



Gustavo Lopes Mourad

**Uma Estratégia para o Enriquecimento de Bases de Dados
Locais Utilizando-se Recursos da Deep Web**

Dissertação de Mestrado

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

Orientador: Prof. Karin Breitman

Rio de Janeiro
Setembro de 2010

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador.

Gustavo Lopes Mourad

Graduou-se em Engenharia de Computação pela Pontifícia Universidade Católica do Rio de Janeiro. Desenvolveu como projeto de mestrado um framework para enriquecimento de bases de dados com recursos da Deep Web.

Mourad, Gustavo L.

Uma Estratégia para o Enriquecimento de Bases de Dados Locais Utilizando-se Recursos da Deep Web / Gustavo Lopes Mourad; orientador: Karin Breitman. – Rio de Janeiro : PUC-Rio, Departamento de Informática, 2010

v., 67 f: il. ; 29,7 cm

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

Inclui referências bibliográficas.

1. Informática – Dissertação. 2. Detecção de duplicatas. 3. Casamento de entidades . 4. Integração de Dados 5. Deep Web. I. Breitman, Karin. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título

Dedico este trabalho a meus pais e avós pelo apoio aos estudos em todas as fases da minha vida, minha esposa Rachel e minha irmã.

Agradecimentos

A minha orientadora, Dr. Karin Breitman, por ter me aceito e acreditado no meu trabalho. Sua competência, dedicação e paciência foram fundamentais para a realização desta dissertação.

Aos professores do Departamento de Informática da Puc-Rio pelas aulas, conselhos e questões estimulantes.

A todos os colegas de curso e trabalho que sempre me apoiaram com coleguismo, respeito e amizade.

Por fim, agradeço à CAPES e Puc-Rio pelo apoio financeiro e ao Departamento de Informática da Puc-Rio pela excelente formação.

Resumo

Mourad, Gustavo L.; Breitman, K. **Uma Estratégia para o Enriquecimento de Bases de Dados Locais Utilizando-se Recursos da Deep Web**. Rio de Janeiro, 2010. 67p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

À medida em que aplicações web que combinam dados de diferentes fontes ganham importância, soluções para a detecção online de dados duplicados tornam-se centrais. A maioria das técnicas existentes são baseadas em algoritmos de aprendizado de máquina, que dependem do uso de bases de treino criadas manualmente. Estas soluções não são adequadas no caso de da Deep Web onde, de modo geral, existe pouca informação acerca do tamanho das fontes de dados, da volatilidade dos mesmos e do fato de que a obtenção de um conjunto de dados relevante para o treinamento é uma tarefa difícil. Nesta dissertação propomos uma estratégia para extração (scrapping), detecção de duplicatas e incorporação de dados resultantes de consultas realizadas em bancos de dados na Deep Web. Nossa abordagem não requer o uso de conjuntos de testes previamente definidos, mas utiliza uma combinação de um classificador baseado no Vector Space Model, com funções de cálculo de similaridade para prover uma solução viável. Para ilustrar nossa proposta, nós apresentamos um estudo de caso onde o framework é instanciado para uma aplicação do domínio dos vinhos.

Palavras-chave

Detecção de duplicatas, casamento de entidades, Integração de dados, Deep Web

Abstract

Mourad, Gustavo L.; Bretiman, K.. **A Strategy for the Enhancement of Local Databases Using Deep Web Resources**. Rio de Janeiro, 2010. 67p. MSc Thesis - Department of Computer Science, Pontifícia Universidade Católica do Rio de Janeiro.

As Web applications that obtain data from different sources (Mashups) grow in importance, timely solutions to the duplicate detection problem become central. Most existing techniques, however, are based on machine learning algorithms, that heavily rely on the use of relevant, manually labeled, training datasets. Such solutions are not adequate when talking about data sources on the Deep Web, as there is often little information regarding the size, volatility and hardly any access to relevant samples to be used for training. In this thesis we propose a strategy to aid in the extraction (scrapping), duplicate detection and integration of data that resulted from querying Deep Web resources. Our approach does not require the use of pre-defined training sets, but rather uses a combination of a Vector Space Model classifier with similarity functions, in order to provide a viable solution. To illustrate our approach, we present a case study where the proposed framework was instantiated for an application in the wine industry domain.

Keywords

Duplicate Detection, Entity Matching, Data Integration, Deep Web

Sumário

1	Introdução	11
1.1	Objetivo	12
1.2	Contribuições	13
1.3	Resumo:	14
2	Conceitos Básicos	15
2.1	Recuperação de Dados da Web	15
2.2	Construção de rastreadores (<i>crawlers</i>)	17
2.3	Casamento de Esquemas - Schema Matching	19
2.4	Casamento de entidades – Entity Matching	20
2.5	Funções de Similaridade entre Cadeias de Caracteres	23
2.5.1	<i>Funções baseadas em caracteres</i>	24
2.5.1.1	<i>Distância de Levenshtein</i>	24
2.5.1.2	<i>Soundex (Busca Fonética)</i>	24
2.5.1.3	<i>Jaro-Winkler</i>	25
2.5.2	<i>Funções baseadas em tokens</i>	25
2.5.2.1	<i>Cosine Similarity</i>	26
2.5.2.2	<i>Dice Coefficient</i>	27
2.5.2.3	<i>Block Distance (Taxicab Geometry)</i>	27
2.5.2.4	<i>Jaccard</i>	27
2.5.3	<i>Funções híbridas</i>	28
2.5.3.1	<i>Monge-Elkan</i>	28
2.6	Classificação	28
2.6.1.1	<i>Técnicas de Classificação</i>	29
2.7	Resumo:	31
3	Estratégia para o enriquecimento de informações	33
3.1	Busca de informações na Deep Web	34
3.2	Incorporação de novas informações a partir da identificação de duplicatas	35
3.3	Algoritmo	36
3.3.1	Comentário sobre o escopo do trabalho	37
3.4	Resumo:	37
4	Framework Proposto	38
4.1	Arquitetura Proposta	38
4.2	O Processo de Utilização do Framework Proposto	42
4.3	Resumo:	45
5	Winetag.com.br: um estudo de caso	46
5.1	Metodologia de trabalho	47
5.1.1	Identificação das fontes de dados	47
5.1.2	Casamento de Esquemas - Schema Matching	47
5.1.3	Descoberta de padrões em URL's e montagem de consultas	48
5.1.4	Criação de parsers das fontes de dados	49
5.1.5	Escolha de funções de cálculo similaridade	51

5.1.6	Classificador	51
5.1.7	Fusão de dados	52
5.2	Resultados	54
5.3	Resumo:	58
6	Trabalhos relacionados	59
6.1	Rastreador (crawler)s de Deep Web	59
6.2	Casamento de Instâncias - Entity Matching	59
6.3	Comparação de resultados	60
7	Conclusão	62
7.1	Trabalhos Futuros	63
8	Bibliografia	64

Lista de Figuras

Figura 1 - Vista conceitual da Deep Web [7]	11
Figura 2 - Classificação de Extratores Web [21]	16
Figura 3 - Rastreador (crawler) tradicional [22]	17
Figura 4 - Rastreador (crawler) de Deep Web [22]	18
Figura 5 - Visão geral do algoritmo de similaridade de MARLIN [3]	22
Figura 6 - Representação gráfica de 2 documentos no VSM	26
Figura 7 - Exemplo de KNN	29
Figura 8 - Exemplo de árvore de decisão	30
Figura 9 - Exemplo de estrutura básica de nós de uma rede neural	31
Figura 10 - Visão geral do processo de enriquecimento de informações proposto: (a) Busca, (b) Incorporação de dados	33
Figura 11 - Algoritmo para a busca de informações na Deep Web	34
Figura 12 - Algoritmo para o casamento de entidades	36
Figura 13 - Diagrama de Classes	39
Figura 14 - Diagrama de Componentes do Framework	42
Figura 15 - Passos 1, 2 e 3 do processo (estratégia de busca)	43
Figura 16 - Passos 4 e 5 do processo (incorporação)	44
Figura 17 - Exemplo de fonte de dados	49
Figura 18 - Exemplo elementos identificados pelo Developer Tools	50
Figura 19 - Elementos destacados no código-fonte	50
Figura 20 - Diagrama de classes para instância de vinhos	53
Figura 21 - Winetag.com.br: mashup com dados da Deep Web	54

True genius resides in the capacity for evaluation of uncertain, hazardous, and conflicting information.

Winston Churchill

1 Introdução

Estudos recentes demonstram que uma fração significativa dos recursos disponíveis através da Web não podem ser alcançados através de links [2]. Páginas geradas dinamicamente são exemplos típicos destes recursos. Este fato impossibilita que mecanismos de busca tradicionais como Google, Yahoo e Bing possam indexar os conteúdos destas páginas e, conseqüentemente, apresentá-las entre seus resultados. Esses recursos ganharam o nome de *Hidden Web* ou *Deep Web*. Estima-se que menos de 40% das páginas Web são acessíveis através de mecanismos de busca [7]. Estima-se, ainda, que estes dados apresentam grande qualidade, são bem estruturados e crescem em uma proporção bem maior que a Web.

Os sites de e-commerce são os maiores produtores de conteúdo da *Hidden Web*. Segundo He, 94% destes sites que possuem conteúdo dinâmico, e fazem acesso a um banco de dados com apenas três links de navegação [7]. As páginas geradas dinamicamente por este tipo de sites são normalmente obtidas como respostas consultas realizadas através de formulários de pesquisa. Podemos, desta forma, encarar a *Deep Web* como o maior banco de dados (*Very Large Data Base - VLDB*) existente.

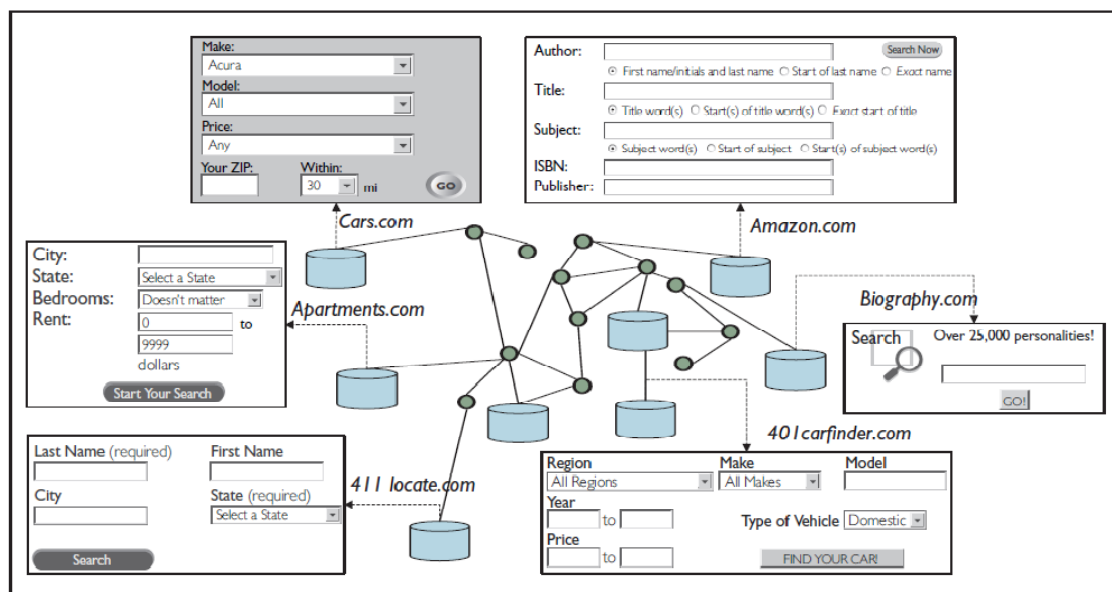


Figura 1 - Vista conceitual da Deep Web [7]

1.1 Objetivo

O objetivo desta dissertação é explorar a possibilidade de se utilizar dados da Deep Web para enriquecer bases de e-commerce existentes. Nossa intenção é propor um framework capaz de (1) identificar fontes de dados relevantes na Deep Web, (2) construir consultas automáticas as estas fontes de dados, (3) filtrar os resultados obtidos, (4) identificar a correspondência entre os resultados e as instâncias da base original (computar similaridade) e, finalmente, (5) enriquecer a base original com novas informações.

Aplicações geradas a partir do framework proposto podem servir a muitos propósitos, e.g., construção de novos sites, construção de mecanismos para a consulta e comparação de atributos, e aplicativos de *mashup* de dados. O foco deste trabalho, porém, é sua utilização como um mecanismo para enriquecer informações incompletas ou faltantes.

São dois desafios da deste trabalho. Em primeiro lugar temos que construir mecanismos de rastreamento (*crawler*) capazes de buscar informações na Deep Web. Rastreadores do tipo tradicional foram concebidos para funcionamento na Web de Superfície, e são programados para a coleta de informações contidas em paginas Web estáticas, que podem ser atingidas através da navegação por links. No caso da Deep Web, é precisos que estes rastreadores sejam capazes de submeter consultas através da interface Web de bancos de dados relacionais. A maior parte da literatura da área [10] trata de casos onde se deseja obter a totalidade dos dados, através do menor numero de consultas possível (*recall*). Desta forma, a maior parte dos artigos trata de questões como o limite do numero de respostas imposto por determinados sites, a identificação de atributos relevantes de forma a reduzir o numero de consultas necessárias, sobreposição de respostas, entre outras [40, 41, 16, 17].

Note que, como desejamos encontrar potencial informação adicional sobre registros da nossa base de dados, não estamos interessados em obter um grande número de respostas, mas sim respostas mais precisas. A situação ideal é encontrar uma combinação de atributos (não nulos) do registro de nossa base que retorne o mesmo objeto na base que está sendo pesquisada.

O segundo desafio, fundamental a qualquer sistema capaz de integrar dados de diversas fontes, é reconhecer se uma determinada entidade presente em uma fonte corresponde a uma entidade no mundo real, ou seja, encontrar duplicatas.

Este é um problema tem atraído a atenção de pesquisadores de diferentes áreas, incluindo Bancos de Dados, Data Mining, Inteligência Artificial e Processamento de Linguagens Naturais. Na literatura é apresentado sob várias denominações, entre outras: “Record Linkage”, “Merge/Purge”, “Reference Conciliation”, “Entity Resolution” ou “Duplicate Detection” [3, 5, 8, 9, 11, 15, 27, 28].

Muitas soluções já foram propostas para tratar este problema [8] A maioria, no entanto, funcionam muito bem para dados estruturados e em casos onde é possível aplicar técnicas de aprendizado de máquina a uma base de dados conhecida. Nosso foco, no entanto, é utilizar recursos encontrados na *Deep Web*. Neste caso, soluções codificadas e/ou treinamento *offline* não são adequados, primeiro porque o conjunto de dados a ser analisado não está disponível e é difícil de obter. Segundo, e mais importante, mesmo que haja um conjunto representativo de dados etiquetado para treinamento, as regras aprendidas podem não adequadas a todo o conjunto de dados ainda desconhecido.

Neste trabalho propomos uma estratégia que não requer uma base para treinamento. Como nossa proposta utiliza uma base de dados, a qual deseja-se enriquecer de dados, utilizamos funções de cálculo de similaridade em conjunto com um classificador para responder de forma online se duas entidades representam a mesma.

1.2 Contribuições

A contribuição deste trabalho é apresentar uma solução para o enriquecimento de bases de dados através de consultas em fontes de dados na *Deep Web*. Esta solução pode ser decomposta em duas contribuições:

1. Uma estratégia para a busca (rastreamento) de informações na *Deep Web* orientado a duplicatas, cujo objetivo é obter resultados precisos de consultas construídas a partir de um conjunto conhecido de objetos e seus atributos.

2. Uma estratégia para a detecção de duplicatas em resultados de consultas realizadas sobre fontes de dados da Deep Web, independente de bases de treinamento especificadas a priori. Apresentamos uma solução baseada em classificadores e funções de cálculo de similaridade.

Uma contribuição adicional deste trabalho é fornecer uma descrição detalhada da instanciação do framework proposto para o domínio dos vinhos, que podem ajudar na processo de instanciação deste framework em contextos análogos no futuro.

Esta dissertação está estruturada como se segue: no próximo capítulo, apresentamos os conceitos básicos que sustentam a solução apresentada neste trabalho; no capítulo 3, é demonstrada a estratégia utilizada neste trabalho; o framework proposto na dissertação é apresentado no capítulo 4; o capítulo 5 apresenta a implementação de uma instância do framework para o domínio dos vinhos; no capítulo 6 temos a apresentação de trabalho relacionados; os resultados desde projeto encontram-se no capítulo 7 juntamente com a conclusão do trabalho e, finalmente, no capítulo 8 temos a bibliografia estudada.

