UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JOÃO ADOLFO FROEDE LUTZ

Descoberta de ruído em páginas da *web* oculta através de uma abordagem de aprendizagem supervisionada

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Carlos Alberto Heuser
Orientador

Porto Alegre, novembro de 2013

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Lutz, João Adolfo Froede

Descoberta de ruído em páginas da *web* oculta através de uma abordagem de aprendizagem supervisionada / João Adolfo Froede Lutz. – Porto Alegre: PPGC da UFRGS, 2013.

60 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2013. Orientador: Carlos Alberto Heuser.

1. Web oculta. 2. Recuperação de Informações. 3. Extração de dados *web*. 4. Eliminação de ruído *web*. I. Heuser, Carlos Alberto. II. Título.

“Aprender é a única coisa de que a mente nunca se cansa,
nunca tem medo e nunca se arrepende.”
— LEONARDO DA VINCI

AGRADECIMENTOS

Gostaria de agradecer a todos que estiveram envolvidos de alguma maneira durante o processo de desenvolvimento deste trabalho. Vocês foram muito importantes nesta etapa da minha vida.

Em especial, agradeço ao meu professor orientador Carlos Alberto Heuser, que esteve comigo desde o final da graduação até o final do mestrado. Obrigado por me dar a confiança necessária, e sempre me direcionar para o caminho correto. Sem seu apoio e sua dedicação eu não teria chegado até aqui.

Agradeço à Universidade Federal do Rio Grande do Sul, que mais uma vez me acolheu como aluno e me possibilitou o acesso a um ensino de grande qualidade. Também quero agradecer ao corpo docente do Instituto de Informática, que ajudou a moldar meu conhecimento acadêmico. Em especial aos professores do grupo de Modelagem Conceitual e Bancos de Dados, representados em nomes como Renata Galante, Viviane Moreira, José Palazzo e Carlos Heuser.

Obrigado a todos os meus familiares, avós, tios, primos e sogros, pelo constante apoio e incentivo. Agradeço também aos meus amigos e colegas de trabalho pela constante camaradagem e suporte necessário.

Quero agradecer especialmente à minha esposa Luiza. Sem ela ao meu lado este trabalho não teria sido possível. Teu amor, carinho, amizade e companheirismo foram fundamentais durante todo o trabalho e nossa vida juntos.

Por fim, agradeço aos meus pais, Lenir e Graça, que nunca pouparam esforços para que eu realizasse todos os meus sonhos. Esta conquista também é um pouquinho de cada um de vocês.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AICF	*Average Inverse Document Frequency*
BCS	*Block Class Spread*
BL	*Baseline*
CC	Conteúdo como conteúdo
CR	Conteúdo como ruído
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
DT	Densidade de Texto
DTC	Densidade de Texto Composta
F1-BL	F1 do *baseline*
F1-BL	F1 da implementação do *baseline*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IBDF	*Inverse Block Document Frequency*
ICF	*Inverse Class Frequency*
IDF	*Inverse Document Frequency*
LCS	*Longest Common Substring*
P-BL	Precisão do *baseline*
P-IMPL-BL	Precisão da implementação do *baseline*
R-BL	Revocação do *baseline*
R-IMPL-BL	Revocação da implementação do *baseline*
RC	Ruído como conteúdo
RR	Ruído como ruído
SST	*Site StyleTree*
ST	*StyleTree*
SVM	*Support Vector Machines*

TF	*Term Frequency*
TF-IDF	*Term Frequency Inverse Document Frequency*
URL	*Uniform Resource Locator*
TP	Técnica Proposta
VIPS	*Vision Based Page Segmentation*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Um dos problemas da extração de dados na *web* é a remoção de ruído existente nas páginas. Esta tarefa busca identificar todos os elementos não informativos em meio ao conteúdo, como por exemplo cabeçalhos, menus ou propagandas. A presença de ruído pode prejudicar seriamente o desempenho de motores de busca e tarefas de mineração de dados na *web*. Este trabalho aborda o problema da descoberta de ruído em páginas da *web* oculta, a parte da *web* que é acessível apenas através do preenchimento de formulários. No processamento da *web* oculta, a extração de dados geralmente é precedida por uma etapa de inserção de dados, na qual os formulários que dão acesso às páginas ocultas são automaticamente ou semi-automaticamente preenchidos. Durante esta fase, são coletados dados do domínio em questão, como os rótulos e valores dos campos. A proposta deste trabalho é agregar este tipo de dados com informações sintáticas dos elementos que compõem a página. É mostrado empiricamente que esta combinação atinge resultados melhores que uma abordagem baseada apenas em informações sintáticas.

Palavras-chave: Web oculta, Recuperação de Informações, Extração de dados *web*, Eliminação de ruído *web*.

A supervised learning approach for noise discovery in web pages found in the hidden web

ABSTRACT

One of the problems of data extraction from web pages is the identification of noise in pages. This task aims at identifying non-informative elements in pages, such as headers, menus, or advertisement. The presence of noise may hinder the performance of search engines and web mining tasks. In this paper we tackle the problem of discovering noise in web pages found in the hidden web, i.e., that part of the web that is only accessible by filling web forms. In hidden web processing, data extraction is usually preceeded by a form filling step, in which the query forms that give access to the hidden web pages are automatically or semi-automatically filled. During form filling relevant data about the queried domain are collected, as field names and field values. Our proposal combines this type of data with syntactic information about the nodes that compose the page. We show empirically that this combination achieves better results than an approach that is based solely on syntactic information.

Keywords: Hidden web, Information retrieval, Web data extraction, Web noise removal.

1 INTRODUÇÃO

Desde o surgimento da *web*, um grande desafio se faz constante nesta área de pesquisa: a extração de seus dados. A cada dia aumenta a quantidade de dados relevantes existentes na Internet, e métodos automatizados de coleta dessas informações necessitam ser desenvolvidos. Inúmeros trabalhos tratam da extração de dados não estruturados, com diferentes focos, objetivos e metodologias (LAENDER et al., 2002) (CHANG et al., 2006).

Entretanto, um grande problema para a extração dos dados é a separação das informações relevantes e do ruído existente em meio a esses dados. Neste trabalho, abordamos a descoberta deste ruído em páginas *web*, focando esta tarefa de descoberta ao ruído existente em páginas da *web* oculta.

1.1 *Web* Oculta

O crescimento exponencial da *web* gerou uma divisão em diferentes segmentos dentro desta. Denomina-se "*web* visível" a parte da *web* cujo conteúdo é possível acessar diretamente, seja através de motores de busca ou *URLs (Uniform Resource Locators)* estáticas. Porém existe outra porção da *web* chamada "*web* oculta", que é composta de páginas geralmente acessíveis apenas através do preenchimento de formulários. Após a submissão deste, uma página resultante é renderizada dinamicamente com informações provenientes de um banco de dados. Um exemplo de formulário e a página resultante da submissão do mesmo pode ser visto na Figura 1.1.

Bergman (2001) estima que a *web* oculta é 550 vezes maior do que a *web* visível, e que a qualidade do conteúdo existente neste segmento é superior à qualidade dos dados da *web* indexável. Segundo He et al. (2007), além das páginas da *web* oculta possuírem um caráter mais estruturado, a rapidez da expansão da parte oculta da *web* é maior que da web visível. Considerando a dificuldade de precisar valores de tamanho, o que pode-se afirmar com certeza é que há uma grande quantidade de dados acessíveis apenas através de formulários na *web*, e que as informações contidas nestas páginas podem ser de alta relevância.

A exploração da parte oculta da *web* é dividida em fases distintas (RAGHAVAN; GARCIA-MOLINA, 2000), (JIANG et al., 2009), como mostrado na Figura 1.2. Primeiramente, é necessário descobrir os pontos de entrada para este conteúdo, ou seja, encontrar formulários que, quando preenchidos corretamente, direcionam o usuário para páginas com conteúdo normalmente não acessível via *URLs* ou motores de busca. A dificuldade desta tarefa reside não apenas em encontrar tais formulários, mas em selecionar aqueles em que após seu preenchimento, levarão a páginas com dados relevantes. A segunda fase envolve o uso de técnicas para preencher tais formulários, visto que diferentes

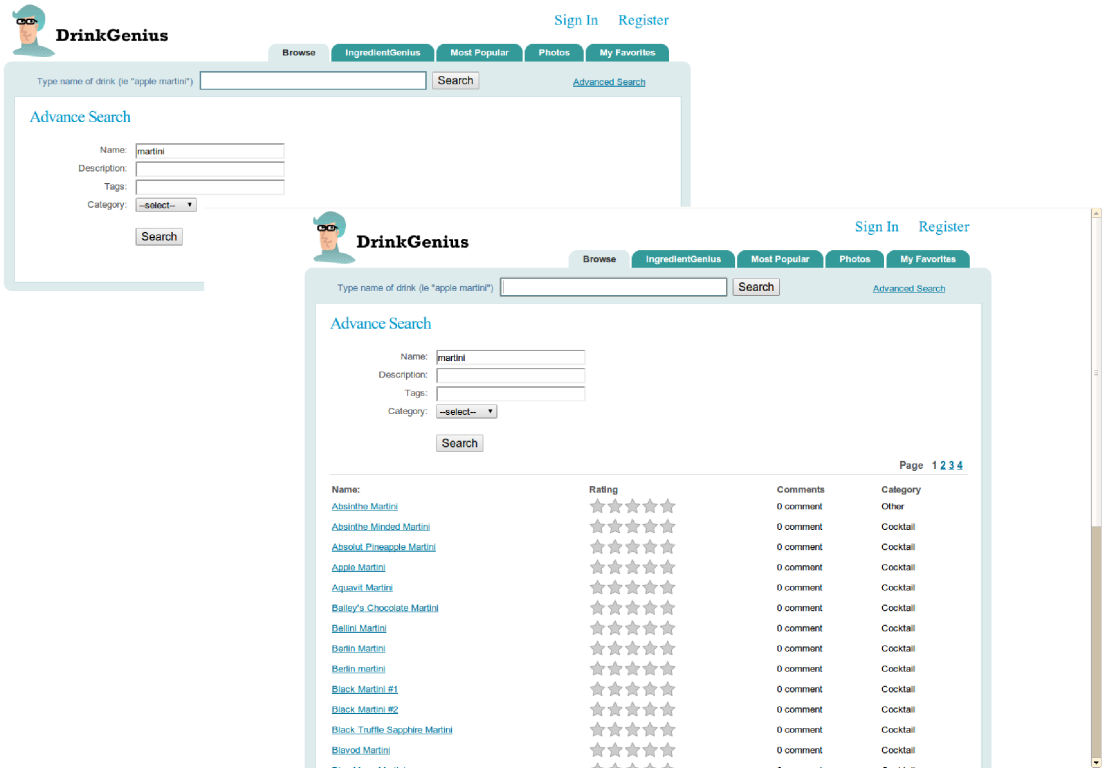

Figura 1.1: Exemplo de *web* oculta: um formulário e o resultado de sua submissão.

preenchimentos levam a diferentes resultados. O desafio desta fase é preencher os formulários de forma eficiente, minimizando o número de consultas e custo de preenchimento e maximizando o número de resultados úteis. Após a submissão do formulário, a última fase compreende a extração dos dados da página resultante.

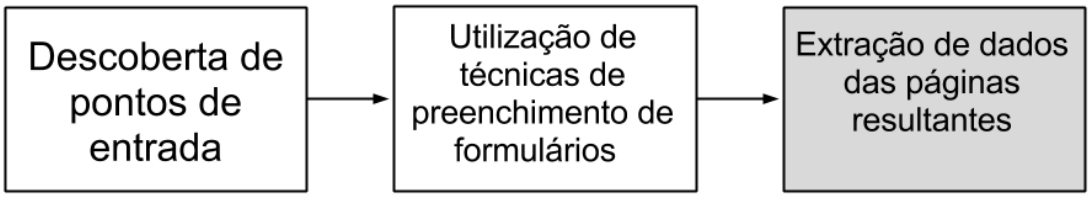

Figura 1.2: Etapas da exploração da *web* oculta.

1.2 Ruído em páginas web

Este trabalho está inserido no contexto da terceira fase, onde os resultados de uma busca devem ser extraídos. Busca-se neste trabalho identificar o ruído nas páginas *web* resultantes.

Ruído em páginas *web* é todo conteúdo que não é informativo nem de interesse do usuário final (YI; LIU; LI, 2003). Entre estes elementos não informativos, podem ser

citados banners, propagandas, painéis de navegação, informações de contato, e até mesmo políticas de privacidade. Gibson, Punera e Tomkins (2005) estimam que até 50% de todo conteúdo da *web* é composto por algum tipo de ruído. As Figuras 1.3 e 1.4 mostram exemplos de ruído em páginas *web*.

Figura 1.3: Exemplo de ruído em uma página *web*.

A importância da remoção de ruído está no fato de que estas porções de conteúdo prejudicam seriamente a performance de sistemas de recuperação de informações na *web*. Os maiores prejudicados são os motores de busca, que acabam não só perdendo precisão devido ao excessivo conteúdo irrelevante, mas porque necessitam armazenar um índice de tamanho muito maior. Tarefas de mineração na *web*, como classificação e clusterização, por exemplo, também são seriamente prejudicadas, já que seu resultado é afetado negativamente por esse tipo de conteúdo.

Além disso, a eliminação do ruído também pode ser usada tanto para a construção

Figura 1.4: Cabeçalho de página *web* composto em sua quase totalidade por ruído.

de corpora de texto, quanto para a adaptação de páginas *web* para dispositivos de tela reduzida, por exemplo.

1.3 A remoção de ruído na literatura

Diversos métodos foram desenvolvidos para alcançar este objetivo, e é possível classificá-los em grupos relativos à maneira como o ruído é identificado e removido.

Cai et al. (2003), Fernandes et al. (2007), Li e Ezeife (2006), Song et al. (2004), Kovacevic et al. (2002) e Burget e Rudolfova (2009) utilizam algoritmos baseados em visão, já que todos estes trabalhos classificam o conteúdo da página *web* em ruído ou não baseado nos seus atributos visuais.

O segundo grupo, composto pelos trabalhos de Bar-Yossef e Rajagopalan (2002), Debnath, Mitra e Giles (2005), Lin e Ho (2002), Chen, Ye e Li (2006) e Wang et al. (2008), tenta identificar o ruído procurando por *templates* nas páginas *web*. Este procedimento é feito procurando-se por repetição de conteúdo nestas páginas através de heurísticas bem definidas.

O outro grupo, com os trabalhos de Yi, Liu e Li (2003) e Vieira et al. (2006), tenta identificar o ruído baseado na similaridade estrutural da árvore *DOM (Document Object Model)* das páginas *web*.

Por fim, existe um grupo que busca identificar o ruído baseado em propriedades textuais, como Kohlschütter, Fankhauser e Nejdl (2010), Weninger, Hsu e Han (2010) e Sun, Song e Liao (2011). Este último será usado neste trabalho como *baseline*.

O grande problema da maior parte destes trabalhos é que praticamente todos assumem que as páginas *web* possuem uma estrutura específica, ou procuram por elementos *HTML* especiais. Isso torna a eliminação de ruído destes trabalhos muito específica para determinado conjunto de dados.

1.4 Objetivos deste trabalho

O objetivo principal deste trabalho, dada sua inserção no contexto da *web* oculta, é melhorar os resultados da identificação de ruído em páginas resultantes da submissão de formulários. Para isto, o trabalho baseia-se na hipótese de que é possível atingir o objetivo através da utilização de dados de fases anteriores da exploração da *web* oculta. Mais especificamente, isto é feito coletando-se rótulos e valores de preenchimentos de campos de formulários, e procurando por correspondências destes termos nas páginas resultantes das submissões. A presença de um maior número de correspondências é um indicativo de que determinado trecho é conteúdo, e não ruído.

Neste trabalho, utiliza-se como *baseline* uma técnica que utiliza a densidade de texto e *links* em uma página *web* para encontrar o ruído (SUN; SONG; LIAO, 2011). O trabalho referido assume que elementos que representam conteúdo informativo possuem uma densidade de texto puro mais alta que o ruído. Elementos ruidosos, por sua vez, possuem um número alto de elementos de formatação, além de muito mais *links* em seu conteúdo.

Após a implementação da técnica descrita por Sun, Song e Liao (2011), foi possível observar que pode-se obter resultados melhores ao incorporarmos dados provenientes da submissão do formulário que origina a página resultante. Este objetivo é atingido utilizando-se um método de aprendizagem supervisionado para classificar os elementos de texto em ruído ou não.

1.5 Organização dos capítulos

Este trabalho está organizado da seguinte maneira: O Capítulo 2 descreve trabalhos da literatura relacionada, com o objetivo de remoção de ruído. No fim deste capítulo, é mostrada uma técnica para remoção de ruído que é utilizada como *baseline*, além de uma comparação entre as técnicas citadas; No Capítulo 3, é explicado como é possível melhorar a remoção de ruído e obter melhores resultados utilizando um método de aprendizagem supervisionado; No Capítulo 4, são descritos e comparados os experimentos realizados com ambas as técnicas, utilizando-se diferentes conjuntos de dados; No Capítulo 5, é feita uma conclusão do trabalho e são mostradas possibilidades de trabalhos futuros.

