2 Conceitos Básicos

Neste capítulo apresentamos um breve sumário da pesquisa realizada nas áreas de recuperação de dados na web, construção de rastreadores e resolução 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://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=99170
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 reúso.

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 [27].

Essas aplicações têm em comum a arquitetura orientada a serviços: baseada em baixo acoplamento e orientada a meta dados. Por isso, esse tipo de aplicação deve ser desenvolvida 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 [23]. 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.

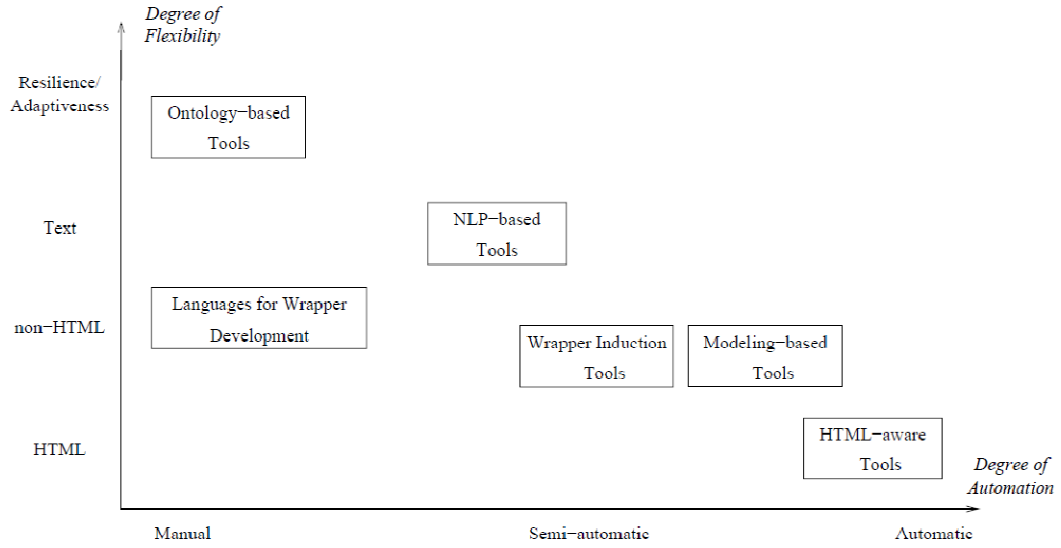

Figura 2 - Classificação de Extratores Web [23]

2.2 Construção de rastreadores (*crawlers*)

O desenvolvimento de rastreadores (*crawlers*) específicos para a a exploração da Deep Web foi abordado por Peisu [32]. 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.

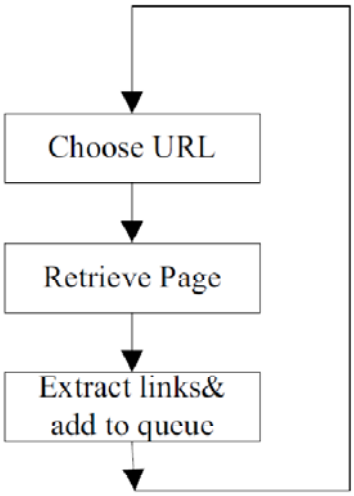

Figura 3 - Rastreador (*crawler*) tradicional [32]

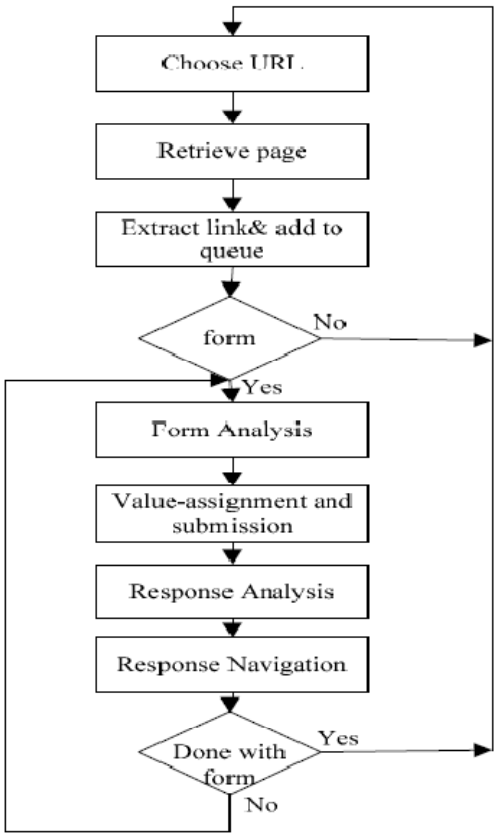

Figura 4 - Rastreador (*crawler*) de Deep Web [32]

Rastreadores de Deep Web têm a sua eficiência intimamente ligada à 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 [30]. 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 [30] faz uma comparação entre as políticas randômicas, baseadas na freqüência 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 Alinhamento 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 [39] propôs 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 [9, 10], 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 Resolução de entidades – Entity Matching

As soluções mais comuns para resolução do problema de resolução 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 [6, 21, 33, 36]. 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 à 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” [40], “Merge/Purge” [18], “Reference Conciliation” [16], “Entity Resolution” [5] ou “Duplicate Detection” [36].

