Um Modelo Adaptativo para a Filtragem de Spam

Ígor Assis Braga, Marcelo Ladeira

Departamento de Ciência da Computação
Universidade de Brasília (UnB) – Brasília, DF – Brasil

igorassisbraga@aluno.unb.br, mladeira@unb.br

Abstract. *Spamming has been a great problem to internet users and service providers. Recent research points towards the employment of machine learning algorithms to spam filtering. However, many works that have been done in the area do not recognize the dynamic aspect of spam, rendering the filtering process solely as a text categorization problem. This work presents an adaptive model for spam filtering that takes into consideration the dynamicity of spam. The model is applied making use of adaptive Huffman trees, support vector machines and aging of messages.*

Resumo. *O aumento do número de spam tem sido um grande problema para usuários e provedores de Internet. Pesquisas recentes apontam para o uso de algoritmos de aprendizagem de máquina para a construção de filtros, mas a maioria dos trabalhos não reconhece o caráter dinâmico do spam, julgando a filtragem como uma atividade de classificação de texto. Este artigo apresenta um modelo adaptativo para a filtragem de spam que leva em consideração a dinamicidade do spam. O modelo é aplicado utilizando-se árvores de Huffman adaptativas, support vector machines e envelhecimento de mensagens.*

1. Introdução

O ato de verificar a caixa de entrada de *e-mail* é cada vez uma atividade menos prazerosa para um número crescente de usuários do sistema, devido ao que comumente é chamado de *spam*, isto é, mensagens não solicitadas e indesejadas que foram enviadas em massa por um remetente que não possui ligação com o destinatário [Cormack & Lynam, 2005]. O *spam* se perde em meio às mensagens legítimas do usuário, e, dependendo do volume daquele, torna-se oneroso separá-los destas.

Os provedores de Internet têm reportado o aumento do número de *spam* por mensagem entregue, acarretando sobrecarga nos *links* de comunicação da Rede e perda de produtividade nas empresas. Tudo isso vem gerando muita reação, inclusive por parte da comunidade acadêmica, a qual estuda métodos de controle e filtragem de *spam* para serem aplicados nas esferas técnica, legislativa e social.

Várias técnicas de filtragem já foram empregadas para combater o *spam*, como busca de padrões por expressões regulares, listas negras e listas brancas. As primeiras respondiam bem às necessidades da época em que foram empregadas. Contudo, os *spammers* perceberam que suas mensagens estavam sendo barradas e passaram a modificá-las a fim de que elas pudessem passar pelos filtros, geralmente mudando o vocabulário utilizado nas mensagens. Os mantenedores dos filtros tinham, então, que atualizá-los manualmente, já que não havia nenhuma previsão para que esses filtros

aprendessem com novas mensagens que chegavam. Esse processo de tentar burlar os filtros continua até hoje, fazendo com que o modelo de *spam* seja bastante dinâmico.

As mais recentes pesquisas no combate ao *spam* envolvem o uso de algoritmos de aprendizagem de máquina juntamente com técnicas de mineração de texto para treinar classificadores que aprendam com mensagens previamente julgadas pelo usuário. A maioria das ferramentas de filtragem atualmente disponíveis tem por base alguma implementação do classificador *naïve* Bayes. Essa aplicação foi primeiramente proposta em [Sahami *et al.*, 1998] e detalhada em [Graham, 2002]. Outros classificadores já empregados incluem *support vector machines* [Drucker *et al.*, 1999], k-NN [Sakkis *et al.*, 2003] e árvores de decisão com *boosting* [Carreras & Màrques, 2001]. Esses trabalhos ainda apresentaram filtros estáticos, que precisam de retreinamento explícito do usuário para se adaptarem a mudanças no conteúdo e forma do *spam*.

O último ataque aos filtros parece ser o uso de imagens nas mensagens de *spam*. Esse fato evidencia que a filtragem de *spam* não é um problema de classificação de texto, como se vem tratando até agora. Esse artigo apresenta um modelo de filtragem adaptativa que leva em conta a natureza dinâmica do *spam*. O modelo é flexível para incorporar uma classificação baseada em outras fontes que não texto (imagens, por exemplo). O modelo é composto de: a) uma representação dinâmica dos conceitos de *spam* e mensagens legítimas com árvores de Huffman adaptativas (FGK) [Faller, 1973], [Gallager, 1978], [Knuth, 1985], b) uma classificação baseada em *Support Vector Machines* (SVM) [Vapnik, 1995] e c) um procedimento de envelhecimento exponencial para dar suporte à adaptação [Zhou *et al.*, 2005].