2 TRABALHOS RELACIONADOS

Diferentes trabalhos foram divulgados nos últimos anos na tentativa de eliminar ruído em páginas *web*. A grande diversidade de metodologias utilizadas faz com que não exista um padrão seguido por todas. Entretanto, os métodos podem ser categorizados em 5 grupos de acordo com o tipo de metodologia utilizada:

- Técnicas que baseiam a eliminação na identificação de blocos
- Técnicas que buscam segmentar a página visualmente e classificar os segmentos
- Técnicas que fazem uso da similaridade estrutural para encontrar ruído
- Técnicas híbridas
- Técnicas que utilizam propriedades textuais como principal atributo

Nas subseções seguintes, são descritas as metodologias utilizadas em cada trabalho, buscando mostrar a contribuição de cada trabalho buscando a eliminação de ruído. Será dado uma maior ênfase na descrição das técnicas do que nos resultados obtidos, mediante a não verificação dos mesmos neste estudo, com exceção do *baseline* utilizado.

2.1 Técnicas para eliminação do ruído

2.1.1 Técnicas baseadas em identificação de blocos

Algumas técnicas baseiam a detecção de ruído na busca por blocos dentro de uma página *web*, para depois classificar os mesmos quanto a sua importância ou utilidade. É o caso de Bar-Yossef e Rajagopalan (2002), onde é feita uma definição formal do problema de detecção de *templates*, e este é tratado como uma instância do problema *frequent item set*, normalmente utilizado em tarefas de mineração de dados. Neste contexto, páginas *web* são consideradas *items*, e o conjunto de páginas *web* citadas em uma página particular representariam um *item set*, portanto, um *frequent item set* corresponderia a um *template*.

O estudo descreve alguns princípios que norteiam o projeto de uma página *web*, permitindo que seja feita uma inferência da relevância do conteúdo baseado em algumas propriedades. Os *templates* violariam esses princípios, prejudicando o desempenho de sistemas de recuperação, como a geração do *ranking* de um sistema de busca, por exemplo.

Bar-Yossef e Rajagopalan (2002) definem um *template* como um trecho de código *HTML* pré-pronto, organizado por uma autoridade central, na qual trechos de conteúdo

são embutidos para padronização de estilo. Como estes trechos contêm muitos *links*, que normalmente são utilizados para auxílio à navegação, estes *links* provavelmente não pertencem ao conteúdo principal da página, podendo ser considerados, portanto, como ruído. Além disso, o trabalho também faz a definição de um "*pagelet*", que é uma região lógica da página *web*, contendo um tópico ou funcionalidade bem definida. Uma página pode ser decomposta em um ou mais *pagelets*, e os autores afirmam que estes trechos constituem uma unidade melhor do que páginas inteiras para recuperação de informações, dada sua maior coesão.

A partir dessas definições, surge um algoritmo de particionamento de uma página em *pagelets*: como estes não podem ser aninhados em outros *pagelets* com o mesmo tópico, é feita uma verificação da quantidade de *links* existentes em determinada região, e caso este número ultrapasse determinado limiar, esta região é considerada um *pagelet*. O algoritmo então detecta a duplicação de *pagelets* em diversas páginas coletadas, através de seu conteúdo textual, considerando-os como *templates* quando excederem outro limiar de semelhança.

Segundo o autor, a aplicação do algoritmo traz melhoria significativa na precisão para diversos valores de revocação, baseado na utilização de algoritmos de recuperação de informações.

Em Debnath, Mitra e Giles (2005), dois algoritmos são propostos para fazer a separação do conteúdo informativo e do conteúdo não informativo, não se utilizando de nenhuma intervenção por parte do usuário nem de algoritmos de aprendizagem. Os algoritmos demonstrados possuem como entrada um conjunto de páginas *web* de mesma classe e domínio, ou seja, com *design* ou conteúdo similar, e a saída é uma classificação para os blocos identificados.

Blocos, neste trabalho, são identificados como componentes menores que uma página e com conteúdo semânticamente homogêneo, sendo identificados através de um particionamento que leva em consideração algumas *tags HTML* específicas, como <tr>, <p>, <hr> e <ul>. Cada bloco pode ser caracterizado como um bloco redundante ou não, onde blocos redundantes são aqueles que, além de conter certos atributos, ocorrem diversas vezes em várias páginas, caracterizando-o como não informativo.

Primeiramente, é executado o algoritmo "*FeatureExctractor*", que se baseia em heurísticas próprias para realizar a verificação de atributos que ocorrem nos blocos, através dos quais o mesmo será classificado como relevante ou não. Os atributos que são utilizados pelo algoritmo são: conteúdo textual, presença de *tags* indicando texto, listas e propriedades de estilo.

Em seguida, é executado outro algoritmo, "*ContentExtractor*", cujo objetivo é identificar os blocos irrelevantes, baseando-se na ocorrência de blocos similares em múltiplas páginas *web* do mesmo domínio. Para isso, é utilizada a medida "*IBDF*" (Inverse Block Document Frequency), que é inversamente proporcional ao número de documentos em que um mesmo bloco ocorre ou possui bloco similar, onde a similaridade é calculada a partir dos vetores de atributos dos blocos. Constituem exemplos de atributos utilizados para este cálculo o número de termos, o número de imagens, o número de trechos de código *javascript*, além de uma matriz binária de termos contidos nos blocos. Após o cálculo da medida *IBDF* para cada bloco, são classificados como irrelevantes aqueles cujo número de vezes que aparece em outras páginas seja maior que determinado limiar.

Segundo os autores, a técnica produz ótimos valores de precisão e revocação, utilizando blocos como métrica. Em comparação com Lin e Ho (2002), o método proposto

atingiria melhores resultados.

Assim como o trabalho recém descrito, Lin e Ho (2002) também tem como objetivo a separação do conteúdo informativo de uma página *web*, partindo do pressuposto que blocos redundantes são aqueles que são comuns a várias páginas *web* dentro de um mesmo domínio. Entretanto, o método possui outra forma de lidar com a identificação dos blocos e classificação dos mesmos.

Dentro de um contexto onde uma coleção de páginas *web* pertence ao mesmo domínio, as páginas são particionadas conforme as *tags HTML* <table>, já que, na época, em torno de 70% dos *sites* utilizavam tabelas para disposição do conteúdo. Durante esta fase de *parsing*, são recuperados os blocos, estejam eles aninhados ou não, e o conteúdo textual contido nestes trechos.

Em um segundo momento, calcula-se o grau de entropia de cada bloco, para que se possa definir automaticamente se o mesmo é informativo ou não. Para isto, inicialmente são extraídos os termos de cada bloco já removendo *stop words* e aplicando um algoritmo de *stemming* de Porter. Em seguida, é calculado o valor de entropia de cada atributo, com base em uma distribuição ponderada de termos no domínio, utilizando-se *TF-IDF* entre termos e documentos (as páginas *web*). O valor de entropia de um termo, portanto, será correspondente à distribuição de probabilidade de uma linha da matriz gerada. Desta forma, para calcular o valor de entropia de um bloco, basta somar as entropias dos termos que o bloco contém.

Por fim, cada bloco tem seu grau de entropia comparado a um limiar, e se o bloco exceder este valor será considerado redundante. O estudo também demonstra uma abordagem gulosa que é utilizada para determinar automaticamente este limiar para diferentes *sites*.

Chen, Ye e Li (2006), assim como outros trabalhos, também propõem um método que funciona em dois estágios - detecção seguida da remoção dos *templates* - mas o objetivo do trabalho foca mais em melhorar o desempenho da remoção do que aumentar a precisão em si, visto que a idéia é anexar o método ao estágio de construção do índice de um motor de busca. Este estudo também supõe que os *templates* de um determinado *site* compartilham um mesmo conjunto de estilos e trechos de conteúdo.

Para a primeira fase, onde devem ser detectados os blocos que constituem *templates*, cada página é segmentada em blocos através das *tags HTML* <table>, <p> e <ul>. Segundo o artigo, não são utilizadas as *tags* <td> e <tr> pois as mesmas normalmente encapsulam apenas um objeto, não identificando um *template* por si só. Ainda nesta fase são extraídas informações de estilo dos blocos que serão utilizadas na segunda fase, como posição na página e atributos do elemento *HTML*, tamanho da tabela e cor de fundo.

É construída, então, uma árvore representando os blocos, e estes são numerados. Como seria ineficiente procurar *templates* em uma só página, após a construção desta árvore os blocos são agrupados com base nos seus atributos e numeração, e somente após isso acontece a busca por *templates* nestes agrupamentos.

Na segunda fase do método, quando o índice do motor de busca está sendo construído, é necessária a aplicação de uma medida de similaridade para que se possa encontrar blocos semelhantes. É computada então, para cada bloco, a *word offset distribution*, pois esta medida provê dados suficientes como a frequência dos termos e as posições de cada um deles. Tendo calculado esta medida, é construída uma tabela *hash* para armazenar os termos, onde cada um destes aponta para uma lista *linkada*, e cada elemento destes

armazena o identificador do bloco e a sequência de *offset* deste bloco. Os blocos então são submetidos a um filtro, chamado "*Bloom Filter*", que encontra sequências de *offset* similares, garantindo que o método utilize pouca memória e tenha um baixo *overhead* computacional. A técnica acaba por detectar *templates* em agrupamentos que tenham mais de 5 blocos com similaridade acima de um determinado limiar.

Como resultado final, o trabalho obtém precisão próxima aos demais estudos, entretanto, afirma-se que o método é até 40% mais rápido que outros trabalhos.

Em Wang et al. (2008), é argumentado que um dos problemas da remoção de *templates* consiste na necessidade de examinar-se diversas páginas *web* para verificar a existência de conteúdo repetido até que ocorra a remoção do ruído. O processamento, portanto, exige a coleta de um lote de páginas e isso acaba atrasando a atualização de dados, aumentando o espaço necessário e afetando aplicações de tempo real, visto que é preciso manter um *cache* das páginas em coleta.

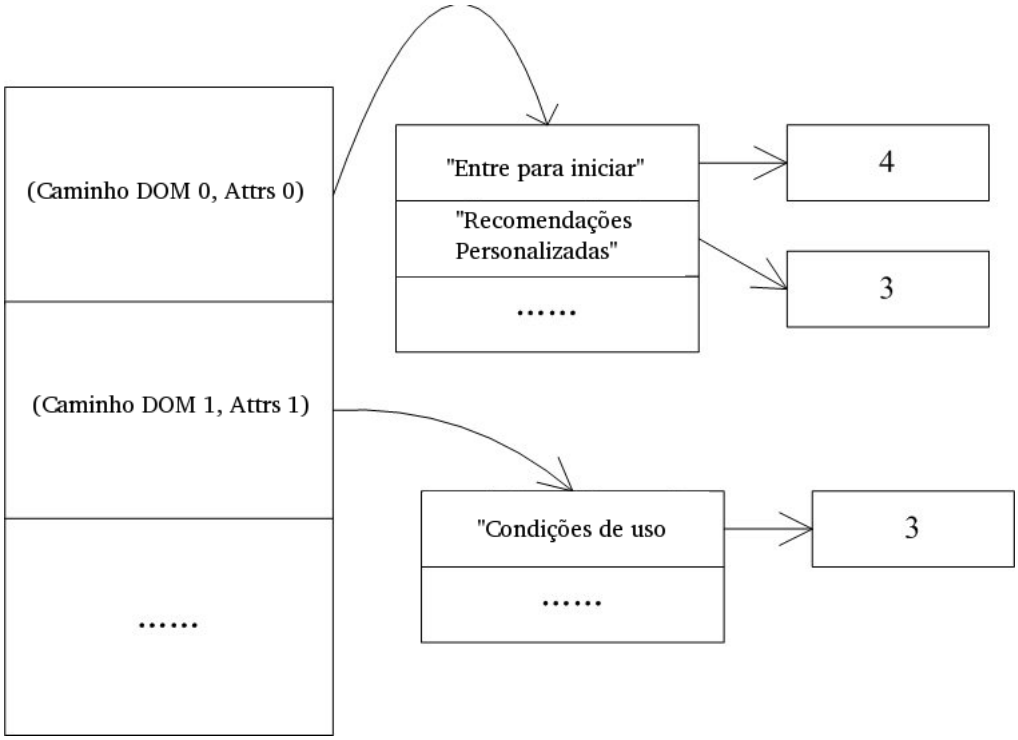

Figura 2.1: Tabela de segmentos de texto - imagem adaptada de Wang et al. (2008)

O método proposto sugere o armazenamento de uma representação compacta, ao invés de toda página *web*, o que reduz o armazenamento das páginas para até 7% do espaço em memória de outros métodos, também eliminando o atraso de atualização. Este trabalho também assume que a repetição de conteúdo constitui ruído.

A técnica primeiramente segmenta cada página em blocos de acordo com *tags HTML* utilizadas como separadores, como <table> e . Cada bloco, por sua vez, é dividido em muitos segmentos de texto, que são constituídos por nodos de texto da árvore *DOM* que ainda são visíveis após a página ser renderizada. A estrutura principal do trabalho - uma tabela de segmentos de texto que contém os conteúdos e as frequências de documentos do *site* - é, então, atualizada, seja através de inserção, atualização ou remoção. A chave desta tabela é composta por dois níveis: o primeiro nível é o estilo do bloco (representado pelo caminho do bloco e atributos do elemento separador mais próximo), e o

segundo nível é o conteúdo do segmento de texto, como pode ser visto na Figura 2.1. Já o campo valor da tabela armazena a quantidade de documentos que contém este segmento de texto. Quando novas páginas vão sendo coletadas, a inserção de novos segmentos de texto na tabela é trivial, porém, a deleção destes segmentos é mais complexa, visto que é utilizada uma estratégia logística que tenta prever quais segmentos de texto não serão mais utilizados, tentando assim reduzir os custos desta remoção.

Para a fase de detecção de ruído, o trabalho faz algumas definições:

- dois blocos são do mesmo tipo se ambos possuem o mesmo papel semântico, onde este papel é determinado pela definição de estilo do artigo
- dois segmentos de texto são iguais se e somente se o conteúdo de dois segmentos for literalmente igual e se os dois pertencem a tipos iguais de blocos

Portanto, os segmentos de texto considerados como ruído são aqueles que possuem uma frequência de documentos maior do que um limiar determinado. Se a taxa de ruído de um bloco, composta pelo somatório dos segmentos de texto considerados como *template* dividido pelo somatório de todos os segmentos de texto de um bloco for maior que o limiar 0.7 utilizado pelo artigo, o bloco é considerado como *template*.

Segundo os autores, os níveis de precisão e revocação atingidos pelo artigo na eliminação de ruído são semelhantes a outros trabalhos, entretanto, o armazenamento é reduzido a apenas 7% do armazenamento de outras técnicas e o atraso de atualização é eliminado.

2.1.2 Técnicas baseadas em segmentação visual

Cai et al. (2003) propõe um novo método para análise da estrutura de conteúdo de uma página baseada na representação visual que é mostrada ao usuário. Segundo os autores, essa segmentação é benéfica para diversos métodos de recuperação de informações, mineração de dados e dispositivos móveis, ao passo que métodos que utilizam a estrutura da página dão mais ênfase para a apresentação do que para a estrutura semântica da página, podendo ainda conter erros quando o código *HTML* é mal formatado. O trabalho em questão se baseia no fato de que dicas visuais e espaciais ajudam o usuário a dividir inconscientemente a página *web* em segmentos semânticos, além do fato dos usuários já esperarem certos segmentos em determinados lugares da página.

"*VIPS*" (Vision Based Page Segmentation), o algoritmo proposto para efetuar a segmentação, tem como objetivo principal a extração desta estrutura semântica da página. Primeiramente, são extraídos blocos através de uma série de heurísticas que analisam tanto a estrutura *DOM* da página (onde cada *tag HTML* obedece regras próprias) quanto informações visuais e espaciais de cada elemento. Após isso, busca-se encontrar separadores entre os blocos, que são constituídos por linhas horizontais e verticais que não cruzam nenhum bloco, como mostra a Figura 2.2. Cada um destes separadores recebe um peso diferente a partir das características visuais dos blocos que estão sendo separados.

Tendo realizado estas tarefas, é criada uma estrutura hierárquica contendo os blocos e separadores, levando em consideração que um bloco pode conter outros, desde que não haja sobreposição. Um exemplo desta estrutura é mostrado na Figura 2.3.