1.3 Resumo:

Neste capítulo apresentamos a motivação básica, e principais contribuições deste trabalho. Nosso objetivo é explorar a possibilidade de utilização da Deep Web em um processo que batizamos de enriquecimento de dados. Este processo consiste na obtenção de mais informações, e.g., texto e imagens, para complementar a informação de uma base de dados existente. Este processo pode ser decomposto em duas partes: busca e incorporação. No passo de busca é necessário construir rastreadores capazes de buscar informações contidas na Deep Web. A peculiaridade do nosso caso é a necessidade de se encontrar na bases de dados da Deep Web os objetos mais próximos daqueles pertencentes a base de dados original (precisão). A segunda parte do processo é identificação de duplicatas, i.e., identificar se o(s) objeto(s) recuperado(s) e aquele da base original são, de fato, o mesmo objeto real. No caso da Deep Web, o grande desafio é a impraticabilidade de se utilizar conjunto de dados para o treinamento, comuns a maior parte das estratégias de aprendizado de máquina utilizadas na identificação de duplicatas.

2 Conceitos Básicos

Neste capítulo apresentamos um breve sumário da pesquisa realizada nas áreas de recuperação de dados na web, construção de rastreadores e casamento de entidades, fundamentais para a estratégia aqui proposta.

Ainda apresentamos funções de cálculo de similaridade entre cadeias de caracteres e técnicas de classificação.

2.1 Recuperação de Dados da Web

O primeiro passo para a construção de uma aplicação que obtém dados da web é a escolha do meio para extração das informações. Existem várias formas de recuperação de dados presentes da web. Entre as diversas técnicas existentes discutimos as mais convencionais a seguir:

1. Protocolos Web: SOAP e REST. São protocolos neutros para comunicação de serviços remotos. Utilizam o paradigma da arquitetura orientada a serviços. São independentes de plataforma e baseiam-se na troca de mensagens. São exemplos Google Contacts e Yahoo (YQL, Geo, Search).
2. Protocolos RSS e ATOM. São protocolos de uma família de formatos baseados em XML. Sites que desejam disponibilizar conteúdos podem gerar arquivos deste tipo para leitores específicos do formato. É possível acompanhar inclusões e alterações de conteúdos em sites.
3. Protocolo RDF e similares. Através de marcações em códigos HTML é possível extrair informações com semântica relacionada a um arquivo RDF. Exemplos de sites que coletam dados deste tipo para *mashups* são o Google e o Yahoo. O Google mostra em seus resultados “*Snippets*”¹ se encontrar algumas marcações no código. Já o Yahoo (Search Monkey) tenta indexar páginas de comércio eletrônico com anotações do vocabulário “Good Relations”² para prover funcionalidades nos resultados de buscas.

1 <http://developer.yahoo.com/searchmonkey/>

2 <http://www.heppnetz.de/projects/goodrelations/>

4. Screen Scraping. É considerada a última alternativa a ser escolhida, já que a técnica baseia-se no parsing e análise de código HTML. Não há um contrato entre as aplicações envolvidas e por isso é de difícil manutenção e reuso.

De um modo geral, aplicações que utilizam esses dados são classificadas como *Mashups*:, uma aplicação web que combina conteúdo de uma ou mais fontes para criar serviços novos e inovadores [44].

Essas aplicações tem em comum a arquitetura orientada a serviços: baseada em baixo acoplamento e orientada a meta-dados. Por isso, esse tipo de aplicação deve ser desenvolvidos de forma iterativa, levando em consideração as mudanças de contratos entre sistemas. Dada a difícil manutenção dos contratos é comum o uso de ferramentas de apoio.

Para exploração do tema vale aprofundar-se nas ferramentas pesquisadas por Laender [21]. Em sua pesquisa, foi criada uma taxonomia para classificação das ferramentas quanto ao grau de flexibilidade e grau de automação como pode ser visto na figura abaixo.

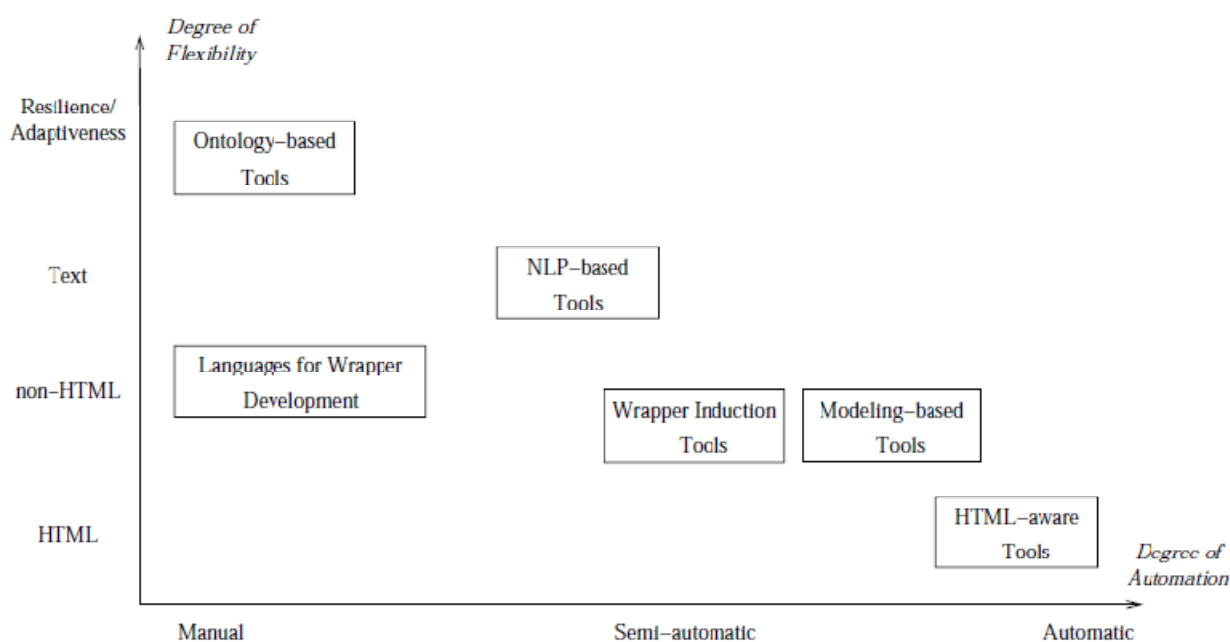


Figura 2 - Classificação de Extratores Web [21]

2.2 Construção de rastreadores (*crawlers*)

O desenvolvimento de rastreador (*crawler*)s específicos para a a exploração da Deep Web foi abordado por Peisu [22]. O objetivo do autor foi criar um rastreador genérico que pudesse navegar nos links gerados a partir de formulários de busca de sites. Peisu acredita que o que diferencia um rastreador de uma ferramenta de busca comum é a capacidade de identificação de formulários nas páginas, análise, e geração de URLs a partir de combinações possíveis entre os campos.

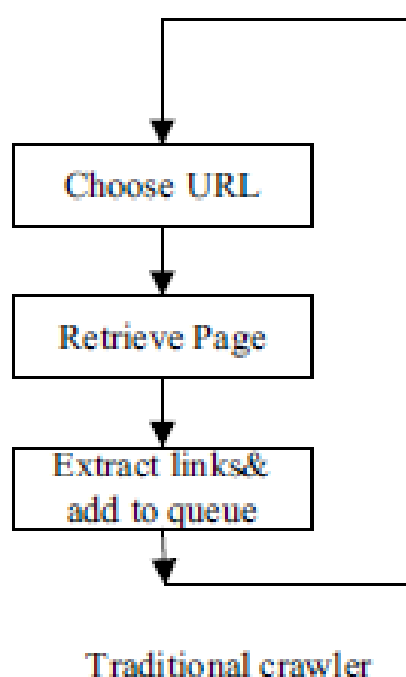


Figura 3 - Rastreador (crawler) tradicional [22]

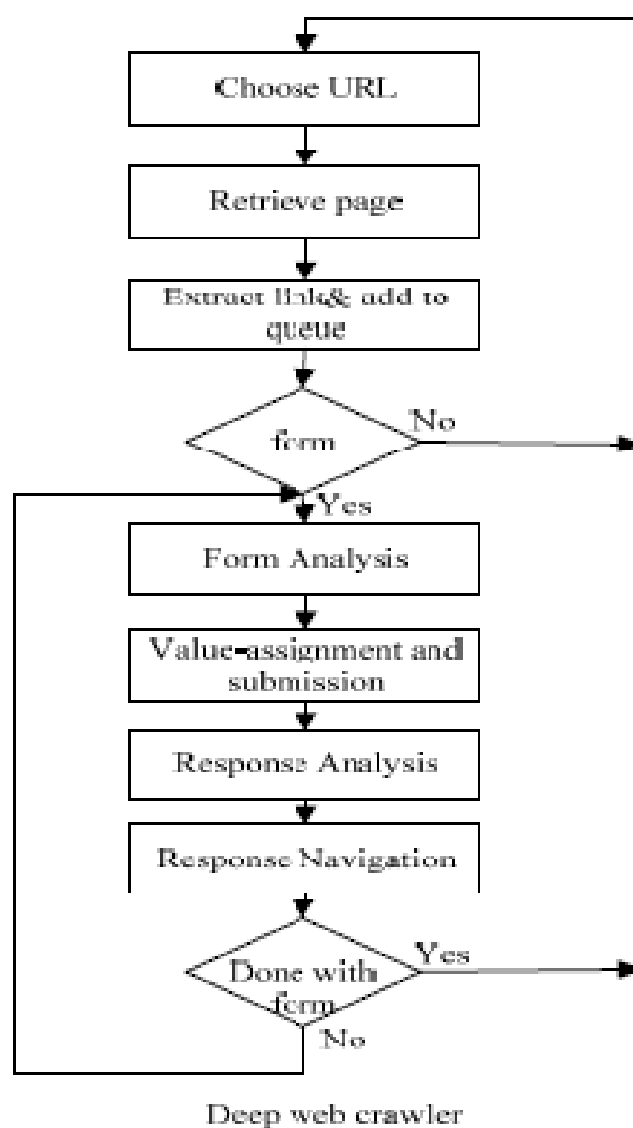


Figura 4 - Rastreador (crawler) de Deep Web [22]

Rastreadores de Deep Web tem a sua eficiência intimamente ligada a seleção das entradas dadas aos formulários de pesquisa. O rastreador proposto por Peisu, que visa retornar todos os objetos contidos na base visitada, explora a combinação de entradas do formulário de busca para gerar consultas. Se por um lado consegue capturar muitos dados, por outro é ineficiente pois realiza diversas buscas sem sentido. Como o rastreador não conhece a semântica entre os itens dos formulários, acaba fazendo consultas que não retornarão resultados, uma vez que o banco não apresentará itens que atendem às combinações possíveis entre os campos.

Uma segunda abordagem possível é a navegação pelos resultados de buscas com URLs montadas a partir de palavras de interesse [10]. Para que isto seja possível, é preciso que se conheça previamente a URL base para as consultas e que a busca no site possa ser textual.

Com este método é impossível saber qual a cobertura alcançada pelo rastreador, já que a priori não é possível conhecer os sub-conjuntos de páginas relacionadas a cada palavra.

Além disso, há o problema das escolhas das palavras utilizadas para maximizar o número de downloads de páginas únicas. Ntoulas discute políticas para escolha das palavras-chave. Em [10] faz uma comparação entre as políticas randômicas, baseadas na frequência de aparecimento de termos e políticas adaptativas, sendo esta última a mais eficiente pois utiliza termos utilizados nas páginas de retorno. Ntoulas observou que esta ainda é a política mais eficiente para sites com múltiplos idiomas.

2.3 Casamento de Esquemas - Schema Matching

Um dos problemas centrais para as potenciais aplicações que utilizarão nosso framework é o problema de mapeamento entre esquemas de dados. É fato que cada fonte pesquisada pelo rastreador (*crawler*) exporta, na maior parte das vezes, um esquema de dados bem diferente do esquema de dados do projeto. Construir estes mapeamentos de forma semi-automática é um grande desafio.

Wang [24] propõe um mecanismo de obtenção dos esquemas de dados de uma fonte da internet através de uma técnica de *Query Probing*, baseada na utilização de instâncias previamente conhecidas pelo esquema local. Através de cálculos realizados em matrizes de ocorrência das palavras pesquisadas é possível determinar, de forma automática, esquemas, do mesmo domínio do esquema local. Experimentos realizados por Brauner [16,17], utilizando web services e dados do domínio de Geoinformática mostraram resultados satisfatórios. Uma das maiores dificuldades deste método é realizar o parsing automático das informações obtidas nas páginas de resultados das buscas.

2.4 Casamento de entidades – Entity Matching

As soluções mais comuns para resolução do problema de casamento de entidades (*Entity Matching*), problema central endereçado neste trabalho, são compostas de algoritmos de cálculo de similaridade entre objetos e algum algoritmo de clusterização ou classificação [3, 8, 11, 23]. Geralmente os algoritmos de similaridade são aplicados sobre os atributos dos objetos e os algoritmos de clusterização/classificação separam itens duplicados de acordo com as similaridades calculadas. No caso ideal, os objetos que estão no mesmo grupo são referentes a mesma entidade real. A seguir discutimos alguns conceitos básicos que apóiam este trabalho:

Este é um problema que atraiu a atenção de pesquisadores de diferentes áreas, incluindo Bancos de Dados, Mineração de Dados, Inteligência Artificial e Processamento em Linguagem Natural. Na literatura este problema se apresenta através de diversos nomes, tais como “Record Linkage” [28], “Merge/Purge” [33], “Reference Conciliation” [34], “Entity Resolution” [35] ou “Duplicate Detection” [11].

Davis propôs um algoritmo para comparação de cadeias de caracteres contendo múltiplas palavras [5]. Chama a atenção neste trabalho a variedade de técnicas empregadas visando a melhoria do algoritmo proposto. Foram utilizadas tabelas de equivalência de caracteres e dicionários para lidar com problemas tais como abreviações, inversões e omissões de palavras. Também foram executados testes sobre nomes de pessoas e endereços gerando bons resultados.

É possível ainda explorar a utilização de algoritmos genéticos para resolver o problema de deduplicação de registros e pareamento de esquemas. Esta solução foi proposta por Carvalho [18] e mostrou-se eficiente quando comparada com técnicas tradicionais. A técnica combina diferentes evidências extraídas dos dados armazenados para sugerir funções de deduplicação capazes de identificar situações onde dois registros são, de fato, replicados.

Pereira [23] propôs o framework WER, que implementa um método de resolução de entidades baseado em informações disponíveis na Web. O método consiste em coletar informações sobre as referências, submetê-las como consultas a uma máquina de busca Web, analisar, e extrair informações para desambiguação criando um arquivo de autoridades. Para isto o autor se utilizou de técnicas de clustering, bem como de algoritmos de cálculo de similaridade entre cadeias

de caracteres. O trabalho foi enriquecido com uma solução para a resolução de nomes de autores em citações bibliográficas.

Köpcke e Rahm [8] compararam frameworks para Entity Matching. Neste trabalho foram definidos requisitos e classificações para este tipo de ferramenta. Segundo os autores, existem quatro requisitos básicos para estes tipos de framework:

1. Efecácia: O objetivo principal de um framework para o pareamento de entidades é a capacidade de identificar duplicatas respeitando valores de recall e precisão, ou seja, deve-se conseguir separar as entidades duplicatas das consideradas não duplicatas.
2. Eficiência: O problema deve ser resolvido de forma rápida, mesmo em se tratando de grandes volumes de dados.
3. Generalidade: Os métodos utilizados devem ser aplicáveis a diversos domínios, e a diferentes modelos de dados, e.g., XML, relacional. Além disto, os métodos devem ser aplicáveis tanto de forma online quanto off-line. Ferramentas de ETL (Extraction, Transformation e Loading) para a mineração de dados são exemplos deste tipo de framework que funciona de forma off-line.
4. Baixo esforço: O trabalho manual necessário para execução do framework deve ser o menor possível. Idealmente, o framework deve ser auto ajustável e aprender a reconhecer duplicatas automaticamente à medida em que é executado.

Os autores afirmam, ainda, que é possível classificar frameworks através dos critérios de tipos de entidade, métodos de bloqueio (separação de candidatos visando aumento de eficiência) e métodos de identificação de duplicatas.

Um dos frameworks citados neste artigo é o MARLIN de Bilenko [3]. Segundo o último, utilizar apenas funções de similaridade entre cadeia de caracteres não é suficiente para encontrar entidades duplicatas. Dependendo do domínio, palavras ou caracteres podem ter significado ou não para o cálculo de similaridade. Por exemplo, a palavra “Street” pode ser ignorada quando comparamos endereços, mas não pode quando comparamos nomes próprios como “Wall Street Journal”. Além disto, caracteres podem ser omitidos ou substituídos sem variação de significado: “St.” é a abreviação de “Street”.