Este artigo, na Seção 2, descreve o modelo de filtragem adaptativa. As Seções 3 a 5 ilustram as técnicas envolvidas com cada uma das três partes do modelo, quais sejam, o pré-processamento, a classificação e a adaptação. Os resultados obtidos são mostrados na Seção 6. Por fim, a Seção 7 apresenta as conclusões e aponta possíveis linhas para trabalho futuro.

2. Modelo Adaptativo

O modelo está estruturado em três módulos: pré-processamento, classificação e adaptação (Figura 1). No pré-processamento, a mensagem será transformada em um vetor de valores numéricos que será utilizado para representá-la nos outros módulos.

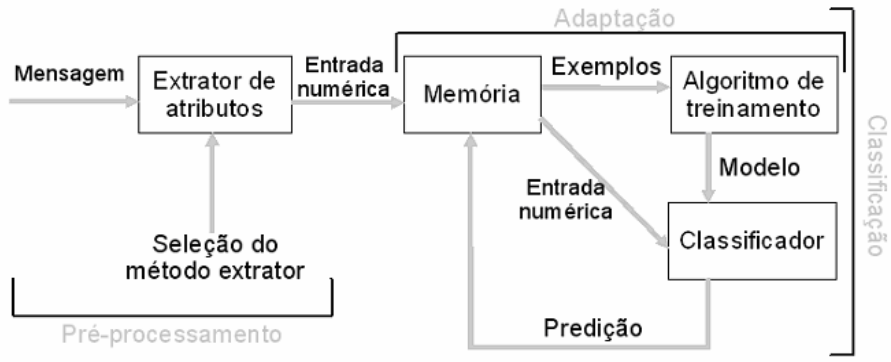

Figura 1. Modelo de filtro adaptativo de *spam*

A classificação gera predições, *spam* ou mensagem legítima, para cada entrada numérica relacionada a cada mensagem. O modelo é aprendido e revisto na fase de

adaptação, na qual exemplos e suas classes estão disponíveis para o algoritmo de treinamento. O usuário especifica um intervalo de tempo no qual o modelo ficará estático. Quando se esgota esse intervalo, há o retreinamento do classificador.

Para transformar as mensagens em uma entrada numérica, utiliza-se um extrator de atributos da mensagem. Como a informação necessária para classificar a mensagem como *spam* está em textos, em imagens ou em páginas formatadas em HTML, o extrator é uma abstração para algoritmos que lidam diretamente com essas fontes. O usuário deve poder selecionar, dentre os extratores disponíveis, aqueles que ele deseja utilizar.

Um classificador treinado no início de cada intervalo de tempo recebe o vetor numérico e prediz a classe da mensagem. Essa predição é guardada na memória, juntamente com o respectivo vetor. Se o usuário não concordar com a predição, ele pode intervir gravando a classe correta na memória. Quando o intervalo de tempo se esgota, os exemplos armazenados, preferencialmente aqueles mais novos, são usados para retreinar o classificador.

3. Pré-processamento com árvores FGK e ordenação

Classificadores podem lidar com dados textuais ou dados numéricos. Comparações com dados textuais são mais dispendiosas do ponto de vista computacional. Como mensagens de e-mail podem apresentar um número elevado de palavras, optou-se por utilizar classificadores com dados numéricos, tornando necessário extrair características numéricas dos textos das mensagens para formar o espaço de entrada para o classificador. Essa extração é crucial para se obter um bom desempenho e deve extrair o maior número de características relevantes possível da mensagem, pois nem mesmo o melhor classificador conseguirá separar duas mensagens de classes diferentes cujas características extraídas sejam iguais.

