Avaliação de Desempenho de Métodos de Classificação Aplicados na Identificação de *Spam Hosts*

Renato Moraes Silva[1], **Alex Garcia Vaz**[2], **Tiago A. Almeida**[2], **Akebo Yamakami**[1]

[1]Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (UNICAMP) – Campinas – SP – Brasil

[2]Departamento de Computação (DComp)
Universidade Federal de São Carlos (UFSCar) – Sorocaba – SP – Brasil

{renatoms,akebo}@dt.fee.unicamp.br, {alexvaz,talmeida}@ufscar.br

Abstract. *The web is becoming an increasingly important tool and a huge source of entertainment, communication, research, news and trade. Consequently, the web sites compete to attract the attention of users and many of them achieve a good visibility through strategies that 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 paper presents a performance analysis of established machine learning methods employed to automatically detect and filter hosts that disseminate web spam. Through a statistical validation of the results, we have observed 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 concluded that the machine learning methods performs better when we combine the content-based features with the transformed link-based features.*

Resumo. *A web vem se tornando cada vez mais importante para seus usuários, tanto como fonte de diversão, comunicação, pesquisa, notícias e comércio. Consequentemente, os sites concorrem entre si para atrair a atenção dos usuários, sendo que muitos ganham maior visibilidade através de estratégias que enganam os motores de busca. Esses sites, conhecidos como web spam, causam prejuízos pessoais e econômicos aos usuários. Diante desse cenário, este trabalho apresenta uma análise de desempenho de diversos métodos de aprendizagem de máquina aplicadas na detecção automática de web hosts que propagam web spam. Por meio de uma validação estatística dos resultados observou-se 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 spam hosts. Os resultados também indicam que os métodos de aprendizagem de máquina tem melhor desempenho quando é feita a combinação dos vetores de atributos extraídos do conteúdo das páginas web com os vetores formados pela transformação dos atributos extraídos da relação de links.*

1. Introdução

A maioria dos usuários da Internet utilizam com frequência alguma ferramenta de busca para auxiliar sua navegação no emaranhado de sites e informações disponíveis na rede.

Logo, os motores de busca são responsáveis por uma porcentagem expressiva das visitas recebidas pelos *web sites*. Então, para ter sucesso é importante que o *web site* mantenha um alto *ranking* de relevância nos motores de busca para aparecer entre os primeiros resultados quando o usuário realizar uma busca com termos relacionados aos seus serviços oferecidos ou assuntos abordados em suas páginas. Para atingir esse propósito, é aconselhável utilizar 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. Algumas estratégias básicas de SEO incluem oferecer 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, conhecidas como *web spamming* ou *spamdexing* [Gyongyi e Garcia-Molina 2005], que enganam os motores de busca para ganhar relevância. Essas estratégias prejudicam os usuários que ao fazerem suas consultas recebem respostas inesperadas, irrelevantes e muitas vezes infectadas por conteúdos maliciosos e prejudiciais, como *malwares* ou outras pragas virtuais. Segundo [Gyongyi e Garcia-Molina 2005], os *sites* que usam técnicas de *web spamming* e que são conhecidos como *web spam*, podem possuir tanto conteúdo *spam* quanto *spam links*. O primeiro consiste em criar páginas com milhares de palavras-chaves irrelevantes e o segundo consiste em adicionar *links* que apontam para as páginas que pretende-se promover.

Estudos recentes apontam que o volume de *web spam* vem aumentando consideravelmente nos últimos anos. Segundo [Ntoulas et al. 2006], cerca de 13,8% da páginas de língua inglesa, 25% das francesas e 22% das germânicas são *spams*. Em outra pesquisa, Provos *et al.* [Provos et al. 2008] fizeram uma análise de bilhões de URLs entre janeiro e outubro de 2007 e constataram que cerca de 3 milhões de URLs foram consideradas maliciosas, pois tentam instalar e executar *malwares* automaticamente no computador dos usuários. Além disso, eles concluiram 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% das primeiras 1 milhão de URLs mais frequentes nos resultados de busca do Google levam a conteúdos maliciosos.

Em outro estudo, [John et al. 2011] observaram que 36% dos resultados dos motores de busca Google e Bing contém URLs maliciosas. Ainda, [Lu et al. 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[1] mostra que 22,4% dos resultados de busca sobre entretenimento levam a *links* maliciosos. Além disso, segundo um relatório publicado pela empresa McAfee[2], 49% dos termos de busca mais populares retornam algum *site* malicioso entre os 100 primeiros resultados da busca. A mesma pesquisa aponta que 1,2% das consultas retornam *links* de *sites* maliciosos entre os 100 primeiros resultados.

Diante desse cenário, este trabalho apresenta uma análise de desempenho de diversos métodos bem conhecidos de aprendizado de máquina (*machine lear-*

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

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

ning) [Mitchell 1997] que podem ser aplicadas para auxiliar no combate desse problema. O objetivo é encontrar métodos promissores que podem ser explorados e empregados para auxiliar na detecção automática de *spam hosts*. 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* e *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*. Resultados preliminares foram publicados em [Silva et al. 2012e, Silva et al. 2012d, Silva et al. 2012f, Silva et al. 2012b, Silva et al. 2012a, Silva et al. 2012c]. Contudo, neste artigo são oferecidas as seguintes contribuições:

- criação de vetores de atributos formados por combinações de diferentes tipos de atributos que podem ser extraídos das páginas *web*;
- experimentos que ilustram o impacto no desempenho dos métodos de aprendizagem devido ao balanceamento dos dados de treinamento;
- análise estatística dos resultados a fim de determinar os melhores métodos para o problema de classificação de *web spam*.
- experimentos que analisam a viabilidade do emprego de redução de dimensionalidade por meio do método de análise de componentes principais;
- comparação dos resultados apresentados nesse trabalho com outros resultados da literatura.

Este artigo está estruturado da seguinte forma: na Seção 2 são apresentados os conceitos básicos sobre os métodos classificadores avaliados neste trabalho. Na Seção 3 são descritas as configurações adotadas nos experimentos e a base de dados empregada. Os resultados experimentais são apresentados na Seção 4. Por fim, conclusões e direções para trabalhos futuros são descritos na Seção 5.

2. Classificadores

Apesar da existência de trabalhos cujo intuito é filtrar *web spam*, ainda não há um consenso se a aplicação de métodos de aprendizado de máquina é realmente eficaz na detecção automática de *spam hosts*. Tendo isso em vista, foram avaliados neste trabalho diversos métodos de classificação bem conhecidos.

Nessa seção, são apresentados os métodos classificadores avaliados nesse trabalho: redes neurais artificiais (RNAs) perceptron de múltiplas camadas (MLP – *multilayer perceptron*), máquina 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*. A escolha de tais métodos reside no fato de terem sido avaliados e listados como os melhores métodos de mineração de dados atualmente disponíveis [Wu et al. 2008]. As RNAs não fazem parte dessa lista, mas foram escolhidas para serem avaliadas devido à sua alta capacidade de generalização. O método floresta aleatória também foi escolhido, mesmo não estando na lista, pois ele faz uma combinação de árvores de decisão. Logo, como o método C4.5 é um algoritmo de árvore de decisão e está na lista dos melhores métodos, acredita-se que a combinação de árvores de decisão também possa gerar bons resultados.