Para comparar entidades, Bilenko propõe um algoritmo de distância de edição adaptável, onde cada operação pode ter um peso diferente de acordo com o

domínio. Por exemplo, substituir o caractere “-” por “/” pode ser insignificante para comparação de registros de telefones, mas não para registros de operações matemáticas.

O framework MARLIN é capaz de ajustar os pesos de cada operação deste algoritmo utilizando uma SVM (Support Vector Machine) em dois momentos: no primeiro, para treinar os pesos de cada operação de edição, e, a seguir, para determinar qual a melhor combinação de matchers do passo anterior. O framework ainda apresenta dois métodos para um treinamento semi-automático da SVM: um estático, e outro baseado em treino de casos negativos, mais apropriado para bases de dados legadas.

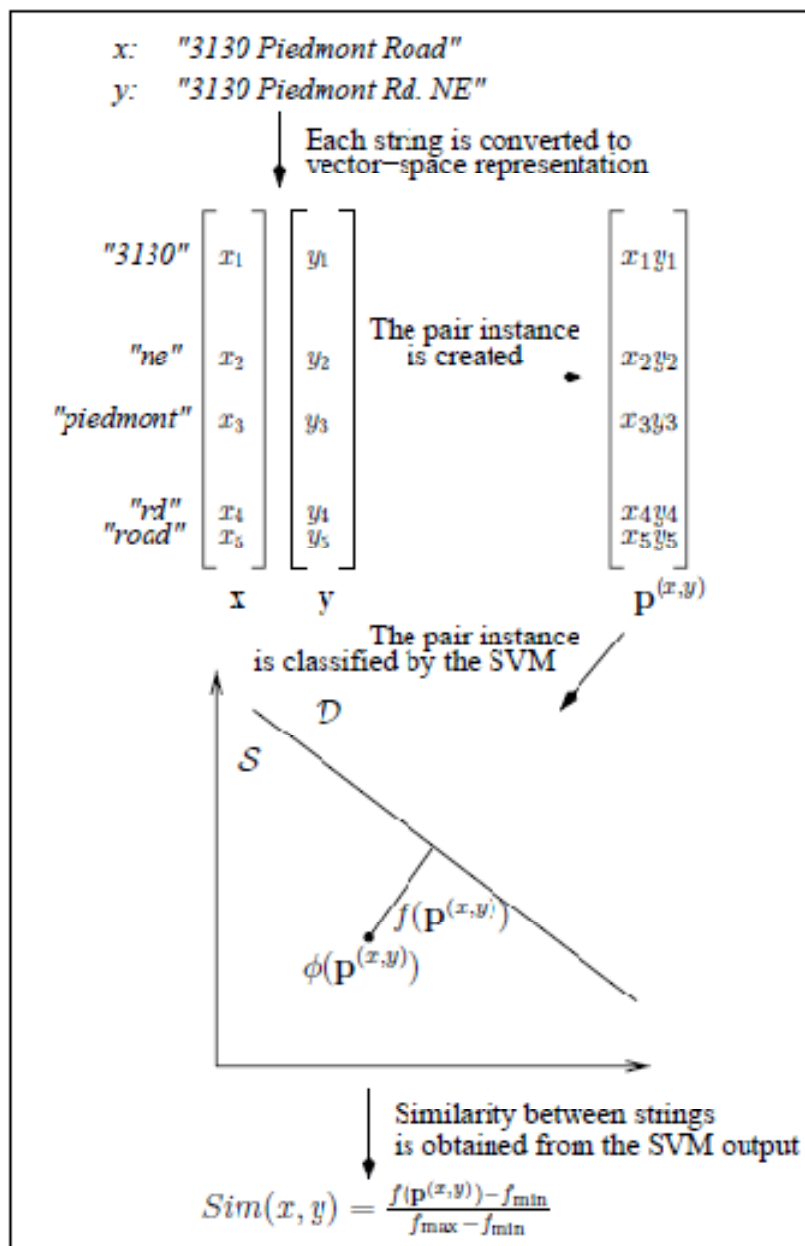


Figura 5 - Visão geral do algoritmo de similaridade de MARLIN [3]

Nota-se que todos os frameworks apresentados no estudo de Köpcke utilizaram bases de dados previamente conhecidas, e utilizaram bases de treinamento grandes para uma boa calibração dos algoritmos de classificação. É necessário que os usuários destes classifiquem manualmente uma série de instâncias do modelo previamente. Esta é uma tarefa muito difícil para aplicações que utilizam apenas bancos de dados web, já que até coleta destes dados é manual.

O framework UDD [11], segundo os autores, é o primeiro que resolve o problema de detecção de duplicatas online e o primeiro que consegue obter vantagens das dissimilaridades existentes sobre os registros de uma única base de dados Web.

Su [11] afirma que qualquer técnica de treinamento do classificador *offline* é inadequada, já que os resultados das consultas enviadas a esses sistemas é altamente dependente das palavras utilizadas, e os resultados representam apenas uma pequena porção de todos os dados disponíveis.

Dado que a quantidade de instâncias a ser comparadas é relativamente pequena, o autor afirma que não é necessário preocupar-se com aspectos de design referentes a minimizar o número de comparações entre entidades (*Blocking Methods*).

O algoritmo do UDD se ajusta conforme as entradas. Inicialmente são calculados pesos para cada campo da entidade utilizando uma função de similaridade entre cadeias de caracteres. Um classificador é treinado com estes dados, identificando duplicatas. Os pesos de cada campo são ajustados de acordo com as saídas do classificador. A iteração só termina quando não há mais novas duplicatas identificadas.

Como o algoritmo é iterativo, Su afirma que é necessário escolher um classificador que seja insensível a relação da quantidade de casos positivos e negativos. A escolha foi de um classificador SVM com kernel linear.

2.5 Funções de Similaridade entre Cadeias de Caracteres

A maior parte das funções de comparação entre cadeia de caracteres mapeiam as cadeias em um número real, i.e., quanto maior o valor, maior a similaridade entre as cadeia comparadas. É possível também medir a similaridade entre

cadeia de caracteres utilizando-se funções de distância. Estas são análogas as funções de comparação, com a diferença que, quanto menor o valor, maior a similaridade.

Essas funções podem ser classificadas conforme a técnica utilizada: baseada em caracteres, em tokens (termos), ou híbridas. Discutimos estas técnicas a seguir.

2.5.1 Funções baseadas em caracteres

2.5.1.1 Distância de Levenshtein

A similaridade por distância de edição, ou Distância de Levenshtein [25] é a função mais conhecida baseada em caracteres. Dadas duas cadeia de caracteres é retornado um valor de acordo com as operações de edição, com respectivos pesos. São operações de substituição, deleção e adição.

O algoritmo é criado segundo o paradigma da programação dinâmica e é de complexidade $O(mn)$, onde m e n são o tamanho de cada uma das cadeias de caracteres. A função de calculo da similaridade é definida como:

$$edSim(s_i, s_j) = 1 - \frac{ed(s_i, s_j)}{\min(|s_i|, |s_j|)}$$

Onde $ed(s_i, s_j)$ é a função de cálculo da distância de edição e $\min(s_i, s_j)$ retorna o tamanho mínimo das cadeias de caracteres.

2.5.1.2 Soundex (Busca Fonética)

A similaridade Soundex (Busca Fonética) [26] é um algoritmo de indexação fonética de palavras, originalmente no idioma inglês. O algoritmo calcula a similaridade ignorando homófonos previamente codificados. Este algoritmo é útil para encontrar duplicatas de nomes próprios e já foi usado em aplicações tais como censo americano.

Na codificação Soundex cada palavra corresponde a uma letra consoante (a primeira do nome) concatenada com três números de 0 a 6, que variam de acordo com uma determinada tabela. Consoantes com o mesmo som dividem o mesmo dígito.

Palavras como “Robert” e “Rupert” retornam a mesma cadeia de caractere “R163”, na versão inglesa do algoritmo.

2.5.1.3 Jaro-Winkler

A similaridade Jaro [27] é baseada no número e ordem de caracteres comuns entre duas cadeias de caracteres. Dadas duas cadeias de caracteres s_i e s_j , é dito que um caractere na cadeia s_i é igual a um caractere na cadeia s_j se somente ele está contido em um intervalo de distância. A transposição T é a distância em caracteres para igualar as posições. A função de similaridade é dada por:

$$Jaro(s_i, s_j) = \frac{1}{3} \times \left(\frac{|s'_i|}{|s_i|} + \frac{|s'_j|}{|s_j|} + \frac{|s'_i| - T_{s'_i, s'_j}}{s'_i} \right)$$

A extensão promovida por Winkler [28] utiliza um fator de escala p (valor padrão 0.1) para dar maior ênfase as distâncias encontradas nos primeiros caracteres.

$$Jaro-Winkler(s_i, s_j) = Jaro(s_i, s_j) + p \times l \times (1 - Jaro(s_i, s_j))$$

Esta é uma função de similaridade apropriada para cadeia de caracteres pequenas, e.g. nomes próprios, como o algoritmo Soundex.

2.5.2 Funções baseadas em tokens

Muitas funções baseadas em tokens que são descritas a seguir utilizam o modelo “Vector Space Model” (VSM) [38]. Este é um modelo algébrico, utilizado pela primeira vez nos anos 60 no sistema SMART, para representação de documentos que serve para filtragem de informação, indexação e consultas.

Neste modelo, documentos e consultas são representados como vetores onde cada dimensão corresponde a um termo, geralmente uma palavra. Cada termo é acompanhado de um peso de acordo com a frequência do termo no vetor, sendo estes calculados de diversas formas.

$$d_j = (w_{1,j}, w_{2,j}, \dots, w_{t,j})$$

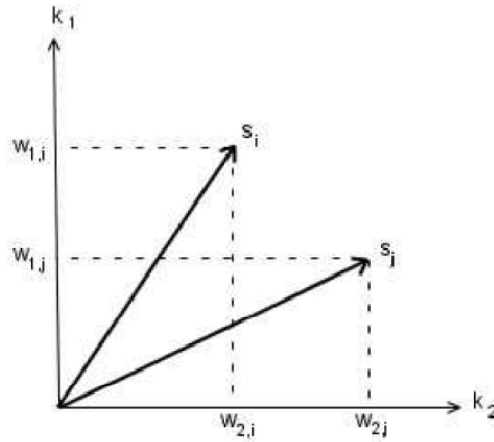


Figura 6 - Representação gráfica de 2 documentos no VSM

É possível realizar consultas sobre os documentos calculando o cosseno do ângulo entre 2 vetores. Quanto mais próximo a 1, mais semelhante é o documento.

2.5.2.1 Cosine Similarity

A similaridade de cossenos [1] é uma função baseada em *tokens* que mede a similaridade entre duas cadeia de caracteres utilizando o *Vector Space Model*. Esta é uma função útil para calcular a relevância de palavras em documentos, através do cálculo do cosseno entre dois vetores.

Dados dois vetores, um vetor s contendo cadeias de caracteres, e outro vetor k , contendo *tokens* destas cadeias, associamos um peso w a cada token de acordo com a frequência em que aparecem no vetor de cadeias de caracteres. O cosseno do ângulo entre os dois vetores dá a similaridade entre as cadeias de caracteres.

$$sim(s_i, s_j) = \frac{\sum_{l=1}^t w_{l,i} \times w_{l,j}}{\sqrt{\sum_{l=1}^t w_{l,i}^2} \times \sqrt{\sum_{l=1}^t w_{l,j}^2}}$$

Os pesos podem ser calculados da seguinte forma:

$$w_{l,j} = \frac{freq_{l,j}}{\max_p freq_{p,j}} \times \log \frac{N}{n_i}$$

Esta estratégia de cálculo é a conhecida pelo nome de frequência de termos (term-frequency) e também é utilizada no algoritmo TF-IDF [29]. Quanto maior a frequência, mais importante é a palavra.

2.5.2.2 *Dice Coefficient*

Esta função de cálculo de similaridade também utiliza o *Vector Space Model*. O seu resultado é o valor obtido a partir do dobro da interseção dos termos comuns sobre a soma dos termos das duas cadeias de caracteres comparadas [6]. Se o coeficiente é 1, então os vetores são idênticos, enquanto que o valor zero indica que os vetores são ortogonais.

2.5.2.3 *Block Distance (Taxicab Geometry)*

Esta distância também é chamada de L1, “city block” ou “Manhattan Distance”. Dados dois vetores de cadeia de caracteres no *Vector Space Model*, a similaridade é calculada através das somas das diferenças entre os vetores [30].

$$L_1(q, r) = \sum_y |q(y) - r(y)|$$

2.5.2.4 *Jaccard*

Esta função de similaridade é muito popular e semelhante às funções Cosine e Dice, e foi muito utilizada para medir a similaridade de componentes químicos, no início do século passado. O coeficiente de Jaccard [31] mede a similaridade entre dois conjuntos dividindo a interseção dos conjuntos pela sua união, segundo a formula abaixo.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

2.5.3 Funções híbridas

2.5.3.1 Monge-Elkan

Alvaro E. Monge e Charles P. Elkan [32] propuseram uma estratégia de matching recursivo. O algoritmo proposto pelos autores calcula a similaridade entre cadeias de caracteres utilizando as sub cadeias A e B derivadas a partir das cadeias s e t respectivamente, e uma função de similaridade auxiliar. O resultado da medida de similaridade proposta é obtido através da seguinte fórmula:

$$sim(s, t) = \frac{1}{K} \sum_{i=1}^K \max_{j=1}^L sim'(A_i, B_j)$$

Onde K é o número de tokens da cadeia de caracteres A, e L o número de tokens da cadeia de caracteres B.

Em um estudo produzido por Cohen [19] utilizando bases de dados de nomes próprios com as funções de similaridade: Levenstein, Jaro, Jaro-Winkler, Smith-Waterman, Jaccard, Monge-Elkan e TF-IDF, SFS e Soft-TFIDF, chegou-se a conclusão que a função de Monge-Elkan apresentou os melhores resultados no que diz respeito a qualidade da identificação de duplicatas na categoria “algoritmos baseados em distância de edição”, seguido por Jaro-Winkler. Dentre as funções baseadas em tokens, a função TF-IDF foi a que obteve melhores resultados.

2.6 Classificação

Entendemos classificação como o processo de atribuir etiquetas, ou categorias, a itens baseando-se em um modelo de classificação previamente definido, com regras que incluem árvores de decisão ou fórmulas matemáticas. Geralmente um algoritmo de classificação recebe um conjunto de itens já previamente etiquetados (conjunto de treinamento) e, a partir desse conjunto, é capaz de classificar outros itens. Classificadores são úteis quando as classes, ou etiquetas, são conhecidas a priori [23].

Quando não conhecemos as classes, devemos utilizar algoritmos de clusterização. Esta é uma técnica de mineração de dados que visa agrupar elementos

de um conjunto baseando-se em alguma medida de similaridade, podendo assim descobrir relacionamentos ocultos entre as entidades agrupadas.

Um exemplo de clusterização é o algoritmo K-Nearest Neighbor (NN) [39]. No início cada item é representado por um nó de grafo e representa um cluster. Se um nó é similar a outro, é criada uma aresta do nó inicial até um dos nós desse cluster. Ao final do processo temos os itens agrupados de acordo com a similaridade.

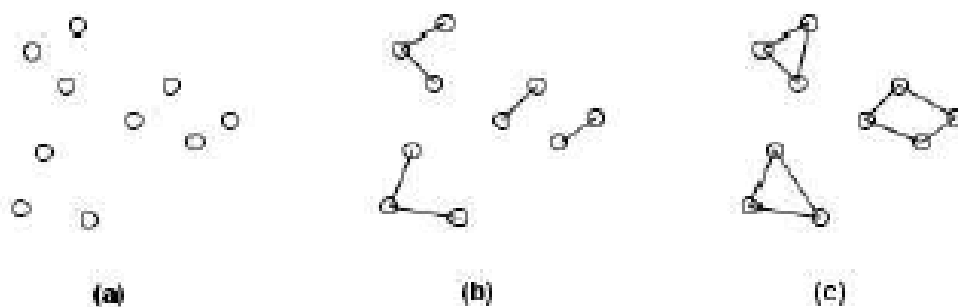


Figura 7 - Exemplo de KNN

2.6.1.1 Técnicas de Classificação

Classificadores Bayesianos são baseados na Teoria de Bayes [37]. Eles calculam as probabilidades de uma amostra pertencer a uma determinada classe. Classificadores Bayesianos assumem que todos os atributos de uma entidade são independentes. Atribui-se pesos com os valores 0 ou 1, de acordo com a probabilidade de um item pertencer a uma categoria. (Esse tipo de classificador é muito simples, porém, segundo Zhang, demonstra-se tão eficiente como outros métodos para muitos problemas difíceis [36].