O espaço de entrada será codificado utilizando-se árvores de Huffman adaptativas (árvores FGK) e um algoritmo de ordenação das mensagens com base na representação vetorial das palavras que a compõe, tendo em vista a projeção desse vetor no espaço de mensagens legítimas e no espaço de mensagens de *spam*. Essa técnica foi apresentada em [Zhou *et al.*, 2005].

Uma árvore FGK pode ser usada para gerar códigos de mínima redundância, com aplicação prática na compressão de dados. As árvores FGK desta abordagem trabalham com palavras como seus símbolos, de modo que a freqüência das palavras e a árvore FGK sejam atualizadas em tempo real. No método adaptativo, cada árvore possui um nó especial, chamado de nó zero, porque seu peso é sempre zero. Quando uma nova palavra ocorre no conjunto de mensagens, o nó zero gera dois filhos: um outro nó zero e uma folha que representa a nova palavra. Cada nó recebe o peso que é a soma dos pesos dos seus nós filhos. No final da construção da árvore para um conjunto de treinamento, o peso de cada palavra é a freqüência dessa palavra no conjunto. A cada inserção de uma nova palavra, pode ser necessário que a árvore sofra ajustes para satisfazer a condição de *sibling* (irmandade) [Gallager, 1978], que diz que cada nó – exceto o raiz – deve possuir um nó irmão e deve ser possível listar todos os nós em ordem decrescente dos pesos, com os nós irmãos adjacentes na listagem. A Figura 2 mostra a construção dinâmica de uma árvore FGK. Por simplicidade de desenho, está sendo ilustrada a inserção de caracteres e não de palavras. A cada caractere incluído, a árvore atual sofre uma alteração para acomodar o novo caractere. O nó zero está marcado com a sigla

NYT (de *not yet translated* – não traduzido ainda). Cada nó tem um número de identificação e um valor que indica o seu peso. Somente os nós folhas armazenam palavras. A esses nós é associado um código binário de tamanho variável, definido pelo caminho do nó até o nó raiz. A idéia básica subjacente à árvore de Huffman é que aos nós que representam símbolos (no caso, palavras) mais freqüentes seja associado um código binário menor (portanto o nó deve estar mais próximo do nó raiz). Obviamente ao símbolo menos freqüente é associado o maior código binário. Esse código tem o tamanho (número de bits) igual à altura da árvore que o gerou.

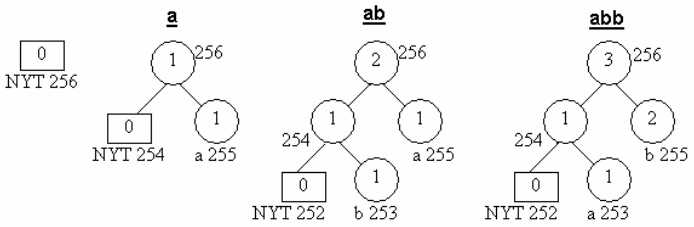

Figura 2. Construção de uma árvore FGK para caracteres

Para derivar a ordem para cada mensagem, deve-se, primeiramente, construir uma árvore FGK para o conjunto de mensagens de *spam* e uma outra árvore FGK para o conjunto de mensagens legítimas. Então, cada uma das árvores é transformada em um vetor $\vec{w} = (w_1, w_2, \ldots, w_N)$ onde **N** é o número de palavras na árvore e

$$w_i = 1 - \frac{L_i}{H},$$

é a representação de cada palavra no vetor. $\mathbf{L_i}$ é o tamanho do código binário da palavra e **H** a altura da árvore. Desta forma, w_i representa um peso proporcional à freqüência da i-ésima palavra codificada na árvore. Esse passo gera os vetores $\vec{w}_S = (w_1, w_2, \ldots, w_{N_S})$ para a árvore de *spam* e $\vec{w}_L = (w_1, w_2, \ldots, w_{N_L})$ para a árvore de mensagens legítimas.