Cada um destes blocos possui uma propriedade chamada Grau de Coerência, que mede o quão coerente ele é - quanto maior este grau, mais consistente é o conteúdo dentro do bloco. É utilizada uma escala de 1 a 10 e, na estrutura hierárquica montada, o grau de coerência de um bloco filho não pode ser menor que o grau de coerência de seu ancestral. Em uma última fase, são analisados os graus de coerência dos blocos, e é decidido

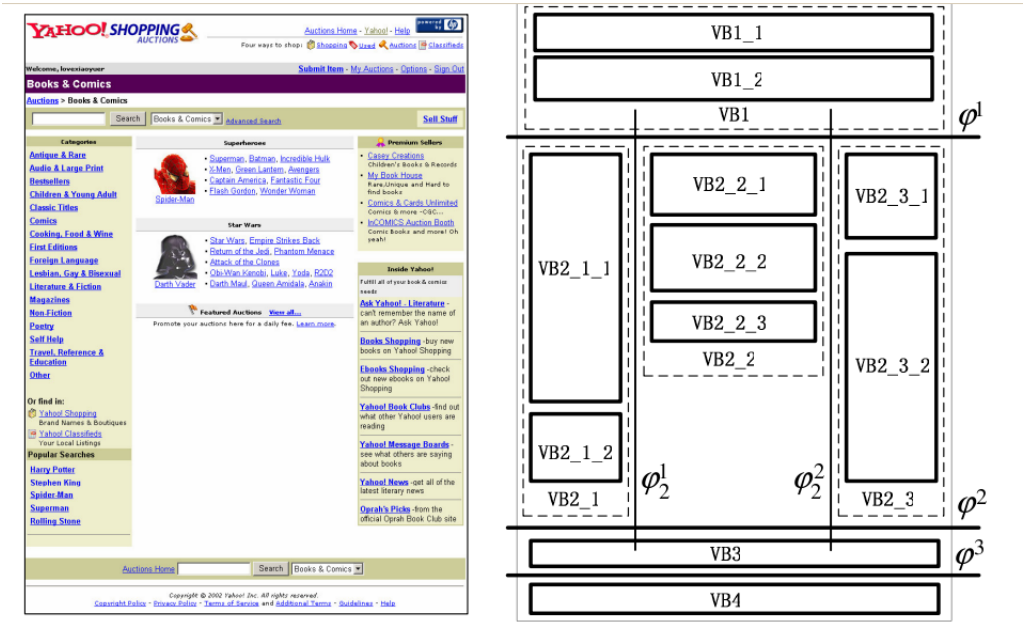

Figura 2.2: Exemplo de página *web* segmentada - imagem adaptada de Cai et al. (2003)

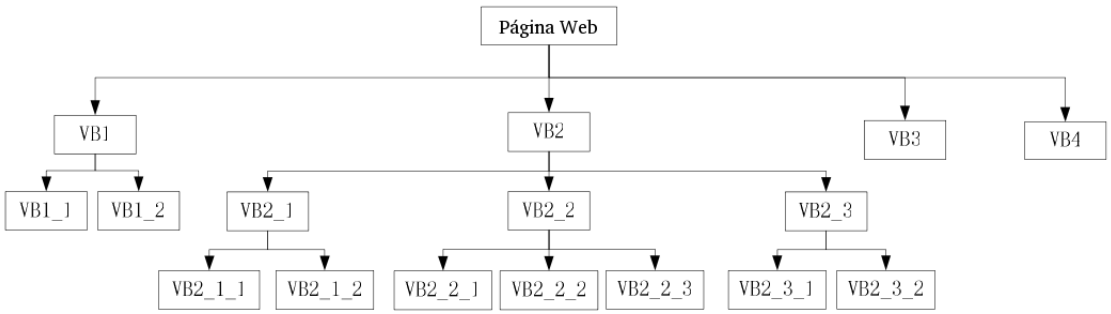

Figura 2.3: Estrutura hierárquica de blocos - imagem adaptada de Cai et al. (2003)

se o processo deve ser refinado ou não. Caso não tenha atingido o grau de coerência mínimo, cuja granularidade é controlada pelo algoritmo, o processo inteiro deve ser repetido recursivamente.

O resultado do estudo foi comparado com o julgamento de avaliadores humanos quanto ao grau de coerência da segmentação da página, e em 93% das vezes o algoritmo pôde detectar a estrutura corretamente. Além de mostrar que essa segmentação auxilia nas tarefas de recuperação de informações, este trabalho, como veremos adiante, servirá de base na tarefa de segmentação para diversos outros trabalhos.

O método proposto em Fernandes et al. (2007) não foca exatamente na detecção de ruído, e sim na atribuição de importância a blocos em páginas *web*. Ainda assim, esta tarefa pode ser de grande auxílio para a detecção de blocos que constituem *templates*, pois o resultado do método - uma coleção de páginas *web* com blocos contendo grau de importância - pode melhorar o resultado de motores de busca e outros sistemas de recuperação de informações.

Este trabalho parte da premissa que a ocorrência de um termo em cada bloco tem um fator de importância no processo de construção de *ranking*, e deve ser levado em consideração quando do cálculo do peso total do termo na página. Como existem diferentes tipos de blocos, entre eles blocos que não estão associados com o tópico central da página, como menus e propagandas, alguns blocos devem receber um valor de importância menor em relação a outros. Surge então a necessidade do cálculo automático dessa importância, que é feito através de informações estatísticas e sem nenhum método de aprendizado, devido à grande quantidade de blocos existentes em uma coleção.

A técnica faz uma série de definições:

- uma página é um conjunto de blocos não sobrepostos
- cada bloco é composto de um rótulo e do conteúdo textual do bloco
- duas páginas são estruturalmente equivalentes se ambas possuem o mesmo número de blocos, e se os blocos possuem o mesmo rótulo em ambas as páginas
- um conjunto de páginas estruturalmente equivalentes forma uma classe de páginas (Figura 2.4)
- uma classe de blocos é um conjunto de blocos de diferentes páginas *web*, mas de um mesmo site, onde todos os blocos contêm o mesmo rótulo

Para a atribuição dos pesos aos blocos, são analisadas todas as classes de blocos da coleção de páginas *web*, e é calculada então a medida "*ICF*" (Inverse Class Frequency) de um termo em uma classe de blocos. Esta medida é composta pelo *log* do número de classes dividido pelo número de blocos da classe em que o termo desejado aparece, sendo semelhante ao *IDF* (Inverse Document Frequency). É derivada então a medida *ICF* média, "*AICF*" (Average Inverse Document Frequency), que representa o valor médio de todos os *ICFs* dos termos que aparecem na classe dos blocos, e esta medida informa que classes de blocos com muita repetição terão um valor *AICF* baixo, podendo assim atribuir um grau de importância baixa ao bloco. Outra medida também é proposta, a "*BCS*" (Block Class Spread), que diz que a importância de cada classe de blocos também deve levar em consideração a similaridade de cada bloco da classe com os outros existentes na página. Assim, blocos com termos relacionados em outros blocos provavelmente estariam

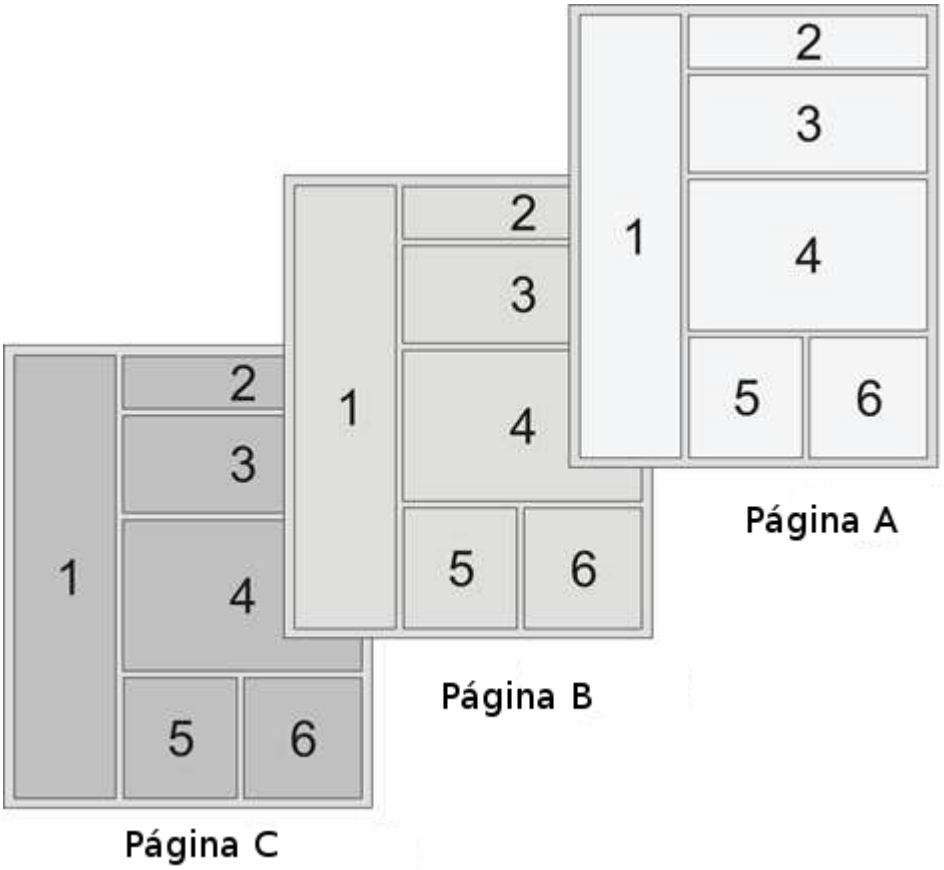

Figura 2.4: Páginas *web* de uma mesma classe - imagem adaptada de Fernandes et al. (2007)

relacionados com o conteúdo principal de página, como por exemplo um bloco contendo o título, que tem ligação com blocos do conteúdo da notícia.

O resultado do trabalho afirma que a qualidade dos resultados é melhorada em comparação quando nenhum método é utilizado, e que a aplicação da técnica não impacta no desempenho.

Li e Ezeife (2006) desenvolvem um sistema chamado "*WebPageCleaner*", para detecção de ruído baseado na extração de blocos do sistema *VIPS*. O sistema identifica blocos de conteúdo importante através de características físicas do bloco, como localização, porcentagem de *links* no bloco e similaridade de um bloco com outros. O sistema parte de 4 suposições:

- a coleção de páginas *web* compartilha conteúdos e apresentação padrão
- cada página do *site* contém blocos cujo conteúdo se repete em outras páginas
- blocos ruidosos normalmente são localizados nas extremidades das páginas
- blocos ruidosos geralmente contém muitos *links* no seu conteúdo

O funcionamento do sistema se dá em 3 fases distintas, que serão explicadas a seguir. A primeira fase consiste em obter os blocos utilizando o sistema *VIPS*, e extrair informações importantes que são são armazenadas como um registro em um banco de dados. As informações extraídas são as seguintes:

- identificador do bloco
- identificador da página
- conteúdo do bloco após remoção de pontuação
- *fingerprint* do conteúdo, que são *tags* curtas para representar objetos grandes (neste caso inteiros que identificam textos longos através da fórmula de *fingerprint* de Rabins), utilizada para encontrar rapidamente blocos duplicados
- posição, calculada através de atributos disponibilizados pelo *VIPS*, cujo valor reside entre 0 e 1, e quanto mais próximo destes extremos, indica que o bloco está posicionado nas extremidades da página, sendo considerado menos importante
- porcentagem de *links* existentes (normalizados através da divisão da quantidade de *links* pela quantidade total de texto), que, em maior quantidade, indicam um bloco menos importante
- nível de similaridade com outros blocos
- importância geral do bloco

As duas últimas propriedades ainda são alteradas no passo seguinte, e todos esses atributos são armazenados na forma de uma tabela em um banco de dados.

Na fase seguinte, o sistema busca identificar a importância de cada bloco, assumindo que existem 3 possibilidades entre 2 blocos: os blocos serem exatamente iguais, aproximadamente iguais ou totalmente diferentes. Blocos com conteúdo totalmente igual são

detectados pelo *fingerprint*, são tratados como ruidosos e imediatamente descartados do sistema, tendo seu registro removido do banco de dados. Mais difícil, entretanto, é encontrar blocos aproximadamente iguais, e este passo é feito calculando-se o nível de similaridade entre todos os blocos, que é definido pelo número de *tokens* comuns sobre o número de *tokens* únicos nos dois blocos.

O cálculo da similaridade se dá da seguinte forma: a tabela de blocos é ordenada pelo conteúdo, de forma que blocos com conteúdos parecidos se mantenham próximos; a similaridade então é calculada de 2 em 2 registros, e quando a janela de comparação desliza uma posição para comparação de 2 novos registros, se o nível de similaridade encontrado for maior do que o valor já existente no registro já utilizado para comparação, o valor é atualizado, caso contrário, é mantido a similaridade antiga.

Por fim, é calculada a importância total do bloco através da atribuição de pesos diferentes para cada uma das métricas: o nível de similaridade é dividido por 2, a porcentagem de *links* é divida por 3 e a posição é divida por 6, ou seja, a maior importância é entre a similaridade de conteúdos. O valor encontrado, quando diminuído de 1, será um valor entre 0 e 1, e quanto mais próximo de 1, mais importante é o bloco. No final, os N blocos mais importantes, onde N é definido manualmente através de observações da média de blocos importantes em páginas *web*, são exportados para um arquivo único que é repassado a um classificador de textos, no qual são feitos os experimentos para avaliação do trabalho.

Song et al. (2004) também investiga como encontrar um modelo para atribuir valores de importância a blocos nas páginas automaticamente, com a página já tendo sido segmentada pelo sistema *VIPS*, e define este problema como um problema de aprendizagem. O trabalho extrai atributos espaciais e de conteúdo que são resultantes do sistema *VIPS*, e constrói um vetor de atributos para cada bloco, quando estes, por sua vez, são utilizados como conjunto de treinamento para algoritmos de aprendizado. Os algoritmos utilizados são *SVM* (Support Vector Machines) e redes neurais, que são utilizados para o aprendizado de modelos de importância dos blocos.

O estudo inicia conduzindo um estudo com usuários para verificar se realmente há percepção de diferença de importância entre diferentes blocos, e o resultado é positivo, ou seja, os usuários têm uma opinião consistente sobre diferentes blocos, validando o objetivo do trabalho em questão. Os autores também partem do princípio que os projetistas de páginas *web* colocam informações mais importantes no centro das páginas, deixando as extremidades para conteúdo irrelevante. Além disso, o conteúdo dos blocos também é utilizado para a criação do modelo.

Em um primeiro momento, características absolutas de posicionamento são extraídas do sistema *VIPS*, e são transformadas em coordenadas relativas, normalizando-as com um valor fixo ao invés do valor do tamanho total da página, pois para páginas longas, conteúdos que ficam no topo normalmente devem ser considerados mais importantes do que conteúdos nos quais é necessário deslizar o botão de rolagem para sua visualização. São utilizadas, portanto, 9 características para descrição do conteúdo de um bloco:

- número de imagens do bloco
- tamanho das imagens do bloco
- número de *links HTML*
- quantidade de texto de cada *link*

- quantidade de texto total do bloco
- número de componentes de interação com o usuário
- tamanho dos componentes de interação com o usuário
- número de formulários
- tamanho dos formulários

Estes valores sim, são normalizados a partir dos valores totais contidos em cada página.

No segundo estágio, o artigo adota uma abordagem de aprendizado através de exemplos, onde alguns blocos são préviamente rotulados por diversas pessoas, formando o conjunto de treinamento. Se o problema for encarado como um problema de regressão, onde o valor da importância de cada bloco é contínuo, é aplicado o método de aprendizado de redes neurais, e se o valor da importância for discreto, o problema é tratado como um problema de classificação, utilizando-se o método de aprendizado *SVM*. Ambos os métodos encontram uma função cuja entrada são os atributos dos blocos e geram um resultado de importância como saída.

Segundo os autores, o método proposto atinge resultados em torno de 80% de precisão na atribuição de importância aos blocos, que, segundo o estudo feito pelos autores, é um valor muito próximo da atribuição feita por um humano.

O trabalho de Kovacevic et al. (2002) inicia motivando o problema, explicando que classificadores geralmente levam em conta para classificação apenas medidas entrópicas das palavras, através ou de Ganho de Informação, ou *TF-IDF*, e que informações visuais não são levadas em consideração. Entretanto, assim como em outros trabalhos já demonstrados, os usuários tendem a esperar que projetistas de páginas *web* coloquem certas informações em áreas pré-definidas do navegador, ajudando a definir heurísticas implícitas para o reconhecimento das informações importantes. Pode-se notar que geralmente palavras que estão no centro da página, por exemplo, são mais importantes do que aquelas mais para as extremidades, e que estas características devem ser ponderadas. O mesmo acontece com *links*, cuja importância também deve ser ponderada não só de acordo com sua localização, mas também pelo fator densidade, visto que regiões com muitos *links* normalmente constiuem menus de navegação, o que diminui sua importância.

O trabalho propõe então uma nova representação de uma página a partir do código fonte *HTML*, incluindo informações visuais, e mostra como essas informações podem ser utilizadas para reconhecimento de áreas comuns em páginas *web*. Esta representação é hierárquica, inclui coordenadas do navegador para cada objeto *HTML*, e através dela é possível definir heurísticas para o reconhecimento das áreas comuns como cabeçalho, menus à direita e à esquerda, e centro da página.

O algoritmo primeiramente define uma tela virtual que por sua vez define um sistema de coordenadas dentro da página *web*, que será utilizada em um outro momento. É feito então um *parsing* do código fonte *HTML*, separando as *tags HTML* e seus atributos do conteúdo existente, montando uma árvore chamada "*mTree*".

Após a construção dessa árvore, é possível aplicar um algoritmo de cálculo de coordenadas, que é basicamente o que um navegador faz ao renderizar uma página. O renderizador utilizado é baseado no navegador *Internet Explorer*, mas tem restrições como não aceitar *frames* nem folhas de estilo, só aceitando elementos que estão na árvore *mTree*.

Após esta renderização, a nova árvore contém as coordenadas dos polígonos de cada elemento dessa árvore em relação à tela virtual. Por fim, com esta estrutura torna-se possível definir uma série de heurísticas para o reconhecimento das áreas comuns das páginas, entre elas cabeçalhos, rodapés, menus da esquerda, direita e centro.

