aplicado às sentenças parcialmente rotuladas para obter uma estrutura de árvore sintática. Essa estrutura é comparada a regras predefinidas para tipos específicos de relações. Três regras foram definidas para este trabalho:

1. A regra normal "*é-um*": (NP ...) (VP (VBZ/VBD/VBP ...)) (NP ...));
2. A regra de subordinação "*é-parte-de*": (NP ...) (VP (NP (NP ...) (PP (IN ...) ...));
3. A regra passiva "*nascido-em*": (VP (VBZ ...) (VP (VPN ...) ...)).

Os rótulos e seus siginificados estão descritos na tabela 9.

Rótulo	**Descrição**
NN	Substantivo singular ou coletivo
NNS	Substantivo no plural
NNP	Substantivo próprio singular
NNPS	Substantivo próprio no plural
NP	Sintagma nominal
VB	Verbo no infinitivo
VBD	Verbo no passado
VBZ	Verbo na terceira pessoa do singular
VBP	Verbo em outro tempo que não seja a terceira pessoa do singular do presente
VP	Sintagma verbal
JJ	Adjetivo
CC	Conjunção coordenativa
IN	Conjunção subordinativa

Tabela 9 - Rótulos e descrições.

A figura 39 mostra o processo completo de extração de relações com a estrutura gerada a partir da sentença parcialmente rotulada "*[[Madrid]]/NN is the [[capital]]/NN and largest city of [[Spain]]/NN.*"

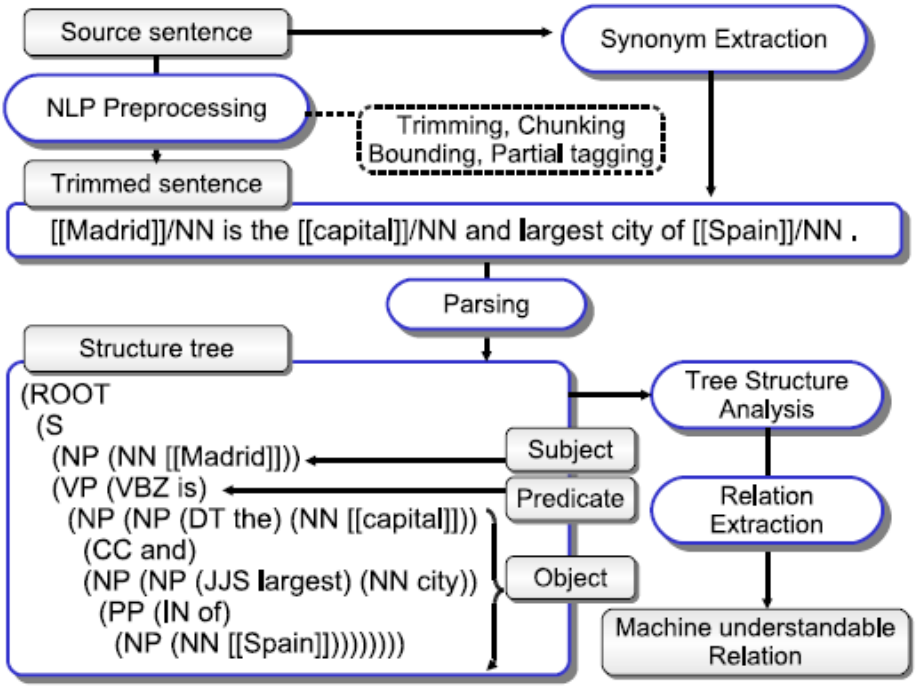

Figura 39 - Processo de extração de relações apresentado em [Nakayama et al., 2008].

A etapa de extração de termos principais aplica uma heurística para identificar os termos principais no sujeito e no objeto da sentença. A estratégia consiste em selecionar os últimos substantivos singulares presentes no sujeito e no objeto.

5.1.1. Extração de Relações em Textos em Português

Extração de relações de textos em português ainda é uma área de pesquisa recente e que somente começou a obter resultados satisfatórios nos últimos anos. Isso se deve principalmente ao fato de que essa área depende de um esforço de avaliação conjunta e do amadurecimento de ferramentas para processamento de linguagem natural para Português, como analisadores de dependência liguística e analisadores de partes do discurso.

Apesar de haver avaliações conjuntas para sistemas de extração de informação para o idioma Inglês desde os anos 80, como a MUC[36] (*Message Understanding Conference*) e a ACE[37] (*Automatic Content Extraction*), a única iniciativa para língua portuguesa teve início somente em 2008, como uma seção

[36] http://www.itl.nist.gov/iaui/894.02/related_projects/muc/

[37] http://www.itl.nist.gov/iad/mig/tests/ace/

específica da conferência HAREM[38] (Avaliação e Reconhecimento de Entidades Mencionadas).

O trabalho realizado em [Garcia e Gamallo, 2011] apresenta uma abordagem baseada em regras para Extração de Relações que tem o objetivo de amenizar a baixa cobertura inerente a simplificação de estruturas linguísticas. Para isso, são aplicadas diversas técnicas de compressão de sentenças utilizando-se análise de dependência parcial para remover elementos satélites. Para obter regras de extração de alta qualidade, esse trabalho utiliza uma fonte semi-estruturada de exemplos: as caixas de informação dos artigos da Wikipedia.

Inicialmente, são obtidos grandes conjuntos de pares de entidades que compartilham uma relação semântica. Após isso, são coletadas sentenças que contém instâncias da relação desejada. Então, essas sentenças são transformadas em padrões lexico-sintáticos e generalizados através de um algoritmo de obtenção dos *strings* mais longos comuns entre esses padrões. Por último, esses padrões são convertidos em regras semânticas que, por sua vez, são adicionadas à uma gramática de dependência.

Os conjuntos de pares de entidades obtidos são instâncias das relações *hasBirthPlace* (local de nascimento) e *hasProfession* (profissão). Esses pares são obtidos através da mineração das caixas de informação da Wikipedia e são utilizados para selecionar as sentenças dos textos. As sentenças, então, passam por um processo de análise de partes do discurso e de dependência parcial. A figura 40 mostra o grafo de dependências para a sentença

"*Nick Cave was born in the small town of Warracknabeal.*"

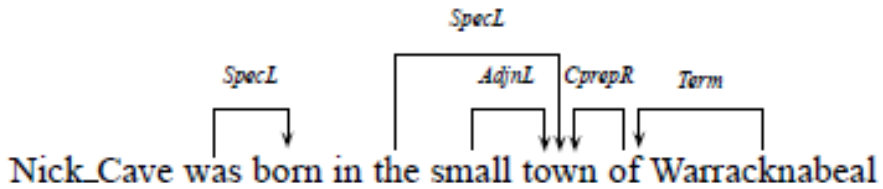

Figura 40 - Dependências para a sentença "*Nick Cave was born in the small town of Warracknabeal*" (fonte: [Garcia e Gamalo, 2011]).

Na figura 40, o início do arco direcionado representa um termo dependente. O termo que governa a dependência é aquele apontado pelo arco.

[38] http://linguateca.dei.uc.pt/harem/encontro/EncontroSegundoHAREM.html

Somente os termos não dependentes são mantidos para formar as regras. Dessa forma, a sentença da figura 40 é resumida na seguinte forma:

<**Nick_Cave** born in **town**>

Após a análise de dependência, as menções das entidades são substituídas por rótulos, como **X** ou **Y**, formando os padrões. Alguns exemplos de padrões obtidos em [Garcia e Gamallo, 2011] são:

- <**X** nascer_V em_PRP **Y**>
- <**X** nascer_V em_PRP a_DA cidade_N de_PRP **Y**>
- <**X** nascer_V em_PRP NP Fc **Y**>
- ...

O padrão obtido pela fase de generalização através de um algoritmo para calcular a maior *string* comum é <**X** nascer_V em_PRP **Y**>**.** Finalmente, são geradas regras compiladas em uma gramática para serem aplicadas a um corpus para obter triplas na forma "**X** relação **Y**".

A avaliação apresentada em [Garcia e Gamallo, 2011] para português foi realizada na descoberta das relações *hasBirthPlace* e *hasProfession* em dois corpus distintos, a Wikipedia Português e um corpus jornalístico da Público, que é um jornal português de propósito geral. Nessa avaliação são abordados somente valores referentes a precisão do método, que é de 90 % na Wikipedia e 84 % no corpus jornalístico. Não são comentados valores de *recall* nem de *F1*.

