Renato Moraes Silva

CONTRIBUIÇÕES AO COMBATE DE *WEB SPAMMING*

Campinas
2013

Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Renato Moraes Silva

CONTRIBUIÇÕES AO COMBATE DE *WEB SPAMMING*

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.
Área de concentração: Automação.

Orientador: Prof. Dr. Akebo Yamakami
Coorientador: Prof. Dr. Tiago Agostinho de Almeida

Este exemplar corresponde à versão final da dissertação defendida pelo aluno Renato Moraes Silva e orientada pelo Prof. Dr. Akebo Yamakami

---

Campinas
2013

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Si38c

Silva, Renato Moraes, 1988-
Contribuições ao combate de web spamming / Renato Moraes Silva. --Campinas, SP: [s.n.], 2013.

Orientador: Akebo Yamakami
Coorientador: Tiago Agostinho de Almeida.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Spam (mensagens eletrônicas). 2. Aprendizado de máquina. 3. Reconhecimento de padrões. I. Yamakami, Akebo, 1947-. II. Almeida, Tiago Agostinho de. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Título em Inglês: Contributions to the battle against web spamming
Palavras-chave em Inglês: Spam (Electronic mail), Machine learning, Pattern recognition
Área de concentração: Automação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Sahudy Montenegro González, Romis Ribeiro de Faissol Attux
Data da defesa: 22-02-2013
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Renato Moraes Silva

Data da Defesa: 22 de fevereiro de 2013

Título da Tese: "Contribuições ao Combate de Web Spamming"

Prof. Dr. Akebo Yamakami (Presidente): _____

Profa. Dra. Sahudy Montenegro González: _____

Prof. Dr. Romis Ribeiro de Faissol Attux: _____

ao meus pais David e Maria e aos meus irmãos Leandro e Bruno

Agradecimentos

Agradeço,

à Deus por abençoar minha vida;

ao meu orientador e co-orientador, Profs. Akebo Yamakami e Tiago A. Almeida, que são pessoas que admiro e respeito, pela disponibilidade para orientações, sugestões, críticas construtivas e incentivos que foram muito valiosos para a realização desse trabalho e para a minha vida;

aos meus pais David e Maria pela educação, apoio, carinho, amor e incentivo que me deram em todas as etapas da minha vida;

aos meus irmãos Leandro e Bruno pelo apoio, pelo companheirismo e por serem sempre meus melhores amigos;

à minha namorada Simone pelo carinho, apoio, paciência e compreensão;

aos meus colegas do Departamento de Telemática pela ótima convivência e troca de experiências;

à minha orientadora do trabalho de conclusão de curso da graduação, prof.ª Tatiana A. Pazeto, pelo apoio, ensinamentos e troca de experiências;

aos professores Jacques Wainer, Fernando José Von Zuben, Romis Ribeiro de Faissol Attux e Ricardo R. Gudwin pelos ótimos cursos oferecidos;

aos membros da banca examinadora pelos comentários, sugestões e contribuições, que ajudaram a melhorar a qualidade dessa dissertação;

à agência de fomento CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo apoio financeiro concedido durante todo o período de mestrado;

à Faculdade de Engenharia Elétrica e de Computação e a Universidade Estadual de Campinas pela ótima estrutura que oferece aos estudantes e pesquisadores.

Resumo

Com o crescente aumento do volume de informações disponíveis na *Web*, as ferramentas de busca tornam-se cada vez mais importantes para os usuários da Internet. Consequentemente, com o objetivo de se tornar mais visíveis, os *sites* concorrem entre si para ganhar melhores posições nos resultados das buscas feitas por esses usuários. Porém, muitos ganham maior visibilidade através de estratégias que enganam as ferramentas de busca. Esses *sites*, conhecidos como *Web spam*, causam prejuízos pessoais e econômicos aos usuários.

Diante desse cenário, este trabalho apresenta uma análise do desempenho de diversos métodos de aprendizado de máquina aplicados na detecção automática de *Web hosts* que propagam *Web spam*. Os experimentos foram realizados usando duas bases de dados reais, públicas e de grande porte, das quais foram extraídos três diferentes conjuntos de vetores de atributos: baseados no conteúdo das páginas *Web*, baseados nos *links* das páginas *Web* e formados pela transformação dos atributos baseados nos *links*. Também foi analisada a viabilidade da redução de dimensionalidade do espaço dos atributos. Outra contribuição desse trabalho é a proposta de uma abordagem de classificação de *Web spam*, em que as predições obtidas com cada tipo de vetor de atributos são combinadas e uma decisão final é obtida usando-se voto majoritário simples.

Os resultados obtidos indicam que os métodos de *bagging* de árvores de decisão, redes neurais perceptron de múltiplas camadas, floresta aleatória e *boosting* adaptativo de árvores de decisão são promissores na tarefa de detecção de *Web spam*. Além disso, verificou-se que os métodos de aprendizado tem melhor desempenho quando os vetores de atributos baseados no conteúdo e os vetores formados pela transformação dos atributos baseados nos *links* são combinados. Por fim, a combinação das predições obtidas com cada tipo de vetor de atributos gera bons resultados e por isso, essa é uma abordagem recomendada para o combate de *Web spamming*.

Palavras-chave: *Web spam*, *spamdexing*, *spam host*, aprendizado de máquina, classificação, reconhecimento de padrões.

Abstract

Due to the increasing volume of information available on the Web, search engines become increasingly important to Internet users. Consequently, with the purpose of becoming more visible, the Web sites compete to achieve better positions in the results of the searches made by such users. However, many of them achieve a good visibility through strategies that try to circumvent the search engines. This kind of Web sites are known as Web spam and they are responsible for personal injury and economic losses to users.

Given this scenario, this work presents a performance analysis of established machine learning techniques employed to automatically detect Web hosts that disseminate Web spam. The experiments were performed with two real, public and large datasets, from which were extracted three different sets of features vectors: content-based ones, link-based ones and features vectors generated by the transformation of the link-based features. We also analyzed the viability of the dimensionality reduction of the feature space. Another contribution of this work is the proposal of a Web spam classification approach which combines the predictions achieved by each type of features vector and using a simple majority voting.

The results indicate that bagging of decision trees, multilayer perceptron neural networks, random forest and adaptive boosting of decision trees are promising in the task of spam hosts classification. Furthermore, we have conclude that the learning techniques perform better when we have combined the content-based features vectors and the features vectors generated by the transformation of the link-based features. Finally, the combination of the predictions achieved with each type of features vector has achieved superior results and therefore it is a recommended approach to automatically detect Web spam.

Keywords: Web spam, spamdexing, spam host, machine learning, classification, pattern recognition.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

AdaBoost	*Adaptive boosting* (boosting adaptativo)
ASC	Aprendizado sensível ao custo
AUC	*Area under the ROC curve* (área sob a curva ROC)
Bagging	*Bootstrap aggregating*
CFS	*Correlation based feature selection*
CNN	*Condensed nearest neighbor rule* (regra do vizinho mais próximo condensado)
CSS	*Cascading style sheets*
CS-ST	*Cost-sensitive self-training method* (método de autotreinamento sensível ao custo)
DF	*diversity features*
EQM	Erro quadrático médio
IBL	*Instance-based learning* (aprendizado baseado em instâncias)
IG	*Information gain*
KLD	Kullback-Leibler *divergence*
KNN	*K-nearest neighbor* (k-vizinho mais próximo)
LDA	*Latent Dirichlet allocation*
LM	*Language models* (modelos de linguagem)
LVQ	*Learning vector quantization* (quantização vetorial por aprendizagem)
MLP	*Multilayer perceptron* (perceptron de múltiplas camadas)
MLP-GD	Rede neural *multilayer perceptron* treinada com o algoritmo *backpropagation* e método do gradiente descendente
MLP-LM	Rede neural *multilayer perceptron* treinada com o método de Levenberg-Marquardt
OneR	(1-*rules*)
OSS	*One-side selecion* (seleção unilateral)
PCA	*Principal component analysis* (análise de componentes principais)
PHP	*Hypertext preprocessor*
POS	*Part-of-speech* (partes do discurso)
QL	*Qualified-link* (qualidade dos *links*)
RBF	*Radial basis function neural network* (rede neural de função de base radial)
RNA	Rede neural artificial
SEO	*Search engine optimization* (otimização para motores de busca)

SF	*Statistical features*
SGL	*Stacked graphical learning*
SMO	*Sequential minimal optimization* (otimização mínima sequencial)
SMOTE	*Synthetic minority oversampling technique*
SOM	*Kohonen's self-organizing maps* (mapas auto-organizáveis de kohonen)
SOM + LVQ	Combinação das redes neurais *kohonen's self-organizing maps* e *learning vector quantization*
SVM	*Support vector machines* (Máquinas de vetores de suporte)
WEKA	*Waikato environment for knowledge analysis*

Sumário

Introdução

As ferramentas de busca são grandes aliadas dos usuários da Internet para encontrar informações e, por isso, são responsáveis por uma porcentagem expressiva das visitas recebidas pela maioria dos *Web sites*. Logo, para que um *Web site* tenha sucesso, é importante que ele mantenha um alto *ranking* de relevância nos motores de busca. Dessa forma, poderá melhorar seu posicionamento nas páginas de resultados dessas ferramentas, quando forem consultados termos relacionados ao seu conteúdo ou serviços oferecidos. Para atingir esse propósito, podem ser usadas diversas estratégias, conhecidas como técnicas de otimização para motores de busca (SEO – *search engine optimization*) (Ledford 2009).

Existem diversas estratégias éticas de SEO, porém, como afirma (Ledford 2009), para aprender as mais bem sucedidas, é preciso tempo e dedicação. Todos os elementos do *Web site* devem ser projetados para maximizar seu *ranking* nos motores de busca. No entanto, como mencionado no guia de SEO publicado pela empresa Google[1], esse processo deve ser pensado principalmente para os consumidores do *Web site* e não para as ferramentas de busca. É importante que o projeto do *Web site* ofereça conteúdo relevante, facilidade de navegação e outras características que beneficiem o usuário.

O problema é que muitos *sites* preferem investir em técnicas antiéticas de SEO, que enganam os motores de busca manipulando o *ranking* de relevância das páginas apresentadas, degradando a eficiência destas ferramentas. Esses métodos induzem os algoritmos de busca a atribuir a algumas páginas maior relevância do que elas realmente têm, o que deteriora os resultados e frustra os usuários, além de expô-los a conteúdos inadequados e inseguros. Essas técnicas enganosas são conhecidas como *Web spamming* ou *spamdexing*. As páginas *Web* que usam essas técnicas são denominadas *Web spam* e as pessoas que projetam tais páginas são conhecidas como *spammers* (Gyongyi & Garcia-Molina 2005b, Svore, Wu & Burges 2007).

Web spamming vem se tornando um transtorno cada vez maior para os usuários da Internet. Um dos menores problemas causados por essa "praga virtual" diz respeito ao incômodo causado pelo forjamento de respostas não merecidas, irrelevantes e inesperadas que promovem o anúncio de páginas indesejadas. Além disso, usuários que estão em busca de informações acabam gastando mais tempo para encontrar aquilo que desejam e acabam sofrendo prejuízos pessoais e econômicos devido à perda de produtividade (Shen, Gao, Liu, Feng, Song & Li 2006).

[1] *Search Engine Optimization Starter Guide*. Consultar: http://www.google.co.jp/intl/en/Webmasters/docs/search-engine-optimization-starter-guide.pdf

Porém, existem problemas mais graves que oferecem maior risco aos usuários, uma vez que *Web spam* pode possuir conteúdos maliciosos que instalam *malwares* nos computadores das vítimas, facilitando o roubo de senhas bancárias e de informações pessoais, degradando o desempenho dos computadores e da rede, entre outros (Egele, Kolbitsch & Platzer 2011). Outros *sites* também são prejudicados, pois podem perder posições no *ranking* dos motores de busca e, consequentemente, visitas de usuários.

Um dos motivos que fomentam o crescimento do volume de *Web spam* é o incentivo econômico a essas práticas. Muitas empresas preferem investir em técnicas anti-éticas para melhorar o *ranking* de seu *site* nos motores de busca, pois o custo é mais baixo do que se investir em técnicas como melhoria visual, criação de conteúdos que motivem a visita de usuários e melhoria da acessibilidade do *site*. Além disso, muitos *spammers* lucram com o mercado de publicidade *online*. Eles criam páginas com conteúdo mínimo ou irrelevante, que conseguem boas posições no *ranking* dos motores de busca por meio de técnicas de *spamming*, e colocam anúncios para produtos de *sites* de compras ou para outras modalidades de *sites*. Assim, eles recebem algum dinheiro por cada clique dado pelos usuários (Gyongyi & Garcia-Molina 2005b).

Estudos recentes apontam que o volume de *Web spam* vem aumentando consideravelmente nos últimos anos. (Eiron, McCurley & Tomlin 2004) classificou cem milhões de páginas da Internet e descobriu que, em média, onze dos vinte melhores resultados eram páginas pornográficas que obtiveram um elevado índice de relevância por meio da manipulação de *links*. Segundo (Gyongyi & Garcia-Molina 2005b), em 2002, havia cerca de 6% a 8% de *Web spam* nos resultados de pesquisa dos motores de busca, e, em 2003 e 2004, esse valor saltou para 13% e 18%, respectivamente. Outro estudo aponta que cerca de 9% dos resultados de pesquisa contêm pelo menos um *link* de uma página *spam* entre os dez melhores resultados, enquanto que 68% das consultas contêm algum tipo de *spam* nos quatro melhores resultados apresentados pelos motores de busca (Gyongyi & Garcia-Molina 2005b).

Segundo (Ntoulas, Najork, Manasse & Fetterly 2006), cerca de 13,8% das páginas de língua inglesa, 25% das francesas e 22% das germânicas são *spam*. Em outra pesquisa, (Provos, Mavrommatis, Rajab & Monrose 2008) fizeram uma análise de bilhões de URLs entre janeiro e outubro de 2007 e constataram que cerca de três milhões delas eram maliciosas, pois tentavam instalar e executar *malwares* automaticamente nos computadores dos usuários. Além disso, eles concluíram que cerca de 1,3% das consultas realizadas no motor de busca Google retornam pelo menos uma URL mal-intencionada entre os resultados da busca e 0,6% do primeiro milhão de URLs mais frequentes nos resultados de busca do Google levam a conteúdos maliciosos.

Em outro estudo, iniciado em agosto de 2010, (John, Yu, Xie, Krishnamurthy & Abadi 2011) observaram que 36% dos resultados dos motores de busca Google e Bing contém URLs maliciosas. (Lu, Perdisci & Lee 2011) classificaram os termos de busca mais populares nos motores de busca Google e Bing, entre setembro de 2010 e abril 2011, e verificaram que, em média, 50% dos termos de busca mais populares retornam resultados com URLs maliciosas. Um relatório produzido pela empresa Websense[2] mostra que 22,4% dos resultados de busca sobre entretenimento levam a *links* maliciosos. Além disso, segundo um relatório publicado

[2] *Websense 2010 Threat Report*. Consultar: http://www.Websense.com/assets/reports/report-Websense-2010-threat-report-en.pdf

pela empresa McAfee[3], 49% dos termos de busca mais populares retornam algum *site* malicioso entre os cem primeiros resultados da busca. A mesma pesquisa aponta que 1,2% das consultas retornam *links* de *sites* maliciosos entre os cem primeiros resultados e que, em média, surgem 8.600 novos *sites* maliciosos por dia. Ainda, segundo foi publicado no blog de segurança da empresa Google em junho de 2012[4], eles encontram cerca de 9.500 novos *sites* maliciosos por dia e, em aproximadamente doze a quatorze milhões de consultas por dia, realizadas em sua ferramenta de busca, eles mostram seu aviso para advertir os usuários que tentam acessar *sites* infectados presentes nos seus resultados.

Alguns relatórios também apresentam dados sobre o problema de *Web spam* no Brasil. Por exemplo, um relatório da empresa Websense[5], publicado em 2010, mostra que o Brasil hospeda cerca de 4% das páginas *Web* infectadas e, por isso, ele é o quarto país do mundo em número de *malware* ligados a páginas *Web*, perdendo apenas para Estados Unidos (53,7%), China (24,8%) e Espanha (5,4%). Um relatório[6] publicado pela empresa McAfee em 2009 aponta que em cerca de 2.600 termos de consulta mais populares que foram analisados em cinco das maiores ferramentas de busca, 12,1% expõem os usuários a *sites* maliciosos. Esse relatório aponta que entre os termos de busca analisados, os mais perigosos eram, respectivamente, "globo", "juliana paes", "google talk", "google toolbar", "orkut", "corinthians", "palmeiras", "tradutor", "msn" e "músicas". Outro relatório[7] da empresa McAfee, publicado em 2011, afirma que o Brasil figurou como o segundo maior país da América Latina em número de *hosts* que contém páginas maliciosas, perdendo apenas para o Panamá. Já, outro estudo[8] publicado pela mesma empresa mostra que, no segundo trimestre de 2012, o Brasil figurou como o terceiro maior país da América Latina em número de *hosts* que hospedam conteúdos maliciosos, perdendo apenas para as Bahamas e as Ilhas Virgens Britânicas.

Felizmente, algumas propostas para a detecção automática de *Web spam* vêm sendo apresentadas na literatura. Entre elas, as mais utilizadas são as técnicas de aprendizado de máquina, tais como seleção de conjuntos (*ensemble selection*) (Erdélyi, Garzó & Benczúr 2011, Geng, Wang, Li, Xu & Jin 2007), *clustering* (Castillo, Donato & Gionis 2007, Largillier & Peyronnet 2010), floresta aleatória (Erdélyi et al. 2011), *boosting* (Erdélyi et al. 2011, Geng et al. 2007), máquinas de vetores de suporte (SVM) (Svore et al. 2007) e árvores de decisão (Castillo et al. 2007).

Apesar de alguns métodos para detecção de *Web spam* já terem sido avaliados na literatura, é difícil compará-los, pois os autores usam bases de dados, atributos e metodologias diferentes. Por isso, é importante que seja feita uma análise mais detalhada de diferentes métodos de aprendizado para a detecção automática de *Web spam*. Os resultados dessa análise podem

[3] *McAfee Threats Report: First Quarter 2011*. Consultar: http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2011.pdf

[4] *Google Online Security Blog – Safe Browsing: Protecting Web Users for 5 Years and Counting*. Consultar: http://googleonlinesecurity.blogspot.com.br/2012/06/safe-browsing-protecting-web-users-for.html

[5] *Websense 2010 Threat Report*. Consultar: http://www.Websense.com/assets/reports/report-Websense-2010-threat-report-en.pdf

[6] *The Web's Most Dangerous Search Terms*. Consultar: http://www.mcafee.com/us/resources/reports/rp-most-dangerous-search-terms.pdf

[7] *McAfee Threats Report: First Quarter 2011*. Consultar: http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2011.pdf

[8] *McAfee Threats Report: Second Quarter 2012*. Consultar: http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2012.pdf

contribuir significativamente para comparações futuras e para direcionamentos promissores de novos estudos que envolvam melhorias dos métodos avaliados ou a proposta de novas abordagens.

Objetivos e contribuições

Esse trabalho tem como objetivo analisar o desempenho de métodos tradicionais de aprendizado de máquina para classificar automaticamente *Web spam* e contribuir para o combate a essa "praga" virtual, detectando os métodos viáveis e não viáveis para a solução desse problema. Além disso, é necessário identificar os atributos mais importantes para a classificação e testar adaptações como redução de dimensionalidade, além de adaptações e possíveis combinações de métodos para maximizar a acurácia da classificação.

Diante disso, esse trabalho oferece as seguintes contribuições:

- foram analisados diferentes métodos tradicionais de aprendizado de máquina, tais como redes neurais artificiais, métodos de agregação de múltiplos classificadores, métodos baseados em árvores, métodos baseados em *clustering* e máquinas de vetores de suporte na detecção de *Web spam*;
- os métodos de aprendizado de máquina foram avaliados na classificação de amostras reais de *Web spam* extraídas de duas bases de dados públicas e de grande porte;
- os dados utilizados nos experimentos foram tratados para resolver o problema de desbalanceamento de classes;
- foram realizados experimentos com atributos baseados no conteúdo das páginas *Web*, atributos baseados na relação dos *links* e atributos formados pela transformação dos atributos baseados na relação dos *links*;
- foram analisadas combinações dos atributos mencionados no item anterior;
- foi proposta a classificação de *Web spam* por meio da combinação das predições obtidas com os diferentes tipos de atributos, usando como decisão final o voto majoritário;
- foi realizada uma análise estatística dos resultados a fim de determinar os melhores métodos para o problema de classificação de *Web spam*;
- os resultados obtidos nesse trabalho foram comparados com outros resultados disponíveis na literatura.

Organização do trabalho

Para facilitar a leitura e a compreensão deste trabalho, ele foi estruturado da seguinte maneira.

- No Capítulo 1, são apresentados os conceitos básicos sobre o funcionamento dos motores de busca. Também são apresentadas as principais técnicas de *Web spamming*, em que se destacam as baseadas em conteúdo e as baseadas na relação de *links*, que são as mais usadas pelos *spammers* e serão as técnicas tratadas nesse trabalho. Nesse capítulo, são apresentados também alguns trabalhos da literatura que oferecem propostas para o combate de *Web spamming*.

- No Capítulo 2, são apresentados os conceitos básicos dos métodos de aprendizado de máquina analisados nesse trabalho. Alguns dos métodos escolhidos para serem analisados são as redes neurais artificiais, métodos de agregação de múltiplos classificadores, métodos baseados em árvores, métodos baseados em *clustering* e máquinas de vetores de suporte.

- No Capítulo 3, são apresentadas as bases de dados utilizadas nesse trabalho. Também são detalhados os diferentes tipos de atributos que foram usados nos experimentos.

- No Capítulo 4, são apresentados os conceitos básicos do método de análise de componentes principais que é um dos principais métodos utilizados na literatura para a redução de dimensionalidade. Também são apresentadas as principais estratégias usadas na literatura para tratar o problema de desbalanceamento de classes.

- No Capítulo 5, são apresentadas as configurações adotadas para os algoritmos de classificação, as métricas de avaliação do desempenho dos métodos de aprendizado de máquina na classificação de *Web spam*, bem como os resultados que foram obtidos. Nesse contexto, destaca-se a análise dos resultados obtidos com todos os tipos de atributos e combinações dos mesmos e resultados com tratamento e não tratamento do problema de classes desbalanceadas. Também é analisada a viabilidade da redução de dimensionalidade dos vetores de atributos para reduzir o tempo de treinamento e geração do modelo (hipótese) e, possivelmente, eliminar redundâncias dos dados existentes. Ainda, é proposta uma abordagem de classificação de *Web spam* que faz a combinação das predições obtidas com os diferentes tipos de atributos. Por fim, é realizada uma validação estatística para determinar os melhores métodos e é feita uma comparação dos resultados obtidos nesse trabalho com outros resultados disponíveis na literatura.

- Finalmente no Capítulo 6 são apresentadas as conclusões e direções para trabalhos futuros.

Capítulo 1

Técnicas de *Web spamming*

Neste capítulo, são apresentadas noções básicas sobre a dinâmica do processo de classificação das páginas *Web* pelos motores de busca, bem como as principais técnicas usadas pelos *spammers* para enganar essas ferramentas. Adicionalmente, são apresentados diferentes tipos de conteúdo *spam* e definições e características dos *spam links*. Também são abordadas as características das seguintes técnicas de *Web spamming*: *cloaking spam*, redirecionamento *spam* e *click spam*. Apesar de serem apresentadas diversas técnicas de *Web spamming*, esse trabalho tratará apenas das duas principais: conteúdo *spam* e *spam links*. Nesse capítulo, são apresentados também algumas técnicas propostas na literatura para o combate de *Web spamming*.

1.1 Noções básicas sobre motores de busca

A *Web* vem se tornando cada vez mais importante para seus usuários, tanto como fonte de diversão, comunicação, pesquisa, notícias e comércio. Cada vez mais usuários estão tendo acesso à Internet. O tempo que esses usuários permanecem conectados também é cada vez maior, usando dispositivos como computadores pessoais, *notebooks*, *smartphones*, *tablets* ou outros meios. Diante disso, é cada vez maior o interesse das pessoas e empresas em utilizarem a *Web* para se promoverem, divulgarem ideias e obter lucros. Por isso, milhares de novos *sites* são criados por dia. Esse cenário, aumenta a importância dos motores de busca (Google, Yahoo, Baidu, Bing, Ask, entre outros) que têm a intenção de ajudar os usuários a encontrarem as informações desejadas, apresentando resultados relevantes, confiáveis e de forma organizada, rápida e eficiente.

Para estimar a relevância dos *sites* em relação a um termo ou conjunto de termos, os motores de busca precisam manter uma base de dados atualizada dos *sites* existentes na Internet. Para isso, eles usam programas de computadores que navegam pela Internet coletando informações em relação aos *sites*. Esses programas são conhecidos como *Web crawlers*, *Web spiders* ou *Web robots* (robôs *Web*). Eles visitam cada URL na *Web* que não seja bloqueada e coleta palavras chaves, frases, *links* e outras informações que ajudem a classificar as páginas *Web*. Depois que essas informações são armazenadas na base de dados do motor de busca, elas são processadas e as URLs são classificadas por algum método de classificação e recuperação, que normalmente é diferente para cada motor de busca. Dessa forma, quando um usuário faz uma pesquisa

usando um determinado termo, o motor de busca tentará encontrar entre todas as URLs que seus *Web crawlers* coletaram, quais são as mais relevantes em relação ao termo pesquisado pelo usuário (Ledford 2009).

Um dos mais famosos métodos de classificação de URLs é o *PageRank* (Brin & Page 1998, Page, Brin, Motwani & Winograd 1998) usado pelo motor de busca Google. Segundo (Page et al. 1998), o *PageRank* é um método de classificação das páginas *Web* que analisa o grafo *Web*, ou seja, analisa a estrutura de *links* da Internet. Cada página da Internet possui um número de *links* que apontam para outras *páginas*, conhecidos como *forward links* e possui também, *links* de outras páginas que apontam para ela, os *backlinks*. Então, o cálculo da importância de uma página *Web* feito pelo *PageRank* considera o número de *backlinks* da página e a importância das páginas que apontam para ela. Logo, receber um *link* de uma página relevante da *Web* é melhor que receber vários *links* de páginas não relevantes. Porém, o *PageRank* também considera outros fatores quando apresenta os resultados para o usuário. Além de ser importante na *Web*, a página precisa ser importante em relação ao termo de consulta usado pelo usuário. Logo, ele também usa técnicas de correspondência de texto e examina dezenas de aspectos do conteúdo da página para encontrar páginas que sejam relevantes para a consulta (Page et al. 1998, Ledford 2009).

O grande sucesso e importância dos motores de busca para os usuários da Internet e a consequente necessidade de estar bem posicionado nessas ferramentas para que um *site* tenha sucesso, aumentou a importância da utilizaçao de técnicas de SEO. Como foi mencionado anteriormente, essas técnicas são usadas para melhorar a classificação de um *site* nos motores de busca, investindo em sua estrutura e conteúdo. Para obter bons resultados, é necessário conhecer algumas características consideradas pelos motores de busca em seus métodos de classificação (Ledford 2009).

Técnicas de SEO que seguem as diretrizes dos motores de busca são consideradas formas éticas e podem ser chamadas de *white hat* SEO (chapéu branco). Porém, existem técnicas que tentam manipular as características observadas pelos motores de busca para determinar a relevância de um *site*, de forma que o *site* aparente ter mais qualidade do que realmente tem. Essas técnicas são consideradas anti-éticas e podem ser chamadas de *black hat* SEO (chapéu preto). Quando os motores de busca encontram *sites* que usam *black hat* SEO normalmente o consideram irregular, penalizam o *site* com perda de posições nos resultados de busca e, às vezes, retiram todas as informações desse *site* de sua base de dados para que ele não apareça mais nos resultados das pesquisas feitas pelos usuários. O termo *black hat* SEO usado pela comunidade de SEO se refere às técnicas de criação de *Web spam* e é um dos principais problemas enfrentados pelos motores de busca (Ledford 2009, Cahill & Chalut 2009). Existem diversas técnicas de *black hat SEO* ou *Web spamming*, o que dificulta a detecção das páginas *Web* que as utilizam.

1.2 Conteúdo *spam*

Alguns motores de busca determinam o *ranking* de uma página por meio da avaliação de seu conteúdo textual. Nesse processo são coletados termos (palavras) em diferentes posições da página *Web*. Esses termos são usados para categorizar a página *Web* em um ou mais grupos de páginas que tratam de assuntos semelhantes e determinar a relevância dessa página em relação

a um termo ou grupo de termos de consulta. Então, a técnica de conteúdo *spam* manipula os termos apresentados na página *spam*, de forma que ela seja categorizada em grupos de páginas da qual ela não pertence e ganhe relevância que não condiz com a qualidade do seu conteúdo. Segundo (Gyongyi & Garcia-Molina 2005b), um exemplo de conteúdo *spam* é uma página de pornografia que adiciona milhares de palavras-chaves em sua página inicial sobre assuntos que não tem ligação com conteúdo pornográfico e deixa essas palavras invisíveis por meio de recursos das linguagens de programação para *Web*. Assim, quando uma pessoa realiza uma pesquisa usando um dos termos presentes nessas palavras-chaves, os motores de busca podem retornar como resultado essa página de pornografia, o que na maioria das vezes não atende aos anseios do usuário.

Existem diversos tipos de *Web spam* baseados em conteúdo, tais como (Gyongyi & Garcia-Molina 2005b, Spirin & Han 2012):

- Título *spam*: essa categoria de *Web spamming* insere palavras chaves populares no título da página. Isso é feito porque a maioria dos motores de busca dão grande peso aos termos presentes nesse campo.

- *Meta tags spam*: as *meta tags* são linhas de código HTML que descrevem o conteúdo de uma página ou fornecem palavras-chaves, nome do autor ou outras informações que não são visíveis aos usuários, mas podem ser analisadas pelos motores de busca. Os *spammers* costumam adicionar palavras-chaves nas *meta tags*, pois alguns motores de busca dão algum tipo de relevância a esses termos. Um exemplo de *meta tag spam* é apresentado na Figura 1.1.

```
<meta name="keywords" content="câmeras, fotográficas, digitais, câmeras fotográficas,
câmeras digitais,  mega pixel, zoom, foto digital, fotos, promoção, preço barato,
melhores câmeras fotográficas, samsung, nikon, canon, sony, lindas paisagens, melhor preço,
lentes, fotos profissionais, fotos amadoras, alta resolução, fotos do ano, melhores fotos do mês,
fotos na web, tecnologia, fotos de famosos, fotos de atores, fotos sensuais, câmera de bolso,
câmera a prova d'água, câmera resistente, câmera para presente, câmera dia das mães, câmera natal,
câmera dia dos pais, câmera dia dos namorados, câmera para as férias, fotos panorâmicas">
```

Figura 1.1: Exemplo de *meta tag spam*.

- Texto âncora *spam*: os textos âncoras são usados para explicar, por meio de palavras chaves, o assunto abordado pelo destino apontado por um *link*. Assim como no título *spam*, os *spammers* inserem palavras chaves que não correspondem ao conteúdo real da página de destino para o qual o *link* está apontando.

- URL *spam*: alguns motores de busca quebram a URL de uma página em um conjunto de termos e utilizam esses termos para determinar a relevância da página. Então, por exemplo, se um *spammer* quer aumentar a relevância de uma página do seu *site* em relação ao termo "*smartphones* baratos", ele pode criar a seguinte URL: *smartphones-baratos.com/baratos/smartphones.html*.

- Texto *alt spam*: a *tag alt* é utilizada para descrever uma imagem em uma página *Web*. Essa descrição só é vista pelo usuário quando a imagem não é carregada pelo *browser* (navegador de Internet). Porém, ela é vista pelos motores de busca e alguns deles, dão alguma relevância para os termos presentes nessa *tag*. Consequentemente, alguns *spammers* adicionam imagens nas páginas *Web* e inserem palavras chaves populares nas *tags alt*, sem se preocupar se estão relacionadas com a imagem que representam. Muitas vezes, os *spammers* inserem inúmeras imagens, porém com tamanhos mínimos ou as tornam invisíveis, pois o único objetivo é conseguirem pontuação com as palavras chaves presentes nas *tags alt*. Exemplo de texto *alt* spam: **.

- Corpo *spam*: essa técnica de *Web spamming* é uma das mais simples e mais usadas pelos *spammers*. Ela consiste em adicionar os termos chaves no corpo da página *Web*. Alguns *spammers* adicionam esses termos na página *spam* e os escondem dos usuários através de recursos das linguagens de programação *Web*. Uma das estratégias empregadas é deixar o texto com a mesma cor do fundo da página *Web* ou colocá-lo dentro de camadas (*layers*) da linguagem de estilo CSS (*Cascading Style Sheets*) e, após isso, tornar essas camadas invisíveis. Desta forma, esses termos são "vistos" pelos motores de busca, mas para o usuário é apresentado somente o conteúdo de interesse do *spammer*.

Como apresentado, cada tipo de conteúdo *spam* adiciona palavras chaves em áreas específicas da página *Web*. Porém, além das áreas em que esses termos podem ser adicionados, os *spammers* também se preocupam em como inserir esses termos de forma a conseguir melhores resultados. Algumas estratégias usadas pelos *spammers* são apresentadas a seguir (Gyongyi & Garcia-Molina 2005b, Spirin & Han 2012):

- Repetição de um ou mais termos chaves: através dessa estratégia, os motores de busca podem ser enganados e atribuir alta relevância para a página *Web* em relação a um termo ou um grupo de termos de consulta usados pelos usuários.

- Inserção de palavras chaves raras, que representam assuntos pouco explorados na *Web*: essa estratégia pode ser bastante efetiva, pois quando usuários pesquisam termos raros, o número de *sites* retornados como resultado e considerados relevantes para esses termos é pequeno. Logo, a chance de uma página *spam*, que usa essa estratégia, estar entre os primeiros resultados é maior, o que pode favorecê-la a conseguir o principal objetivo dos *spammers*: o acesso dos usuários. Normalmente essa estratégia não é usada sozinha. Os *spammers* costumam usá-la juntamente com estratégias que manipulam termos de busca populares.

- Inserção de palavras aleatórias: alguns *spammers* costumam adicionar palavras chaves de forma aleatória na página *Web*, sem se preocupar se terá algum sentido textual. Essas palavras podem referir-se a algum assunto específico, bastante pesquisado pelos usuários,

ou muitas vezes são adicionadas palavras dos mais variados assuntos. Alguns *spammers* chegam a inserir dicionários inteiros na página *Web* para que ela tenha mais chance de aparecer como resultado de pesquisa de um grande número de termos.

- Inserção de palavras chaves em posições aleatórias de conteúdos relevantes: nessa estratégia, os *spammers* criam uma cópia de algum conteúdo relevante e bastante pesquisado, tais como notícias, novos artigos de autores importantes, tutoriais, sinopses de novos capítulos de seriados, entre outros. Depois, adicionam palavras chaves em diferentes posições do conteúdo. Com essa estratégia, os *spammers* aproveitam ao mesmo tempo a relevância do conteúdo que foi copiado e a relevância dos termos que foram inseridos, aumentando a probabilidade de enganar os motores de busca.

- Inserção de frases ou parágrafos de diferentes temas: essa estratégia consiste em criar conteúdo para as páginas *spam* a partir de frases ou parágrafos de diferentes fontes que abordem diferentes temas. Isso pode ser associado à formação de conteúdo a partir de recortes de jornais. Porém, nesse caso, as fontes são digitais e os recortes são de temas diversos. Com essa estratégia, os *spammers* conseguem aumentar a relevância de uma página *Web* em relação aos temas abordados nos recortes.

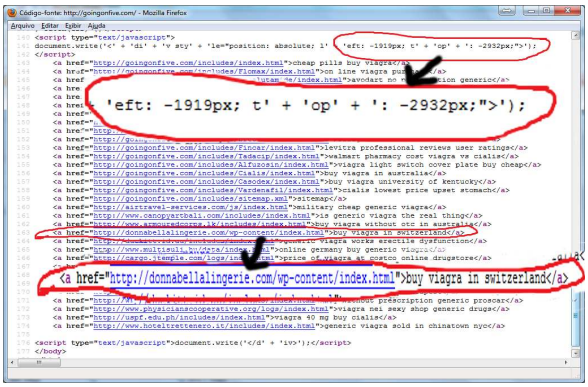
1.3 *Spam links*

Spam links é uma categoria de técnicas de *Web spamming* que explora o sistema de análise de *links* dos motores de busca para aumentar seu *ranking* de relevância. Um dos critérios usados pelas ferramentas de busca para determinar a relevância de uma página é a análise da quantidade de *links* que apontam para ela. Alguns motores de busca ponderam a quantidade de *links* pela qualidade dos *sites* que possuem algum *link* apontado para a página que está sendo analisada (Najork 2009, Shen et al. 2006, Henzinger, Motwani & Silverstein 2002).

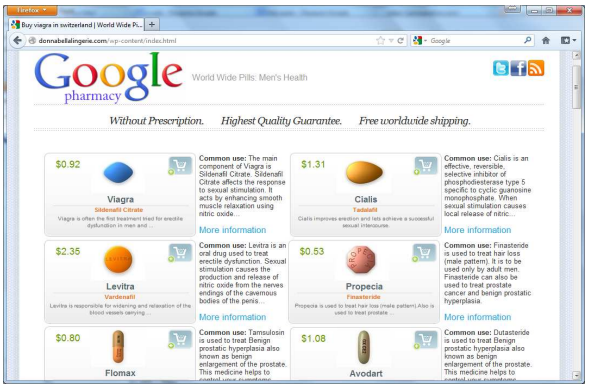
Uma das principais estratégias usadas pelos *spammers* é a criação de um ou vários *sites* em que cada uma de suas páginas possui inúmeros *links* que apontam para outras páginas do mesmo *site* e para outras páginas dos outros *sites*. Desta forma, o *ranking* dessas páginas aumenta. Então, o *spammer* adiciona *links* nessas páginas que apontam para uma página objetivo. Logo, os motores de busca podem ser enganados, classificando essa página objetivo como relevante devido à falsa popularidade criada por essa estratégia de *Web spamming*. A maior dificuldade de tratar esse tipo de *Web spam* é que a página objetivo, muitas vezes é uma página *Web* idônea que vende algum produto ou transmite alguma mensagem ou ideia, o que dificulta sua detecção. É importante destacar que tanto as páginas que foram criadas especificamente para enviar *links* quanto a página objetivo são consideradas *Web spam* (Henzinger et al. 2002, Gyongyi & Garcia-Molina 2005b, Spirin & Han 2012). A Figura 1.2 apresenta um exemplo de *spam links*.