É possível criar classificadores que utilizam a técnica de Vector Space Model de modo análogo aos classificadores bayesianos [14]. Ao invés de utilizar as probabilidades para categorizar itens, são utilizados os pesos das frequências. Esse tipo de classificador apresenta algumas vantagens em relação aos classificadores do tipo Bayesianos, tais como possibilitar o cálculo de graus de similaridade contínuos, uma vez que os pesos não são binários, o que permite ordenar os elementos de acordo com sua relevância. No entanto, apresentam limitações:

cadeias de caractere muito grandes podem apresentar pouca similaridade, já que as frequências dos termos ficam reduzidas e a busca de cadeias de caracteres deve ser realizada a partir de cadeias exatas.

Classificadores do tipo árvore de decisão são classificadores que realizam diversos testes sobre os atributos da entidade. O caminho realizado na árvore é que classifica a entidade com determinada etiqueta.

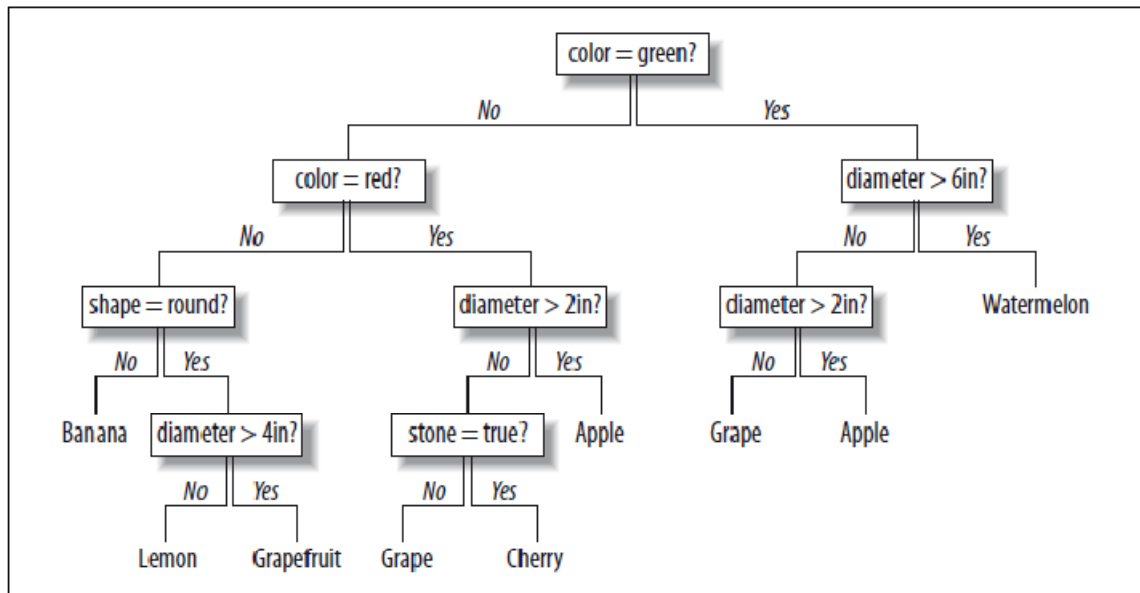


Figura 8 - Exemplo de árvore de decisão

A maior desvantagem desse tipo de classificador é que não há adaptabilidade, uma vez que não é possível treinar o classificador uma vez que este foi criado. Além disso, uma vez que um item percorre um caminho na árvore, não pode mudar de ramo na árvore.

A vantagem é a simplicidade e a possibilidade de acompanhar a classificação.

Quando as regras para formação de uma árvore de decisão não são conhecidas previamente, podem ser utilizados outros recursos tais como redes neurais para construção de classificadores. Rede Neural em computação é um modelo matemático inspirado nas conexões entre os neurônios do cérebro humano. Dadas as entradas, são produzidas saídas e são retroalimentadas as camadas internas da rede, ajustando seus pesos (aprendizado).

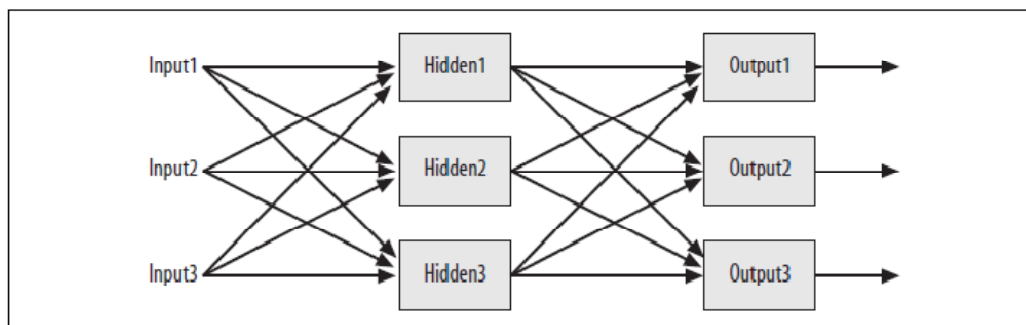


Figura 9 – Exemplo de estrutura básica de nós de uma rede neural

Entre as vantagens oferecidas por redes neurais se encontram a capacidade de trabalhar com funções não lineares (utilizadas para excitação dos neurônios), e o treinamento incremental. As dificuldades em utilizar redes neurais são (1) definir os conjuntos de treinamento, e (2) selecionar a estrutura de rede mais adequada para representar o problema a ser analisado.

A técnica de Support Vector Machines (SVM) utiliza o aprendizado de máquina para problemas de reconhecimento de padrão. A técnica foi proposta por Vapnik [42], e vem sendo usada com grande sucesso em problemas de categorização de texto. O método SVM consiste em uma abordagem geométrica para o problema de classificação, onde cada elemento conjunto de treinamento pode ser visto como um ponto x_i num espaço R^M , e o aprendizado consiste em dividir os elementos positivos dos negativos no espaço. Uma vez finalizada a fase de treinamento, classificar novos itens significa descobrir em qual lado do hiperplano um novo item deve ser colocado [43].

Para garantir bons resultados com a técnica de SVMs, é necessário dispor de uma grande quantidade de exemplos de treinamento, e escolher bem os parâmetros da SVM, e.g., a função de transformação do seu *kernel*, responsável por realizar os cálculos que dividem os elementos em classes distintas.

2.7 Resumo:

Neste capítulo apresentamos a conceitos fundamentais que apoiaram o desenvolvimento deste trabalho. Foram apresentadas algumas técnicas de recuperação de dados na web, construção de rastreadores (*crawler*), casamento de esquemas (*schema matching*) e casamento de entidades (*entity matching*).

Ainda foram apresentados conceitos básicos e algumas técnicas que permitem o cálculo de similaridade entre cadeias de caracteres. Discutimos fun-

ções baseadas em caracteres, em termos (tokens) e híbridas. Para embasar a exposição das funções baseadas em tokens e híbridas foi exposta uma introdução do *Vector Space Model* (VSM), um modelo para armazenamento e recuperação de informações de texto.

No fim do capítulo foi introduzido o conceito de classificação e clusterização. Para exemplificar clusterização o algoritmo KNN foi apresentado. Algumas técnicas de classificação foram apresentadas, entre elas as baseadas na Teoria de Bayes, VSM, árvores de decisão, redes neurais e *Vector Space Machines*.

3 Estratégia para o enriquecimento de informações

Podemos resumir o processo de enriquecimento de informações em duas grandes etapas, a saber, busca e incorporação de dados, como ilustrado na Figura 10, a seguir.

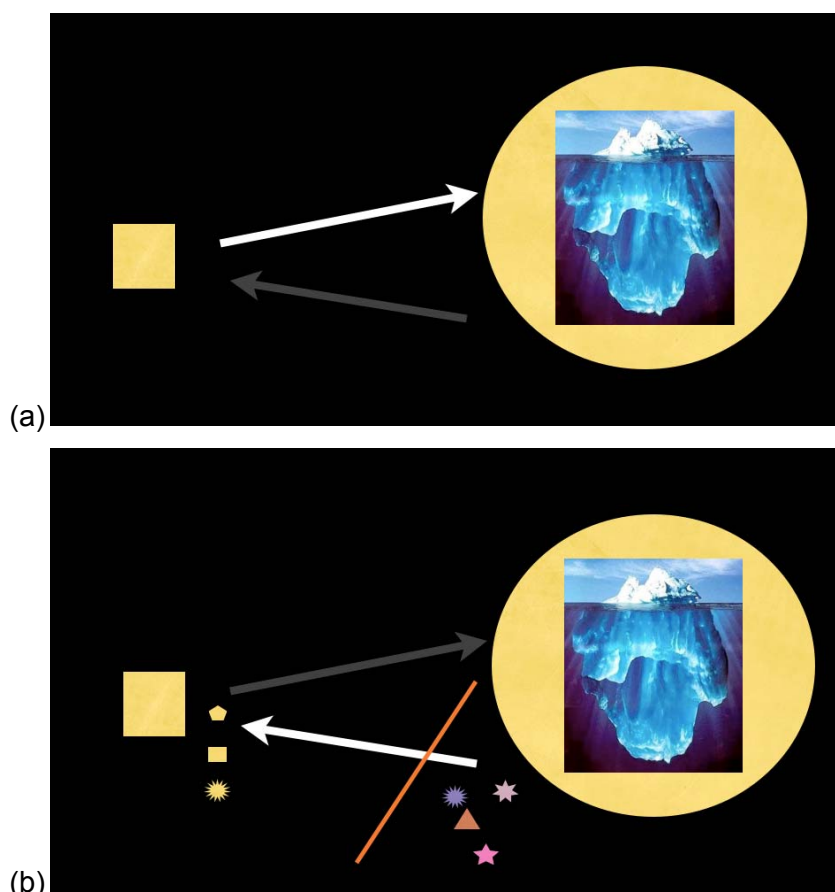


Figura 10 - Visão geral do processo de enriquecimento de informações proposto: (a) Busca, (b) Incorporação de dados

O processo de busca consiste na (1) escolha da fonte de dados, pois esta deve apresentar um formulário de entrada para realização de buscas, (2) seleção das palavras-chave adequadas ao domínio visando minimizar o número de consultas para uma determinada entidade. O processo de incorporação, por sua vez, consiste em selecionar aqueles que, com alguma segurança, representam a mesma entidade local e extrair informações adicionais que a entidade local não possui. No que se segue apresentamos as estratégias utilizadas pelos processos de busca e incorporação de informações, que constituem a estratégia de enriquecimento proposta neste trabalho. Daremos especial atenção a etapa de i-

identificação de duplicatas, ou deduplicação, na Deep Web, contribuição principal do trabalho.

3.1 Busca de informações na Deep Web

A busca por informações na Deep Web, como demonstrada por trabalhos anteriores [22, 10, 40, 41], resume-se a escolher um termo (ou uma combinação de termos) e executar a consulta em um formulário de busca. Os links gerados pela busca são rastreados.

O algoritmo proposto para nosso rastreador segue o mesmo princípio. Dada uma entrada, no caso um objeto modelo, é gerado um conjunto de termos para consulta. Para cada consulta é executada uma busca e os resultados são rastreados.

No nosso framework o rastreamento resulta na geração de um conjunto de objetos candidatos a duplicata. Para esse comportamento, cada página possui um parser específico capaz de retirar as informações que desejamos. Deste modo, o rastreador (*crawler*) só segue links para páginas que possuem um parser associado.

Entrada: Instância de objeto M (modelo)

Saída: Conjunto P de produtos

Q é um conjunto de consultas

$q(m,d)$ é uma função geradora de consultas

Algoritmo:

1. P = vazio
2. Q = $q(m)$
3. Para cada consulta de Q
4. Executa a consulta
5. Se a página resultado possui um parser associado
6. Para cada produto p extraído pelo parser
7. P = P + p
8. Se p possui um link para outra página
9. Executa o link e vai para o passo 6.
10. Retorna P

Figura 11 - Algoritmo para a busca de informações na Deep Web

3.2 Incorporação de novas informações a partir da identificação de duplicatas

O foco desta etapa é encontrar uma solução para o problema de deduplicação em bancos de dados Web pertencentes a um mesmo domínio de forma a identificar informações que possam ser, posteriormente, incorporadas a base original.

Entendemos que o problema de detecção de duplicatas da seguinte maneira: Dado dois registros distintos, podemos afirmar que estes representam a mesma entidade real se o resultado da aplicação de uma função de cálculo de similaridade, que tipicamente retorna um valor numérico, for maior que um limiar determinado.

Na prática há duas questões a serem resolvidas: eficiência e eficácia [4]. A eficiência refere-se a capacidade de resolver o problema utilizando os recursos computacionais de forma racional. É necessário ressaltar que já que lidamos com grandes quantidades de registros e os métodos de comparação muitas vezes tem ordem de complexidade elevada.

A eficácia refere-se a qualidade da medida de similaridade. Uma medida é eficaz se, no caso ideal, for capaz de reconhecer, em um conjunto de possíveis candidatos, todas as duplicatas e todas as não duplicatas. Central a esta questão é a determinação do limiar de detecção, que deve ser ajustado a cada domínio de aplicação [3]. Valores altos resultam em alta precisão, mas baixos recalls. Assim, a eficácia está ligada a capacidade de maximizar estes valores.

O resultado da detecção de duplicatas é a criação de uma identificação única para cada referência. Se duas ou mais entidades compartilham a mesma identificação, são ditas duplicatas.

O passo seguinte a detecção de duplicatas é a fusão dos dados. Para que haja a integração dos dados é preciso escolher uma estratégia que consiga atingir dois objetivos: completeza e concisão. A completeza é aumentada quando inserimos novos elementos ou novas fontes de dados ao processo de integração, já a concisão é alcançada quando conseguimos remover dados redundantes e mesclar informações comuns de objetos considerados duplicatas [4].

O problema central da fusão de dados é lidar com incertezas e contradições que podem ser apresentados em dados de fontes diferentes. O processo de fusão de dados é definido como a mediação de conflitos tanto no nível dos esquemas de dados como no nível dos próprios dados [4].

3.3 Algoritmo

A partir da revisão da literatura apresentada no capítulo 2, da nossa experiência pregressa, e levando em consideração as necessidades apresentadas, propomos o algoritmo para identificação de duplicatas a seguir:

Entrada: Conjunto P de instâncias candidatas a duplicata

Conjunto N de modelos não duplicados

Saída: Conjunto D de duplicatas

C é o classificador que identifica duplicatas em P.

f é uma função de cálculo de similaridade

Algoritmo:

1. D = vazio.
2. Os parâmetros de entrada de C e f são atribuídos
3. Para cada n em N
4. c é treinado com n (positivo)
5. Para cada p em P
6. c é treinado com p utilizando f
7. Para cada p em P
8. Se p é classificado por p como positivo
9. D = D + p
10. Retorna D

Figura 12 - Algoritmo para o casamento de entidades

Para solucionar o problema de detecção de duplicatas online podemos utilizar um classificador em conjunto com uma função de cálculo de similaridade.

Como não é possível utilizar uma base de dados confiável para treinamento, dadas as razões expostas no capítulo 1, podemos treinar o classificador com todos os candidatos utilizando uma função de cálculo de similaridade.

No fim selecionamos os candidatos duplicados testando-os no classificador que fora parametrizado.

3.3.1 Comentário sobre o escopo do trabalho

É importante deixar claro as hipóteses sobre as quais repousa este trabalho, de modo a facilitar o entendimento do escopo da solução proposta. Em primeiro lugar assumimos a existência de um esquema global pré-definido para cada uma das fontes da Deep Web consultadas. Este esquema é compatível com o esquema utilizado pela base de origem, ou seja, conhece-se o tipo de objeto a ser retornado nas buscas e estes fazem parte do mesmo domínio dos objeto alvo cuja informação se deseja enriquecer.

Além disso, temos que todo resultado retornado por uma busca possui um parser específico capaz de retirar informações e criar uma instância do objeto que obedece ao esquema global. Logo, a estratégia propostas não é capaz de reconhecer, de forma totalmente automática, informações relativas ao objeto alvo.

Finalmente, também assumimos que todos os objetos retornados de uma mesma fonte podem ser identificados univocamente pela sua URL e/ou nome, podendo assim guardar uma referencia da duplicata apenas pelo seu link. Em outras palavras, cada objeto possui uma página própria com suas informações.

3.4 Resumo:

Neste capítulo apresentamos uma visão geral da estratégia proposta neste trabalho para o enriquecimento de dados de uma base local a partir de informações obtidas na Deep Web.