2.1. 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. Esse algoritmo pode ser resumido em duas etapas: *forward* e *backward* [Bishop 1995].

Na etapa *forward*, o sinal é propagado pela rede, camada a camada, da seguinte forma: $u_j^l(n) = \sum_{i=0}^{m^{l-1}} w_{ji}^l(n) y_i^{l-1}(n)$, sendo $l = 0, 1, 2, ..., L$ o índice das camadas da rede. Quando $l = 0$, ele representa a camada de entrada e quando $l = L$ representa a camada de saída. Já, $y_i^{l-1}(n)$ é a função de saída do neurônio i na camada anterior $l-1$, $w_{ji}^l(n)$ é o peso sináptico do neurônio j na camada l e m^l é a quantidade de neurônios na camada l. Para $i = 0$, $y_0^{l-1}(n) = +1$ e $w_{j0}^l(n)$ representa o bias aplicado ao neurônio j da camada l. A saída do neurônio é dada por: $y_j^l(n) = \varphi_j(u_j^l(n))$, onde φ_j é a função de ativação do neurônio j. O erro pode ser calculado por: $e_j^l(n) = y_j^l(n) - d(n)$, sendo que $d(n)$ é a saída desejada para o padrão de entrada $x(n)$.

Na etapa *backward*, inicia-se a derivação do algoritmo *backpropagation*, a partir da camada de saída, onde tem-se: $\delta_j^L(n) = \varphi_j'(u_j^L(n)) e_j^L(n)$, sendo φ_j' a derivada da função de ativação. Para $l = L, L-1, ..., 2$, calcula-se: $\delta_j^{l-1}(n) = \varphi_j'(u_j^{l-1}(n)) \sum_{i=1}^{m^l} w_{ji}^l(n) * \delta_j^l(n)$, para $j = 0, 1, ..., m^l - 1$.

Para maiores detalhes sobre as MLPs, consulte [Bishop 1995, Haykin 1998].

2.1.1. Algoritmo de Levenberg-Marquardt

O algoritmo de Levenberg-Marquardt é um método de otimização e aceleração da convergência do algoritmo *backpropagation*. Ele é considerado um método de segunda ordem, assim como os métodos do gradiente conjugado e do método quase-Newton, pois utiliza informações sobre a derivada segunda da função de erro [Bishop 1995].

Considerando que a função de erro usada na rede MLP é dada pelo erro quadrático médio (EQM) [Haykin 1998], a equação usada no método de Gauss-Newton para atualização dos pesos e consequente minimização do EQM é $W_{i+1} = W_1 - H^{-1} \nabla f(W)$, onde o gradiente $\nabla f(W)$ pode ser representado por $\nabla f(W) = J^T e$ e a matriz Hessiana H pode ser calculada por $\nabla^2 f(W) = J^T J + S$, sendo J uma matriz Jacobiana e $S = \sum_{i=1}^{n} e_i \nabla^2 e_i$. Supondo que S é um valor pequeno se comparado ao produto de J, a Hessiana pode ser representada por $\nabla^2 f(W) \approx J^T J$. Então, a atualização dos pesos no método de Gauss-Newton pode ser expressada por $W_{i+1} = W_1 - (J^T J)^{-1} J^T e$.

Como a matriz Hessiana simplificada não pode ser invertida, o algoritmo de Levenberg-Marquardt atualiza os pesos usando a equação $W_{i+1} = W_i - (J^T J + \mu I)^{-1} J^T e$, sendo que I é a matriz identidade e μ um parâmetro que torna a matriz Hessiana definida positiva.

Maiores detalhes sobre o algoritmo de Levenberg-Marquardt podem ser encontrados em [Bishop 1995, Hagan e Menhaj 1994].

2.2. Máquinas de vetores de suporte

Máquinas de vetores de suporte (SVM – *support vector machines*) [Cortes e Vapnik 1995] é um método de aprendizagem de máquina que pode ser usado para problemas de classificação e regressão além de outras tarefas de aprendizagem [Haykin 1998, Chang e 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 et al. 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 é um parâmetro de deslocamento, que 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, o método recomendado em [Hsu et al. 2003] é a *grid search* (busca em grade). Esse método 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 com valores 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.

2.4. IBK

O algoritmo IBK é um algoritmo de aprendizagem baseado em instâncias (IBL – *instance-based learning*) [Aha et al. 1991]. Esse tipo de algoritmo é derivado do método de classificação k-vizinhos mais próximos (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].

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 et al. 2011].

2.5. Métodos de agregação de classificadores

Métodos de agregação ou combinação de múltiplos classificadores, também conhecidos com métodos *ensemble*, tem sido cada vez mais usados na literatura para aumentar a eficiência do processo de aprendizagem 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. Alguns dessas estratégias incluem a manipulação dos conjuntos de treinamento, dos atributos extraídos das amostras do problema e dos algoritmos de aprendizagem 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. *Bagging*

O *bagging* (abreviação de *bootstrap aggregating*) [Breiman 1996] é um método de geração 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], diferente do que ocorre no segundo método, no *bagging*, os diferentes modelos de classificadores recebem o mesmo peso na geração de uma predição.

No método de *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].

2.5.2. Métodos de *boosting*

Boosting adaptativo

O método de *boosting* adaptativo (*AdaBoost – adaptive boosting*) [Freund e Schapire 1996] 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. Porém, segundo [Freund e Schapire 1996], ele possui algumas propriedades que o tornam mais prático e fácil de ser implementado do que os algoritmos de *boosting* que o antecederam, pois ele não necessita de nenhum conhecimento prévio das predições obtidas por classificadores ruins. Em vez disso ele se adapta as predições ruins e gera uma hipótese majoritária ponderada, em que o peso das predições fornecidas pelos classificadores ruins torna-se uma função de sua predição.

LogitBoost

O método *LogitBoost* [Friedman et al. 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.

2.5.3. 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. Nesse método, depois que um determinado número de árvores são geradas, cada uma lança um voto para uma classe do problema, considerando um vetor de entrada. Então, a classe mais votada será escolhida na predição do classificador.

3. Base de dados e configurações

Os experimentos foram realizados com a base de dados pública *WEBSPAM-UK2006 collection*[3]. Ela é composta por 77,9 milhões de páginas *web* hospedadas em 11.000 *hosts*. Essa base foi utilizada no *Web Spam Challenge Track I* e *II*[4], que trata-se de uma competição de métodos de detecção de *web spam*.

Assim como empregado nos campeonatos, nos experimentos realizados foram utilizados 3 conjuntos de 8.487 vetores de atributos pré-computados. Cada conjunto é composto por 1.978 *hosts* rotulados como *spam* e 6.509 *hosts* rotulados como *ham* (não-spam). Essas informações foram extraídas da coleção de dados e fornecidas pelos organizadores do evento aos times participantes. O primeiro conjunto de vetores de características é composto por 96 características baseadas no conteúdo [Castillo et al. 2007], o segundo é composto por 41 características baseadas nos *links* [Becchetti et al. 2006] e o terceiro é composto por 138 características baseadas nos *links* transformados [Castillo et al. 2007], que são simples combinações ou operações logarítmicas sobre as características baseadas em *links*.