Após a construção das duas árvores FGK citadas, para cada nova mensagem **m** a ser classificada são gerados os vetores $\vec{p}_L = (p_1, p_2, \ldots, p_{N_L})$ e $\vec{p}_S = (p_1, p_2, \ldots, p_{N_S})$, os quais representam, respectivamente, a projeção de **m** sobre o espaço de mensagens definido pela árvore de mensagens legítimas e sobre o espaço de mensagens definido pela árvore de *spam*. Cada $\mathbf{p_i}$ é a probabilidade da i-ésima palavra armazenada na respectiva árvore. Se a i-ésima palavra da árvore não consta da mensagem **m**, então $\mathbf{p_i}$ recebe o valor zero. Se uma palavra na mensagem não estiver representada na árvore correspondente, então a palavra é ignorada na formação desses vetores.

Para cada mensagem **m**, são calculadas as ordens $r_S = \frac{\vec{p}_S \bullet \vec{w}_S}{\|\vec{w}_S\|}$ e $r_L = \frac{\vec{p}_L \bullet \vec{w}_L}{\|\vec{w}_L\|}$, respectivamente ordem de spam e ordem de mensagem legítima. Essas ordens representam o comprimento do vetor $\vec{p}$ projetado no respectivo vetor $\vec{w}$. Por fim, a ordem da mensagem é calculada como $R_m = \frac{r_S}{r_L}$.

4. Classificação com SVM

Support Vector Machines (SVM) são um conjunto de algoritmos para regressão e classificação conhecidos pela excelente capacidade de generalização. O algoritmo de classificação baseado em SVM baseia-se num hiperplano de decisão que deixa a maior margem possível de separação entre as classes. A técnica tem esse nome por causa dos vetores mais próximos do hiperplano, chamados de *support vectors* (vetores de suporte ou de apoio). Esses vetores são os únicos necessários para descrever o hiperplano ótimo [Haykin, 2002].

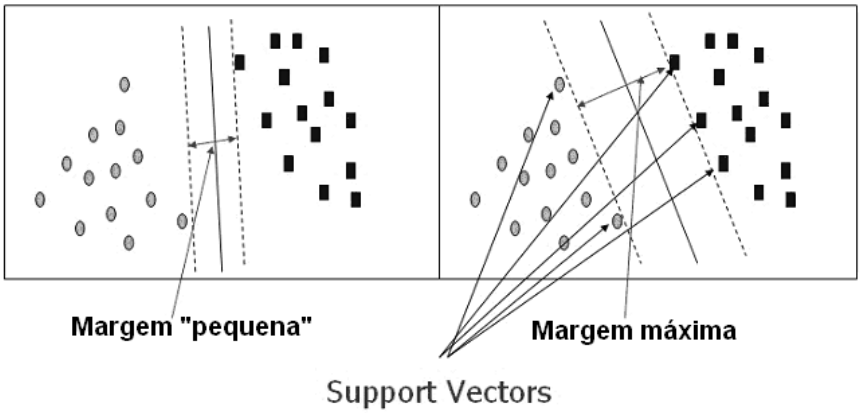

Figura 3. Classificador de margem máxima

A derivação da modelagem de SVM para padrões linearmente separáveis pode ser encontrada com detalhes em [Cristianini & Shawe-Taylor, 2000]. Seja um conjunto de treinamento $\{(\vec{x}_i, d_i)\}_{i=1}^{N}$. Cada elemento do conjunto de treinamento tem o rótulo da classe $\mathbf{d_i}$ igual a +1 para uma classe e -1 para a outra classe. Procura-se o hiperplano que satisfaça as condições

$$\vec{w} \bullet \vec{x}_i + b \geq 1, \;\; d_i = +1 \text{ e } \vec{w} \bullet \vec{x}_i + b \leq -1, \;\; d_i = -1$$

conjuntamente para todo exemplo de treinamento $\vec{x}_i$, e cuja margem de separação entre os padrões seja a maior possível. Os vetores de suporte são os únicos vetores que satisfazem a igualdade das equações acima. Pode-se mostrar que a distância entre um vetor de suporte ao hiperplano de separação é $r = \pm \frac{1}{\|\vec{w}\|}$. O valor da margem de separação entre os vetores de suporte é, então, $\rho = 2r = \frac{2}{\|\vec{w}\|}$. Sabendo, também, que a margem que deve ser maximizada é aquela entre os vetores de suporte, temos que a margem é máxima quando a norma do vetor peso é mínima. Pode-se enunciar, agora, o problema *primal* que emerge de SVM:

$$\text{minimizar } \Phi(\vec{w}) = \vec{w} \bullet \vec{w} \text{, sujeito à restrição } d_i(\vec{w} \bullet \vec{x}_i + b) \geq 1$$

para todos os vetores $\vec{x}_i$ do conjunto de treinamento.

