Universidade do Minho
Escola de Engenharia

Artur Jorge Paiva Correia Machado

Filtragem Colaborativa de Correio Electrónico não Solicitado

Outubro de 2009

Universidade do Minho
Escola de Engenharia

Artur Jorge Paiva Correia Machado

Filtragem Colaborativa de Correio Electrónico não Solicitado

Mestrado de Informática

Trabalho efectuado sob a orientação do
Professor Doutor Pedro Nuno Miranda de Sousa
e
Professor Doutor Paulo Alexandre Ribeiro Cortez

Outubro de 2009

DECLARAÇÃO

Nome : Artur Jorge Paiva Correia Machado

Endereço electrónico: ajpcm1@gmail.com Telefone: 913342223

Número do Bilhete de Identidade: 12583166

Título dissertação ?/tese ?: Filtragem Colaborativa de Correio Electrónico não Solicitado

Orientador(es): Professor Doutor Pedro Nuno Miranda de Sousa e Professor Doutor Paulo Alexandre Ribeiro Cortez

Ano de conclusão: 2009

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento: Mestrado Informática

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

Universidade do Minho, ___/___/_____

Assinatura: _____

Agradecimentos

Para o sucesso do desenvolvimento e conclusão deste trabalho foi necessária muita dedicação e empenho, não só por minha parte, mas também por parte dos que me rodeiam. Por tal, não posso deixar de assinalar esta preciosa ajuda.

Entre professores, família e amigos alguns intervenientes se destacam pela orientação e paciência que dedicaram no desenrolar deste projecto.

A nível académico, queria expressar a minha gratidão aos meus orientadores: Professor Doutor Pedro Sousa e Professor Doutor Paulo Cortez, que me aconselharam e me guiaram sempre no caminho certo para que fosse possível a concretização desta dissertação.

Também queria deixar um especial obrigado à minha namorada Ana que, não só me deu forças para lutar contra as adversidades deste projecto, também me foi ajudando na escrita deste documento.

Por fim, queria também agradecer por todo o apoio financeiro e moral aos meus pais, que ao longo destes anos me apoiaram em todas as minhas decisões.

A todos estes um grande obrigado pela ajuda.

Este trabalho foi suportado por uma Bolsa de Investigação (BI), financiada pela Fundação para a Ciência e a Tecnologia (FCT), dentro do projecto de R&D PTDC/EIA/64541/2006 – "SPAM Telescope Miner: detecção a nível mundial de correio electrónico não solicitado via técnicas de data mining".

Resumo

O serviço de correio electrónico assume uma crescente importância na sociedade actual e, em particular, no âmbito dos serviços de comunicações que actualmente são alvo preferencial de utilização por parte dos utilizadores da Internet. No entanto, são ainda muitos os desafios que se colocam a este serviço, nomeadamente no que diz respeito à crescente proliferação do fenómeno de correio electrónico não solicitado, i.e., a proliferação do *spam*.

Nos dias de hoje, este problema assume um elevado grau de complexidade, pois não só o utilizador perderá tempo na leitura, mas também põe em causa a segurança e a privacidade dos dados informáticos deste. Para combater este problema actualmente existem muitas propostas que possibilitam o combate ao *spam*. Algumas destas são ineficazes ou pouco utilizadas, visto que visam mais penalizar quem envia *spam* e não tanto inviabilizar a sua prática. Outras soluções recorrem por exemplo a técnicas de filtragem das mensagens de correio electrónico tendo como objectivo a identificação automática de *spam*, sendo pois uma mais valia para o utilizador final.