Observe que a Figura 1.2(a) apresenta uma página *Web* que não tem vestígio de *spam*. Porém, observando seu código fonte, mostrado na Figura 1.2(b), observa-se que a página é na verdade um *spam link*, pois possui inúmeros *links* que apontam para outras páginas, com o único intuito de aumentar o *ranking* delas nos motores de busca. Note que uma das partes destacadas na Figura 1.2(b) apresenta os valores "-1919px" e "-2932px". Esses valores estão sendo usados

(a) Página *Web* aparentemente normal.



(b) Código fonte da página *Web*.



(c) Página *Web* objetivo.

Figura 1.2: Exemplo de *spam links*.

em um código da linguagem JavaScript que determina o posicionamento da camada onde estão contidos os *links*. Como os valores são negativos, os *links* que estão nessa camada não são vistos por um usuário normal, mas são vistos pelos *Web crawlers*. Observe também que um dos *links* apontam para o endereço do *site* que é mostrado na Figura 1.2(c). Como pode ser observado, esse *site* possui fins comerciais e por isso há interesse em aumentar seu posicionamento nos resultados das consultas feitas nos motores de busca. O carácter anti-ético desse *site* é bem visível, pois vende remédios sem prescrição médica e ainda usa de forma ilegal o logotipo da empresa Google.

Outra estratégia usada pelos *spammers* é inserir *links* em inúmeros fóruns, grupos de discussão e comentários em *blogs* e *sites* que apontem para sua página objetivo. Dessa forma, a página *spam* parece ter uma grande popularidade, já que vários fóruns, *blogs* e *sites*, regulares e muitas vezes relevantes, possuem *links* apontados para ela. Esses *links* são inseridos de maneira discreta, em alguma palavra de frases que parabenizam o *blog* ou o *site*, ou em comentários considerados normais. Então, mesmo que haja um moderador (pessoa que analisa comentários em *sites*, fóruns e *blogs* e decide se podem ser publicados ou não), às vezes ele publica esses comentários achando que são normais (Gyongyi & Garcia-Molina 2005b, Spirin & Han 2012).

Existe ainda outra estratégia de criação de *spam links* em que alguns *spammers* se unem, formam um grupo e criam uma rede de troca de favores, em que cada *spammer* adiciona inúmeros *links* em seu *site* que aponta para os *sites* dos outros *spammers* do grupo (Gyongyi & Garcia-

Molina 2005b).

1.4 *Cloaking spam*

A técnica de *cloaking spam* (camuflagem) consiste em fazer com que o conteúdo apresentado aos *Web crawlers* seja diferente do conteúdo mostrado ao usuário. Dessa forma, para os usuários é apresentado um *Web spam*, enquanto que para os *Web crawlers* é apresentada uma página *Web* comum, sem vestígios de *spam* (Gyongyi & Garcia-Molina 2005b).

Para diferenciar um usuário de um *Web crawler*, uma das formas usadas pelos *sites* que usam a técnica de *cloaking spam* é verificar o endereço IP do visitante. Existem *softwares* e *sites* que disponibilizam uma lista atualizada de endereços IP usados pelos *Web crawlers*. Então, antes de apresentar o conteúdo, o *site* verifica se o endereço IP do visitante está nessa lista. Caso esteja, é enviado para ele um conteúdo diferente do que é apresentado a um usuário. Outra maneira usada para identificar se o visitante é um *Web crawler* é por meio dos cabeçalhos *user-agent*, que fornecem detalhes da aplicação que está requisitando uma determinada página *Web*. Os cabeçalhos *user-agent* dos *browsers* são diferentes dos cabeçalhos dos *Web crawlers*, o que facilita o trabalho dos *sites* que usam *cloaking spam*. No entanto, alguns *Web crawlers* contornam esse problema usando cabeçalhos *user-agent* falsos, iguais aos usados por algum *browser* (Gyongyi & Garcia-Molina 2005b, Lin 2009, Sunil & Sardana 2012).

1.5 Redirecionamento *spam*

Na técnica de redirecionamento *spam*, o *spammer* cria uma página *Web* comum e aumenta o seu *ranking* por meio de outras técnicas de *Web spamming*. O problema é que o conteúdo dessa página *Web* é visto apenas pelos motores de busca e é usado estrategicamente para ganhar boas posições nos resultados de pesquisa. Quando o usuário clica no resultado de pesquisa que deveria abrir essa página, ele é redirecionado para uma página *spam*, com conteúdo diferente. Os *spammers* conseguem redirecionar o usuário por meio de códigos de linguagens de programação *Web*. Alguns *Web crawlers* não interpretam todos esses códigos devido ao alto custo computacional e, por isso, fazem apenas uma análise superficial, que pode ser insuficiente para detectar *scripts* mais sofisticados. Porém, é importante destacar que nem todas as páginas *Web* com redirecionamento são *spam*. Veja um exemplo simples de redirecionamento usando a linguagem PHP (*Hypertext Preprocessor*):

<?php header('location: http://www.seusite.teste.br'); ?>

1.6 *Click spam*

Click spam é uma técnica de *Web spamming* em que os *spammers* criam mecanismos automáticos que fazem consultas nas ferramentas de busca usando termos presentes nas suas páginas *spam* e simulam cliques no resultado correspondente a sua página de interesse. Diante disso, os motores de busca entendem que há uma grande procura por essa página e que, consequentemente, ela é relevante, aumentando seu *ranking*. Para dificultar o trabalho dos motores de

busca em identificar que esse cliques não foram feitos por usuários humanos, os *spammers* usam diversos computadores para obter uma grande diversidade de endereços IP. Em alguns casos os computadores usados são computadores "zumbis" de usuários da Internet, ou seja, computadores infectados por *malwares* (Najork 2009, Spirin & Han 2012).

1.7 Técnicas para o combate de *Web spamming*

Na literatura, existem propostas para combater os diferentes tipos de *Web spamming*. O problema é que a maioria dos trabalhos usam metodologias e base de dados distintas, o que dificulta comparações. Existem trabalhos na literatura que abordam conteúdo spam (Westbrook & Greene 2002, Fetterly, Manasse & Najork 2004, Ntoulas et al. 2006, Attenberg & Suel 2008, Erdélyi et al. 2011), *spam links* (Gyongyi, Garcia-Molina & Pedersen 2004, Gyöngyi & Garcia-Molina 2005a, Wu & Davison 2006, Becchetti, Castillo, Donato, Leonardi & Baeza-Yates 2006b, Becchetti, Castillo, Donato, Leonardi & Baeza-Yates 2006a, Becchetti, Castillo, Donato, Baeza-Yates & Leonardi 2008, Largillier & Peyronnet 2012), *cloaking spam* (Wu & Davison n.d., Wu & Davison 2006, Lin 2009, Sunil & Sardana 2012), redirecionamento *spam* (Wu & Davison n.d., Chellapilla & Maykov 2007, Wang, Ma, Niu & Chen 2007) e *click spam* (Dou, Song, Yuan & Wen 2008, Wei, Liu, Zhang, Ma, Ru & Zhang 2012). Algumas propostas da literatura para o problema de *Web spamming* são detalhadas a seguir.

Em um dos trabalhos da literatura de *Web spam*, (Ntoulas et al. 2006) exploraram uma variedade de métodos para detectar automaticamente *Web spam*, usando atributos extraídos do conteúdo das páginas. Seus experimentos foram realizados com uma base de dados composta de 105.484.446 páginas extraídas da ferramenta de busca *MSN Search*, em agosto de 2004. Dessa base de dados, eles selecionaram, aleatoriamente, 17.168 páginas e, manualmente, rotularam 2.364 páginas como *spam* e 14.804 como *ham*. Os métodos usados pelos autores foram: métodos baseados em árvores de decisão, métodos baseados em regras, redes neurais e SVM. Entre eles, o que obteve melhor desempenho foi o método de árvores de decisão C4.5, cuja sensitividade (*recall*) na classificação de *spam* foi de 82,1%, enquando que a precisão foi de 84,2%. Eles também fizeram agregação de árvores decisão usando os métodos *bagging* e *boosting*. O primeiro método obteve 84,4% de sensitividade e 91,2% de precisão em relação a classe *spam*, enquanto o segundo obteve uma sensitividade de 86,2% e uma precisão de 91,1%.

(Wu & Davison 2006) propuseram um método automático de duas etapas para detectar *cloaking spam* (chamado pelos autores de *semantic cloaking*). A primeira etapa do método proposto, é a etapa de filtragem. Nela, é gerada uma lista de possíveis páginas *cloaking spam*. Na segunda etapa, um classificador é usado para detectar páginas *cloaking spam* na lista gerada na primeira etapa. Para criar uma das base de dados que foi usada nos experimentos, eles coletaram 257 termos de consulta populares das seguintes ferramentas de busca: Google, Lycos, Ask e AOL. Depois, eles fizeram buscas na ferramenta Google usando esses termos e coletaram os 200 primeiros resultados obtidos com cada um deles, o que resultou em 47.170 URLs. Dessa lista de URLs, eles rotularam 539 como *cloaking spam* e 746 como páginas *ham*. Eles usaram essas páginas para treinar um classificador SVM, que obteve 93% de precisão e 85% de sensitividade (*recall*). Após isso, os autores extraíram 3.378.870 URLs da base de dados 2004 ODP RDF

file[1]. No processo de filtragem, 364.993 URLs foram listadas como possíveis *cloaking spam*. Dessa lista, 400 páginas *Web* foram aleatoriamente selecionadas para serem apresentadas ao classificador SVM, onde 52 eram *cloaking spam*. Com essa base de dados, o desempenho do classificador foi de 91,5% de precisão e 82,7% de sensitividade.

No artigo produzido por (Svore et al. 2007) é apresentado um método de detecção de *Web spam* que usa atributos baseados em conteúdo e que consideram o tempo de *rank*. A base de dados usados pelos autores é composta por 31.300 páginas, das quais 3.133 são *spam*. Para a criação dessa base de dados, foram feitas buscas na ferramenta *Microsoft Live Search* usando 1.697 termos encontrados no seu registro de consultas. Para cada termo, as dez primeiras páginas *Web* retornadas na consulta foram julgadas por um humano como *spam*, *ham* ou indefinida. As páginas julgadas como indefinidas foram descartadas. Os experimentos foram realizados usando um classificador SVM com *kernel* linear usando atributos que consideram o tempo de *rank* e são *query-dependent* (atributos baseados no conteúdo e que estão relacionados ao termo de busca) e atributos que consideram o tempo de *rank* e são *query-independent* (atributos baseados no conteúdo e que não estão relacionados ao termo de busca). De acordo com os autores, os resultados indicam que o desempenho dos classificadores é melhor quando eles são treinados com atributos que consideram o tempo de *rank* e são *query-dependent*, do que quando eles são treinados apenas com atributos baseados no conteúdo.

No trabalho de (Zhang, Zhu, Zhang, Zhou & Xu 2011) é proposto um algoritmo de aprendizado para detecção de *Web spam* chamado HFSSL (*harmonic functions based semi-supervised learning*). No método proposto, as páginas *Web*, rotuladas ou não rotuladas, são representadas como vértices de um grafo ponderado e o problema de aprendizado é modelado como um campo Gaussiano aleatório neste gráfico. A média desse campo é caracterizada por funções harmônicas que possuem uma forma fechada e podem ser obtidas por meio de manipulação matricial. Os autores usaram em seus experimentos a base de dados WEBSPAM-UK2006. Eles obtiveram como resultado uma precisão de 83,8%, uma sensitividade de 93,1% e F-medida de 0.882. Segundo os autores, uma das vantagens do método proposto é que ele permite a utilização de uma grande quantidade de dados sem rótulos, diferente da maioria dos métodos usados para a detecção de *Web spam*. Essa característica é importante, pois a rotulagem das páginas *Web* como *spam* ou *ham* é um trabalho demorado e exige experiência quando é feito por humanos. Além disso, os resultados dos experimentos mostraram que o algoritmo é eficaz e melhor do que outros métodos de aprendizado semissupervisionado.

(Egele et al. 2011) criaram um método capaz de identificar e remover *Web spam* da lista de resultados dos motores de busca. Para isso, eles analisaram quais atributos são importantes para que uma página tenha um bom *ranking* de relevância nos motores de busca, por meio da observação, por cerca de três meses, do efeito causado por diferentes combinações de atributos no *ranking* de algumas páginas de teste. Os atributos que mais influenciaram o valor do *ranking* dessas páginas foram considerados mais importantes. Depois disso, os autores criaram uma base de dados para treinamento, formada pelos cinquenta primeiros resultados de cada consulta realizada com doze termos de busca populares do Google. Dessa base de dados, foram descartados os *links* para páginas que não são HTML, tais como arquivos *.pdf* ou *.ppt*. Após, esse

[1] *Open Directory RDF Dump*. Disponível em http://rdf.dmoz.org/.

pré-processamento, as páginas foram rotuladas como *spam* ou *ham*. Os atributos considerados importantes foram extraídos dessas páginas Web e foram classificados por meio de um método de árvores de decisão. De acordo com os autores, com o método proposto por eles, uma em cada cinco páginas *spam* podem ser detectadas, sem remover nenhum resultados válido dos motores de busca.

No trabalho proposto por (Rungsawang, Taweesiriwate & Manaskasemsak 2011) é apresentada uma abordagem de classificação de *Web spam* baseada no algoritmo de otimização por colônia de formigas, usando heurísticas padrões e funções básicas de atualização do feromônio. Nos experimentos, eles usaram atributos baseados no conteúdo e nos *links*, bem como o valor *spamicity* (probabilidade de um *host* ser *spam*) dos *hosts* que compõem a base de dados WEBSPAM-UK2006. Os resultados obtidos indicam que o método proposto teve melhor desempenho que os métodos árvores de decisão e SVM.

(Liu, Chen, Kong, Yu, Zhang, Ma & Ru 2012) propuseram seis atributos, para separar páginas *Web spam* de páginas *ham*, baseados no comportamento dos usuários quando visitam *Web spam*. Eles apresentaram também um novo *framework* que é capaz de identificar vários tipos de *Web spam* por meio da análise do comportamento dos usuários. A construção da base de dados usada nesse trabalho foi feita por três profissionais. Após todos os procedimentos de rotulagem e filtragem das páginas *Web*, eles coletaram 802 *sites* para o conjunto de treinamento. Por outro lado, o conjunto de teste foi formado por 491 páginas *spam* e 1.248 páginas *ham*. Após a formação da base de dados e a extração dos atributos baseados no comportamento, os autores usaram o método *naive Bayes* para o processo de classificação de *Web spam*. O melhor resultado obtido foi uma AUC (*area under the ROC curve* – área sob a curva ROC) de 0,915 que, segundo os autores, supera os resultados de outros algoritmos de aprendizado, como o SVM e o C4.5, além de resultados de outros trabalhos da literatura. De acordo com os autores, por meio de alguns experimentos eles verificaram que o *framework* proposto, também consegue identificar conteúdo *spam*, *spam links* e outros tipos de técnicas de *Web spamming*.

(Jayanthi & Sasikala 2012) apresentaram um algoritmo para detecção de *Web spam* chamado `WESPACT` que usa algoritmos genéticos para classificar atributos baseados em *links* e em conteúdo. Os autores usaram a base de dados WEBSPAM-UK2007[2] em seus experimentos. O melhor resultado obtido foi uma precisão de 1,8 em relação a classe *spam*.

Seguindo a mesma linha do trabalho anterior, fazendo uso de algoritmos de computação natural, (Taweesiriwate, Manaskasemsak & Rungsawang 2012) propuseram um algoritmo de otimização por colônia de formigas para a detecção de *Web spam*, porém seguem a estratégia de *TrustRank* (Gyongyi et al. 2004) a fim de gerar regras de classificação. Para os experimentos, eles usaram a base de dados WEBSPAM-UK2006[3] e atributos baseados no conteúdo e nos *links*. De cada *host* da base de dados, eles usaram também o valor de *spamicity*, que indica qual a probabilidade de um *host* ser *spam*. Esse valor já foi fornecido com a base de dados. O melhor resultado obtido foi uma acurácia de aproximadamente 87%, que, segundo os autores, superou os resultados obtidos com outros modelos populares de aprendizado baseado em regras, tais

[2] *Yahoo! Research: "Web Spam Collections"*. Disponível em http://barcelona.research.yahoo.net/Webspam/datasets/.

[3] *Yahoo! Research: "Web Spam Collections"*. Disponível em http://barcelona.research.yahoo.net/Webspam/datasets/.

como o C4.5, RIPPER, PART e floresta aleatória.

(Largillier & Peyronnet 2012) usaram uma abordagem diferente para o problema e apresentaram quatro métodos de agrupamento de nós para a diminuição dos efeitos causados por *Web spam* no algoritmo *PageRank*. Esses métodos, chamados respectivamente de *Tech1*, *Tech2*, *Tech3* e *Tech4*, têm a finalidade de construir agrupamentos de nós, usando a estrutura de *links* das páginas *Web*, que, depois, são considerados um único nó no cálculo do *PageRank*. Para os experimentos, eles usaram a base de dados WEBSPAM-UK2007. Os resultados obtidos indicaram que os métodos *Tech2*, *Tech3* e *Tech4* diminuiram o *PageRank* das páginas *spam* e aumentaram o *PageRank* das páginas *ham*. Eles também observaram que o método de agrupamento *Tech2* é capaz de separar as páginas *ham* das páginas *spam* e, por isso, além de poder ser usado para diminuir a influência de *Web spam* no algoritmo *PageRank*, pode ser usado também como técnica de detecção dessa "praga" virtual.

Na Seção 5.7 são apresentados outros trabalhos da literatura que apresentam propostas para o combate de *Web spamming*, mas que usam em seus experimentos as mesmas bases de dados e métricas de desempenho que foram adotadas nessa dissertação.

Capítulo 2

Métodos de aprendizado de máquina

Apesar da existência de trabalhos cujo intuito é filtrar *Web spam*, ainda não há um consenso na literatura se a aplicação de técnicas de aprendizado de máquina é realmente eficaz na detecção automática de *Web spam*. Os artigos existentes avaliam diferentes métodos empregando bases de dados e metodologias distintas, fato este que dificulta uma comparação direta entre os métodos. Tendo isso em vista, foram avaliados neste trabalho diversos métodos de classificação bem conhecidos e largamente empregados na literatura.

Nesse capítulo, é oferecida, de maneira sucinta, uma introdução dos métodos avaliados nesse trabalho: redes neurais artificiais (RNAs), máquinas de vetores de suporte (SVM – *support vector machines*), métodos baseados em árvores (C4.5 e floresta aleatória), IBK, *boosting* adaptativo (*AdaBoost – adaptive boosting*), *bagging* e *LogitBoost*, *OneR* e *naive Bayes*. A escolha de tais métodos reside no fato de terem sido avaliados e listados como os melhores métodos de classificação e mineração de dados atualmente disponíveis (Wu, Kumar, Quinlan, Ghosh, Yang, Motoda, McLachlan, Ng, Liu, Yu, Zhou, Steinbach, Hand & Steinberg 2008).

As RNAs não fazem parte da lista proposta por (Wu et al. 2008), mas foram escolhidas para serem avaliadas devido a sua alta capacidade de generalização. A técnica de floresta aleatória também foi escolhida, mesmo não estando na lista, pois ela faz uma combinação de árvores de decisão. Logo, como o método C4.5 é um algoritmo de árvores de decisão e está na lista dos melhores métodos, acredita-se que a combinação de árvores de decisão também possa gerar bons resultados. Por fim, também foram feitos experimentos com o método *OneR*, pois ele tem um baixo custo computacional e é um dos algoritmos de aprendizado de máquina mais simples da literatura. Portanto, o seu desempenho pode ser usado para analisar o desempenho obtido por algoritmos mais complexos.

2.1 Redes neurais artificiais

Redes neurais artificiais (RNAs) são sistemas massivamente paralelos e distribuídos, constituídos de unidades de processamento simples, chamadas neurônios, que têm capacidade de armazenar conhecimento e utilizá-lo. Nesse sistema, o conhecimento é adquirido por um processo chamado treinamento ou aprendizado que adapta as forças de conexões entre os neurônios, chamadas de pesos sinápticos, aos estímulos recebidos pelo ambiente (Haykin 1998).

Segundo (Braga, Ludermir & Carvalho 2007), as RNAs conseguem aprender através de um conjunto reduzido de exemplos e generalizar o conhecimento adquirido, sendo capazes de dar respostas coerentes para dados desconhecidos. Além disso, também são capazes de extrair informações não apresentadas de forma explícita e possuem grande capacidade de auto-organização.

Um modelo básico de RNA possui os seguintes componentes (Braga et al. 2007):

- *Conjunto de sinapses:* conexões entre os neurônios da RNA. Cada uma delas possui um peso sináptico.
- *Integrador:* realiza a soma dos sinais de entrada da RNA, ponderados pelos pesos sinápticos.
- *Função de ativação:* restringe a amplitude do valor de saída de um neurônio.
- *Bias:* valor aplicado externamente a cada neurônio e tem o efeito de aumentar ou diminuir a entrada líquida da função de ativação.

2.1.1 Rede neural artificial perceptron de múltiplas camadas

Uma RNA perceptron de múltiplas camadas (MLP – *multilayer perceptron*) é uma rede do tipo *perceptron* que possui um conjunto de unidades sensoriais que formam a camada de entrada, uma ou mais camadas intermediárias de neurônios computacionais e uma camada de saída (Haykin 1998). Por padrão, o seu treinamento é supervisionado e usa o algoritmo *backpropagation* (retropropagação do erro), que tem a função de encontrar as derivadas da função de erro com relação aos pesos e bias da rede (Haykin 1998).

Esse algoritmo pode ser resumido em duas etapas: *forward* e *backward*. Na etapa *forward*, os dados são apresentados às unidades de entrada e o sinal é propagado pela rede, camada a camada. Então, um conjunto de saídas é produzido como resposta da rede. Na etapa *backward*, inicia-se a derivação do algoritmo *backpropagation*, a partir da camada de saída. Nessa fase, os pesos sinápticos são ajustados por meio de uma regra de correção de erro, que calcula a diferença entre a saída fornecida pela rede e a saída desejada e propaga essa diferença, que pode ser chamada de sinal de erro, da camada de saída para a camada de entrada da rede (Bishop 1995, Haykin 1998, Braga et al. 2007). Segundo (Braga et al. 2007), o algoritmo *backpropagation* se baseia no método do gradiente descendente para ajustar os pesos sinápticos de forma a diminuir o erro entre a resposta desejada e a saída fornecida pela rede.

Algoritmo de Levenberg-Marquardt

O algoritmo de Levenberg-Marquardt é um método de treinamento de redes neurais de múltiplas camadas que combina a estabilidade do método do gradiente descendente e a velocidade do método de Newton . Ele é considerado um método de segunda ordem, assim como os métodos gradiente conjugado e quase-Newton, pois estima informações sobre a derivada segunda da função de erro (Bishop 1995).

Assim como o método Quase-Newton, o algoritmo de Levenberg-Marquardt pode tornar-se computacionalmente impraticável em redes muito grandes, pois necessita armazenar uma matriz

quadrada cuja dimensão é da ordem do número de conexões da rede neural. Porém, ele é mais eficiente do que o método do gradiente conjugado em redes com algumas centenas de pesos e possui menor custo computacional do que o método Quase-Newton (Hagan & Menhaj 1994, Liu 2010).

Maiores detalhes sobre o algoritmo de Levenberg-Marquardt podem ser encontrados em (Bishop 1995, Hagan & Menhaj 1994, Haykin 1998).

2.1.2 Mapa auto-organizável de Kohonen

O mapa auto-organizável de Kohonen (SOM – *Kohonen's self-organizing map*) (Kohonen 1990) é uma RNA com aprendizado competitivo e não supervisionado. Seu principal objetivo é transformar, de maneira topologicamente ordenada, um padrão de entrada de dimensão qualquer em um padrão de resposta de um mapa unidimensional ou bidimensional (Haykin 1998).

O algoritmo de treinamento de uma rede neural SOM pode ser resumido em dois estágios: competição e cooperação. No primeiro, um padrão de entrada x_j é escolhido aleatoriamente e é calculada a similaridade entre esse padrão e todos os neurônios da rede. O neurônio mais similar a esse padrão será ativado. No estágio da cooperação, o peso sináptico w_{id} que liga o neurônio vencedor à entrada x_i é atualizado. Os pesos sinápticos que estão relacionados aos seus neurônios vizinhos também são atualizados por: $w_i(t+1) = w_i(t)+\alpha(t)h(t)(x_i-w_i(t))$, onde t é o número da época/iteração de treinamento, $w_i(t+1)$ é o vetor de novos pesos, $w_i(t)$ é o vetor de pesos no instante t, α é a taxa de aprendizado, $h(t)$ é a função de vizinhança no instante t e x_i é o padrão de entrada da rede. A função de vizinhança $h(t)$ determina a vizinhança topológica ao redor do neurônio vencedor definido pelo raio de vizinhança σ. A amplitude dessa vizinhança decresce monotonicamente com o aumento da distância lateral entre um neurônio vizinho e o neurônio vencedor (Kohonen 1990, Haykin 1998).

Os estágios de competição e de cooperação são realizados para todos os padrões de entrada. Então, o raio de vizinhança σ e a taxa de aprendizado α são atualizados por meio de alguma função monotônica decrescente no tempo. Esse processo se repete até que algum critério de parada seja satisfeito (Kohonen 1990).

2.1.3 Quantização vetorial por aprendizagem

O método de quantização vetorial por aprendizagem (LVQ - *learning vector quantization*) consiste em uma RNA com aprendizado supervisionado que tem o objetivo de melhorar a qualidade da região de decisão do classificador, ajustando o mapa de características através do uso de informações sobre as classes (Haykin 1998).

Segundo (Kohonen 1990) a rede SOM pode ser utilizada para inicializar o mapa de características, definindo o ajuste dos vetores de peso w_{ij}. Depois disso, cada neurônio da rede neural recebe o rótulo da classe em que ele é mais ativado. Então, o algoritmo LVQ pode ser executado. O processo de treinamento desse algoritmo é similar ao apresentado para a rede SOM, porém ele não usa relações de vizinhança. Portanto, a cada iteração, é verificado se o rótulo da classe do vetor de entrada x_i é igual ao rótulo do neurônio vencedor. Então os pesos da rede são

atualizados da seguinte forma:

$$w_{id}(t+1) = \begin{cases} w_{id}(t) + \alpha(t)(x_i - w_{id}(t)), \text{ classes iguais} \\ w_{id}(t) - \alpha(t)(x_i - w_{id}(t)), \text{ classes diferentes} \end{cases}$$

onde α é a taxa de aprendizado, id é o índice do neurônio vencedor e t é o número da iteração atual.

2.1.4 Rede neural de função de base radial

Uma rede neural de função de base radial (RBF - *radial basis function*), em sua forma mais básica, possui três camadas. A primeira é a camada de entrada que possui unidades sensoriais que conectam a rede ao seu ambiente. A segunda é a camada intermediária ou escondida que é formada por um conjunto de neurônios que utilizam funções de base radial. Essa camada aplica uma transformação não linear do espaço de entrada para o espaço escondido. Já, a terceira é a camada de saída que é linear e fornece uma resposta da rede à ativação aplicada à camada de entrada (Haykin 1998).

A função de ativação mais usada nas redes neurais RBF é a Gaussiana, definida por:

$$h(x) = \exp\left(-\frac{\|x_i - c_j\|^2}{2\gamma^2}\right), \gamma > 0,$$

onde x_i é o vetor de entrada, c_j é o centro e γ é uma parâmetro de suavização (Bishop 1995, Orr 1996).

O procedimento de treinamento de uma rede neural RBF é realizado em duas etapas. Na primeira, os parâmetros que regem as funções de base da camada escondida são determinados através de algum método de treinamento não-supervisionado, como o algoritmo k-médias (k-means). Na segunda fase de treinamento, são ajustados os pesos da camada de saída por meio de algum método que resolva o problema de minimização do erro, tais como o método dos mínimos quadrados, o método da regra delta ou o método da matriz pseudo-inversa (Bishop 1995, Haykin 1998).

Para maiores informações sobre as redes neurais RBF consulte (Bishop 1995, Haykin 1998, Orr 1996).

2.2 Máquinas de vetores de suporte

Máquinas de vetores de suporte (SVM – *support vector machines*) (Cortes & Vapnik 1995) são um método de aprendizado de máquina que pode ser usado para problemas de classificação e regressão, além de outras tarefas de aprendizagem (Haykin 1998, Chang & Lin 2011). Elas foram conceitualmente implementadas seguindo a ideia de que vetores de entrada são não-linearmente mapeados para um espaço de atributos de alta dimensão. Nesse espaço, é construída uma superfície de decisão que permite distinguir as classes dos exemplos de entrada.

Segundo (Haykin 1998), a ideia principal do SVM, no contexto de dados linearmente separáveis, é construir uma superfície de decisão por meio de um hiperplano ótimo com máxima margem de separação entre os dados de classes diferentes. Já no contexto de dados não-separáveis,

o objetivo do SVM é construir um hiperplano ótimo que minimize a probabilidade de erro de classificação em relação ao conjunto de treinamento.

Para conseguir separar dados linearmente ou não-linearmente separáveis, um dos principais elementos usados pelo método SVM é uma função de *kernel*. Através dela, o SVM constrói uma superfície de decisão que é não-linear no espaço de entrada, mas é linear no espaço de atributos (Haykin 1998). Algumas funções de *kernel* que podem ser utilizadas no SVM são (Haykin 1998, Hsu, Chang & Lin 2003):

- linear: $k(x_i, x_j) = x_i^T x_j$;
- *radial basis function* (RBF): $k(x_i, x_j) = \exp(-\frac{1}{2\gamma^2}\|x_i - x_j\|^2), \gamma > 0$;
- polinomial: $k(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$;
- sigmoidal: $k(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$.

Nas funções de *kernel* apresentadas acima, o parâmetro γ controla a forma da superfície de decisão. Pequenos valores de γ torna a superfície de decisão quase linear, enquanto o aumento do valor desse parâmetro aumenta a flexibilidade da superfície de decisão. O parâmetro r controla o limiar de deslocamento dos *kernels* polinomial e sigmoidal. O parâmetro d é grau do *kernel* polinomial. Os parâmetros γ, r e d devem ser especificados a priori pelo usuário.

Para a escolha dos parâmetros do SVM, a técnica recomendada em (Hsu et al. 2003) é a *grid search* (busca em grade). Essa técnica consiste em fazer uma busca exaustiva, combinando valores de determinados intervalos para cada parâmetro que se deseja otimizar. Para cada combinação de parâmetros, o SVM deve ser executado. Ao fim da *grid search*, a combinação de parâmetros que gerar melhor resultado é escolhida.

Considerando o SVM com *kernel* RBF, em que deve-se definir o parâmetro de regularização C e o parâmetro γ, (Hsu et al. 2003) propõem testar as seguintes sequências exponenciais: $C = 2^{-5}, 2^{-4}, 2^{-3}..., 2^{15}$ e $\gamma = 2^{-15}, 2^{-14}..., 2^{3}$.

2.3 C4.5

O C4.5 (Quinlan 1993) é um dos mais clássicos algoritmos de árvores de decisão e trabalha tanto com atributos categóricos quanto contínuos. Além disso, ele permite o uso de atributos desconhecidos, desde que sejam representados por "?". O C4.5 usa um método de dividir e conquistar para aumentar a capacidade de predição das árvores de decisão. Dessa forma, um problema é dividido em vários sub-problemas, criando-se sub-árvores no caminho entre a raiz e as folhas da árvore de decisão.

O C4.5 gera um classificador na forma de árvores de decisão que em sua estrutura possuem folhas, que indicam alguma classe do problema, e nós de decisão, que especificam algum teste que deve ser realizado sobre um valor de atributo (Quinlan 1993).

2.4 IBK

O algoritmo IBK é um algoritmo de aprendizado baseado em instâncias (IBL – *instance-based learning*) (Aha, Kibler & Albert 1991). Esse tipo de algoritmo é derivado do método de classificação k-vizinho mais próximo (KNN – *k-nearest neighbor*). Porém, este último é um algoritmo não-incremental e tem como objetivo principal manter uma consistência perfeita com o conjunto inicial de treinamento. Já o algoritmo do tipo IBL é incremental e tem como objetivo maximizar a acurácia sobre novas instâncias do problema (Aha et al. 1991).

O algoritmo IBL usa uma função que mapeia padrões de um problema para categorias, então, a saída inicial do classificador é uma categoria. Logo, dado um padrão do conjunto de dados, a classificação produzida pelo algoritmo será o valor previsto para a categoria a qual este padrão pertence. Então, uma das funções principais do algoritmo IBL é a função de similaridade, que calcula a similaridade entre o padrão de treinamento i e outros padrões que já estão incluídos em alguma categoria. Outra função importante é a função de classificação. Ela recebe o resultado da função de similaridade e os registros de desempenho da classificação dos padrões que possuem a mesma categoria que este padrão i e gera uma classificação para o mesmo. Depois disso, o atualizador de categorias é chamado e ele mantém os registros sobre o desempenho da classificação e decide quais padrões devem ser incluídos em cada categoria (Aha et al. 1991).

Assim como no método KNN, no método IBK existe uma função de similaridade que obtém um valor numérico obtido pelo cálculo da distância euclidiana. Então a classificação gerada para um padrão i será influenciada pelo resultado da classificação dos seus k-vizinhos mais próximos, pois padrões similares devem ter classificações similares (Aha et al. 1991, Witten, Frank & Hall 2011).

2.5 Métodos de agregação de classificadores

Métodos de agregação ou combinação de múltiplos classificadores, também conhecidos com métodos *ensemble*, têm sido cada vez mais usados na literatura para aumentar a eficiência do processo de aprendizado em tarefas de classificação de padrões. Esses métodos tem como objetivo superar as limitações individuais dos classificadores através da combinação de suas predições (Witten et al. 2011).

Os métodos de agregação podem adotar diferentes estratégias para gerar múltiplos classificadores. Algumas dessas estratégias incluem a manipulação dos conjuntos de treinamento, dos atributos extraídos das amostras do problema e dos algoritmos de aprendizado de máquina (Witten et al. 2011, Kuncheva 2004). Diversos métodos de agregação de classificadores já foram propostos na literatura, mas entre eles, alguns que se destacam são os métodos *bagging*, *AdaBoost*, *LogitBoost* e floresta aleatória.

2.5.1 *Boosting* adaptativo

O método *boosting* adaptativo (*AdaBoost* – *adaptive boosting*) (Freund & Schapire 1996, Freund & Schapire 1997) é um algoritmo de *boosting* amplamente utilizado para problemas de

classificação. Em geral, assim como qualquer método de *boosting*, ele faz uma combinação de classificadores, que gera resultados melhores do que os gerados por um classificador isolado. No método *boosting* os classificadores se complementam e cada novo modelo é influenciado pelos modelos construídos em etapas anteriores. Isso estimula que novos modelos se tornem especialistas em casos que foram classificados incorretamente em etapas anteriores (Witten et al. 2011, Bishop 2006).

O algoritmo *AdaBoost* começa atribuindo pesos de igual valor para todas as amostras de treinamento. Depois, em cada iteração $t = 1, ..., k$, ele chama o algoritmo de aprendizado para formar um modelo m_t para classificar as amostras e atualiza os pesos das mesmas da seguinte forma: ele diminui os pesos das amostras corretamente classificadas e aumenta os pesos das amostras incorretamente classificadas. Ao final de cada iteração, os pesos refletem qual a frequência em que cada amostra foi incorretamente classificada. A intenção é que seja construído um modelo que se concentre mais nas amostras que possuem maior peso (Witten et al. 2011).

Em cada iteração, o modelo m_t gerado pelo algoritmo de aprendizado recebe um peso h_t que reflete o quão bem ele atuou nas predições das amostras que lhe foram apresentadas durante o processo de treinamento. Então, quando uma amostra desconhecida é apresentada, cada classe recebe como voto a soma dos pesos dos modelos que votaram nela. Portanto, a amostra é atribuída a classe com maior voto (Witten et al. 2011, Han & Kamber 2006).

2.5.2 *LogitBoost*

O método *LogitBoost* (Friedman, Hastie & Tibshirani 1998) é uma versão estatística do método de *boosting* e, segundo (Witten et al. 2011), possui algumas semelhanças com o método *AdaBoost*. Porém ele otimiza a probabilidade de ocorrência de uma classe, enquanto o *AdaBoost* otimiza uma função de custo exponencial. Em (Friedman et al. 1998), esse método é definido como um algoritmo para montagem de modelos aditivos de regressão logística.

De acordo com (Friedman et al. 1998), um dos exemplos de uso bem sucedido do algoritmo *LogitBoost* é na indução de árvores com modelos de regressão logística linear nas folhas, que são chamadas de árvores de modelo logístico.

2.5.3 *Bagging*

O *bagging* (abreviação de *bootstrap aggregating*) (Breiman 1996) é um método de geração de múltiplas versões de um classificador que são combinadas para a obtenção de um classificador agregado. O processo adotado pelo método de *bagging* é semelhante ao do método de *boosting*, porém de acordo com (Witten et al. 2011), diferentemente do que ocorre no segundo método, no *bagging*, os diferentes modelos de classificadores recebem o mesmo peso na geração de uma predição.

No método *bagging*, quando a predição do classificador deve ser numérica, é feita uma média sobre os resultados de todos os modelos de classificadores. Por outro lado, se a predição deve ser uma classe, é feita uma votação e a classe com maior pontuação será escolhida para representar o padrão (Breiman 1996).

Para gerar várias versões de um classificador, o método de *bagging* usa um *bootstrap* no conjunto de treinamento. Em outras palavras, o conjunto de treinamento é utilizado para gerar n novos conjuntos de treinamento de tamanhos iguais tomados aleatoriamente e com reposição do conjunto original. Logo, alguns desses conjuntos de treinamento podem ter padrões iguais e alguns padrões podem não ser nunca escolhidos. É essa perturbação no conjunto de treinamento que ao mesmo tempo que pode mudar significativamente o classificador construído, melhora a precisão do método *bagging* (Breiman 1996, Witten et al. 2011, Bishop 2006).