Normalmente, o problema *primal* é substituído por um outro problema, o problema *dual*. Aplica-se o método dos multiplicadores de Lagrange ao problema *primal* para que se chegue ao problema *dual*. O problema *dual* é expresso como:

$$\text{maximizar}\, Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \left(\vec{x}_i \bullet \vec{x}_j \right)$$

em relação aos multiplicadores de Lagrange α_i de cada elemento do conjunto de treinamento $\{(\vec{x}_i, d_i)\}_{i=1}^{N}$. A otimização está sujeita, no entanto, às restrições

$$\sum_{i=1}^{N} \alpha_i d_i = 0 \text{ e } \alpha_i \geq 0, 1 \leq i \leq N .$$

A facilidade do problema *dual* é que a função a ser otimizada depende somente do conjunto de treinamento. Depois que os multiplicadores de Lagrange são determinados, o w_o ótimo é encontrado fazendo-se

$$w_o = \sum_{i=1}^{s} \alpha_i d_i \vec{x}_i \ .$$

O somatório é para todos os **s** α_i diferentes de zero, que são justamente aqueles associados a vetores de suporte. Para calcular o b_o ótimo basta fazer $b_o = 1 - \vec{w}_0 \bullet \vec{x}_i$ onde $\vec{x}_i$ é um vetor de suporte com $\mathbf{d_i} = +1$.

Uma modificação do classificador de margem máxima chamada *soft margin* SVM permite encontrar um hiperplano de separação para conjuntos de treinamentos não-lineares. O compromisso entre tentar encontrar um hiperplano ótimo e permitir violações é expresso por uma constante **C**. O problema *dual* permanece o mesmo, porém a restrição sobre os multiplicadores de Lagrange fica $0 \leq \alpha_i \leq C$.

A maior dificuldade que surge em SVM é a otimização encontrada no problema *dual*, que se trata de um problema de programação quadrática (QP). Para solucionar o problema QP, podem ser usados pacotes comercias ou algum dos métodos que foram desenvolvidos para a solução do problema específico de SVM (SMO e *Chunking*, por exemplo). O SMO, *Sequential Minimal Optimization* [Platt, 1998], soluciona o problema QP de forma aproximada. Entre as vantagens do SMO estão a eficiência para a classificação *soft margin* descrita acima e a exploração de esparsidade em vetores do conjunto de treinamento.

Neste trabalho, foi utilizado a *soft margin* SVM treinada com o SMO para ajustar, implicitamente, um limiar para a ordem $\mathbf{R_m}$ a partir das mensagens de treinamento. Assim, o vetor $\vec{x}_i$ é particularmente unidimensional neste caso. O rótulo de *spam* é associado ao valor +1 e o de mensagens legítimas ao valor -1.

5. Adaptação com envelhecimento de mensagens

Zhou *et al.* (2005) apresentaram um método de adaptação que inclui o que eles chamam de envelhecimento exponencial das mensagens. Para entender o procedimento, considere o conjunto $\chi = \{X_1, X_2, \ldots, X_t\}$ onde cada $\mathbf{X_i}$ é um conjunto de mensagens recebidas para treinamento no tempo **i**. Para retreinar o classificador no tempo **t+1**, são usadas as mensagens que chegaram até o tempo **t**, porém nem todas as mensagens de cada conjunto $\mathbf{X_i}$ são escolhidas.