Um caso específico de extração de relações são as extrações de citações. Esse problema consiste em reconhecer citações em textos e relacioná-las aos seus respectivos autores. Considere o exemplo a seguir:

"Ou a gente vai para perto ou então chama os amigos para casa mesmo", diz Anna Sofia.

Um processo de extração de citações deve ser capaz de reconhecer a citação e relacioná-la com a sua autora Anna Sofia:

"Ou a gente vai para perto ou então chama os amigos para casa mesmo"$_1$, diz Anna Sofia$_1$.

O trabalho realizado em [Fernandes, 2012] aborda esse problema para o idioma Português utilizando os algoritmos de Aprendizado de Transformações Guiado por Entropia e Perceptron Estrutural. O melhor resultado é atingido pelo Perceptron Estrutural com 76.80% de medida F1. Os algoritmos foram treinados

e avaliados no corpus GloboQuotes, gerado a partir do portal de notícias em Português globo.com.

Algumas tarefas da área de processamento de linguagem natural podem ser enxergadas como um problema de extração de relações. Com isso, a ferramenta RelHunter [Fernandes, Milidiú e Rentería, 2010] foi desenvolvida para extrair informações estruturadas de textos através de aprendizado de máquina, onde, a estrutura alvo é modelada como relações entre entidades.

Como exemplo, considere o problema de extração de cláusulas. Essa tarefa consiste em reconhecer sequências de palavras que contêm um sujeito e um predicado. Em [Fernandes, Milidiú e Rentería, 2010] esse problema é modelado como a descoberta de uma relação entre pares de entidades (t_1, t_2), onde, t_1 e t_2 são *tokens* delimitadores das cláusulas.

Em [Fernandes, Milidiú e Rentería, 2010], essa mesma abordagem foi adotada para outras quatro tarefas da área de PLN para os idiomas Inglês e Português: reconhecimento de frases, detecção de elementos que expressam incerteza em textos, extração de citações e análise de dependências. RelHunter utiliza a técnica de Aprendizado Baseado em Transformações Guiado por Entropia (ETL) mostrando resultados competitivos em todas as tarefas e, atingindo o estado-da-arte em três corpora: reconhecimento de frases em Português, identificação de cláusulas em Português e extração de citações para Inglês

5.2. Métodos Baseados em Características

Os trabalhos apresentados a seguir utilizam modelos de aprendizado de máquina para realizar a extração de relações, variando principalmente nas técnicas escolhidas, na etapa de filtragem de sentenças relevantes e na extração de características.

Em [Kambthala, 2004] foram construídos modelos de Entropia Máxima para extração de relações, onde são combinadas características léxicas, sintáticas e semânticas.

Os tipos de características extraídas das sentenças são descritos a seguir:

- Palavras e indicadores de posição: são utilizadas todas as palavras com rótulos identificando se são menções de entidades ou palavras que se encontram entre as menções;
- Tipos das entidades envolvidas;

- Classe gramatical das menções: se são substantivos, pronomes ou nominais;
- Dependência: palavras com suas indicações de dependência e partes do discurso;
- Árvore sintática: são extraídos apenas os caminhos que ligam as duas menções.

O exemplo descrito em [Kambthala, 2004] utiliza a sentença

"*been the* **chairman** *of its* **board ..."**.

Neste exemplo, as características extraídas são:

- **Palavras:** chairman/*m1*, of/*b1*, its/*b2*, board/*m2.*
- **Tipos das Entidades:** $PESSOA_{m1}$, $ORGANIZACAO_{m2}$
- **Classe gramatical das menções:** $NOMINAL_{m1}$, $NOMINAL_{m2.}$
- **Dependência:** o grafo de dependência é mostrado na figura 41.

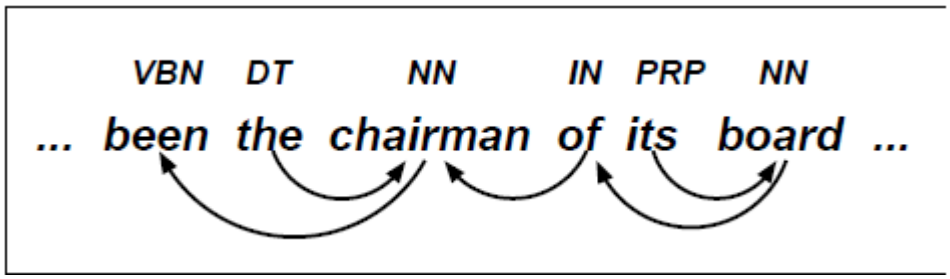

Figura 41 - grafo de dependência e partes do discurso para a sentença "*been the* **chairman** *of its* **board ..."** (fonte: [Kambhtala, 2004]).

- **Árvore sintática:** O caminho derivado da árvore sintática é mostrado na figura 42.

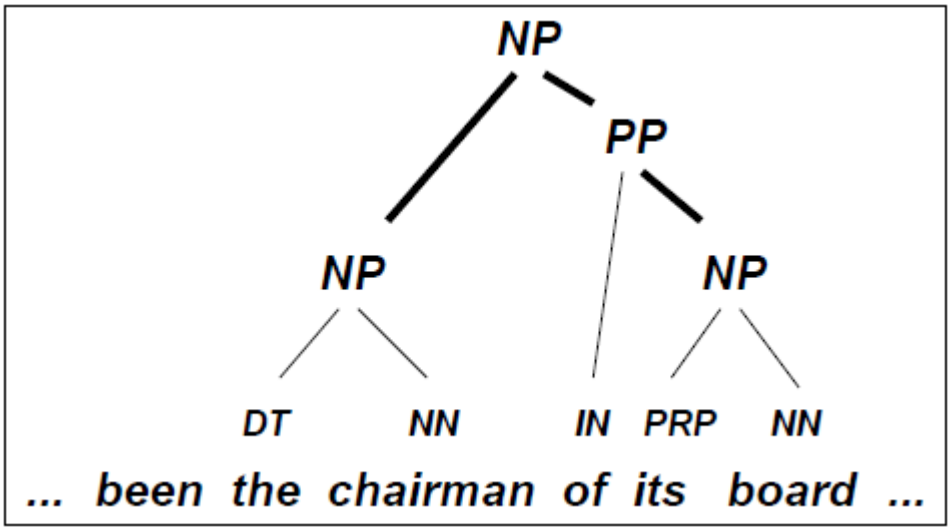

Figura 42 - Caminho da menção "*chairman*" à menção "*board*" extraído da árvore sintática (fonte: [Kambhtala, 2004]).

Os experimentos realizados em [Kambhtala, 2004] utilizam o corpus para avaliação desenvolvido pela *Automatic Content Extraction* (ACE, 2004). O conjunto de dados fornecido pela ACE foi separado em conjuntos de treinamento e de desenvolvimento. O conjunto de treinamento possui 300K palavras e 9.752 instâncias. O conjunto de desenvolvimento possui 46K palavras e 1.679 instâncias. A aplicação do modelo treinado ao conjunto de desenvolvimento resultou em 63,5 % de precisão, 45,2 % de *recall* e 52,8 % de *F1*.

Os métodos apresentados até o momento são supervisionados, onde geralmente, é necessário um esforço considerável para montagem do conjunto de treinamento [Yan et al., 2009]. Com o objetivo de contornar essa dificuldade, foi elaborada uma abordagem não supervisionada em [Yan et al., 2009] utilizando o algoritmo k-means para agrupar pares de entidades que compartilham uma mesma relação binária.

Em [Yan et al., 2009] são utilizados dois tipos de características. Padrões de superfície e padrões de dependência. Os padrões de superfície são coletados da Web através de consultas na engine de busca Google[39] contendo um determinado par de entidades. A partir dos trechos das páginas web retornados pelo Google, são extraídos grupos de palavras, como padrões de superfície, e uma lista de termos ordenada pelo grau de entropia de cada termo. Os padrões de dependência exploram a boa qualidade das sentenças da Wikipedia e são compostos dos sub-caminhos do menor caminho no grafo de dependência entre

[39] www.google.com

as duas entidades. A figura 43 ilustra os principais componentes do extrator de relações apresentado em [Yan et al., 2009].