Em Burget e Rudolfova (2009) é apresentado um método de detecção de áreas interessantes em uma página *web*, que se baseia no reconhecimento desses blocos e na sua posterior classificação com base na sua aparência, ou seja, leva em consideração características como posições mútuas dos blocos e outras informações visuais. Assume-se que usuários identificam o conteúdo principal observando as características visuais dos blocos implícitos nas páginas *web*, que normalmente possuem uma estrutura semelhante e até um pouco previsível. Essa separação, objetivada pelos projetistas para os usuários, é o que torna importante as características visuais para reconhecimento de blocos. O trabalho tenta modelar a percepção humana da página o mais exatamente possível, usando informações visuais que estão disponíveis para o usuário, ou seja, o método trabalha com um modelo da página renderizada ao invés de um modelo do código do documento. A técnica é dividida em quatro estágios, que são descritos a seguir.

A primeira fase do algoritmo envolve a renderização da página, onde são obtidas informações sobre o posicionamento e estilo visual dos elementos básicos do documento através da utilização de um módulo chamado "*CSSBox Engine*", que carrega o código fonte *HTML* e extrai tais informações. O resultado é um conjunto de áreas retangulares, chamadas de caixas, contendo uma parte do documento e que normalmente englobam elementos únicos do código fonte. Desta forma, surgem vários tipos de caixas, como caixas de elementos, caixas de texto, ou caixas de substituição, que são utilizadas por imagens, por exemplo. Cada uma dessas caixas já conta com uma série de atributos, como tamanho e posição na página, cor de fundo, propriedade de fontes, bordas e conteúdo.

O segundo estágio consiste na detecção de caixas que formam blocos visuais independentes, ou seja, necessita-se encontrar caixas que estão separadas visualmente, possuindo conteúdo ou borda definida, e que essa caixa seja aparente na área final. Esta detecção forma as áreas visuais básicas, que são estruturadas em uma árvore, visto que uma área pode englobar completamente outras áreas.

Em um terceiro momento, detecta-se as linhas de texto, onde são unidas áreas que formam linhas de texto únicas. Estas linhas são a menor área visual possível, e áreas deste tipo que estejam na mesma posição vertical da página, não intercaladas com outras áreas, acabam por ser unidas.

Por fim, o quarto estágio envolve o reconhecimento de áreas visuais maiores, ou seja, são criadas novas áreas na árvore, agrupando áreas adjacentes que possuam o mesmo estilo e criando os blocos.

O resultado do algoritmo é uma árvore de áreas, onde a raiz representa a página inteira, e a árvore representa exatamente o que o usuário vê na tela. Para cada área visual, são calculados atributos de fonte, posição, texto e cor, e estes são utilizados como entrada para um classificador *J48*. Vale ressaltar que o método classifica diferentes áreas, mas não chega a atribuir importância a elas, como em outros trabalhos.

2.1.3 Técnicas baseadas em similaridade estrutural

O trabalho de Yi, Liu e Li (2003) faz parte de um conjunto de técnicas que baseiam a detecção de ruído na similaridade estrutural do código fonte *HTML*, partindo do pressuposto que blocos ruidosos compartilham conteúdos e estilos comuns, ao passo que no

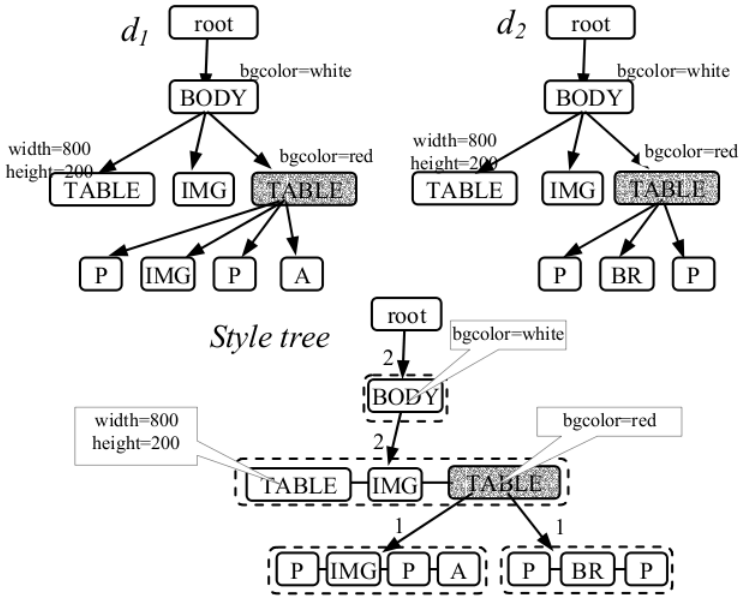

Figura 2.5: Exemplo de árvores *DOM* (acima), e de árvore *SST* (abaixo) - imagem adaptada de Yi, Liu e Li (2003)

conteúdo principal estas características diferem. O estudo afirma que árvores *DOM* são suficientes para representar o conteúdo de uma página, mas não servem para captar os estilos e a entropia existente no conteúdo e estilo para várias páginas.

Dessa forma, o artigo propõe uma estrutura em árvore para este armazenamento, chamada "*ST*" (StyleTree). No momento em que várias páginas são coletadas e as novas informações vão sendo inseridas ou atualizadas na árvore, passa-se a ter uma árvore denominada "*SST*" (Site StyleTree). Essa estrutura permite visualizar com facilidade quais partes das árvores *DOM* são comuns e quais são diferentes, determinando quais ramificações consituem *templates*. Um exemplo de "*SST*" pode ser visualizado na Figura 2.5.

Apesar da semelhança entre as árvores *DOM* e *SST*, pode-se ressaltar algumas diferenças. Um nodo da árvore *SST* pode conter mais de uma *tag* da árvore *DOM*, quando determinada ramificação da árvore é diferente em páginas diferentes. Os elementos da *SST* também armazenam um contador que armazena o número de vezes que tal elemento se repete por diferentes páginas, assim como os atributos de estilo da *tag* armazenada. A construção da *SST* é simples: é construída uma *ST* para a primeira página coletada, e em seguida vai se atualizando a *SST* de uma maneira *top-down* para cada página coletada. Em cada elemento da *SST*, se a sequência de filhos desse elemento são os mesmos da *SST* já existente, o contador do elemento é incrementado. Se não existir, as *tags* são convertidas em nodos da árvore de estilos e são adicionadas. Este procedimento é executado recursivamente até que toda árvore *DOM* tenha sido percorrida.

O método para detecção de ruído parte de duas premissas: quanto mais estilos de apresentação um nodo da *SST* tem, mais importante é o mesmo, e quanto mais diversificado for o conteúdo de um nodo, também aumenta a sua relevância. Desta forma é possível combinar tanto apresentação quanto conteúdo para determinação do conteúdo principal da página.

Medidas baseadas na Teoria da Informação são utilizadas para calcular a entropia de cada nodo, levando em consideração, para cada nodo, a importância dos filhos deste, visto

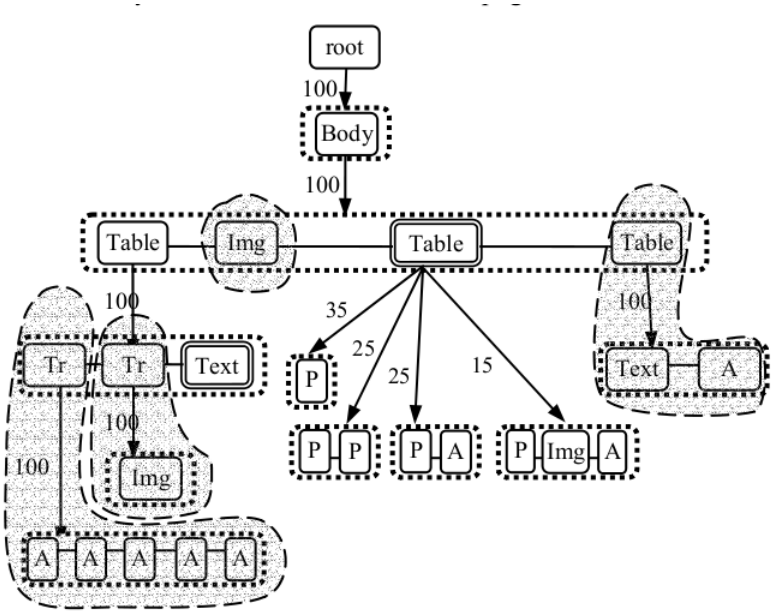

Figura 2.6: Regiões consideradas como ruído - imagem adaptada de Yi, Liu e Li (2003)

que é necessário que todos os filhos de um nodo sejam considerados como ruído para que o elemento seja considerado ruidoso também. Se a importância de um nodo for menor que um limiar, o nodo é classificado como ruído, como visto na Figura 2.6.

O estudo ainda mostra algumas abordagens para aumentar a velocidade de detecção, eliminando a necessidade de percorrimento de toda árvore para encontrar nodos ruidosos. O trabalho mostra que a performance de tarefas de mineração de dados como clusterização e classificação de páginas *web* melhoram consideravelmente.

Vieira et al. (2006) também aborda o problema da detecção de ruído amparado no fato que normalmente *templates* possuem características estruturais idênticas, pois são gerados dinamicamente de maneira semelhante. Dessa forma, o problema pode ser descrito como encontrar um mapeamento ótimo entre as árvores *DOM* das páginas, baseado em uma formulação restrita do problema de mapeamento *top-down* entre árvores, e a partir desse mapeamento, nodos e sub-árvores idênticas representariam os *templates*. O uso dessa estrutura da página, segundo os autores, garante que o método descrito obtenha alta precisão mesmo com poucas páginas como amostragem.

O processo ocorre em duas fases, o que, segundo o autor, torna o processo mais eficiente: a primeira fase, a detecção dos *templates*, é custosa e deve ser realizada sobre uma pequena parte da coleção de documentos; na segunda fase, o *template* derivado é removido das páginas restantes da coleção eficazmente. Inicialmente, cada página *web* da coleção é representada por uma árvore rotulada, ordenada e com raiz, correspondente à árvore *DOM*. Um mapeamento entre duas dessas árvores é composto por uma sequência de operações de edição que transformam uma árvore na outra, ignorando a ordem com que essas operações são aplicadas. Neste mapeamento, 3 operações são possíveis, inserção, remoção, e substituição de nodos, e um exemplo de mapeamento pode ser visto na Figura 2.7. Ao associar um custo para cada uma dessas operações, é possível associar um custo total ao mapeamento, mas como podem existir diversos mapeamentos possíveis, o ideal é encontrar o mapeamento ótimo, o que chamamos de distância de edição de árvores.

Diversas soluções foram propostas para esse problema, e o artigo utiliza o mapea-

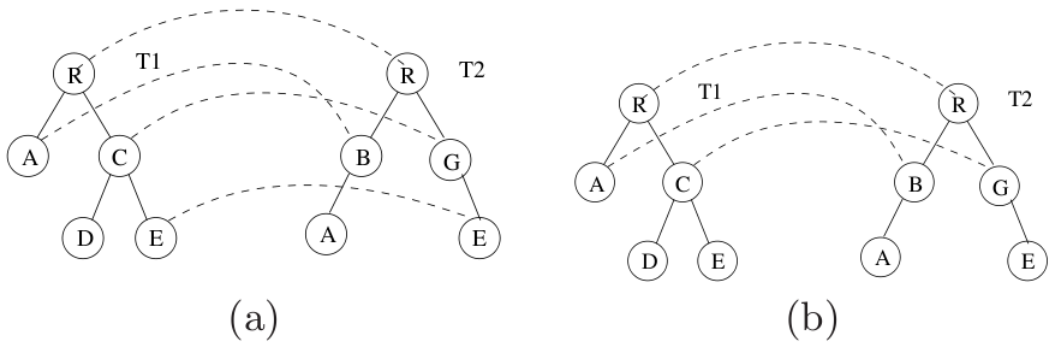

Figura 2.7: Mapeamentos entre diferentes árvores - imagem adaptada de Vieira et al. (2006)

mento *top-down* entre páginas com a restrição de que as operações ocorram apenas nos nodos folha da árvore. Um algoritmo adaptado é utilizado para determinar o mapeamento de custo mínimo, mas o algoritmo é estendido para buscar a sequência de operações que levaram a esse mapeamento, já que essas operações serão posteriormente utilizadas para reconstruir o *template* detectado. Tendo sido encontrado o ruído, basta remover esta subárvore das outras páginas da coleção, algoritmo que também é apresentado no artigo.

O artigo afirma, em seus resultados, que é preciso apenas de 5 a 10% do número de páginas necessárias para detecção do *template*, em comparação com o método que utiliza *SSTs*. A eficiência da remoção é medida, assim como o impacto no desempenho da clusterização e classificação em páginas web, também afirmando ser superior ao método proposto em *SST*.

O trabalho de Velloso e Dorneles (2013) busca segmentar e remover o ruído de páginas *web* baseado na sequência de caminhos de *tags* da árvore *DOM*. A técnica leva em consideração a estrutura e os estilos de uma página, sendo totalmente automática e independente de domínio. Além disso, a técnica não depende de estruturas *HTML* específicas e é capaz de identificar o ruído com apenas uma página. Segundo os autores, a proposta atinge ótimos resultados para a remoção de ruído de *sites* comerciais e institucionais, sem comprometer a região principal da página *web*.

2.1.4 Técnicas híbridas

Kushmerick (1999) é um trabalho um pouco mais antigo que os demais, mas que tem grande importância no desenvolvimento de técnicas de eliminação de ruído. O trabalho afirma que imagens de propaganda em páginas *web* causam diversos problemas, desde inconveniência ao usuário que tenta encontrar conteúdo, até o aumento do tempo de carregamento da página.

Para isto, é implementado um sistema chamado "*AdEater*", que é um assistente de navegação que semi-automaticamente remove imagens de propagandas das páginas *web* através de uma abordagem de aprendizado indutivo. Após uma fase de treinamento, o sistema pode remover as imagens antes de carregar a página, evitando que seja feito o *download* das mesmas e consequentemente diminuindo o tempo de espera para o carregamento da página.

Primeiramente é coletado um conjunto de imagens de treino que, necessitando-se de intervenção do usuário, são rotulados como propaganda ou não. Estas imagens são sub-

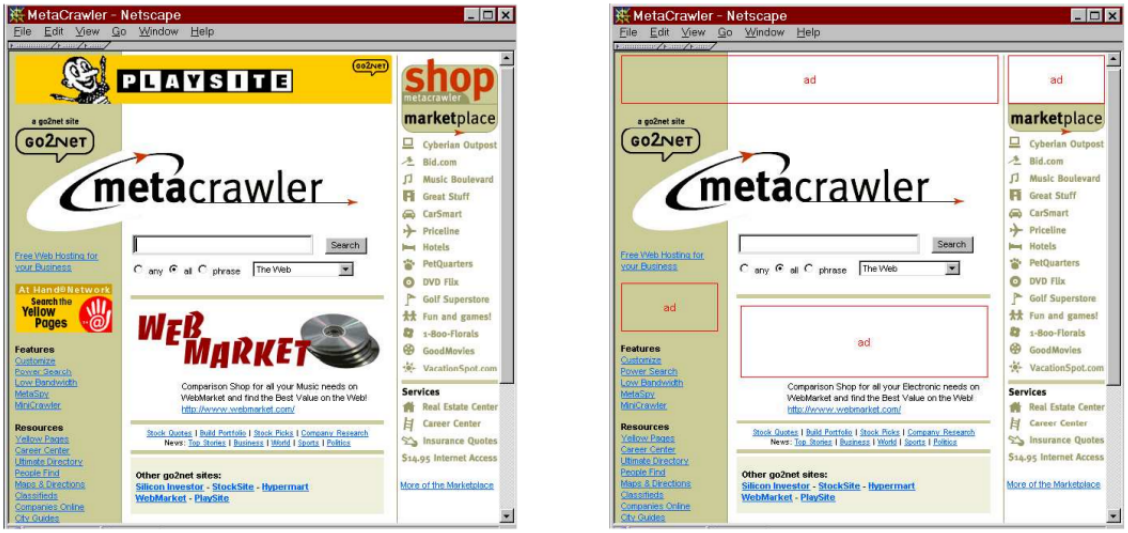

Figura 2.8: Exemplo de imagens removidas de uma página *web* - imagem adaptada de Kushmerick (1999)

metidas a um classificador, para que o mesmo aprenda as regras que mapearão uma imagem em valores correspondentes a propaganda ou não. Para que a imagem possa ser removida antes do seu *download*, é necessário utilizar um vetor de atributos extraídos diretamente do código fonte *HTML*, como por exemplo tamanho e posição da imagem, localização do servidor da imagem e conteúdos de texto da imagem. Só são utilizadas imagens que também sejam *links*, visto que raramente imagens propagandas não contém *links*.

Em um segundo momento, é realizada uma fase *offline* de treinamento, onde é utilizado um método de aprendizado indutivo que resultará em um classificador. Este deve ter como propriedades a rapidez, a baixa sensibilidade a ruído devido a códigos *HTML* mal formatados ou faltantes, escalabilidade para números maiores de atributos e incrementabilidade, visto que a *web* evolui e o classificador acaba por se tornar obsoleto. O algoritmo de aprendizado utilizado é o *C4.5*, derivando um conjunto de 25 regras para classificar imagens em propagandas ou não. Por fim, as páginas classificadas como propaganda são removidas, e este módulo é implementado como um *proxy*, que remove os candidatos a propaganda antes que os mesmos sejam carregados.