Para dar uma importância maior às mensagens mais novas, o número de mensagens escolhidas em cada conjunto decresce exponencialmente em relação ao índice **i**. O conjunto de mensagens escolhidas para retreinamento é

$$\chi' = \bigcup_{i=1}^{t} \mathrm{X}_i'$$

onde $\mathbf{X_i}'$ é o conjunto de mensagens escolhidas aleatoriamente a partir do conjunto $\mathbf{X_i}$. O número de mensagens escolhidas segue a equação

$$\left|\mathrm{X}_i'\right| = \gamma^{t-i} \cdot \left|\mathrm{X}_i\right|, \quad 0 < \gamma \leq 1$$

onde γ é uma taxa que diz o quão rápido o processo deve esquecer as mensagens mais antigas. A Figura 4 ilustra o crescimento do conjunto de mensagens selecionadas para treinamento no tempo **t+1** em relação à taxa γ quando o tamanho dos conjuntos de mensagens coletadas $\mathbf{X_i}$ são iguais. Pode-se observar, pela figura, que o tamanho do conjunto de treinamento cresce exponencialmente com o valor da taxa γ.

Na abordagem de Zhou, não são as mensagens que são guardadas em cada tempo, mas a ordem das mensagens junto com o indicador da classe de cada mensagem. A grande vantagem desse método é de não ser necessário guardar as mensagens para retreinamento. O problema que pode ocorrer é que aconteça alguma não-linearidade no espaço de entrada para o classificador. Para contornar esse problema, duas soluções foram pensadas. Uma é incluir a informação do tempo em que a mensagem foi classificada, o que preserva a vantagem do não armazenamento das mensagens. A outra é preservar as mensagens e reavaliar as suas ordens. Nesta última solução, o problema do armazenamento volta e é necessário ter um controle rígido sobre a taxa γ, para que o número de mensagens de treinamento não se torne inaceitavelmente grande.

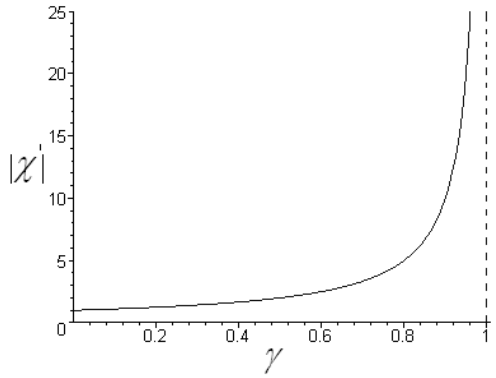

Figura 4: Crescimento do conjunto de treinamento χ' em relação a γ

6. Resultados e Análise

O modelo adaptativo para filtragem de *spam* proposto foi implementado em linguagem Java 5. Como extrator de atributos foi utilizada a Biguá, ferramenta aberta para mineração de textos que está sendo desenvolvida na COPPE/UFRJ. Dessa ferramenta foram utilizadas as facilidades de *stop-list* (lista de palavras a serem desconsideradas das mensagens) e *stemming* (extração de radicais das palavras) para pré-processamento do conjunto de mensagens. A aplicação dessas técnicas permite reduzir o número de palavras (Seção 3) que serão inseridas nas árvores FGK para *spam* e para mensagens

legítimas. O conjunto de mensagens utilizado para avaliação do modelo proposto foi o *corpus* LingSpam, produzido por Ion Androutsopoulos para uso em seus artigos sobre *spam* [Androutsopoulos *et al.*, 2000]. Desde então, esse *corpus* tem sido utilizado em diversos estudos comparativos de performance de filtros de *spam*. O LingSpam consiste de 481 *spam* e 2412 mensagens legítimas obtidas de uma lista de e-mails acadêmicos sobre lingüística.

A metodologia de teste utilizada é baseada nas seguintes etapas: a) extração de palavras com freqüência maior ou igual a 3 das mensagens, com o uso do Biguá (caso normal para controle, uso de *stop-list*, uso de *stemming* e uso conjunto de *stop-list* e *stemming*), b) geração de árvores FGK a partir dessas palavras e determinação da ordem de cada mensagem, c) construção do classificador e classificação das mensagens com o uso do método SVM/SMO com **C** = 1 e d) utilização de validação cruzada (do inglês, *10-fold cross-validation*) para avaliação dos classificadores obtidos. A validação cruzada consiste em dividir de forma aleatória estratificada o conjunto de dados em **k** partes, usar **k**-1 partes para treinar o modelo e a parte restante para avaliá-lo. Esse procedimento é repetido **k** vezes de forma que todas as instâncias do conjunto de dados sejam utilizadas **k**-1 vezes para treinamento e apenas uma vez para validação. No caso de classificadores é natural medir a performance em termos de erro e de sucesso na classificação. Para classificação multiclasse, as classificações no conjunto de avaliação são apresentadas na forma de uma matriz bidimensional com uma linha e uma coluna para cada classe. Cada elemento desta matriz, chamada matriz de confusão, apresenta o número de casos avaliados na qual a classe real (R) é a linha e a classe predita (P) pelo classificador é a coluna. Bons resultados são caracterizados por valores altos na diagonal principal e valores nulos para elementos fora da diagonal principal. Nesse artigo o classificador está sendo utilizado para classificar uma mensagem em apenas duas classes, "*spam*" ou "legítima" (Tabela 1).