Davis propôs um algoritmo para comparação de cadeias de caracteres contendo múltiplas palavras [14]. 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 [11] 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 [33] 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 [21] 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. Eficácia: 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 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 [6]. Segundo o autor, 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, em seguida, 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.

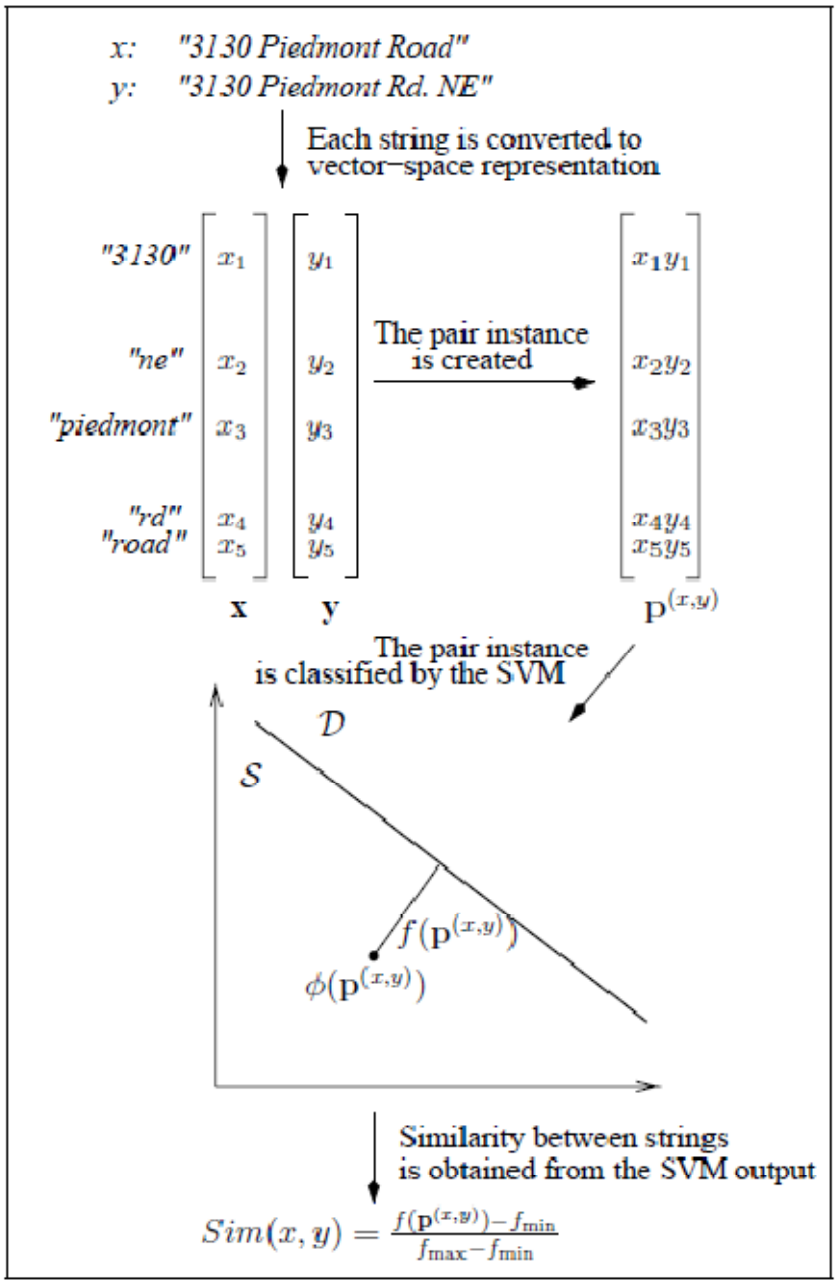

Figura 5 - Visão geral do algoritmo de similaridade de MARLIN [6]

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 [36], 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 [36] 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 serem 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 à 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 cadeias comparadas. É possível também medir a similaridade entre cadeias 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 cadeias 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 cálculo 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) [38] 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 [20] é baseada no número e ordem de caracteres comuns entre duas cadeias de caracteres. Dadas duas cadeias de caracteres si e sj, é dito que um caractere na cadeia si é igual a um caractere na cadeia sj 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 [40] 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) [34]. 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 freqüência do termo no vetor, sendo estes pesos calculados de diversas formas.