O autor alega que o sistema demora apenas 6 minutos para rodar a fase de treinamento *online*, e em torno de 70ms para remover cada propaganda, alcançando uma precisão de 97% de imagens removidas. Exemplos de remoção de imagens com o sistema "*AdEater*" podem ser visualizados nas Figuras 2.8 e 2.9.

2.1.5 Eliminação de ruído através de propriedades textuais

Há um grupo de trabalhos, composto entre outros por Kohlschütter, Fankhauser e Nejdl (2010), Weninger, Hsu e Han (2010) e Sun, Song e Liao (2011), que buscam identificar o ruído baseado em propriedades textuais contidas nas páginas *web*. Neste trabalho, foi implementada e utilizada como *baseline* a técnica descrita em Sun, Song e Liao (2011), dado que a mesma possibilitava a construção de um modelo de extração com apenas uma página, além de não pressupor nenhuma estrutura específica. Além disso, este trabalho apresentava bons resultados em relação a outros da literatura.

O método proposto pelos autores do *baseline* assume que o conteúdo principal da

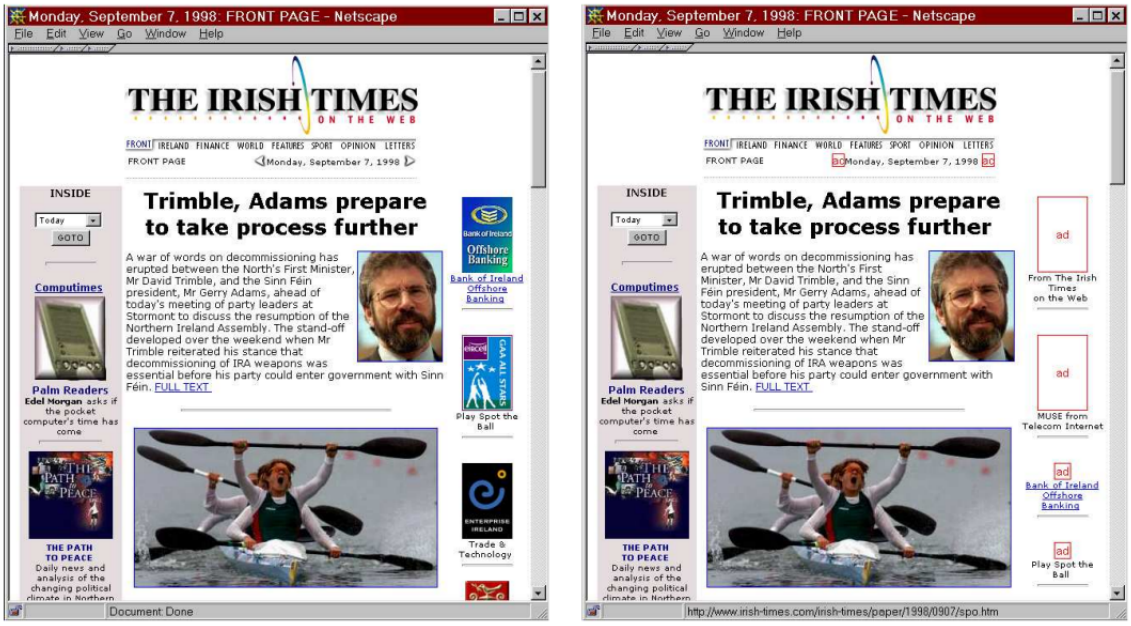

Figura 2.9: Outro exemplo de página *web* com imagens removidas- imagem adaptada de Kushmerick (1999)

página possui mais elementos de texto do que elementos de formatação. Já um trecho ruidoso, por sua vez, seria composto de um número maior de elementos de formatação, além de possuir textos mais curtos e mais *links*.

São definidas então duas medidas, chamadas de Densidade de Texto (DT_i) e Densidade de Texto Composta (DTC_i). Estas medidas funcionam da seguinte maneira: após realizar o *parsing* dos elementos do código *HTML* e representar a página *web* como uma árvore, todos os comentários, *scripts* e elementos de estilo são removidos. Então, para cada nodo da árvore resultante, são contados o número de caracteres abaixo deste nodo, juntamente com o número de *tags*. A proporção entre estes dois parâmetros é chamada de Densidade de Texto, representada abaixo.

$$DT_i = \frac{C_i}{T_i}$$

Assim, quanto maior o número de caracteres existentes em uma ramificação da árvore *DOM*, maior é a probabilidade desta sub-árvore representar conteúdo relevante. Se o número de *tags* for muito alto, a Densidade de Texto possuirá um valor baixo, o que significa que a taxa de formatação é alta e a probabilidade desta ramificação ser ruído também será alta.

Para ilustrar o funcionamento desta medida, pode-se utilizar como exemplo o código *HTML* a seguir:

```
<li class="medium-image">
  <h3 class="feature-header">
    <a class="story">
        Model journalist
    </a>
  </h3>
  <p char_count="52">
    How the BBC's Brian Hanrahan became a household name
```

```
  </p>
  <hr>
</li>
```

Dado este trecho, abaixo é calculado a Densidade de Texto de cada elemento existente no código, evidenciando-se o número de caracteres e de *tags* existentes.

$$Elemento < li >: C_i = 68, T_i = 4, DT_i = 17$$

$$Elemento < h3 >: C_i = 16, T_i = 1, DT_i = 16$$

$$Elemento < a >: C_i = 16, T_i = 1, DT_i = 16$$

$$Elemento < p >: C_i = 52, T_i = 1, DT_i = 52$$

$$Elemento < hr >: C_i = 0, T_i = 1, DT_i = 0$$

Apesar de funcionar como um bom indicativo, esta medida ainda pode ser acrescida de mais informações, como a quantidade de *links*, por exemplo. Isto é possível devido ao fato da maior parte de ruído em páginas *web* ser constituídos de *links* para outras páginas. Assim, outra medida é criada, a Densidade de Texto Composta, que incorpora o número de caracteres de *links* em cada sub-árvore, juntamente com o número de elementos de *links* existentes em cada ramificação. A Densidade de Texto Composta pode ser calculada através da fórmula abaixo:

$$DTC_i = \frac{C_i}{T_i} \log_{\ln(\frac{C_i}{\neg CL_i} CL_i + \frac{CL_b}{C_b} C_i + e)} (\frac{C_i}{CL_i} \frac{T_i}{TL_i})$$

Nesta medida, os seguintes elementos são utilizados para o cálculo:

- C_i - Número de caracteres
- T_i - Número de *tags*
- CL_i - Número de caracteres de *links*
- TL_i - Número de elementos *links*
- $\neg CL_i$ - Número de caracteres que não são de *links*
- CL_b - Número de caracteres de *links* abaixo da *tag* **<body>**
- C_b - Número de caracteres de texto abaixo de **<body>**

Quando qualquer denominador é zero, tal valor é ajustado para um. Nesta fórmula, observa-se a existência de uma proporção entre caracteres de texto e caracteres de *links* ($\frac{C_i}{CL_i}$), assim como uma proporção entre *tags* comuns e *tags* de *links* ($\frac{T_i}{TL_i}$). Este trecho da fórmula garante que altos valores de densidade serão atribuídos quando existirem mais caracteres de texto e menos *links*. A outra porção da fórmula que envolve a *tag* **<body>** é útil para manter um equilíbrio, não permitindo que nodos muito longos e homogeneamente formatados recebam valores altos, nem que nodos compostos de textos curtos recebam valores baixos. Desta maneira, é garantido que nodos com mais texto e menos formatação irão receber valores altos e vice-versa.

Contudo, após o cálculo da Densidade de Texto Composta, alguns nodos ruidosos podem receber erroneamente altos valores de densidade, assim como nodos de conteúdo

podem receber valores baixos. Desta maneira, nodos de conteúdo acabariam sendo descartados na fase de extração. Para resolver o problema, os autores propuseram outra medida, chamada de Soma das Densidades, que como a fórmula abaixo mostra, soma todas as Densidades de Texto Compostas dos descendentes de um elemento:

$$\textit{Soma das Densidades} = \sum_{i \in C} \textit{Densidade de Texto Composta}_i$$

Essa medida é baseada no fato de que todos os nodos de conteúdo são descendentes de outro nodo na árvore *DOM*. Se o ancestral possui muitos nodos filhos considerados como conteúdo, suas densidades serão somadas e o ancestral irá receber um valor alto em sua Soma das Densidades. Esse valor é utilizado principalmente para a extração dos nodos. Como exemplo, é possível supor que a página *web* contenha apenas um bloco de conteúdo: seria necessário encontrar o elemento com maior Soma das Densidades, e então extrair todos seus descendentes e ancestrais. Como no mundo real é praticamente impossível uma página possuir apenas um bloco de conteúdo, então para cada nodo que possuir a Densidade de Texto Composta acima de determinado limiar, o procedimento acima é repetido.

Contudo, a descoberta do limiar correto não é simples. Em um primeiro momento, Sun, Song e Liao (2011) propuseram utilizar como limiar a *Densidade de Texto Composta* da *tag* **<body>**. Entretanto, em certo momento perceberam que alguns blocos de conteúdo possuíam *Densidade de Texto Composta* menor que o a do componente **<body>**. A solução foi então primeiramente obter o valor máximo de *Soma das Densidades* de toda a página, e após isso, transformar o limiar no mesmo valor do elemento com menor *Densidade de Texto Composta* presente no caminho deste nodo de *Soma das Densidades* máximo até a *tag* **<body>**.

2.2 Comparação entre as técnicas

Após a descrição do método utilizado por cada um dos trabalhos, é possível sumarizar algumas das características presentes ou não em cada técnica. Para realizar este cruzamento, duas tabelas são utilizadas, uma contendo todos os critérios, como visualizado na Tabela 2.1, e outra contendo a lista de trabalhos (Tabela 2.2). Na Tabela 2.3, podemos ver o cruzamento entre os trabalhos selecionados e os critérios utilizados para sumarização.

Tabela 2.1: Critérios utilizados para sumarização

1	Necessidade de intervenção manual
2	Utilização de métodos de aprendizagem
3	*Tags* utilizadas para segmentação
4	Utilização do sistema *VIPS* para segmentação
5	Utilização de conteúdo textual
6	Utilização de folhas de estilo
7	Utilização da densidade de links

O cruzamento entre os diferentes critérios e a aplicação dos mesmos por cada trabalho pode ser vista na tabela a seguir.

A Tabela 2.3 evidencia a falta de um padrão claro na literatura para a identificação e eliminação de ruído, embora não seja esta a intenção deste trabalho. Pode-se observar que, embora pertencentes a um mesmo grupo (conforme a classificação proposta), os trabalhos utilizam diferentes metodologias em busca do objetivo final, e nenhum deles foca esforços para remoção de ruídos da *web* oculta.

Tabela 2.2: Trabalhos

A	(BAR-YOSSEF; RAJAGOPALAN, 2002)
B	(DEBNATH; MITRA; GILES, 2005)
C	(LIN; HO, 2002)
D	(CHEN; YE; LI, 2006)
E	(WANG et al., 2008)
F	(CAI et al., 2003)
G	(FERNANDES et al., 2007)
H	(LI; EZEIFE, 2006)
I	(SONG et al., 2004)
J	(KOVACEVIC et al., 2002)
K	(BURGET; RUDOLFOVA, 2009)
L	(YI; LIU; LI, 2003)
M	(VIEIRA et al., 2006)
N	(KUSHMERICK, 1999)
O	(KOHLSCHüTTER; FANKHAUSER; NEJDL, 2010)
P	(WENINGER; HSU; HAN, 2010)
Q	(SUN; SONG; LIAO, 2011)
R	(VELLOSO; DORNELES, 2013)

Tabela 2.3: Comparação entre as técnicas

Trabalho * Critérios	1	2	3	4	5	6	7
A	Não	Sim	**<a>**	Não	Sim	Não	Sim
B	Não	Não	**<tr>**, **<p>**, **<hr>** e ** **	Não	Sim	Sim	Não
C	Não	Não	**<table>**	Não	Sim	Não	Não
D	Não	Não	**<table>**, **<p>**, ** **	Não	Sim	Sim	Não
E	Não	Não	**<table>**, ** **	Não	Sim	Sim	Não
F	Não	Não	N/A	Sim	Não	Sim	Não
G	Não	Não	N/A	Sim	Sim	Não	Não
H	Não	Não	N/A	Sim	Sim	Não	Sim
I	Sim	Sim	N/A	Sim	Sim	Não	Sim
J	Não	Não	N/A	Não	Sim	Não	Sim
K	Não	Não	N/A	Não	Sim	Sim	Não
L	Não	Não	N/A	Não	Sim	Sim	Não
M	Não	Não	N/A	Não	Sim	Sim	Não
N	Sim	Sim	N/A	Não	Não	Não	Não
O	Não	Não	N/A	Não	Sim	Não	Sim
P	Não	Sim	N/A	Não	Sim	Não	Sim
Q	Não	Não	N/A	Não	Sim	Não	Sim
R	Não	Não	N/A	Não	Não	Não	Não

3 INCORPORANDO EVIDÊNCIAS OBTIDAS EM FASES ANTERIORES DA EXPLORAÇÃO DA *WEB* OCULTA

Dado que este trabalho está inserido no contexto da *web* oculta, ele se baseia na premissa de que é possível melhorar os resultados da eliminação de ruído fazendo uso de dados obtidos em fases anteriores da exploração da *web* oculta.

3.1 Coleta de dados da submissão de formulários da web oculta

Em busca da melhoria proposta, são coletadas outros dados a partir do trabalho de Kantorski et al. (2012). Estes novos conjuntos de dados servirão tanto para a realização de experimentos quanto para a coleta de evidências adicionais a serem utilizados com a abordagem proposta.

Como o único meio de acesso à *web* oculta se faz por meio do preenchimento de formulários, Kantorski et al. (2012) propõem um método automático para o preenchimento de tais formulários. O método mais simples de preenchimento seria apenas combinar todos os valores possíveis de preenchimento, fazendo um produto cartesiano. Entretanto, o crescimento do tamanho do formulário e de suas opções tornam inviáveis a submissão de todas as combinações. Dessa forma, o trabalho busca a seleção de bons valores para o preenchimento dos campos, minimizando o número de submissões e maximizando o número de registros retornados do banco de dados.

A partir das submissões de formulários realizada em Kantorski et al. (2012), é possível coletar os rótulos e valores utilizados durante uma submissão, como podemos ver na Figura 3.1

Figura 3.1: Exemplo de *URL* obtida de Kantorski et al. (2012), com rótulos e valores.

A *URL* da Figura 3.1, retirada de um dos conjuntos de dados utilizados neste trabalho, indica que foi gerada uma requisição *HTTP* para a submissão de dados do formulário do domínio http :// www.drinkgenius.com/, com os rótulos item_name e description , e seus

respectivos valores chocolate e cherry. Para cada *dataset* diferente, composto por páginas da *web* oculta resultantes de diferentes submissões de um mesmo formulário para um mesmo domínio, coletou-se todos os rótulos e todos os valores de todas as submissões. Como as informações referentes a cada *dataset* pertencem um mesmo domínio, ou seja, a um mesmo *site web*, foi criado um grande conjunto com estes valores textuais extraídos, sendo que termos repetidos são descartados. Este conjunto de termos representa uma grande *query* de domínio, que pode ser utilizado para a busca de correspondências nas páginas resultantes, renderizadas dinamicamente. Essas informações servem como indicativos de que determinado trecho de uma página *web* é realmente conteúdo relevante e não ruído.

3.2 Criação de evidências adicionais

Ao percorrermos a árvore *DOM*, é verificada a existência de termos desta *query* em determinada sub-árvore. Assim, para cada nodo da árvore *DOM* da página *web* resultante, são criadas outras evidências baseadas nos dados previamente coletados. Uma destas evidências criadas a partir destes dados é a medida chamada de *QueryScore*. Ela representa o somatório do número de vezes que um termo da *query* é encontrado em algum dos nodos descendentes (TF), divido pela distância relativa. Essa distância é o número de níveis existentes na árvore *DOM* entre o nodo descendente que contém tal correspondência textual até o nodo ancestral o qual está sendo calculado. A fórmula abaixo demonstra esta medida:

$$QueryScore = \sum_{i \in C} \frac{correspondências_i}{distância\ relativa_i}$$

Esta medida é relevante para todos os nodos que contém não apenas texto, mas para nodos contendo texto e outras sub-árvores. Sua vantagem é poder contabilizar todas as correspondências na sub-árvore do nodo, resultando em valores mais altos para correspondências encontradas com profundidades baixas.

```
<body>
  <p>
    Nome do candidato aprovado
  </p>
  <table>
    <tr id="tr1">
      <td id="td1">
          Julia
      </td>
      <td id="td2">
          Marcelo
      </td>
    </tr>
    <tr id="tr2">
      <td id="td3">
          Silvia
      </td>
    </tr>
  </table>
</body>
```

O trecho fictício de uma página *HTML* mostrado acima, pode ser visualizado em forma de árvore *DOM*, onde sua representação seria equivalente à Figura 3.2.