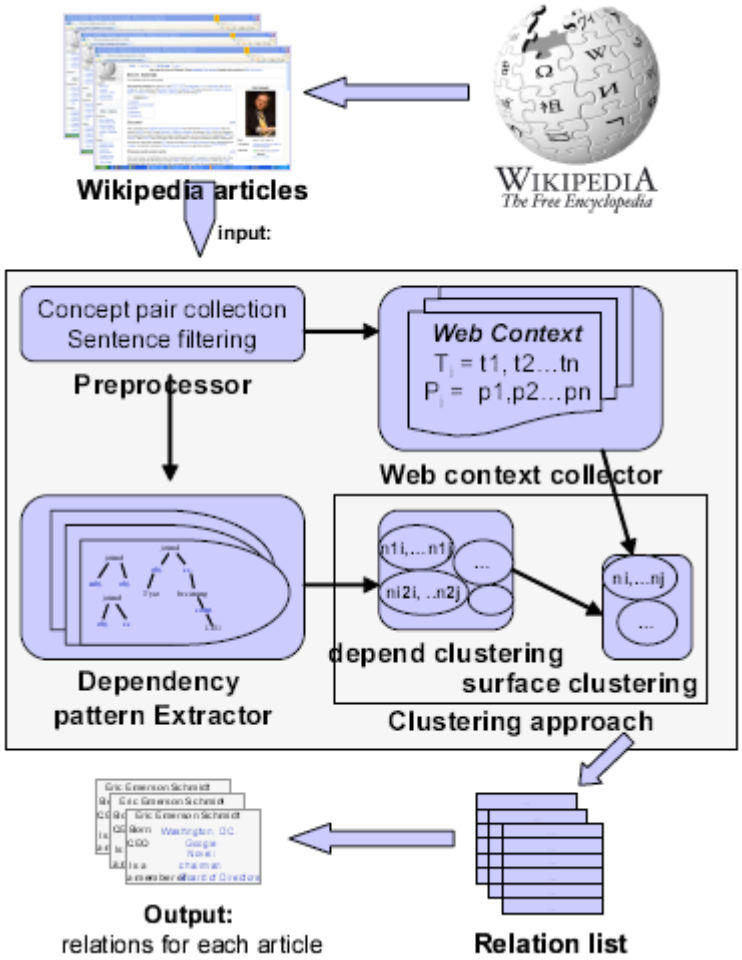

Figura 43 - Módulos de extração de relações apresentado em [Yan et al., 2009] (fonte: [Yan et al., 2009]).

O extrator de relações, apresentado na figura 43, recebe como entrada artigos da Wikipedia e retorna uma lista de relações entre entidades. Inicialmente, esses artigos passam por um pré-processador (*preprocessor*), onde são extraídos pares de entidades e as suas respectivas sentenças dos artigos. Após a obtenção dos pares de entidades e das sentenças, são extraídos padrões de superfície (*Web Context Collector*) através de consultas na Web que coletam indícios da presença de uma relação. Paralelamente a obtenção dos padrões de superfície, são gerados padrões de dependência (*Dependency Pattern Extractor*) através da análise sobre o grafo de dependência das sentenças. Esses dois padrões são utilizados no processo de agrupamento em

duas etapas com a técnica k-means (*Clustering Approach*). Como saída, têm-se uma lista de relações para cada artigo de entrada (*Relation List* e *Output*).

O agrupamento das entidades com base nos padrões de superfície e de de dependência é realizado em duas etapas. Primeiramente é realizado o agrupamento utilizando os padrões de dependência para obter uma boa precisão, porém, esse tipo de padrão não gera grupos com boa cobertura [Yan et al., 2009]. A função de distância utilizada nessa etapa é a quantidade de sub-caminhos do grafo de dependência que se sobrepõem. Em seguida, é realizado o agrupamento por padrões de superfície para aumentar a cobertura dos grupos gerados pelo agrupamento por padrões de dependência. A função de distância utilizada nessa etapa é baseada na distância de edição *Leveinchtein* dos padrões de superfície.

Para realizar a avaliação, foram extraídos 7.310 pares de conceitos que formam um total de 18 grupos, onde, cada grupo equivale a uma relação de interesse. A cobertura foi medida como a fração obtida de pares corretos entre todos pares presentes no conjunto de dados. Dos 18 grupos, 15 foram identificados, gerando os resultados apresentados na tabela 10.

Tipo do Padrão	Instâncias	Precisão (%)	Cobertura (%)	F1 (%)
Dependência	1.127	84,29	13,00	22,53
Superfície	1.510	68,27	14,00	23,23
Combinados	2.314	75,63	23,94	36,37

Tabela 10 - Resultados obtidos pela abordagem apresentada em [Yan et al., 2009].

5.3. Métodos Baseados em Kernel

Com o objetivo de otimizar as características extraídas a partir do grafo de dependência dos termos, o trabalho realizado em [Bunescu e Mooney, 2005] procura descartar características desnecessárias extraindo apenas o menor caminho entre as duas entidades. A hipótese levantada é a de que o menor caminho no grafo captura de forma mais precisa os indícios de uma relação. [Bunescu e Mooney, 2005] apresenta um novo kernel baseado no menor caminho entre as duas entidades que supera o estado da arte dos métodos baseados em kernels sobre ávores de dependência no corpus da ACE, versão de 2002. A figura 44 mostra um exemplo de grafo de dependência para duas sentenças.

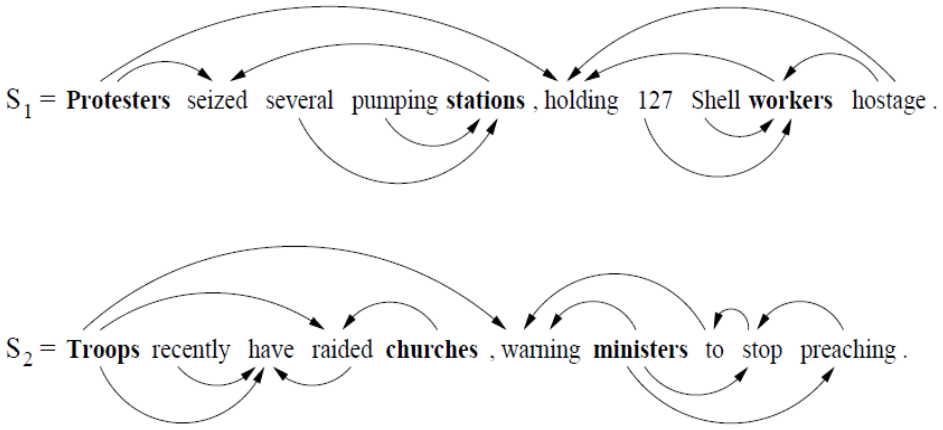

Figura 44 - Grafo de dependência para sentenças S1 e S2 (fonte: [Bunescu e Mooney, 2005]).

Considerando o exemplo da sentença S1 (figura 44), pode-se extrair o menor caminho não direcionado entre o termo "***Protesters"*** (Protestantes) e o termo "***stations"*** (Estações) como um exemplo positivo para a relação "*LocalizadoEm*", pois, o caminho "**Protesters** ⟶ seized ⟵ **stations**" captura bem a idéia de que os protestantes estão localizados nas estações.

Juntamente com o caminho mais curto no grafo de dependência, são utilizados os tipos das entidades, as partes do discurso de cada termo e as classes gramaticas gerais, como substantivo, adjetivo, verbo, etc. O vetor de características da sentença, então, é gerado pelo produto cartesiano das características de cada termo. O produto cartesiano das características da sentença S1 e o seu resultado são ilustrados pela figura 45.

Em [Bunescu e Mooney, 2005] foi utilizada a técnica *Support Vector Machines*[40] (SVM) , onde, a função *kernel* projetada calcula a quantidade de características comuns entre duas instâncias. Os experimentos foram realizados no corpus da ACE na versão para a avaliação de setembro de 2002. Foram utilizados 422 documentos para treinamento e 97 documentos para teste, resultando em 5 relações, 6.156 exemplos para treinamento e 1.490 exemplos para teste.

Foram utilizadas duas estratégias para realizar o aprendizado. A primeira estratégia treina um classificador multi-classes SVM para distinguir entre as 5 relações, adicionando mais uma classe para representar a não-existência de

[40] http://www.support-vector-machines.org/

uma relação. Essa estratégia obteve um resultado de 71,1% de precisão, 39,2% de *recall* e 50,5% de *F1*.

A segunda estratégia utiliza dois classificadores SVM, um classificador binário para detectar a presença de uma relação e um classificador multi-classes para distinguir entre as 5 relações. Essa estratégia obteve um resultado de 65,5% de precisão, 43,8% *recall* e 52,5% de *F1*.