2.5.4 Floresta aleatória

Uma floresta aleatória (*random forest*) (Breiman 2001) é uma combinação de árvores de decisão, em que cada árvore depende dos valores de vetores aleatórios amostrados de forma independente e distribuídos igualmente para todas as árvores na floresta.

Na construção de uma floresta aleatória, para a k-ésima árvore de decisão, um vetor aleatório Θ é gerado, independente dos vetores aleatórios $\Theta_1, ..., \Theta_{k-1}$ gerados no passado, porém usando a mesma distribuição. A árvore é cultivada usando o vetor aleatório gerado e um conjunto de treinamento. Desta forma, o resultado é a criação de um classificador $h(x, \Theta_k)$, onde x é um vetor de entrada. Então, para cada vetor aleatório gerado, uma nova árvore de decisão é construída. Depois que um determinado número de árvores são geradas, cada uma lança um voto para uma classe do problema, considerando o vetor de entrada x. Consequentemente, a classe mais votada será escolhida na predição do classificador.

2.6 *OneR*

O método *OneR* ou 1R (abreviação de 1-*rules*) (Holte 1993) pode ser considerado uma árvore de decisão com 1 nível, pois gera um conjunto de regras, uma para cada atributo do conjunto de dados e classifica uma amostra com base em um único atributo. O atributo escolhido é o correspondente à regra que produz o menor erro.

Logo, o funcionamento desse método de classificação pode ser expresso pelos seguintes passos (Witten et al. 2011):

1. selecione um atributo do conjunto de treinamento;

2. encontre a classe mais frequente para este atributo e crie uma regra que adote esta classe como o valor correto para este atributo;

3. calcule a porcentagem de erro dessa regra, ou seja, o número de amostras de treinamento cuja classe seja diferente da classe adotada pela regra deste atributo;

4. repita os passos anteriores para todos os atributos;

5. escolha a regra que possui a menor porcentagem de erro.

2.7 *Naive* Bayes

O método *naive* Bayes (John & Langley 1995) é um classificador probabilístico simples baseado no teorema de Bayes. Então, supondo que existem k classes $c_1, c_2, ..., c_k$ e uma amostra X pertencente a uma das classes desse conjunto e representada por m atributos $x_1, x_2, ..., x_m$, o método *naive* Bayes aplica o teorema de Bayes para determinar a probabilidade de cada classe dada a amostra:

$$P(c_i|X) = \frac{P(X|c_i)P(c_i)}{P(X)},$$

onde $P(c_i)$ é a probabilidade a *priori* da classe c_i, $P(X)$ é a probabilidade a *priori* da amostra X e $P(X|c_i)$ é a probabilidade da amostra X dada a classe c_i. Depois de calcular essas probabilidades, o algoritmo atribui a amostra X para a classe que possui a maior probabilidade global (Han & Kamber 2006, John & Langley 1995).

Uma das características do classificador *naive* Bayes é que ele assume que os atributos contribuem de forma independente para formar probabilidades estimativas da ocorrência de uma determinada classe do problema. Por isso, esse classificador é denominado ingênuo (*naive*). Diante disso, tem-se que:

$$P(X|c_i) = \prod_{j=1}^{m} P(x_j|c_i),$$

onde $P(x_j|c_i)$ representa a probabilidade condicional armazenada com cada classe (Langley 1993). De acordo com (Langley & Sage 1994) esta abordagem simplifica bastante o cálculo das probabilidades das classes dada uma determinada amostra.

Segundo (Witten et al. 2011), uma das desvantagens do método *naive* Bayes é que com a abordagem usada por ele, onde se considera que os atributos são condicionalmente independentes, a existência de redundância nos mesmos pode distorcer o processo de aprendizado.

Capítulo 3

Base de Dados e Vetores de Atributos

Neste capítulo, são apresentadas as bases de dados utilizadas nos experimentos realizados nesse trabalho. Também são apresentados os detalhes dos atributos baseados no conteúdo, bem como dos atributos baseados nos *links* e de sua transformação, que resultaram nos atributos baseados nos *links* transformados. Além disso, para facilitar a compreensão de como foram extraídos os atributos baseados nos *links*, este capítulo apresenta também algumas noções básicas sobre grafo *Web*.

3.1 Bases de dados

Para dar credibilidade aos resultados, foram feitos experimentos com duas bases de dados públicas, de grande porte e compostas de amostras reais de *Web spam*: as coleções WEBSPAM-UK2006 e WEBSPAM-UK2007[1]. A primeira coleção possui um conjunto de 77,9 milhões de páginas hospedadas em 11.000 *hosts*, enquanto que a segunda é composta por 105.896.555 páginas hospedadas em 114.529 *hosts*. Todas as páginas que compõe as duas bases de dados foram coletadas de domínios do Reino Unido. As duas coleções foram utilizadas nas edições do evento *Web Spam Challenge*[2] que trata-se de uma competição de técnicas de detecção de *Web spam*. A primeira coleção foi usada na edição de 2007 dessa competição e a segunda na edição de 2008.

3.2 Vetores de atributos

Assim como nas competições *Web Spam Challenge*, nos experimentos realizados nesse trabalho, foram utilizados três conjuntos de vetores de atributos (também podem ser chamados de vetores de características ou vetores de *features*) pré-computados, extraídos de cada base de dados e fornecidos pelos organizadores do evento aos times participantes. O primeiro conjunto é composto por 96 atributos baseados no conteúdo (Castillo et al. 2007), o segundo é composto por 41 atributos baseados nos *links* (Becchetti et al. 2006b) e o terceiro é composto por 138

[1] *Yahoo! Research: "Web Spam Collections"*. Disponível em http://barcelona.research.yahoo.net/Webspam/datasets/.

[2] Disponível em http://Webspam.lip6.fr/

atributos baseados nos *links* transformados (Castillo et al. 2007), que são simples combinações ou operações logarítmicas sobre os atributos baseados nos *links*.

Os atributos fornecidos nas competições *Web Spam Challenge* foram extraídos dos *hosts* das bases de dados WEBSPAM-UK2006 ou WEBSPAM-UK2007. Um *host* é um grupo de páginas de um mesmo domínio. Então, por exemplo, sejam duas páginas *Web* que possuem, respectivamente, os seguintes endereços:

- *www.meusite.com.br/contato.html*;
- *www.meusite.com.br/artigos/congressos.html*.

Essas páginas *Web* pertencem ao domínio *www.meusite.com.br* e junto com as outras páginas desse domínio, formam um *host*.

3.2.1 Vetores de atributos baseados no conteúdo

Os vetores de atributos baseados no conteúdo das páginas *Web* são formados pela extração dos seguintes atributos propostos por (Castillo et al. 2007):

1. Número de palavras na página, número de palavras no título e média do tamanho das palavras: estes atributos consideram apenas as palavras visíveis de uma página e que são formadas apenas por caracteres alfanuméricos.

2. Fração do texto âncora: fração do número de palavras no texto âncora pelo número total de palavras no texto visível na página. Texto âncora é uma anotação em um *link* que descreve o conteúdo da página de destino do mesmo.

3. Fração do texto visível: fração do número de palavras visíveis pelo número de palavras na página (inclui *tags* html e outros textos invisíveis).

4. Taxa de compressão: o texto visível da página foi compactado usando o programa `bzip`. Então a taxa de compressão é a relação entre o tamanho do texto compactado e tamanho do texto não compactado.

5. Precisão e revocação da coleção: foram encontradas as k palavras mais populares dos dados coletados. Então, a precisão da coleção é dada pela fração de palavras em uma página que aparecem no conjunto de termos mais populares e revocação é a fração de termos mais populares que aparecem na página. Foram extraídos 4 atributos em relação à precisão e 4 em relação à revocação, usando $k = 100$, 200, 500 e 1000.

6. Precisão e revocação da consulta: foram considerados os q termos mais populares nas consultas. Dessa forma, a precisão da coleção é dada pela fração de palavras em uma página que aparecem no conjunto de termos populares das consultas e revocação é a fração de termos populares das consultas que aparecem na página. Foram extraídos 4 atributos em relação à precisão e 4 em relação à revocação, usando $q = 100$, 200, 500 e 1000.

7. Probabilidade de trigrama independente: um trigrama é um conjunto de três termos (*tokens*) consecutivos. Seja p_w a probabilidade de distribuição de trigramas em uma página, $T = w$ o conjunto de todos os trigramas em uma página e $k = |T(p)|$ o número de trigramas diferentes. Então, a probabilidade de ocorrência de trigramas diferentes é uma medida de independência da distribuição de trigramas e é definida por $\frac{1}{k} \sum_{w \in T} \log p_w$.

8. Entropia de trigramas: entropia é a medida de compressibilidade de uma página, mas nesse caso considera-se apenas a distribuição de trigramas p_w, sendo calculada por $H = -\sum_{w \in T} p_w \log p_w$.

No total, 24 atributos extraídos do conteúdo são usados para discriminar os *hosts*. Contudo, para cada *host* são consideradas essas informações em relação à página inicial, à página com maior *PageRank*, à média dos valores para todas as páginas e ao desvio padrão de todas as páginas (Castillo et al. 2007). Logo, cada *host* é associado a um vetor composto por 96 atributos.

3.2.2 Grafo *Web*

Conforme foi mencionado na Seção 1.1, alguns métodos de classificação de páginas *Web* utilizados pelos motores de busca, tal como o algoritmo *PageRank* do Google, usam grafo *Web* (Figura 3.1) para determinar a relevância dos *sites*.

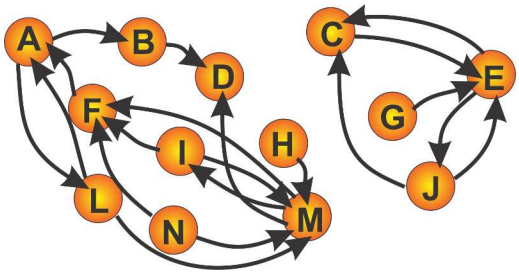


Figura 3.1: Grafo *Web*.

Um grafo *Web* é um grafo direcionado ou dígrafo, em que seus nós representam páginas *Web*. Cada aresta que liga dois nós do grafo *Web* representa um *link* entra duas páginas. Mesmo que uma página possua vários *links* direcionados para outra página, considera-se uma única ligação. Então tomando como exemplo os nós A e L do grafo *Web* apresentado na Figura 3.1, observa-se que existe uma aresta (*link*) que sai do nó A para o nó L, o que pode ser denotado por $A \rightarrow L$. Esta aresta será chamada de aresta de saída de A e aresta de entrada de L ou *forward link* de A e *backlink* de L. Também existe uma aresta que sai do nó L para o nó A, o que pode ser representado por $L \rightarrow A$. Esta aresta será chamada de aresta de saída de L e aresta de entrada de A ou *forward link* de L e *backlink* de A. Se for considerado que o grafo *Web* é um grafo não direcionado, então uma aresta de entrada e uma aresta de saída que liga os mesmos nós devem ser consideradas uma única aresta. Logo a ligação entre os nós A e L pode ser denotada por AL ou LA.

Tomando como exemplo o nó M do grafo *Web* da Figura 3.1, é possível observar que ele possui quatro arestas de entrada: $L \rightarrow M$, $N \rightarrow M$, $I \rightarrow M$ e $H \rightarrow M$. Por isso, ele possui um

grau de entrada igual a 4. Ele também possui 3 arestas de saída: $M \rightarrow D$, $M \rightarrow I$, e $M \rightarrow F$. Logo, ele possui grau de saída igual a 3. Se for considerado que o grafo *Web* é um grafo não direcionado, então pode-se dizer também que o nó M possui 6 arestas ou está ligado a 6 nós: LM, NM, IM, FM DM e HM. Diante disso, ele possui grau 6.

Outro conceito importante é o de vizinhança. Dois nós são considerados vizinhos ou adjacentes se eles foram unidos por uma aresta, tanto de saída quanto de entrada. Além disso, existem os conceitos de vizinhança de entrada e vizinhança de saída. Um nó U faz parte da vizinhança de entrada de um nó V se o nó U possui uma aresta de saída para o nó V, isto é, $U \rightarrow V$. Um nó U faz parte da vizinhança de saída de um nó V se o nó V possui uma aresta de saída para o nó U, isto é, $V \rightarrow U$. Diante disso, usando como exemplo o grafo *Web* da Figura 3.1, suas listas de vizinhanças são apresentadas na Tabela 3.1.

Tabela 3.1: Listas de vizinhanças do grafo *Web* apresentado na Figura 3.1.

(a) Vizinhança

Vértices	Vizinhos
A	B, F, L
B	A, D
C	E, J
D	B, M
E	C, G, J
F	A, N, I, M
G	E
H	M
I	M, F
J	C, E
L	A, M
M	L, N, I, F, D, H
N	M, F

(b) Vizinhança de entrada

Vértices	Vizinhos de entrada
A	F, L
B	A
C	E, J
D	M
E	C, G, J
F	N, I, M
G	–
H	–
I	M
J	E
L	A
M	L, N, I, H
N	–

(c) Vizinhança de saída

Vértices	Vizinhos de saída
A	B, L
B	D
C	E
D	–
E	C, J
F	A
G	E
H	M
I	M, F
J	C, E
L	A, M
M	D, I, F
N	M, F

No estudo de grafos *Web*, é importante também conhecer o conceito de distância. A distância entre dois nós é o comprimento ou número de arestas do menor caminho que os liga. Então, usando como exemplo os nós M e F do grafo *Web* apresentado na Figura 3.1, existem as seguintes opções de sair do nó M para chegar ao nó F: $MLAF$, $MDBAF$, MNF, MIF e MF. Entre as opções apresentadas, a que possui o menor número de arestas é MF, logo, a distância entre os nós M e F é igual a 1.

Através do conceito de distância entre dois nós, é possível descobrir a vizinhança de um nó que tenha distância n ou que seja menor ou igual a n. Então, tomando como exemplo o nó M, pode-se deduzir que seus vizinhos que estão a uma distância 3 são: B e A. Já, seus vizinhos que estão a uma distância menor ou igual a 3 são: A, B, D, F, H, I, L, M e N. Também pode-se determinar por meio do conceito de vizinhança, os vizinhos de entrada e de saída de um nó que estão a uma distância n. Então, considerando novamente o nó M, pode-se concluir que não existe nenhum vizinho de entrada com distância 4, mas existe um vizinho de saída, o nó B. Existe também um vizinho de saída com distância 5, o nó D. Já, com distância 3, não existem vizinhos de entrada e nem de saída. Por outro lado, se for considerada uma distância 2, o nó A é vizinho de entrada de M e os nós A e F são vizinhos de saída de M.

3.2.3 Vetores de atributos baseados nos *links*

Os atributos apresentados a seguir foram propostos por (Becchetti et al. 2006b) e extraídos do grafo *Web* formado pelas amostras presentes nas bases de dados WEBSPAM-UK2006 e WEBSPAM-UK2007:

1. Página inicial é a página com maior *PageRank*: a primeira característica é setada como 1 se a página inicial tiver o maior *PageRank* do *host*, senão é setada como 0.
2. Coeficiente de assortatividade: corresponde a divisão do grau do nó (*página Web*) pela média dos graus dos nós vizinhos.
3. Média do grau de entrada dos vizinhos de saída do nó.
4. Média do grau de saída dos vizinhos de entrada do nó.
5. Grau de entrada do nó.
6. Número de vizinhos com distância n do nó. Os valores usados foram $n = 2, 3, 4$.
7. Grau de saída do nó.
8. *PageRank* do nó.
9. Desvio padrão do *PageRank* dos vizinhos de entrada do nó.
10. Reciprocidade: divisão do número de arestas de saída, direcionados para vizinhos de entrada do nó, pelo número de arestas de saída do nó. Se o nó não possui arestas de saída, então o valor de reciprocidade é 0.
11. Número de diferentes *hosts* vizinhos com distância n. Os autores usaram $n = 1, 2, 3, 4$. Nesse caso, os *hosts* vizinhos são obtidos da mesma forma que nós (páginas) vizinhos. A diferença é que os nós de um mesmo *host* devem ser considerados um único nó.
12. *PageRank* truncado (*Truncated PageRank*) do nó usando uma distância de truncamento n: o *PageRank* truncado (Becchetti et al. 2006b) é uma algoritmo semelhante ao *PageRank*, mas que penaliza os nós que obtém grande parte do seu *PageRank* por meio de *arestas* de entrada vindos de nós muito próximos. Isso significa que no cálculo da relevância do nó (página), os vizinhos de entrada que estão a uma distância n não irão contribuir. Os autores usaram $n = 1, 2, 3, 4$.
13. *TrustRank* do nó: o *TrustRank* é um algoritmo proposto por (Gyongyi et al. 2004) que mede o grau de confiança de uma página *Web* com base no grau de confiança dos seus vizinhos de entrada. Logo, ter vizinhos de entrada a pequenas distâncias e que sejam considerados confiáveis, aumenta o grau de confiança da página.

Entre os itens 2 e 13, há um total de 20 atributos. Eles foram extraídos da página inicial e da página com maior *PageRank*, resultando em 40 atributos, que somados ao primeiro item da lista, fornecem um total de 41 atributos que podem ser utilizados para discriminar os *hosts* como *spam* ou *ham*.

3.2.4 Vetores de atributos baseados nos *links* transformados

Os vetores de atributos baseados nos *links* das páginas *Web*, apresentados na seção anterior, foram alterados por (Castillo et al. 2007) e deram origem a novos atributos que serão chamadas de atributos baseados nos *links* transformados. Esses atributos são o resultado de operações logarítmicas e simples combinações dos atributos baseados nos *links*, conforme é detalhado a seguir:

1. Página inicial é a página com maior *PageRank*: a primeira característica é setada como 1 se a página inicial tiver o maior *PageRank* do *host*, senão é setada como 0.

2. Foi calculado o logaritmo de quase todos os atributos baseados nos *links*, exceto do primeiro, descrito na Seção 3.2.3.

3. Os atributos baseados nos *links* apresentados nos itens 5, 6, 7, 9, 11, 12 e 13 da Seção 3.2.3 foram extraídos da página *Web* e divididos pelo valor de seu *PageRank*. Em seguida, foi calculado o logaritmo de cada um dos valores resultantes, formando 20 novos atributos.

4. Considere que um nó u possui um conjunto de n vizinhos de saída $V = v_1, ..., v_n$ e que o grau de entrada desses vizinhos é $E = e_i, ..., e_n$, onde e_i representa o grau de entrada do vizinho de saída v_i. Então, foi calculado: $\log \sum_{i=1}^{n} e_i$.

5. Considere que um nó u possui um conjunto de n vizinhos de entrada $V = v_1, ..., v_n$ e que o grau de saída desses vizinhos é $S = s_i, ..., s_n$, onde s_i representa o grau de saída do vizinho de entrada v_i. Então, foi calculado: $\log \sum_{i=1}^{n} s_i$.

6. Considere que o *TrustRank* do nó seja t e que seu grau de entrada seja e. Assim, foi calculado $\log \frac{t}{e}$.

7. Seja $n\_pag^{(n)}$ o número de diferentes nós vizinhos com distância n e $n\_pag^{(n-1)}$ o número de diferentes nós vizinhos com distância $n-1$ de um nó u. Então foi calculado $\log \frac{n\_pag^{(n)}}{n\_pag^{(n-1)}}$. O cálculo foi feito para os seguintes valores: $n = 2, 3, 4$.

8. Considere que a notação $n\_pag^{(n)}$ se refira ao número de diferentes nós vizinhos com distância n de um nó u. Então calculou-se: $temp_1 = \frac{n\_pag^4}{n\_pag^3}$, $temp_2 = \frac{n\_pag^3}{n\_pag^2}$, $temp_3 = \frac{n\_pag^2}{n\_pag^1}$. Dessa forma, 3 novos atributos foram criados:

   - $\log(min(temp_1, temp_2, temp_3))$,
   - $\log(max(temp_1, temp_2, temp_3))$ e
   - $\log(\bar{x}(temp_1, temp_2, temp_3))$, onde min representa o mínimo, max se refere ao máximo e $\bar{x}$ representa a média.

9. Seja $n\_pag^{(n)}$ o número de diferentes nós vizinhos com distância n de um nó u e p o *PageRank* desse nó. Então foi calculado: $\log \frac{n\_pag^{(n)} - n\_pag^{(n-1)}}{p}$. Foram usados $n = 2, 3, 4$.

10. Seja $n\_hosts^{(n)}$ o número de diferentes *hosts* vizinhos com distância n e $n\_hosts^{(n-1)}$ o número de diferentes *hosts* vizinhos com distância $n-1$ de um nó u. O seguinte cálculo foi realizado: $\log \frac{n\_hosts^{(n)}}{n\_hosts^{(n-1)}}$. Foram usados $n = 2, 3, 4$.

11. Considere que a notação $n\_hosts^{(n)}$ se refira ao número de vizinhos que estão a uma distância n do nó. Então calculou-se: $temp_1 = \frac{n\_hosts^4}{n\_hosts^3}$, $temp_2 = \frac{n\_hosts^3}{n\_hosts^2}$, $temp_3 = \frac{n\_hosts^2}{n\_hosts^1}$. Assim, 3 novos atributos foram criados:

    - $\log(min(temp_1, temp_2, temp_3))$,
    - $\log(max(temp_1, temp_2, temp_3))$ e
    - $\log(\bar{x}(temp_1, temp_2, temp_3))$, onde min representa o mínimo, max se refere ao máximo e $\bar{x}$ representa a média.

12. Seja $n\_hosts^{(n)}$ o número de diferentes *hosts* vizinhos com distância n, $n\_hosts^{(n-1)}$ o número de diferentes *hosts* vizinhos com distância $n-1$ de um nó u e p o *PageRank* de u. Então foi feito o seguinte cálculo: $\log \frac{n\_hosts^{(n)} - n\_hosts^{(n-1)}}{p}$. Foram usados os seguintes valores $n = 2, 3, 4$.

13. Seja $trun^{(n)}$ o *PageRank* truncado do nó com uma distância de truncamento n. Assim, foi calculado: $\frac{trun^{(n)}}{trun^{(n-1)}}$. Foram usados os seguintes valores: $n = 2, 3, 4$.

14. Considere que a notação $trun^{(n)}$ se refira ao *PageRank* truncado de uma página com distância n de truncamento. Então calculou-se: $temp_1 = \frac{trun^4}{trun^3}$, $temp_2 = \frac{trun^3}{trun^2}$, $temp_3 = \frac{trun^2}{trun^1}$. Assim, 3 novos atributos foram criados:

    - $\log(min(temp_1, temp_2, temp_3))$,
    - $\log(max(temp_1, temp_2, temp_3))$ e
    - $\log(\bar{x}(temp_1, temp_2, temp_3))$, onde min representa o mínimo, max se refere ao máximo e $\bar{x}$ representa a média.

15. Número de vizinhos da página inicial que estão a uma distância n dividido pelo número de vizinhos da página com maior *PageRank* que estão a uma distância n. Os autores usaram $n = 2, 3, 4$.

16. Número de *hosts* vizinhos da página inicial que estão a uma distância n dividido pelo número de *hosts* vizinhos da página com maior *PageRank* e que estão a uma distância n. Os autores usaram $n = 1, 2, 3, 4$.

17. *PageRank* da página inicial dividido pelo *PageRank* da página com maior *PageRank*.

18. *TrustRank* da página inicial dividido pelo *TrustRank* da página com maior *PageRank*.

Entre os itens 2 e 15 existe um total de 62 atributos. Eles foram extraídos da página inicial e da página com maior *PageRank*, resultando em 124 atributos, que somados aos atributos apresentados nos outros itens, fornecem um total de 138 atributos que podem ser utilizados para discriminar os *hosts* como *spam* ou *ham*.

Em resumo, os vetores de atributos fornecidos pela competição *Web Spam Challenge* tem as seguintes dimensões:

- vetores baseados no conteúdo: 96 dimensões;
- vetores baseados nos *links*: 41 dimensões;
- vetores baseados nos *links* transformados: 138 dimensões.

Um problema encontrado é que nem todos os *hosts*, representados pelos vetores de atributos fornecidos na competição, são rotulados. Assim, os *hosts* não rotulados ou rotulados como *indefinido* foram descartados, sobrando apenas os *hosts* rotulados como *spam* ou não-*spam* (*ham*). Após esse pré-processamento, o conjunto de vetores de atributos extraídos da base de dados WEBSPAM-UK2006 ficou com 6.509 (76.6%) *hosts* rotulados como *ham* e 1.978 (23.4%) rotulados como *spam*. Já, o conjunto de vetores de atributos extraídos da base de dados WEBSPAM-UK2007 ficou com 5.476 (94.5%) *hosts* rotulados como *ham* e 321 (5.5%) rotulados como *spam*.

Capítulo 4

Tratamento dos dados

A dimensionalidade dos vetores de atributos usados neste trabalho é alta e, consequentemente, o custo computacional do processo de classificação também. Porém, é possível que haja atributos pouco representativos para o problema de detecção automática de *Web spam*. Se estes realmente existirem, há um desperdício de custo computacional no processo de classificação, uma vez que as amostras podem ser representadas por vetores de atributos de menor dimensão, o que acelera o processo de treinamento dos métodos de aprendizado. Logo, é importante que seja analisado o impacto da redução da dimensionalidade na classificação de *Web spam*. Para isso, esse trabalho utiliza uma das principais técnicas voltadas a esse problema, a análise de componentes principais. Neste capítulo são apresentadas algumas fundamentações teóricas dessa técnica, bem como do problema de alta dimensionalidade.

Outro problema abordado neste capítulo é o desbalanceamento entre dados de classes diferentes. Esse é um dos problemas encontrados na classificação de *Web spam*, pois o número de exemplos de páginas *ham* é bem superior ao número de exemplos de páginas *spam*. Dessa forma, são apresentadas algumas das principais técnicas utilizadas na literatura para abordar tal problema.

4.1 Redução de dimensionalidade

O termo dimensionalidade pode ser usado para definir a dimensão do espaço de atributos, ou seja, o número de características que representam as amostras de um problema de classificação. O processo de extração dessas caraterísticas é complexo e necessita de atenção e bom conhecimento sobre o problema abordado, para que sejam extraídos atributos que representem bem as amostras. Além disso, é importante que não haja atributos redundantes, pois eles podem atrapalhar o processo de aprendizado dos algoritmos de classificação e/ou aumentar o custo computacional da tarefa de aprendizagem.

Uma das maneiras adotadas na literatura para reduzir o espaço de atributos e diminuir o custo computacional da tarefa de aprendizagem é o uso de técnicas de redução de dimensionalidade. Segundo (Han & Kamber 2006) essas técnicas aplicam codificações ou transformações nos dados para obter uma representação reduzida dos dados originais. Matematicamente, a redução de dimensionalidade pode ser representada da seguinte forma: dada uma amostra qualquer de

um problema com m dimensões, definida como $x = (x_1, ..., x_m)$, encontra-se uma representação com dimensão reduzida dessa amostra, expressa como $z = (z_1, ..., z_k)$ com $k \leq m$.

Considerando que o problema abordado nesse trabalho é representado por três conjuntos de vetores de atributos de alta dimensão, o custo computacional do treinamento no processo de classificação de *Web spam* é alto. Além disso, acredita-se que podem haver atributos com pouca representatividade e redundâncias que atrapalhem o desempenho dos métodos de aprendizado. Desta forma, é importante que a análise de redução de dimensionalidade seja realizada.

Existem diversas técnicas na literatura para reduzir a dimensão do espaço de atributos, tais como: análise fatorial (*factor analysis*) (Johnson & Wichern 1988), análise de componentes independentes (*independent component analysis*) (Hyvärinen & Oja 2000), busca de projeção (*Projection pursuit*) (Friedman & Tukey 1974), projeção aleatória (*random projection*) (Bingham & Mannila 2001), mapas auto-organizáveis de Kohonen (Kohonen 1990) (Seção 2.1.2) e redes neurais articiais (Hinton & Salakhutdinov 2006). Porém, segundo (Haykin 1998), a técnica padrão utilizada para a redução da dimensionalidade em tarefas de aprendizagem e reconhecimento de padrões é a análise de componentes principais (PCA – *principal component analysis*).

4.2 Análise de componentes principais

A técnica de análise de componentes principais (PCA - *principal component analysis*) tem como objetivo encontrar combinações lineares ortogonais dos atributos de um problema para formar um novo conjunto composto por um menor número de atributos, mas que contenha a maior parte das informações do conjunto original.

Suponha que existam n amostras x_i que contenham m atributos, formando a seguinte matriz de dados:

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}.$$

De acordo com (Han & Kamber 2006, Smith 2002), o primeiro passo para a aplicação do PCA na base de dados X é garantir que os dados tenham média amostral 0. Uma das formas de fazer isso é subtrair cada exemplo pela média de cada um dos atributos, conforme é apresentado nas equações a seguir:

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{4.1}$$

$$x_k = x_k - \bar{X}, \tag{4.2}$$

onde $\bar{X}$ é a vetor médio.

O próximo passo é calcular a matriz de correlação R (se os dados não tivessem sido manipulados para terem média amostral 0, essa matriz seria a matriz de covariância) que terá dimensão $m \times m$, usando a seguinte equação:

$$R = \frac{1}{n} \sum_{i=1}^{n} x_i x_i' \tag{4.3}$$

Uma vez que a matriz de correlação R é simétrica, pode-se calcular os autovetores $A = a_1, ..., a_m$ dessa matriz e seus autovalores correspondentes $\Lambda = \lambda_1, ..., \lambda_m$, conforme apresentado em (Smith 2002). O autovalor correspondente a um componente principal indica a quantidade de variância associada a ele. Quanto maior a variância maior a significância do componente. Por isso, deve-se classificar em ordem decrescente os autovetores de acordo com seus autovalores. Logo, para a redução de dimensionalidade, pode-se ignorar os componentes associados aos piores autovalores, escolhendo apenas os p primeiros autovetores, sendo $p < m$, formando-se então o conjunto $A_{reduzido} = a_1, ..., a_p$. Uma das formas de escolher o valor de p é determinando a quantidade de variância pretendida para a nova projeção dos dados. Então, por exemplo, se o usuário deseja que os novos dados possuam 99% de variância, deve ser escolhido o menor valor de p que satisfaça a seguinte condição:

$$\frac{\sum_{i=1}^{p} \lambda_i}{\sum_{i=1}^{m} \lambda_i} \geq \frac{99}{100}. \tag{4.4}$$

Depois de selecionar os componentes principais, formando o conjunto $A_{reduzido}$, deve-se usá-los para obter uma nova projeção dos dados X, que agora terão uma dimensão reduzida, igual a p (mesma dimensão de $A_{reduzido}$). Para isso, deve-se usar a seguinte equação:

$$z_i = A'_{reduzido} * x_i, \tag{4.5}$$

onde z_i será a i-ésima nova amostra com dimensão reduzida.

4.3 O problema de desbalanceamento entre classes

Na tarefa de classificação e reconhecimento de padrões, o problema de desbalanceamento entre classes acontece quando o número de amostras de uma classe é significativamente menor que de outras. Normalmente, a classe com menor número de representantes é a classe de interesse da tarefa de classificação, o que aumenta a importância do tratamento desse problema (Galar, Fernández, Barrenechea, Bustince & Herrera 2012, He & Garcia 2009). Por exemplo, nesse trabalho, as bases de dados utilizadas possuem uma quantidade de amostras de páginas *ham* bem superior à quantidade de amostras de páginas *spam*. Diante disso, os métodos de aprendizado de máquina tendem a favorecer a classe *ham* em detrimento a classe *spam*.

Felizmente, como o problema de desbalanceamento entre classes é recorrente na área de aprendizado e classificação de padrões, ele atraiu a atenção de diversos pesquisadores da área de aprendizado de máquina. Por isso, várias técnicas foram propostas na literatura para solucionar ou amenizar os seus efeitos na tarefa de aprendizagem. Algumas dessas técnicas são apresentadas nas seções a seguir.

4.3.1 Subamostragem aleatória

A técnica de subamostragem aleatória (*random undersampling*) resolve o problema de desbalanceamento entre classes removendo aleatoriamente amostras da classe com maior número de representantes. O problema desse método é que ele pode eliminar dados que sejam importantes para o processo de aprendizado (Galar et al. 2012, He & Garcia 2009).

4.3.2 Sobreamostragem aleatória

A técnica de sobreamostragem aleatória (*random oversampling*) aumenta o número de representantes da classe minoritária replicando de forma aleatória as suas amostras. No entanto, esse método pode provocar superajustamento (*overfitting*) nos dados de treinamento, ou seja, o classificador se especializa na predição de exemplos conhecidos, mas não consegue generalizar suas predições para novos exemplos (Galar et al. 2012, He & Garcia 2009). Esse problema pode ocorrer porque apesar de aumentar o número de exemplos da classe minoritária, as novas amostras são cópias exatas das amostras que já existiam. Logo uma mesma amostra será apresentada mais de uma vez para o classificador em cada iteração do processo de aprendizado.

4.3.3 *EasyEnsemble*

O método *EasyEnsemble* foi proposto por (Liu, Wu & Zhou 2006) com o objetivo de solucionar o principal problema da técnica de subamostragem aleatória: a perda de dados importantes para o processo de aprendizado.

Dado um conjunto S_{min} formado pelas amostras da classe minoritária do conjunto de treinamento e um conjunto S_{maj} formado pelas amostras da classe majoritária, o método *EasyEnsemble* seleciona aleatoriamente e com reposição t subconjuntos $S_{maj_1}, S_{maj_2}, ..., S_{maj_t}$ do conjunto S_{maj}. Para cada subconjunto S_{maj_i}, um classificador *AdaBoost* é treinado usando esse conjunto e o conjunto S_{min}, gerando então um conjunto H_i de saídas, composto por $h_{i1}, h_{i2}, ..., h_{in}$, sendo n o número de amostras do conjunto de treinamento, ou seja $n = |S_{min}| + |S_{maj_i}|$. Depois disso, todas as saídas do conjunto H são combinadas para obter uma decisão final para cada amostra do conjunto de treinamento. A principal ideia desse método é que as características extraídas dos diferentes subconjuntos S_{maj_i} contêm diferentes aspectos da classe do conjunto original S_{maj} e, por isso, o problema de perda de dados do método de subamostragem aleatória é minimizado (Liu et al. 2006, He & Garcia 2009).

4.3.4 *BalanceCascade*

O método *BalanceCascade*, proposto por (Liu et al. 2006), possui o mesmo objetivo do método *EasyEnsemble*: minimizar o problema de perda de dados do método de subamostragem aleatória.

O processo usado pelo método *BalanceCascade* é similar ao que ocorre com o método *EasyEnsemble*. Ele seleciona aleatoriamente e com reposição um conjunto S_{maj_i} do conjunto S_{maj}. Depois disso, um classificador *AdaBoost* é treinado usando o subconjunto S_{maj_i} e o conjunto S_{min}, gerando um conjunto de saídas $H_i = h_{i1}, h_{i2}, ..., h_{in}$, onde $n = |S_{min}| + |S_{maj_i}|$.

Depois que o classificador é treinado, todas as amostras $x \in S_{maj}$ consideradas redundantes são excluídas do conjunto S_{maj}. Uma amostra $x_j \in S_{maj}$ é considerada redundante em S_{maj} se a predição feita para essa amostra for correta, pois nesse caso pode-se afirmar que o classificador já aprendeu a classificá-la, logo, ela não precisa ser apresentada para ele novamente. Depois dessa etapa, o processo é repetido até que uma condição de parada seja satisfeita. Ao final, as saídas do conjunto H, obtidas em cada iteração do processo, são combinadas para obter uma decisão final para cada amostra do conjunto de treinamento (Liu et al. 2006, He & Garcia 2009).

4.3.5 *Synthetic minority oversampling technique*

Synthetic minority oversampling technique (SMOTE) é uma técnica de sobreamostragem proposta por (Chawla, Bowyer, Hall & Kegelmeyer 2002) que cria amostras sintéticas para a classe minoritária baseadas nos representantes já existentes dessa classe. Segundo os autores, os exemplos sintéticos gerados pela técnica SMOTE são resultantes de operações realizadas no espaço dos atributos e não no espaço dos dados. Essas operações criam amostras sintéticas nos segmentos das linhas que ligam cada amostra da classe minoritária a m de suas k vizinhas mais próximas, escolhidas de forma aleatória.

O processo de geração de exemplos sintéticos é o seguinte: escolha a porcentagem n de SMOTE, ou seja, a porcentagem de novos dados sintéticos que o SMOTE deve criar. Selecione uma amostra x_i do conjunto S_{min} formado pelas amostras da classe minoritária. Encontre os k vizinhos mais próximos da amostra x_i, formando o conjunto de vizinhos $V = v_1, ..., v_k$. Dado que $m = int(\frac{n}{100})$, selecione aleatoriamente m vizinhos de x_i que estão no conjunto V, formando o conjunto $V^{(2)} = v_1^{(2)}, ..., v_m^{(2)}$. Depois disso, para cada vizinho de x_i que está no conjunto $V^{(2)}$, gere um dado sintético $x_a^{(2)}$ da seguinte maneira: $x_a^{(2)} = x_i + (v_j^{(2)} - x_i)\alpha$, onde α é um valor aleatório entre 1 e 2 (Chawla et al. 2002).

A técnica SMOTE é umas das técnicas de balanceamento de dados mais bem sucedidas da literatura sendo usada como base para outros métodos de sobreamostragem, tais como os métodos apresentados em (Ramentol, Caballero, Bello & Herrera 2012, Bunkhumpornpat, Sinapiromsaran & Lursinsap 2012, Dong & Wang 2011). Ela também vem sendo aplicada em diversos problemas de classificação de padrões, como os apresentados em (Nahar, Imam, Tickle, Ali & Chen 2012, Del Gaudio & Branco 2009, Carter & Dewan 2010).

4.3.6 Aprendizado sensível ao custo

A técnica de aprendizado sensível ao custo (ASC) (Elkan 2001) não usa a abordagem tradicional de outros métodos que tentam balancear os dados no processo de treinamento com o objetivo de minimizar o problema de desbalanceamento entre as classes. Em vez disso, ela mantém a distribuição de dados das classes, porém considera os custos relacionados aos exemplos classificados incorretamente. Isso significa que na ASC uma penalidade é aplicada aos falsos positivos ou falsos negativos, usando-se uma matriz de custos, que normalmente possui a estrutura apresentada na Tabela 4.1 (Elkan 2001).

A penalidade aplicada pela técnica ASC pode ser igual para os dois tipos de erro (falso positivo e falso negativo) ou pode ser maior para um ou outro dependendo do problema que

Tabela 4.1: Matriz de custo.