Para avaliar o desempenho de cada classificador foi usada uma validação por subamostragem aleatória, também conhecida como validação cruzada de Monte Carlo [Shao 1993]. Essa método de validação foi escolhido pois dá mais liberdade na

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

[4]*Web Spam Challenge:* `http://webspam.lip6.fr/`

seleção dos subconjuntos de treinamento e teste, já que diferente do método de validação k-*folds*, permite que sejam feitas quantas repetições foram desejadas, usando qualquer porcentagem de dados para treinamento e teste. Então, foram feitas 10 simulações com cada classificador e em cada uma, 80% dos dados foram usados para treinamento e 20% 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 dos classificadores foram utilizadas as seguintes medidas de desempenho amplamente empregadas na literatura [Witten et al. 2011]:

- *Sensitividade (recall):* proporção de padrões da classe positiva (*spam*) identificada corretamente. Indica o quão bom o classificador é para identificar a classe positiva. Essa medida de desempenho é definida por: $\frac{Vp}{Vp+Fn}$
- *Precisão (precision):* porcentagem de padrões classificados como pertencentes a classe positiva e que realmente pertencem a classe positiva. Essa medida de desempenho pode ser calculada por: $\frac{Vp}{Vp+Fp}$
- *F-medida:* média harmônica entre precisão e sensitividade. Essa medida de desempenho é calculada da seguinte forma: $2 * \frac{\text{Precisão}*\text{Sensitividade}}{\text{Precisão}+\text{Sensitividade}}$.

Nas fórmulas matemáticas das medidas de desempenho apresentadas acima, Vp (verdadeiros positivos) refere-se ao número de exemplos corretamente classificados como *spam*, Fp (falsos positivos) refere-se aos exemplos que foram incorretamente classificados como *spam* e Fn (falsos negativos) refere-se ao número de exemplos classificados incorretamente como *ham*.

3.1. Configurações

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

3.1.1. Redes neurais artificiais

Neste trabalho, foram avaliadas as seguintes RNAs: MLP treinada com o algoritmo *backpropagation* e método do gradiente descendente (MLP-GD) e MLP treinada com o método de Levenberg-Marquardt (MLP-LM).

Todas as redes MLP implementadas usam uma única camada intermediária e possuem 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 equação $x = 2 \times \frac{x - x_{min}}{x_{max} - x_{min}} - 1$, sendo x a matriz com todos os vetores de atributos e x_{min} e x_{max} o menor e o maior valor da matriz x, respectivamente. Além disso, em todas as simulações com as RNAs 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 os seguintes:

- MLP treinada com o método do gradiente descendente:
  - Limite máximo de iterações $\theta = 10,000$

  - Limite mínimo do EQM $\gamma = 0.001$
  - Passo de aprendizagem $\alpha = 0.005$
  - Número de neurônios na camada intermediária: 100
- MLP treinada com o método de Levenberg-Marquardt:
  - Limite máximo de iterações $\theta = 500$
  - Limite mínimo do EQM $\gamma = 0.001$
  - Passo de aprendizagem $\alpha = 0.001$
  - Número de neurônios na camada intermediária: 50

3.1.2. Máquinas de vetores de suporte

O SVM foi implementado utilizando a biblioteca LIBSVM [Chang e Lin 2011] disponível para a ferramenta MATLAB. Foram feitas simulações com as funções de *kernel* linear, RBF, sigmoidal e polinomial. O método de *grid search* foi empregado para a definição dos parâmetros. Porém, nas SVMs 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, adotou-se os valores padrões da LIBSVM para os demais parâmetros.

A *grid search* foi realizada com um conjunto de treino (80% dos dados) e teste (20% dos dados), escolhidos aleatoriamente para cada configuração de classificação. Depois de executado, os melhores parâmetros foram escolhidos e usados para realizar os experimentos com o SVM. Nesses experimentos, todos os tipos de *kernel* foram avaliados. Porém, optou-se por apresentar apenas os resultados obtidos com o *kernel* RBF, uma vez que este obteve o melhor desempenho. A Tabela 1 apresenta os parâmetros usados nas simulações com o método SVM.

Tabela 1. Parâmetros usados no método SVM com *kernel* RBF, obtidos por *grid search*.

	Conteúdo		*Links*		*Links* transformados		*Links*+Conteúdo	
	C	γ	C	γ	C	γ	C	γ
Classes balanceadas	2^{14}	2^{3}	2^{15}	2^{-1}	2^{5}	2^{-5}	2^{14}	2^{-4}
Classes desbalanceadas	2^{15}	2^{3}	2^{15}	2^{-1}	2^{10}	2^{-9}	2^{15}	2^{-2}

3.1.3. Demais métodos

Os demais classificadores foram implementados usando a ferramenta WEKA [Hall et al. 2009]. 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 fornecidos como padrão da ferramenta utilizada.

4. Resultados

Nessa seção, são apresentados os resultados da detecção automática de *spam hosts* obtidos pelos métodos de aprendizado de máquina considerados os melhores atualmente disponíveis [Wu et al. 2008]. Para cada método e conjunto de vetores de características, foram feitas simulações usando classes desbalanceadas, conforme originalmente fornecido na competição de *web spam*, 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 utilizado o método de subamostragem aleatória (*random undersampling*) [He e Garcia 2009] por ser um dos métodos mais simples e mais usados na literatura em problemas de classificação em que há desbalanceamento entre as classes. Esse método remove aleatoriamente amostras da classe com maior número de representantes. Depois da aplicação desse método ambas as classes passaram a ter 1.978 representantes cada.

As Tabelas 2 a 8 apresentam os resultados obtidos por cada método de classificação explorando os atributos baseados no conteúdo, nos *links*, nos *links* transformados e todas as combinações entre os vetores de atributos. Os valores em negrito indicam os melhores resultados usando cada configuração de classes (balanceadas ou desbalanceadas). Os valores em negrito precedidos pelo símbolo "*" indicam os melhores resultados considerando todas as simulações.

Tabela 2. Resultados obtidos na detecção de *spam hosts* usando atributos extraídos do conteúdo das páginas.

	Classes desbalanceadas			Classes balanceadas		
	Sensitividade	Precisão	F-medida	Sensitividade	Precisão	F-medida
Bagging	68.7±2.3	**84.4**±1.9	**0.757**±0.014	83.5±2.3	86.1±1.3	0.848±0.013
AdaBoost	66.6±3.2	78.4±2.3	0.720±0.024	81.8±2.5	83.3±1.6	0.825±0.014
LogitBoost	54.2±3.9	71.6±3.2	0.616±0.023	77.0±3.2	78.9±1.1	0.779±0.015
MLP-LM	**69.3**±4.2	77.6±4.6	0.731±0.032	**86.5**±2.0	**89.1**±2.4	**0.877**±0.017
MLP-GD	57.0±4.6	77.5±2.7	0.656±0.039	82.9±2.0	86.1±2.4	0.845±0.015
C4.5	67.7±4.6	68.0±0.6	0.678±0.024	79.2±2.5	78.3±1.6	0.787±0.012
Floresta aleatória	65.7±4.4	83.4±3.7	0.734±0.024	81.8±3.3	83.4±2.3	0.825±0.015
IBK	64.6±1.3	72.8±2.0	0.685±0.011	77.0±2.6	81.4±1.5	0.791±0.017
SVM	36.0±2.4	76.2±2.8	0.488±0.023	60.6±2.7	76.7±1.5	0.677±0.019

Tabela 3. Resultados obtidos na detecção de *spam hosts* usando atributos extraídos da relação dos *links*.