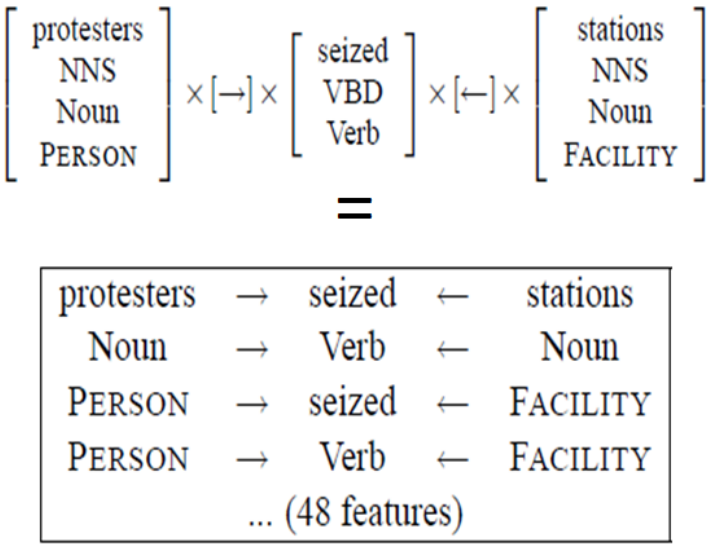

Figura 45 - Produto cartesiano das características de cada termo do caminho entre "***Protesters***" e "***stations***", apresentado na figura 44 (fonte: [Bunescu e Mooney, 2005]).

Com o objetivo de expandir o conjunto de exemplos e aumentar a cobertura dos métodos no que diz respeito a quantidade de relações, o trabalho apresentado em [Nguyen, Matsuo e Ishizuka, 2007] utiliza a Wikipedia como corpus. A hipótese defendida nesse trabalho contradiz a apresentada em [Bunescu e Mooney, 2005] alegando que o menor caminho no grafo de dependência não é eficaz na captura de indícios de relações em sentenças da Wikipedia. Portanto, a estratégia consiste em expandir o caminho entre as entidades primárias e secundárias integrando caminhos entre as entidades secundárias e as palavras-chave para uma determinada relação. Logo, a sub-árvore que contém as duas entidades é aumentada com o caminho entre a entidade secundária e uma palavra-chave. As palavras-chave são extraídas das próprias sentenças da Wikipedia e ranqueadas através de um processo semi-automatizado com base na frequência com que aparecem no conjunto de dados. Alguns exemplos de palavras-chave são apresentados na tabela 11.

Relação	Palavras-Chave
***CEO* (Diretor Executivo)**	CEO, chief, executive, officer
Founder (Fundador)	found, founder, founded, estabilish, form, foundation, open
Director (Diretor)	Director
Birth place (Local de nascimento)	Born, bear

Tabela 11 - exemplos de palavras-chave extraídas da Wikipedia

A figura 46 mostra dois exemplos de sentenças com suas respectivas árvores de dependência.

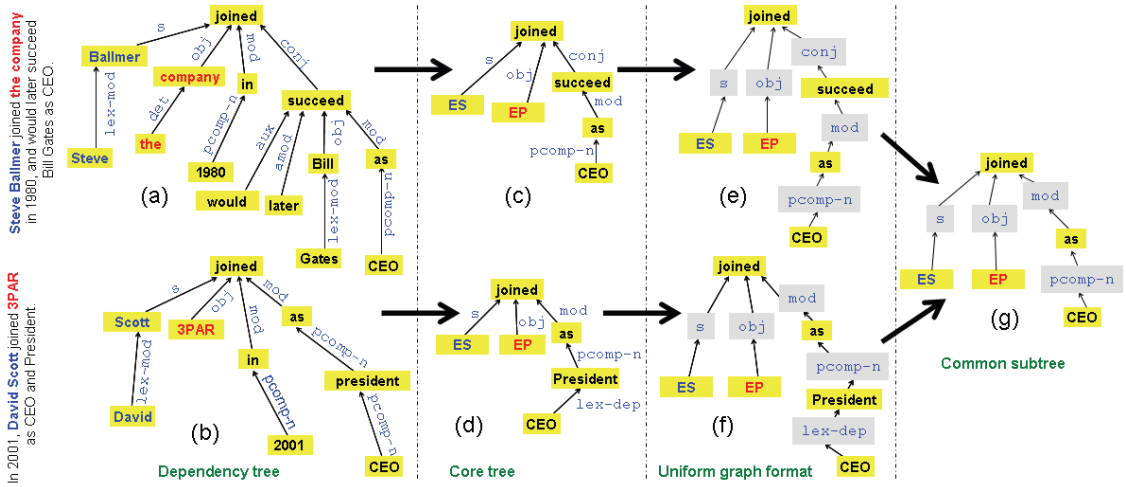

Figura 46 - geração das *core trees* e da subárvore comum para treinamento (fonte: [Nguyen, Matsuo e Ishizuka, 2007]).

A figura 46 ilustra o processo de geração de um exemplo positivo para a relação *CEO* (diretor executivo). O processo de extração de características tem início com a extração das árvores de dependência das sentenças (*(a)* e *(b)*). Após isso, são atribuídos rótulos as entidades principais e secundárias (“EP” e “ES” respectivamente). Após a rotulação das entidades, é adicionado o menor caminho entre a ES e a palavra-chave *CEO*, como é mostrado na figura 46 *(c)* e *(d)*. Em *(e)* e *(f)* ocorre uma conversão para um formato de grafo mais apropriado. O último passo é obter uma sub-árvore comum através da mineração das árvores obtidas no passo anterior, como é mostrado na figura 46 *(g)*. A subárvore comum é utilizada como exemplo positivo para treinar um classificador SVM binário para a relação *CEO*.

A avaliação realizada utilizou 5.595 artigos para treinamento e 45 artigos para teste. Foi treinado um classificador SVM binário para cada relação, obtendo um resultado geral de 29,07% de precisão, 53,86% de *recall* e 37.76% de *F1*.

Atualmente, o estado da arte na área de extração de relações é alcançado no trabalho realizado pelo centro de pesquisa Watson da IBM em [Wang et al., 2011]. Esse trabalho utiliza uma abordagem diferente que tem como objetivo reutilizar o conhecimento previamente extraído de relações já existentes para gerar características para novas relações.

O método apresentado em [Wang et al., 2011] utiliza a Wikipedia para extrair um conjunto de treinamento suficiente para caracterizar mais de 7000 relações exraídas diretamente das caixas de informação dos artigos. Além disso, são realizadas diversas consultas ao DBpedia para obtenção dos tipos das entidades, fornecidos pelo esquema de classificação YAGO [Suchanek, Kasneci e Weikum, 2007].

Após a montagem do conjunto de treinamento, são extraídas regras que caracterizam as relações, juntamente com os seus graus de popularidade. Uma regra extraída de uma relação entre duas entidades é formada por cinco componentes: tipo da entidade1, tipo da entidade2, substantivo, preposição e verbo. Um exemplo de regra para a relação *ActiveYearsEndDate* (relaciona uma pessoa com o ano da sua aposentadoria) é estruturada da seguinte forma:

person100007846|*year11520379*|-|*in*|*retire*

Nesse exemplo, o tipo da entidade1 é "*person100007846"*, o tipo da entidade2 é "*year11520379"*, como o substantivo é ausente, ele é representado por "-", a preposição e o verbo são as palavras-chave "*in"* e "*retire"*. As palavras-chave são extraídas do menor caminho entre as entidades no grafo de dependência linguística.

Várias regras podem ser extraídas para uma mesma relação, portanto, também é medido o grau de popularidade das regras baseado na frequência com que ocorrem no conjunto de dados.

Os principais problemas que ocorrem quando as relações são mineradas diretamente das caixas de informação são: a existência de subclasses de relações, como "*AcademyAward*" e "*Award*"; a possibilidade de sobreposição, como ocorre com as relações "*Composer*" e "*Artist*"; ou ainda podem existir relações equivalentes sobrepostas, como "*DateOfBirth*" e "*BirthDate*".

A solução utilizada para amenizar os problemas citados acima é a realização de uma análise de correlação entre as relações, resultando em um espaço de tópicos de relação, onde, cada tópico é uma distribuição multinomial sobre relações existentes. O agrupamento de relações similares é baseado, principalmente, na quantidade de regras compartilhadas e nos graus de popularidade das regras. Quanto mais regras compartilhadas, mais similares são as relações. Através da análise dos grupos, são modelados tópicos de relações. Quando uma nova relação é minerada da Wikipedia, ela é projetada em um dos tópicos criados.