Classe	Exemplos positivos	Exemplos negativos
Classificados como positivo	$C(1,1) = c_{11}$	$C(1,0) = c_{10}$
Classificados como negativo	$C(0,1) = c_{01}$	$C(0,0) = c_{00}$

está sendo abordado e do nível de desbalanceamento entre as classes. Porém, é comum que nos problemas de classificação em que há desbalanceamento entre classes, um erro de classificação da classe minoritária seja considerado mais sério e por isso receba uma penalidade maior (Liu et al. 2006).

A técnica ASC pode ser dividida em duas categorias. A primeira delas consiste em tornar um método de classificação sensível ao custo alterando seu processo de treinamento, como por exemplo: árvores de decisão sensíveis ao custo (Ling, Yang, Wang & Zhang 2004, Turney 1995), redes neurais sensíveis ao custo (Kukar & Kononenko 1998, Bertoni, Frasca & Valentini 2011) e método de autotreinamento sensível ao custo (CS-ST – *cost-sensitive self-training method*) (Guo, Zhang & Spencer 2012). A segunda categoria transforma um algoritmo de classificação em um algoritmo sensível ao custo sem alterar seu processo de treinamento. Para isso, é feito um pré-processamento das amostras de treinamento ou manipulação dos resultados obtidos pelo classificador. Os algoritmos que usam esse método são conhecidos como metaclassificadores. Alguns exemplos dessa categoria são: *MetaCost* (Domingos 1999), classificador *naive* Bayes sensível ao custo (Chai, Deng, Yang & Ling 2004), método de ponderação de instâncias para indução de árvores de decisão sensível ao custo (Ting 1998), *thresholding* (Sheng & Ling 2006) e classificação sensível ao custo suave (*soft cost-sensitive classification*) (Jan, Wang, Lin & Lin 2012).

4.3.7 Regra do vizinho mais próximo condensado

O método regra do vizinho mais próximo condensado (CNN - *condensed nearest neighbor rule*), proposto por (Hart 1968), cria um subconjunto consistente S_2 a partir do conjunto original de amostras S. O subconjunto S_2 é considerado consistente quando um método de aprendizado KNN, com $K = 1$, consegue classificar corretamente todas as instâncias do conjunto S usando o conjunto S_2. Segundo os autores, todo conjunto S possui algum subconjunto consistente. Entre eles existe um subconjunto consistente mínimo que é um subconjunto S_2 consistente e com o mínimo número de amostras. Porém, esse último, às vezes, é difícil de ser encontrado.

4.3.8 Ligações Tomek

O método de ligações Tomek (Tomek *links*) (Tomek 1976) é uma variação do método CNN. Ele faz a remoção de amostras da classe majoritária que são consideradas ruídos ou que estão próximas a superfície de decisão do classificador. Então, considere que uma amostra x_i, pertencente ao conjunto majoritário S_{maj}, é vizinha a uma distância $d(x_i, x_j)$ de uma amostra x_j pertencente a classe oposta, representada pelo conjunto minoritário S_{min}. Se não existir nenhum x_k tal que $d(x_i, x_k) < d(x_i, x_j)$ ou $d(x_j, x_k) < d(x_i, x_j)$, então o par de amostras x_i, x_j é chamado de ligação Tomek. Além disso, nessa situação, pode-se inferir que as amostras que estão próximas a superfície de decisão, ou uma delas, ou as duas, são ruídos. Então, em problemas com

desbalanceamento entre as classes, as amostras x da classe majoritária S_{maj}, que pertencem a alguma ligação Tomek, são excluídas do conjunto original de amostras.

4.3.9 Seleção unilateral

O método de seleção unilateral (OSS - *One-side selecion*) (Kubat & Matwin 1997) é caracterizado por aplicar os métodos CNN e ligações Tomek para minimizar o problema de desbalanceamento entre classes. Nessa técnica, primeiramente, uma certa quantidade de amostras redundantes da classe majoritária S_{maj} é eliminada através da criação de um subconjunto consistente S_2 usando a técnica CNN. Depois disso, por meio da técnica de ligações Tomek, um novo conjunto S_3 é formado eliminando-se as amostras do conjunto S_2 que pertencem a alguma ligação Tomek.

Capítulo 5

Experimentos e resultados

Neste capítulo, são apresentadas as configurações adotadas para os algoritmos de classificação e as métricas usadas para avaliar o seu desempenho. Também é apresentada uma análise dos resultados obtidos por esses algoritmos com todos os tipos de atributos e combinações entre os mesmos, usando classes balanceadas e desbalanceadas. Adicionalmente, é analisada a viabilidade da redução de dimensionalidade dos vetores de atributos e proposta uma abordagem de classificação de *Web spam* que faz a combinação das predições obtidas com os diferentes tipos de atributos. Em seguida, é realizada uma validação estatística para determinar os melhores métodos e, por fim, são apresentados trabalhos correlatos e uma comparação de seus resultados com os resultados apresentados nessa dissertação.

5.1 Configurações

Para tornar os resultados completamente reprodutíveis, são apresentadas nessa seção as configurações adotadas para cada classificador.

5.1.1 Redes neurais artificiais

Neste trabalho, foram avaliadas as seguintes redes neurais artificiais (RNAs) para a detecção automática de *Web spam*: MLP treinada com o algoritmo *backpropagation* e método do gradiente descendente (MLP-GD), MLP treinada com o método de Levenberg-Marquardt (MLP-LM), rede neural de função de base radial (RBF - *radial basis function*) e quantização vetorial por aprendizagem (LVQ - *learning vector quantization*) usando mapas auto-organizáveis de Kohonen (SOM - *Kohonen's self-organizing maps*) para a inicialização do mapa de características (SOM + LVQ). Todas as RNAs foram implementadas usando a ferramenta MATLAB.

Todas as redes neurais MLP foram implementadas com uma única camada intermediária e com um neurônio na camada de saída. Além disso, adotou-se uma função de ativação linear para o neurônio da camada de saída e uma função de ativação do tipo tangente hiperbólica para os neurônios da camada intermediária. Dessa forma, os pesos e o bias da rede foram inicializados com valores aleatórios entre 1 e -1 e os dados usados para a classificação foram normalizados

para o intervalo $[-1, 1]$, por meio da seguinte equação:

$$x = 2 \times \frac{x - x_{min}}{x_{max} - x_{min}} - 1, \tag{5.1}$$

sendo x a matriz com todos os vetores de características e x_{min} e x_{max} o menor e o maior valor da matriz x, respectivamente. Nas simulações com as redes neurais RBF e SOM + LVQ, os dados também foram normalizados usando a Equação 5.1.

Em todas as simulações com as redes neurais MLP, os critérios de parada adotados foram: número de épocas maior que um limiar θ, erro quadrático médio (EQM) do conjunto de treino menor que um limiar γ ou aumento do EQM do conjunto de validação (verificado a cada 10 épocas).

Os parâmetros de cada RNA foram empiricamente calibrados e são apresentados na Tabela 5.1.

Tabela 5.1: Parâmetros das RNAs.

Parâmetros	MLP-GD	MLP-LM	RBF	SOM + LVQ	
				Etapa SOM	Etapa LVQ
Limite máximo de iterações θ	10000	500	–	2000	2000
Limite mínimo do EQM γ	0,001	0,001	–	–	–
Passo de aprendizado α	0,005	0,001	–	0,01	0,001
Passo de aprendizado mínimo α_{min}	–	–	–	0,001	0,0001
Número de neurônios na camada intermediária	100	50	50	150	150
Função de vizinhança	–	–	–	unidimensional	–
Raio inicial de vizinhança σ	–	–	–	6	–

Para as simulações com a rede neural RBF, não usou-se nenhum critério de parada porque o treinamento adotado não é iterativo, pois usa o método de pseudo-inversa (Bishop 1995). Além disso, para essa rede, a dispersão para cada neurônio foi calculada pela seguinte equação: $disp_i = \frac{1}{m} \sum_{j=1}^{m} dist(C_i, C_j)$, onde m é o número de neurônios ou centros da rede neural e $dist(C_i, C_j)$ é a distância Euclidiana entre o centro C_i e o centro C_j.

5.1.2 Máquinas de vetores de suporte

O método SVM foi implementado utilizando a biblioteca LIBSVM (Chang & Lin 2011) disponível para a ferramenta MATLAB. Foram feitas simulações com as funções de *kernel* linear, RBF, sigmoidal e polinomial. A técnica de *grid search* foi empregada para a definição dos parâmetros. Porém, nos classificadores SVM com *kernel* polinomial e sigmoidal, que possuem um maior número de parâmetros a serem definidos, optou-se por realizar a *grid search* apenas sobre os parâmetros C e γ, devido ao excessivo custo computacional. Neste caso, adotaram-se os valores padrões da LIBSVM para os demais parâmetros. Os dados usados para a classificação também foram normalizados usando a Equação 5.1.

A técnica de *grid search* foi aplicada com um conjunto de treino (80% dos dados) e teste (20% dos dados), escolhidos aleatoriamente para cada configuração de classificação. Após isso, os melhores parâmetros foram escolhidos e usados para realizar os experimentos com o SVM.

Nesses experimentos, todos os tipos de *kernel* foram avaliados. Porém, neste capítulo optou-se por apresentar apenas os resultados obtidos com o *kernel* RBF, uma vez que este obteve o melhor desempenho. Os parâmetros usados nos experimentos e os resultados obtidos pelo método SVM para todos os tipos de *kernel* são apresentados no Apêndice A.

5.1.3 Demais métodos

Os demais classificadores foram implementados usando a ferramenta WEKA (*Waikato Environment for Knowledge Analysis*) (Hall, Frank, Holmes, Pfahringer, Reutemann & Witten 2009) que agrega diversos algoritmos e abordagens da área de aprendizado de máquina. Os métodos *AdaBoost* e *bagging* foram treinados com 100 iterações, sendo que ambos empregam agregação de múltiplas versões do método C4.5. Para os demais métodos, foram empregados os parâmetros padrões da ferramenta utilizada.

5.2 Medidas de desempenho

Para avaliar o desempenho de cada classificador foi usada uma validação por subamostragem aleatória, também conhecida como validação cruzada de Monte Carlo (Shao 1993). Esse método foi escolhido pois oferece maior liberdade na seleção dos subconjuntos de treinamento e teste, já que, diferentemente do método de validação k-*folds*, permite que sejam feitas quantas repetições foram desejadas, usando qualquer porcentagem de dados para treinamento e teste. Dessa forma, foram feitas 10 simulações com cada classificador e em cada uma, 80% dos dados foram usados para treinamento e 20% para teste (nas MLPs, os dados de teste foram divididos em 10% para validação e 10% para teste). Eles foram selecionados aleatoriamente e com reposição a cada simulação. Ao final de todas as simulações, foi calculada a média e o desvio padrão dos resultados.

Para avaliar e comparar os resultados, foram utilizadas as medidas de desempenho, amplamente empregadas na literatura, apresentadas na Tabela 5.2, onde v_p (verdadeiros positivos) refere-se ao número de exemplos corretamente classificados como *spam*, f_p (falsos positivos) refere-se aos exemplos que foram incorretamente classificados como *spam*, v_n (verdadeiros negativos) refere-se ao número de exemplos classificados corretamente como *ham* e f_n (falsos negativos) refere-se ao número de exemplos classificados incorretamente como *ham*.

Tabela 5.2: Matriz de confusão.

	Classe verdadeira	
	Spam	*Ham*
Classificados como *spam*	v_p	f_p
Classificados como *ham*	f_n	v_n

Acurácia:	$Acc = \frac{v_p+v_n}{v_p+v_n+f_p+f_n}$
Sensitividade:	$Sen = \frac{v_p}{v_p+f_n}$
Especificidade:	$Esp = \frac{v_n}{v_n+v_p}$
Precisão:	$Pre = \frac{v_p}{v_p+f_p}$
F-medida:	$Fmed = 2 * \frac{Pre*Sen}{Pre+Sen}$

As medidas de desempenho apresentadas na Tabela 5.2 podem ser definidas da seguinte forma:

- *Acurácia:* taxa de acerto global, ou seja, a proporção de predições corretas em relação ao tamanho do conjunto de dados.
- *Sensitividade (recall):* proporção de padrões da classe positiva (*spam*) identificada corretamente. Indica o quão bom o classificador é para identificar a classe positiva.
- *Especificidade:* proporção de padrões da classe negativa (*ham*) identificada corretamente. Indica o quão bom o classificador é para identificar a classe negativa.
- *Precisão (precision):* porcentagem de padrões classificados como pertencentes à classe positiva e que realmente pertencem a classe positiva.
- *F-medida:* média harmônica entre precisão e sensitividade.

5.3 Resultados obtidos com a base de dados WEBSPAM-UK2006

Nessa seção são apresentados os resultados da detecção automática de *Web spam* obtidos pelos métodos de aprendizado de máquina descritos no Capítulo 2. Inicialmente, para cada método e conjunto de vetores de atributos, foram feitas simulações usando classes desbalanceadas, conforme originalmente fornecido na competição *Web Spam Challenge 2007*, sendo 6.509 *hosts* (76,6%) da classe *ham* e 1.978 (23,4%) da classe *spam*. Em seguida, para avaliar se o desbalanceamento dos dados interfere no desempenho dos métodos, também foram realizados experimentos usando o mesmo número de amostras em cada classe. Para promover o balanceamento, foi utilizada a técnica de subamostragem aleatória (Seção 4.3.1) por ser uma técnica simples e bastante empregada na literatura. Neste caso, depois da aplicação do método, cada classe passou a ter 1.978 representantes.

As Tabelas 5.3, 5.4 e 5.5 apresentam os resultados obtidos por cada método na classificação dos atributos baseados no conteúdo, nos *links* e nos *links* transformados. Cada coluna apresenta a média e o desvio padrão dos resultados obtidos usando a técnica de validação por subamostragem aleatória. Os valores em negrito precedidos pelo símbolo "↑" indicam os melhores resultados, enquanto os valores em negrito precedidos pelo símbolo "↓" indicam os piores resultados obtidos usando cada configuração de classes (balanceadas ou desbalanceadas). Por outro lado, os valores em negrito precedidos pelo símbolo "*" indicam os melhores ou piores resultados considerando os três tipos de vetores de atributos.

De maneira geral, os resultados indicam que quando os métodos classificadores foram treinados usando classes balanceadas seu desempenho foi superior comparado às simulações com classes desbalanceadas. Portanto, verifica-se que, em geral, os métodos de classificação analisados tendem a favorecer a classe com maior número de representantes. Isso pode ser confirmado, pois os valores da sensitividade obtidos pelos métodos usando classes desbalanceadas foram, em geral, mais baixos, o que mostra que os classificadores erraram mais na identificação dos

Tabela 5.3: Resultados obtidos na detecção de *Web spam* usando atributos baseados no conteúdo.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes desbalanceadas				
Bagging	↑**89,7** ± 0,5	68,7 ± 2,3	96,1 ± 0,6	*↑**84,4** ± 1,9	↑**0,757** ± 0,014
Floresta aleatória	88,9 ± 1,0	65,7 ± 4,4	96,0 ± 1,2	83,4 ± 3,7	0,734 ± 0,024
MLP-LM	88,6 ± 1,4	69,3 ± 4,2	94,2 ± 1,2	77,6 ± 4,6	0,731 ± 0,032
AdaBoost	87,9 ± 0,9	66,6 ± 3,2	94,4 ± 0,7	78,4 ± 2,3	0,720 ± 0,024
IBK	86,2 ± 0,8	64,6 ± 1,3	92,7 ± 0,7	72,8 ± 2,0	0,685 ± 0,011
C4,5	85,1 ± 1,2	67,7 ± 4,6	90,4 ± 0,5	68,0 ± 0,6	0,678 ± 0,024
MLP-GD	86,2 ± 1,2	57,0 ± 4,6	95,0 ± 0,4	77,5 ± 2,7	0,656 ± 0,039
LogitBoost	84,3 ± 1,0	54,2 ± 3,9	93,5 ± 1,1	71,6 ± 3,2	0,616 ± 0,023
SVM	82,5 ± 0,6	36,0 ± 2,4	*↑**96,6** ± 0,5	76,2 ± 2,8	0,488 ± 0,023
OneR	80,8 ± 0,7	38,3 ± 2,4	93,7 ± 0,6	65,0 ± 2,7	0,482 ± 0,023
SOM + LVQ	35,3 ± 3,7	94,7 ± 0,9	80,9 ± 0,7	67,0 ± 2,7	0,461 ± 0,033
RBF	80,7 ± 0,6	↓**30,3** ± 2,7	96,1 ± 0,4	70,0 ± 2,2	0,422 ± 0,028
Naive Bayes	↓**32,3** ± 4,6	*↑**97,4** ± 1,0	*↓**12,5** ± 6,2	*↓**25,4** ± 1,3	↓**0,402** ± 0,015
	Classes Balanceadas				
MLP-LM	↑**87,6** ± 1,5	↑**86,5** ± 2,0	*↑**88,8** ± 1,9	*↑**89,1** ± 2,4	↑**0,877** ± 0,017
Bagging	85,0 ± 1,2	83,5 ± 2,3	86,5 ± 1,5	86,1 ± 1,3	0,848 ± 0,013
MLP-GD	84,7 ± 1,6	82,9 ± 2,0	86,4 ± 2,4	86,1 ± 2,4	0,845 ± 0,015
AdaBoost	82,7 ± 1,3	81,8 ± 2,5	83,6 ± 1,9	83,3 ± 1,6	0,825 ± 0,014
Floresta aleatória	82,8 ± 1,5	81,8 ± 3,3	83,8 ± 3,1	83,4 ± 2,3	0,825 ± 0,015
IBK	79,9 ± 1,4	77,0 ± 2,6	82,7 ± 1,5	81,4 ± 1,5	0,791 ± 0,017
C4,5	78,7 ± 0,4	79,2 ± 2,5	78,2 ± 1,9	78,3 ± 1,6	0,787 ± 0,012
LogitBoost	78,3 ± 1,6	77,0 ± 3,2	79,7 ± 0,6	78,9 ± 1,1	0,779 ± 0,015
Naive Bayes	↓**63,6** ± 7,5	83,6 ± 17,7	↓**43,6** ± 30,5	↓**62,8** ± 10,2	0,694 ± 0,039
SOM + LVQ	65,6 ± 2,4	74,9 ± 2,5	70,3 ± 0,8	72,4 ± 1,5	0,688 ± 0,011
SVM	71,1 ± 1,4	60,6 ± 2,7	81,6 ± 1,5	76,7 ± 1,5	0,677 ± 0,019
OneR	66,5 ± 1,5	61,6 ± 1,0	71,4 ± 2,4	68,3 ± 2,0	0,648 ± 0,013
RBF	68,9 ± 1,7	*↓**55,6** ± 1,9	82,3 ± 2,2	75,9 ± 2,6	*↓**0,641** ± 0,019

Tabela 5.4: Resultados obtidos na detecção de *Web spam* usando atributos baseados nos *links*.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes desbalanceadas				
Bagging	↑**89,7** ± 0,6	79,2 ± 1,6	92,8 ± 0,9	↑**77,2** ± 2,0	↑**0,781** ± 0,009
Floresta aleatória	89,1 ± 0,9	76,5 ± 7,9	92,8 ± 2,2	77,0 ± 3,7	0,764 ± 0,027
AdaBoost	87,9 ± 0,8	72,6 ± 3,0	92,6 ± 0,8	74,9 ± 1,9	0,737 ± 0,019
C4,5	87,2 ± 0,5	73,2 ± 2,1	91,5 ± 0,8	72,3 ± 1,4	0,727 ± 0,011
MLP-LM	88,1 ± 1,6	70,8 ± 6,6	93,0 ± 0,7	74,1 ± 3,7	0,723 ± 0,049
LogitBoost	86,7 ± 0,5	71,8 ± 4,0	91,2 ± 1,4	71,4 ± 2,1	0,715 ± 0,009
MLP-GD	86,4 ± 1,3	61,6 ± 3,1	93,9 ± 0,9	75,3 ± 2,9	0,677 ± 0,026
IBK	83,7 ± 1,4	67,8 ± 2,8	88,5 ± 1,3	64,2 ± 2,4	0,659 ± 0,023
Naive Bayes	73,4 ± 1,2	↑**94,6** ± 1,3	↓**66,9** ± 1,5	↓**46,5** ± 1,2	0,624 ± 0,012
SVM	81,2 ± 0,7	47,1 ± 2,5	91,5 ± 0,6	62,8 ± 1,9	0,538 ± 0,021
OneR	78,6 ± 0,8	49,4 ± 3,5	87,5 ± 1,2	54,7 ± 1,9	0,518 ± 0,023
SOM + LVQ	*↓**21,8** ± 3,6	94,3 ± 1,3	77,4 ± 0,6	54,0 ± 3,3	0,309 ± 0,035
RBF	77,7 ± 0,5	*↓**17,2** ± 1,1	↑**96,1** ± 0,7	57,4 ± 4,3	*↓**0,264** ± 0,015
	Classes Balanceadas				
Bagging	↑**87,7** ± 1,3	91,7 ± 1,9	83,7 ± 1,7	84,9 ± 1,4	↑**0,882** ± 0,013
Floresta aleatória	↑**87,7** ± 0,7	88,8 ± 2,5	↑**86,7** ± 2,1	↑**87,1** ± 1,6	0,879 ± 0,010
MLP-LM	86,4 ± 1,7	92,1 ± 3,8	81,1 ± 2,6	82,3 ± 2,6	0,868 ± 0,019
AdaBoost	85,5 ± 0,8	87,7 ± 1,7	83,2 ± 1,0	83,9 ± 0,8	0,858 ± 0,009
MLP-GD	83,9 ± 1,9	90,2 ± 2,6	77,9 ± 3,0	80,0 ± 2,9	0,848 ± 0,019
LogitBoost	83,2 ± 1,5	88,0 ± 3,7	78,4 ± 1,7	80,4 ± 1,0	0,840 ± 0,018
C4,5	83,4 ± 0,9	85,4 ± 2,2	81,5 ± 1,7	82,3 ± 1,4	0,838 ± 0,011
Naive Bayes	80,5 ± 1,8	↑**94,8** ± 0,9	↓**66,1** ± 3,3	73,7 ± 2,0	0,829 ± 0,014
OneR	81,4 ± 1,2	88,7 ± 1,0	74,2 ± 2,2	77,5 ± 1,5	0,827 ± 0,010
IBK	78,3 ± 0,4	80,8 ± 1,4	75,8 ± 1,6	77,1 ± 0,8	0,789 ± 0,003
SVM	76,3 ± 1,4	77,1 ± 2,2	75,4 ± 2,4	75,8 ± 1,7	0,765 ± 0,014
SOM + LVQ	70,1 ± 3,5	70,0 ± 2,8	70,0 ± 1,1	↓**70,0** ± 1,3	0,700 ± 0,017
RBF	↓**70,0** ± 1,3	↓**69,5** ± 2,8	70,6 ± 2,7	70,3 ± 1,7	↓**0,699** ± 0,015

Tabela 5.5: Resultados obtidos na detecção de *Web spam* usando atributos baseados nos *links* transformados.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes desbalanceadas				
Bagging	***↑89,8** ± 0,7	78,9 ± 1,2	93,1 ± 0,8	**↑77,6** ± 2,1	***↑0,782** ± 0,014
Floresta aleatória	88,8 ± 0,4	76,5 ± 3,1	92,5 ± 1,5	75,8 ± 4,1	0,760 ± 0,012
MLP-LM	89,0 ± 0,8	74,9 ± 3,9	93,1 ± 0,5	76,1 ± 2,1	0,754 ± 0,027
SVM	88,3 ± 0,6	76,6 ± 2,3	91,8 ± 0,8	73,9 ± 1,6	0,752 ± 0,012
MLP-GD	88,3 ± 0,9	75,2 ± 2,2	92,1 ± 1,4	73,9 ± 3,1	0,745 ± 0,014
AdaBoost	87,9 ± 0,8	72,6 ± 3,0	92,6 ± 0,8	74,9 ± 1,9	0,737 ± 0,019
LogitBoost	86,4 ± 0,4	70,2 ± 4,9	91,5 ± 1,5	72,3 ± 3,2	0,711 ± 0,019
C4,5	86,3 ± 0,4	73,5 ± 1,8	90,0 ± 0,7	68,5 ± 2,2	0,709 ± 0,016
SOM + LVQ	66,6 ± 2,6	92,2 ± 0,9	86,2 ± 0,6	72,2 ± 2,0	0,692 ± 0,015
IBK	84,7 ± 0,7	67,5 ± 2,4	90,0 ± 1,5	67,3 ± 3,5	0,673 ± 0,011
OneR	82,9 ± 1,0	**↓64,3** ± 2,1	88,6 ± 1,3	63,2 ± 2,8	0,637 ± 0,018
RBF	84,7 ± 1,0	52,4 ± 6,2	**↑94,5** ± 1,1	74,3 ± 2,6	0,612 ± 0,043
Naive Bayes	**↓35,9** ± 1,0	**↑96,5** ± 1,0	**↓17,5** ± 1,5	**↓26,2** ± 0,3	**↓0,412** ± 0,004
	Classes Balanceadas				
Bagging	***↑88,3** ± 1,1	91,3 ± 1,6	85,3 ± 1,2	**↑86,2** ± 1,1	***↑0,886** ± 0,011
MLP-GD	87,4 ± 1,6	89,6 ± 2,2	85,2 ± 1,1	85,9 ± 2,0	0,877 ± 0,019
Floresta aleatória	87,4 ± 1,4	89,5 ± 3,7	85,3 ± 2,4	86,0 ± 1,7	0,876 ± 0,016
AdaBoost	87,1 ± 1,2	88,8 ± 0,9	**↑85,5** ± 2,0	85,9 ± 1,7	0,873 ± 0,011
MLP-LM	86,3 ± 2,7	87,5 ± 4,0	85,2 ± 3,6	85,8 ± 3,3	0,866 ± 0,027
SVM	86,0 ± 0,7	88,5 ± 1,7	83,5 ± 1,8	84,4 ± 1,3	0,864 ± 0,007
LogitBoost	84,4 ± 2,2	87,6 ± 3,7	81,3 ± 1,3	82,7 ± 1,0	0,850 ± 0,021
C4,5	84,1 ± 1,9	85,6 ± 2,4	82,5 ± 1,8	83,1 ± 1,8	0,844 ± 0,020
SOM + LVQ	85,2 ± 2,1	**↓81,3** ± 2,1	83,3 ± 1,1	82,1 ± 1,5	0,836 ± 0,011
OneR	82,7 ± 0,7	86,5 ± 2,0	79,0 ± 1,9	80,5 ± 1,2	0,834 ± 0,008
RBF	81,3 ± 0,9	82,0 ± 2,2	80,6 ± 2,0	80,9 ± 1,4	0,814 ± 0,010
IBK	80,7 ± 0,1	81,6 ± 0,7	79,8 ± 0,6	80,2 ± 0,4	0,809 ± 0,002
Naive Bayes	***↓61,0** ± 4,7	***↑96,8** ± 0,8	***↓25,2** ± 9,6	***↓56,6** ± 3,2	**↓0,714** ± 0,025

padrões pertencentes à classe *spam*. Por outro lado, a taxa de especificidade foi alta nas simulações realizadas sobre classes desbalanceadas, o que indica que os métodos aprenderam melhor a classificar as amostras *ham*, que possuem maior número de representantes.

Outro ponto a ser observado é que o método escolhido para balancear as classes (subamostragem aleatória), que segundo alguns autores pode ter problema com a perda de informações, não prejudicou o aprendizado dos algoritmos. Essa afirmação pode ser feita pois se houvesse esse problema, os desvios padrões obtidos nas simulações tenderiam a ser maiores, já que com o método de validação por subamostragem aleatória, a cada repetição, dados diferentes dos usados na simulação anterior podem ser escolhidos para formar o conjunto de teste. Diante disso, pode-se concluir que esse método de balanceamento é satisfatório para o problema de classificação de *Web spam*, pois além de gerar resultados com pequenos desvios padrões, melhora bastante o desempenho dos métodos de aprendizado, se comparado ao desempenho obtido com classes desbalanceadas.

Entre os métodos avaliados, nota-se que o método *bagging* apresenta os melhores resultados. Ele foi capaz de detectar, em média, 82,2% dos *spam hosts* com uma precisão média de 82,7%, independente do tipo de atributo utilizado. Apenas no cenário apresentado na Tabela 5.3, em que são explorados os atributos baseados no conteúdo das páginas usando classes balanceadas, a rede neural MLP treinada com o método de Levenberg-Marquardt é superior ao método *bagging*. Por outro lado, os métodos que tiveram o pior desempenho foram o *naive* Bayes e a rede neural RBF.

Em relação ao tipo de atributo, os melhores resultados foram obtidos no cenário em que foram

utilizados os vetores de características baseados nos *links* transformados (Tabela 5.5), que obteve a melhor taxa de acurácia. O valor da acurácia não é um bom parâmetro de avaliação quando as classes estão desbalanceadas, porém quando o número de representantes em cada classe é igual, sua importância aumenta. De toda forma, o valor da F-medida, que é considerada uma das medidas mais importantes quando há problema de desbalanceamento entre classes, obtido nas simulações com características baseadas nos *links* transformados, também foi maior que nos outros tipos de atributos.

5.3.1 Análise do impacto da redução de dimensionalidade nos vetores de atributos extraídos da base de dados WEBSPAM-UK2006

Com o objetivo de melhorar os resultados obtidos pelos algoritmos e enriquecer a análise de desempenho dos mesmos, foi feita uma seleção dos atributos usados nos experimentos. Esse procedimento foi realizado para diminuir o custo computacional do processo de classificação, pois observou-se que os vetores de atributos possuem alta dimensionalidade. Também, cogitou-se a hipótese de que poderiam haver redundâncias nos vetores de atributos usados nos experimentos, que poderiam estar impactando o desempenho dos métodos de classificação. Nesse sentido, foi empregado o método de análise de componentes principais (PCA – *principal component analysis*) (Seção 4.2) para tentar obter uma estrutura simplificada dos dados e eliminar ou reduzir a existência de redundâncias nos vetores de atributos. Esse método foi escolhido por ser amplamente empregado no processo de redução de dimensionalidade em problemas de classificação e reconhecimento de padrões.

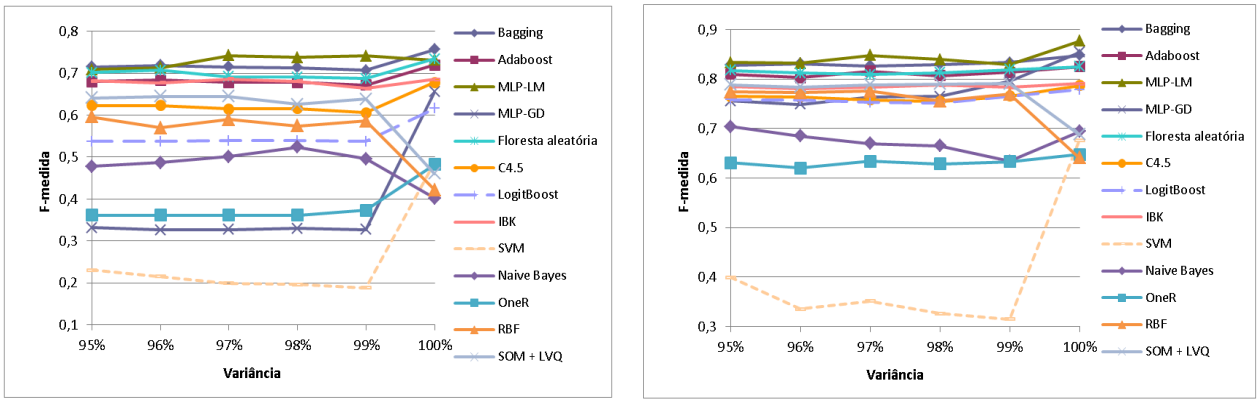
O método PCA foi empregado por meio do software WEKA, sendo usada uma porcentagem de variância que diversificou entre 95% e 99%. A Tabela 5.6 apresenta a redução em número de dimensões em relação a cada porcentagem de variância utilizada. A última coluna que está rotulada como "100%" apresenta o número de dimensões original dos vetores de atributos.

Tabela 5.6: Número de dimensões obtidas após redução da dimensionalidade dos vetores de atributos.

	Variância					
	95%	96%	97%	98%	99%	100%
Tipos de atributos	Nº de dimensões					
Conteúdo	40	43	47	52	60	96
Links	18	20	21	23	27	41
Links transformados	40	43	48	54	61	138

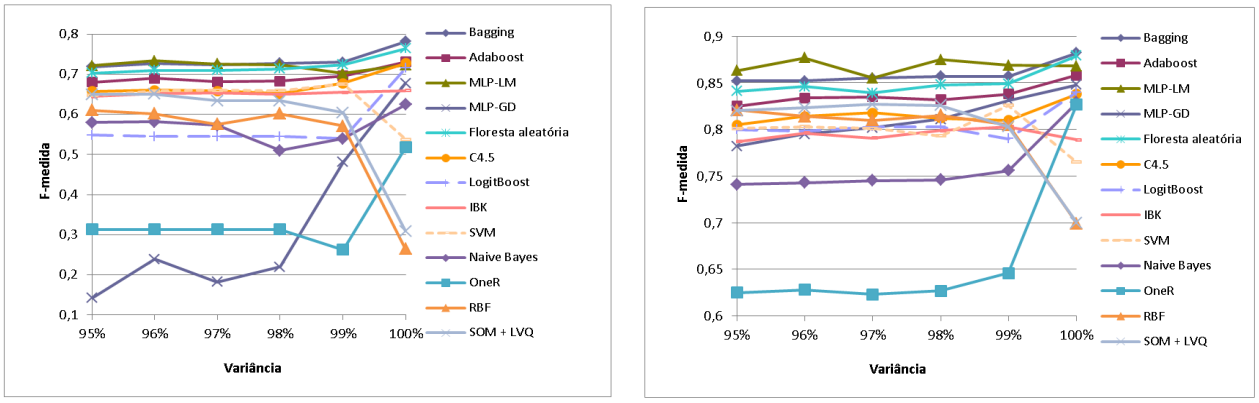
As Figuras 5.1, 5.2 e 5.3 mostram a variação dos resultados em relação ao número de dimensões reduzidas, sendo usada como medida de desempenho a F-medida. Em cada figura, "100%" de variância mostra os resultados obtidos sem aplicação da técnica de PCA.

Através dos resultados apresentados nas figuras é possível observar que, em geral, houve uma queda de desempenho dos classificadores após a redução de dimensionalidade por meio do método PCA. Os métodos mais afetados foram os que tiveram melhores resultados na classificação de *Web spam*, conforme foi apresentado na seção anterior, tais como o *bagging*, *AdaBoost*, MLP-GD e floresta aleatória. A rede neural MLP-LM também foi um dos melhores métodos



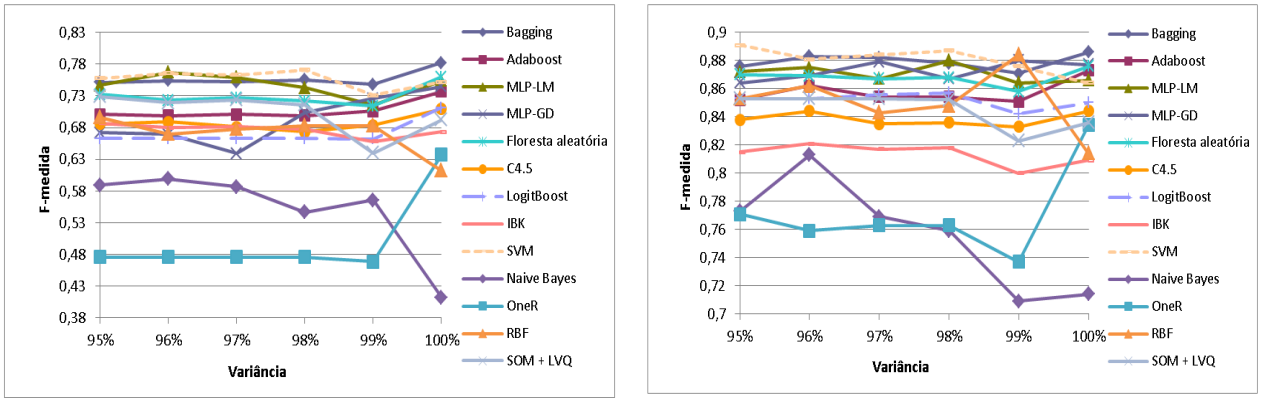
(a) Classes desbalanceadas. (b) Classes balanceadas.

Figura 5.1: Resultados obtidos na detecção de *Web spam* usando vetores de atributos baseados no conteúdo e com dimensionalidade reduzida.



(a) Classes desbalanceadas. (b) Classes balanceadas.

Figura 5.2: Resultados obtidos na detecção de *Web spam* usando vetores de atributos baseados nos *links* e com dimensionalidade reduzida.



(a) Classes desbalanceadas. (b) Classes balanceadas.

Figura 5.3: Resultados obtidos na detecção de *Web spam* usando vetores de atributos baseados nos *links* transformados e com dimensionalidade reduzida.

na detecção de *Web spam*, porém diferente do que ocorreu com os outros métodos, ela foi beneficiada pela redução da dimensionalidade de quase todos os vetores de atributos, exceto dos vetores baseados no conteúdo quando as classes usadas foram balanceadas (Figura 5.1(b)). Em alguns cenários, a rede neural MLP-LM teve melhores resultados usando PCA com variância de 98% (Figuras 5.1(a) e 5.3(b)), enquanto que em outros o desempenho foi melhor com 96% de variância (Figuras 5.2 e 5.3(a)).

Alguns métodos de classificação que obtiveram resultados insatisfatórios usando os vetores de atributos na sua forma original, obtiveram melhor desempenho quando foi aplicada a técnica de PCA, tais como o *naive* Bayes, quando usou os vetores baseados no conteúdo com classes desbalanceadas (Figura 5.1(a)) e os vetores de atributos baseados nos *links* transformados (Figura 5.3), e o IBK, quando usou características baseadas nos *links* transformados. No entanto, entre os métodos com piores resultados, os maiores beneficiados com a redução de dimensionalidade foram as redes neurais RBF e SOM + LVQ. Em todos os cenários esses dois métodos melhoraram o desempenho após a aplicação da técnica de PCA e em alguns deles deixaram de estar entre os cinco piores métodos.