Tabela 1. Matriz de confusão para classificação de e-mails

R \ P	***spam***	**legítima**
spam	Verdadeiro Positivo (VP)	Falso Negativo (FN)
legítima	Falso Positivo (FP)	Verdadeiro Negativo (VN)

A partir da matriz de confusão podem ser calculados índices de desempenho, dentre eles os apresentados na Tabela 2. Todos eles assumem valores no intervalo [0,1]. A precisão da classe positiva é a taxa de verdadeiro positivo (**TVp**); o *recall* da classe positiva é a taxa de verdadeiro positivo dentro da classe positiva ($\mathbf{TVp^p}$), a acurácia é a probabilidade de acerto do classificador e F-*measure* é a media harmônica ponderada que representa uma ponderação entre precisão e *recall*. O ponto de máximo para as métricas de acurácia e F-*measure* ocorre quando **TVp** e $\mathbf{TVp^C}$ tendem para 1. As fórmulas da Tabela 2 referem-se às medidas acima para a classe de *spam*.

Tabela 2. Índices para Discriminação entre Classificadores Dicotômicos

Precisão de *spam* (S)	***Recall* de *spam* (E)**	**Acurácia**	**F-*measure***
VP / (VP+FN)	VP / (VP+FP)	(VP+VN) / (VP+FP+VN+FN)	2 * S * E / (S+E)

Como índices de desempenho foram calculados os parâmetros de acurácia, precisão de *spam*, *recall* de *spam* e F-*measure*. Esses índices são calculados para cada uma das **k** dobras. Os valores médios obtidos para as **k** dobras são considerados as medidas de performance do classificador e estão apresentados na Tabela 3. A Tabela 3

também apresenta outros dois resultados a fim de comparação. Esses resultados se encontram em [Sakkis *et al.*, 2003] e referem-se ao uso dos classificadores k-NN e *naïve* Bayes quando não se diferencia o custo dos dois tipos de erro de classificação. Para esse estudo específico de LingSpam não foi considerado envelhecimento de mensagens por não haver dados seqüenciais (mensagens) disponíveis.

Tabela 3. Performance média dos classificadores de *spam*

Classificador	Precisão de *spam*	*Recall* de *spam*	Acurácia	F-*measure*
Normal (sem *stop-list* ou *stemming*)	0,9967	0,9858	0,9851	0,9912
Pré-processamento e *stop-list*	0,9979	0,9768	0,9782	0,9873
Pré-processamento e *stemming*	0,9975	**0,9866**	**0,9865**	**0,9920**
Pré-processamento, *stop-list* e *stemming*	**0,9988**	0,9792	0,9810	0,9889
k-NN (k = 8)	0,9739	0,8860	-	-
naïve Bayes	0,9902	0,8235	-	-

Todos os classificadores desenvolvidos apresentaram desempenho superior ao k-NN e ao *naïve* Bayes, indicando que a proposta de pré-processamento das mensagens via árvores FGK e treinamento e classificação com uso do algoritmo SVM/SMO é promissora para o desenvolvimento de filtros de *spam*. É interessante notar que os classificadores que utilizaram pré-processamento com *stop-list* e/ou *stemming* não evidenciaram resultados muito melhores. Os números acima são compatíveis com informes disponíveis na literatura que reportam casos em que o uso de *stop-list* tende a deteriorar a performance da classificação com SVM [Drucker *et al.*, 1999].