Definimos que o processo de enriquecimento pode ser dividido em dois macro processos: (1) busca e (2) incorporação de dados. Para cada um desses processos propomos um algoritmo. O algoritmo de busca envolve o conhecimento da construção de rastreadores para Deep Web, enquanto que o algoritmo de incorporação mescla conceitos de casamento de entidades e de Fusão de Dados (Data Fusion).

No final da exposição discutimos o escopo da solução proposta.

4 Framework Proposto

Neste capítulo apresentamos um framework que implementa a estratégia para enriquecimento de dados a partir de informações da Deep Web, descrita no capítulo anterior.

4.1 Arquitetura Proposta

Na Figura 13, a seguir, apresentamos o diagrama de classes do framework proposto.

O framework foi desenvolvido utilizando-se a linguagem Java dados os mecanismos do paradigma de orientação a objetos que facilitam a construção de frameworks e facilidade de obtenção de *frameworks* e API's *open-souce* que apóiam a construção deste sistema.

As classes principais do framework estão no pacote `br.com.scrapers`. As classes referentes a agentes estão no sub-pacote “agents”.

Toda documentação referente as classes do framework e suas instâncias está registrada no formato *Javadoc*, bem como estão presentes da pasta “docs/javadoc” do projeto e podem ser acessadas através do link <http://www.winetag.com.br/scrapers/>.

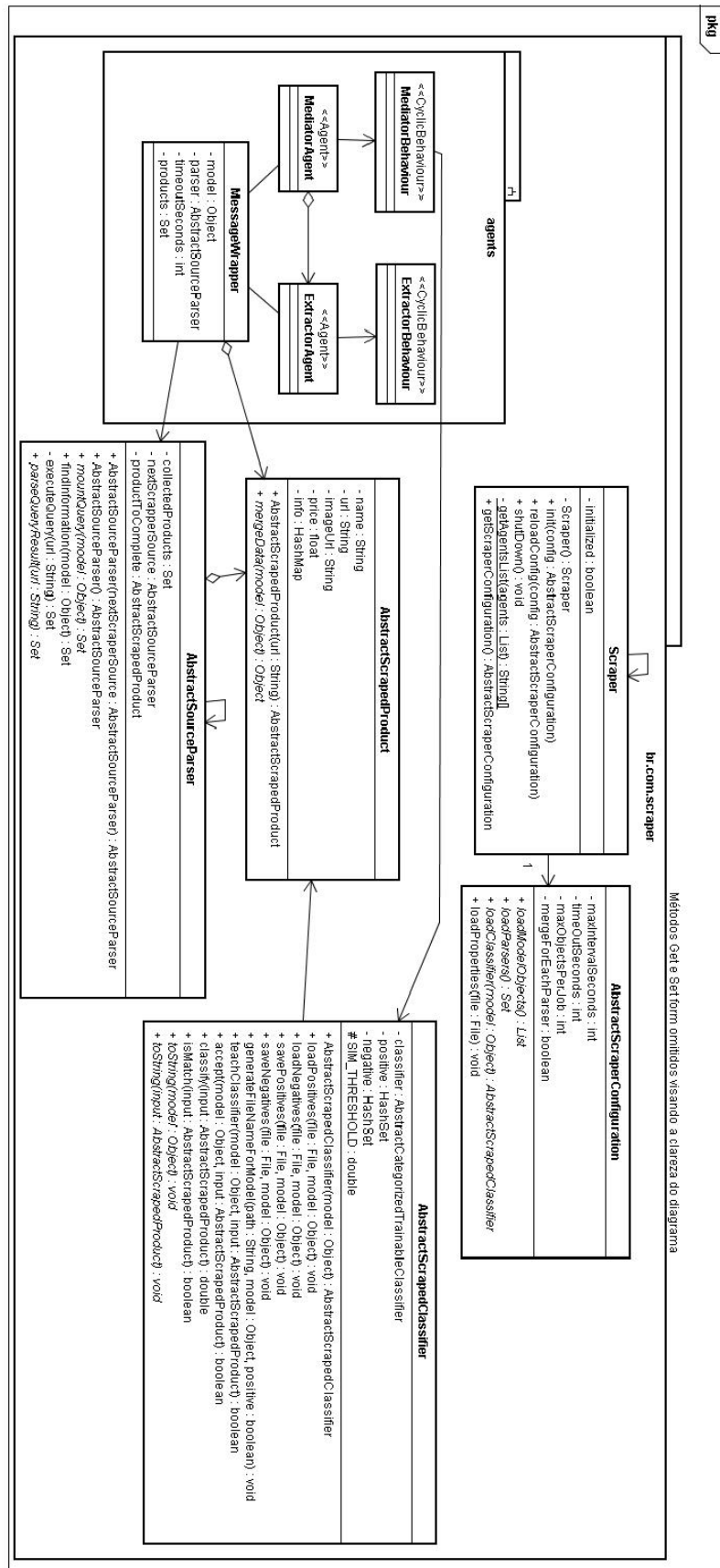


Figura 13 - Diagrama de Classes

A classe `Scraper` é a responsável pelo controle da execução de todo o processo. Ela configura, inicia e desliga os agentes que fazem a extração, transformação e carregamento dos dados. `Scraper` é uma classe que segue o padrão *Singleton* e deve ser inicializada com um objeto de configuração. É possível utilizar o framework sem esta classe, no entanto, sem o comportamento esperado.

A classe abstrata `AbstractScraperConfiguration` é responsável por armazenar as configurações do framework. Ela deve conter informações sobre o tempo máximo de intervalo entre cada extração (`maxIntervalSeconds`), o tempo máximo de espera pela resposta de um agente de extração (`timeOutSeconds`), a quantidade máxima de objetos a procurar por execução (`maxObjectsPerJob`) e se o framework deve realizar merge dos dados para cada parser, ou seja para cada fonte, ou para todas escolhendo o melhor candidato. As classes que estendem esta devem sobrescrever os métodos `loadModelObjects` retornando os produtos que devem ser utilizados nas buscas, `loadParsers` retornando os parsers responsáveis por interpretar as páginas web retornadas nas buscas e `loadClassifier` retornando um classificador apropriado ao objeto que está sendo procurado.

A classe abstrata `AbstractScrapedProduct` representa os produtos encontrados nas buscas realizadas. Tem como atributos principais, comuns a maioria dos produtos disponíveis na internet, nome, URL da imagem, preço, informações. O atributo URL é o atributo principal e aquele que diferencia os produtos já que cada recurso da Web é acessado univocamente através de sua URL. As classes que estendem esta devem ser capazes de saber como realizar a operação de merge, sobrescrevendo o método `mergeData`.

A classe abstrata `AbstractSourceParser` é responsável por gerar as URLs de busca e realizar o parsing dos resultados. É possível concatenar classes (parsers) desse tipo passando outras instâncias como parâmetro. A classe guarda os produtos coletados em `collectedProducts`, e a instância a ser chamada em caso de concatenação em `nextScraperSource`. É possível passar `productToComplete` para esta nova instância afim de completar o produto já coletado com novas informações. As classes que estendem esta devem sobrescrever o método `mountQuery` retornando uma lista de URLs a serem exploradas e `parseQueryResult` retornando os produtos coletados. A atividade de exploração dos links (*crawling*) é realizada pelo método `executeQuery`.

A classe abstrata `AbstractScrapedClassifier` é a interface para um classificador a ser utilizado na instância do framework. As classes que estendem esta de-

vem sobrescrever os métodos `toString` para o modelo e o produto coletado gerando uma cadeia de caractere que seja capaz de identificar cada um destes. É necessário também sobrescrever a classe `accept` retornando se um modelo pode ser dito similar a um produto. O método `generateFileNameForModel` é utilizado para resolver o correto local de armazenamento dos arquivos temporários do classificador. Os métodos `classify` e `isMatch` retornam o grau de similaridade entre os produtos após utilizar o classificador treinado. O método `teachClassifier` ensina o classificador de acordo com a resposta dada pelo método `accept`, logo este é o método que deve ser sobrescrito com maior atenção. O método `accept`, que pode ser subscrito, está implementado como padrão para comparar as duas cadeia de caracteres geradas pelos métodos `toString`, utilizando como limiar o valor `SIM_THRESHOLD` de 90%.

Foi escolhido utilizar um classificador baseado no Vector Space Model, que é similar a um classificador bayesiano.

Os agentes contidos no framework manipulam as classes descritas de maneira correta, de acordo com as configurações dadas. O fato do sistema ser multi-agentes diminui o tempo total de procura por um determinado produto já que é possível realizar buscas em diversos sites ao mesmo tempo.

`MediatorAgent` é o agente que centraliza as operações. Ele cria agentes do tipo `ExtractorAgent` para cada `AbstractSourceParser` disponível. Os agentes utilizam instâncias de `MessageWrapper` para facilitar a troca de objetos dentro das mensagens. `MediatorAgent` ainda treina e classifica os produtos encontrados pelos agentes extratores afim de encontrar os mais similares.

O Diagrama de componentes apresenta as dependências externas do framework. Estas bibliotecas devem ser incluídas nos projetos que o utilizarão. Pode-se ver que é utilizado o framework JADE para apoio a criação de agentes, a API `Classifier4J` para a criação de classificadores e as API's `Simmetrics` e `SecondString` para cálculo de similaridade entre cadeias de caracteres.

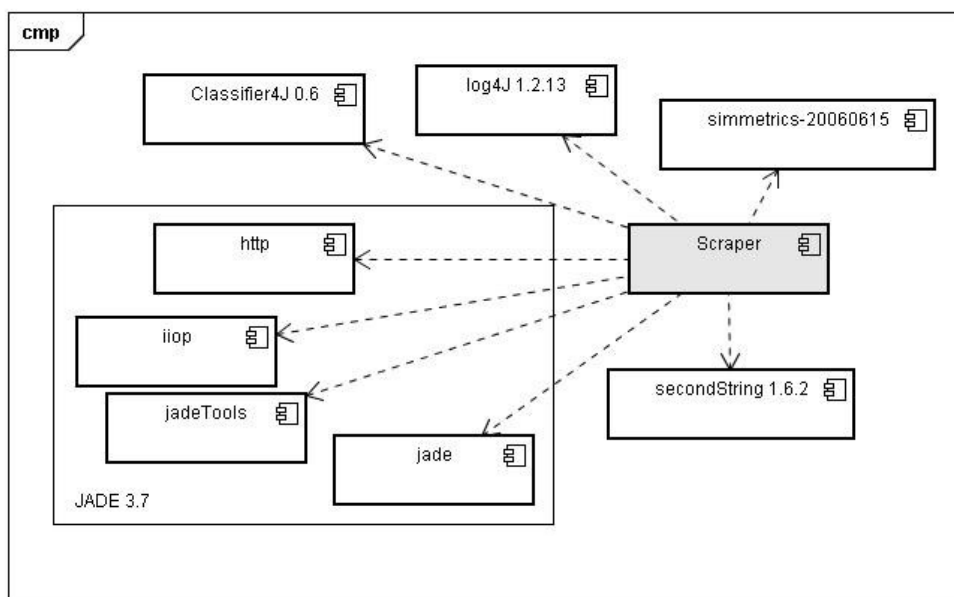


Figura 14 - Diagrama de Componentes do Framework

4.2 O Processo de Utilização do Framework Proposto

O framework proposto implementa a estratégia para busca e incorporação de dados descrita no capítulo anterior. Seu funcionamento pode ser descrito por cinco passos fundamentais. Nestes passos refinam os macro processos de busca (passos 1, 2 e 3) e incorporação (passos 4 e 5) da estratégia proposta.

Passo 1: Inicialização: Uma entidade modelo (tupla), parsers específicos de fontes de dados e parâmetros adicionais são carregados no mediador.

Passo 2: Distribuição: Para cada fonte de dados, um agente extrator é criado.

Passo 3: Rastreamento de dados na Deep Web: Cada extrator realiza consultas sobre sua fonte diretamente URL's ou formulários web. Uma vez com a página de resultados, ele navega pelos links capturando informações sobre os candidatos a duplicata.

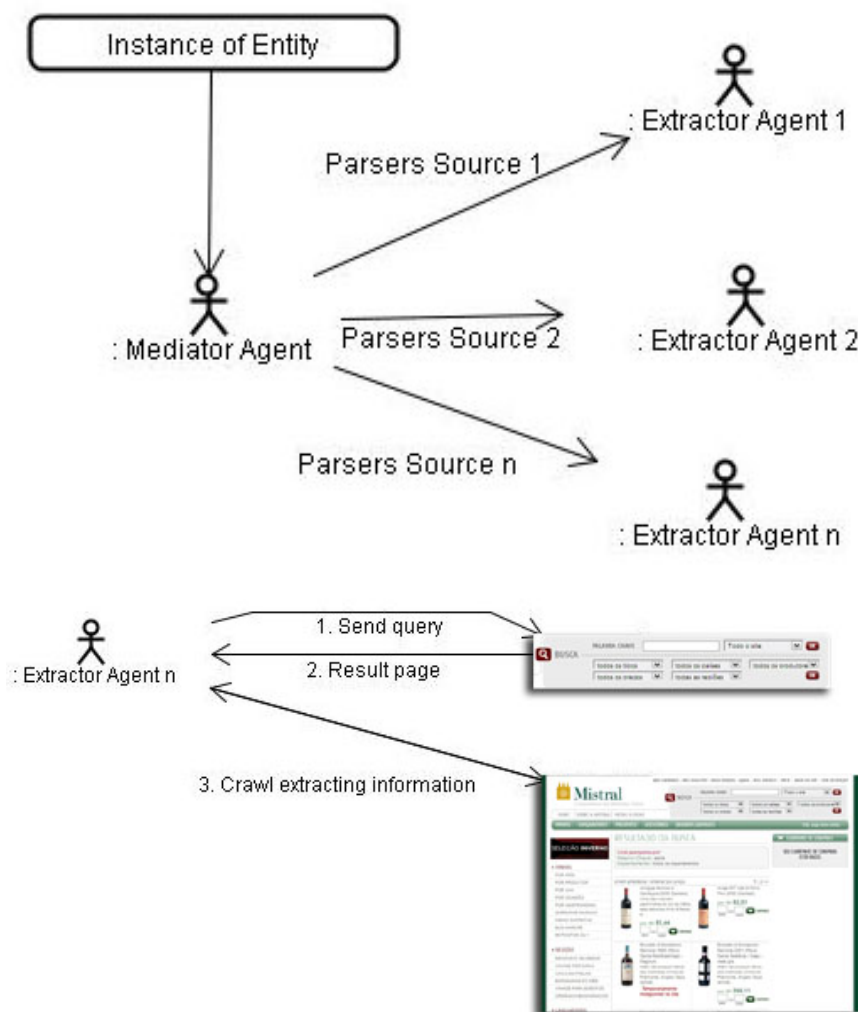


Figura 15 - Passos 1, 2 e 3 do processo (estratégia de busca)

É necessário observar que neste processo o framework não deve obter recursos de uma mesma fonte em um intervalo pequeno de tempo, como 5 segundos, para evitar esgotar os recursos do site que hospeda as informações alvo. Além disso, este não deve obter recursos de fontes que não autorizam a captura de dados automatizada.

Passo 4: Casamento de Entidades (Entity Matching): Cada agente extrator retorna seus candidatos a duplicata ao mediador. Após este passo o mediador treina o classificador com o modelo original e os dados das duplicatas. O classificador é treinado online, utilizando uma função de cálculo de similaridade que compara os candidatos com o modelo (tupla). Um candidato é considerado positivo se o valor retornado pela função for maior que um limiar fixo, caso contrário é considerado negativo.

O classificador escolhido para este framework é um classificador baseado no Vector Space Model descrito em [14]. Simples e eficiente, este classificador utiliza as frequências dos termos com pesos, ao invés de probabilidades. Este tipo de classificador apresenta vantagens quando comparados aos classificadores Bayesianos. Como os pesos não são binários, é possível obter graus de similaridade de forma contínua, possibilitando ordenar os itens classificados. No entanto esta abordagem apresenta algumas limitações como em casos de comparação de cadeia de caracteres grandes e buscas exatas.

O classificador escolhido separa os itens em duas classes, aqueles com o valor maior ou igual ao valor de corte (*cutoff*) e os abaixo deste.

Passo 5: Fusão dos dados: O agente mediador mescla as informações adicionais obtidas que não estão nos registros originais do modelo, tais como preço e imagem, a partir das duplicatas descobertas.