5.3.2 Combinação de vetores de atributos extraídos da base de dados WEBSPAM-UK2006

Um dos problemas encontrados na detecção automática de *Web spam* é que algumas páginas *Web* usam mais de uma técnica de *spamming* para enganar os motores de busca. É comum que os *spammers* criem páginas que usem ao mesmo tempo *spam links* e conteúdo *spam*, pois normalmente os motores de busca atribuem maior relevância para as páginas *Web* que tenham uma estrutura de *links* que indica que ela seja importante para a comunidade *Web*, em relação a um ou mais termos de busca, ao mesmo tempo que seu conteúdo aborde esse(s) termo(s) de forma adequada. Então, por exemplo, uma página *Web* que aborde o assunto "copa do mundo", com título, textos, descrição de imagens ou outros elementos relacionados a esse assunto, será considerada ainda mais importante se possuir *links* direcionados a *sites* também considerados relevantes sobre esse assunto e, principalmente, se receber *links* direcionados por *sites* relevantes sobre futebol, grandes eventos esportivos ou outras categorias relacionadas.

Diante disso, nessa seção são apresentados os resultados de experimentos realizados com todas as combinações entre os vetores de atributos. O objetivo é melhorar os resultados da classificação de *Web spam* identificando *hosts* híbridos, que possuem páginas *Web* que empregam conteúdo *spam* e *spam links*, e analisar o desempenho dos métodos nesses novos cenários.

As Tabelas de 5.7 a 5.10 apresentam, respectivamente, os resultados de cada método na classificação das seguintes combinações de atributos: conteúdo com *links*, *links* com *links* transformados, *links* transformados com conteúdo e *links* com *links* transformados com conteúdo. Cada coluna apresenta a média e o desvio padrão dos resultados. Os valores em negrito precedidos pelo símbolo "↑" indicam os melhores resultados, enquanto os valores em negrito precedidos pelo símbolo "↓" indicam os piores resultados obtidos usando cada configuração de classes (balanceadas ou desbalanceadas). Os valores em negrito precedidos pelo símbolo "*" indicam os melhores ou piores resultados considerando todas as combinações de atributos.

Os resultados apresentados nas Tabelas de 5.7 a 5.10 indicam que a combinação de atributos

Tabela 5.7: Resultados obtidos na detecção de *Web spam* usando combinação dos atributos baseados nos *links* e no conteúdo.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes desbalanceadas				
AdaBoost	***↑92,5** ± 0,6	80,4 ± 1,9	**↑96,1** ± 0,4	***↑86,3** ± 1,2	***↑0,832** ± 0,014
MLP-LM	92,1 ± 0,6	81,7 ± 2,0	95,3 ± 0,2	84,4 ± 1,9	0,830 ± 0,008
Bagging	92,0 ± 0,7	81,1 ± 1,8	95,3 ± 0,6	84,0 ± 1,7	0,825 ± 0,016
Floresta aleatória	91,0 ± 0,3	74,4 ± 2,5	96,0 ± 1,1	85,0 ± 2,4	0,793 ± 0,006
MLP-GD	89,3 ± 0,6	74,0 ± 3,4	94,1 ± 0,9	79,7 ± 2,7	0,767 ± 0,017
C4,5	88,2 ± 0,5	75,9 ± 2,3	91,9 ± 0,3	73,8 ± 0,9	0,748 ± 0,014
LogitBoost	88,5 ± 0,6	70,2 ± 2,6	93,9 ± 0,7	77,7 ± 2,3	0,737 ± 0,019
IBK	86,7 ± 0,6	68,7 ± 2,3	92,1 ± 0,7	72,5 ± 2,0	0,705 ± 0,015
Naive Bayes	72,7 ± 0,9	95,3 ± 1,0	**↓65,8** ± 1,3	**↓45,9** ± 0,8	0,620 ± 0,008
SVM	83,6 ± 0,8	52,4 ± 2,0	93,0 ± 0,8	69,6 ± 2,5	0,598 ± 0,018
OneR	78,5 ± 0,9	50,1 ± 2,4	87,2 ± 1,2	54,4 ± 2,2	0,521 ± 0,018
SOM + LVQ	***↓24,2** ± 2,3	**↑95,7** ± 0,9	79,1 ± 0,7	63,5 ± 4,7	0,349 ± 0,027
RBF	77,8 ± 0,5	***↓20,1** ± 1,0	95,2 ± 0,4	56,3 ± 2,7	***↓0,296** ± 0,014
	Classes Balanceadas				
AdaBoost	**↑90,3** ± 0,6	92,5 ± 1,3	**↑88,1** ± 1,3	**↑88,7** ± 1,0	**↑0,905** ± 0,006
Bagging	90,1 ± 0,6	**↑93,2** ± 0,8	87,1 ± 1,2	87,8 ± 1,0	0,904 ± 0,005
MLP-GD	89,2 ± 1,0	93,0 ± 1,7	85,5 ± 2,2	86,4 ± 2,2	0,895 ± 0,011
Floresta aleatória	89,0 ± 0,7	92,1 ± 2,1	85,7 ± 1,9	86,9 ± 1,2	0,894 ± 0,008
MLP-LM	88,4 ± 2,1	91,9 ± 3,5	85,0 ± 3,3	85,6 ± 2,7	0,886 ± 0,021
LogitBoost	86,4 ± 0,5	89,9 ± 1,3	82,9 ± 2,0	84,3 ± 1,2	0,870 ± 0,005
C4,5	85,7 ± 0,8	86,9 ± 2,2	84,5 ± 1,4	85,1 ± 0,9	0,860 ± 0,009
IBK	83,1 ± 0,8	84,7 ± 2,7	81,5 ± 1,6	82,4 ± 1,0	0,835 ± 0,012
Naive Bayes	80,8 ± 1,2	92,7 ± 6,5	68,9 ± 6,7	75,2 ± 3,5	0,828 ± 0,016
OneR	81,5 ± 1,2	88,8 ± 1,7	74,2 ± 1,8	77,5 ± 1,3	0,827 ± 0,012
SVM	76,0 ± 1,5	71,4 ± 1,7	80,6 ± 2,6	78,7 ± 2,3	0,749 ± 0,015
RBF	68,7 ± 2,3	**↓65,0** ± 1,6	72,4 ± 4,5	70,3 ± 3,4	0,675 ± 0,018
SOM + LVQ	**↓64,6** ± 3,6	70,7 ± 3,8	**↓67,7** ± 2,4	**↓68,9** ± 3,0	**↓0,666** ± 0,026

Tabela 5.8: Resultados obtidos na detecção de *Web spam* usando combinação dos atributos baseados nos *links* e nos *links* transformados.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes desbalanceadas				
Bagging	**↑89,7** ± 0,8	78,5 ± 3,0	93,1 ± 0,5	**↑77,6** ± 1,5	**↑0,780** ± 0,020
Floresta aleatória	89,0 ± 1,0	78,7 ± 2,2	92,2 ± 0,9	75,4 ± 2,3	0,770 ± 0,020
MLP-GD	89,1 ± 1,5	76,9 ± 3,8	92,7 ± 1,5	76,1 ± 4,5	0,764 ± 0,031
MLP-LM	88,8 ± 1,8	75,8 ± 4,7	92,9 ± 1,2	76,6 ± 4,1	0,761 ± 0,038
AdaBoost	88,3 ± 0,6	73,9 ± 2,2	92,6 ± 0,5	75,3 ± 1,3	0,746 ± 0,015
C4,5	86,4 ± 0,9	71,8 ± 3,3	90,9 ± 1,1	70,6 ± 2,2	0,712 ± 0,019
LogitBoost	86,4 ± 0,7	69,9 ± 4,5	91,4 ± 1,7	71,4 ± 2,7	0,705 ± 0,018
IBK	84,3 ± 0,8	67,2 ± 2,0	89,5 ± 0,7	66,0 ± 1,8	0,666 ± 0,017
SVM	83,6 ± 1,0	62,5 ± 2,1	90,0 ± 1,3	65,6 ± 2,8	0,639 ± 0,018
Naive Bayes	73,5 ± 0,8	**↑94,9** ± 1,1	**↓67,0** ± 1,0	**↓46,7** ± 0,8	0,626 ± 0,008
OneR	82,4 ± 0,7	61,6 ± 2,0	88,7 ± 0,7	62,4 ± 1,6	0,620 ± 0,015
SOM + LVQ	**↓27,6** ± 2,6	94,3 ± 0,7	78,8 ± 0,5	59,6 ± 2,3	0,377 ± 0,025
RBF	78,4 ± 0,8	**↓22,6** ± 1,2	**↑95,4** ± 0,8	60,0 ± 4,9	**↓0,328** ± 0,019
	Classes Balanceadas				
Bagging	**↑88,2** ± 1,2	91,6 ± 1,3	84,9 ± 1,9	**↑85,8** ± 1,6	**↑0,886** ± 0,011
Floresta aleatória	87,7 ± 0,6	92,1 ± 0,6	83,3 ± 1,5	84,7 ± 1,1	0,882 ± 0,005
MLP-GD	87,6 ± 2,5	91,7 ± 1,6	83,4 ± 3,9	84,6 ± 3,4	0,880 ± 0,023
AdaBoost	86,9 ± 0,9	89,7 ± 0,9	84,2 ± 1,4	85,0 ± 1,2	0,873 ± 0,008
LogitBoost	85,8 ± 0,8	90,3 ± 1,3	81,4 ± 2,2	83,0 ± 1,5	0,864 ± 0,006
MLP-LM	86,7 ± 2,7	87,7 ± 2,9	**↑85,8** ± 3,7	85,2 ± 3,5	0,863 ± 0,026
OneR	84,0 ± 0,9	87,8 ± 2,0	80,1 ± 1,9	81,6 ± 1,3	0,846 ± 0,009
C4,5	84,2 ± 1,1	84,7 ± 1,5	83,6 ± 1,5	83,8 ± 1,3	0,842 ± 0,011
Naive Bayes	80,8 ± 1,5	**↑94,7** ± 0,7	**↓66,9** ± 3,0	74,1 ± 1,7	0,832 ± 0,011
SVM	80,9 ± 1,0	84,0 ± 1,7	77,7 ± 1,2	79,0 ± 0,9	0,814 ± 0,010
IBK	80,6 ± 1,4	83,2 ± 1,6	78,0 ± 1,6	79,1 ± 1,4	0,811 ± 0,014
SOM + LVQ	**↓69,8** ± 3,7	72,7 ± 1,9	71,3 ± 1,7	71,9 ± 1,4	0,708 ± 0,023
RBF	69,9 ± 2,0	**↓68,5** ± 1,7	71,3 ± 3,4	**↓70,6** ± 2,6	**↓0,695** ± 0,018

Tabela 5.9: Resultados obtidos na detecção de *Web spam* usando combinação dos atributos baseados nos *links* transformados e no conteúdo.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes desbalanceadas				
AdaBoost	↑**92,4** ± 0,8	81,1 ± 2,7	95,8 ± 0,5	↑**85,6** ± 1,6	***↑0,832** ± 0,018
Bagging	92,1 ± 0,7	82,2 ± 2,4	95,1 ± 0,4	83,6 ± 1,4	0,829 ± 0,017
Floresta aleatória	91,0 ± 0,3	81,0 ± 2,0	94,1 ± 0,3	80,7 ± 0,6	0,808 ± 0,010
MLP-GD	90,3 ± 0,8	78,5 ± 2,1	94,0 ± 0,8	79,8 ± 3,2	0,791 ± 0,021
MLP-LM	89,9 ± 1,1	77,3 ± 4,3	93,7 ± 1,5	79,0 ± 4,2	0,780 ± 0,032
LogitBoost	88,9 ± 0,5	73,0 ± 3,5	93,7 ± 0,8	77,9 ± 1,7	0,753 ± 0,016
C4,5	87,5 ± 0,8	72,9 ± 1,8	91,9 ± 1,0	73,4 ± 2,3	0,731 ± 0,015
IBK	87,1 ± 0,4	68,0 ± 1,8	92,9 ± 0,6	74,5 ± 1,4	0,711 ± 0,010
OneR	83,4 ± 1,3	63,3 ± 2,1	89,5 ± 1,3	64,9 ± 3,3	0,641 ± 0,024
SVM	85,1 ± 0,6	50,7 ± 2,0	95,6 ± 0,5	77,7 ± 2,2	0,613 ± 0,018
SOM + LVQ	↓**33,2** ± 3,8	***↑96,0** ± 0,6	81,4 ± 0,7	71,5 ± 2,5	0,452 ± 0,037
RBF	81,0 ± 0,6	↓**29,9** ± 2,4	***↑96,5** ± 0,4	71,9 ± 2,8	0,422 ± 0,027
Naive Bayes	36,1 ± 0,9	***↑96,0** ± 1,1	***↓17,9** ± 1,1	***↓26,2** ± 0,3	↓**0,412** ± 0,005
	Classes Balanceadas				
AdaBoost	***↑91,4** ± 0,7	93,7 ± 1,2	↑**89,2** ± 1,4	↑**89,7** ± 1,1	***↑0,916** ± 0,007
Bagging	90,0 ± 0,8	92,7 ± 1,1	87,3 ± 1,2	88,0 ± 1,0	0,903 ± 0,008
MLP-GD	89,6 ± 1,6	91,2 ± 2,0	88,0 ± 2,4	88,2 ± 2,6	0,897 ± 0,016
Floresta aleatória	89,1 ± 0,7	92,5 ± 0,8	85,6 ± 1,1	86,6 ± 0,9	0,894 ± 0,006
MLP-LM	89,1 ± 1,8	90,5 ± 2,9	87,8 ± 2,8	88,2 ± 2,1	0,893 ± 0,017
LogitBoost	87,2 ± 1,1	90,8 ± 1,7	83,7 ± 2,1	84,8 ± 1,6	0,877 ± 0,011
C4,5	85,9 ± 1,1	86,8 ± 1,5	85,0 ± 1,9	85,3 ± 1,5	0,860 ± 0,010
OneR	83,1 ± 0,7	87,3 ± 2,0	78,9 ± 1,8	80,6 ± 1,1	0,838 ± 0,008
IBK	83,3 ± 1,1	81,5 ± 2,0	85,1 ± 1,5	84,6 ± 1,3	0,830 ± 0,012
SVM	77,9 ± 1,1	66,5 ± 2,4	↑**89,2** ± 1,7	86,1 ± 1,7	0,750 ± 0,015
Naive Bayes	***↓61,7** ± 4,7	***↑96,5** ± 0,9	***↓26,9** ± 9,9	***↓57,1** ± 3,5	0,717 ± 0,026
SOM + LVQ	63,8 ± 4,6	77,4 ± 2,9	70,6 ± 1,7	73,9 ± 1,9	0,684 ± 0,026
RBF	69,1 ± 1,2	***↓51,6** ± 3,1	86,6 ± 3,1	79,6 ± 3,2	***↓0,625** ± 0,019

Tabela 5.10: Resultados obtidos na detecção de *Web spam* usando combinação de todos os atributos (*links*, *links* transformados e conteúdo).

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes desbalanceadas				
AdaBoost	↑**92,1** ± 0,7	81,6 ± 1,2	95,3 ± 0,9	↑**84,2** ± 2,7	↑**0,829** ± 0,014
Bagging	91,8 ± 1,0	82,1 ± 2,0	94,8 ± 1,0	82,7 ± 2,9	0,824 ± 0,019
Floresta aleatória	91,2 ± 0,7	82,4 ± 1,1	93,9 ± 0,9	80,4 ± 2,3	0,814 ± 0,012
MLP-LM	90,6 ± 1,0	81,7 ± 3,7	93,3 ± 0,7	79,2 ± 2,5	0,804 ± 0,027
MLP-GD	90,2 ± 1,0	79,9 ± 2,7	93,4 ± 0,9	78,9 ± 3,3	0,793 ± 0,024
LogitBoost	89,3 ± 0,9	75,9 ± 2,3	93,3 ± 0,8	77,6 ± 2,2	0,767 ± 0,020
C4,5	87,4 ± 0,6	73,4 ± 2,7	91,7 ± 1,0	73,0 ± 2,0	0,731 ± 0,014
IBK	86,4 ± 0,6	69,2 ± 1,4	91,7 ± 0,6	71,6 ± 1,6	0,704 ± 0,013
OneR	82,9 ± 0,7	63,5 ± 2,5	88,8 ± 0,8	63,3 ± 1,6	0,634 ± 0,017
SVM	84,4 ± 0,7	55,9 ± 2,0	93,1 ± 0,5	71,1 ± 1,9	0,626 ± 0,018
Naive Bayes	73,4 ± 0,7	94,8 ± 1,2	↓**66,9** ± 1,0	↓**46,6** ± 0,7	0,625 ± 0,006
SOM + LVQ	↓**25,5** ± 4,1	↑**95,4** ± 1,6	79,2 ± 0,8	63,8 ± 6,0	0,361 ± 0,040
RBF	78,3 ± 0,5	↓**20,2** ± 3,2	↑**95,9** ± 0,4	60,0 ± 2,5	↓**0,302** ± 0,039
	Classes Balanceadas				
AdaBoost	↑**91,3** ± 1,0	93,2 ± 0,9	***↑89,4** ± 1,5	***↑89,8** ± 1,3	↑**0,915** ± 0,010
MLP-GD	90,4 ± 1,0	92,4 ± 1,4	88,3 ± 1,6	88,8 ± 1,5	0,906 ± 0,010
Bagging	90,2 ± 1,1	92,5 ± 1,3	87,9 ± 1,6	88,4 ± 1,3	0,904 ± 0,010
Floresta aleatória	89,4 ± 0,8	92,7 ± 1,5	86,1 ± 1,6	87,0 ± 1,2	0,897 ± 0,008
MLP-LM	87,5 ± 3,7	89,7 ± 4,6	85,2 ± 5,4	86,1 ± 4,0	0,878 ± 0,034
LogitBoost	86,9 ± 1,1	89,8 ± 2,2	84,0 ± 1,9	84,9 ± 1,5	0,873 ± 0,011
C4,5	85,6 ± 1,6	86,3 ± 2,6	84,9 ± 2,0	85,1 ± 1,7	0,857 ± 0,017
OneR	83,1 ± 0,8	86,8 ± 2,0	79,4 ± 1,4	80,9 ± 0,9	0,837 ± 0,009
Naive Bayes	81,0 ± 1,5	↑**93,8** ± 1,2	↓**68,2** ± 2,6	74,7 ± 1,6	0,832 ± 0,012
IBK	83,0 ± 1,7	83,0 ± 2,3	82,9 ± 2,5	83,0 ± 2,1	0,830 ± 0,017
SVM	80,4 ± 1,1	78,2 ± 1,4	82,6 ± 1,9	81,9 ± 1,5	0,800 ± 0,011
SOM + LVQ	↓**67,1** ± 3,2	71,0 ± 3,7	69,0 ± 1,3	↓**69,9** ± 2,1	0,684 ± 0,015
RBF	69,5 ± 1,7	↓**64,6** ± 1,5	74,5 ± 2,7	71,8 ± 2,3	↓**0,679** ± 0,016

é mais eficiente na classificação de *Web spam* do que usar os vetores de atributos de forma isolada. Apenas com a combinação de atributos baseados nos *links* e nos *links* transformados não foi obtido nenhum resultado superior ao melhor resultado obtido sem combinar vetores de atributos. Com classes desbalanceadas, o melhor resultado dessa combinação de atributos foi 0,780 (Tabela 5.8), enquanto nas simulações com atributos baseados nos *links* transformados foi 0,782 (Tabela 5.5). Com classes balanceadas, o melhor resultado dessa combinação de atributos foi 0,886, que é igual ao melhor resultado das simulações com atributos baseados nos *links* transformados.

A combinação de atributos que gerou os melhores resultados foi a combinação de atributos baseados nos *links* transformados com atributos baseados no conteúdo, que obteve as melhores taxas de acurácia e de F-medida quando os classificadores foram treinados com classes balanceadas, conforme pode ser visto na Tabela 5.9. Porém, diante dos resultados apresentados na Tabela 5.4, nas simulações com classes desbalanceadas, a combinação de atributos baseados nos *links* e no conteúdo é mais efetiva, pois obteve a melhor acurácia, a melhor precisão e F-medida igual à obtida pela combinação de atributos baseados nos *links* transformados com atributos baseados no conteúdo. Os resultados apresentados pela Tabela 5.10 indicam que a combinação de todos os vetores de atributos também gera bons resultados. Porém, considerando que essa combinação possui a maior dimensionalidade e, consequentemente, necessita de mais recursos computacionais, pode-se inferir que as outras combinações são mais recomendadas.

Outro ponto importante é que, novamente, os métodos de aprendizado tiveram desempenho bem superior quando foram usadas classes balanceadas. Por exemplo, nas simulações apresentadas na Tabela 5.7, em que foram combinados os vetores de atributos baseados no conteúdo e nos *links*, as taxas de precisão obtidas com classes balanceadas foram, em média, 9.6% superiores ao que foi obtido com classes desbalanceadas, enquanto que em relação as taxas de sensitividade essa diferença foi, em média, 14,8%.

Em relação aos métodos de aprendizado de máquina, o que obteve os melhores resultados foi o método *AdaBoost*, diferente do que ocorreu nos cenários em que os vetores de atributos não foram combinados, onde o método *bagging* foi melhor, conforme apresentado na Seção 5.3. Usando combinação de vetores de atributos, o método *bagging* só foi superior ao método *AdaBoost* quando os atributos baseados nos *links*, juntamente com os atributos baseados nos *links* transformados, foram usados para treinamento e teste, como pode ser visto na Tabela 5.8.

Nesses novos cenários, as redes neurais MLP também tiveram bons resultados, mantendo-se entre os cinco melhores métodos, assim como ocorreu com o método floresta aleatória. Por outro lado, as redes neurais RBF e SOM + LVQ tiveram os piores resultados, sendo inferiores até mesmo a um dos métodos mais simples de classificação de padrões da literatura, o *OneR*. Outro método que merece atenção é o *naive* Bayes que, nos cenários com combinação de atributos, obteve resultados bem melhores do que nos cenários em que os vetores de atributos não foram combinados e, desta forma, saiu do grupo dos quatro piores métodos.

Análise do impacto da redução de dimensionalidade na combinação dos vetores de atributos extraídos da base de dados WEBSPAM-UK2006

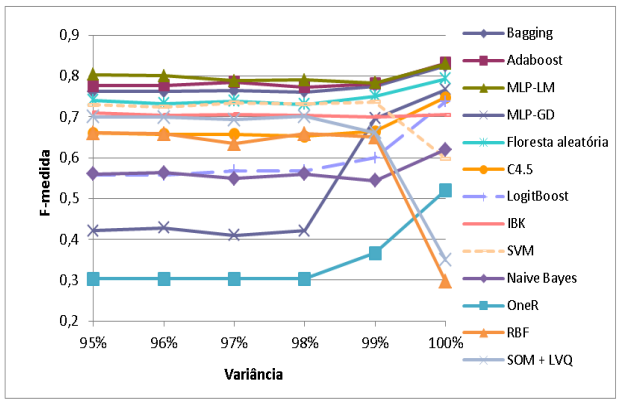
Nessa seção, é apresentada uma análise dos efeitos da redução de dimensionalidade por meio da técnica de análise de componentes principais na combinação dos vetores de atributos. O objetivo dessa análise é verificar se é possível melhorar os resultados obtidos pelas combinações dos vetores de atributos e ao mesmo tempo reduzir o custo computacional do processo de aprendizado, com a diminuição do número de atributos que representa cada amostra de *host*. Acredita-se que a combinação de atributos aumenta a possibilidade de ocorrência de redundância que possa atrapalhar o aprendizado dos classificadores. Esse problema pode ser amenizado com a aplicação da técnica de PCA.

Como foi feito na seção 5.3.1, a técnica de PCA foi aplicada usando variância entre 95% e 99%. A Tabela 5.11 apresenta a redução em número de dimensões em relação a cada porcentagem de variância utilizada. A última coluna que está rotulada como "100%" apresenta o número de dimensões original dos vetores formados pela combinação dos vetores de atributos.

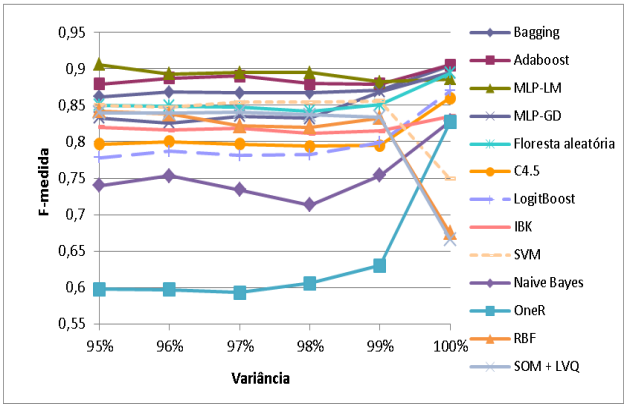
Tabela 5.11: Número de dimensões obtidas após redução da dimensionalidade das combinações dos vetores de atributos.

	Variância					
	95%	96%	97%	98%	99%	100%
Tipos de atributos	Nº de dimensões					
Links + conteúdo	57	61	67	75	86	137
Links + *Links* transformados	50	54	60	68	79	179
Links transformados + conteúdo	77	83	92	103	119	234
Links + *Links* transformados + conteúdo	87	95	105	118	138	275

As Figuras de 5.4 a 5.7 mostram a variação do valor da F-medida em relação ao número de dimensões reduzidas. Em cada figura, "100%" de variância mostra os resultados obtidos sem aplicar redução de dimensionalidade nas combinações dos vetores de atributos.



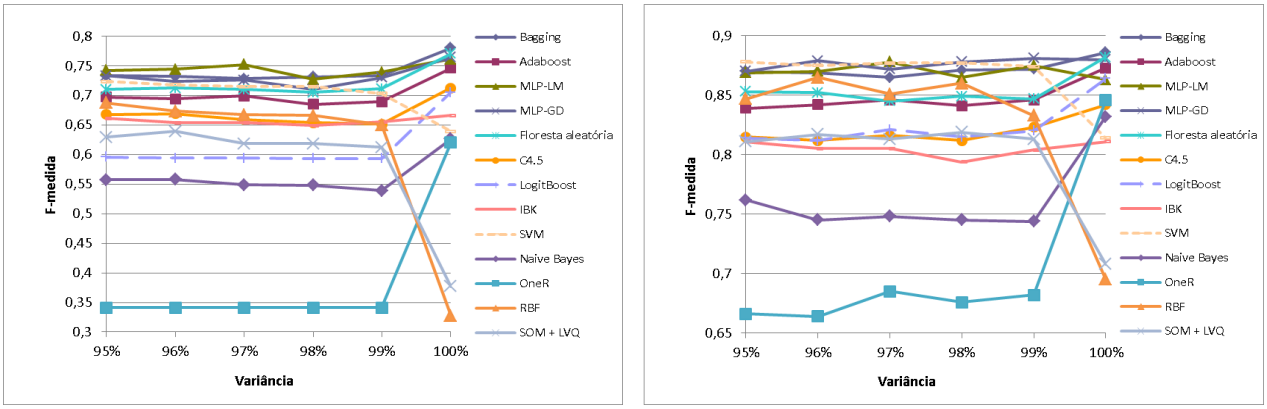
(a) Classes desbalanceadas.



(b) Classes balanceadas.

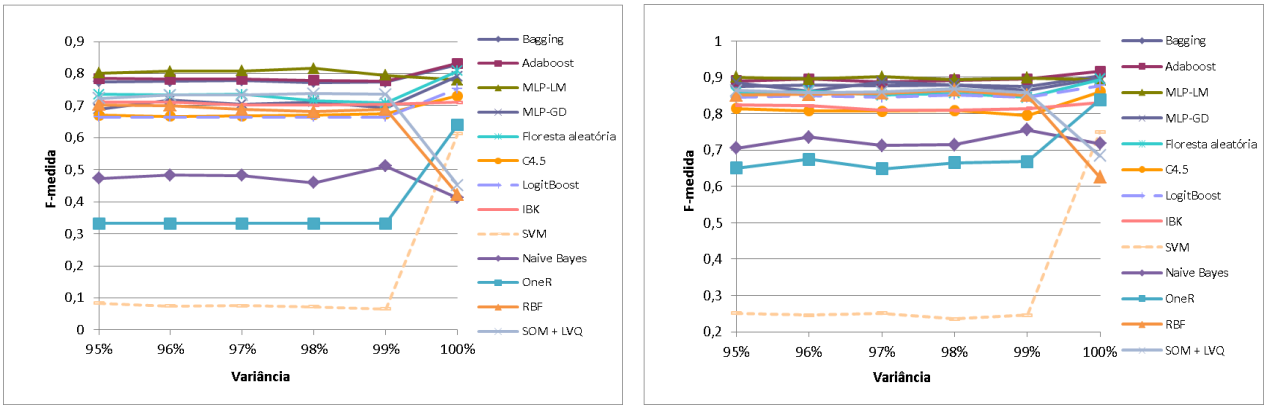
Figura 5.4: Resultados obtidos na detecção de *Web spam* usando combinação dos atributos baseados no conteúdo e nos *links* e com dimensionalidade reduzida.

Os resultados obtidos com as combinações de atributos após a aplicação da técnica de PCA mostram que, para a maioria dos métodos, reduzir a dimensionalidade do espaço dos atributos



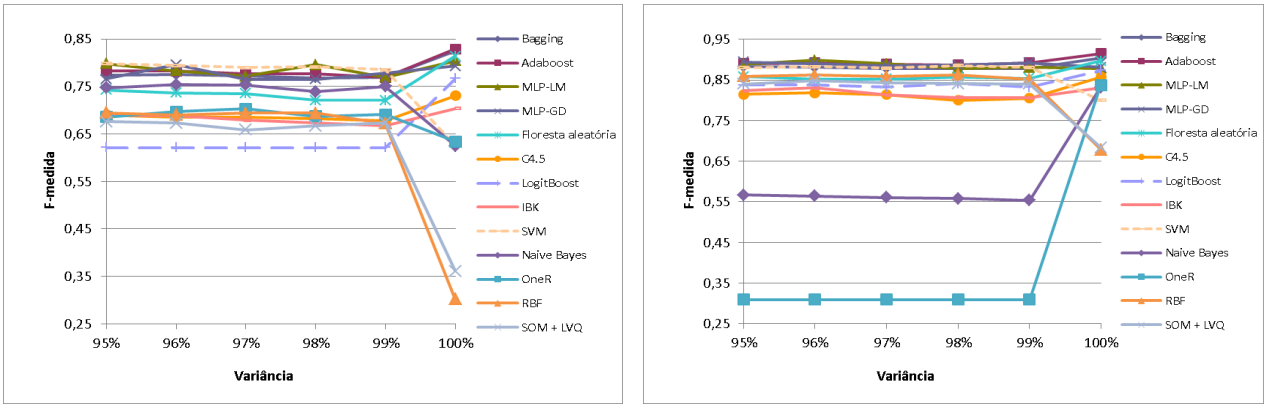
(a) Classes desbalanceadas. (b) Classes balanceadas.

Figura 5.5: Resultados obtidos na detecção de *Web spam* usando combinação dos atributos baseados nos *links* e nos *links* transformados e com dimensionalidade reduzida.



(a) Classes desbalanceadas. (b) Classes balanceadas.

Figura 5.6: Resultados obtidos na detecção de *Web spam* usando combinação dos atributos baseados nos *links* transformados e no conteúdo e com dimensionalidade reduzida.



(a) Classes desbalanceadas. (b) Classes balanceadas.

Figura 5.7: Resultados obtidos na detecção de *Web spam* usando combinação de todos os tipos de atributos (*links*, *links* transformados e conteúdo) e com dimensionalidade reduzida.

prejudica o desempenho do processo de classificação de *Web spam*, assim como ocorreu nas simulações com os vetores de atributos treinados isoladamente (Seção 5.3.1). Os melhores métodos são os mais prejudicados com a redução de dimensionalidade. A única exceção é a rede neural MLP-LM que obteve melhores resultados em todos os experimentos com classes balanceadas e combinações de atributos, após a aplicação da técnica de PCA. Com classes desbalanceadas, a técnica de PCA melhorou o desempenho da rede neural MLP-LM apenas quando foi aplicada sobre a combinação dos atributos baseados nos *links* transformados e no conteúdo, como apresentado na Figura 5.6(a).

Alguns métodos foram prejudicados pela redução de dimensionalidade de algumas combinações de vetores de atributos, enquanto foram beneficiados quando o mesmo processo foi realizado com outras combinações. O método SVM, por exemplo, foi prejudicado pela técnica de PCA nas simulações em que as amostras de treinamento foram formadas pela combinação de vetores baseados nos *links* e nos *links* transformados (Figura 5.5) e foi beneficiado quando a técnica de PCA foi aplicada nas demais combinações. Em contrapartida, as redes neurais RBF e SOM + LVQ melhoraram bastante seu desempenho após a redução da dimensionalidade de todas as combinações de vetores de atributos. Por exemplo, na simulação com combinação entre os vetores de atributos baseados no conteúdo e nos *links* transformados (Figura 5.6), quando a técnica de PCA foi aplicada usando uma variância de 95% e foram usadas classes desbalancedas, o resultado obtido pela rede neural RBF foi 28,1% melhor que o resultado obtido sem PCA. No mesmo cenário, só que usando classes balanceadas, o resultado da rede neural RBF, quando a técnica de PCA foi aplicada usando uma variância de 98%, foi 17% superior ao resultado obtido sem aplicação da técnica de PCA. Isso mostra que a redução de dimensionalidade pode ser efetiva ou não na detecção de *Web spam* dependendo do método de aprendizado e dos vetores de atributos utilizados.

5.4 Resultados obtidos com a base de dados WEBSPAM-UK2007

Nessa seção, são apresentados os resultados da classificação de *Web spam* usando conjuntos de vetores de atributos pré-computados extraídos das base de dados WEBSPAM-UK2007. Esses conjuntos foram disponibilizados pelos organizadores da competição *Web Spam Challenge 2008*. Cada conjunto é formado por 5.476 (94,5%) amostras de *ham hosts* e 321 (5,5%) amostras de *spam hosts*.

Como pode ser visto, o problema de desbalanceamento entre as classes da base de dados WEBSPAM-UK2007 é muito maior do que o problema de desbalanceamento da base de dados WEBSPAM-UK2006. Diante disso, ficou inviável utilizar apenas a técnica de balanceamento por subamostragem aleatória para igualar o número de representantes de cada categoria do problema de *Web spam*, pois seria necessário descartar 5.155 amostras *ham*. Esse alto número de amostras descartadas pode prejudicar o aprendizado dos algoritmos de classificação, devido a grande perda de informações. Portanto, a solução encontrada foi usar a técnica SMOTE, apresentada na Seção 4.3.5, que também é uma técnica de balanceamento de dados amplamente empregada na literatura.

SMOTE é uma técnica de sobreamostragem que cria amostras sintéticas para a classe minoritária usando conhecimentos sobre as amostras já existentes dessa classe. Usando apenas essa técnica, seria necessário criar 5.155 exemplos *spam* sintéticos para balancear as classes do problema. Esse alto número de exemplos sintéticos poderia provocar o problema de *overfitting* (sobreajustamento), que é o ajuste demasiado nos dados de treinamento. Então, para evitar esse problema, a técnica SMOTE foi usada para gerar 1.605 novas amostras para a categoria *spam*, que passou a ter 1.926 representantes. Então, para completar o balanceamento entre as classes, a técnica de subamostragem aleatória foi aplicada em cada simulação, descartando 3.550 exemplos da categoria *ham*. Com isso, cada categoria do problema passou a ter 1.926 representantes.

A aplicação das técnicas SMOTE e subamostragem aleatória permitiram a realização de simulações usando classes balanceadas. No entanto, ficou inviável realizar simulações com classes desbalanceadas, pois muitos métodos de classificação sequer conseguiram detectar alguma amostra de *spam* durante a fase de teste, já que apenas 321 exemplos de *spam* foram insuficientes para o aprendizado. Dessa forma, foram feitas simulações apenas com classes balanceadas.

As Tabelas 5.12, 5.13 e 5.14 apresentam os resultados de cada método na classificação dos atributos baseados no conteúdo, nos *links* e nos *links* transformados usando classes balanceadas pelas técnicas SMOTE e subamostragem aleatória. Cada coluna das tabelas apresenta a média e o desvio padrão dos resultados obtidos usando a técnica de validação por subamostragem aleatória, conforme apresentado na Seção 5.2. Os valores em negrito precedidos pelo símbolo "↑" indicam os melhores resultados, enquanto os valores em negrito precedidos pelo símbolo "↓" indicam os piores resultados. Por outro lado, os valores em negrito precedidos pelo símbolo "*" indicam os melhores ou piores resultados considerando os três tipos de vetores de atributos.

Tabela 5.12: Resultados obtidos na detecção de *Web spam* usando atributos baseados no conteúdo.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes balanceadas				
Bagging	***↑89,7** ± 1,3	89,2 ± 1,3	90,2 ± 1,7	***↑90,0** ± 2,0	***↑0,896** ± 0,014
AdaBoost	88,0 ± 2,2	88,1 ± 2,9	87,9 ± 2,3	87,9 ± 2,2	0,880 ± 0,022
Floresta aleatória	86,6 ± 1,2	89,5 ± 1,4	83,8 ± 1,7	84,6 ± 1,4	0,870 ± 0,011
IBK	85,8 ± 1,3	83,8 ± 2,7	87,8 ± 2,0	87,3 ± 1,6	0,855 ± 0,015
MLP-LM	82,9 ± 2,1	83,1 ± 4,1	82,5 ± 4,0	82,2 ± 2,5	0,826 ± 0,023
C4.5	79,6 ± 1,6	79,6 ± 1,8	79,6 ± 2,5	79,6 ± 2,1	0,796 ± 0,015
OneR	80,1 ± 2,2	74,8 ± 4,6	85,3 ± 4,1	83,7 ± 3,6	0,789 ± 0,026
MLP-GD	76,2 ± 1,8	74,8 ± 2,9	77,7 ± 4,3	77,2 ± 3,3	0,759 ± 0,019
LogitBoost	75,0 ± 0,9	75,3 ± 2,9	74,6 ± 3,0	74,8 ± 1,7	0,750 ± 0,011
Naive Bayes	***↓52,6** ± 2,0	***↑95,0** ± 2,3	***↓10,2** ± 4,4	***↓51,4** ± 1,1	0,667 ± 0,011
SOM + LVQ	56,7 ± 2,3	83,0 ± 3,2	69,9 ± 1,4	77,1 ± 3,2	0,653 ± 0,016
RBF	68,8 ± 2,3	53,4 ± 3,0	84,3 ± 2,3	77,3 ± 3,1	0,631 ± 0,029
SVM	70,6 ± 0,9	↓**49,8** ± 2,0	***↑91,5** ± 1,1	85,4 ± 1,5	↓**0,628** ± 0,016

Os resultados obtidos com a base de dados WEBSPAM-UK2007 são melhores, mas semelhantes aos valores obtidos com a base de dados WEBSPAM-UK2006. Porém, nos experimentos com essa nova base de dados, os algoritmos de aprendizado tem melhor desempenho nas simulações que usam vetores de atributos baseados no conteúdo das páginas. Na base de dados anterior, os resultados foram melhores quando os vetores de atributos baseados nos *links* transformados foram usados nos experimentos.