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

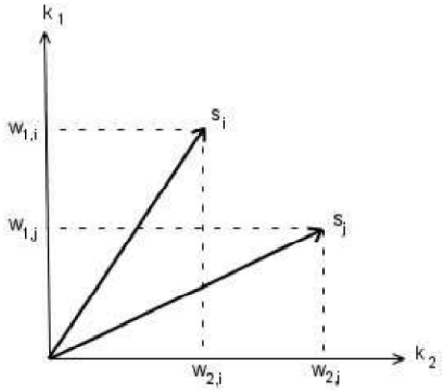

Figura 6 - Representação gráfica de 2 documentos no VSM

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 t*okens* que mede a similaridade entre duas cadeias 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 freqüência 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:

$$w_{t,d} = tf_{t,d} \times \log \frac{|D|}{|\{d \in D \mid t \in d\}|}$$

Onde $tf_{t,d}$ é a freqüência do termo t no documento d e $|D|$ é o conjunto total de documentos. $|\{d \in D \mid t \in d\}|$ é o conjunto de documentos contendo o termo t.

Este cálculo de peso também é chamado de algoritmo TF-IDF [35] e é muito em diversos algoritmos da área de recuperação de informações. Na teoria, quanto maior a freqüência, mais importante é a palavra em um documento.

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 [15]. 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 [22].

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

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 inicio do século passado. O coeficiente de Jaccard [19] mede a similaridade entre dois conjuntos dividindo a interseção dos conjuntos pela sua união, segundo a fórmula abaixo.

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

2.5.3 *Funções híbridas*

Existem ainda funções híbridas que combinam técnicas de cálculo de similaridade entre cadeias de caracteres baseadas em caracteres e tokens.

Alvaro E. Monge e Charles P. Elkan [28] propuseram uma estratégia de *matching* recursivo. O algoritmo proposto pelos autores calcula a similaridade entre cadeias de caracteres utilizando as subcadeias 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:

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

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 [12] utilizando bases de dados de nomes próprios com as funções de similaridade: Levenshtein, Jaro, Jaro-Winkler, Smith-Waterman, Jaccard, Monge-Elkan e TF-IDF, SFS e Soft-TFIDF, chegou-se à conclusão que a função de Monge-Elkan apresentou os melhores resultados no que diz respeito à qualidade da identificação de duplicatas na categoria “algoritmos baseados em distância de edição”, seguido por Jaro-Winkler. Dentre as 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 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* [33].

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 (KNN) [13]. 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.

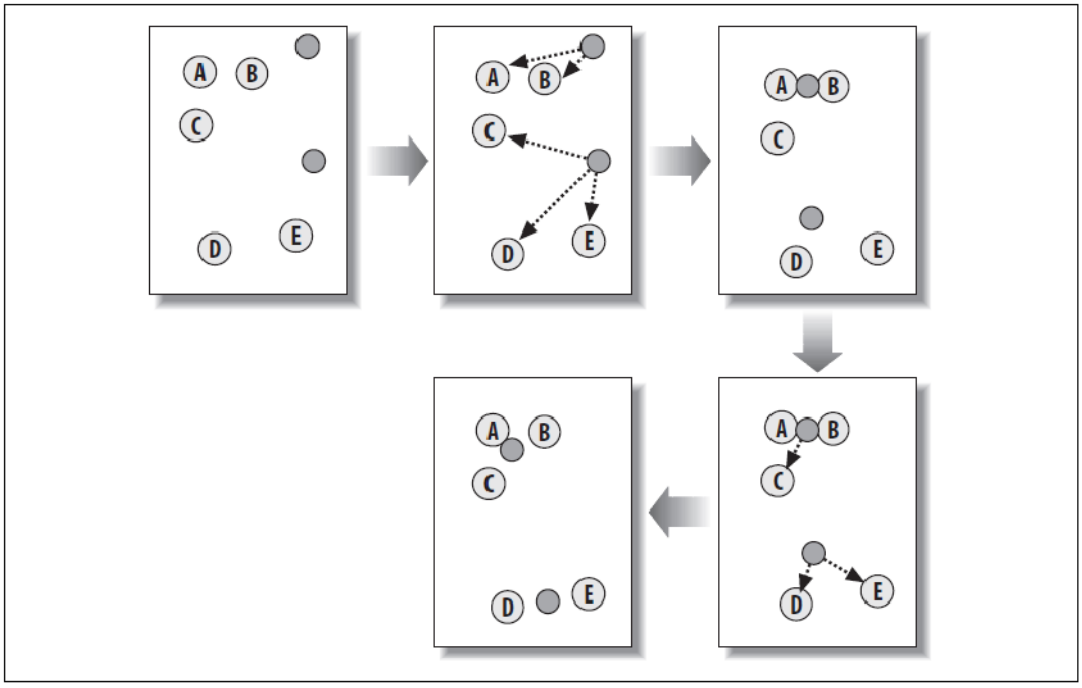

Figura 7 - Exemplo de execução do KNN

Classificadores Bayesianos são baseados na Teoria de Bayes [2]. 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 o 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 [43].