Após o mapeamento das novas relações nos seus respectivos tópicos, as características do tópico são utilizadas para o projeto de um novo *kernel*, utilizado juntamente com a técnica SVM, onde a função para geração do espaço de características considera os seguintes fatores:

1. $K_{Argument}$: quantidade de tipos compartilhados entre as entidades;
2. K_{Path}: quantidade de substantivos, preposições e verbos compartilhados nos caminhos no grafo de dependência;
3. K_{Bow}: quantidade de substantivos, preposições e verbos compartilhados que não se encontram nos caminhos no grafo de dependência;
4. $K_{TF}(x,y)$: similaridade entre as relações x e y.

A função *kernel* é dada por: $\alpha_1 K_{Argument} + \alpha_2 K_{Path} + \alpha_3 K_{Bow} + \alpha_4 K_{TF}$. Onde o parâmentro α_i é ajustado para cada domíno.

O trabalho em [Wang et al., 2011] realiza experimentos em dois corpora distintos, o conjunto de documentos de notícias fornecido pela ACE (ACE 2004) e um corpus montado a partir da Wikipedia contendo 100 relações com 200 exemplos por relação. No corpus da ACE, a abordagem atinge um valor de *F1* de 73.24%. No corpus da Wikipedia, a abordagem atingiu um valor de 81.18% de *F1* em sua melhor configuração.

5.4. Supervisionamento Distante

Essa técnica é baseada na utilização de fontes de recursos externas e informações de *background* para adquirir ou aumentar o conhecimento a respeito das relações de interesse. Informações de *background* podem ser implícitas, como a correlação existente entre as relações [Wang et al., 2011], ou

explícitas, como a existência de menções da entidade objeto da relação no artigo do sujeito na Wikipedia [Chan e Roth, 2010].

As principais diferenças entre o trabalho que originou a abordagem de "supervisionamento distante", em [Mintz et al., 2009], e o presente trabalho concentram-se basicamente nos tipos de características utilizados. A abordagem proposta procura explorar características do menor caminho entre as entidades como característica sintática. Ao invés de registrar um caminho qualquer como característica, o presente trabalho captura alguns indícios do menor caminho entre as entidades que são os termos e as respectivas direções dos arcos na árvore de dependência. Além disso, o esquema de classificação de onde são extraídos os tipos das entidades como características semânticas é bem mais amplo, uma vez que utilizamos a ontologia do DBpedia como referência. Em [Mintz et al., 2009] são utilizados apenas 5 tipos, obtidos a partir da aplicação de um reconhecedor de entidades nomeadas (REN). A utilização de um REN no presente projeto não se fez necessária, uma vez que as entidades relacionadas são todas classificadas no DBpedia.

Outra diferença relevante está na forma como são montados os vetores de características. Em [Mintz et al., 2009], características de diferentes sentenças que descrevem uma mesma tupla <entidade1, relação, entidade2> são reunidas em um único vetor de características, partindo do princípio de que as características combinadas capturam de forma mais precisa a semântica da relação-alvo estabelecida entre a entidade1 e a entidade2. No presente trabalho, cada sentença, independente de descrever uma mesma tupla ou não, é utilizada como exemplo positivo para a relação-alvo.

Um dos trabalhos futuros levantados em [Mintz et al., 2009] é a aplicação de resolvedores de correferências. No presente trabalho foram aplicadas três estratégias específicas para textos da Wikipédia que apresentam 97.25% de precisão quando utilizadas em conjunto.

5.5. Frameworks

Com o objetivo de facilitar a experimentação de diversos algoritmos para extração de informação, o trabalho em [Iria, 2005] apresenta o *framework* T-Rex. O T-Rex foi projetado para permitir a implementação de qualquer sistema de EI.

A abordagem utilizada consiste em dividir o problema em vários sub-problemas independentes, permitindo com que vários algoritmos possam ser

trabalhados separadamente para melhorar o resultado final. A figura 47 mostra a arquitetura modular do T-Rex.

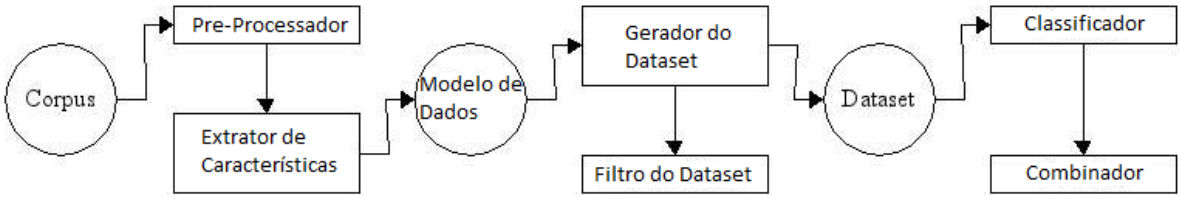

Figura 47 - *Framework* T-Rex (fonte: [Iria, 2005]).

O T-Rex é dividido em dois subsistemas, Pré-processamento e Classificação, tendo como fronteira o modelo de dados. O Pré-processamento possui um componente processador e um extrator de características. Esse subsistema pode ser visto como dependente de ferramentas de Processamento de Linguagem Natural (PLN) que analisam o corpus e geram uma representação para um modelo de dados.

O subsistema de classificação é composto do gerador do *dataset*, de um componente de filtragem (Filtro do *Dataset*), de um componente classificador e de um combinador. Esse subsistema pode ser visto como dependente de técnicas de aprendizado de máquina, onde um ou mais classificadores podem ser aplicados ao *dataset* gerado a partir do modelo de dados, tendo os resultados combinados para formar a resposta final.

Procurando aproveitar o conhecimento e as ferramentas adquiridos durante décadas de pesquisas na área de Extração de Informação, o framework FOX (Federated KnOwledge eXtraction) provê meios para combinar resultados de várias ferramentas PLN através de aprendizado de máquina, gerando serviços de anotação com alta precisão [Ngomo et al., 2011]. Com isso, FOX agrega resultados de várias ferramentas PLN, implementadas com diferentes abordagens, utilizando redes neurais *feed-forward*. Como saída, são gerados conjuntos de triplas RDF. FOX acomoda funcionalidades para reconhecimento de entidades nomeadas, extração de palavras-chave e extração de relações.

O *framework* FOX é baseado na idéia de modelos de assembléia, onde um classificador é construído combinando-se as predições obtidas de vários classificadores $C_1 \ldots C_n$ básicos. O diferencial desse *framework* para os modelos de assembléia usuais está na forma de combinação dos resultados, onde, ao invés de votação simples ou ponderada, os resultados são combinados através de uma rede neural. A figura 48 apresenta a topologia da rede neural utilizada em [Ngomo et al., 2011].

Cada classificador C_i gera um valor $v \in \{0,1\}$ para cada classe P_j. Esses valores são utilizados para ativar a camada de entrada da rede neural $\mathcal{I}_j^l$, onde a quantidade de nós equivale à quantidade de classificadores multiplicada pela quantidade de classes possíveis.