Neste sentido, o tema do trabalho documentado nesta dissertação enquadra-se no contexto anteriormente descrito, e em particular no âmbito de um projecto I&D denominado por "*Spam Telescope Miner*" que está actualmente em curso. Desta forma, este trabalho posiciona-se na perspectiva do desenvolvimento de um sistema colaborativo que permita aos utilizadores finais contribuir para a obtenção de mecanismos de detecção de *spam* mais eficientes. Isto será realizado através do desenvolvimento de um sistema que permita, de uma forma simples e flexível, o envio de *training sets* dos utilizadores finais para servidores especializados (repositórios). Desta forma, o sistema deverá possibilitar o acesso e a troca colaborativa destes ficheiros entre os diferentes utilizadores de rede através do acesso ao servidor/repositório central. Posteriormente o sistema possibilitará o desenvolvimento e a experimentação de diferentes técnicas colaborativas que combinem os filtros partilhados por forma a melhorar os processos de detecção de correio electrónico não solicitado.

Esta colaboração no domínio da filtragem permitirá melhorar os sistemas de detecção actualmente existentes, contribuindo assim para um melhor desempenho do serviço de correio electrónico e uma maior satisfação dos utilizadores finais.

Palavras-chave: *Spam*, *Ham*, Filtros Colaborativos, Filtros Heurísticos, *Naive Bayes, Java, Python, XUL, JavaScript*

Abstract

The importance of the email service is growing in today's society and, in particular, within the communications services that are currently prime target for use by Internet users. However there are still many challenges facing this service, in particular the proliferation of the phenomenon of unsolicited email (spam).

Nowadays, this problem has a high degree of complexity, because not only users spend many time reading unsolicited email, but also by the fact that spam may undermine the security and privacy of computers data. To combat this problem there are many techniques currently in practice. Some of these are ineffective or poorly used, trying to penalize the sender rather than derail the practice. Other techniques aim to hinder the spam practice by the use of several filtering solutions to detect spam messages. Techniques like such ones will help the users in the classification of email messages, thus avoiding the waste of time with spam.

Given this scenario, the theme of the dissertation to be held falls within the context described above, and in particular within the I&D project entitled "SPAM Telescope Miner" which is currently in course. Thus, this work stands in the development of a collaborative system that allows users to contribute to the achievement of more efficient spam detection mechanisms. This will be accomplished through the development of a system that allows, in a simple and flexible way, the sending of training sets of the end users to specialized servers (repositories). Similarly, the system should facilitate access and collaborative exchange of files between different users of the network through access to the central repository. The system should also allow the development of distinct techniques able to combine the shared filters in order to improve the spam detection capabilities of end users.

This collaboration in the field of filtering will improve detection systems currently available, thus contributing to a better performance of the email service and greater satisfaction of end users.

Keywords: Spam, Ham, Collaborative Filters, Naive Bayes, heuristical filters, Java, Python, XUL, JavaScript

Índice

Índice de Imagens

Índice de Tabelas

Acrónimos

API	Application Programming Interface
AUC	Area Under the Curve
BF	Bayes Flexível
CSS	Cascading Style Sheets
DNS	Domain Name System
DTD	Document Type Definition
FN	Falso Negativo
FP	Falso Positivo
FPR	False Positive Rate
FTP	File Transfer Protocol
GR	Gestor de Repositório
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IMAP	Internet Message Access Protocol
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extensions
MTAs	Mail Transfer Agents
MVBNB	Multi-Variávell Bernoulli
MVGNB	Multi-Variável Gauss
NB	Naive Bayes
PHP	PHP: Hypertext Preprocessor
POP	Post Office Protocol
RDF	Resource Description Framework
RF	Repositório de Ficheiros
ROC	Receiver Operating Characteristic
RSS	Really Simple Syndication
SFTP	Secure File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
SO	Sistema Operativo
SSL	Secure Sockets Layer

TFP	Taxa de Falsos Positivos
TLS	Transport Layer Security
TN	Total de Negativos
TP	Total Positivos
TPR	True Positive Rate
TVP	Taxa dos Verdadeiros Positivos
VP	Verdadeiros Positivos
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XUL	XML User Interface