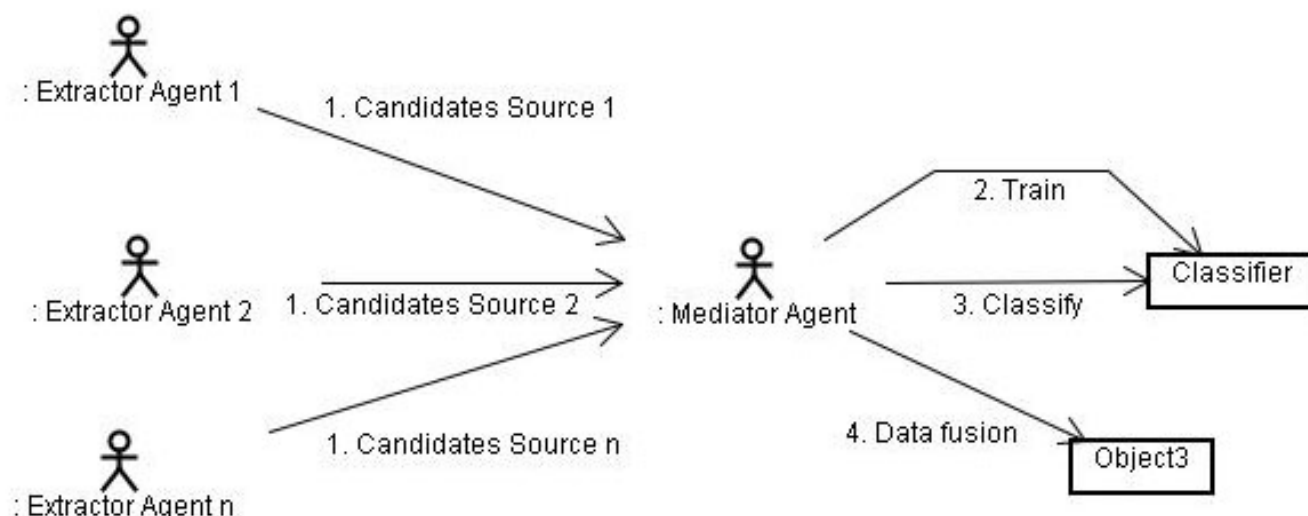


Figura 16 - Passos 4 e 5 do processo (incorporação)

4.3 Resumo:

Neste capítulo foi apresentada a solução codificada que implementa a estratégia de enriquecimento de dados proposta no Capítulo anterior. São discutidos em detalhes aspectos relacionados ao código, e a instanciação do framework, para um domínio qualquer.

A partir da apresentação do processo de enriquecimento de dados, discutimos detalhes de implementação e funcionamento do framework computacional proposto, tais como as responsabilidades dadas a cada agente de software na realização de atividades ligadas à extração de informações e na mediação de dados. De forma a facilitar a implementação, decompomos a estratégia proposta no Capítulo 3 em cinco passos, a saber: (1) Inicialização, (2) Distribuição, (3) Rastreamento de Informações na Deep Web, (4) Casamento de Entidades e (5) Fusão de dados (Data Fusion). O macro processo de busca envolve os passos 1, 2 e 3, enquanto que o de incorporação envolve os processos 4 e 5.

5 Winetag.com.br: um estudo de caso

Foi realizado um estudo de caso instanciando o framework proposto. O domínio escolhido foi o da comercialização de vinhos, que apresenta ângulos e desafios muito interessantes para os problemas da busca de informação em bases da Deep Web e de detecção de duplicatas. O objetivo dessa aplicação foi utilizar a nossa abordagem para enriquecer uma base de dados pré-existente.


O primeiro passo do processo foi estudar quais os atributos deste tipo de entidade seriam suficientes para identificá-los. Assumimos que um vinho típico é produzido por uma vinícola e apresenta um nome. O nome pode ser composto de um substantivo próprio, assim como elementos que ajudam a caracterizar o vinho. Estes elementos podem referenciar regiões produtoras (típico em vinhos franceses como “Rothschild Médoc R  serve Sp  ciale”, da regi  o de M  doc), podem ter o nome de uma classifica  o oficial (“DOCG” em “Batasiolo Barbaresco DOCG”), ou podem conter nomes pr  prios da vin  cola onde foram produzidos (“Reservado” em “Concha Y Toro Reservado Cabernet Sauvignon”).

Em muitas ocasi  es, vin  colas produzem o mesmo tipo de vinho, i.e. vinhos produzidos utilizando-se uma mesma metodologia, a partir de diferentes uvas.    comum os r  tulos dos vinhos serem diferenciados pelo nome de sua uva. Exemplos s  o “Cobos Felino Chardonnay” e “Cobos Felino Cabernet Sauvignon”. O mesmo padr  o    observado quando o produtor deseja destacar que utilizou totalmente, ou em grande parte, apenas um tipo de uva para produ  o de um vinho.

Neste estudo de caso estamos lidando exclusivamente com vinhos finos, desta forma    necess  rio considerar tamb  m a safra, pois al  m do vinho apresentar diferen  as caracter  sticas de acordo com o ano da colheita das uvas, o mercado tamb  m comercializa os vinhos de acordo com a qualidade obtida em cada safra.

Informa  es adicionais, tais como o volume da garrafa e percentual alco  lico, normalmente est  o presentes nos r  tulos, n  o s  o relevantes para a diferencia  o de vinhos, pois s  o informa  es que repetem-se freq  entemente nestas inst  ncias e, portanto, n  o tem utilidade no processo de diferencia  o.

Desta forma, para identifica  o de vinhos, no cen  rio web utilizamos os seguintes tr  s atributos, do tipo cadeia de caracteres (string):

 (vinícola, nome, safra)

Com este modelo, foi iniciada a instanciação de nosso framework. O passo seguinte foi identificar as fontes de dados: sites de e-commerce de vinhos. Os candidatos foram escolhidos levando em consideração a nacionalidade e o tamanho da coleção de produtos disponíveis para coleta. Neste estudo de caso levamos em conta apenas sites brasileiros por dois motivos: o primeiro que a aplicação que terá informações enriquecidas é voltada para o mercado nacional e o segundo é que seriam inseridas outras dificuldades neste trabalho, desnecessariamente, relacionadas ao idioma.

5.1 Metodologia de trabalho

5.1.1 Identificação das fontes de dados

Nesta etapa foi feito um levantamento sobre possíveis sites candidatos a fontes de pesquisa entre os sites de e-commerce de vinhos disponíveis. Os candidatos foram escolhidos levando em consideração a nacionalidade (neste caso consideramos apenas sites localizados no Brasil), e o tamanho da coleção de produtos disponíveis para coleta. Também foi observada a viabilidade de se conseguir obter informações através de URL's. Foram escolhidas as lojas Expand³, Grand Cru⁴, Mistral⁵, Vinci⁶, Wine.com.br⁷, Zahir⁸ e Estação do Vinho⁹.

5.1.2 Casamento de Esquemas - Schema Matching

Nesta etapa foi realizado o mapeamento entre modelo de objeto o qual deseja-se enriquecer conteúdo e objetos retornados nas pesquisas nas fontes. Foram identificadas as entidades relacionadas aos produtos disponíveis do site e qual o relacionamento delas com o modelo proposto pelo site do projeto.

3 www.expand.com.br

4 www.grancru.com.br

5 www.mistral.com.br

6 www.vincivinhos.com.br

7 www.wine.com.br

8 www.zahir.com.br

9 www.estacaodovinho.com.br

Para o mapeamento correto verificou-se ainda a necessidade de criar mediadores para converter dados. Verificou-se por exemplo que cada loja apresentava uma classificação própria de categoria do vinho. Para resolver isso foram criadas tabelas de conversão.

É preciso notar algumas peculiaridades relacionadas as uvas. Existem nomes de uva que referenciam a mesma entidade. Essas variações foram criadas em grande parte devido a questões culturais. Existe, por exemplo, a uva Tempranillo que também é chamada de Tinto Fino, Cencibel ou Aragonês.

5.1.3 Descoberta de padrões em URL's e montagem de consultas

Nesta etapa foram estudadas quais as regras de montagem de URL deveriam ser seguidas para executar as consultas nas fontes. Códigos referentes a categorias ou departamentos de loja e tamanho de nomes necessários para busca foram estudados.

A eficácia do framework é altamente dependente dos resultados obtidos por estas consultas, deste modo, é preciso encontrar meios de produzir consultas que, idealmente, retornem apenas a entidade procurada. Se a consulta for muito genérica será preciso confrontar o modelo com uma quantidade muito grande de possíveis duplicatas. Por outro lado, se muito específica, pode nunca retornar a referência para a entidade procurada.

É comum encontrar formulários de busca de sites que executam consultas diretamente sobre um banco de dados utilizando expressões do tipo “like” no campo título da tabela correspondente. Este tipo de implementação impede que façamos consultas utilizando o nome completo, pois raramente o nome é exatamente igual, (ou uma sub cadeia exata) do nome armazenado na fonte utilizada.

A estratégia utilizada nesta instância foi de executar diversas consultas, explorando varias formas do modelo procurado. Para cada nome de vinho, foi enviada uma consulta por token presente no nome. Para evitar a realização de consultas genéricas demais, tokens com termos comuns (*stopwords*), tokens com nomes de uvas, palavras de classificação como “Reserva”, ou nomes de regiões produtoras foram suprimidas.

Desta forma esperamos garantir que ao menos uma das consultas (aquela com um termo mais significativo para o vinho) retorne o vinho desejado nas primeiras posições. Por outro lado, com esta estratégia é possível que um mes-

mo vinho retorne no resultado de diversas consultas, ou que o vinho procurado nunca apareça (nos casos onde o nome do vinho é composto por uma combinação de termos muito comum) .

5.1.4 Criação de parsers das fontes de dados

Como o framework não contempla nenhum mecanismo de Identificação automática entidades nas páginas retornadas pelas pesquisas, foi preciso identificar as tags, manualmente, capazes de encapsular informação em cada delas. Para encontrar as marcações corretas foram utilizados aplicativos como o *Developer Tools do Internet Explorer 8¹⁰*, já que fica fácil de separar os elementos visualmente.

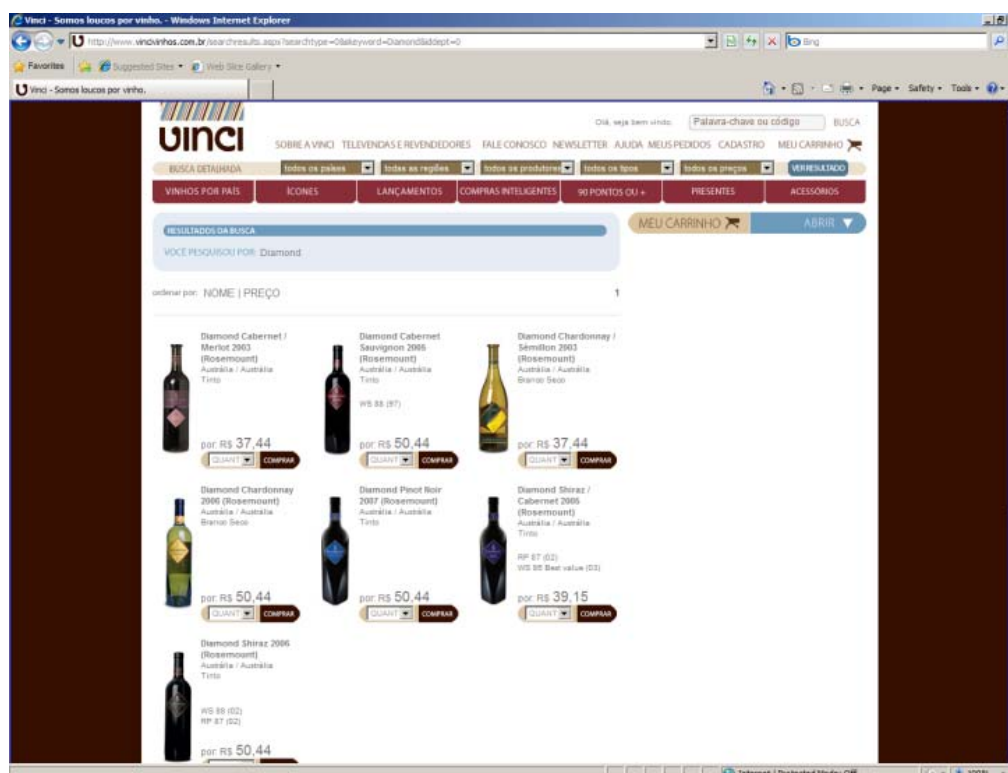


Figura 17 - Exemplo de fonte de dados

¹⁰ <http://msdn.microsoft.com/en-us/library/dd565628%28VS.85%29.aspx>

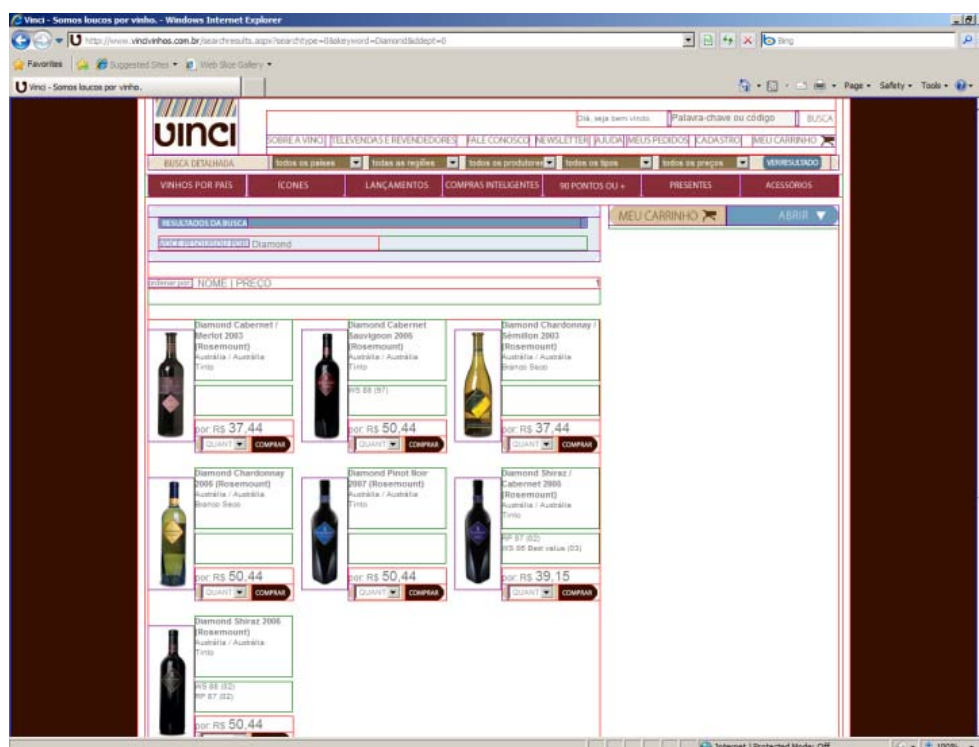


Figura 18 - Exemplo elementos identificados pelo Developer Tools

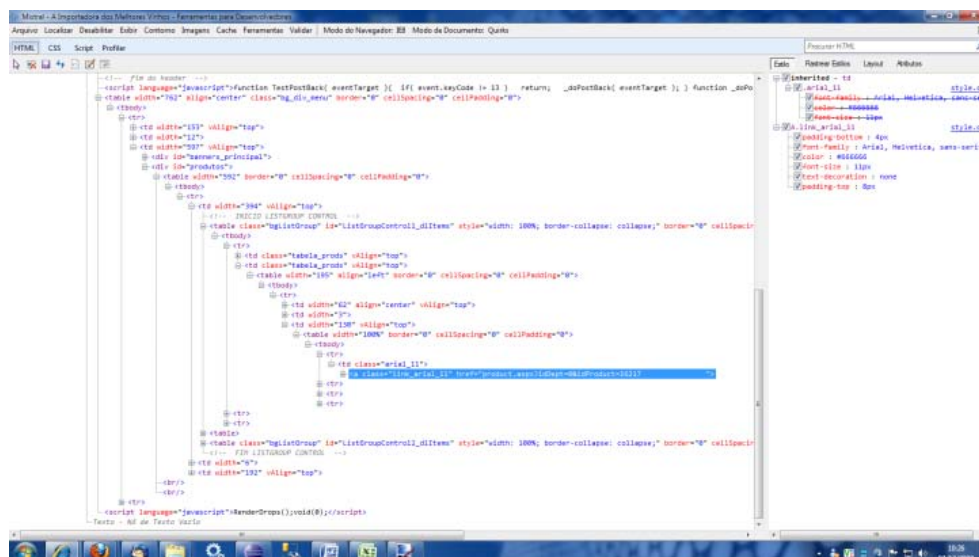


Figura 19 - Elementos destacados no código-fonte

Todas as fontes disponibilizaram seus dados apenas no formato HTML, com alguns metadados, utilizados para aplicação do layout. Foi utilizada a técnica de Screen Scraping, referenciada no capítulo 2, para a captura da informação destas páginas.