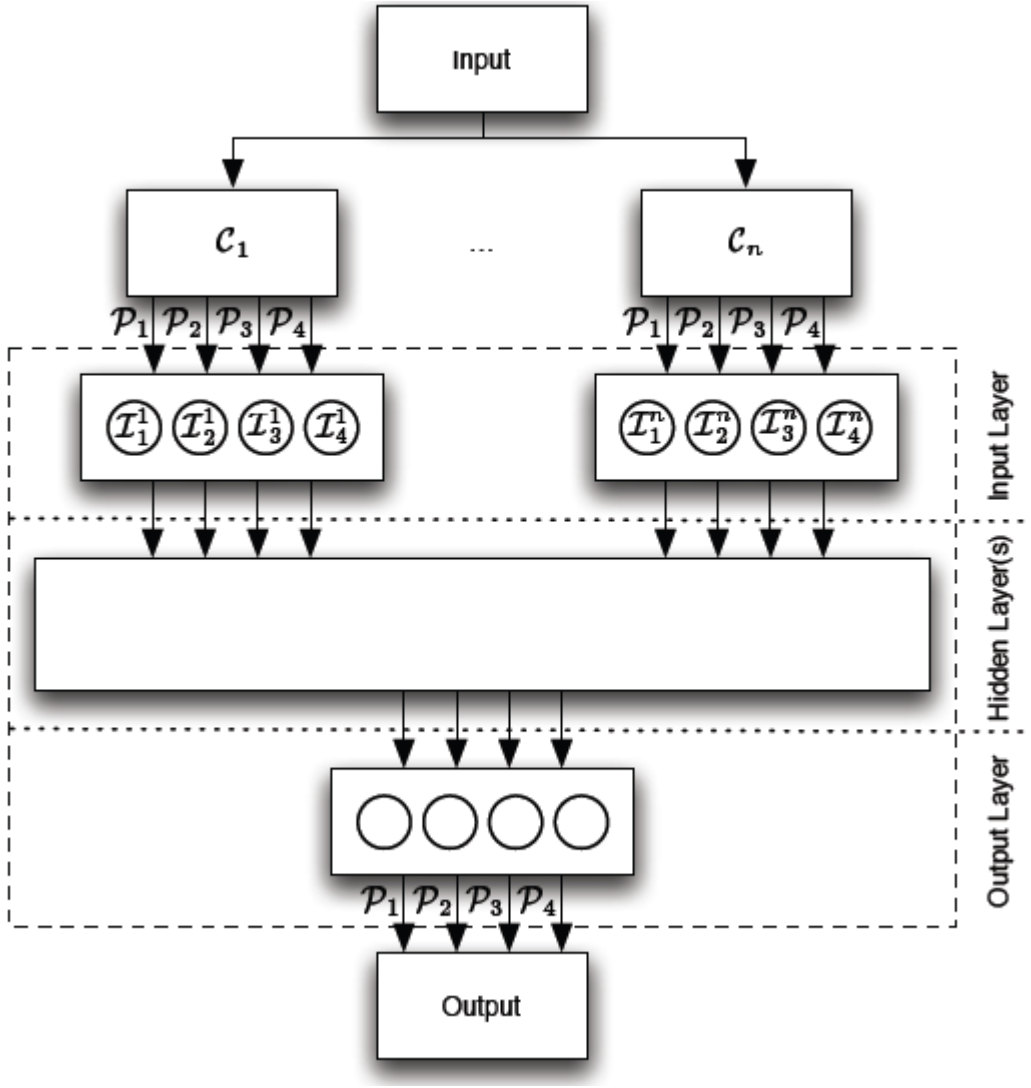

Figura 48 - Topologia da rede neural apresentada em [Ngomo et al., 2011] (fonte: [Ngomo et al., 2011]).

Para realizar o treinamento da rede, os classificadores previamente treinados são aplicados ao *dataset* gerando conjuntos de predições. Em seguida é aplicado o algoritmo de treinamento *backpropagation* com a função de transferência sigmóide.

A figura 49 apresenta a arquitetura do *framework* FOX.

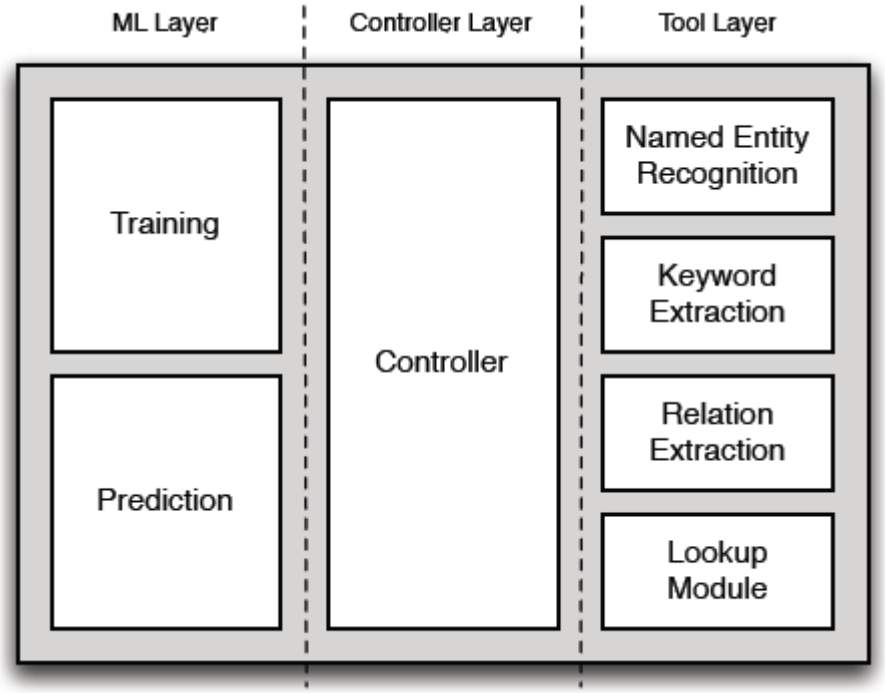

Figura 49 - Arquitetura do *framework* FOX (fonte: [Ngomo et al., 2011])

O *framework* apresentado na figura 49 é formado por três camadas. A camada de aprendizado disponibiliza interfaces para implementação de técnicas de modelos de assembléia. Essa camada se divide em dois módulos principais. O módulo de treinamento permite a carga do *dataset* de treinamento e configurações de ferramentas, possibilitando que o FOX aprenda a melhor combinação de ferramentas para o *dataset* carregado. O módulo de predição permite a aplicação do FOX a dados de entrada carregando os resultados de uma sessão de treinamento. O módulo controlador inicializa e coordena o acesso aos módulos dependentes de PLN, e coleta os resultados obtidos. A última camada armazena as ferramentas PLN incorporadas ao FOX.

Os resultados dos experimentos em reconhecimento de entidades nomeadas mostram que o FOX supera as ferramentas do estado da arte atual em mais de 15 % de *F1*, atingindo 90 % em um dos casos aplicados.

6
Conclusão

Este trabalho apresenta uma abordagem supervisionada a distância para Extração de Relações que combina duas das maiores fontes de conteúdo estruturado e não estruturado disponíveis na Web, o DBpedia e a Wikipedia.

6.1.
Contribuições

As contribuições deste trabalho são listadas a seguir:

- Uma abordagem supervisionada a distância utilizando o DBpedia e a Wikipedia para construção de extratores de relações para Inglês e Português;
- Este trabalho introduz um novo tipo de característica extraída a partir do menor caminho na árvore de dependências, onde, são considerados apenas os termos e as direções dos arcos ao invés do caminho completo.
- Um conjunto de 161.829 exemplos descrevendo mais de 90 relações do DBpedia para Inglês;
- Um conjunto de 149.579 exemplos descrevendo mais de 50 relações do DBpedia para Português;
- Um extrator capaz de reconhecer 90 relações do DBpedia em textos em Inglês avaliado com 86.09% de medida F1 através de validação cruzada e 81.08% de medida F1 em um conjunto de testes de 28.773 sentenças;
- Um extrator capaz de reconhecer 50 relações do DBpedia em textos em Português com 89.75% de medida F1 através de validação cruzada e 81.91% de medida F1 em um conjunto de testes de 18.333 sentenças;
- Um *framework* para extração de relações que abrange as principais etapas de um processo de ER supervisionado. O framework possibilitou uma implementação flexível e

independente de cada etapa do processo, agilizando a realização dos experimentos.

6.2. Trabalhos Futuros

Durante o desenvolvimento do projeto desta dissertação, identificamos os seguintes trabalhos futuros:

- Necessidade de uma comparação mais precisa com o trabalho em [Wang et al., 2011], que representa o estado da arte. Como o conjunto de exemplos minerados da Wikipedia em [Wang et al., 2011] se encontra indisponível, não foi possível realizar uma comparação precisa. Portanto, os experimentos e comparações realizados no capítulo 3, que trata da abordagem, ainda são preliminares.
- Avaliação dos detectores de relações gerados em, pelo menos, mais um domínio. Esse trabalho futuro vem da necessidade de avaliar a capacidade dos detectores de reconhecer relações, não só em textos da Wikipedia, como também em outros domínios. Futuramente, planejamos aplicar os detectores gerados no domínio de notícias utilizando um corpus específico para esse fim.
- Evolução da arquitetura do *framework*. Atualmente, a arquitetura do *framework* foi projetada para uma execução sequencial dos módulos, embora, em algumas situações, seja possível uma execução paralela. Podemos, por exemplo, executar paralelamente os extratores de características ou pré-processar lotes de artigos em paralelo. Além disso, assim como na ferramenta FOX, pretendemos estender o *framework* para acomodar outros elementos de extração de informação, como reconhecedores de entidades e extratores de palavras-chave, por exemplo.
- Exploração de outros tipos de características. Futuramente, pretendemos acrescentar outros tipos de características nos experimentos, como as partes do discurso dos termos, outros esquemas de classificação e palavras-chave.
- Utilização do analisador de dependências para Português estado-da-arte, apresentando em [Fernandes e Milidiú, 2012], para extrair

caracerísticas do menor caminho entre as entidades em sentenças em Português.