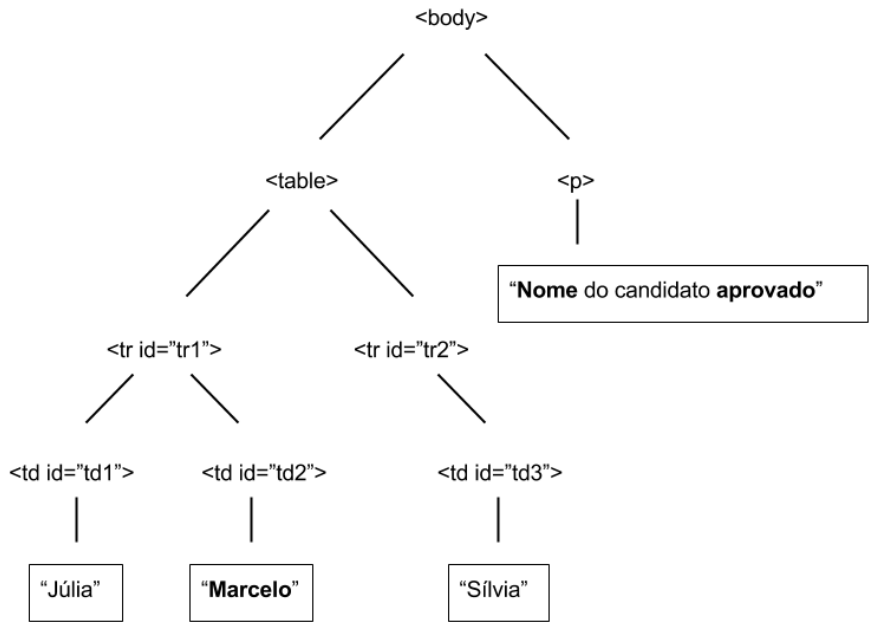

Figura 3.2: Árvore *DOM* de um trecho *HTML* exemplo.

Tabela 3.1: Termos de uma *query* de domínio exemplo

nome
Marcelo
Bruno
Amanda
aprovado
S
N

Baseado no código fictício *HTML*, e considerando um exemplo de *query* de domínio representada pelos termos constantes na Tabela 3.1, o *QueryScore* de cada nodo do trecho *HTML* pode ser calculado da seguinte maneira:

$$\textit{Elemento <body>} : QueryScore = (2/2) + (1/4) = 1.25$$

$$\textit{Elemento <p>} : QueryScore = (2/1) = 2$$

$$\textit{Elemento <table>} : QueryScore = (1/3) = 0.33$$

$$\textit{Elemento <tr id=“tr1”>} : QueryScore = (1/2) = 0.5$$

$$\textit{Elemento <td id=“td1”>} : QueryScore = 0$$

$$\textit{Elemento <td id=“td2”>} : QueryScore = (1/1) = 1$$

$$\textit{Elemento <tr id=“tr2”>} : QueryScore = 0$$

$$\textit{Elemento <td id=“td3”>} : QueryScore = 0$$

Também são utilizadas outras duas evidências. Diferentemente da *QueryScore*, que leva em consideração todo texto existente abaixo de um nodo, é adicionado o número de correspondências simples diretamente abaixo de um nodo. Esse valor levará em consideração apenas o texto deste nodo, e não textos encontrados em sub-árvores deste nodo. A outra evidência coletada é a profundidade absoluta do nodo em processamento e a *tag* **<body>** do documento.

3.3 Utilização de método de aprendizagem para combinação de evidências

A fim de poder agregar as medidas criadas no *baseline* (Densidade de Texto Composta e Soma das Densidades) e estas evidências adicionais, propõe-se neste trabalho a utilização de uma abordagem de aprendizagem de máquina. Empregando um classificador do tipo árvore de decisão, é possível classificar os nodos de texto em ruído ou não.

Um classificador possui duas fases (HAN; KAMBER; PEI, 2006), e funciona da seguinte maneira: na primeira fase, um classificador é construído baseado em um conjunto de classes (no caso deste trabalho, conteúdo ou ruído são as duas classes possíveis). O classificador baseia-se nas tuplas (cada nodo *HTML*) contendo vetores de atributos, e este grupo é chamado de conjunto de treinamento. Como a classe de cada tupla é previamente dada, o treinamento é chamado de “supervisionado”. Esse processo pode ser racionalizado como o mapeamento de uma função, onde dado os atributos como parâmetros, o resultado da função é uma das possíveis classes indicadas.

Após a fase de treinamento, um conjunto semelhante de tuplas, contendo um mesmo padrão de atributos e classes, é provido para uma fase de testes. Este conjunto de teste deve ser selecionado aleatoriamente, e durante este processo cada tupla deve ser testada contra o modelo do classificador. O resultado, então, deve ser verificado com o rótulo de classe existente na tupla.

O classificador selecionado para este trabalho é um classificador de árvore de decisão. O modelo gerado por este tipo de classificador é uma estrutura em árvore (HAN; KAMBER; PEI, 2006), onde cada nodo interno é um teste de um atributo, cada ramificação é um resultado de um teste, e cada nodo folha contém um rótulo de classe. Como os atributos utilizados neste trabalho representam grandezas, a árvore de decisão gerada não é binária.

Após a geração da árvore de decisão modelo, o conjunto de testes deve ser utilizado para verificação da precisão do classificador gerado. Dada uma tupla, no caso deste trabalho um nodo *HTML*, os atributos deste registro vão sendo testados contra os nodos da árvore gerada, até que um caminho da raiz até uma folha tenha sido construído. Como o modelo é supervisionado, testa-se então se o caminho leva ao mesmo rótulo existente na tupla.

Segundo Han, Kamber e Pei (2006), a popularidade dos classificadores de árvore de decisão provém do fato da construção destes modelos não exigirem conhecimento do domínio, sendo apropriados para descoberta de conhecimento exploratória. Também é característica destes modelos poder lidar com dados multidimensionais, ter uma construção simples e rápida, além de possuir uma boa acurácia. Estes classificadores são utilizados

Figura 3.3: Exemplo de arquivo utilizado como entrada para o algoritmo de classificação de árvore de decisão, onde cada linha representa um nodo da árvore DOM e seus atributos calculados a partir das medidas propostas no *baseline* e medidas propostas neste trabalho.

```
ctd,density_sum,query_score,matches,distance,result
20.80,23.77,14.1,6.0,3.0,S
28.88,32.51,8.0,32.0,6.0,N
16.20,39.56,7.88,8.0,5.0,S
0.0,8.62,3.0,12.0,6.0,S
16.20,39.56,7.88,8.0,5.0,S
48.47,37.57,9.0,36.0,6.0,S
25.27,59.15,8.33,8.0,5.0,S
0.0,8.62,3.0,12.0,6.0,S
```

em diversas áreas, já que a única coisa que exigem são uma boa qualidade dos dados à disposição.

3.4 Execução do algoritmo de classificação

Desta forma, para cada conjunto de dados é criado um arquivo contendo 6 atributos:

- Densidade de Texto Composta
- Soma das Densidades
- *QueryScore*
- Correspondências simples
- Distância absoluta
- Rótulo de classe (ruído ou não)

Neste mesmo arquivo, foram agrupados os valores de todas as páginas *web* resultantes de submissões, já que todas as páginas pertencem ao mesmo domínio. A rotulação de cada nodo, isto é, a indicação se o mesmo é ruído ou conteúdo (representado pela última coluna do arquivo), é fruto da avaliação manual do conteúdo existente em cada página *web* do *dataset*. Um trecho exemplo do arquivo gerado com os atributos mencionados pode ser visualizado na Figura 3.3, onde o valor de cada coluna é separado por vírgulas, e o último atributo indica se o nodo é ruído (S) ou conteúdo (N).

Na Figura 3.4, podemos ver um trecho exemplo da árvore gerada pelo classificador de árvore de decisão deste trabalho. Para isto, foi utilizado o *software* de mineração de dados Weka (HALL et al., 2009), com uma implementação do algoritmo C4.5 em Java (J48). Como neste trabalho todos os atributos de uma tupla são numéricos e não discretos, as operações de comparação em cada nodo intermediário são de grandeza ou igualdade.

O resultado do teste é indicado pelo software, indicando a precisão e revocação do teste. O software *Weka* também provê como resultado a matriz de confusão da execução do teste, possibilitando a verificação das grandezas de valores de falsos positivos e falsos negativos.

Figura 3.4: Exemplo de árvore de decisão gerada pelo *software* Weka, mostrando as decisões em cada nodo conforme os valores dos atributos.

```
query_score <= 4
|   cetd <= 36.819934
|   |   cetd <= 36.785629
|   |   |   cetd <= 4.970103: S (12077.0/795.0)
|   |   |   cetd > 4.970103
|   |   |   |   density_sum <= 2.151315: S (123.0/8.0)
|   |   |   |   density_sum > 2.151315
|   |   |   |   |   cetd <= 11.552928: N (13.0/1.0)
|   |   |   |   |   cetd > 11.552928: S (7154.0/11.0)
|   |   cetd > 36.785629: N (52.0/16.0)
|   cetd > 36.819934: S (3024.0/1.0)
query_score > 4: S (23804.0)
```

4 EXPERIMENTOS

Para demonstrar que a utilização de termos de domínio coletados durante a submissão de formulários de acesso a páginas da *web* oculta, pode melhorar o resultado da identificação de ruído em páginas resultantes da submissão destes formulários, uma série de experimentos foi realizada.

Neste capítulo, serão descritas as métricas utilizadas para verificação dos resultados da descoberta do ruído. Também serão apresentados os resultados da aplicação da técnica implementada como *baseline* com *datasets* da *web* oculta, assim como os experimentos com uma abordagem supervisionada para melhora destes resultados.

4.1 Métricas para avaliação

Para avaliar os resultados da implementação do *baseline*, foi utilizado um subconjunto das métricas contidas em Sun, Song e Liao (2011), entre elas a Precisão, Revocação, e F1. Para o cálculo destas medidas, o conteúdo textual extraído após a execução dos experimentos é comparado ao conteúdo correto, ou seja, os trechos de texto que comprovadamente representam conteúdo. Com o intuito de comparar estas duas sequências de textos, o método utilizado para comparação textual é encontrar a subsequência comum mais longa de ambos os resultados ($LCS(a, b)$).

4.1.1 Subsequência comum mais longa

Leiserson et al. (2001) define como objetivo de comparação de duas sequências a determinação do grau de semelhança entre ambas. Este grau de semelhança pode ser definido de diferentes maneiras. Uma das maneiras seria a verificação se uma sequência é subcadeia da outra. Outra forma seria verificar o número de alterações necessárias para transformar uma sequência em outra, realizando assim um mapeamentro entre as sequências.

Neste trabalho, é assumida como medida de semelhança a busca por uma terceira sequência, onde os termos desta terceira sequência apareçam nas duas primeiras, na mesma ordem, mas não necessariamente consecutivamente. Quanto maior for o tamanho desta terceira sequência, maior será a semelhança entre as duas primeiras.

Além de definir o problema de subsequência comum mais longa como a busca por uma subsequência comum de comprimento máximo entre duas cadeias, Leiserson et al. (2001) mostra a resolução deste problema de diferentes formas. Os dois primeiros métodos são por força bruta, e por uma solução recursiva.

Apesar de ser possível a resolução com os métodos citados, o fato do problema apresentar um número limitado de subproblemas distintos permite que seja utilizada progra-

mação dinâmica para o cálculo de soluções. Leiserson et al. (2001) define então o algoritmo $LCS - LENGTH$, com a tomada de duas sequências $X =< x_1, x_2, \ldots x_m >$, $Y =< y_1, y_2, \ldots y_n >$ como entrada do algoritmo. O algoritmo abaixo utiliza duas tabelas, b e c, para o cálculo de soluções intermediárias. No fim, o algoritmo retorna estas tabelas, sendo que o elemento $c(m, n)$ contém o comprimento de uma LCS entre X e Y.

```
LCS-LENGTH(X, Y)
 1  m = X.length
 2  n = Y.length
 3  let b[1..m, 1..n] and c[0..m, 0..n] be new tables
 4  for i = 1 to m
 5      c[i, 0] = 0
 6  for j = 0 to n
 7      c[0, j] = 0
 8  for i = 1 to m
 9      for j = 1 to n
10          if x_i == y_j
11              c[i, j] = c[i − 1, j − 1] + 1
12              b[i, j] = "↖"
13          elseif c[i − 1, j] ≥ c[i, j − 1]
14              c[i, j] = c[i − 1, j]
15              b[i, j] = "↑"
16          else c[i, j] = c[i, j − 1]
17              b[i, j] = "←"
18  return c and b
```

Figura 4.1: Algoritmo para descoberta de subsequência comum mais longa, retirado de Leiserson et al. (2001).

Segundo Leiserson et al. (2001), o tempo de execução do algoritmo é $O(mn)$, visto que cada entrada de tabela leva $O(1)$ para ser calculada.

4.1.2 Precisão, revocação e F1

A comparação ocorre palavra a palavra, ou seja, os tamanhos são referentes ao número de palavras encontradas no conjunto de texto extraído e no conteúdo avaliado.

Para o cálculo da precisão, deve-se encontrar a proporção entre:

- O tamanho da substring comum mais longa entre o conteúdo textual extraído após os experimentos, e o conteúdo correto (manualmente avaliado)
- O tamanho do conteúdo extraído pelos experimentos

Assim, a precisão, que traz a fração de termos relevantes extraídos, pode ser representada da seguinte maneira (MANNING; RAGHAVAN; SCHÜTZE, 2008):

$$\textit{Precisão} = \frac{LCS(a, b).tamanho}{a.tamanho}$$

Para calcular a revocação, busca-se a proporção entre:

- O tamanho da substring comum mais longa entre o conteúdo textual extraído após os experimentos, e o conteúdo correto (manualmente avaliado)

- O tamanho do conteúdo correto, avaliado manualmente

A revocação, que traz a fração de termos relevantes que são extraídos, é representada da forma abaixo (MANNING; RAGHAVAN; SCHÜTZE, 2008):

$$\textit{Revocação} = \frac{LCS(a,b).tamanho}{b.tamanho}$$

Após o cálculo das métricas mencionadas, além de ser possível calcular a precisão média e revocação média para cada conjunto de dados, é possível calcular a medida F1, que agrega os resultados de precisão e revocação em um mesmo valor utilizando a média harmônica ponderada entre ambos os valores (MANNING; RAGHAVAN; SCHÜTZE, 2008). O cálculo é realizado da seguinte forma:

$$F_1 = \frac{2 \times P \times R}{P + R}$$

4.2 Experimentos realizados com *datasets* do *baseline*

Para validar a hipótese de que o uso de informações da *web* oculta pode auxiliar a identificação de ruído em páginas *web*, implementou-se a solução proposta em Sun, Song e Liao (2011). Primeiramente, utilizou-se os conjuntos de dados fornecidos pelos autores da técnica em seu trabalho. O conteúdo das páginas deste conjunto foi extraído e avaliado manualmente pelos autores do *baseline*, e todos estes dados foram coletados do *website* dos autores.

4.2.1 Descrição dos *datasets* do *baseline*

Em Sun, Song e Liao (2011) são utilizadas duas fontes de dados diferentes. A primeira fonte de dados é formada por 5 conjuntos de páginas *web*, cada um contendo 100 páginas:

- *Ars Technica* (Ars Technica, 2012)

- *BBC* (BBC, 2012)

- *Yahoo* (Yahoo, 2012)

- *New York Times* (New York Times, 2012)

- *Wikipedia* (Wikipedia, 2012)

Além destes cinco, ainda há um sexto elemento contendo 200 páginas aleatórias, chamado *Chaos*. O conteúdo dessas páginas foi extraído e avaliado manualmente pelos autores da técnica.

A segunda fonte de dados é a da *CleanEval*, uma avaliação competitiva de limpeza de páginas *web* (BARONI et al., 2008). As páginas existentes já possuem o conteúdo extraído corretamente para avaliação, permitindo que se possa utilizá-lo para testes dos algoritmos.

Tabela 4.1: Comparação entre os resultados descritos em Sun, Song e Liao (2011) (BL) e da implementação do *baseline* (IMPL-BL)

Fonte de dados	P-BL	P-IMPL-BL	R-BL	R-IMPL-BL	$F_1 - BL$	$F_1 - IMPL - BL$
Ars Technica	98.04%	96.87%	99.51%	98.36%	98.76%	97.61%
BBC	86.15%	83.55%	97.95%	89.25%	91.67%	86.30%
Chaos	96.21%	92.08%	96.10%	92.35%	96.15%	92.22%
New York Times	99.69%	94.91%	98.16%	95.31%	98.92%	95.11%
Wikipedia	98.25%	97.46%	92.77%	91.44%	95.43%	94.36%
Yahoo	84.59%	83.83%	93.99%	83.88%	89.04%	83.86%
CleanEval	95.87%	93.60%	97.15%	92.59%	96.51%	93.10%

4.2.2 Resultados da aplicação da técnica *baseline* com seus *datasets*

Testando a implementação do *baseline* com os conjuntos de dados dos autores, obteve-se resultados semelhantes aos originais descritos no artigo *baseline*. Os resultados podem ser visualizados na Tabela 4.1.