Em relação aos algoritmos de aprendizado, assim como ocorreu na base de dados WEBSPAM-

Tabela 5.13: Resultados obtidos na detecção de *Web spam* usando atributos baseados nos *links*.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes balanceadas				
Bagging	↑**88,5** ± 1,6	87,6 ± 1,5	↑**89,4** ± 2,2	↑**89,2** ± 2,1	↑**0,884** ± 0,015
Floresta aleatória	84,7 ± 1,3	↑**88,9** ± 1,4	80,5 ± 2,3	82,0 ± 1,7	0,853 ± 0,011
AdaBoost	85,2 ± 2,0	85,4 ± 2,0	84,9 ± 3,3	85,0 ± 2,9	0,852 ± 0,019
OneR	80,9 ± 2,7	80,4 ± 7,0	81,3 ± 8,9	82,1 ± 7,7	0,807 ± 0,025
MLP-LM	80,4 ± 1,4	78,6 ± 3,7	82,3 ± 3,7	82,2 ± 3,3	0,803 ± 0,012
IBK	79,9 ± 0,9	78,2 ± 2,2	81,6 ± 1,8	81,0 ± 1,3	0,795 ± 0,011
SVM	80,0 ± 1,1	75,7 ± 4,2	84,4 ± 4,2	83,1 ± 3,3	0,791 ± 0,015
C4.5	78,8 ± 1,4	78,5 ± 2,6	79,1 ± 2,6	79,0 ± 1,9	0,787 ± 0,015
LogitBoost	70,8 ± 1,6	71,5 ± 4,0	70,0 ± 3,8	70,5 ± 2,1	0,709 ± 0,020
MLP-GD	66,7 ± 2,2	66,3 ± 3,5	↓**67,2** ± 3,2	↓**67,5** ± 3,1	0,668 ± 0,024
RBF	70,1 ± 1,7	52,3 ± 3,3	88,0 ± 2,3	81,4 ± 2,8	0,636 ± 0,027
SOM + LVQ	↓**53,3** ± 2,2	82,1 ± 3,4	67,7 ± 1,6	75,0 ± 3,4	0,622 ± 0,017
Naive Bayes	54,1 ± 3,4	***↓20,3** ± 24,3	87,8 ± 17,9	67,8 ± 9,3	***↓0,250** ± 0,201

Tabela 5.14: Resultados obtidos na detecção de *Web spam* usando atributos baseados nos *links* transformados.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes balanceadas				
Bagging	↑**87,0** ± 1,2	89,2 ± 1,6	84,7 ± 2,3	↑**85,4** ± 1,8	↑**0,872** ± 0,012
Floresta aleatória	84,5 ± 1,2	↑**90,6** ± 1,5	78,5 ± 2,4	80,8 ± 1,7	0,854 ± 0,010
IBK	84,9 ± 1,1	88,2 ± 1,2	81,6 ± 1,9	82,7 ± 1,5	0,853 ± 0,010
SVM	85,1 ± 1,2	85,2 ± 2,1	85,0 ± 1,8	85,1 ± 1,5	0,851 ± 0,012
AdaBoost	83,8 ± 1,5	85,0 ± 2,2	82,6 ± 2,2	83,0 ± 1,9	0,840 ± 0,015
OneR	81,3 ± 1,7	84,4 ± 2,9	78,2 ± 2,3	79,4 ± 1,8	0,818 ± 0,018
MLP-LM	81,4 ± 3,3	81,2 ± 5,9	81,9 ± 2,7	81,8 ± 3,6	0,814 ± 0,038
C4.5	78,5 ± 1,5	79,9 ± 1,5	77,2 ± 3,0	77,8 ± 2,2	0,788 ± 0,013
MLP-GD	74,1 ± 1,3	74,8 ± 1,6	73,4 ± 2,8	74,4 ± 2,6	0,746 ± 0,015
LogitBoost	71,6 ± 2,0	68,9 ± 3,3	74,4 ± 3,4	72,9 ± 2,6	0,708 ± 0,022
SOM + LVQ	71,2 ± 2,9	69,3 ± 2,8	70,3 ± 1,7	69,9 ± 1,9	0,705 ± 0,018
RBF	64,6 ± 1,8	59,6 ± 4,2	↓**69,6** ± 2,4	↓**66,2** ± 1,7	0,627 ± 0,027
Naive Bayes	↓**58,6** ± 1,8	↓**28,2** ± 5,9	↑**89,0** ± 3,2	72,0 ± 2,8	↓**0,402** ± 0,058

UK2006, o melhor método foi o *bagging*. Note que, em todos os cenários, ele obteve as melhores taxas de acurácia, precisão e F-medida. Como as classes dessas simulações foram balanceadas, a acurácia é uma medida válida e importante para medir o desempenho de um classificador.

Os quatro piores classificadores, considerando todas as simulações, foram os métodos *naive* Bayes, SOM + LVQ, RBF e *LogitBoost*. Em todos os cenários esses classificadores tiveram desempenho inferior ao *OneR*, que é um dos métodos mais simples da literatura. Entre eles, o *naive* Bayes foi o método que teve maior queda de desempenho com a nova base de dados.

Outros métodos que tiveram desempenho inferior com a nova base de dados, em comparação ao que foi observado nas simulações com a base de dados WEBSPAM-UK2006, foram as redes neurais MLP. Na primeira base de dados, os dois métodos estavam entre os cinco melhores algoritmos de aprendizado em todos os cenários com classes balanceadas e sem combinação de atributos. Por outro lado, com a base de dados WEBSPAM-UK2007, a MLP-LM está entre os cinco melhores métodos apenas nas simulações com atributos baseados no conteúdo (Tabela 5.12) e nos *links* (Tabela 5.13). A MLP-GD não está entre os melhores métodos em nenhum cenário e está entre os quatro piores métodos nas simulações com atributos baseados nos *links* (Tabela 5.13) e nos *links* transformados (Tabela 5.14).

5.4.1 Análise do impacto da redução de dimensionalidade nos vetores de atributos extraídos da base de dados WEBSPAM-UK2007

Assim como foi feito com a base de dados WEBSPAM-UK2006, nessa seção é apresentada uma análise do impacto que a redução de dimensionalidade, por meio da técnica de análise de componentes principais (PCA), pode provocar no desempenho dos métodos na classificação das amostras extraídas da base de dados WEBSPAM-UK2007.

O método PCA foi aplicado usando uma variância que diversificou entre 95% e 99%, conforme pode ser observado na Tabela 5.15. Nessa Tabela é apresentada o número de dimensões obtidas após a aplicação da técnica de PCA em relação a cada porcentagem de variância utilizada. A última coluna, que está rotulada como "100%", apresenta o número de dimensões original dos vetores de atributos.

Tabela 5.15: Número de dimensões obtidas após redução da dimensionalidade dos vetores de atributos.

	Variância					
	95%	96%	97%	98%	99%	100%
Tipos de atributos	Nº de dimensões					
Conteúdo	37	40	45	51	61	96
Links	20	22	24	27	31	41
Links transformados	32	35	40	46	58	138

As Figuras 5.8, 5.9 e 5.10 mostram a variação dos resultados em relação ao número de dimensões reduzidas. A medida de desempenho utilizada foi a F-medida. Em cada figura, "100%" de variância mostra os resultados obtidos sem aplicação da técnica de PCA.

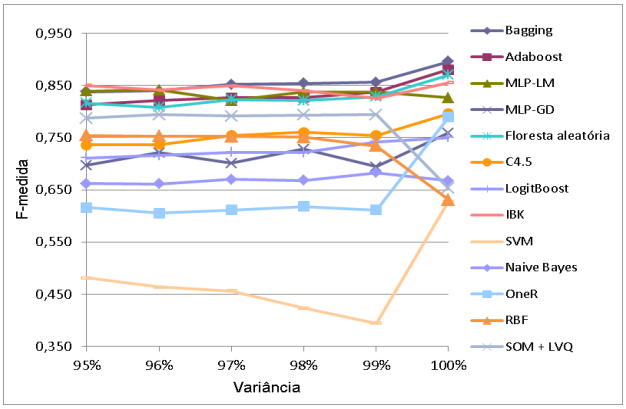


Figura 5.8: Resultados obtidos na detecção de *Web spam* usando atributos baseados no conteúdo e com dimensionalidade reduzida.

O impacto da redução de dimensionalidade dos vetores de atributos extraídos da base de dados WEBSPAM-UK2007 é semelhante ao impacto observado nos resultados obtidos com a base de dados WEBSPAM-UK2006. Para a maioria dos métodos de classificação, principalmente para os melhores, aplicar a técnica de PCA foi prejudicial. Os métodos prejudicados em todos os cenários foram: *bagging*, *AdaBoost*, MLP-GD, floresta aleatória, C4.5, *LogitBoost* e *OneR*.

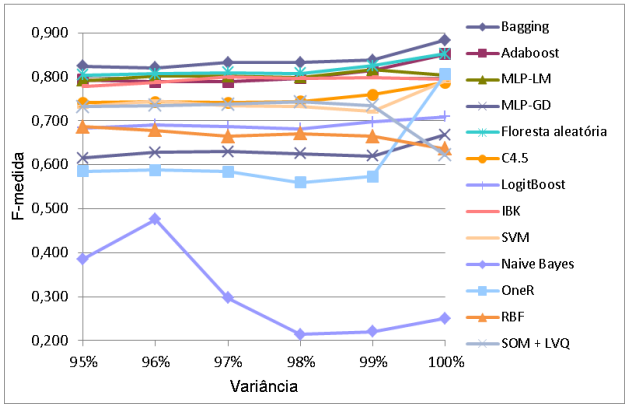


Figura 5.9: Resultados obtidos na detecção de *Web spam* usando atributos baseados nos *links* e com dimensionalidade reduzida.

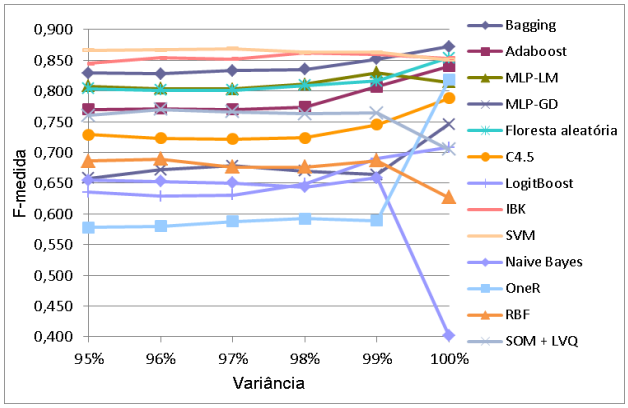


Figura 5.10: Resultados obtidos na detecção de *Web spam* usando atributos baseados nos *links* transformados e com dimensionalidade reduzida.

Porém alguns métodos foram beneficiados com a aplicação da técnica de PCA em todos os cenários, tais como os métodos *naive* Bayes, MLP-LM, RBF e SOM + LVQ. Entre esses métodos, o mais beneficiado com a redução da dimensionalidade dos vetores de atributos é o método SOM + LVQ, pois ele saiu do grupo dos quatro piores métodos em todos os cenários.

Outros métodos de aprendizado foram beneficiados com a redução de dimensionalidade de alguns tipos de atributos e prejudicados quando o mesmo processo foi realizado em outros, tais como os métodos SVM e IBK. Entre esses métodos, se destaca o SVM, que na classificação dos vetores de atributos baseados nos *links* transformados (Figura 5.10), após a aplicação da técnica de PCA com 97% de variância, passou a ter o segundo melhor desempenho entre todos os algoritmos de classificação.

5.4.2 Combinação de vetores de atributos extraídos da base de dados WEBSPAM-UK2007

Nessa seção são apresentadas as simulações realizadas com todas as combinações entre os vetores de atributos. Como foi explicado anteriormente, a combinação de atributos pode ser

útil para a detecção de *spam hosts* híbridos, que possuem páginas que usam conteúdo *spam* e *spam links*. Assim como na Seção 5.4, foram feitas simulações apenas com classes balanceadas. O balanceamento foi feito por meio das técnicas SMOTE e subamostragem aleatória.

As Tabelas de 5.16 a 5.19 apresentam, respectivamente, os resultados de cada método na classificação das seguintes combinações de atributos: conteúdo com *links*, *links* com *links* transformados, *links* transformados com conteúdo e *links* com *links* transformados com conteúdo. Em cada coluna são apresentadas a média e o desvio padrão dos resultados obtidos pelos classificadores, usando a técnica de validação por subamostragem aleatória com 10 repetições. Os valores em negrito precedidos pelo símbolo "↑" indicam os melhores resultados, enquanto os valores em negrito precedidos pelo símbolo "↓" indicam os piores resultados. Os valores em negrito precedidos pelo símbolo "*" indicam os melhores ou piores resultados considerando todas as combinações de atributos.

Tabela 5.16: Resultados obtidos na detecção de *Web spam* usando combinação dos atributos baseados nos *links* e no conteúdo.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes balanceadas				
AdaBoost	↑**93,5** ± 0,5	↑**94,5** ± 0,7	↑**92,6** ± 1,2	↑**92,7** ± 1,1	↑**0,936** ± 0,005
Bagging	89,5 ± 1,3	91,4 ± 1,1	87,7 ± 2,4	88,1 ± 2,0	0,897 ± 0,012
MLP-LM	87,2 ± 2,4	87,2 ± 3,2	87,3 ± 3,3	86,3 ± 4,4	0,866 ± 0,029
Floresta aleatória	85,8 ± 1,0	90,3 ± 1,8	81,3 ± 1,5	82,8 ± 1,1	0,864 ± 0,010
IBK	85,6 ± 1,5	85,9 ± 2,1	85,4 ± 1,4	85,4 ± 1,4	0,857 ± 0,016
OneR	84,3 ± 1,9	79,0 ± 4,0	89,5 ± 5,7	88,7 ± 5,5	0,834 ± 0,017
C4.5	79,4 ± 1,7	79,0 ± 1,6	79,8 ± 3,2	79,7 ± 2,5	0,793 ± 0,015
MLP-GD	78,4 ± 1,8	78,5 ± 3,5	78,4 ± 2,5	78,1 ± 2,6	0,783 ± 0,020
SVM	78,2 ± 1,6	72,6 ± 2,6	83,6 ± 1,6	81,6 ± 1,6	0,769 ± 0,019
LogitBoost	76,3 ± 1,1	75,5 ± 3,4	77,1 ± 3,3	76,8 ± 2,0	0,761 ± 0,014
Naive Bayes	↓**59,5** ± 4,8	81,6 ± 19,0	***↓37,4** ± 25,7	***↓58,8** ± 8,7	0,660 ± 0,057
SOM + LVQ	61,2 ± 3,6	73,9 ± 4,3	67,6 ± 1,5	70,3 ± 2,7	0,653 ± 0,018
RBF	65,8 ± 1,3	↓**58,8** ± 1,6	72,9 ± 3,0	68,5 ± 2,1	↓**0,632** ± 0,011

Tabela 5.17: Resultados obtidos na detecção de *Web spam* usando combinação dos atributos baseados nos *links* e nos *links* transformados.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes balanceadas				
Bagging	↑**88,7** ± 1,5	86,5 ± 2,2	↑**90,9** ± 2,0	↑**90,5** ± 1,9	↑**0,885** ± 0,016
Floresta aleatória	86,4 ± 1,0	↑**89,5** ± 1,3	83,3 ± 2,1	84,3 ± 1,6	0,868 ± 0,009
IBK	86,1 ± 0,8	87,2 ± 0,6	85,1 ± 1,5	85,4 ± 1,2	0,863 ± 0,007
AdaBoost	85,5 ± 0,7	84,9 ± 1,5	86,2 ± 1,0	86,0 ± 0,8	0,854 ± 0,008
MLP-LM	83,4 ± 2,8	83,2 ± 4,4	83,8 ± 4,0	84,2 ± 3,4	0,836 ± 0,028
C4.5	81,3 ± 1,2	80,3 ± 2,6	82,3 ± 1,3	81,9 ± 1,0	0,811 ± 0,014
OneR	81,1 ± 0,8	78,5 ± 6,2	83,6 ± 6,5	83,3 ± 5,2	0,805 ± 0,014
MLP-GD	79,8 ± 2,2	78,5 ± 3,1	81,2 ± 2,2	80,9 ± 2,5	0,797 ± 0,025
SVM	79,8 ± 1,5	73,0 ± 4,5	86,6 ± 2,8	84,6 ± 2,1	0,783 ± 0,021
LogitBoost	76,7 ± 1,7	78,7 ± 3,6	74,6 ± 2,9	75,6 ± 1,9	0,771 ± 0,020
SOM + LVQ	62,7 ± 3,2	76,2 ± 2,5	↓**69,5** ± 1,8	72,5 ± 2,2	0,672 ± 0,022
RBF	67,8 ± 2,7	57,6 ± 1,9	78,0 ± 4,1	72,5 ± 4,1	0,641 ± 0,025
Naive Bayes	***↓56,3** ± 1,5	***↓23,8** ± 3,2	88,6 ± 2,4	↓**67,8** ± 4,5	***↓0,352** ± 0,036

Pelos resultados apresentados nas Tabelas de 5.16 a 5.19 é possível notar que o balanceamento entre as classes usando as técnicas SMOTE e subamostragem aleatória foi bem sucedido. O principal problema que poderia ter ocorrido pelo uso da técnica SMOTE seria o *overfitting*. Porém, nesse caso, a taxa de sensitividade, que é a taxa de acertos em relação a classe *spam*, seria bem superior à taxa de especificidade, que é a taxa de acertos em relação a classe *ham*.

Tabela 5.18: Resultados obtidos na detecção de *Web spam* usando combinação dos atributos baseados nos *links* transformados e no conteúdo.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes desbalanceadas				
AdaBoost	***↑94,4** ± 0,7	***↑94,9** ± 0,7	**↑94,0** ± 1,3	**↑94,0** ± 1,2	***↑0,944** ± 0,007
Bagging	92,2 ± 0,9	92,8 ± 1,4	91,7 ± 0,7	91,8 ± 0,7	0,923 ± 0,009
Floresta aleatória	88,0 ± 1,1	91,9 ± 0,9	84,0 ± 1,7	85,1 ± 1,4	0,884 ± 0,010
IBK	87,9 ± 1,5	88,9 ± 2,0	86,9 ± 1,9	87,1 ± 1,6	0,880 ± 0,015
MLP-LM	85,7 ± 2,4	85,3 ± 3,0	86,1 ± 2,8	85,8 ± 2,8	0,855 ± 0,025
MLP-GD	82,7 ± 2,8	84,0 ± 3,2	81,4 ± 3,5	81,6 ± 4,3	0,827 ± 0,033
C4.5	82,3 ± 1,4	82,4 ± 2,8	82,2 ± 2,0	82,2 ± 1,5	0,823 ± 0,015
OneR	82,9 ± 2,4	76,1 ± 2,7	89,7 ± 6,1	88,6 ± 6,1	0,817 ± 0,020
LogitBoost	79,1 ± 1,6	79,9 ± 2,8	78,3 ± 1,4	78,6 ± 1,3	0,792 ± 0,018
SVM	72,5 ± 1,1	60,7 ± 3,0	84,3 ± 2,3	79,5 ± 2,0	0,688 ± 0,018
Naive Bayes	69,3 ± 3,8	63,8 ± 12,1	74,8 ± 10,1	**↓72,4** ± 4,6	0,670 ± 0,066
SOM + LVQ	**↓58,0** ± 6,2	81,0 ± 3,8	**↓69,5** ± 1,9	75,5 ± 2,5	0,654 ± 0,036
RBF	67,9 ± 1,8	**↓53,3** ± 2,6	82,6 ± 1,8	75,4 ± 2,4	**↓0,624** ± 0,024

Tabela 5.19: Resultados obtidos na detecção de *Web spam* usando combinação de todos os atributos (*links*, *links* transformados e conteúdo).

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes balanceadas				
AdaBoost	**↑94,1** ± 0,8	**↑94,0** ± 1,0	***↑94,2** ± 1,2	***↑94,2** ± 1,2	**↑0,941** ± 0,008
Bagging	90,9 ± 1,2	91,0 ± 1,7	90,8 ± 1,6	90,8 ± 1,5	0,909 ± 0,012
IBK	88,9 ± 1,2	90,6 ± 1,9	87,2 ± 2,1	87,6 ± 1,7	0,890 ± 0,012
Floresta aleatória	87,8 ± 1,0	91,2 ± 1,4	84,4 ± 1,7	85,3 ± 1,3	0,881 ± 0,010
MLP-LM	87,9 ± 1,4	88,4 ± 2,8	87,3 ± 2,7	87,1 ± 2,4	0,877 ± 0,016
MLP-GD	83,7 ± 2,0	85,5 ± 2,3	81,9 ± 2,8	83,1 ± 3,1	0,842 ± 0,020
OneR	84,1 ± 1,9	77,0 ± 3,6	91,1 ± 4,6	90,0 ± 4,7	0,828 ± 0,019
C4.5	82,5 ± 1,3	82,6 ± 3,3	82,5 ± 1,4	82,5 ± 0,9	0,825 ± 0,016
LogitBoost	79,3 ± 1,4	79,2 ± 2,9	79,4 ± 1,4	79,3 ± 1,1	0,792 ± 0,017
SVM	78,6 ± 1,4	72,1 ± 2,4	85,1 ± 2,2	82,9 ± 2,0	0,771 ± 0,017
Naive Bayes	66,6 ± 5,6	69,8 ± 11,1	**↓63,4** ± 20,3	**↓67,8** ± 8,9	0,675 ± 0,032
RBF	68,0 ± 1,6	**↓60,7** ± 1,8	75,4 ± 2,1	71,2 ± 2,1	0,655 ± 0,017
SOM + LVQ	**↓58,4** ± 3,0	76,1 ± 3,5	67,2 ± 1,5	71,0 ± 2,7	**↓0,640** ± 0,018

Observe que os valores das duas métricas estão bem equilibrados e, em várias simulações, a taxa de especificidade foi superior à taxa de sensitividade. Além disso, com o problema de *overfitting*, os classificadores detectariam muitos falsos positivos e isso faria com que a taxa de precisão fosse muito baixa. No entanto, as taxas de precisão foram satisfatórias e, em geral, superiores ao que foi obtido nas simulações com a base de dados WEBSPAM-UK2006, em que não foi utilizada a técnica SMOTE para balancear as classes. Da mesma forma, o emprego da técnica de subamostragem aleatória também foi bem sucedido, uma vez que os desvios padrões dos resultados foram baixos.

Os resultados indicam que a combinação dos vetores de atributos melhora o desempenho dos algoritmos de classificação. A única combinação que não obteve melhores resultados, comparado aos experimentos em que os vetores de atributos não foram combinados, foi a combinação de vetores de atributos baseados nos *links* com vetores de atributos baseados nos *links* transformados (Tabela 5.17), que obteve um desempenho inferior ao que foi obtido nas simulações com atributos baseados no conteúdo (Tabela 5.12) e com atributos baseados nos *links* transformados (Tabela 5.14). Além disso, assim como ocorreu nas simulações com a base de dados WEBSPAM-UK2006, os classificadores tiveram melhor desempenho com a combinação dos atributos baseados nos *links* transformados e no conteúdo (Tabela 5.18). Os resultados apresentados na Tabela 5.19, referentes a combinação de todos os vetores de atributos, também

foram bons, porém a alta dimensionalidade do espaço de atributos (275 dimensões) exige um custo computacional para a tarefa de classificação superior ao exigido pelas outras combinações de atributos.

Em relação aos métodos de aprendizado, assim como ocorreu nos experimentos com combinações de atributos extraídos da base de dados WEBSPAM-UK2006, o método *AdaBoost* obteve os melhores resultados. No entanto, o método *bagging*, que foi o melhor método nas simulações sem combinação de vetores de atributos, obteve, em geral, o segundo melhor desempenho. O grupo dos quatro piores métodos novamente foi formados pelos métodos *naive* Bayes, RBF, SOM + LVQ e *LogitBoost*.

Análise do impacto da redução de dimensionalidade na combinação dos vetores de atributos extraídos da base de dados WEBSPAM-UK2007

Nessa seção, é apresentada uma análise dos efeitos da redução de dimensionalidade das combinações dos vetores de atributos. Assim como foi feito na Seção 5.4.1, a técnica de PCA foi aplicada com uma porcentagem de variância que diversificou entre 95% e 99%. A Tabela 5.20 apresenta a redução em número de dimensões em relação a cada porcentagem de variância utilizada. A última coluna que está rotulada como "100%" apresenta o número de dimensões original dos vetores formados pela combinação dos vetores de atributos.

Tabela 5.20: Número de dimensões obtidas após redução da dimensionalidade das combinações dos vetores de atributos.

	Variância					
	95%	96%	97%	98%	99%	100%
Tipos de atributos	Nº de dimensões					
Links + conteúdo	57	63	70	79	95	137
Links + *Links* transformados	48	54	62	74	94	179
Links transformados + conteúdo	71	79	90	105	132	234
Links + *Links* transformados + conteúdo	85	95	108	127	157	275

As Figuras de 5.11 a 5.14 mostram a variação do valor da F-medida em relação ao número de dimensões reduzidas. Em cada figura, "100%" de variância mostra os resultados obtidos pelos classificadores sem a aplicação do método de PCA nas combinações dos vetores de atributos.

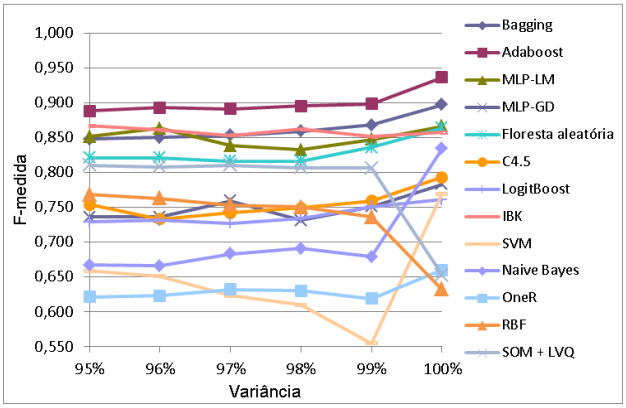


Figura 5.11: Resultados obtidos na detecção de *Web spam* usando combinaçao dos atributos baseados no conteúdo e nos *links* e com dimensionalidade reduzida.

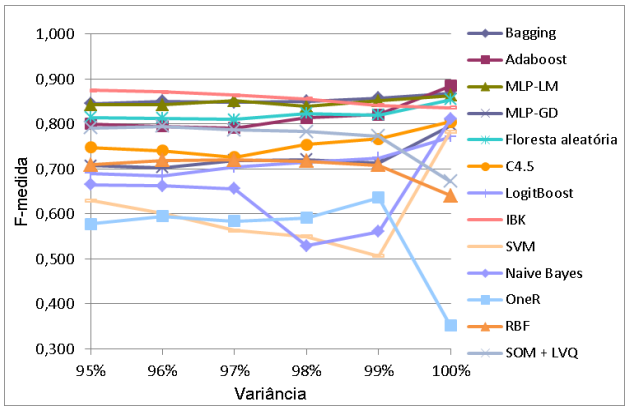


Figura 5.12: Resultados obtidos na detecção de *Web spam* usando combinaçao dos atributos baseados nos links e nos *links* transformados e com dimensionalidade reduzida.

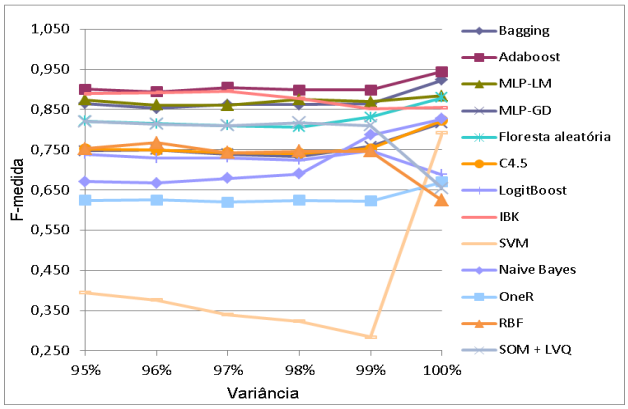


Figura 5.13: Resultados obtidos na detecção de *Web spam* usando combinaçao dos atributos baseados no conteúdo e nos *links* transformados e com dimensionalidade reduzida.

Os resultados obtidos com as combinações de atributos, após a aplicação de técnica de PCA, indicam que a maioria dos métodos de aprendizado tiveram seu desempenho prejudicado, assim como foi observado nas simulações sem combinação de atributos. Os únicos métodos que foram beneficiados com a redução de dimensionalidade foram as redes neurais RBF e SOM + LVQ.

Os métodos MLP-LM e *naive* Bayes, que melhoraram seu desempenho com a redução de dimensionalidade dos vetores de atributos, tiveram um comportamento diferente em relação a aplicação da técnica de PCA sobre as combinações dos vetores de atributos, pois em todas as simulações seu desempenho foi reduzido.

Outro método que melhorou seu desempenho após a aplicação da técnica de PCA foi o *LogitBoost* nas simulações em que foram combinados os vetores de atributos baseados no conteúdo e nos *links* transformados (Tabela 5.18) e nas simulações em que foram combinados todos os tipos de atributos (Tabela 5.19).

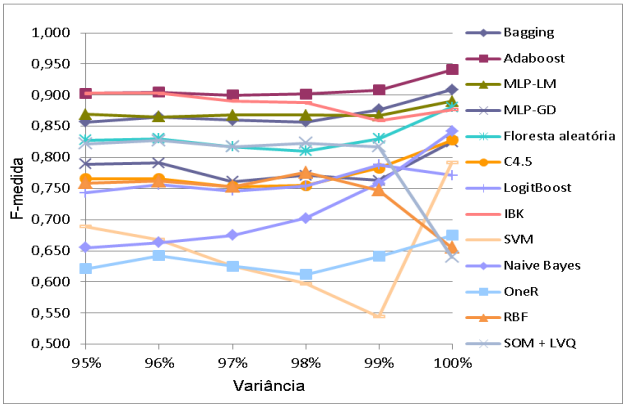


Figura 5.14: Resultados obtidos na detecção de *Web spam* usando combinaçao de todos os tipos de atributos (conteúdo, *links* e *links* transformados) e com dimensionalidade reduzida.

5.5 Combinação das predições obtidas com diferentes tipos de atributos

Diante do bom desempenho obtido pelos métodos de classificação usando combinações de atributos, resolveu-se testar uma nova metodologia de classificação de *Web spam* em que é feita a combinação das predições obtidas pelos métodos de aprendizado com cada tipo de vetor de atributo. O processo de classificação usado nessa nova estratégia é apresentado na Figura 5.15.

Conforme pode ser observado na Figura 5.15, cada *host* do conjunto de treinamento é representado por três vetores de atributos: vetor de atributos baseados no conteúdo, vetor de atributos baseados nos *links* e vetor de atributos baseados nos *links* transformados. Depois que os vetores são extraídos de todos os *hosts*, eles formam 3 conjuntos de treinamento. Cada conjunto é usado no treinamento de uma instância do algoritmo de aprendizado de máquina. Então supondo que o classificador seja o método SVM, ele será treinado para classificar os *hosts* com base nos atributos extraídos do conteúdo, formando o primeiro modelo de predição. Ele também será treinado para classificar os *hosts* com base nos atributos extraídos dos *links*, gerando um segundo modelo de predição. Por fim, um terceiro modelo de predição será criado pelo SVM por meio do treinamento com atributos extraídos dos *links* transformados.

Depois que é formado um modelo de predição para cada um dos tipos de atributos, o classificador está preparado para tentar identificar a classe de um *host* desconhecido. Dessa forma, quando um novo *host* é apresentado, são extraídos três vetores de atributos, conforme ocorreu na fase de treinamento. Cada um desses vetores é apresentado para o modelo de predição correspondente, ou seja, se o vetor é formado por atributos baseados no conteúdo, ele será classificado pelo modelo treinado com o mesmo tipo de atributos. Cada modelo irá gerar uma predição, que pode ser 1 (*spam*) ou 0 (*ham*). Depois disso, o *host* recebe o rótulo por meio da combinação das predições de cada um dos modelos, usando voto majoritário simples. Logo, se 2 modelos classificarem o *host* como *spam*, a predição final para esse *host* será 1, indicando que ele é *spam*, senão a predição final será 0, indicando que ele é *ham*.

A nova abordagem de classificação de *Web spam* foi testada com todos os métodos de aprendizado de máquina usados nas simulações apresentadas nas seções anteriores. A Tabela 5.21

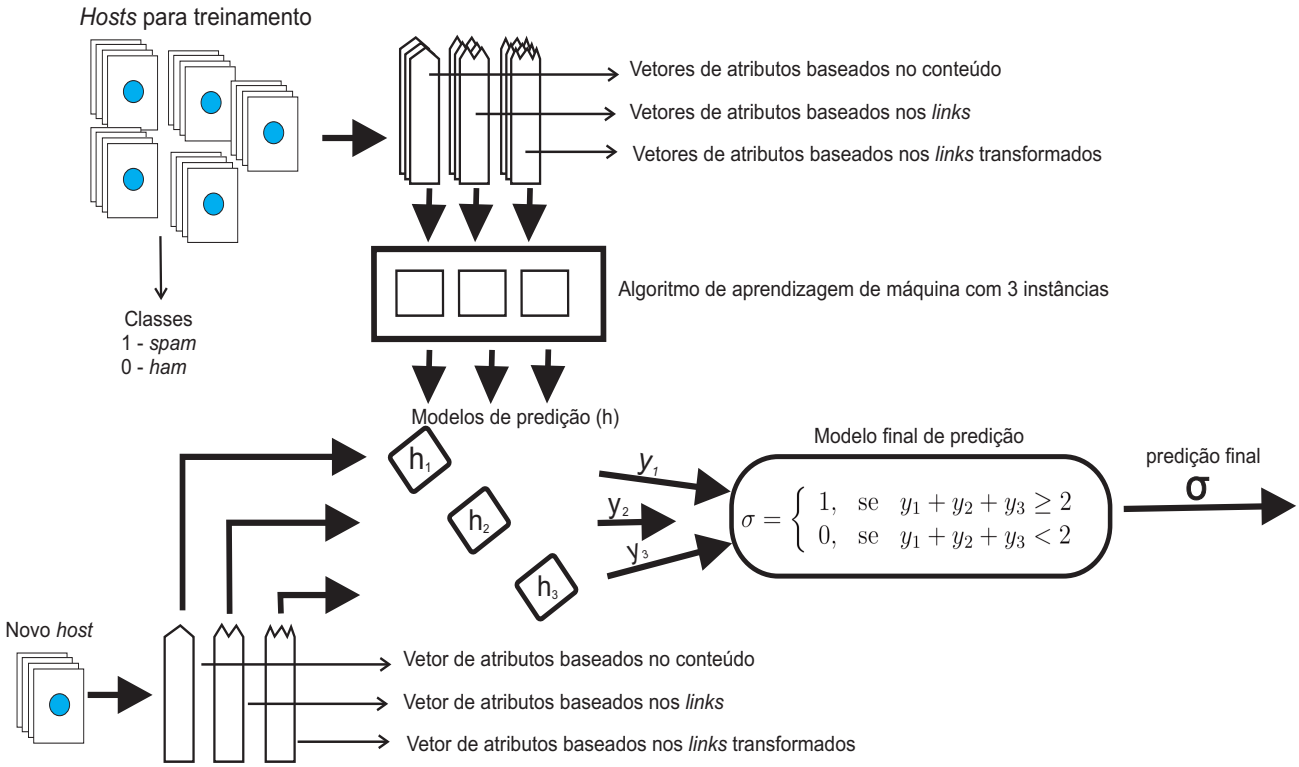


Figura 5.15: Processo de classificação por combinação das predições obtidas com diferentes tipos de atributos.

apresenta os resultados obtidos com os atributos extraídos da base de dados WEBSPAM-UK2006, usando as mesmas configurações apresentadas na Seção 5.3. Com essa base de dados foram feitos experimentos usando classes balanceadas e classes desbalanceadas. Por outro lado, a Tabela 5.22 apresenta os resultados obtidos com as amostras extraídas da base de dados WEBSPAM-UK2007. Com essa base de dados foram feitos experimentos apenas com classes balanceadas pelos mesmos motivos apresentados na Seção 5.4 e usando as mesmas estratégias de balanceamento apresentadas nessa seção. Os valores em negrito precedidos pelo símbolo "↑" indicam os melhores resultados, enquanto os valores em negrito precedidos pelo símbolo "↓" indicam os piores resultados.

Pelos resultados apresentados nas Tabelas 5.21 e 5.22 é possível notar que a abordagem de classificação mostrada na Figura 5.15 obteve resultados inferiores aos obtidos com a combinação de atributos, com exceção da combinação dos atributos baseados nos *links* com o atributos baseados nos *links* transformados. No entanto, essa abordagem produz melhores resultados do que a classificação sem combinação dos vetores de atributos e por isso, é uma boa estratégia de classificação de *Web spam*. Além disso, a nova abordagem que está sendo proposta tem uma vantagem sobre algumas combinações de atributos: a classificação dos vetores de atributos pode ser feita de forma paralela, já que usa três instâncias diferentes do algoritmo de aprendizado. Logo, o tempo computacional do processo de classificação usando essa abordagem, em que a maior dimensão é 138 (*links* transformados), é menor do que no processo de classificação que usa a combinação de todos atributos (275 dimensões) e a combinação dos atributos baseados nos *links* transformados com os atributos baseados no conteúdo (234 dimensões).