	Classes desbalanceadas			Classes balanceadas		
	Sensitividade	Precisão	F-medida	Sensitividade	Precisão	F-medida
Bagging	**79.2**±1.6	**77.2**±2.0	**0.781**±0.009	91.7±1.9	84.9±1.4	**0.882**±0.013
AdaBoost	71.8±1.8	74.4±2.2	0.731±0.017	87.7±1.7	83.9±0.8	0.858±0.009
LogitBoost	71.8±4.0	71.4±2.1	0.715±0.009	88.0±3.7	80.4±1.0	0.840±0.018
MLP-LM	70.8±6.6	74.1±3.7	0.723±0.049	**92.1**±3.8	82.3±2.6	0.868±0.019
MLP-GD	61.6±3.1	75.3±2.9	0.677±0.026	90.2±2.6	80.0±2.9	0.848±0.019
C4.5	73.2±2.1	72.3±1.4	0.727±0.011	85.4±2.2	82.3±1.4	0.838±0.011
Floresta aleatória	76.5±7.9	77.0±3.7	0.764±0.027	88.8±2.5	**87.1**±1.6	0.879±0.010
IBK	67.8±2.8	64.2±2.4	0.659±0.023	80.8±1.4	77.1±0.8	0.789±0.003
SVM	47.1±2.5	62.8±1.9	0.538±0.021	77.1±2.2	75.8±1.7	0.765±0.014

Tabela 4. Resultados obtidos na detecção de *spam hosts* usando atributos extraídos da relação de *links* transformados.

	Classes desbalanceadas			Classes balanceadas		
	Sensitividade	Precisão	F-medida	Sensitividade	Precisão	F-medida
Bagging	**78.9**±1.2	**77.6**±2.1	**0.782**±0.014	**91.3**±1.6	**86.2**±1.1	**0.886**±0.011
AdaBoost	72.6±3.0	74.9±1.9	0.737±0.019	88.8±0.9	85.9±1.7	0.873±0.011
LogitBoost	70.2±4.9	72.3±3.2	0.711±0.019	87.6±3.7	82.7±1.0	0.850±0.021
MLP-LM	74.9±3.9	76.1±2.1	0.754±0.027	87.5±4.0	85.8±3.3	0.866±0.027
MLP-GD	75.2±2.2	73.9±3.1	0.745±0.014	89.6±2.2	85.9±2.0	0.877±0.019
C4.5	73.5±1.8	68.5±2.2	0.709±0.016	85.6±2.4	83.1±1.8	0.844±0.020
Floresta aleatória	76.5±3.1	75.8±4.1	0.760±0.012	89.5±3.7	86.0±1.7	0.876±0.016
IBK	67.5±2.4	67.3±3.5	0.673±0.011	81.6±0.7	80.2±0.4	0.809±0.002
SVM	76.6±2.3	73.9±1.6	0.752±0.012	88.5±1.7	84.4±1.3	0.864±0.007

Tabela 5. Resultados obtidos na detecção de *spam hosts* usando combinação dos atributos extraídos do conteúdo das páginas e da relação de *links*.

	Classes desbalanceadas			Classes balanceadas		
	Sensitividade	Precisão	F-medida	Sensitividade	Precisão	F-medida
Bagging	**81.1**±1.8	84.0±1.7	0.825±0.016	**93.2**±0.8	87.8±1.0	0.904±0.005
AdaBoost	80.4±1.9	***86.3**±1.2	***0.832**±0.014	92.6±1.4	**88.5**±0.9	**0.905**±0.005
LogitBoost	70.2±2.6	77.7±2.3	0.737±0.019	89.9±1.3	84.3±1.2	0.870±0.005
MLP-LM	81.7±2.0	84.4±1.9	0.830±0.008	91.9±3.5	85.6±2.7	0.886±0.021
MLP-GD	74.2±3.2	79.5±2.5	0.767±0.016	92.6±2.4	86.7±2.1	0.895±0.013
C4.5	75.9±2.3	73.8±0.9	0.748±0.014	86.9±2.2	85.1±0.9	0.860±0.009
Floresta aleatória	74.4±2.5	85.0±2.4	0.793±0.006	92.1±2.1	86.9±1.2	0.894±0.008
IBK	68.7±2.3	72.5±2.0	0.705±0.015	84.7±2.7	82.4±1.0	0.835±0.012
SVM	52.4±2.0	69.6±2.5	0.598±0.018	71.4±1.7	78.7±2.3	0.749±0.015

Tabela 6. Resultados obtidos na detecção de *spam hosts* usando combinação dos atributos extraídos da relação de *links* e de *links* transformados.

	Classes desbalanceadas			Classes balanceadas		
	Sensitividade	Precisão	F-medida	Sensitividade	Precisão	F-medida
Bagging	78.5 ± 3.0	**77.6** ± 1.5	**0.780** ± 0.020	91.6 ± 1.3	**85.8** ± 1.6	**0.886** ± 0.011
AdaBoost	73.9 ± 2.2	75.3 ± 1.3	0.746 ± 0.015	89.7 ± 0.9	85.0 ± 1.2	0.873 ± 0.008
LogitBoost	69.9 ± 4.5	71.4 ± 2.7	0.705 ± 0.018	90.3 ± 1.3	83.0 ± 1.5	0.864 ± 0.006
MLP-LM	75.8 ± 4.7	76.6 ± 4.1	0.761 ± 0.038	87.7 ± 2.9	85.2 ± 3.5	0.863 ± 0.026
MLP-GD	76.9 ± 3.8	76.1 ± 4.5	0.764 ± 0.031	91.7 ± 1.6	84.6 ± 3.4	0.880 ± 0.023
C4.5	71.8 ± 3.3	70.6 ± 2.2	0.712 ± 0.019	84.7 ± 1.5	83.8 ± 1.3	0.842 ± 0.011
Floresta Aleatória	**78.7** ± 2.2	75.4 ± 2.3	0.770 ± 0.020	**92.1** ± 0.6	84.7 ± 1.1	0.882 ± 0.005
IBK	67.2 ± 2.0	66.0 ± 1.8	0.666 ± 0.017	83.2 ± 1.6	79.1 ± 1.4	0.811 ± 0.014
SVM	62.5 ± 2.1	65.6 ± 2.8	0.639 ± 0.018	84.0 ± 1.7	79.0 ± 0.9	0.814 ± 0.010

Tabela 7. Resultados obtidos na detecção de *spam hosts* usando combinação dos atributos extraídos do conteúdo das páginas e da relação de *links* transformados.