É possível visualizar através da tabela que existe uma leve diferença entre os resultados obtidos pelo autor da técnica e os resultados encontrados após a implementação do *baseline*. Esta diferença se deve a uma dificuldade encontrada no momento da implementação: o método não leva em conta elementos ocultos nas páginas *web*, eles devem ser removidos antes de fazer a extração de conteúdo. Entretanto, muitos destes elementos são ocultos através de elementos *display:none* ou *visibility:hidden* em folhas de estilo *CSS* (*Cascade Style Sheet*), e definidos em arquivos externos à página *web*. Desta forma, a página faz referência ao seu estilo através de uma classe, que é computada em tempo de renderização pelo *browser*. O *framework* utilizado para o *parsing* do *HTML* das páginas não consegue prever isto, portanto estes elementos ocultos acabaram sendo computados em nossa implementação, diminuindo o valor dos resultados encontrados.

Outro problema é que foram encontrados alguns dados incorretos nas fontes de dados dos autores, já que resultados intermediários incorretos, que deveriam ser extraídos como conteúdo, não aparecem nos arquivos resultantes dos autores.

4.3 Experimentos realizados com *datasets* coletados da *web* oculta

4.3.1 Detalhamento dos *datasets* de Kantorski et al. (2012)

Introduziu-se então neste trabalho outros conjuntos de dados, um grupo restrito dos dados utilizados em Kantorski et al. (2012). Os conjuntos de dados utilizados são:

- *e4s* (E4S, 2013), uma ferramenta de busca de empregos para estudantes
- *missing* (Missing Children, 2013), um formulário para busca de pessoas desaparecidas
- *pubs* (BITE, 2013), um motor de busca de bares
- *drinkgenius* (DrinkGenius, 2013), um formulário para procura de receitas e drinques
- *onlinerace* (Online Race Results, 2013), uma busca de resultados de corridas de carro
- *posteritati* (Posteritati, 2013), uma ferramenta para procura de cartazes de filmes

- *book* (Bolen Books, 2013), uma busca de livros à venda
- *career* (Careerbuilder.com, 2013), uma ferramenta para procura de empregos
- *mldb* (MLDb, 2013), uma busca em um banco de dados de música e letras

Nestes formulários de entrada para a *web* oculta, diversas combinações possíveis de preenchimento são realizadas, já que cada preenchimento leva a uma página diferente, resultante de uma consulta em um banco de dados. Após a submissão de cada uma destas combinações, foi possível executar experimentos com o código *HTML* resultante.

Como estes dados não estavam avaliados, selecionou-se um grupo de páginas resultantes após a submissão dos formulários para a avaliação manual, definindo quais elementos fazem parte do conteúdo e quais são ruído. Dado o grande número de páginas a serem avaliadas manualmente, foi necessário realizar uma amostragem deste conjunto. Para isto, utilizou-se um intervalo de confiança de 5 e um grau de confiança de 95%. O conteúdo correto foi então extraído e armazenado em um arquivo separado, para os testes a seguir. A Tabela 4.2 mostra o detalhamento deste conjunto de dados, contendo a quantidade de páginas utilizadas na avaliação.

Tabela 4.2: Detalhamento dos novos conjuntos de dados

Nome	Descrição	Quantidade avaliada manualmente
e4s	Ferramenta de busca de empregos para estudantes	239
missing	Formulário para busca de pessoas desaparecidas	345
pubs	Busca de bares	340
drinkgenius	Formulário para procura de receitas e drinques	291
onlinerace	Busca de resultados de corridas de carro	156
posteritati	Ferramenta para procura de cartazes de filmes	261
book	Busca de livros à venda	395
career	Procura de empregos	196
mldb	Busca em um banco de dados de música e letras	294

4.3.2 Resultados da aplicação da implementação do *baseline* aos *datasets* da *web* oculta

Após a submissão das *URLs* e seus parâmetros obtidas dos *datasets* de Kantorski et al. (2012), foram coletadas todas as páginas *HTML* resultantes. Para cada uma delas foi executado o algoritmo de remoção de ruído do *baseline*.

Em seguida, os resultados foram comparados com os arquivos contendo a avaliação manual destas páginas, e as métricas apropriadas foram calculadas. O resultado pode ser visualizado na Tabela 4.3.

4.3.3 Execução dos experimentos utilizando abordagem supervisionada

Buscando agregar as medidas existentes em Sun, Song e Liao (2011) com as evidências adicionais coletadas a partir do preenchimento dos formulários de acesso à *web* oculta, optou-se por utilizar uma abordagem de aprendizagem de máquina.

Assim, para cada *dataset* existente, foram reunidas todas as páginas *HTML* resultantes da submissão de formulários, e que tiveram seu conteúdo avaliado manualmente. Foi criado então um arquivo, contendo todos os nodos da árvore *DOM* de todas estas páginas do *dataset*. No arquivo, cada registro corresponde a um nodo da árvore *DOM*, e cada registro contém os seguintes atributos:

Tabela 4.3: Resultados do *baseline* com conjunto de dados da *web* oculta

Conjunto de dados	Precisão	Revocação	F1
e4s	38.09%	47.75%	42.37%
missing	60.28%	90.42%	72.33%
pubs	23.34%	56.64%	33.05%
drinkgenius	41.82%	65.32%	50.99%
onlinerace	25.33%	17.06%	20.38%
posteritati	00.34%	23.46%	0.67%
book	89.05%	88.97%	89.00%
career	56.85%	83.55%	67.66%
mldb	14.91%	36.53%	21.17%

- Densidade de Texto Composta
- Soma das densidades
- *QueryScore*, que leva em consideração o número de correspondências entre termos da *query* e termos abaixo de um nodo, junto com todos seus descendentes e a distância relativa ao nodo sendo calculado.
- Número de correspondências entre a *query* e termos diretamente abaixo do nodo sendo analisado
- Distância absoluta do nodo da árvore até o elemento **<body>**
- Rótulo de classe, indicando se o nodo é ruído ou não.

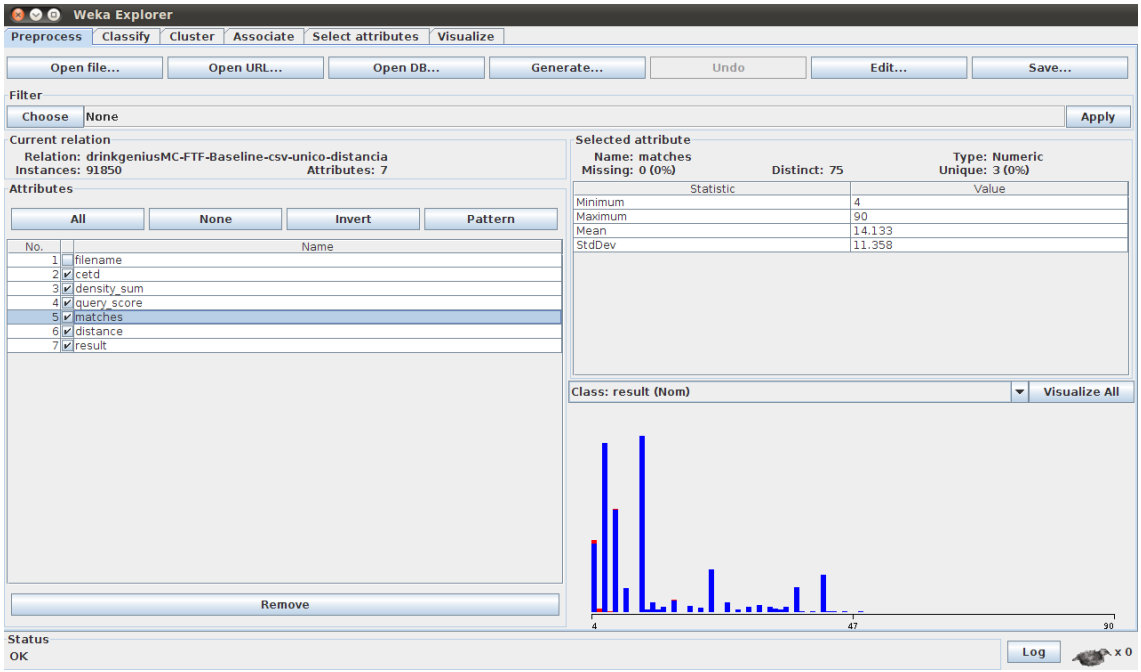

Figura 4.2: Tela de pré-processamento dos dados do *software* de mineração de dados Weka.

Com essas informações agrupadas, é possível utilizar o *software* de mineração Weka (HALL et al., 2009) para a classificação dos nodos. Na Figura 4.2, pode-se ver a tela

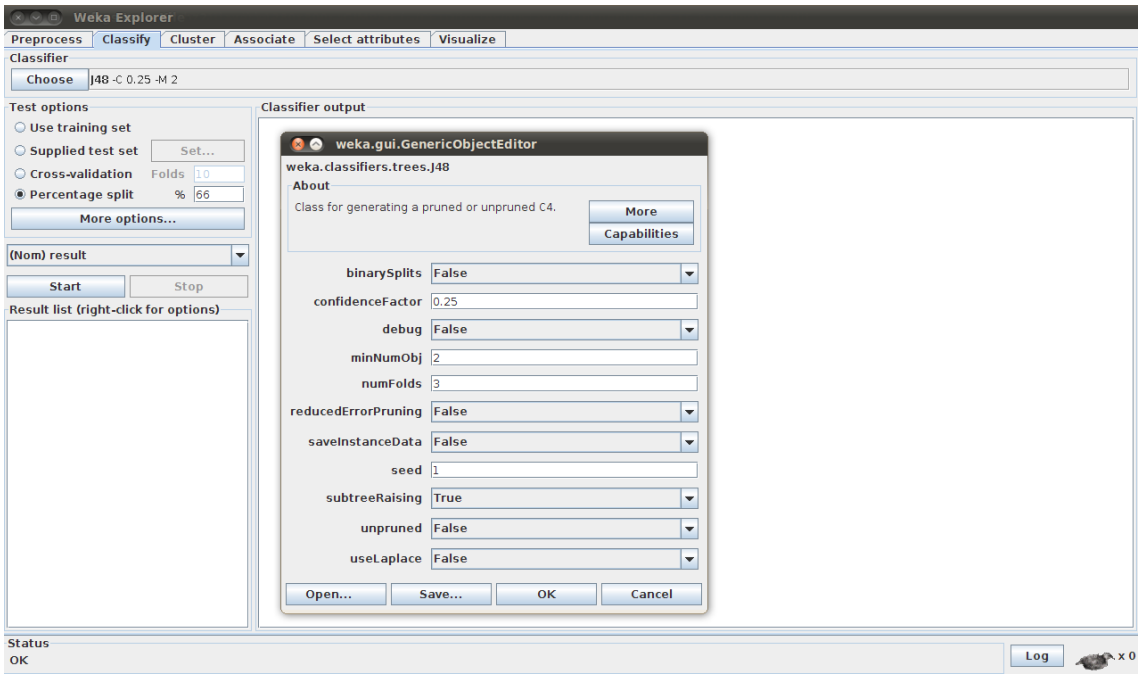

Figura 4.3: Tela de parametrização dos algoritmos de classificação do software de mineração de dados Weka.

de entrada de dados do Weka. Neste momento, podem ser determinados quais atributos participarão da tarefa de classificação, além de permitir a realização de tarefa de limpeza dos dados. Estatísticas sobre os dados contidos no *dataset* também podem ser observadas.

Em um segundo momento, deve ser escolhido e parametrizado o algoritmo de classificação. A Figura 4.3 mostra este procedimento. Para este trabalho, foi utilizado o algoritmo de classificação de árvore de decisão J48 (implentação Java do algoritmo de classificação C4.5). A figura também mostra a escolha deste trabalho em relação à divisão dos nodos em conjuntos de treinamento e de teste. A escolha foi por dividir os nodos aleatóriamente, utilizando 66% dos mesmos para a fase de treinemento do algoritmo, e o restante sendo utilizado para a fase de testes. Como existiam muito mais nodos ruído, o resultado da fase de teste é uma média ponderada dos resultados de cada categoria (ruído ou não), levando-se em consideração os resultados de precisão e revocação.

Por fim, após o término do procedimento de classificação, o software Weka mostra a árvore de decisão criada, como na Figura 4.4. Resultados como precisão, revocação, matriz de confusão e outros, também são informados pelo software.

4.3.4 Resultados da aplicação da abordagem de aprendizagem supervisionada para *datasets* da *web* oculta

Após a execução da abordagem supervisionada com o *software* Weka para todos os *datasets* de Kantorski et al. (2012), pode-se visualizar os resultados obtidos na Tabela 4.4.

4.3.5 Discussão dos resultados

Após a realização dos experimentos descritos, pode-se comparar o resultado da simples aplicação da solução desenvolvida no *baseline* às páginas resultantes da submissão de formulários, com os resultados obtidos após a utilização de informações de preenchimento dos formulários. Os gráficos de comparação das medidas de precisão e revocação,

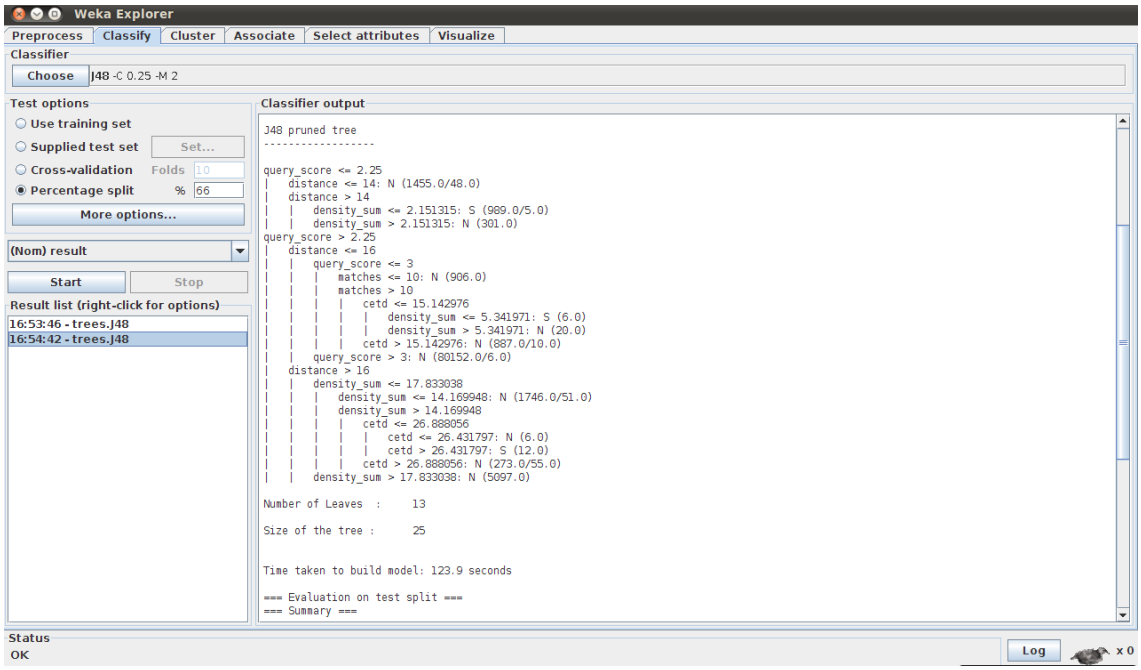

Figura 4.4: Tela com árvore de decisão resultante da aplicação do algoritmo de classificação J48 a um *dataset*.

Tabela 4.4: Resultados da abordagem supervisionada (TP) com conjunto de dados da *web* oculta

Conjunto de dados	Precisão-TP	Revocação-TP	F1-TP
e4s	98.3%	98.4%	98.3%
missing	99.7%	99.7%	99.6%
pubs	99.8%	99.8%	99.8%
drinkgenius	99.8%	99.8%	99.8%
onlinerace	99.2%	99.2%	99.2%
posteritati	99.7%	99.7%	99.6%
book	97.5%	98.3%	97.5%
career	98.7%	98.5%	98.5%
mldb	96.6%	95.6%	95.9%

gerados a partir das Tabelas 4.3 e 4.4, podem ser visualizados nas figuras 4.5 e 4.6.

Além de ficar claro que as informações obtidas durante o preenchimento auxiliam na identificação do ruído, a discrepância também resulta de diferentes características dos *datasets*. Nos *datasets* utilizados por Sun, Song e Liao (2011), grande parte das páginas *HTML* são provenientes de *websites* de notícias. Isso faz com que grande parte do conteúdo de interesse estivesse agrupado em uma região única da árvore. No momento em que a técnica proposta no *baseline* foi aplicada às páginas da *web* oculta, o resultado decaiu drasticamente. Isso se deve possivelmente ao formato dos dados que, neste tipo de página, são tabulares e não possuem informações relevantes tão concentradas quanto em *websites* de notícias.