Para cada página de busca, e de informações detalhadas de vinhos, foram codificados parsers específicos. O código foi escrito visando oferecer alguma flexibilidade as mudanças de layout que pudessem vir a ocorrer nas páginas de pesquisa.

Para evitar problemas durante o processo de captura de informação, foram criados testes automatizados de extração de entidades. Assim, antes da execução do aplicativo era possível verificar se os parsers para cada loja estavam adequados (corretos).

5.1.5 Escolha de funções de cálculo similaridade

Foram estudadas funções de cálculo de similaridade capazes de atender aos requisitos deste projeto. Durante esta etapa, realizamos uma serie de testes de modo a medir a confiabilidade das medidas de cálculo de similaridade escolhidas para o domínio dos vinhos, em particular. Para um conjunto base de instâncias de vinho foram calculados os graus de similaridade utilizando 15 funções de baseadas em caracteres, tokens e híbridas.

A partir dos resultados foram determinadas as funções que seriam mais adequadas para este domínio. Foram escolhidas as funções capazes de gerar valores de similaridade altos, para itens considerados duplicatas, e valores baixos para itens considerados não duplicatas. Verificou-se que as funções Dice, Cosine, Block e TFIDF, baseadas em tokens, foram as mais adequadas para nosso problema.

Para evitar que erros de ortografia interferissem nos resultados, todas as cadeia de caracteres utilizadas para comparação, de modelos e possíveis duplicatas, foram filtradas. O primeiro filtro consistiu em transformar todos os caracteres em minúsculos.

O segundo filtro consistiu em transformar alguns caracteres no equivalente de uma tabela. Caracteres acentuados, por exemplo, foram transformados em caracteres simples. Caracteres especiais como aspas foram substituídos por espaços. Esta é a mesma idéia apresentada no trabalho de Davis [5].

5.1.6 Classificador

Nesta etapa, já com as funções de cálculo de similaridade definidas, codificou-se o classificador. No framework é preciso sobrescrever um método para

treinar o classificador, diferenciando casos positivos e negativos. Para ser aceito como caso exemplo, definiu-se que os vinhos que apresentassem similaridade maior que um determinado limiar, a ser escolhido empiricamente, nas funções Cosine, Dice, Block ou TFIDF seriam aceitos como possíveis casos positivos, caso contrário seriam aceitos como casos negativos.

Treinar o classificador desta forma não implica em afirmar que o melhor objeto classificado é o que obteve maior similaridade, nem que somente os que tem similaridade maior que o valor limiar (*threshold*) definido serão classificados como positivo.

O classificador recebe como entrada sempre uma cadeia de caracteres representando a entidade do vinho (nome do produtor e do vinho) e retorna um valor referente ao resultado da classificação. Se o valor é maior que o valor de corte (*cutoff*) do classificador, definido empiricamente, podemos afirmar que o classificador considera o item como duplicata.

Como não incluímos a safra no processo de classificação, mesmo que classificado como positivo pelo classificador, o vinho só pode ser considerado duplicata se os valores das safras forem iguais. Não utilizar a safra no treinamento do classificador é importante pois o valor da safra, com quatro dígitos, deve ser comparado de forma exata. Se incluíssemos esta informação no classificador estaríamos incluindo ruído.

Nos testes realizados verificou-se que alguns vinhos do porto, com diferentes anos de envelhecimento no nome não foram identificados corretamente. O classificador não foi capaz de diferenciar os vinhos apenas por causa deste token numérico.

Se um vinho apresenta a safra e outro não, com o mesmo nome, não podemos afirmar nada, logo consideramos como caso negativo.

5.1.7 Fusão de dados

Após a identificação de duplicatas é preciso realizar a operação de merge dos dados. Dados provindos de diferentes fontes podem estar em diferentes formatos ou conter informações incompletas ou contraditórias. Como o objetivo principal da aplicação dos *Mashups* era apenas coletar dados de preços e imagens dos vinhos, optou-se pela estratégia mais simples. A primeira informação não existente na base é persistida.

Para fins ilustrativos, temos algumas das classes representadas no diagrama da figura 20:

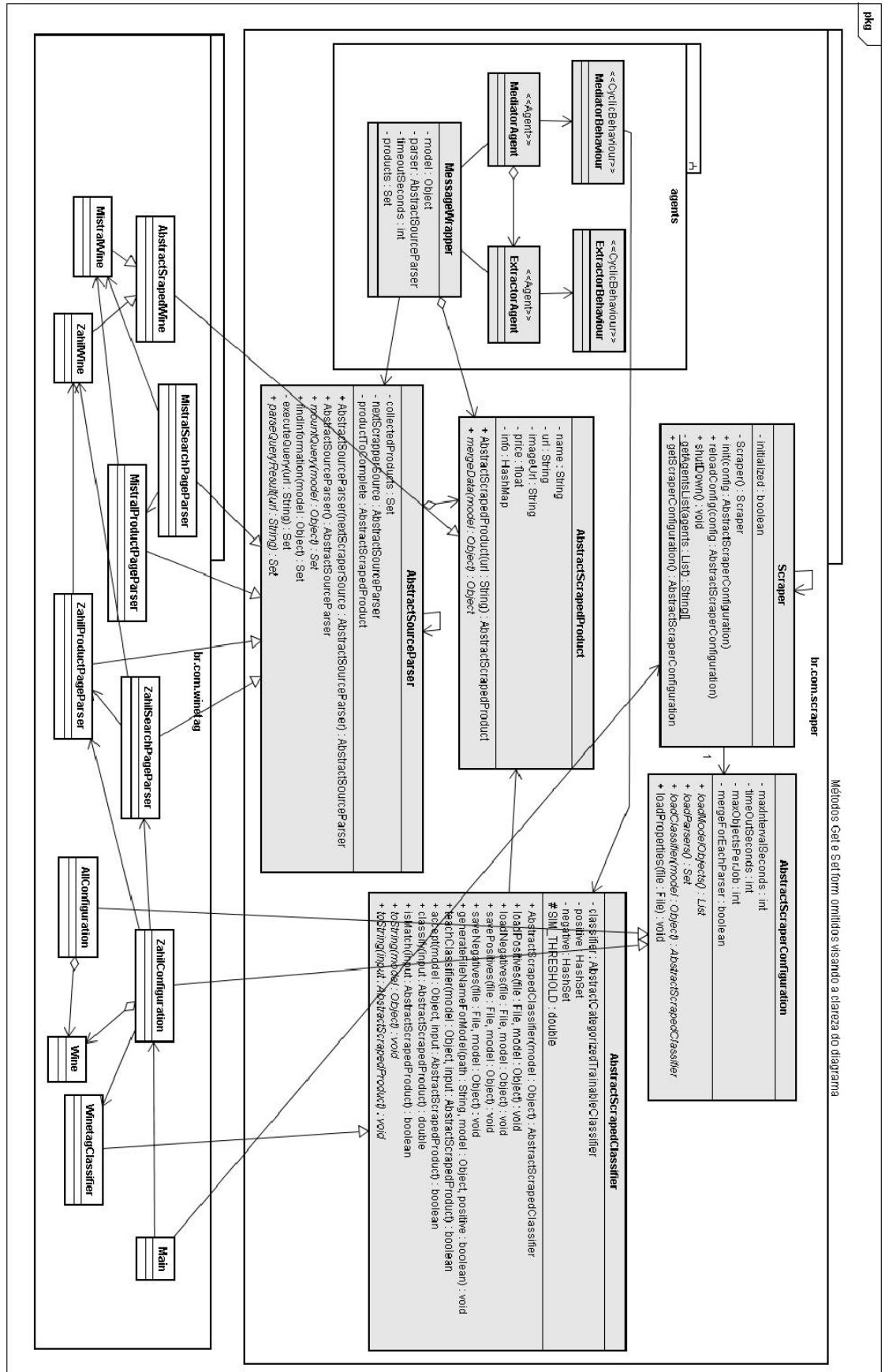


Figura 20 – Diagrama de classes para instância de vinhos

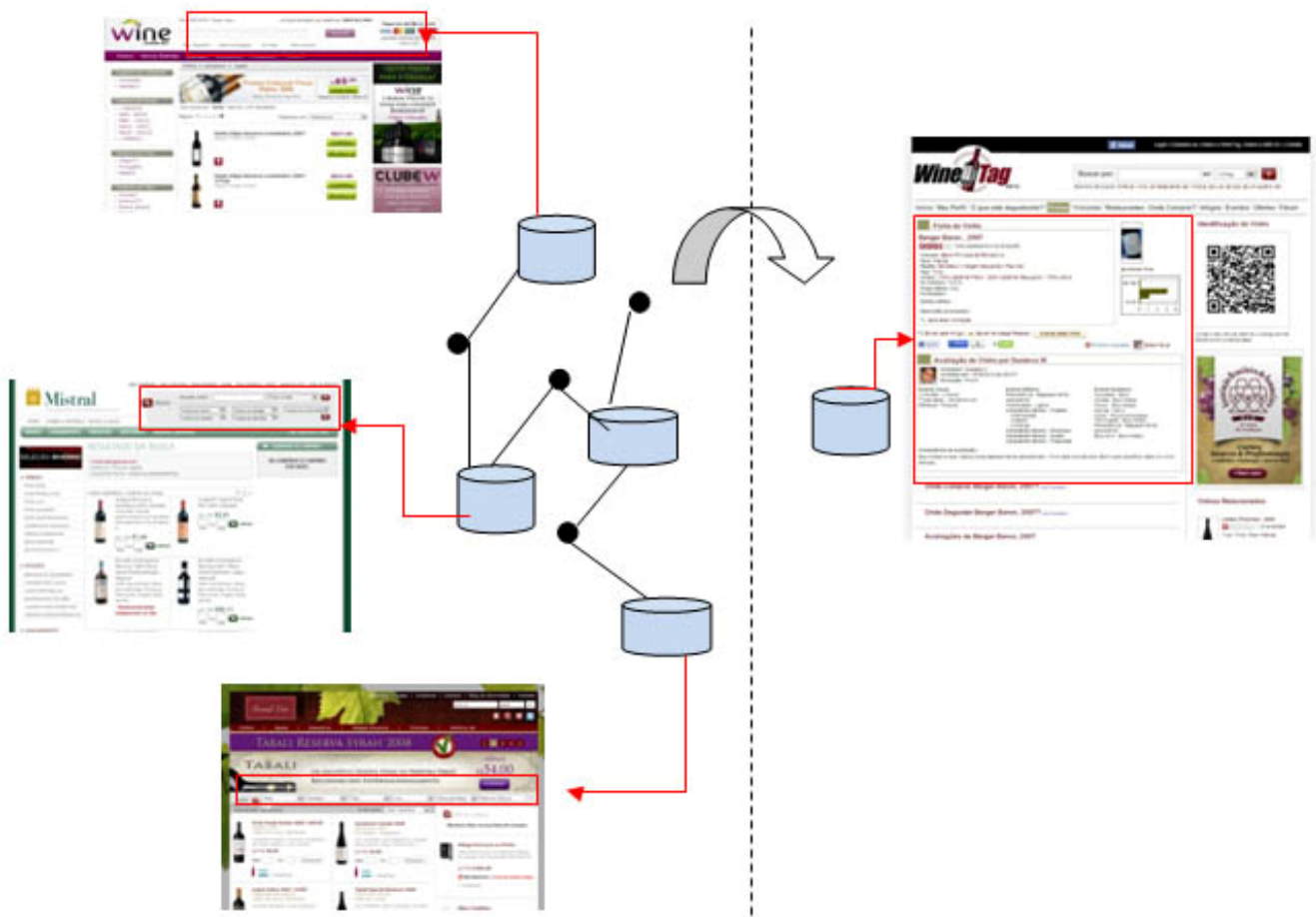


Figura 21 - Winetag.com.br: mashup com dados da Deep Web

5.2 Resultados

Para validar a solução proposta ao problema dos vinhos temos que nos apoiar em algumas métricas comuns de estatística para classificação. Estas métricas são baseadas nas quantidades de itens classificados corretamente como “positivo” (TP) e “negativo” (TN), além dos itens classificados incorretamente como “positivos” (FP) e “negativo” (FN). Temos as definições:

Precisão (*Precision*): É a probabilidade de que um item, aleatoriamente escolhido, seja relevante.

$$\text{Precision} = \frac{tp}{tp + fp}$$

Recall: É a probabilidade de que um item, aleatoriamente escolhido, seja recuperado em uma busca.

$$\text{Recall} = \frac{tp}{tp + fn}$$

Acurácia (*Accuracy*): Reflete o grau de certeza de que uma medida é reprodutível.

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

F-Measure: É uma média harmônica que combina precisão e recall. É útil para medir a qualidade da classificação.

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Foram realizadas consultas a duas fontes de dados em Fevereiro de 2010. Utilizamos uma amostra de 1.402 vinhos com nomes únicos, de safra maior que 2004. A instância do framework retornou 21.302 duplicatas, ou seja, vinhos que poderiam ser classificados como similares.

Deduzimos que temos ao menos 142 duplicatas corretas. Este valor foi estimado observando o maior número de casos considerados verdadeiramente positivos, menos a quantidade de falsos positivos encontrados nos testes.

Visando obter bons resultados e explorar o funcionamento do framework para este domínio, foram realizados experimentos variando os parâmetros “threshold” da função de similaridade e o “cutoff” do classificador. Estes são os principais parâmetros envolvidos na solução do problema de “Entity Matching” proposta por este framework.

Temos a tabela de itens considerados relevantes, ou seja, casos positivos e respectivamente a porcentagem de “merges” em relação a amostra:

		Cutoff					
Threshold	P	1.00	0.95	0.90	0.85	0.80	0.75
	1.00	29	29	32	66	85	125
	0.95	49	49	58	105	162	212
	0.90	79	79	85	128	187	236

Tabela 1 - Quantidade de classificações positivas (P)

Threshold	Merge	Cutoff					
		1.00	0.95	0.90	0.85	0.80	0.75
	1.00	2,07%	2,07%	2,28%	4,71%	6,06%	8,92%
	0.95	3,50%	3,50%	4,14%	7,49%	11,55%	15,12%
	0.90	5,63%	5,63%	6,06%	9,13%	13,34%	16,83%

Tabela 2 – Percentual de operações realizadas de merge em relação a amostra

Seguem as tabelas com os valores encontrados de TN, FP, FN, verificados manualmente:

Threshold	TP	Cutoff					
		1.00	0.95	0.90	0.85	0.80	0.75
	1.00	29	29	32	66	84	113
	0.95	49	49	58	103	144	177
	0.90	56	53	57	81	134	181

Tabela 3 - Quantidade de classificações de verdadeiramente positivas (TP)

Threshold	FP	Cutoff					
		1.00	0.95	0.90	0.85	0.80	0.75
	1.00	0	0	0	0	1	12
	0.95	0	0	0	2	18	35
	0.90	23	26	28	47	53	55

Tabela 4 – Quantidade de classificações falsamente positivas (FP)

Threshold	TN	Cutoff					
		1.00	0.95	0.90	0.85	0.80	0.75
	1.00	21.273	21.273	21.270	21.236	21.217	21.177
	0.95	21.253	21.253	21.244	21.297	21.140	21.090
	0.90	21.223	21.223	21.217	21.174	21.115	21.066

Tabela 5 – Quantidade de classificações verdadeiramente negativas (TN)

Threshold	FN	Cutoff					
		1.00	0.95	0.90	0.85	0.80	0.75
	1.00	113	113	110	76	59	41
	0.95	93	93	84	41	16	0
	0.90	109	115	113	108	61	16

Tabela 6 – Quantidade de classificações falsamente negativas (FN)

Nas tabelas a seguir apresentamos os resultados referentes a performance do framework:

Threshold	Precision	Cutoff					
		1.00	0.95	0.90	0.85	0.80	0.75
	1.00	100.00%	100.00%	100.00%	100.00%	98.82%	90.40%
	0.95	100.00%	100.00%	100.00%	98.10%	88.89%	83.49%
	0.90	70.89%	67.09%	67.06%	63.28%	71.66%	76.69%

Tabela 7 - Valores de Precisão

Threshold	Recall	Cutoff					
		1.00	0.95	0.90	0.85	0.80	0.75
	1.00	20.42%	20.42%	22.54%	46.48%	58.74%	73.38%
	0.95	34.51%	34.51%	40.85%	71.53%	90.00%	100.00%
	0.90	33.94%	31.55%	33.53%	42.86%	68.72%	91.88%

Tabela 8 - Valores de Recall

Threshold	Acurácia	Cutoff					
		1.00	0.95	0.90	0.85	0.80	0.75
	1.00	99.47%	99.47%	99.49%	99.64%	99.72%	99.75%
	0.95	99.57%	99.57%	99.61%	99.80%	99.84%	99.84%
	0.90	99.38%	99.34%	99.34%	99.28%	99.47%	99.67%

Tabela 9 - Valores de Acurácia

Threshold	F-measure	Cutoff					
		1.00	0.95	0.90	0.85	0.80	0.75
	1.00	33.92%	33.92%	36.78%	63.46%	73.68%	81.00%
	0.95	51.31%	51.31%	58.00%	82.73%	89.44%	91.00%
	0.90	45.90%	42.91%	44.71%	51.10%	70.16%	83.60%

Tabela 10 - Valores de F-Measure

O experimento foi útil para determinar quais seriam os valores de *threshold* e *cutoff* para nosso domínio. Como o framework ainda não é capaz de auto ajustar-se, temos que manualmente selecionar os valores que maximizam as métricas utilizadas: 0,95 para o *threshold* e 0,80 para o *cutoff* do classificador.