	Classes desbalanceadas			Classes balanceadas		
	Sensitividade	Precisão	F-medida	Sensitividade	Precisão	F-medida
Bagging	**82.2** ± 2.4	83.6 ± 1.4	0.829 ± 0.017	92.7 ± 1.1	88.0 ± 1.0	0.903 ± 0.008
AdaBoost	81.1 ± 2.7	**85.6** ± 1.6	***0.832** ± 0.018	***93.7** ± 1.2	**89.7** ± 1.1	***0.916** ± 0.007
LogitBoost	73.0 ± 3.5	77.9 ± 1.7	0.753 ± 0.016	90.8 ± 1.7	84.8 ± 1.6	0.877 ± 0.011
MLP-LM	77.3 ± 4.3	79.0 ± 4.2	0.780 ± 0.032	90.5 ± 2.9	88.2 ± 2.1	0.893 ± 0.017
MLP-GD	78.5 ± 2.1	79.8 ± 3.2	0.791 ± 0.021	1.2 ± 2.0	88.2 ± 2.6	0.897 ± 0.016
C4.5	72.9 ± 1.8	73.4 ± 2.3	0.731 ± 0.015	86.8 ± 1.5	85.3 ± 1.5	0.860 ± 0.010
Floresta Aleatória	81.0 ± 2.0	80.7 ± 0.6	0.808 ± 0.010	92.5 ± 0.8	86.6 ± 0.9	0.894 ± 0.006
IBK	68.0 ± 1.8	74.5 ± 1.4	0.711 ± 0.010	81.5 ± 2.0	84.6 ± 1.3	0.830 ± 0.012
SVM	50.7 ± 2.0	77.7 ± 2.2	0.613 ± 0.018	66.5 ± 2.4	86.1 ± 1.7	0.750 ± 0.015

Tabela 8. Resultados obtidos na detecção de *spam hosts* usando combinação de todos os atributos (*links*, *links* transformados e conteúdo).

	Classes desbalanceadas			Classes balanceadas		
	Sensitividade	Precisão	F-medida	Sensitividade	Precisão	F-medida
Bagging	82.1 ± 2.0	82.7 ± 2.9	0.824 ± 0.019	92.5 ± 1.3	88.4 ± 1.3	0.904 ± 0.010
AdaBoost	81.6 ± 1.2	**84.2** ± 2.7	**0.829** ± 0.014	**93.2** ± 0.9	***89.8** ± 1.3	**0.915** ± 0.010
LogitBoost	75.9 ± 2.3	77.6 ± 2.2	0.767 ± 0.020	89.8 ± 2.2	84.9 ± 1.5	0.873 ± 0.011
MLP-LM	81.7 ± 3.7	79.2 ± 2.5	0.804 ± 0.027	89.7 ± 4.6	86.1 ± 4.0	0.878 ± 0.034
MLP-GD	79.9 ± 2.7	78.9 ± 3.3	0.793 ± 0.024	92.4 ± 1.4	88.8 ± 1.5	0.906 ± 0.010
C4.5	73.4 ± 2.7	73.0 ± 2.0	0.731 ± 0.014	86.3 ± 2.6	85.1 ± 1.7	0.857 ± 0.017
Floresta Aleatória	***82.4** ± 1.1	80.4 ± 2.3	0.814 ± 0.012	92.7 ± 1.5	87.0 ± 1.2	0.897 ± 0.008
IBK	69.2 ± 1.4	71.6 ± 1.6	0.704 ± 0.013	83.0 ± 2.3	83.0 ± 2.1	0.830 ± 0.017
SVM	55.9 ± 2.0	71.1 ± 1.9	0.626 ± 0.018	78.2 ± 1.4	81.9 ± 1.5	0.800 ± 0.011

De maneira geral os métodos avaliados obtiveram bons resultados, pois foram capazes de detectar com alta precisão uma quantidade expressiva de *spam hosts*, independentemente da característica empregada. Além disso, se for considerado apenas os cenários sem combinação de atributos, os resultados indicam que entre os métodos avaliados, o método de *bagging*, na média, apresentou os melhores resultados. Ele foi capaz de

detectar em média 82,2% dos *spam hosts* com uma precisão média de 82,7%. Porém, nos cenários em que foram feitas combinações de atributos, em geral, a técnica de *AdaBoost* foi mais eficiente. Portanto, é importante observar que a escolha do método de aprendizagem de máquina mais adequado para a classificação de *spam hosts* pode variar de acordo os atributos disponíveis. Por outro lado, o classificador SVM, em média, apresentou o pior desempenho, tanto nos cenários com combinação de atributos, quanto nos cenários em que os atributos não foram combinados.

Os resultados também indicam que os classificadores obtiveram melhor desempenho quando foram treinados usando classes balanceadas. Portanto, verifica-se que, em geral, os métodos de classificação analisados tendem a favorecer a classe com maior número de representantes apresentados na fase de treinamento. 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 padrões pertencentes à classe *spam*.

Outro ponto a ser observado é que os métodos de aprendizagem de máquina tiveram melhor desempenho na classificação de *spam hosts* nos cenários em que foram feitas combinações de atributos. O melhor resultado foi obtido quando foram combinados os atributos extraídos da relação de *links* transformados com os atributos extraídos do conteúdo das páginas. Nesse cenário, usando classes balanceadas, a técnica de *AdaBoost* obteve o melhor valor da F-medida entre todas as simulações, conforme pode ser visto na Tabela 7.

4.1. Seleção de atributos

Com o objetivo de melhorar os resultados obtidos pelos algoritmos de aprendizagem de máquina nos cenários anteriores e enriquecer a análise de desempenho dos mesmos, foi feita uma seleção dos atributos usados nos experimentos. Esse procedimento foi realizado pois cogitou-se a hipótese de que poderia haver redundâncias nos vetores de características usados nos experimentos, que poderiam estar impactando o desempenho dos métodos de classificação. Então, foi empregado o método de análise de componentes principais (PCA – *Principal Component Analysis*) [Haykin 1998] para tentar obter uma estrutura simplificada dos dados e eliminar ou reduzir a existência de redundâncias nos vetores de características. Esse método foi escolhido por ser amplamente empregado no processo de redução de dimensionalidade em problemas de classificação e reconhecimento de padrões.

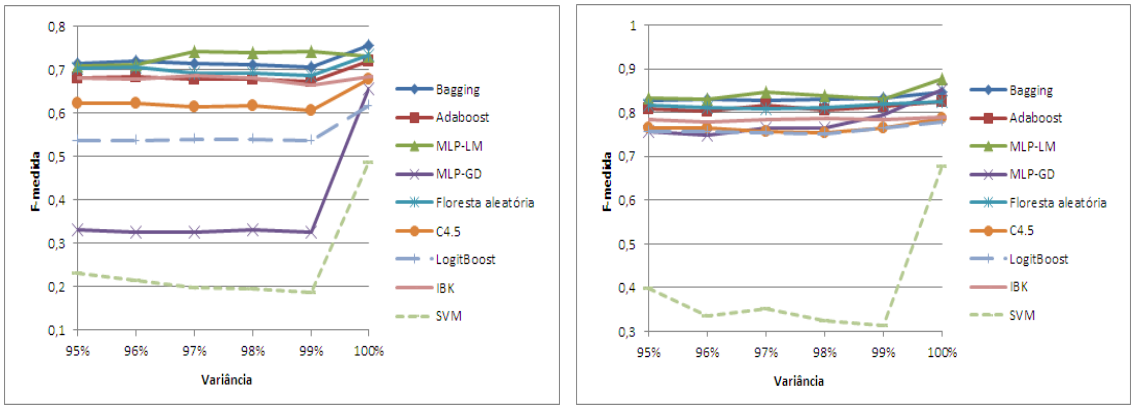
O método PCA foi empregado por meio do software WEKA, sendo usada uma porcentagem de variância que diversificou entre 95% e 99%. A Tabela 9 apresenta a redução em número de dimensões em relação a cada porcentagem de variância utilizada. A última coluna que está rotulada como “100%” apresenta o número de dimensões original dos vetores de atributos.