Tabela 5.21: Resultados obtidos pela combinação das predições obtidas com os diferentes tipos de atributos extraídos da base de dados WEBSPAM-UK2006.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes desbalanceadas				
Bagging	↑**91,2** ± 0,5	78,2 ± 1,4	95,1 ± 0,6	82,8 ± 2,1	↑**0,804** ± 0,014
Floresta aleatória	90,9 ± 0,7	79,8 ± 2,3	94,2 ± 0,5	80,6 ± 2,0	0,802 ± 0,017
AdaBoost	90,5 ± 0,5	74,3 ± 1,6	95,4 ± 0,5	82,9 ± 1,5	0,784 ± 0,008
MLP-LM	90,3 ± 1,1	73,2 ± 6,8	95,5 ± 1,2	↑**83,2** ± 3,4	0,776 ± 0,037
C4.5	89,1 ± 0,7	72,1 ± 2,6	94,2 ± 0,7	79,0 ± 2,3	0,753 ± 0,018
IBK	88,3 ± 0,9	68,3 ± 2,6	94,4 ± 0,7	78,6 ± 2,3	0,730 ± 0,019
MLP-GD	87,9 ± 1,6	62,8 ± 4,2	95,6 ± 0,9	81,4 ± 4,0	0,709 ± 0,038
LogitBoost	86,6 ± 7,1	68,1 ± 16,6	92,2 ± 4,4	72,4 ± 16,8	0,701 ± 0,164
SVM	87,3 ± 0,7	58,1 ± 2,2	96,2 ± 0,4	82,1 ± 1,7	0,680 ± 0,019
OneR	83,7 ± 0,9	52,3 ± 2,9	93,2 ± 1,0	69,8 ± 3,2	0,597 ± 0,021
SOM + LVQ	83,6 ± 0,4	37,6 ± 2,5	97,5 ± 0,7	82,6 ± 3,6	0,516 ± 0,021
Naive Bayes	↓**37,9** ± 1,4	↑**99,5** ± 0,4	↓**19,3** ± 1,5	↓**27,1** ± 1,2	0,426 ± 0,014
RBF	81,6 ± 0,5	↓**27,6** ± 2,7	↑**98,0** ± 0,5	80,5 ± 2,7	↓**0,410** ± 0,030
	Classes Balanceadas				
MLP-GD	90,0 ± 1,8	94,2 ± 1,7	85,7 ± 3,1	87,2 ± 3,0	↑**0,905** ± 0,018
Bagging	↑**90,1** ± 1,3	93,3 ± 1,5	87,2 ± 1,9	↑**87,3** ± 2,0	0,901 ± 0,013
Floresta aleatória	89,5 ± 1,2	94,0 ± 1,3	85,4 ± 1,7	85,8 ± 1,8	0,897 ± 0,013
AdaBoost	89,4 ± 1,4	91,8 ± 1,8	87,2 ± 1,6	87,1 ± 1,9	0,894 ± 0,016
MLP-LM	88,8 ± 2,0	92,6 ± 2,3	85,1 ± 3,9	85,7 ± 2,7	0,890 ± 0,017
C4.5	87,4 ± 1,3	89,2 ± 1,9	85,7 ± 1,6	85,5 ± 1,6	0,873 ± 0,013
LogitBoost	86,9 ± 1,0	90,2 ± 1,2	83,7 ± 1,3	84,0 ± 1,5	0,869 ± 0,011
IBK	86,3 ± 1,2	86,8 ± 1,7	85,8 ± 2,1	85,3 ± 2,1	0,860 ± 0,012
SVM	85,4 ± 1,3	84,1 ± 2,8	86,6 ± 1,4	86,3 ± 1,2	0,852 ± 0,015
OneR	84,1 ± 1,3	89,5 ± 1,4	79,0 ± 2,2	80,1 ± 2,0	0,845 ± 0,014
SOM + LVQ	81,1 ± 2,1	80,6 ± 2,6	81,6 ± 3,1	81,5 ± 2,7	0,810 ± 0,021
Naive Bayes	↓**67,9** ± 8,6	↑**99,0** ± 1,2	↓**38,6** ± 16,7	↓**61,0** ± 7,3	0,752 ± 0,053
RBF	76,7 ± 1,9	↓**63,5** ± 1,4	↑**89,9** ± 2,8	86,3 ± 3,4	↓**0,731** ± 0,019

Tabela 5.22: Resultados obtidos pela combinação das predições obtidas com os diferentes tipos de atributos extraídos da base de dados WEBSPAM-UK2007.

	Acurácia	Sensitividade	Especificidade	Precisão	F-medida
	Classes desbalanceadas				
Bagging	↑**91,1** ± 0,5	90,8 ± 0,8	91,5 ± 1,1	91,4 ± 1,1	↑**0,911** ± 0,004
AdaBoost	90,3 ± 0,8	89,8 ± 1,3	90,7 ± 0,7	90,5 ± 0,9	0,902 ± 0,010
IBK	89,1 ± 0,7	87,0 ± 1,6	91,2 ± 1,9	90,8 ± 1,9	0,888 ± 0,008
Floresta aleatória	88,3 ± 2,6	↑**92,3** ± 1,7	84,3 ± 3,6	85,6 ± 3,0	0,888 ± 0,024
MLP-LM	87,0 ± 3,6	88,7 ± 4,5	85,2 ± 8,7	86,6 ± 5,9	0,875 ± 0,031
C4.5	86,0 ± 1,3	86,2 ± 1,9	85,9 ± 2,1	85,9 ± 2,2	0,860 ± 0,015
OneR	85,6 ± 2,5	79,6 ± 5,6	91,8 ± 8,8	↑**91,6** ± 8,1	0,847 ± 0,021
SVM	84,8 ± 1,2	77,7 ± 2,0	91,9 ± 1,0	90,6 ± 1,1	0,836 ± 0,014
MLP-GD	80,3 ± 1,9	82,5 ± 3,6	78,1 ± 6,4	79,5 ± 4,0	0,808 ± 0,015
LogitBoost	75,4 ± 7,5	76,0 ± 7,5	↓**74,8** ± 8,1	74,8 ± 8,2	0,754 ± 0,076
SOM + LVQ	78,7 ± 1,9	65,1 ± 4,3	↑**92,3** ± 2,3	89,6 ± 2,4	0,753 ± 0,028
RBF	76,7 ± 1,9	63,5 ± 1,4	89,9 ± 2,8	86,3 ± 3,4	0,731 ± 0,019
Naive Bayes	↓**62,1** ± 2,7	↓**40,6** ± 10,6	83,4 ± 6,8	↓**71,8** ± 5,6	↓**0,509** ± 0,082

Em relação aos métodos de aprendizado, como ocorreu nos cenários anteriores, os métodos *bagging*, *AdaBoost*, redes neurais MLP e floresta aleatória se destacaram com melhores resultados que os outros métodos de aprendizado. Por outro lado, os métodos RBF, SOM+LVQ e *naive* Bayes tiveram os piores desempenhos. Além disso, os resultados obtidos com classes balanceadas foram superiores aos resultados obtidos nas simulações com desbalanceamento entre as classes, conforme pode ser observado na Tabela 5.21.

5.6 Análise estatística dos resultados

Para tornar a análise do desempenho dos métodos na classificação de *Web spam* mais confiável, foi feita uma análise estatística dos resultados (Tabela 5.23). Os métodos foram ordenados pelo valor da F-medida usando o teste da soma dos postos de Wilcoxon (*Wilcoxon rank-sum test*) (Montgomery & Runger 2002) com um intervalo de confiança de 95%. Este é um teste de hipóteses não-paramétrico também chamado de teste de Mann-Whitney (*Mann-Whitney test*) (Montgomery & Runger 2002). Não foi utilizado o teste t de *Student* para a análise dos resultados porque eles não seguem uma distribuição normal. Para a análise estatística, os resultados obtidos após redução de dimensionalidade não foram considerados. Os métodos que estão no mesmo nível na Tabela 5.23 tem resultados estatisticamente equivalentes.

Tabela 5.23: Análise estatística dos resultados usando o teste da soma dos postos de Wilcoxon.

Nível	Métodos		
	Resultados obtidos com a base de dados WEBSPAM-UK2006		Resultados obtidos com a base de dados WEBSPAM-UK2007
	Classes desbalanceadas	Classes balanceadas	Classes balanceadas
1	*Bagging* e floresta aleatória	*Bagging*, *AdaBoost*, floresta aleatória, MLP-GD e MLP-LM	*Bagging* e *AdaBoost*
2	Floresta aleatória, *AdaBoost* e MLP-LM	C4.5, e *LogitBoost*	Floresta aleatória e IBK
3	MLP-GD	IBK	MLP-LM
4	C4.5 e *LogitBoost*	*OneR*	*OneR*
5	IBK	SVM e *naive* Bayes	C4.5
6	SVM e RBF	SOM+LVQ	MLP-GD e SVM
7	*OneR*	RBF	*LogitBoost*
8	*Naive* Bayes		SOM+LVQ
9	SOM + LVQ		RBF
10			*Naive* Bayes

Diante do que é apresentado pela Tabela 5.23, pode-se notar que, para a base de dados WEBSPAM-UK2006, o método *bagging* de árvores de decisão é estatisticamente equivalente ao método floresta aleatória e superior aos outros métodos avaliados nos experimentos com classes desbalanceadas, mas é estatisticamente equivalente ao método *AdaBoost* e às redes neurais MLP nos experimentos com classes balanceadas. Por outro lado, os métodos *OneR*, SVM, *naive* Bayes, SOM + LVQ e RBF são estatisticamente inferiores aos outros métodos analisados nesse trabalho.

Em relação aos resultados da base de dados WEBSPAM-UK2007, a análise estatística apresentada na Tabela 5.23 mostra que os métodos *bagging* e *AdaBoost* têm melhor desempenho que os outros métodos de aprendizado. Em contrapartida, os métodos *LogitBoost*, SOM + LVQ, RBF e *naive* Bayes são estatisticamente inferiores.

5.7 Comparação entre os resultados obtidos nesse trabalho e os resultados disponíveis na literatura

Para facilitar a avaliação dos métodos analisados nessa dissertação, é apresentado na Tabela 5.24 uma comparação entre os melhores resultados obtidos nesse trabalho e os resultados disponíveis na literatura de *Web spamming*. São usados para comparação apenas trabalhos que

usaram as bases de dados WEBSPAM-UK2006 e WEBSPAM-UK2007 nos experimentos. Antes de comparar os resultados, é apresentada uma breve descrição de cada trabalho usado para comparação:

- (Castillo et al. 2007) propõem um conjunto de atributos baseados no conteúdo das páginas *Web* para separar os *spam hosts* dos *ham hosts*, seguindo a abordagem proposta por (Ntoulas et al. 2006). Eles também propõem um conjunto de atributos baseados nos *links* das páginas *Web*, derivados dos atributos propostos por (Becchetti et al. 2006a). Esses últimos atributos foram disponibilizados na competição *Web Spam Challenge* com o nome de atributos baseados nos *links* transformados. Nos experimentos, os autores usaram a base de dados WEBSPAM-UK2006 e usaram para a classificação de *Web spam* o método de árvores de decisão sensíveis ao custo e o método *bagging* para agregar múltiplas versões de árvores de decisão sensíveis ao custo. Os resultados apresentados por (Castillo et al. 2007) foram calculados usando validação cruzada *k-folds* com $k = 10$. O melhor resultado foi obtido pelo método *bagging* no experimento em que foi feita a combinação dos atributos baseados nos *links* transformados com os atributos baseados no conteúdo, conforme pode ser visto na Tabela 5.24.

  No mesmo trabalho os autores usaram um algoritmo de agrupamento para grafos (*graph clustering algorithm*) para criar grupos de *hosts*. Depois disso, eles classificaram os *hosts* e atribuíram a mesma predição para todos os *hosts* do mesmo grupo (*cluster*) por voto majoritário. Nessa abordagem, o melhor resultado foi obtido quando a classificação foi realizada pelo método *bagging*.

  Outra abordagem usada pelos autores faz a propagação das predições de um determinado *host* para seus *hosts* vizinhos. Nessa abordagem, eles fizeram experimentos em que a propagação foi feita apenas para vizinhos de entrada, apenas para vizinhos de saída e para ambos. O melhor resultado foi obtido no experimento com o método *bagging* e que considerou apenas os vizinhos de entrada.

  (Castillo et al. 2007) também propuseram uma abordagem em que as predições de *hosts* vizinhos são usadas como novos atributos e o classificador é retreinado. Essa abordagem é conhecida como *stacked graphical learning* (SGL) (Cohen & Kou 2006). Eles fizeram experimentos com 1 e 2 iterações da abordagem proposta. O melhor resultado foi obtido usando 2 iterações.

- (Bíró, Szabó & Benczúr 2008) aplicaram uma modificação do método *latent Dirichlet allocation* (LDA)(Blei, Ng & Jordan 2003), que é um modelo probabilístico gerador de coleções que considera que os documentos de texto são representados como uma mistura aleatória de tópicos, onde cada tópico é caracterizado como uma distribuição de termos ou palavras representativas. A modificação proposta pelos autores é chamada de *multicorpus* LDA. Nessa nova abordagem, eles aplicaram o método LDA na coleção de amostras de páginas *spam* e na coleção de amostras de páginas *ham* separadamente na fase de treinamento. Todavia, na fase de teste, eles uniram a coleção de tópicos *spam* e *ham* e consideraram que um site é *spam* se a probabilidade de tópicos *spam* for maior que um determinado

limiar. Os autores fizeram experimentos usando as bases de dados WEBSPAM-UK2006 e WEBSPAM-UK2007. Para a coleção de amostras *spam* eles consideraram 2, 5, 10 e 20 tópicos, enquanto que para a coleção de amostras *ham* eles consideraram 10, 20 e 50 tópicos. Depois eles fizeram todas as uniões das coleções de cada classe obtidas após executar o LDA, o que resultou em 12 predições. O melhor resultado foi obtido na união das coleções obtidas após a aplicação do LDA considerando 10 tópicos *spam* e 50 tópicos *ham*, conforme é apresentado na Tabela 5.24.

Os autores também usaram a base de dados WEBSPAM-UK2007 para fazer experimentos usando os atributos pré-computados fornecidos na competição *Web Spam Challenge 2008*, além de atributos extraídos usando a abordagem Pivoted tf-idf (Singhal, Salton, Mitra & Buckley 1995) e atributos baseados em conectividade *sonar* (Amitay, Carmel, Darlow, Lempel & Soffer 2003). Os métodos de aprendizado usados para os experimentos foram o SVM, o C4.5 e a regressão logística. O melhor resultado foi obtido pelo método regressão logística no experimento em que foi combinada a melhor predição do LDA e todos os outros atributos, conforme pode ser visto na Tabela 5.24.

- (Martinez-Romo & Araujo 2009) usaram uma abordagem de modelos de linguagem para analisar diferentes fontes de informações extraídas de páginas *Web* com o objetivo de providenciar bons indicadores para a detecção de *Web spam*. Então, eles aplicaram o método Kullback-Leibler *divergence* (KLD) para medir a divergência entre a probabilidade de distribuição de termos entre duas páginas *Web* que estejam conectadas por algum *link*. Usando esse método eles extraíram 42 atributos, baseados em modelo de linguagem, das bases de dados WEBSPAM-UK2006 e WEBSPAM-UK2007.

  Nos experimentos, (Martinez-Romo & Araujo 2009) só consideraram as amostras de *hosts* que foram rotuladas por duas pessoas diferentes e que receberam o mesmo rótulo (*spam* ou *ham*) das duas pessoas. Depois disso, eles também eliminaram as páginas que não possuem texto suficiente, que não possuem *links* de saída e outras características que não foram mencionadas. Dessa forma, para a base de dados WEBSPAM-UK2006 eles usaram 3.083 *hosts* (1.811 *hosts ham* e 1.272 *hosts spam*) e para a base de dados WEBSPAM-UK2007 eles usaram 4.166 *hosts* (4.012 *hosts ham* e 154 *hosts spam*).

  O método de aprendizado que os autores usaram nos experimentos foi o método *metacost* (árvores de decisão sensíveis ao custo com *bagging*). Eles analisaram os resultados da classificação dos atributos baseados em modelos de linguagem (LM – *language models*) propostos por eles, dos atributos pré-computados baseados no conteúdo e nos *links* transformados[1] fornecidos nas competições *Web Spam Challenge* e de todas as combinações desses atributos. Os melhores resultados foram obtidos pelos autores nos experimentos em que foram combinados os atributos baseados no conteúdo com os atributos baseados nos *links* transformados e nos experimentos em que foram combinados todos os atributos, conforme pode ser visto na tabela 5.24.

[1]No trabalho dos autores (Martinez-Romo & Araujo 2009), eles chamam os atributos baseados nos *links* transformados de atributos baseados nos *links*. Porém, como eles afirmam que o espaço desses atributos possui 139 dimensões, concluiu-se que na verdade eles estavam usando os atributos baseados nos *links* transformados fornecidos nas competições *Web Spam Challenge*.

- (Araujo & Martinez-Romo 2010) apresentaram um trabalho que pode ser considerado uma extensão de sua outra publicação (Martinez-Romo & Araujo 2009), que foi apresentado anteriormente nessa seção. Além de apresentar os resultados obtidos com a abordagem de modelos de linguagem (LM – *language models*) de seu trabalho anterior, os autores fizeram uma análise da relação de *links* na *Web* do ponto de vista de qualidade, tais como: verificar se os *links* estão quebrados, medir a diferença entre *links* internos e externos ou entre *links* de entrada e de saída, analisar o conteúdo dos textos âncoras, analisar os termos que compõem a URL, entre outros. Por meio dessa análise, eles extraíram 12 atributos baseados na qualidade dos *links* (QL – *qualified-link*).

  Nos experimentos, eles usaram as mesmas bases de dados, quantidades de amostras e atributos do trabalho (Martinez-Romo & Araujo 2009) que foi apresentado anteriormente. A única alteração em relação ao trabalho anterior é que foram acrescentados os atributos baseados em QL. Em relação aos métodos de aprendizado, eles fizeram experimentos com os métodos *metacost* (árvores de decisão C4.5 sensíveis ao custo com *bagging*), *naive* Bayes, regressão logística e otimização mínima sequencial (SMO – *sequential minimal optimization*) usando para o treinamento os atributos baseados em LM e QL. Entre esses métodos, o que obteve melhor desempenho foi o *metacost*. Diante disso, os autores o usaram em novos experimentos com combinações dos atributos baseados em LM, QL, conteúdo e *links* transformados. Os melhores resultados foram obtidos na combinação de todos os atributos e na combinação dos atributos baseados em QL, conteúdo e *links* transformados, conforme é apresentado na Tabela 5.24.

- (Mahmoudi, Yari & Khadivi 2010) estudaram o espaço de atributos da tarefa de detecção de *Web spam* com o objetivo de encontrar os atributos mais efetivos e discriminativos. A base de dados usada pelos autores foi a WEBSPAM-UK2007. Além disso, eles usaram os atributos pré-computados baseados no conteúdo, nos *links* transformados e os atributos diretos (número de páginas do *host* e número de caracteres do nome do *host*) oferecidos na competição *Web Spam Challenge* 2008. Desses conjuntos de atributos, eles selecionaram, com base na idéia de escolha facilmente computável, 140 atributos para os experimentos. Para aplicar a seleção de atributos, eles usaram os seguintes métodos: IG (*information gain*), χ^2 (chi-*squared*), CFS (*correlation based feature selection*) e SVM (*Support Vector Machine*). Depois de aplicar esses métodos de seleção de atributos eles fizeram experimentos com os algoritmos de classificação redes neurais, SVM, naive Bayes e LADTree (Holmes, Pfahringer, Kirkby, Frank & Hall 2002), usando os atributos com dimensionalidade reduzida e com todos os 140 atributos. Nos experimentos eles usaram a mesma quantidade de amostras *spam* e *ham* e aplicaram o método de validação cruzada *k-folds* com $k = 10$, para calcular os resultados. Os melhores resultados foram obtidos nos experimentos com o método floresta aleatória usando todos os atributos e com o método LADTree usando 26 atributos selecionados após a redução de dimensionalidade por meio do método CFS, conforme é apresentado na Tabela 5.24.

- (Shengen, Xiaofei, Peiqi & Lin 2011) usaram programação genética para criar novos atributos para a identificação de *Web spam* derivados dos atributos baseados em *links*. Os

experimentos foram realizados com a base de dados WEBSPAM-UK2006. Eles selecionaram 470 *spam hosts* e 500 *ham hosts* para o treinamento e 204 *spam hosts* e 300 *ham hosts* para o teste. Os autores analisaram os efeitos do número de indivíduos, do número de gerações e da profundidade da árvore binária nos desempenho da programação genética e geraram 10 novos atributos para ajudar na classificação de *Web spam*. Depois disso, eles fizeram experimentos usando métodos SVM e programação genética na classificação das páginas *Web* representadas pelos novos atributos propostos por eles, pelos atributos baseados nos *links* e pelos atributos baseados nos *links* transformados. Os melhores resultados foram obtidos pelo classificador baseado em programação genética, usando os atributos baseados nos *links* transformados e os novos atributos gerados por programação genética, conforme é apresentado na tabela 5.24.

- (Pavlov & Dobrov 2011) criaram um *framework* para detecção de *Web spam* baseado na concepção de que os *spammers* não conseguem emular todos os aspectos naturais dos textos que compõem uma página *Web* normal. Então, eles propuseram novos atributos baseados na qualidade do texto e nas características de sua linguagem natural. Os autores usaram um etiquetador das partes do discurso (POS – *part-of-speech*) (Piskorski, Sydow & Weiss 2008) para etiquetar cada palavra da base de dados WEBSPAM-UK2007 e extrair estatísticas linguísticas, o que resultou em um total de 145 atributos. Eles extraíram também 7 atributos baseados na legibilidade do texto e, dessa forma, passaram a existir 152 atributos baseados em estatísticas linguísticas (*SF – statistical features*). Pelo fato de acreditarem que esses atributos são insuficientes para detectar tipos avançados de *spam*, os autores também estraíram atributos baseados na diversidade do conteúdo (DF – *diversity features*) das páginas *Web* e atributos baseados em tópicos (temas abordados pela página *Web*, como por exemplo pornografia). Este último tipo de atributo foi extraído usando latent Dirichlet allocation (LDA) (Blei et al. 2003). Os autores fizeram experimentos com textos sintéticos gerados por um gerador de texto Markoviano, experimentos que mediram o benefício dos atributos propostos e experimentos com a base de dados WEBSPAM-UK2007. O classificador usado por eles foi a regressão logística (*logistic regression*). O melhor resultado do experimento feito pelos autores usando a base de dados WEBSPAM-UK2007 foi obtido na combinação dos atributos SF, DF e LDA e é apresentado na Tabela 5.24.

Diante do que é apresentado na Tabela 5.24, pode-se concluir que os métodos *bagging* e *AdaBoost* obtiveram desempenho superior aos métodos propostos em outros trabalhos da literatura. Portanto, é conclusivo que métodos de aprendizado de máquina podem ser empregados com sucesso para auxiliar o processo de detecção automática de *Web spam*. Além disso, a abordagem proposta nesse trabalho, que faz a combinação das predições obtidas com cada tipo de atributo, foi superior aos trabalhos usados para comparação.

Tabela 5.24: Comparação entre os melhores resultados obtidos nesse trabalho e os resultados disponíveis na literatura.

Classificadores F-medida	
Resultados disponíveis na literatura e obtidos com a base de dados WEBSPAM-UK2006	
Bagging (Castillo et al. 2007) - conteúdo + *links* transformados	0,723
Agrupamento + *Bagging* (Castillo et al. 2007)	0,728
Propagação + *Bagging* (Castillo et al. 2007)	0,733
SGL (Castillo et al. 2007)	0,763
LDA (Bíró, Siklósi, Szabó & Benczúr 2009)	0,735
Metacost (Martinez-Romo & Araujo 2009) – conteudo + *links* 0,75	
Metacost (Martinez-Romo & Araujo 2009) – conteudo + *links* + LM	0,81
Metacost (Araujo & Martinez-Romo 2010) – conteudo + *links* + QL	0,83
Metacost (Araujo & Martinez-Romo 2010) – conteudo + *links* + LM + QL	0,86
Programação genética (Shengen et al. 2011) - atributos criados por programação genética	0,80
Programaçao genética (Shengen et al. 2011) - links transformados	0,789

Resultados disponíveis na literatura e obtidos com a base de dados WEBSPAM-UK2007	
LDA (Bíró et al. 2009)	0,458
Regressão logística (Bíró et al. 2009) – *Web Spam Challenge* + Pivoted tf-idf + *sonar* + LDA	0,667
Metacost (Martinez-Romo & Araujo 2009) – conteudo + *links*	0,31
Metacost (Martinez-Romo & Araujo 2009) – conteudo + *links* + LM	0,33
Metacost (Araujo & Martinez-Romo 2010) – conteudo + *links* + QL	0,38
Metacost (Araujo & Martinez-Romo 2010) – conteudo + *links* + LM + QL	0,40
Floresta aleatória (Mahmoudi et al. 2010)	0,714
LADTree (Mahmoudi et al. 2010)	0,707
Regressão logística (Pavlov & Dobrov 2011)	0,458

Melhores resultados obtidos nesse trabalho usando a base de dados WEBSPAM-UK2006	
MLP-GD - conteúdo + *links* + *links* transformados (classes balanceadas)	0,906
AdaBoost - conteúdo + *links* transformados (classes balanceadas)	**0,916**
MLP-GD - combinação de predições	0,905

Melhores resultados obtidos nesse trabalho usando a base de dados WEBSPAM-UK2007	
Bagging - conteúdo + *links* transformados (classes balanceadas)	0,923
AdaBoost - conteúdo + *links* transformados (classes balanceadas)	**0,944**
Bagging - combinação de predições	0,911

Capítulo 6

Considerações Finais

Este trabalho apresentou uma compreensiva análise do desempenho de diferentes algoritmos de aprendizado de máquina, conceituados na literatura, aplicados na detecção automática de *Web spam*. Para isso, foram utilizadas duas bases de dados reais, públicas e de grande porte, representadas por vetores de atributos baseados no conteúdo das páginas *Web*, nos *links*, nos *links* transformados e por suas combinações.

Os resultados dos experimentos mostraram que entre os métodos avaliados, o *bagging* de árvores de decisão teve melhor desempenho que os outros métodos na classificação de *Web spam* sem combinação de vetores de atributos. Porém, nos cenários em que foram testadas combinações de atributos, em geral, o método *AdaBoost* se destacou. Os dois métodos também se destacaram nas simulações com a nova abordagem proposta nesse trabalho, que combina as predições obtidas com cada tipo de vetor de atributos. Isso demonstra que esses métodos são adequados para auxiliar na detecção de *Web spam*.

Por meio de análise estatística foi possível observar que, nos cenários em que foram usadas amostras extraídas da base de dados WEBSPAM-UK2006, as redes neurais MLP e o método floresta aleatória obtiveram resultados equivalentes aos métodos *bagging* e *AdaBoost*, quando foram usadas classes balanceadas. Logo, eles também são métodos promissores para a detecção automática de *Web spam*. Nos cenários em que foram usadas amostras da base de dados WEBSPAM-UK2007, os método MLP-LM e floresta aleatória também ficaram no grupo dos melhores métodos, mas a MLP-GD não obteve o mesmo sucesso.

Com relação aos vetores de atributos, nas simulações com a base de dados WEBSPAM-UK2006, os melhores resultados foram obtidos quando foram usados os atributos baseados nos *links* transformados. Todavia, nas simulações com a base de dados WEBSPAM-UK2007, os resultados foram melhores no cenário em que foram usados atributos baseados no conteúdo. No entanto, a combinação de vetores de atributos, em geral, foi mais efetiva, gerando resultados bastante superiores ao que foi obtido nas simulações em que os vetores de atributos não foram combinados. Nesse cenário, a combinação dos atributos baseados no conteúdo com os atributos baseados nos *links* transformados foi a mais promissora.

Outro ponto importante analisado nesse trabalho foi o impacto nos resultados causado pelo tratamento do desbalanceamento entre as classes. Os resultados obtidos nas simulações com a base de dados WEBSPAM-UK2006 indicaram que os métodos de aprendizado são mais eficientes

quando são treinados com número igual de representantes em cada classe, pois ficou evidente que a classificação torna-se tendenciosa para a classe com maior número de amostras usadas na etapa de treinamento. Nas simulações com a base de dados WEBSPAM-UK2007 não foi possível realizar simulações com classes desbalanceadas, pois a pequena quantidade de representantes da classe *spam*, em detrimento a quantidade superior de representantes da classe *ham*, não foi suficiente para que alguns classificadores aprendessem a detectar as amostras da classe positiva.

Além disso, as abordagens usadas nesse trabalho para tratar o desbalanceamento entre as classes foram efetivas. Um dos principais problemas discutidos na literatura que podem ser causados pela técnica de subamostragem aleatória, que é a perda de informações, não foi observado nos resultados apresentados nesse trabalho. Isso pode ser concluído, pois os desvios padrões dos resultados foram, em geral, inferiores a 5%, o que pode ser considerado um valor pequeno. Como em cada iteração, um novo conjunto de amostras é aleatoriamente selecionado, se houvesse perda de informações, provavelmente o desempenho dos classificadores sofreria grande oscilação e consequentemente os desvios padrões seriam superiores. Por outro lado, o principal problema discutido na literatura que pode ser causado pela técnica SMOTE (usada para balancear as classes da base de dados WEBSPAM-UK2007), que é o problema de *overfitting*, também não foi observado. Com esse problema, a taxa de sensitividade normalmente é alta, enquanto a taxa de especificadade é muito baixa, e isso não ocorreu nos cenários analisados nesse trabalho.

Esse trabalho analisou também o impacto da redução de dimensionalidade no desempenho dos classificadores. Para diminuir a dimensionalidade do espaço de atributos foi utilizado o método de análise de componentes principais, que é um dos métodos mais populares da literatura para a solução de problemas de alta dimensionalidade. De maneira geral, observou-se que a redução de dimensionalidade prejudicou a eficiência dos métodos de classificação. Os métodos mais afetados foram os que tiveram melhor desempenho na classificação de *Web spam*. Porém, alguns foram beneficiados em alguns cenários de classificação, tais como os métodos MLP-LM, SVM, *naive* Bayes, RBF e SOM + LVQ.

Por fim, a nova abordagem de classificação de *Web spam* proposta nesse trabalho, que faz a combinação das predições obtidas com cada tipo de vetor de atributos, gera resultados inferiores aos obtidos com a combinação de atributos. Porém, essa abordagem produz melhores resultados do que a classificação isolada dos vetores de atributos e por isso, é uma boa estratégia de classificação de *Web spam* e é competitiva com outros métodos da literatura.

Trabalhos futuros

Durante a elaboração e após a conclusão dessa dissertação, foram verificados alguns pontos em que o trabalho pode ser melhorado e complementado. Diante disso, a seguir são apresentadas algumas sugestões de trabalhos futuros:

- criação de uma novo conjunto de atributos formado pela fusão dos 3 tipos de vetores de atributos usados nesse trabalho;
- proposição de novos tipos de atributos que possam aumentar a capacidade de predição dos algoritmos;

- aplicação de outras técnicas de análise e seleção de atributos;
- o estudo de formas de adaptar os métodos mais promissores para otimizar seu desempenho;
- criação de uma base de dados de *Web spam* formada por páginas *Web* da língua portuguesa e análise do desempenho dos algoritmos de aprendizado de máquina na classificação das amostras dessa base de dados.

Publicações

Nessa seção são apresentados os trabalhos publicados que apresentam os resultados de pesquisa relatados nessa dissertação:

- R. M. Silva, A. G. Vaz, T. A. Almeida, A. Yamakami. Avaliação de Desempenho de Métodos de Classificação Aplicados na Identificação de *Spam Host*. *Revista Brasileira de Sistemas de Informação*.
- R. M. Silva, T. A. Almeida, A. Yamakami. An analysis of machine learning methods for spam host detection. *Proceedings of the 11th Conference on Machine Learning and Applications (ICMLA'12)*. Boca Raton, Florida, USA, 2012. (Prêmio de melhor trabalho da seção especial *Machine Learning in Information and System Security Issues*)
- R. M. Silva, T. A. Almeida, A. Yamakami. Análise de Métodos de Aprendizagem de Máquina para Detecção Automática de *Spam Hosts*. *In. Anais do XII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg'12)*, p:2–15, Curitiba, PR, Brasil. Sociedade Brasileira de Computação.
- R. M. Silva, T. A. Almeida, A. Yamakami. Towards Web Spam Filtering with Neural-Based Approaches. *Advances in Artificial Intelligence – IBERAMIA 2012*, volume 7637 of Lecture Notes in Computer Science, p: 199–209, Cartagena das Índias, Colombia, 2012. Springer Berlin Heidelberg.
- R. M. Silva, T. A. Almeida, A. Yamakami. Análise de Desempenho de Redes Neurais Artificiais para Classificação Automática de Web Spam. *Revista Brasileira de Computação Aplicada*, 4(2):42–57,2012.
- R. M. Silva, T. A. Almeida, A. Yamakami. Artificial Neural Networks for Content-based Web Spam Detection. *Proceedings of the 14th International Conference on Artificial Intelligence (ICAI'12)*, p:37–44. Las vegas, Nevada, USA, 2012.
- R. M. Silva, T. A. Almeida, A. Yamakami. Redes Neurais Artificiais para Detecção de Web Spams. *In. Anais do VIII Simpósio Brasileiro de Sistemas de Informação (SBSI'12)*, p:636–641. São Paulo, SP, Brasil, 2012.

Bibliografia

Aha, D. W., Kibler, D. & Albert, M. K. (1991). Instance-based learning algorithms, *Machine Learning* **6**(1): 37–66.

Amitay, E., Carmel, D., Darlow, A., Lempel, R. & Soffer, A. (2003). The connectivity sonar: detecting site functionality by structural patterns, *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia (HYPERTEXT'03)*, ACM, New York, NY, USA, pp. 38–47.

Araujo, L. & Martinez-Romo, J. (2010). Web spam detection: New classification features based on qualified link analysis and language models, *IEEE Transactions on Information Forensics and Security* **5**(3): 581–590.

Attenberg, J. & Suel, T. (2008). Cleaning search results using term distance features, *Proceedings of the 4th international workshop on Adversarial information retrieval on the web (AIRWeb'08)*, ACM, New York, NY, USA, pp. 21–24.

Becchetti, L., Castillo, C., Donato, D., Baeza-Yates, R. & Leonardi, S. (2008). Link analysis for web spam detection, *ACM Transactions on the Web* **2**(1): 2:1–2:42.

Becchetti, L., Castillo, C., Donato, D., Leonardi, S. & Baeza-Yates, R. (2006a). Link-based characterization and detection of web spam, *Proceedings of the 2nd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb'06)*, Seattle, WA, USA.

Becchetti, L., Castillo, C., Donato, D., Leonardi, S. & Baeza-Yates, R. (2006b). Using rank propagation and probabilistic counting for link-based spam detection, *Proceedings of the 2006 Workshop on Web Mining and Web Usage Analysis (WebKDD'06)*, Philadelphia,USA.

Bertoni, A., Frasca, M. & Valentini, G. (2011). COSNet: A cost sensitive neural network for semi-supervised learning in graphs, *Machine Learning and Knowledge Discovery in Databases*, Vol. 6911 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 219–234.

Bingham, E. & Mannila, H. (2001). Random projection in dimensionality reduction: applications to image and text data, *Proceedings of the seventh ACM SIGKDD international*

conference on Knowledge discovery and data mining (KDD'01), ACM, New York, NY, USA, pp. 245–250.

Bíró, I., Siklósi, D., Szabó, J. & Benczúr, A. A. (2009). Linked latent dirichlet allocation in web spam filtering, *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb'09)*, Madrid, Spain, pp. 37–40.

Bíró, I., Szabó, J. & Benczúr, A. A. (2008). Latent dirichlet allocation in web spam filtering, *Proceedings of the 4th international workshop on Adversarial information retrieval on the web (AIRWeb'08)*, ACM, New York, NY, USA, pp. 29–32.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*, 1th edn, Oxford Press, Oxford, UK.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*, 1th edn, Springer, New York, NY, USA.

Blei, D. M., Ng, A. Y. & Jordan, M. I. (2003). Latent dirichlet allocation, *The Journal of Machine Learning Research* **3**: 993–1022.

Braga, A. P., Ludermir, T. B. & Carvalho, A. C. P. L. F. (2007). *Redes Neurais Artificiais: Teorias e Aplicações*, 1th edn, Livros Técnicos e Científicos, Rio de Janeiro.

Breiman, L. (1996). Bagging predictors, *Machine Learning* **24**: 123–140.

Breiman, L. (2001). Random forests, *Machine Learning* **45**(1): 5–32.

Brin, S. & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine, *Proceedings of the seventh International Conference on World Wide Web (WWW'98*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, pp. 107–117.

Bunkhumpornpat, C., Sinapiromsaran, K. & Lursinsap, C. (2012). DBSMOTE: Density-based synthetic minority over-sampling technique, *Applied Intelligence* **36**: 664–684.

Cahill, K. & Chalut, R. (2009). Optimal results: What libraries need to know about google and search engine optimization, *The Reference Librarian* **50**(3): 234–247.

Carter, J. & Dewan, P. (2010). Design, implementation, and evaluation of an approach for determining when programmers are having difficulty, *Proceedings of the 16th ACM international conference on Supporting group work (GROUP'10)*, ACM, New York, NY, USA, pp. 215–224.

Castillo, C., Donato, D. & Gionis, A. (2007). Know your neighbors: Web spam detection using the web topology, *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'07)*, Amsterdam, The Netherlands, pp. 423–430.

Chai, X., Deng, L., Yang, Q. & Ling, C. X. (2004). Test-cost sensitive naive bayes classification, *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04)*, IEEE Computer Society, Washington, DC, USA, pp. 51–58.

Chang, C. & Lin, C. (2011). LIBSVM: A library for support vector machines, *ACM Transactions on Intelligent Systems and Technology* **2**: 27:1–27:27.

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique, *Journal of Artificial Intelligence Research* **16**(1): 321–357.

Chellapilla, K. & Maykov, A. (2007). A taxonomy of javascript redirection spam, *Proceedings of the 3rd international workshop on Adversarial information retrieval on the web (AIRWeb'07)*, ACM, New York, NY, USA, pp. 81–88.

Cohen, W. W. & Kou, Z. (2006). Stacked graphical learning: approximating learning in markov random fields using very short inhomogeneous markov chains, *Technical report*.

Cortes, C. & Vapnik, V. N. (1995). Support-vector networks, *Machine Learning*, pp. 273–297.