- Exploração da possibilidade de utilização de uma função *kernel* para cálculo de similaridade entre subestruturas presentes no menor caminho.
- Pesquisar sobre como podemos incluir uma classe para representar a não existência de uma relação.
- Disponibilização do corpus deste trabalho. Pretendemos disponibilizar os corpora para Inglês e Português, gerados através da abordagem proposta, para futuros experimentos na área de ER.

7 Referências

[Aasman, 2008] AASMAN, J. **Utilizing Federated Knowledge in Semantic Web Applications**. Proceedings of the 2008 IEEE International Conference on Semantic Computing, IEEE Computer Society, 2008, 486-487.

[Antoniou e Van Harmelen, 2008] ANTONIOU, G.; VAN HARMELEN, F. **A Semantic Web Primer**. Second Edition. The Mit Press, 2008.

[Berners-lee, 2007] BERNERS-LEE, T. **Linked Data**. W3C Design Issues, Jul 2007. Disponível em <http://www.w3.org/DesignIssues/LinkedData.html>, acessado em 10 Jan2012.

[Berners-lee, Hendler e Lassila, 2001] BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. **The Semantic Web**. Scientifc American, 2001.

[Bizer, 2011] BIZER, C. **DBpedia 3.7 released, including 15 localised editions.** Disponível em: http://blog.dbpedia.org/2011/09/11/dbpedia-37-released-including-15-localized-editions/, acessado em 13 Jan2012.

[Bizer, Heath e Berners-lee, 2009] BIZER, C.; HEATH, T.; BERNERS-LEE, T.. **Linked data - the story so far**. International Journal on Semantic Web and Information Systems (IJSWIS), 2009.

[Bizer et al., 2009] BIZER, C.; LEHMANN, J.; KOBILAROV, G.; AUER, S.; BECKER, C.; CYGANIAK, R. & HELLMANN, S. **DBpedia - A crystallization point for the Web of Data.** *Web Semant., Elsevier Science Publishers B. V.,* 2009*, 7*, 154-165.

[Bonfim, 2011] BOMFIM, M. **Um método e um ambiente para o desenvolvimento de aplicações na Web Semântica.** Dissertação de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, PUC-Rio, 2011.

[Bunescu e Mooney, 2005] BUNESCU, R. C.; MOONEY, R. J. **A Shortest Path Dependency Kernel for Relation Extraction**. Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP-05), 2005, 724-731.

[Chan e Roth, 2010] CHAN, Y. S.; ROTH, D. **Exploiting background knowledge for relation extraction**. Proceedings of the 23rd International Conference on Computational Linguistics, 2010, pages 152–160.

[Cristianini e Shawe-Taylor, 2000] CRISTIANINI, N.; SHAWE-TAYLOR, J. **An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, 1st ed**. Cambridge University Press, 2000.

[Culotta e Sorensen, 2004] CULOTTA, A.; SORENSEN. J. **Dependency tree kernels for relation extraction**. Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (ACL '04). Association for Computational Linguistics, Stroudsburg, PA, USA, 2004. Artigo 423.

[De Marnee e Manning, 2011] DE MARNEE, M. C.; MANNING, C. D. **Stanford typed dependencies manual.** 2011. Disponível em

<http://nlp.stanford.edu/software/dependencies_manual.pdf>, acessado em 13 Jan2012.

[Fernandes e Milidiú, 2012] FERNANDES, E. R.; MILIDIÚ, R. L. **Entropy-Guided Feature Generation for Structured Learning of Portuguese Dependency Parsing**. Proceedings of the 10th International Conference on Computational Processing of the Portuguese Language, PROPOR 2012, Coimbra, Portugal, April 17-20, 2012. Lecture Notes in Computer Science, v. 7243, p. 146-156, Helena Caseli, Aline Villavicencio, António Teixeira, Fernando Perdigão (eds.), Springer, 2012.

[Fernandes, Milidiú e Rentería, 2010] FERNANDES, E.; MILIDIÚ, R.; RENTERÍA, R. **RelHunter: a machine learning method for relation extraction from text**. Journal of the Brazilian Computer Society, 2010, 191-199.

[Fernandes, 2012] FERNANDES, W. **Quotation Extraction for Portuguese.** Dissertação de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, PUC-Rio, 2012.

[Fernandes, Dos Santos e Milidiú, 2012] FERNANDES, E, R.; DOS SANTOS, C, N.; MILIDIÚ, R, L. **Latent Structure Perceptron with Feature Induction for Unrestricted Coreference Resolution**. Proceedings of the Sixteenth Conference on Computational Natural Language Learning Shared Task (CoNLL 2012), Jeju, Korea, 2012.

[Gamma et al., 1995] GAMMA, R. et al. **Design Patterns: Elements of reusable object-oriented software**. Addison Wesley, 1995.

[Garcia e Pablo, 2011] GARCIA, M.; GAMALLO, P. **Dependency-Based Text Compression for Semantic Relation Extraction.** Proceedings of the RANLP 2011 Workshop on Information Extraction and Knowledge Acquisition, 2011.
[Giles, 2005] GILES, J. **Internet encyclopaedias go head to head** Nature, Nature Publishing Group*,* 2005*, 438*, 900-901.

[Hausenblas e Karnstedt, 2010] HAUSENBLAS, M.; KARNSTEDT, M.. **Understanding linked open data as a webscale database**. Advances in Databases, First International Conference on, 0:56–61, 2010.

[Iria, 2005] IRIA, J. **T-Rex: A Flexible Relation Extraction Framework.** Proceedings of the 8th Annual Colloquium for the UK Special Interest Group for Computational Linguistics (CLUK'05), 2005.

[Kambhatla, 2004] KAMBHATLA, N. **Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations**. Proceedings of the ACL 2004 on Interactive poster and demonstration sessions, Association for Computational Linguistics, 2004.

[Miller, 1998] MILLER, E. **An introduction to the resource description *framework*.** D-Lib Magazine, Maio 1998.

[Mintz et al., 2009] MINTZ, M. et al. **Distant supervision for relation extraction without labeled data**. Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics (ACL) and the 4th International Joint Conference on Natural Language Processing (IJCNLP), 2009, pages 1003-1011.

[Nakayama et al., 2008] NAKAYAMA, K. et al. **Wikipedia Link Structure and Text Mining for Semantic Relation Extraction.** Proceedings of the Workshop on Semantic Search (SemSearch 2008) at the 5th European Semantic Web Conference (ESWC 2008) , 2008, Tenerife, Spain, CEUR-WS.org, 2008, 334.

[Ngomo et al., 2011] NGOMO, N. A.; HEINO, N.; SPECK, R.; HILLNER, S. **Federated Knowledge extraction for Semantic Web Applications**. Disponível em http://aksw.org/Projects/FOX/files?get=fox_evaluation.pdf, acessado em 2 Fev2012.

[Nguyen, Matsuo e Ishizuka, 2007] NGUYEN, D. P. T.; MATSUO, Y.; ISHIZUKA, M. **Relation extraction from wikipedia using subtree mining.** Proceedings of the 22nd national conference on Artificial intelligence - Volume 2, AAAI Press, 2007, 1414-1420.

[Oren et al., 2010] OREN, E.; DELBRU, R.; GERKE, S.; HALLER, A.; DECKER, S.; **ActiveRDF: object-oriented semantic web programming**. Proceedings of the 16th Int'l Conference on World Wide Web (WWW 07), 2007, pp. 817–824, Banff, Alberta, Canada. ACM.

[Santos, Mamede e Baptista, 2010] SANTOS, D.; MAMEDE, N.; BAPTISTA, J. **Extraction of Family Relations between Entities.** INForum, 2010.

[Studer, Benjamins e Fensel, 1998] STUDER ,R.; BENJAMINS, R.; FENSEL, D. **Knowledge engineering: Principles and methods Data & Knowledge Engineering**. 25(1-2):161-198, Março 1998.

[Suchanek, Kasneci e Weikum, 2007] SUCHANEK, M., F.; KASNECI, G.; WEIKUM, G. **YAGO: A large ontology from Wikipedia and WordNet.** Web Semantics: Science, Services and Agents on the World Wide Web, 2007, 6(3):203-217.

[Wang et al., 2011] WANG, C.; FAN, J.; KALYANPUR, A.; Gondek. D. **Relation extraction with relation topics**. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '11). Association for Computational Linguistics, Stroudsburg, PA, USA, 2011, 1426-1436.