Observando-se a matriz de confusão resultante do algoritmo de classificação, pode-se perceber que os resultados da detecção de ruído foram positivos, já que foi possível classificar nodos ruidosos com altos valores de precisão e revocação. Todos os conjuntos de dados obtiveram valores de precisão e revocação acima de 95%. Na Tabela 4.5, podem ser observadas quatro colunas: a primeira indica nodos ruidosos que foram acerta-

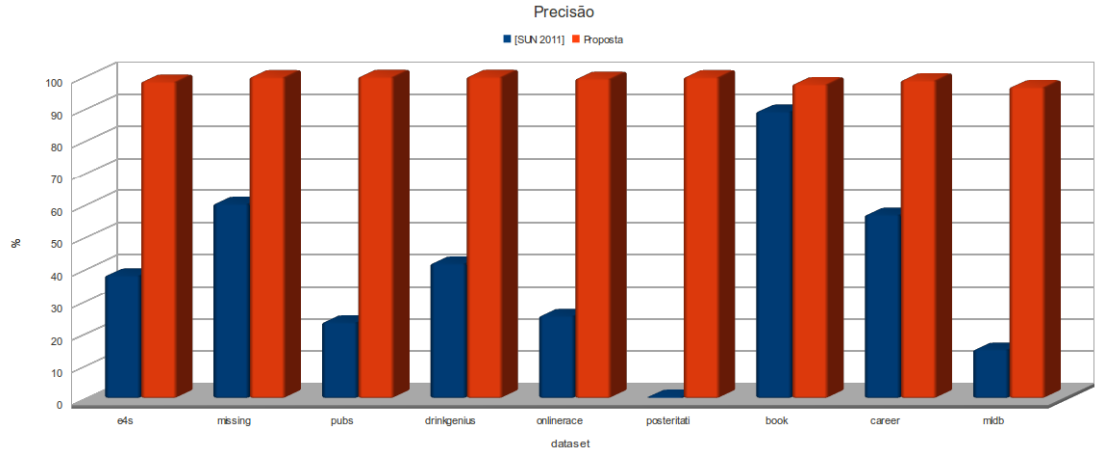

Figura 4.5: Comparação dos resultados de precisão entre o *baseline* e a técnica proposta (aplicados aos *datasets* de Kantorski et al. (2012)).

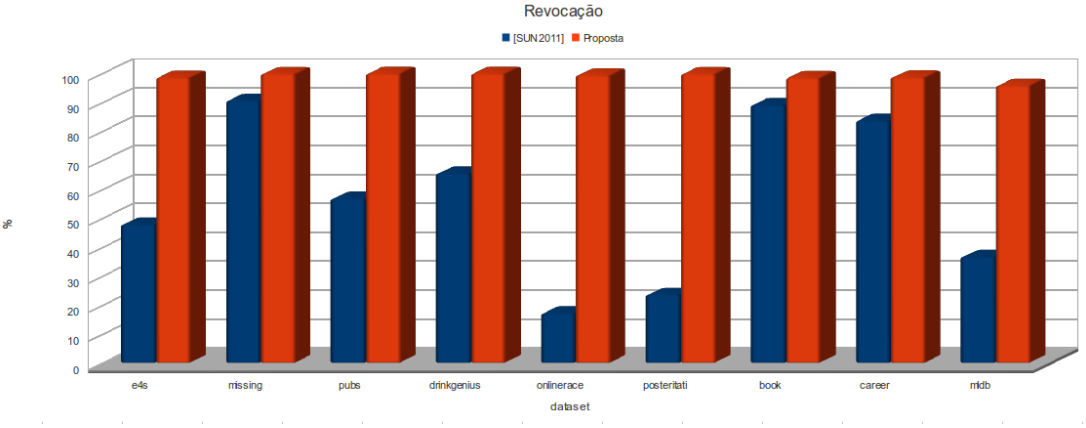

Figura 4.6: Comparação dos resultados de revocação entre o *baseline* e a técnica proposta (aplicados aos *datasets* de Kantorski et al. (2012)).

damente classificados como ruidosos; a segunda, nodos ruidosos que foram classificados como conteúdo; a terceira, nodos de conteúdo relevante que foram corretamente classificados como conteúdo; e por fim, a quarta coluna indica nodos de conteúdo que foram classificados erroneamente como ruído.

A matriz de confusão mostra que a técnica desenvolvida atinge bons resultados para detectar ruído, mas deixa a desejar para classificar nodos de conteúdo em si. Isto acontece devido ao fato de existir uma quantidade muito maior de nodos ruidosos em cada página, o que, além de atrapalhar o algoritmo de classificação, acaba corroborando as estimativas de Gibson, Punera e Tomkins (2005) em relação à quantidade de ruído existente na *web*.

Observando-se os resultados, pode-se ver pequenas diferenças entre os resultados para os diferentes conjuntos de dados. Essas diferenças podem ser explicadas pelo padrão de nomeação dos rótulos utilizados nos elementos de formulário das páginas *HTML* de cada *dataset*. Enquanto alguns *datasets* contêm rótulos com valores "humanos", outros possuem apenas códigos gerados por algum *framework*. Essa diferença faz com que alguns *datasets* possuam uma *query* de domínio maior e mais significativa do que outros, auxiliando no processo de detecção do ruído.

Tabela 4.5: Matriz de confusão

Conjunto	RR	RC	CC	CR
e4s	44488	147	915	592
missing	30576	0	27	104
pubs	15946	0	6	28
drinkgenius	30835	2	331	61
onlinerace	62304	172	255	306
posteritati	31005	3	46	96
book	15438	9	11	266
career	119437	1563	4076	370
mldb	14789	632	1281	109

5 CONCLUSÃO

Neste trabalho, aborda-se o problema da identificação de ruído em páginas *web*. Este ruído, composto por todos os tipos de elementos não informativos ao usuário final, pode afetar negativamente o desempenho de motores de busca, índices *web*, e mineração de dados, como classificação e clusterização. A eliminação deste ruído ainda pode ser útil para a construção de corpora de texto, e para criar uma melhor exibição de páginas *web* em dispositivos com tela reduzida. Diversos trabalhos concentraram esforços neste sentido, mas muitos buscam por uma estrutura específica da página *web* para a extração do conteúdo ou eliminação do ruído.

Implementando a técnica descrita em Sun, Song e Liao (2011), observa-se que esta não obtém bons resultados para páginas encontradas na *web* oculta, a parte da *web* que é acessível apenas através do preenchimento de formulários. Isto se deve ao fato da estrutura das páginas resultantes da submissão de formulários ser um formato tabular, já que as páginas são renderizadas dinamicamente com o resultado de uma consulta a um banco de dados.

Propõe-se neste trabalho a incorporação de diferentes medidas às já existentes em Sun, Song e Liao (2011), utilizando dados obtidos a partir da segunda fase da exploração da *web* oculta. É possível criar evidências adicionais a partir dos rótulos e valores de preenchimento dos formulários que dão acesso às páginas da *web* oculta.

Com estas outras evidências, é possível realizar a combinação destes novos atributos com as métricas descritas no *baseline*. Todas essas informações servem como entrada para um algoritmo de aprendizagem de máquina. Escolheu-se um classificador de árvore de decisão, que é capaz de classificar registros em diferentes classes conforme os valores de seus atributos.

Os resultados mostrados evidenciam que a aplicação de um método de aprendizagem gera uma grande melhora dos resultados para as páginas da *web* oculta, resultantes da submissão de formulários. Com essa abordagem supervisionada, o algoritmo de classificação pode obter resultados médios de precisão e revocação acima de 95%, valores muito superiores aos obtidos da simples aplicação do *baseline* a estes dados. Como resultado deste estudo, um artigo foi publicado no Simpósio Brasileiro de Banco de Dados 2013 (LUTZ; HEUSER, 2013).

5.1 Trabalhos Futuros

Após os experimentos executados este trabalho, é possível identificar espaço para diversas melhorias nos resultados. Para atingir este objetivo, seria possível testar outros algoritmos de classificação com múltiplas escolhas de parâmetros, levando a um melhor

entendimento das mudanças dessas propriedades na descoberta do ruído.

Além disso, pode ser desenvolvido algum *parser* para buscar os rótulos dos nomes que são visíveis na página web, ao invés dos nomes *HTML* dos rótulos utilizados como parâmetros na submissão. Isso daria ainda mais valor aos termos existentes na *query* de domínio de cada *dataset*. Buscar outras fontes de dados da *web* oculta também se torna interessante, visto que isto diminuiria a dependência dos resultados aos dados obtidos de Kantorski et al. (2012).

Ainda poderiam ser explorados outros tipos de conjuntos de dados, não apenas aqueles resultantes de submissão de formulários. Novas evidências, seja da própria página *HTML* ou de formulários da *web* oculta, também poderiam ser criados, aumentando o número de atributos como entrada para o classificador. Todas estas experimentações são vitais para a descoberta de novas possibilidades em relação ao tema abordado.

REFERÊNCIAS

Ars Technica. **Ars Technica**. Disponível em http://www.arstechnica.com/. Data de acesso: Outubro de 2012.

BAR-YOSSEF, Z.; RAJAGOPALAN, S. Template detection via data mining and its applications. In: WORLD WIDE WEB, 11., New York, NY, USA. **Proceedings...** ACM, 2002. p.580–591. (WWW '02).

BARONI, M. et al. Cleaneval: a competition for cleaning web pages. In: LREC. **Proceedings...** [S.l.: s.n.], 2008.

BBC. **BBC - Homepage**. Disponível em http://www.bbc.co.uk/. Data de acesso: Outubro de 2012.

BERGMAN, M. K. The Deep Web: surfacing hidden value. **JEP: The Journal of Electronic Publishing**, [S.l.], v.7, n.1, 2001.

BITE. **BITE pubs and bars - beerintheevening.com**. Disponível em http://www.beerintheevening.com/pubs/. Data de acesso: Março de 2013.

Bolen Books. **Bolen Books Search**. Disponível em http://www.islandnet.com/bolen/search.php. Data de acesso: Março de 2013.

BURGET, R.; RUDOLFOVA, I. Web Page Element Classification Based on Visual Features. In: INTELLIGENT INFORMATION AND DATABASE SYSTEMS, 2009. ACIIDS 2009. FIRST ASIAN CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2009. p.67 –72.

CAI, D. et al. **1 VIPS**: a vision-based page segmentation algorithm. 2003.

Careerbuilder.com. **Jobs & Job Search Advice, Employment & Careers**. Disponível em http://jobs.careerbuilder.com/Jobseeker/Jobs/JobResults.aspx. Data de acesso: Março de 2013.

CHANG, C.-H. et al. A survey of web information extraction systems. **Knowledge and Data Engineering, IEEE Transactions on**, [S.l.], v.18, n.10, p.1411–1428, 2006.

CHEN, L.; YE, S.; LI, X. Template detection for large scale search engines. In: ACM SYMPOSIUM ON APPLIED COMPUTING, 2006., New York, NY, USA. **Proceedings...** ACM, 2006. p.1094–1098. (SAC '06).

DEBNATH, S.; MITRA, P.; GILES, C. L. Automatic extraction of informative blocks from webpages. In: ACM SYMPOSIUM ON APPLIED COMPUTING, 2005., New York, NY, USA. **Proceedings...** ACM, 2005. p.1722–1726. (SAC '05).

DrinkGenius. **Drink recipes database with instant search results and drinks of celebrities**. Disponível em http://www.drinkgenius.com/advanced-search.php. Data de acesso: Março de 2013.

E4S. **Student Jobs, Part Time Jobs, Temporary Jobs, Internships & Summer Jobs - E4S**. Disponível em http://www.e4s.co.uk/search/jobresult. Data de acesso: Março de 2013.

FERNANDES, D. et al. Computing block importance for searching on web sites. In: ACM CONFERENCE ON CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, New York, NY, USA. **Proceedings...** ACM, 2007. p.165–174. (CIKM '07).

GIBSON, D.; PUNERA, K.; TOMKINS, A. The volume and evolution of web page templates. In: SPECIAL INTEREST TRACKS AND POSTERS OF THE 14TH INTERNATIONAL CONFERENCE ON WORLD WIDE WEB, New York, NY, USA. **Proceedings...** ACM, 2005. p.830–839. (WWW '05).

HALL, M. et al. The WEKA data mining software: an update. **SIGKDD Explor. Newsl.**, New York, NY, USA, v.11, n.1, p.10–18, Nov. 2009.

HAN, J.; KAMBER, M.; PEI, J. **Data mining**: concepts and techniques. [S.l.]: Morgan kaufmann, 2006.

HE, B. et al. Accessing the deep web. **Commun. ACM**, New York, NY, USA, v.50, n.5, p.94–101, May 2007.

JIANG, L. et al. Learning Deep Web Crawling with Diverse Features. In: WEB INTELLIGENCE AND INTELLIGENT AGENT TECHNOLOGIES, 2009. WI-IAT '09. IEEE/WIC/ACM INTERNATIONAL JOINT CONFERENCES ON. **Proceedings...** [S.l.: s.n.], 2009. v.1, p.572–575.

KANTORSKI, G. Z. et al. Choosing Values for Text Fields in Web Forms. In: ADBIS (2). **Anais...** [S.l.: s.n.], 2012. p.125–136.

KOHLSCHüTTER, C.; FANKHAUSER, P.; NEJDL, W. Boilerplate detection using shallow text features. In: ACM INTERNATIONAL CONFERENCE ON WEB SEARCH AND DATA MINING, New York, NY, USA. **Proceedings...** ACM, 2010. p.441–450. (WSDM '10).

KOVACEVIC, M. et al. Recognition of common areas in a Web page using visual information: a possible application in a page classification. In: DATA MINING, 2002. ICDM 2003. PROCEEDINGS. 2002 IEEE INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2002. p.250 – 257.

KUSHMERICK, N. Learning to remove Internet advertisements. In: AUTONOMOUS AGENTS, New York, NY, USA. **Proceedings...** ACM, 1999. p.175–181. (AGENTS '99).

LAENDER, A. H. et al. A brief survey of web data extraction tools. **ACM Sigmod Record**, [S.l.], v.31, n.2, p.84–93, 2002.

LEISERSON, C. E. et al. **Introduction to algorithms**. [S.l.]: The MIT press, 2001.

LI, J.; EZEIFE, C. Cleaning Web Pages for Effective Web Content Mining. In: BRESSAN, S.; KNG, J.; WAGNER, R. (Ed.). **Database and Expert Systems Applications**. [S.l.]: Springer Berlin / Heidelberg, 2006. p.560–571. (Lecture Notes in Computer Science, v.4080).

LIN, S.-H.; HO, J.-M. Discovering informative content blocks from Web documents. In: ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, New York, NY, USA. **Proceedings...** ACM, 2002. p.588–593. (KDD '02).

LUTZ, J. A. F.; HEUSER, C. A. Descoberta de ruído em páginas da web oculta através de uma abordagem de aprendizagem supervisionada. In: SBBD, 2013. **Anais...** [S.l.: s.n.], 2013.

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. **Introduction to information retrieval**. [S.l.]: Cambridge University Press Cambridge, 2008. v.1.

Missing Children. **Missing Children Clearinghouse - Missing Children - California Dept. of Justice - Office of the Attorney General**. Disponível em http://dojapp.doj.ca.gov/missing/default.asp. Data de acesso: Março de 2013.

MLDb. **MLDb, the Music Lyrics Database - A huge song lyrics collection.** Disponível em http://www.mldb.org/search-bf. Data de acesso: Março de 2013.

New York Times. **The New York Times - Breaking News, World News & Multimedia**. Disponível em http://www.nytimes.com/. Data de acesso: Outubro de 2012.

Online Race Results. **Online Race Results**. Disponível em http://onlineraceresults.com/search/my_results.php. Data de acesso: Março de 2013.

Posteritati. **Posteritati - Original Movie Posters from the Silent Era to the Present**. Disponível em http://www.posteritati.com/search_results.php. Data de acesso: Março de 2013.

RAGHAVAN, S.; GARCIA-MOLINA, H. **Crawling the Hidden Web**. [S.l.]: Stanford InfoLab, 2000. Technical Report. (2000-36).

SONG, R. et al. Learning block importance models for web pages. In: WORLD WIDE WEB, 13., New York, NY, USA. **Proceedings...** ACM, 2004. p.203–211. (WWW '04).

SUN, F.; SONG, D.; LIAO, L. DOM based content extraction via text density. In: ACM SIGIR CONFERENCE ON RESEARCH AND DEVELOPMENT IN INFORMATION RETRIEVAL, 34., New York, NY, USA. **Proceedings...** ACM, 2011. p.245–254. (SIGIR '11).

VELLOSO, R. P.; DORNELES, C. F. Automatic Web Page Segmentation and Noise Removal for Structured Extraction using Tag Path Sequences. **JIDM**, [S.l.], v.4, n.3, p.173–187, 2013.

VIEIRA, K. et al. A fast and robust method for web page template detection and removal. In: ACM INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, 15., New York, NY, USA. **Proceedings...** ACM, 2006. p.258–267. (CIKM '06).

WANG, Y. et al. Incremental Web Page Template Detection by Text Segments. **Semantic Computing and Systems, IEEE International Workshop on**, Los Alamitos, CA, USA, v.0, p.174–180, 2008.

WENINGER, T.; HSU, W. H.; HAN, J. CETR: content extraction via tag ratios. In: WORLD WIDE WEB, 19., New York, NY, USA. **Proceedings...** ACM, 2010. p.971–980. (WWW '10).

Wikipedia. **Wikipedia**. Disponível em http://www.wikipedia.org/. Data de acesso: Outubro de 2012.

Yahoo. **Yahoo**. Disponível em http://www.yahoo.com. Data de acesso: Outubro de 2012.

YI, L.; LIU, B.; LI, X. Eliminating noisy information in Web pages for data mining. In: ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, New York, NY, USA. **Proceedings...** ACM, 2003. p.296–305. (KDD '03).