As Figuras de 1 a 7 mostram a variação dos resultados em relação ao número de dimensões reduzidas, sendo usada como medida de desempenho a F-medida. A última coluna de cada Figura, que está rotulada como “100%”, mostra o resultado obtido com o vetor de atributos original.

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 de análise de componentes principais. Logo, pode-se concluir que o método de PCA, apesar de reduzir o custo computacional do processo de classificação,

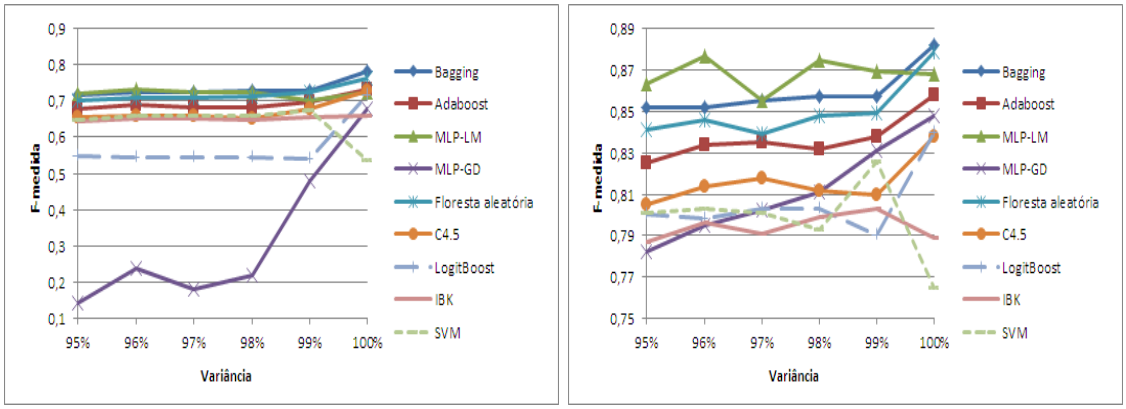
Tabela 9. Número de dimensões obtidas após redução de dimensionalidade.

	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
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



(a) Classes desbalanceadas (b) Classes balanceadas

Figura 1. Resultados obtidos nas simulações com vetores de atributos com dimensionalidade reduzida e extraídos do conteúdo das páginas

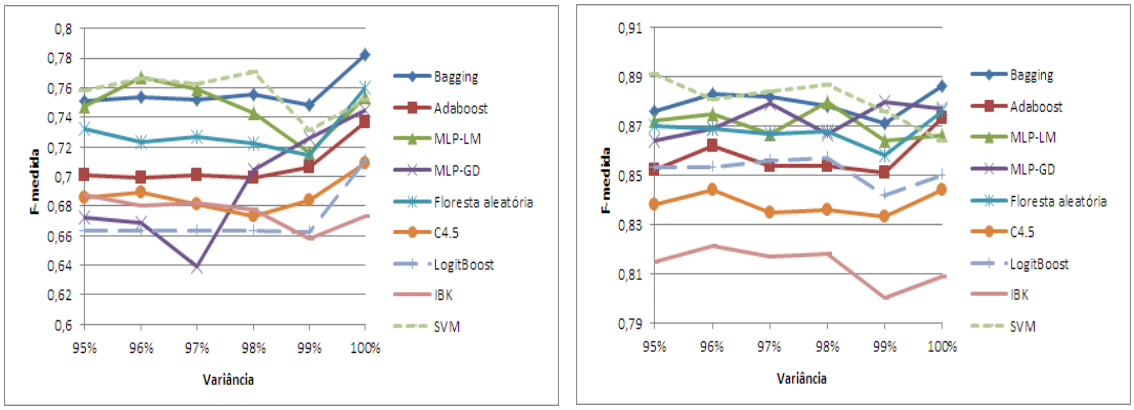


(a) Classes desbalanceadas (b) Classes balanceadas

Figura 2. Resultados obtidos nas simulações com vetores de atributos com dimensionalidade reduzida e extraídos da relação dos *links* das páginas.

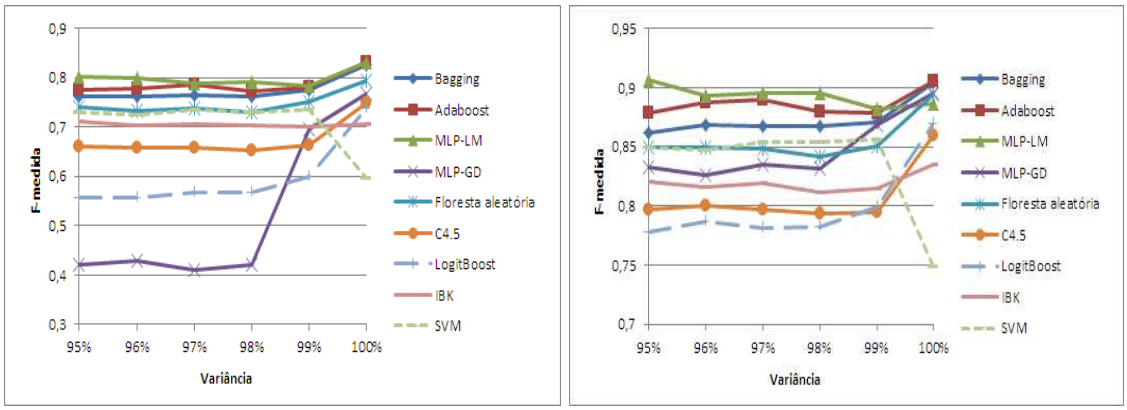
não é indicado para o problema de *web spam*, pois diminui a qualidade dos resultados.

Porém, é necessário destacar que para o método SVM, as Figuras 2, 3, 4, 5 e 7 mostram que quando são usados atributos extraídos dos *links* e dos *links* transformados ou quando é usada a combinação dos atributos extraídos dos *links* e do conteúdo, a combinação dos atributos extraídos dos *links* e dos *links* transformados e a combinação de todos os atributos, o método PCA é viável, pois melhorou os resultados da classificação. A MLP-LM também foi beneficiada pela redução da dimensionalidade em todos os cenários



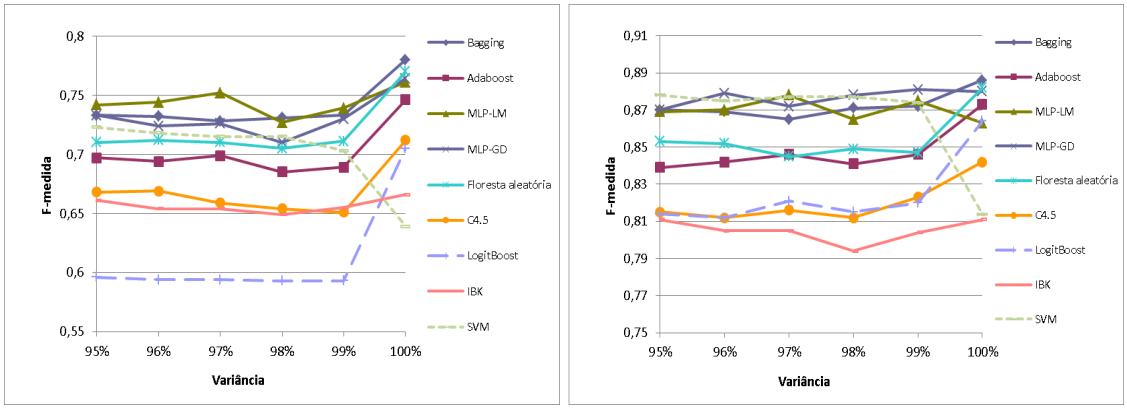
(a) Classes desbalanceadas (b) Classes balanceadas

Figura 3. Resultados obtidos nas simulações com vetores de atributos com dimensionalidade reduzida e extraídos da relação de *links* transformados das páginas.