Esses valores foram obtidos empiricamente e são aqueles que mantêm precisão e recall balanceados com altas taxas. Esses valores devem ser os melhores para o domínio do nosso problema. Com eles obtivemos valores de 88,9% de precisão, 90% de recall e 89,44% de *F-Measure*.

5.3 Resumo:

Neste capítulo apresentamos um estudo de caso, que é a instanciação do framework, para o enriquecimento dos dados para o site Winetag.com.br¹¹. No início deste capítulo há uma breve discussão acerca de detalhes relacionados aos nomes de vinhos, domínio do site estudado. Após uma definição de quais atributos seriam suficientes para diferenciar cada objeto foi demonstrado todos os passos necessários para criação da solução.

A metodologia de trabalho, que contou com 7 fases: (1) identificação de fontes, (2) casamento de esquemas, (3) descoberta de padrões de URL e montagem de consultas, (4) codificação de parsers, (5) escolha de funções de cálculo de similaridade, (6) codificação do classificador e finalmente (7) fusão de dados, contém detalhes específicos do domínio dos vinhos que podem ser úteis para outras experiências de outros domínios.

Para medir o progresso realizado foi apresentado na seção de resultados as métricas que são utilizadas em trabalhos similares [11, 3, 19, 23] assim como o conjunto de testes que foi utilizado. A partir dos resultados obtidos variando dois parâmetros do framework: *threshold* da função de cálculo de similaridade e *cutoff* do classificador obtivemos os valores mais apropriados para esta instância.

¹¹ www.winetag.com.br

6 Trabalhos relacionados

Existem trabalhos na literatura que lidam com diversas questões envolvidas no desenvolvimento deste trabalho. Neste capítulo discutimos diferenças e similaridades na luz dos resultados apresentados nos capítulos anteriores.

6.1 Rastreador (crawler)s de Deep Web

Os trabalhos existentes na literatura [22, 10, 40, 41] demonstram aspectos envolvidos na construção de rastreadores para Deep Web. Estes trabalhos relacionados tem o objetivo de aumentar a cobertura de indexação de páginas por mecanismos de busca. Em nosso trabalho há a preocupação de direcionar o rastreador (*crawler*) para obtenção eficiente de duplicatas, montando consultas capazes de retornar apenas informações que possam ser adicionadas ao nosso banco de dados, já que o processo de incorporação de informações, descrito no capítulo 3, é custoso computacionalmente.

6.2 Casamento de Instâncias - Entity Matching

Relacionando os trabalhos apresentados no capítulo 2 com o framework apresentado, temos algumas diferenças e similaridades, a ver:

O framework UDD, publicado durante o desenvolvimento deste trabalho, é o primeiro, segundo os autores, a tentar realizar o casamento de entidades de forma online na Deep Web [11]. Este é o mesmo propósito deste.

O UDD de Su, assim como nosso trabalho, não apresenta um mecanismo de extração de dados das páginas. Cabe a quem instanciar o framework codificar um mecanismo de extração de informações das páginas.

As principais diferenças entre MARLIN [3], UDD [11] e o método proposto neste trabalho são:

1. Tanto os frameworks MARLIN quanto UDD utilizam classificadores SVM para classificação. Em nossos testes, foi verificado que a utilização de classificadores mais simples pode ser tão eficiente quanto estes para determinados domínios. Dado em que nos problemas que objetivamos tratar

a quantidade de dados é pequena, a utilização de um classificador deste tipo pode vir a introduzir ruídos na classificação.

2. O framework MARLIN foi concebido para cenários de integração de dados off-line, enquanto que os frameworks UDD e o framework proposto neste trabalho foram desenvolvidos para operar em um cenário on-line. Por conta disso, as entidades utilizadas nestes últimos são sempre objetos com múltiplos campos do tipo cadeia de caractere, ao contrário do primeiro que pode ter campos com outros tipos de estruturas como números ou referências a outros objetos.

6.3 Comparação de resultados

É possível obter indícios sobre a qualidade da solução proposta comparando os resultados com os obtidos pelo UDD, um framework que propõe resolver o problema de detecção de duplicatas online [11]. Os experimentos foram realizados com diversos domínios, no entanto, Su [11] compara seu framework com outros utilizando o domínio dos livros. O UDD teve a melhor performance, com 92,4% de precisão, 91,5% de *recall* e 91,9% de *F-Measure*. É claro que muito pouco pode ser dito ao compararmos resultados de domínios diferentes, mas a intuição diz que o domínio de livros deve ser menos problemático que o de vinhos, já que o ISBN normalmente provê uma identificação única e não há tanta variação nos títulos e nomes de autores como há nas etiquetas dos vinhos.

Na proposta de dissertação de doutorado de Bilenko [15], o autor conduziu um experimento com o domínio dos vinhos, de modo a testar algumas técnicas de clusterização do MARLIN. Os seus resultados mostram que nos melhores casos é possível alcançar valores próximos de 90% de *F-Measure* usando seu framework. Não foram dados detalhes sobre precisão e *recall*. Na comparação com essas duas abordagens, é possível constatar que os resultados obtidos estão na mesma faixa, o que nos faz acreditar que a solução proposta é aceitável. A tabela 11 resume os valores de precisão, *recall* e *F-Measure* obtidas nos frameworks.

Framework	Precisão	Recall	F-Measure
Abordagem proposta	88.89%	90.0%	89.44%
UDD ¹² (Su et al, 2010)	92.40%	91.50%	91.90%
MARLIN [15]	N/D	N/D	~90%

Tabela 11 - Comparação de performance

¹² Números referidos ao estudo no domínio dos livros.

7 Conclusão

Aplicações web que obtém dados de diferentes (Mashups) estão ganhando importância na internet. Um importante requisito dessas aplicações é a habilidade de assegurar a qualidade dos dados provendo uma solução viável para o problema de identificação de duplicatas.

A maioria das soluções existentes utilizam algoritmos de aprendizado de máquina, que são dependentes de uma base de treinamento pré-existente.s. Soluções deste tipo não são adequadas quando utilizamos fontes de dados da Deep Web. Nesta dissertação propomos uma solução para o enriquecimento de bases de dados através de consultas em fontes de dados na Deep Web. Esta solução pode ser decomposta em duas contribuições principais. A primeira é uma estratégia para a busca (rastreamento) de informações na Deep Web orientado a duplicatas, cujo objetivo é obter resultados precisos de consultas construídas a partir de um conjunto conhecido de objetos e seus atributos. A segunda contribuição, é uma estratégia para a detecção de duplicatas em resultados de consultas realizadas sobre fontes de dados da Deep Web.

Nossa abordagem não requer uma base de treinamento previamente definida, e utiliza um classificador baseado no Vector Space Model (VSM) em combinação com funções de cálculo de similaridade para prover uma solução viável. Em contrapartida, nossa solução requer um esforço adicional para escrever parsers de dados, filtros e wrappers para normalizar dados minerados da web para um formato consistente com nosso esquema global.

De modo a validar a solução proposta, construímos um framework que implementa as estratégias de busca e incorporação de dados. Ilustramos a utilização do framework através da instanciação do mesmo em uma aplicação de comércio eletrônico voltada para o domínio de vinhos. Os detalhes da implementação e as lições aprendidas no processo foram descritas cuidadosamente, para que possam ser úteis aqueles que pretendem replicar os experimentos aqui realizados. Os resultados, quando comparados com outras técnicas, são promissores.

7.1 Trabalhos Futuros

Dados os resultados obtidos com a estratégia proposta, é possível, como trabalho futuro, instanciar e investigar o framework utilizando outros domínios e investigar o comportamento do framework com um conjunto maior de teste. Espelhando-se em trabalhos de referência como o de Su [11], pode-se experimentar incluir um mecanismo de auto-calibração do classificador.

Alguns experimentos superficiais foram realizados utilizando fontes de dados do tipo OWL com o domínio de livros. Os primeiros resultados demonstraram que utilizar a mesma solução que foi dada ao domínio dos vinhos para este domínio não foi adequado. É necessária uma investigação acerca deste domínio para a criação de uma nova instância do framework.

8 Bibliografia

1. Baeza-Yates, R; Ribeiro-Neto, B; Modern Information Retrieval. Addison-Wesley-Longman, 1999;
2. Bergman, M; The Deep Web: Surfacing Hidden Value and Semantization; Research Challenges in Information Science, Pags 225-236, 22-24 April 2009.
3. Bilenko, M; Mooney, R; Adaptative Duplicate Detection Using Learnable String Similarity Measures; Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003), Washington DC, pp 39-48, August, 2003.
4. Bleiholder, J; Naumann, F; Data Fusion; ACM Computing Surveys, Vol. 41, No. 1, Article 1, Publication date: December 2008.
5. Davis, C; Salles, E; 2007; Aproximate String Matching for Geographical Names and Personal Names; IX Brazilian Symposium on GeoInformatics, Campos do Jordão, Brazil, November 25-28, 2007, INPE p. 49-60.
6. Dice, L; Measures of the Amount of Ecologic Association Between Species; Ecological Society of America, Vol. 26, No. 3, Jul. 1945, pp. 297-302.
7. He, B; Patel, M; Zhang, Z; Chang, K; 2007; Accessing The Deep Web: A Survey; Communications of the ACM, Vol. 50, Issue 5, Pags 94-101, May 2007
8. Köpke, H; Rahm, E; 2009; Frameworks for entity matching: A comparison; Data Knol, Eng 2009, doi:10.1016/j.datak.2009.10.003
9. Newcombe, H; Kennedy, J; Axford, S; James, A; Automatic Linkage of Vital Record; Science, Vol. 130 (16 October 1959), pp. 954-959.
10. Ntoulas, A; Zerefos, P; Cho, J; 2005; Downloading Hidden Web Content; UCLA Computer Science, 2005.
11. Su, W; Wang, J; Lochovsky, F; 2010; Record Matching over Query Results from Multiple Web Databases; IEEE Transactions on Knowledge and Data Engineering, Vol 22, No.4, April 2010.
12. Wright, A; 2004; In Search of the Deep Web; Salon.com; http://www.salon.com/tech/feature/2004/03/09/deep_web/
13. Dice, L; 1945; *Measures of the Amount of Ecologic Association Between Species*; Ecology, Vol. 26, No. 3(Jul., 1945), pp. 297-302.
14. Manning, C; Raghavam, P; Schütze, H; *Introduction to Information Retrieval*; Cambridge University Press, 2008, chapter 14.
15. Bilenko, M; *Learnable Similarity Functions and Their Applications to Record Linkage and Clustering*; Ph.D. proposal, Department of Computer Sciences, University os Texas at Austin, October 2003, 47 pages.

16. Brauner, D; Gazola, A; Casanova, M; Breitman, K; 2007; *An Instance-based Approach for Matching Export Schemas of Geographical Database Web Services*; IX Brazilian Symposium on GeoInformatics (GeoInfo 2007), Brazil.
17. Brauner, D; Gazola, A; Casanova, M; Breitman, K; 2008; *Adaptative Matching of Database Web Services Export Schemas*; 10th International Conference on Enterprise Information Systems (ICEIS 2008), 12-16, June, 2008, Barcelona, Spain
18. Carvalho, M; Laender, A; 2009; *Abordagens Evolucionárias para Problemas Relacionados à Integração de Dados*; Belo Horizonte: UFMG , 2009 (Tese de Doutorado)
19. Cohen W; Ravikumar, P; Fienberg, S; 2003; A Comparison of String Distance Metrics for Name-Matching Tasks; Proceedings of IJCAI-03 Workshop on Information Integration, pp 73-78.
20. Leme, L; Casanova, M; 2009; *Conceptual Schema Matching Based on Similarity Heuristics*; Rio de Janeiro: PUC-Rio, 2009 (Tese de Doutorado)
21. Laender, A; Silva, A; Ribeiro-Neto, B; Teixeira, J; 2009; *A Brief Survey of Web Data Extraction Tools*; Belo Horizonte: UFMG, 2009.
22. Peisu, X; Ke, T; Qinzhen, H; 2008; *A Framework of Deep Web Crawler*, Proceedings of the 27th Chinese Control Conference, July 16-18, 2008, Kunming, Yunnan, China.
23. Pereira, D; Ribeiro-Neto, B; 2009; *A Web-Based Framework for Entity Resolution And Creation of Authority Files*; Belo Horizonte: UFMG , 2009 (Tese de Doutorado)
24. Wang, J; Wen, J; Lochovsky, F; Ma, W; 2004; *Instance-based Schema Matching for Web Databases by Domain-specific Query Probing*; The 30th Very Large Data Base Conference (VLDB'04).
25. Levenshtein, V; 1966; *Binary Codes Capable of Correcting Deletions, Insertions and Reversals*. Soviet Physics Doklady 10, 8, 707-710.
26. Russel, R.; 1918; United States Patent Office, N.1.261.167
27. Jaro, M; 1989; *Advances in Record Linkage Methodology as Applied to the 1985 Census of Tampa Florida*; Journal of The American Statistical Society, pp. 84
28. Winkler, W; 1999; *The State of Record Linkage and Current Research Problems*; Statistics of Income Division, Internal Revenue Service Publication R99/04.
29. Spärk Jones, K; 1972; *A Statistical Interpretation of Term Specificity and Its Applications in Retrieval*; Journal of Documentation 28, V1, pp. 11-21.

30. Krause, E; 1987; *Taxicab Geometry*; Dover. ISBN 0-486-25202-7
31. Jaccard, P; 1901; *Étude Comparative de La Distribution Florale Dans Une Portion des Alpes ET des Jura*; Bulletin de La Société Vaudoise des Sciences Naturelles 37, PP 547-579.
32. Monge, A; Elkan, C; 1996; *The Field Matching Problem: Algorithm and Applications*; In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining.
33. Hernández, M; Stolfo, S; 1995; *The Merge/Purge Problem for Large Databases*; In Conference on the Management of Data (SIGMOD), pp 127-138, San Jose, CA, 1995.
34. Dong, X; Halevy, A; Madhavan, J; 2005; *Reference Conciliation in Complex Information Spaces*; In Conference on the Management of Data (SIGMOD), pp 85-96, Baltimore, MD, 2005.
35. Bhattachary, I; Geetor, L; Licamele, L; 2006; *Query-time Entity Resolution (poster)*; In Conference on Knowledge Discovery and Data Mining (KDD), pp 529-534, Philadelphia PA, 2006.
36. Zhang, H; 2004; *The Optimality of Naïve bayes*; In 17th International Florida Artificial Intelligence Research Society Conference, 2006.
37. Bayes, T; Price, R; 1763; *An Essay Towards Solving a Problem in The Doctrine of Chance*; Philosophical Transaction of The Royal Society of London, 53, pp 370-418.
38. Salton, G; Wong, A; Yang, C; 1975; *A Vector Space Model for Automatic Indexing*; Communications of the ACM, vol. 18, n. 11, pp 613-620.
39. Cover, T; Hart, P; 1967; *Nearest Neighbor Pattern Classification*; IEEE Transactions on Information Theory, vol. 13, Issue 1, pp 21-27, Jan 1967.
40. Raghavan, S; Garcia-Molina, H; 2001; *Crawling the Hidden Web*; Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), pages 129-138, 2001.
41. Bergholz, A; Chidlovskii, B; 2003; *Crawling the Hidden Web*; Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE 2003).
42. Vapnik, V; 1998; *Statistical Learning Theory*; Wiley, 1998.

43. Oguri, P; 2006 ; *Machine Learning For Sentiment Classification*; Dissertação de Mestrado, Departamento de Informática, Puc-Rio, 2006.
44. Merril, D; 2009; Mashups: The new breed of Web app;
<http://www.ibm.com/developerworks/xml/library/x-mashups.html>; acessado em 24/07/2009.