É possível criar classificadores que utilizam a técnica de Vector Space Model de modo análogo aos classificadores Bayesianos [26]. Em vez de utilizar as probabilidades para categorizar itens, são utilizados os pesos das freqüências. Esse tipo de classificador apresenta algumas vantagens em relação aos classificado-

res 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 caracteres muito grandes podem apresentar pouca similaridade, já que as freqüências 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.

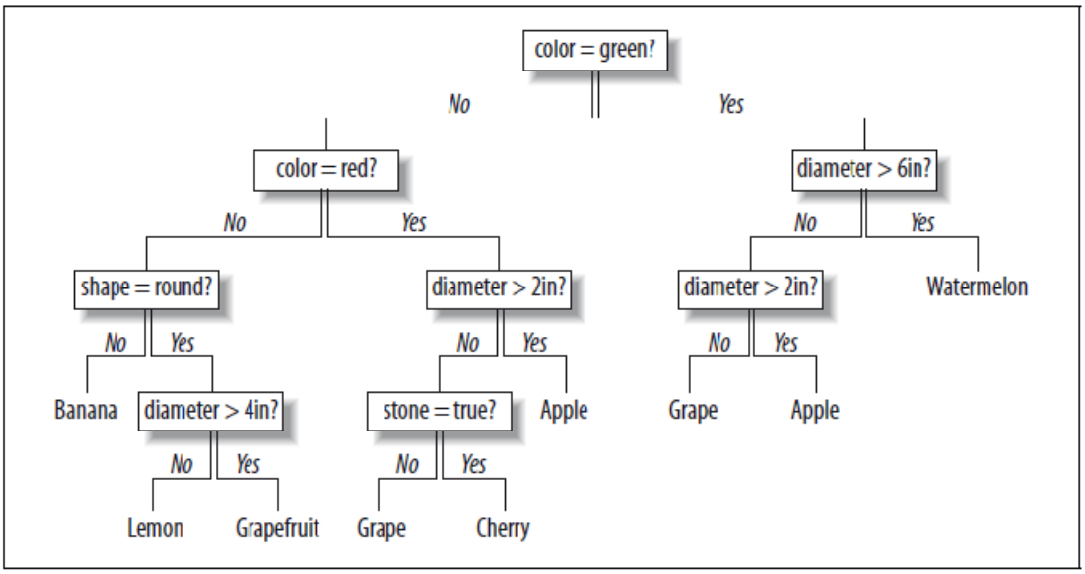

Figura 8 - Exemplo de árvore de decisão

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).

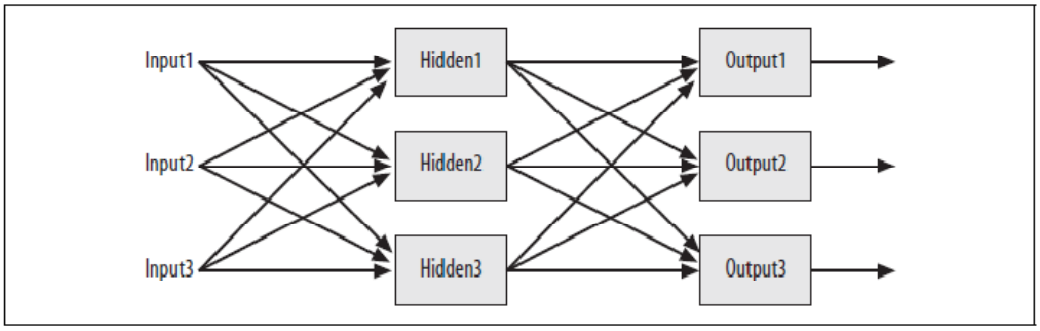

Figura 9 – Exemplo de estrutura básica de nós de uma rede neural

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 [31].

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 conceitos fundamentais que apoiaram o desenvolvimento deste trabalho. Foram apresentadas algumas técnicas de recuperação de dados na web, construção de rastreadores (*crawlers*), alinhamento de esquemas (*schema matching*) e resolução de entidades (*entity matching*).

Ainda foram apresentados conceitos básicos e algumas técnicas que permitem o cálculo de similaridade entre cadeias de caracteres. Discutimos funções 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 *Support Vector Machines*.

Para este trabalho utilizamos idéias presentes em cada um dos trabalhos apresentados. Está presente no framework um rastreador de links (*crawler*) para Deep Web baseado em URL's [30], que oferece eliminação de duplicatas online como em [36] apoiado em um classificador que utiliza o *Vector Space Model* [26, 34] e diversas funções de cálculo de similaridade entre cadeias de caracteres como [1, 15, 22].