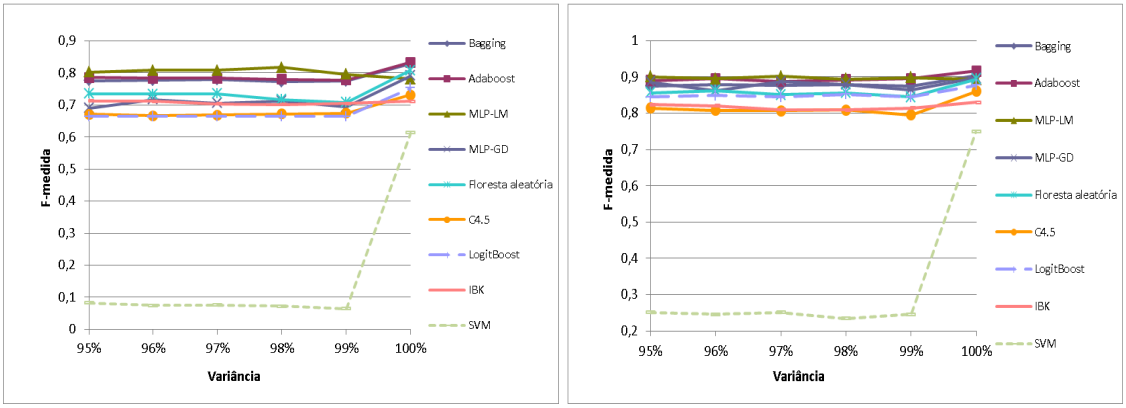
(a) Classes desbalanceadas (b) Classes balanceadas

Figura 4. Resultados obtidos nas simulações com vetores de atributos com dimensionalidade reduzida e formados pela combinação dos vetores de atributos extraídos do conteúdo e da relação dos *links* das páginas.



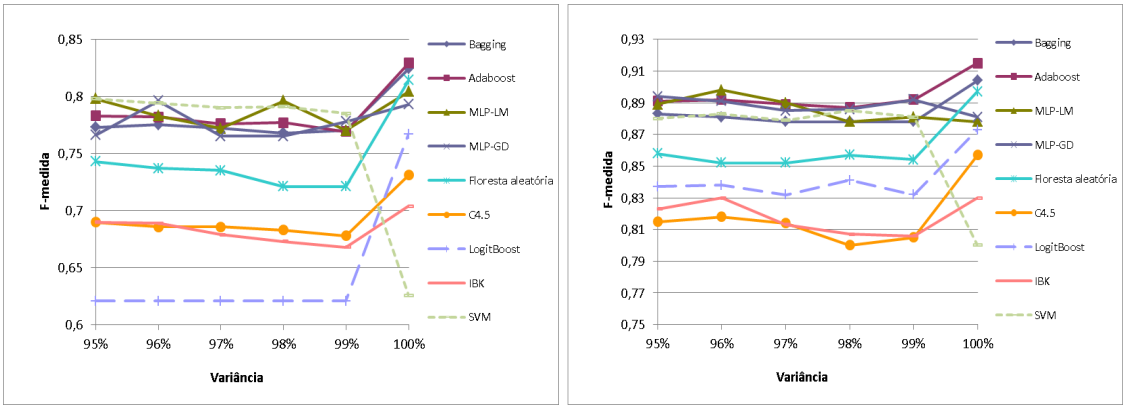
(a) Classes desbalanceadas (b) Classes balanceadas

Figura 5. Resultados obtidos nas simulações com vetores de atributos com dimensionalidade reduzida e formados pela combinação dos vetores de atributos extraídos da relação de *links* e de *links* transformados das páginas.



(a) Classes desbalanceadas (b) Classes balanceadas

Figura 6. Resultados obtidos nas simulações com vetores de atributos com dimensionalidade reduzida e formados pela combinação dos vetores de atributos extraídos do conteúdo e da relação de *links* transformados das páginas.



(a) Classes desbalanceadas (b) Classes balanceadas

Figura 7. Resultados obtidos nas simulações com vetores de atributos com dimensionalidade reduzida e formados pela combinação de todos os vetores de atributos.

em que as classes foram balanceadas.

Outro ponto que deve ser observado é que para a maioria dos cenários, quanto menor a variância usada na redução de dimensionalidade, menor a F-medida resultante. Porém, alguns resultados, tais como alguns apresentados nas Figuras 1 e 3, mostram que ao usar o PCA, adotar uma variância de 98% produz melhores resultados que uma variância de 99%.

4.2. Análise estatística

Para tornar a análise do desempenho dos métodos na classificação de *web spam* mais confiável, foi feita uma análise estatística dos resultados (Tabela 10). Então, os métodos foram ordenados pelo valor da F-medida usando o teste da soma dos postos de Wilcoxon (*Wilcoxon rank-sum test*) [Montgomery e 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 e Runger 2002]. Para a análise estatística, os resultados obtidos após redução de dimensionalidade não foram considerados.

Os métodos que estão no mesmo nível na Tabela 10 tem resultados estatistica-

Tabela 10. Análise estatística dos resultados usando o teste da soma dos postos de Wilcoxon.

Nível	Métodos	
	Classes desbalanceadas	Classes balanceadas
1	*Bagging* e floresta aleatória	*Bagging*, *AdaBoost*, MLP-LM, MLP-GD e floresta aleatória
2	Floresta aleatória e *AdaBoost*	*LogitBoost*
3	*AdaBoost* e MLP-LM	C4.5
4	MLP-LM e MLP-GD	IBK
5	C4.5 e *LogitBoost*	SVM
6	IBK	
7	SVM	

mente equivalentes. Logo, pode-se notar que o método de *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 igual ao *AdaBoost*, às MLPs e a floresta aleatória nos experimentos com classes balanceadas. Por outro lado, o SVM é estatisticamente inferior aos outros métodos avaliados nesse trabalho, nos testes com classes balanceadas e desbalanceadas.

4.3. Comparação dos resultados apresentados nesse trabalho com outros resultados disponíveis na literatura

Para facilitar a avaliação dos métodos analisados, é apresentado na Tabela 11 uma comparação entre os melhores resultados obtidos nesse trabalho e os disponíveis na literatura de *web spamming*. Para oferecer uma comparação justa, os métodos propostos na literatura foram treinados com a mesma base de dados, atributos e configurações usados nos métodos apresentados nesse trabalho. Foram usados exatamente os mesmos parâmetros descritos nos trabalhos propostos na literatura ou os parâmetros padrão da ferramenta ou biblioteca utilizada. Em resumo, os métodos de *bagging* de árvores de decisão [Castillo et al. 2007, Ntoulas et al. 2006] e *boosting* de árvores de decisão [Ntoulas et al. 2006] foram implementados usando a ferramenta WEKA. Já, as máquinas de vetores de suporte [Svore et al. 2007] foram implementadas usando a biblioteca LIBSVM na ferramenta MATLAB. Por outro lado, método de programação genética [Shengen et al. 2011] não foi reimplementado porque os autores adotaram a mesma base de dados usada nesse trabalho, logo, para a comparação foram usados os resultados originais fornecidos pelos autores.