Del Gaudio, R. & Branco, A. (2009). Language independent system for definition extraction: first results using learning algorithms, *Proceedings of the 1st Workshop on Definition Extraction (WDE'09)*, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 33–39.

Domingos, P. (1999). MetaCost: a general method for making classifiers cost-sensitive, *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'99)*, ACM, New York, NY, USA, pp. 155–164.

Dong, Y. & Wang, X. (2011). A new over-sampling approach: Random-SMOTE for learning from imbalanced data sets, *Proceedings of the 5th international Conference on Knowledge Science, Engineering and Management (KSEM'11)*, Springer-Verlag, Berlin, Heidelberg, pp. 343–352.

Dou, Z., Song, R., Yuan, X. & Wen, J. (2008). Are click-through data adequate for learning web search rankings?, *Proceedings of the 17th ACM conference on Information and knowledge management (CIKM'08)*, ACM, New York, NY, USA, pp. 73–82.

Egele, M., Kolbitsch, C. & Platzer, C. (2011). Removing web spam links from search engine results, *Journal in Computer Virology* **7**: 51–62.

Eiron, N., McCurley, K. S. & Tomlin, J. A. (2004). Ranking the web frontier, *Proceedings of the 13rd International Conference on World Wide Web (WWW'04)*, New York, NY, USA, pp. 309–318.

Elkan, C. (2001). The foundations of cost-sensitive learning, *Proceedings of the 17th international joint conference on Artificial intelligence (IJCAI'01)*, Vol. 21, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 973–978.

Erdélyi, M., Garzó, A. & Benczúr, A. A. (2011). Web spam classification: a few features worth more, *Proceedings of the 2011 Joint WICOW/AIRWeb Workshop on Web Quality (WebQuality'11)*, Hyderabad, India, pp. 27–34.

Fetterly, D., Manasse, M. & Najork, M. (2004). Spam, damn spam, and statistics: using statistical analysis to locate spam web pages, *Proceedings of the 7th International Workshop on the Web and Databases (WebDB'04)*, ACM, New York, NY, USA, pp. 1–6.

Freund, Y. & Schapire, R. E. (1996). Experiments with a new boosting algorithm, *Proceedings of the 13th International Conference on Machine Learning (ICML'96)*, Morgan Kaufmann, Bari, Italy, pp. 148–156.

Freund, Y. & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting, *Journal of Computer and System Sciences* **55**(1): 119–139.

Friedman, J. H. & Tukey, J. W. (1974). A projection pursuit algorithm for exploratory data analysis, *IEEE Transactions on Computers* **23**(9): 881–890.

Friedman, J., Hastie, T. & Tibshirani, R. (1998). Additive logistic regression: a statistical view of boosting, *Annals of Statistics* **28**(2): 337–407.

Galar, M., Fernández, A., Barrenechea, E., Bustince, H. & Herrera, F. (2012). A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches, *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* **42**(4): 463–484.

Geng, G., Wang, C., Li, Q., Xu, L. & Jin, X. (2007). Boosting the performance of web spam detection with ensemble under-sampling classification, *Proceedings of the 14th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD'07)*, Haikou, China, pp. 583–587.

Guo, Y., Zhang, H. & Spencer, B. (2012). Cost-sensitive self-training, *Advances in Artificial Intelligence*, Vol. 7310 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 74–84.

Gyöngyi, Z. & Garcia-Molina, H. (2005a). Link spam alliances, *Proceedings of the 31st international conference on Very large data bases (VLDB'05)*, VLDB Endowment, pp. 517–528.

Gyongyi, Z. & Garcia-Molina, H. (2005b). Spam: It's not just for inboxes anymore, *Computer* **38**(10): 28–34.

Gyongyi, Z., Garcia-Molina, H. & Pedersen, J. (2004). Combating web spam with trustrank, *Proceedings of the 30th International Conference on Very Large Databases (VLDB'04)*, Toronto, Canada, pp. 576–587.

Hagan, M. T. & Menhaj, M. B. (1994). Training feedforward networks with the marquardt algorithm, *IEEE Transactions on Neural Networks* **5**(6): 989–993.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I. H. (2009). The WEKA data mining software: an update, *SIGKDD Explorations Newsletter* **11**(1): 10–18.

Han, J. & Kamber, M. (2006). *Data Mining: Concepts and Techniques*, 2th edn, Morgan Kaufmann, San Francisco, CA, USA.

Hart, P. E. (1968). The condensed nearest neighbor rule, *IEEE Transactions on Information Theory* **14**: 515–516.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*, 2th edn, Prentice Hall, New York, NY, USA.

He, H. & Garcia, E. A. (2009). Learning from imbalanced data, *IEEE Transactions on Knowledge and Data Engineering* **21**(9): 1263–1284.

Henzinger, M. R., Motwani, R. & Silverstein, C. (2002). Challenges in web search engines, *SIGIR Forum* **36**(2): 11–22.

Hinton, G. E. & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks, *Science (New York, N.Y.)* **313**(5786): 504–507.

Holmes, G., Pfahringer, B., Kirkby, R., Frank, E. & Hall, M. (2002). Multiclass alternating decision trees, *Proceedings of the 13th European Conference on Machine Learning (ECML'02)*, Springer-Verlag, London, UK, UK, pp. 161–172.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets, *Machine Learning* **11**(1): 63–90.

Hsu, C., Chang, C. & Lin, C. (2003). A practical guide to support vector classification, *Technical report*, National Taiwan University.

Hyvärinen, A. & Oja, E. (2000). Independent component analysis: algorithms and applications, *Neural Networks* **13**(4–5): 411–430.

Jan, T., Wang, D., Lin, C. & Lin, H. (2012). A simple methodology for soft cost-sensitive classification, *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'12)*, ACM, New York, NY, USA, pp. 141–149.

Jayanthi, S. K. & Sasikala, S. (2012). WESPACT: Detection of web spamdexing with decision trees in ga perspective, *Proceedings of the 2012 International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME'12)*, pp. 381–386.

John, G. H. & Langley, P. (1995). Estimating continuous distributions in bayesian classifiers, *Proc. of the 11th UAI)*, Montreal, Quebec, Canada, pp. 338–345.

John, J. P., Yu, F., Xie, Y., Krishnamurthy, A. & Abadi, M. (2011). deSEO: combating search-result poisoning, *Proceedings of the 20th USENIX conference on Security (SEC'11)*, Berkeley, CA, USA, pp. 20–20.

Johnson, R. A. & Wichern, D. W. (1988). *Applied multivariate statistical analysis*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Kohonen, T. (1990). The self-organizing map, *Proceedings of the IEEE*, Vol. 9, pp. 1464–1480.

Kubat, M. & Matwin, S. (1997). Addressing the curse of imbalanced training sets: One-sided selection, *Proceedings of the 14th International Conference on Machine Learning*, Morgan Kaufmann, pp. 179–186.

Kukar, M. & Kononenko, I. (1998). Cost-sensitive learning with neural networks, *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI'98)*, John Wiley e Sons, Chichester, UK, pp. 445–449.

Kuncheva, L. I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*, 1 edn, John Wiley & Sons, Inc., Hoboken, New Jersey, USA.

Langley, P. (1993). Induction of recursive bayesian classifiers, *Proceedings of the European Conference on Machine Learning (ECML'93)*, Springer-Verlag, London, UK, UK, pp. 153–164.

Langley, P. & Sage, S. (1994). Induction of selective bayesian classifiers, *Proceedings of the 10th international conference on Uncertainty in artificial intelligence (UAI'94)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 399–406.

Largillier, T. & Peyronnet, S. (2010). Lightweight clustering methods for webspam demotion, *Proceedings of the 9th IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'10)*, Toronto, Canada, pp. 98–104.

Largillier, T. & Peyronnet, S. (2012). Webspam demotion: Low complexity node aggregation methods, *Neurocomputing* **76**(1): 105–113.

Ledford, J. L. (2009). *Search Engine Optimization Bible*, 2th edn, Wiley Publishing, Indianapolis, Indiana, USA.

Lin, J. (2009). Detection of cloaked web spam by using tag-based methods, *Expert Systems with Applications: An International Journal* **36**(4): 7493–7499.

Ling, C. X., Yang, Q., Wang, J. & Zhang, S. (2004). Decision trees with minimal costs, *Proceedings of the twenty-first international conference on Machine learning (ICML'04)*, ACM, New York, NY, USA, pp. 69–76.

Liu, H. (2010). On the levenberg-marquardt training method for feed-forward neural networks, *Proceedings of the 6th International Conference on Natural Computation (ICNC'10)*, Yantai,China, pp. 456–460.

Liu, X., Wu, J. & Zhou, Z. (2006). Exploratory under-sampling for class-imbalance learning, *Proceedings of the Sixth International Conference on Data Mining (ICDM'06)*, IEEE Computer Society, Washington, DC, USA, pp. 965–969.

Liu, Y., Chen, F., Kong, W., Yu, H., Zhang, M., Ma, S. & Ru, L. (2012). Identifying web spam with the wisdom of the crowds, *ACM Transactions on the Web* **6**(1): 2:1–2:30.

Lu, L., Perdisci, R. & Lee, W. (2011). SURF: detecting and measuring search poisoning, *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS'11)*, New York, NY, USA, pp. 467–476.

Mahmoudi, M., Yari, A. & Khadivi, S. (2010). Web spam detection based on discriminative content and link features, *Proceedings of the 5th International Symposium on Telecommunications (IST'10)*, Tehran, Iran, pp. 542–546.

Martinez-Romo, J. & Araujo, L. (2009). Web spam identification through language model analysis, *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web (AIRWeb'09)*, ACM, New York, NY, USA, pp. 21–28.

Montgomery, D. C. & Runger, G. C. (2002). *Applied Statistics and Probability for Engineers*, 3th edn, John Wiley & Sons, New York, NY, USA.

Nahar, J., Imam, T., Tickle, K. S., Ali, A. S. & Chen, Y.-P. P. (2012). Computational intelligence for microarray data and biomedical image analysis for the early diagnosis of breast cancer, *Expert Systems with Applications* **39**(16): 12371–12377.

Najork, M. (2009). Web spam detection, *Encyclopedia of Database Systems*, Vol. 1, Springer US, pp. 3520–3523.

Ntoulas, A., Najork, M., Manasse, M. & Fetterly, D. (2006). Detecting spam web pages through content analysis, *Proceedings of the World Wide Web conference (WWW'06)*, Edinburgh, Scotland, pp. 83–92.

Orr, M. J. L. (1996). Introduction to radial basis function networks.

Page, L., Brin, S., Motwani, R. & Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web, *Technical report*, Stanford Uiversity.

Pavlov, A. & Dobrov, B. (2011). Detecting content spam on the web through text diversity analysis, *Proceedings of the 7th Spring Young Researcher's Colloquium On Database and Information Systems (SYRCoDIS'11)*, Vol. 735 of *CEUR Workshop Proceedings*, CEUR-WS.org, Moscow, Russia, pp. 11–18.

Piskorski, J., Sydow, M. & Weiss, D. (2008). Exploring linguistic features for web spam detection: a preliminary study, *Proceedings of the 4th international workshop on Adversarial information retrieval on the web (AIRWeb'08)*, ACM, New York, NY, USA, pp. 25–28.

Provos, N., Mavrommatis, P., Rajab, M. A. & Monrose, F. (2008). All your iFRAMEs point to us, *Proceedings of the 17th conference on Security symposium (SS'08)*, Berkeley, CA, USA, pp. 1–15.

Quinlan, J. R. (1993). *C4.5: programs for machine learning*, 1th edn, Morgan Kaufmann, San Mateo, CA, USA.

Ramentol, E., Caballero, Y., Bello, R. & Herrera, F. (2012). SMOTE-RSB: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using smote and rough sets theory, *Knowledge and Information Systems* **33**: 245–265.

Rungsawang, A., Taweesiriwate, A. & Manaskasemsak, B. (2011). Spam host detection using ant colony optimization, *IT Convergence and Services*, Vol. 107 of *Lecture Notes in Electrical Engineering*, Springer Netherlands, pp. 13–21.

Shao, J. (1993). Linear model selection by cross-validation, *Journal of the American Statistical Association* **88**(422): 486–494.

Shen, G., Gao, B., Liu, T., Feng, G., Song, S. & Li, H. (2006). Detecting link spam using temporal information, *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM'06)*, Hong Kong, China, pp. 1049–1053.

Sheng, V. S. & Ling, C. X. (2006). Thresholding for making classifiers cost-sensitive, *Proceedings of the 21st national conference on Artificial intelligence (AAAI'06)*, Vol. 1, AAAI Press, pp. 476–481.

Shengen, L., Xiaofei, N., Peiqi, L. & Lin, W. (2011). Generating new features using genetic programming to detect link spam, *Proceedings of the 2011 International Conference on Intelligent Computation Technology and Automation (ICICTA'11)*, Shenzhen, China, pp. 135–138.

Singhal, A., Salton, G., Mitra, M. & Buckley, C. (1995). Document length normalization, *Technical report*, Ithaca, NY, USA.

Smith, L. I. (2002). A Tutorial on Principal Component Analysis, *Technical report*, Cornell University, USA.

Spirin, N. & Han, J. (2012). Survey on web spam detection: principles and algorithms, *ACM SIGKDD Explorations Newsletter* **13**(2): 50–64.

Sunil, A. V. & Sardana, A. (2012). A reputation based detection technique to cloaked web spam, *Procedia Technology* **4**(0): 566–572.

Svore, K. M., Wu, Q. & Burges, C. J. (2007). Improving web spam classification using rank-time features, *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb'07)*, Banff, Alberta, Canada, pp. 9–16.

Taweesiriwate, A., Manaskasemsak, B. & Rungsawang, A. (2012). Web spam detection using link-based ant colony optimization, *Proceedings of the 26th International Conference on Advanced Information Networking and Applications (AINA'12)*, pp. 868–873.

Ting, K. M. (1998). Inducing cost-sensitive trees via instance weighting, *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98)*, Springer-Verlag, London, UK, pp. 139–147.

Tomek, I. (1976). Two modifications of cnn, *IEEE Transactions on Systems, Man and Cybernetics* **6**(11): 76–772.

Turney, P. D. (1995). Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm, *Journal of Artificial Intelligence Research* **2**: 369–409.

Wang, Y., Ma, M., Niu, Y. & Chen, H. (2007). Spam double-funnel: connecting web spammers with advertisers, *Proceedings of the 16th international conference on World Wide Web (WWW'07)*, ACM, New York, NY, USA, pp. 291–300.

Wei, C., Liu, Y., Zhang, M., Ma, S., Ru, L. & Zhang, K. (2012). Fighting against web spam: a novel propagation method based on click-through data, *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval (SIGIR'12)*, ACM, New York, NY, USA, pp. 395–404.

Westbrook, A. & Greene, R. (2002). Using semantic analysis to classify search engine spam, *Technical report*, Stanford University.

Witten, I. H., Frank, E. & Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd edn, Morgan Kaufmann, San Francisco, CA, USA.

Wu, B. & Davison, B. D. (2006). Detecting semantic cloaking on the web, *Proceedings of the 15th international conference on World Wide Web (WWW'06)*, ACM, New York, NY, USA, pp. 819–828.

Wu, B. & Davison, B. D. (n.d.). Cloaking and redirection: A preliminary study.

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, Ng, A., Liu, B., Yu, P. S., Zhou, Z., Steinbach, M., Hand, D. J. & Steinberg, D. (2008). Top 10 algorithms in data mining, *Knowledge and Information Systems* **14**(1): 1–37.

Zhang, W., Zhu, D., Zhang, Y., Zhou, G. & Xu, B. (2011). Harmonic functions based semi-supervised learning for web spam detection, *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC'11)*, New York, USA, pp. 74–75.

Apêndice A

Parâmetros e resultados do método máquinas de vetores de suporte

A.1 Parâmetros da biblioteca LIBSVM usada para implementar o método máquinas de vetores de suporte

Tabela A.1: Parâmetros obtidos por *grid search* e usados no método SVM para a classificação das amostras de *Web spam* representadas por atributos extraídos do conteúdo, dos *links* e dos *links* transformados (base de dados WEBSPAM-UK2006).

Tipos de vetores de características	Classes balanceadas?	C	γ
Kernel linear			
Conteúdo	Sim	2^{15}	2^{3}
Conteúdo	Não	2^{13}	2^{3}
Links	Sim	2^{14}	2^{3}
Links	Não	2^{15}	2^{3}
Links transformados	Sim	2^{2}	2^{3}
Links transformados	Não	2^{2}	2^{3}
Kernel polinomial			
Conteúdo	Sim	2^{7}	2^{-4}
Conteúdo	Não	2^{13}	2^{2}
Links	Sim	2^{13}	2^{-3}
Links	Não	2^{14}	2^{-4}
Links transformados	Sim	2^{14}	2^{-8}
Links transformados	Não	2^{15}	2^{-8}
Kernel RBF			
Conteúdo	Sim	2^{14}	2^{3}
Conteúdo	Não	2^{15}	2^{3}
Links	Sim	2^{15}	2^{-1}
Links	Não	2^{15}	2^{-1}
Links transformados	Sim	2^{5}	2^{-5}
Links transformados	Não	2^{10}	2^{-9}
Kernel sigmoidal			
Conteúdo	Sim	2^{6}	2^{-6}
Conteúdo	Não	2^{8}	2^{-6}
Links	Sim	2^{14}	2^{-10}
Links	Não	2^{14}	2^{-5}
Links transformados	Sim	2^{13}	2^{-12}
Links transformados	Não	2^{10}	2^{-11}

Tabela A.2: Parâmetros obtidos por *grid search* e usados no método SVM para a classificação das amostras de *Web spam* representadas por todas as combinações entre os atributos extraídos do conteúdo, dos *links* e dos *links* transformados (base de dados WEBSPAM-UK2006).

Tipos de vetores de características	Classes balanceadas?	C	γ
Kernel linear			
Links + conteúdo	Sim	2^{14}	2^{3}
Links + conteúdo	Não	2^{11}	2^{3}
Links + *links* transformados	Sim	2^{12}	2^{3}
Links + *links* transformados	Não	2^{14}	2^{3}
Links transformados + conteúdo	Sim	2^{7}	2^{3}
Links transformados + conteúdo	Não	2^{8}	2^{3}
Links + *links* transformados + conteúdo	Sim	2^{13}	2^{3}
Links + *links* transformados + conteúdo	Não	2^{12}	2^{3}
Kernel polinomial			
Links + conteúdo	Sim	2^{15}	2^{-3}
Links + conteúdo	Não	2^{14}	2^{-4}
Links + *links* transformados	Sim	2^{14}	2^{-5}
Links + *links* transformados	Não	2^{15}	2^{-5}
Links transformados + conteúdo	Sim	2^{15}	2^{-2}
Links transformados + conteúdo	Não	2^{15}	2^{-7}
Links + *links* transformados + conteúdo	Sim	2^{15}	2^{-6}
Links + *links* transformados + conteúdo	Não	2^{15}	2^{-3}
Kernel RBF			
Links + conteúdo	Sim	2^{14}	2^{-4}
Links + conteúdo	Não	2^{15}	2^{-2}
Links + *links* transformados	Sim	2^{15}	2^{-3}
Links + *links* transformados	Não	2^{15}	2^{0}
Links transformados + conteúdo	Sim	2^{15}	2^{3}
Links transformados + conteúdo	Não	2^{15}	2^{3}
Links + *links* transformados + conteúdo	Sim	2^{15}	2^{-3}
Links + *links* transformados + conteúdo	Não	2^{15}	2^{-3}
Kernel sigmoidal			
Links + conteúdo	Sim	2^{9}	2^{-6}
Links + conteúdo	Não	2^{15}	2^{-7}
Links + *links* transformados	Sim	2^{13}	2^{-11}
Links + *links* transformados	Não	2^{13}	2^{-9}
Links transformados + conteúdo	Sim	2^{0}	2^{-10}
Links transformados + conteúdo	Não	2^{15}	2^{-11}
Links + *links* transformados + conteúdo	Sim	2^{10}	2^{-8}
Links + *links* transformados + conteúdo	Não	2^{10}	2^{-8}

Tabela A.3: Parâmetros obtidos por *grid search* e usados no método SVM para a classificação das amostras de *Web spam* representadas por atributos extraídos do conteúdo, dos *links* e dos *links* transformados (base de dados WEBSPAM-UK2007).

Tipos de vetores de características	Classes balanceadas?	C	γ
Kernel linear			
Conteúdo	Sim	2^{15}	2^{3}
Links	Sim	2^{15}	2^{3}
Links transformados	Sim	2^{10}	2^{3}
Kernel polinomial			
Conteúdo	Sim	2^{14}	2^{0}
Links	Sim	2^{15}	2^{-3}
Links transformados	Sim	2^{14}	2^{-6}
Kernel RBF			
Conteúdo	Sim	2^{10}	2^{3}
Links	Sim	2^{11}	2^{3}
Links transformados	Sim	2^{7}	2^{-1}
Kernel sigmoidal			
Conteúdo	Sim	2^{15}	2^{-3}
Links	Sim	2^{-2}	2^{-2}
Links transformados	Sim	2^{15}	2^{-12}

Tabela A.4: Parâmetros obtidos por *grid search* e usados no método SVM para a classificação das amostras de *Web spam* representadas por todas as combinações entre os atributos extraídos do conteúdo, dos *links* e dos *links* transformados (base de dados WEBSPAM-UK2007).

Tipos de vetores de características	Classes balanceadas?	C	γ
Kernel linear			
Links + conteúdo	Sim	2^{15}	2^{3}
Links + *links* transformados	Sim	2^{15}	2^{3}
Links transformados + conteúdo	Sim	2^{5}	2^{3}
Links + *links* transformados + conteúdo	Sim	2^{14}	2^{3}
Kernel polinomial			
Links + conteúdo	Sim	2^{13}	2^{-5}
Links + *links* transformados	Sim	2^{14}	2^{-5}
Links transformados + conteúdo	Sim	2^{14}	2^{-1}
Links + *links* transformados + conteúdo	Sim	2^{15}	2^{0}
Kernel RBF			
Links + conteúdo	Sim	2^{15}	2^{1}
Links + *links* transformados	Sim	2^{12}	2^{2}
Links transformados + conteúdo	Sim	2^{10}	2^{3}
Links + *links* transformados + conteúdo	Sim	2^{12}	2^{1}
Kernel sigmoidal			
Links + conteúdo	Sim	2^{5}	2^{-8}
Links + *links* transformados	Sim	2^{13}	2^{3}
Links transformados + conteúdo	Sim	2^{15}	2^{-2}
Links + *links* transformados + conteúdo	Sim	2^{5}	2^{0}

A.2 Resultados obtidos pelo método máquinas de vetores de suporte

Tabela A.5: Resultados obtidos pelo método SVM na classificação das amostras de *Web spam* representadas por atributos extraídos do conteúdo, dos *links* e dos *links* transformados (base de dados WEBSPAM-UK2006) usando classes desbalanceadas.

	Conteúdo	*Links*	*Links* transformados
SVM com *kernel* linear			
	Média	Média	Média
Acurácia	67,6 ± 19,7	66,9 ± 16,2	87,5 ± 0,7
Sensitividade	31,7 ± 30,3	39,2 ± 29,8	73,7 ± 1,0
Especificidade	78,4 ± 34,7	75,3 ± 28,9	91,7 ± 0,9
Precisão	50,9 ± 24,4	38,1 ± 12,1	73,0 ± 2,1
F-medida	0,285 ± 0,066	0,313 ± 0,159	0,733 ± 0,012
SVM com *kernel* polinomial			
Acurácia	65,9 ± 24,8	66,9 ± 16,2	88,2 ± 0,9
Sensitividade	47,9 ± 28,7	39,2 ± 29,8	77,1 ± 1,7
Especificidade	71,4 ± 41,0	75,3 ± 28,9	91,6 ± 0,8
Precisão	59,3 ± 24,8	38,1 ± 12,1	73,5 ± 2,1
F-medida	0,412 ± 0,034	0,313 ± 0,159	0,752 ± 0,018
SVM com *kernel* RBF			
Acurácia	82,5 ± 0,6	81,2 ± 0,7	88,3 ± 0,6
Sensitividade	36,0 ± 2,4	47,1 ± 2,5	76,6 ± 2,3
Especificidade	96,6 ± 0,5	91,5 ± 0,6	91,8 ± 0,8
Precisão	76,2 ± 2,8	62,8 ± 1,9	73,9 ± 1,6
F-medida	0,488 ± 0,023	0,538 ± 0,021	0,752 ± 0,012
SVM com *kernel* sigmoidal			
Acurácia	77,8 ± 0,7	67,4 ± 1,6	87,2 ± 0,8
Sensitividade	19,7 ± 13	30,2 ± 3,1	72,7 ± 3,0
Especificidade	95,5 ± 3,5	78,7 ± 1,4	91,6 ± 0,7
Precisão	58,9 ± 5,2	30,1 ± 3,2	72,5 ± 1,8
F-medida	0,271 ± 0,135	0,301 ± 0,031	0,726 ± 0,019

Tabela A.6: Resultados obtidos pelo método SVM na classificação das amostras de *Web spam* representadas por atributos extraídos do conteúdo, dos *links* e dos *links* transformados (base de dados WEBSPAM-UK2006) usando classes balanceadas.

	Conteúdo	*Links*	*Links* transformados
SVM com *kernel* linear			
	Média	Média	Média
Acurácia	62,5 ± 7,1	69,8 ± 3,1	86,3 ± 1,5
Sensitividade	42,7 ± 19,9	80,2 ± 4,8	87,3 ± 1,2
Especificidade	82,3 ± 25,9	59,3 ± 10,4	85,3 ± 2,3
Precisão	75,0 ± 13,3	66,9 ± 4,6	85,6 ± 2,0
F-medida	0,514 ± 0,129	0,726 ± 0,012	0,864 ± 0,014
SVM com *kernel* polinomial			
Acurácia	66,2 ± 1,2	69,8 ± 3,1	87,0 ± 1,1
Sensitividade	40,7 ± 2,4	80,2 ± 4,8	88,7 ± 1,4
Especificidade	91,6 ± 1,4	59,3 ± 10,4	85,2 ± 1,2
Precisão	83,0 ± 2,3	66,9 ± 4,6	85,7 ± 1,1
F-medida	0,546 ± 0,023	0,726 ± 0,012	0,872 ± 0,011
SVM com *kernel* RBF			
Acurácia	71,1 ± 1,4	76,3 ± 1,4	86,0 ± 0,7
Sensitividade	60,6 ± 2,7	77,1 ± 2,2	88,5 ± 1,7
Especificidade	81,6 ± 1,5	75,4 ± 2,4	83,5 ± 1,8
Precisão	76,7 ± 1,5	75,8 ± 1,7	84,4 ± 1,3
F-medida	0,677 ± 0,019	0,765 ± 0,014	0,864 ± 0,007
SVM com *kernel* sigmoidal			
Acurácia	54,7 ± 0,7	54,3 ± 1,8	86,3 ± 1,8
Sensitividade	15,0 ± 2,9	53,9 ± 2,9	86,8 ± 2,3
Especificidade	94,6 ± 3,3	54,8 ± 3,4	85,7 ± 1,9
Precisão	75,5 ± 9,0	54,4 ± 1,8	85,9 ± 1,8
F-medida	0,247 ± 0,037	0,541 ± 0,020	0,863 ± 0,019

Tabela A.7: Resultados obtidos pelo método SVM na classificação das amostras de *Web spam* representadas por todas as combinações entre os atributos extraídos do conteúdo, dos *links* e dos *links* transformados (base de dados WEBSPAM-UK2006) usando classes desbalanceadas.

	Conteúdo + *links*	*Links* + *links* transformados	*Links* transformados + conteúdo	*Links* + *links* transformados + conteúdo
SVM com *kernel* linear				
	Média	Média	Média	Média
Acurácia	78,9 ± 0,6	79,3 ± 1,1	75,7 ± 15,9	81,8 ± 0,8
Sensitividade	18,6 ± 2,3	25,6 ± 8,9	33,2 ± 22,0	37,4 ± 7,3
Especificidade	97,2 ± 0,6	95,6 ± 1,5	88,5 ± 27,2	95,3 ± 2,2
Precisão	67,1 ± 4,7	63,8 ± 2,9	69,0 ± 16,0	71,5 ± 4,4
F-medida	0,291 ± 0,030	0,358 ± 0,083	0,387 ± 0,044	0,486 ± 0,051
SVM com *kernel* polinomial				
Acurácia	76,8 ± 6,9	81,1 ± 3,0	82,4 ± 1,9	83,2 ± 1,0
Sensitividade	56,9 ± 12,7	59,4 ± 18,7	35,0 ± 7,7	52,4 ± 10,9
Especificidade	82,9 ± 12,1	87,8 ± 7,5	96,7 ± 0,6	92,6 ± 3,5
Precisão	55,1 ± 12,0	61,8 ± 7,1	75,7 ± 7,9	69,4 ± 5,3
F-medida	0,535 ± 0,049	0,583 ± 0,089	0,476 ± 0,092	0,587 ± 0,053
SVM com *kernel* RBF				
Acurácia	83,6 ± 0,8	83,6 ± 1,0	85,1 ± 0,6	84,4 ± 0,7
Sensitividade	52,4 ± 2,0	62,5 ± 2,1	50,7 ± 2,0	55,9 ± 2,0
Especificidade	93,0 ± 0,8	90,0 ± 1,3	95,6 ± 0,5	93,1 ± 0,5
Precisão	69,6 ± 2,5	65,6 ± 2,8	77,7 ± 2,2	71,1 ± 1,9
F-medida	0,598 ± 0,018	0,639 ± 0,018	0,613 ± 0,018	0,626 ± 0,018
SVM com *kernel* sigmoidal				
Acurácia	68,0 ± 0,7	71,1 ± 1,4	76,4 ± 1,6	72,4 ± 3,0
Sensitividade	31,0 ± 1,5	33,0 ± 3,8	36,8 ± 3,4	22,8 ± 8,9
Especificidade	79,2 ± 1,0	82,7 ± 2,5	88,4 ± 1,5	87,4 ± 6,5
Precisão	31,2 ± 1,2	36,7 ± 2,4	49,2 ± 4,1	36,8 ± 4,0
F-medida	0,311 ± 0,012	0,346 ± 0,024	0,420 ± 0,035	0,269 ± 0,055

Tabela A.8: Resultados obtidos pelo método SVM na classificação das amostras de *Web spam* representadas por todas as combinações entre os atributos extraídos do conteúdo, dos *links* e dos *links* transformados (base de dados WEBSPAM-UK2006) usando classes balanceadas.

	Conteúdo + *links*	*Links* + *links* transformados	*Links* transformados + conteúdo	*Links* + *links* transformados + conteúdo
		SVM com *kernel* linear		
	Média	Média	Média	Média
Acurácia	70,5 ± 3,4	79,0 ± 1,5	63,0 ± 3,5	76,4 ± 3,0
Sensitividade	68,3 ± 6,1	82,9 ± 2,0	44,4 ± 23,5	73,4 ± 6,7
Especificidade	75,1 ± 3,2	81,6 ± 27,5	79,4 ± 11,8	
Precisão	72,2 ± 5,8	77,0 ± 2,1	78,8 ± 11,8	79,1 ± 5,9
F-medida	0,698 ± 0,028	0,798 ± 0,012	0,522 ± 0,092	0,757 ± 0,018
		SVM com *kernel* polinomial		
Acurácia	71,6 ± 2,7	81,3 ± 1,5	74,6 ± 1,4	79,9 ± 0,9
Sensitividade	69,5 ± 14,9	81,7 ± 3,4	59,4 ± 3,3	75,8 ± 2,2
Especificidade	73,6 ± 15,6	80,9 ± 3,0	89,8 ± 1,2	84,0 ± 1,4
Precisão	74,7 ± 8,0	81,2 ± 2,1	85,4 ± 1,2	82,6 ± 1,0
F-medida	0,704 ± 0,053	0,814 ± 0,017	0,700 ± 0,023	0,790 ± 0,011
		SVM com *kernel* RBF		
Acurácia	76,0 ± 1,5	80,9 ± 1,0	77,9 ± 1,1	80,4 ± 1,1
Sensitividade	71,4 ± 1,7	84,0 ± 1,7	66,5 ± 2,4	78,2 ± 1,4
Especificidade	80,6 ± 2,6	77,7 ± 1,2	89,2 ± 1,7	82,6 ± 1,9
Precisão	78,7 ± 2,3	79,0 ± 0,9	86,1 ± 1,7	81,9 ± 1,5
F-medida	0,749 ± 0,015	0,814 ± 0,010	0,750 ± 0,015	0,800 ± 0,011
		SVM com *kernel* sigmoidal		
Acurácia	54,3 ± 1,8	64,3 ± 1,8	50,8 ± 1,1	62,1 ± 5,1
Sensitividade	53,9 ± 2,9	81,3 ± 4,5	69,2 ± 42,4	62,9 ± 5,8
Especificidade	54,8 ± 3,4	47,3 ± 6,8	32,3 ± 41,7	61,4 ± 5,5
Precisão	54,4 ± 1,8	60,8 ± 2,0	50,9 ± 3,4	62,0 ± 5,1
F-medida	0,541 ± 0,020	0,695 ± 0,012	0,503 ± 0,254	0,624 ± 0,052

Tabela A.9: Resultados obtidos pelo método SVM na classificação das amostras de *Web spam* representadas por atributos extraídos do conteúdo, dos *links* e dos *links* transformados (base de dados WEBSPAM-UK2007) usando classes balanceadas.

	Conteúdo	*Links*	*Links* transformados
		SVM com *kernel* linear	
	Média	Média	Média
Acurácia	60,4 ± 9,9	52,5 ± 2,8	67,3 ± 1,7
Sensitividade	56,1 ± 20,7	80,9 ± 14,1	61,1 ± 3,6
Especificidade	64,7 ± 38,9	24,2 ± 18,8	73,5 ± 2,1
Precisão	70,1 ± 15,5	52,1 ± 2,7	69,8 ± 1,7
F-medida	0,578 ± 0,049	0,626 ± 0,037	0,651 ± 0,025
		SVM com *kernel* polinomial	
Acurácia	68,2 ± 2,5	67,8 ± 2,4	79,0 ± 1,1
Sensitividade	46,4 ± 7,2	66,5 ± 15,8	78,0 ± 2,1
Especificidade	90,1 ± 4,9	69,2 ± 17,8	80,1 ± 2,1
Precisão	82,9 ± 5,2	70,9 ± 8,8	79,7 ± 1,6
F-medida	0,591 ± 0,050	0,667 ± 0,055	0,788 ± 0,012
		SVM com *kernel* RBF	
Acurácia	70,6 ± 0,9	78,6 ± 1,4	85,1 ± 1,2
Sensitividade	49,8 ± 2,0	72,1 ± 2,4	85,2 ± 2,1
Especificidade	91,5 ± 1,1	85,1 ± 2,2	85,0 ± 1,8
Precisão	85,4 ± 1,5	82,9 ± 2,0	85,1 ± 1,5
F-medida	0,628 ± 0,016	0,771 ± 0,017	0,851 ± 0,012
		SVM com *kernel* sigmoidal	
Acurácia	60,3 ± 5,7	48,6 ± 4,4	67,6 ± 1,6
Sensitividade	50,3 ± 26,4	66,2 ± 39,0	62,7 ± 3,3
Especificidade	70,2 ± 37,0	30,9 ± 46,8	72,6 ± 1,8
Precisão	70,5 ± 11,1	60,1 ± 19,7	69,6 ± 1,5
F-medida	0,537 ± 0,076	0,492 ± 0,217	0,659 ± 0,023

Tabela A.10: Resultados obtidos pelo método SVM na classificação das amostras de *Web spam* representadas por todas as combinações entre os atributos extraídos do conteúdo, dos *links* e dos *links* transformados (base de dados WEBSPAM-UK2007) usando classes balanceadas.

	Conteúdo + *links*	*Links* + *links* transformados	*Links* transformados + conteúdo	*Links* + *links* transformados + conteúdo
		SVM com *kernel* linear		
	Média	Média	Média	Média
Acurácia	62,9 ± 2,0	54,7 ± 4,0	66,7 ± 0,8	64,3 ± 3,3
Sensitividade	57,2 ± 3,3	65,9 ± 28,7	44,1 ± 1,5	62,1 ± 12,7
Especificidade	68,6 ± 4,8	43,4 ± 35,5	89,4 ± 1,4	66,4 ± 17,6
Precisão	64,7 ± 2,8	58,1 ± 9,6	80,7 ± 1,9	66,6 ± 6,5
F-medida	0,607 ± 0,020	0,565 ± 0,110	0,570 ± 0,013	0,631 ± 0,038
		SVM com *kernel* polinomial		
Acurácia	72,6 ± 2,3	73,4 ± 3,4	65,7 ± 4,0	75,0 ± 1,7
Sensitividade	60,5 ± 2,5	67,1 ± 8,3	47,4 ± 14,2	67,8 ± 5,1
Especificidade	84,7 ± 4,4	79,7 ± 11,2	84,0 ± 21,5	82,1 ± 4,1
Precisão	80,1 ± 4,6	78,0 ± 7,1	79,4 ± 9,7	79,4 ± 3,0
F-medida	0,688 ± 0,022	0,715 ± 0,037	0,574 ± 0,039	0,730 ± 0,025
		SVM com *kernel* RBF		
Acurácia	78,2 ± 1,6	80,0 ± 1,1	72,5 ± 1,1	79,8 ± 1,5
Sensitividade	72,6 ± 2,6	75,7 ± 4,2	60,7 ± 3,0	73,0 ± 4,5
Especificidade	83,6 ± 1,6	84,4 ± 4,2	84,3 ± 2,3	86,6 ± 2,8
Precisão	81,6 ± 1,6	83,1 ± 3,3	79,5 ± 2,0	84,6 ± 2,1
F-medida	0,769 ± 0,019	0,791 ± 0,015	0,688 ± 0,018	0,783 ± 0,021
		SVM com *kernel* sigmoidal		
Acurácia	62,4 ± 1,0	50,0 ± 0,3	48,5 ± 6,3	49,6 ± 2,3
Sensitividade	59,9 ± 1,5	99,5 ± 0,5	48,4 ± 5,3	61,4 ± 13,2
Especificidade	64,9 ± 2,8	0,5 ± 0,3	48,6 ± 7,4	37,7 ± 13,4
Precisão	63,1 ± 1,4	50,0 ± 0,2	48,6 ± 6,4	49,6 ± 1,9
F-medida	0,614 ± 0,008	0,666 ± 0,003	0,485 ± 0,058	0,544 ± 0,054