[Yan et al., 2009] YAN, Y.; OKAZAKI, N.; MATSUO, Y.; YANG, Z.; ISHIZUKA, M. **Unsupervised relation extraction by mining Wikipedia texts using information from the web**. Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2, Association for Computational Linguistics, 2009, 1021-1029.

[Zachte, 2012] ZACHTE, E. **Wikipedia Statistics.** Disponível em: http://stats.wikimedia.org/EN/Sitemap.htm, acessado em 12 jan2012

[Zhou e Zhu, 2011] ZHOU, G.; ZHU, Q. **Kernel-based semantic relation detection and classification via enriched parse tree structure**. Journal of Computer Science and Technology, 2011, 26:45.56.

Apêndice A – Tabela de Exemplos Positivos por Relação para Inglês

Relação	Quantidade de Exemplos
http://dbpedia.org/ontology/riverMouth	4980
http://dbpedia.org/ontology/award	4837
http://dbpedia.org/ontology/musicalBand	4360
http://dbpedia.org/ontology/college	4128
http://dbpedia.org/ontology/birthPlace	3140
http://dbpedia.org/ontology/country	3090
http://dbpedia.org/ontology/isPartOf	2894
http://dbpedia.org/ontology/writer	2469
http://dbpedia.org/ontology/region	2083
http://dbpedia.org/ontology/nearestCity	2025
http://dbpedia.org/ontology/producer	2013
http://dbpedia.org/ontology/family	1966
http://dbpedia.org/ontology/recordLabel	1955
http://dbpedia.org/ontology/distributor	1945
http://dbpedia.org/ontology/district	1935
http://dbpedia.org/ontology/knownFor	1934
http://dbpedia.org/ontology/spouse	1888
http://dbpedia.org/ontology/composer	1864
http://dbpedia.org/ontology/league	1853
http://dbpedia.org/ontology/architect	1831
http://dbpedia.org/ontology/owningCompany	1756
http://dbpedia.org/ontology/album	1752
http://dbpedia.org/ontology/relative	1751
http://dbpedia.org/ontology/associatedMusicalArtist	1707
http://dbpedia.org/ontology/parent	1690
http://dbpedia.org/ontology/location	1687
http://dbpedia.org/ontology/series	1682
http://dbpedia.org/ontology/draftTeam	1658
http://dbpedia.org/ontology/language	1596
http://dbpedia.org/ontology/developer	1548
http://dbpedia.org/ontology/headquarter	1536
http://dbpedia.org/ontology/isPartOfMilitaryConflict	1532
http://dbpedia.org/ontology/genre	1524
http://dbpedia.org/ontology/battle	1498
http://dbpedia.org/ontology/builder	1458
http://dbpedia.org/ontology/manufacturer	1413

http://dbpedia.org/ontology/crosses	1384
http://dbpedia.org/ontology/relatedMeanOfTransportation	1367
http://dbpedia.org/ontology/keyPerson	1358
http://dbpedia.org/ontology/largestCity	1357
http://dbpedia.org/ontology/militaryUnit	1276
http://dbpedia.org/ontology/leader	1274
http://dbpedia.org/ontology/ground	1255
http://dbpedia.org/ontology/federalState	1245
http://dbpedia.org/ontology/operator	1190
http://dbpedia.org/ontology/publisher	1183
http://dbpedia.org/ontology/parentCompany	1169
http://dbpedia.org/ontology/assembly	1160
http://dbpedia.org/ontology/archipelago	1146
http://dbpedia.org/ontology/citizenship	1142
http://dbpedia.org/ontology/child	1086
http://dbpedia.org/ontology/employer	1071
http://dbpedia.org/ontology/formerBandMember	1070
http://dbpedia.org/ontology/trainer	1053
http://dbpedia.org/ontology/guest	1051
http://dbpedia.org/ontology/channel	1042
http://dbpedia.org/ontology/spokenIn	1041
http://dbpedia.org/ontology/narrator	1038
http://dbpedia.org/ontology/capital	1036
http://dbpedia.org/ontology/city	1030
http://dbpedia.org/ontology/network	1001
http://dbpedia.org/ontology/director	957
http://dbpedia.org/ontology/sport	929
http://dbpedia.org/ontology/homeport	889
http://dbpedia.org/ontology/musicBy	886
http://dbpedia.org/ontology/primeMinister	836
http://dbpedia.org/ontology/movement	822
http://dbpedia.org/ontology/tenant	805
http://dbpedia.org/ontology/mission	763
http://dbpedia.org/ontology/affiliation	755
http://dbpedia.org/ontology/starring	743
http://dbpedia.org/ontology/architecturalStyle	663
http://dbpedia.org/ontology/almaMater	660
http://dbpedia.org/ontology/cinematography	649
http://dbpedia.org/ontology/instrument	583
http://dbpedia.org/ontology/predecessor	578
http://dbpedia.org/ontology/format	561
http://dbpedia.org/ontology/subsequentWork	537
http://dbpedia.org/ontology/openingTheme	489
http://dbpedia.org/ontology/secondLeader	484
http://dbpedia.org/ontology/division	467
http://dbpedia.org/ontology/designer	445

http://dbpedia.org/ontology/headquarters	433
http://dbpedia.org/ontology/campus	378
http://dbpedia.org/ontology/team	378
http://dbpedia.org/ontology/manager	317
http://dbpedia.org/ontology/lastAppearance	291
http://dbpedia.org/ontology/managerClub	286
http://dbpedia.org/ontology/order	262
http://dbpedia.org/ontology/phylum	253
http://dbpedia.org/ontology/religion	159
http://dbpedia.org/ontology/discoverer	90
Total	125351

Apêndice B – Tabela de Exemplos Positivos por Relação para Português

Relação	Quantidade de Exemplos
http://dbpedia.org/ontology/birthPlace	20639
http://dbpedia.org/ontology/album	15238
http://dbpedia.org/ontology/region	14954
http://dbpedia.org/ontology/producer	9180
http://dbpedia.org/ontology/district	7551
http://dbpedia.org/ontology/associatedMusicalArtist	7205
http://dbpedia.org/ontology/musicalBand	5912
http://dbpedia.org/ontology/director	4705
http://dbpedia.org/ontology/city	3990
http://dbpedia.org/ontology/discoverer	3963
http://dbpedia.org/ontology/family	3516
http://dbpedia.org/ontology/capital	3484
http://dbpedia.org/ontology/writer	3012
http://dbpedia.org/ontology/largestCity	3008
http://dbpedia.org/ontology/recordLabel	2645
http://dbpedia.org/ontology/channel	2626
http://dbpedia.org/ontology/knownFor	2156
http://dbpedia.org/ontology/riverMouth	2100
http://dbpedia.org/ontology/parent	2076
http://dbpedia.org/ontology/child	1953
http://dbpedia.org/ontology/battle	1744
http://dbpedia.org/ontology/team	1743
http://dbpedia.org/ontology/tenant	1588
http://dbpedia.org/ontology/publisher	1508
http://dbpedia.org/ontology/spouse	1361
http://dbpedia.org/ontology/relatedMeanOfTransportation	1145
http://dbpedia.org/ontology/genre	1131
http://dbpedia.org/ontology/sport	1096
http://dbpedia.org/ontology/predecessor	1045
http://dbpedia.org/ontology/mission	1032
http://dbpedia.org/ontology/spokenIn	985
http://dbpedia.org/ontology/award	942
http://dbpedia.org/ontology/series	916
http://dbpedia.org/ontology/headquarter	869
http://dbpedia.org/ontology/league	863
http://dbpedia.org/ontology/ground	848

http://dbpedia.org/ontology/format	768
http://dbpedia.org/ontology/language	689
http://dbpedia.org/ontology/composer	652
http://dbpedia.org/ontology/country	650
http://dbpedia.org/ontology/distributor	614
http://dbpedia.org/ontology/starring	569
http://dbpedia.org/ontology/developer	519
http://dbpedia.org/ontology/instrument	510
http://dbpedia.org/ontology/doctoralAdvisor	477
http://dbpedia.org/ontology/openingTheme	451
http://dbpedia.org/ontology/subsequentWork	441
http://dbpedia.org/ontology/location	428
http://dbpedia.org/ontology/guest	406
http://dbpedia.org/ontology/relative	362
http://dbpedia.org/ontology/owningCompany	320
http://dbpedia.org/ontology/keyPerson	282
http://dbpedia.org/ontology/almaMater	267
http://dbpedia.org/ontology/federalState	248
http://dbpedia.org/ontology/leader	207
Total	147589