Tabela 11. Comparação entre os melhores resultados obtidos nesse trabalho e os resultados disponíveis na literatura

Classificadores	Precisão	Sensitividade	F-medida
Resultados disponíveis na literatura			
Bagging [Castillo et al. 2007, Ntoulas et al. 2006] - conteúdo	81.5	68.0	0.741
Máquinas de vetores de Suporte [Svore et al. 2007] - conteúdo	55.5	**86.5**	0.677
Boosting [Ntoulas et al. 2006] - conteúdo	78.2	68.2	0.728
Programação genética [Shengen et al. 2011] - *links*	69.8	76.3	0.726
Programaçao genética [Shengen et al. 2011] - *links* transf.	76.5	81.4	0.789
Melhores resultados obtidos nesse trabalho			
Bagging - conteúdo+*links* transf.	83.6	82.2	0.829
AdaBoost - conteúdo+*links* transf.	**85.6**	81.1	**0.832**
Bagging - conteúdo+*links* transf. (classes balanceadas)	88.0	92.7	0.903
AdaBoost - conteúdo+*links* transf. (classes balanceadas)	**89.7**	**93.7**	**0.916**

É importante observar na Tabela 11 que os métodos *bagging* e *AdaBoost* obtiveram desempenho superior aos métodos propostos em outros trabalhos da literatura. Por-

tanto, é conclusivo que técnicas de aprendizado de máquina podem ser empregadas com sucesso para auxiliar o processo de detecção automática de *spam hosts*.

5. Conclusões e Trabalhos Futuros

Este trabalho apresentou uma análise de desempenho de diferentes algoritmos de aprendizagem de máquina conceituados na literatura para auxiliar na tarefa de detecção automática de *spam hosts*. Para isso, foi utilizada uma base de dados real, pública e de grande porte, representada por vetores de atributos baseados no conteúdo, nos *links* e nos *links* transformados e pela combinação de todos os vetores de atributos.

Os resultados dos experimentos mostraram que entre os métodos avaliados, os métodos de agregação de classificadores baseados em árvores, tais como *bagging* e *AdaBoost*, obtiveram os melhores desempenhos, demonstrando serem adequadas para auxiliar na detecção de *spam hosts*. Porém, por meio de uma análise estatística foi possível observar que as redes neurais MLP e o método floresta aleatória tem resultados equivalentes aos métodos *bagging* e *AdaBoost* quando são usadas classes balanceadas. Logo, eles também são métodos promissores para a detecção automática de *web spam*.

Com relação aos vetores de características, os melhores resultados foram obtidos quando os métodos classificadores utilizaram a combinação de informações extraídas do conteúdo das páginas e relação de *links* transformados. É importante destacar também que os métodos avaliados demonstraram ser mais eficientes quando treinadas 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.

Outro ponto importante que foi observado nesse trabalho é que a redução de dimensionalidade por meio do método de análise de componentes principais não melhorou os resultado obtidos pelos algoritmos de aprendizagem de máquina. Logo, apesar de reduzir o custo computacional da tarefa de classificação de *web spam*, esse procedimento não é adequado para o problema abordado, pois afeta negativamente a qualidade dos resultados.

Trabalhos futuros compreendem o estudo de formas de adaptar os métodos mais promissores para otimizar seu desempenho, a aplicação de outros métodos de análise e seleção de atributos e a proposição de novos tipos de atributos que possam aumentar a capacidade de predição dos algoritmos.

Agradecimentos

Os autores são gratos a Capes, Fapesp e CNPq pelo apoio financeiro.

Referências

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

Becchetti, L., Castillo, C., Donato, D., Leonardi, S., e Baeza-Yates, R. (2006). Using rank propagation and probabilistic counting for link-based spam detection. In *Proc. of the WebKDD'06*, Philadelphia,USA.

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

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

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

Castillo, C., Donato, D., e Gionis, A. (2007). Know your neighbors: Web spam detection using the web topology. In *Proc. of the 30th SIGIR*, pages 423–430, Amsterdam, The Netherlands.

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

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

Freund, Y. e Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proc. of the 13th ICML*, pages 148–156, Bari, Italy. Morgan Kaufmann.

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

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

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

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

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

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

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

John, J. P., Yu, F., Xie, Y., Krishnamurthy, A., e Abadi, M. (2011). deSEO: combating search-result poisoning. In *Proc. of the 20th SEC*, pages 20–20, Berkeley, CA, USA.

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

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

Lu, L., Perdisci, R., e Lee, W. (2011). SURF: detecting and measuring search poisoning. In *Proc. of the 18th CCS*, pages 467–476, New York, NY, USA.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Science, New York, NY, USA, 1 edition.

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

Ntoulas, A., Najork, M., Manasse, M., e Fetterly, D. (2006). Detecting spam web pages through content analysis. In *Proc. of the WWW*, pages 83–92, Edinburgh, Scotland.

Provos, N., Mavrommatis, P., Rajab, M. A., e Monrose, F. (2008). All your iFRAMEs point to us. In *Proc. of the 17th SS*, pages 1–15, Berkeley, CA, USA.

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

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

Shengen, L., Xiaofei, N., Peiqi, L., e Lin, W. (2011). Generating new features using genetic programming to detect link spam. In *Proc. of the ICICTA'11*, pages 135–138, Shenzhen, China.

Silva, R. M., Almeida, T. A., e Yamakami, A. (2012a). Análise de desempenho de redes neurais artificiais para classificação automática de web spam. *Revista Brasileira de Computação Aplicada*, 4(2):42–57.

Silva, R. M., Almeida, T. A., e Yamakami, A. (2012b). Análise de métodos de aprendizagem de máquina para detecção automática de spam hosts. In *Anais do XII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg'12)*, pages 2–15, Curitiba, PR, Brasil. Sociedade Brasileira de Computação.

Silva, R. M., Almeida, T. A., e Yamakami, A. (2012c). An analysis of machine learning methods for spam host detection. In *Proc. of the 11th International Conference on Machine Learning and Applications (ICMLA'12)*, pages 1–6, Boca Raton, Florida, USA.

Silva, R. M., Almeida, T. A., e Yamakami, A. (2012d). Artificial neural networks for content-based web spam detection. In *Proc. of the 14th International Conference on Artificial Intelligence (ICAI'12)*, pages 1–7, Las Vegas, NV, USA.

Silva, R. M., Almeida, T. A., e Yamakami, A. (2012e). Redes neurais artificiais para detecção de web spams. In *Anais do VIII Simpósio Brasileiro de Sistemas de Informação (SBSI'12)*, pages 636–641, São Paulo, Brazil.

Silva, R. M., Almeida, T. A., e Yamakami, A. (2012f). Towards web spam filtering with neural-based approaches. In *Advances in Artificial Intelligence – IBERAMIA 2012*, volume 7637 of *Lecture Notes in Computer Science*, pages 199–209, Cartagena de Indias, Colombia. Springer Berlin Heidelberg.

Svore, K. M., Wu, Q., e Burges, C. J. (2007). Improving web spam classification using rank-time features. In *Proc. of the 3rd AIRWeb*, pages 9–16, Banff, Alberta, Canada.

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

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