Os tempos medidos em termo de desempenho computacional estão apresentados na Tabela 4. Todos os classificadores foram treinados e utilizados em microcomputador PC Pentium IV, com 1 GB de RAM. Os tempos medidos são apenas para comparação relativa entre eles. Esses tempos incluem o tempo de execução do Biguá (pré-processamento), do treinamento e da classificação SVM/SMO. Não foram computados os tempos para geração de árvores FGK.

Tabela 4. Tempos médios dos classificadores de *spam* (segundos)

Classificador	Tempo de Execução	Tempo de Pré-processamento
Normal (sem *stop-list* ou *stemming*)	87.67	61.03
Pré-processamento e *stop-list*	91.67	70.64
Pré-processamento e *stemming*	81.23	52.23
Pré-processamento, *stop-list* e *stemming*	78.87	58.58

7. Conclusões e trabalhos futuros

O experimento com o *corpus* LingSpam apresentou excelente resultado e é evidência de que o modelo de filtro de *spam* proposto pode ser promissor. No entanto são necessários experimentos com *corpora* com número maior de mensagens para permitir uma análise mais acurada da adequação desse modelo, assim como experimentos específicos para avaliar o procedimento de envelhecimento exponencial das mensagens. Como trabalho futuro, será investigada a exploração do contexto (grupo de palavras adjacentes) nos textos das mensagens de *e-mail*, com o uso de classificadores probabilísticos baseados em campos aleatórios de Markov. Em uma outra frente, será investigado o uso de

árvores de Huffman adaptativas como método para a classificação de mensagens em um modelo gerativo.

Agradecimentos

Pesquisa realizada com apoio parcial do CNPq (PIC/UnB/CNPq) e da CAPES (programas PROCAD e Cooperação Internacional CAPES/GRICES).

Referências

Androutsopoulos, I., Koutsias, J., Chandrinos, K.V., Paliouras, G. & Spyropoulos, C.D. (2000). An Evaluation of Naive Bayesian Anti-Spam Filtering. In: *Proc. of the Workshop on Machine Learning in the New Information Age*, ECML 2000, p.9-17.

Carreras, X., Màrques, L. (2001). Boosting Trees for Anti-Spam Email Filtering. In: *Proc. of the 4th Intl. Conf. on Recent Advances in Natural Language Processing.*

Cormack, G. & Lynam, T. (2005). *TREC 2005 Spam Track Overview*. http://trec.nist.gov/pubs/trec14/papers/SPAM.OVERVIEW.pdf. Acesso em fev/2007.

Cristianini, N. & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press.

Drucker, H., Wu, D. & Vapnik, V. (1999). Support Vector Machines for Spam Categorization. In: *IEEE Transactions on Neural Networks*, p.1048-1054, v.10, n.5.

Faller, N. (1973). An Adaptive System for Data Compression. In: *Record of the 7th Asilomar Conference on Circuits, Systems and Computers*, p.593-597.

Gallager, R. (1978). Variations on a Theme by Huffman. In: *IEEE Transactions on Information Theory*, p.668-674, v.24, n.6.

Graham, P. (2002). *A Plan for Spam*. http://www.paulgraham.com/spam.html. Acesso em fev/2007.

Haykin, S. (2001). *Redes Neurais: Princípios e Prática*. Porto Alegre: Bookman. 2ª edição (Trad. Paulo Martins Engel).

Knuth, D. (1985). Dynamic Huffman Coding. In: *Journal of Algorithms*, p.163-180, v.6, n.2.

Platt, J. (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. *Technical Report 98-14*, Microsoft Research

Sahami, M., Dumais, S., Heckerman, D. & Horvitz, E. (1998). A Bayesian Approach to Filtering Junk E-mail. In: *Learning for Text Categorization: Papers from the 1998 Workshop*, AAAI Technical Report WS-98-05.

Sakkis, G., Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Spyropoulos, C. & Stamatopoulos, P. (2003). A Memory-Based Approach to Anti-Spam Filtering for Mailing Lists. *Information Retrieval*, p. 49-73, v. 6.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag.

Zhou, Y., Mulekar, M.S. & Nerellapalli, P. (2005). Adaptive Spam Filtering Using Dynamic Feature Space. In: *Proc. of the 17th IEEE Intl. Conf. on Tools with Artificial Intelligence*.