1 Introdução

A crescente proliferação de correio electrónico não solicitado (*spam*) é um problema grave que afecta as redes de comunicações dos nossos dias. Este problema tem inúmeras razões que o explicam, sendo uma delas os factores económicos, que advêm da possibilidade de aceder a um elevado número de consumidores a um baixo custo [1, 2]. A título exemplificativo, alguns estudos indicam que, em 2007, cerca de 50% de todo tráfego mundial de *email* era do tipo *spam* [3]. Como é natural, este problema afecta indivíduos e organizações, devido à invasão de privacidade, divulgação de burlas/vírus informáticos [4], aumento do tráfego Internet e tempo gasto a ler mensagens não desejadas [1, 5]. Para minorar este problema existem diferentes abordagens, como as colaborativas (e.g. *blacklists*) e as baseadas em processos de filtragem (análise de conteúdo de *email*, via *text mining*) [6]. De igual modo, para compreender o fenómeno, existem diversos repositórios de *spam*. Em particular, hoje em dia está a ser criado, dentro do projecto I&D "*Spam Telescope Miner*", um repositório mundial de *spam* que permitirá perceber a evolução deste fenómeno e identificar parâmetros relevantes para o desenvolvimento de estratégias de filtragem de correio *spam*.

1.1 Enquadramento e Motivação

As abordagens existentes de combate ao *spam* são facilmente ultrapassadas, devido às permanentes alterações das técnicas usadas e tipo de *spam* enviado por parte dos *spammers*. Por exemplo, as *blacklists* usam um repositório, normalmente constituído por endereços de *email* usados no *spam*, ou as assinaturas (usando algoritmos de *hash* para conservar a confidencialidade) das próprias mensagens. Estas são muito limitadas na medida em que só detectam como *spam* um tipo específico de mensagens ou endereço de *email*, previamente classificado como *spam* por algum utilizador do grupo. Esta abordagem apresenta desvantagens adicionais, já que cada pessoa poderá ter uma diferente opinião sobre o que é *spam* e o que é *ham*[1]. Para contornar esta técnica, bastará ao *spammer* enviar a mensagem com um diferente endereço de *email*, ou alterar uma determinada letra que modificará a sua assinatura, e consequentemente evitará a sua detecção. Por último, as técnicas baseadas em processos de filtragem de conteúdos de *email*, tal como o *Naive Bayes*, têm como principal desvantagem o fraco desempenho para filtros pouco ou mal treinados, no

[1] Correio Solicitado

entanto são eficazes para filtros já treinados. A razão deste facto, advêm destas técnicas requererem um treino constante sobre o que é *spam*, para que futuros *emails* sejam automaticamente filtrados.

É neste contexto que se desenrola parte deste trabalho. Os contributos esperados estão associados à definição e desenvolvimento de uma arquitectura distribuída que possa auxiliar no desenvolvimento de estratégias colaborativas de filtragem de correio electrónico. Por tal, deste trabalho esperam-se fortes contributos no desenvolvimento de um sistema distribuído que, através de mecanismos simples e flexíveis, seja capaz de cumprir os objectivos que serão mencionados.

No que se refere à estrutura básica o sistema a desenvolver, este deverá basear-se em extensões a clientes do serviço de correio electrónico tipicamente utilizados na *Internet*. Estas extensões, que deverão ser fáceis de instalar e configurar, permitirão o desenvolvimento de funcionalidades adicionais nos clientes de correio electrónico e com fácil interface com o utilizador [7]. As extensões permitirão a interacção com servidores específicos, quer para submissão de amostras de *spam* recebido, quer para troca de filtros de *email*.

No que diz respeito à filtragem colaborativa dar-se-á um destaque à combinação das probabilidades estimadas por cada filtro individual [8]. Para combinar as saídas dos diversos modelos de *data mining* (i.e. filtros) existem diversas possibilidades [9, 10]: valor médio dos valores previstos, média ponderada por correlação, média pesada conforme a eficácia de cada filtro ou uso de um modelo adaptativo [2, 11]. Este último, por exemplo, produziria um resultado final (se o correio é solicitado ou não) através de um modelo de aprendizagem (e.g. *Naive Bayes* [6]) dos valores de cada filtro [12].

Neste trabalho serão adoptados os métodos de combinação de filtros recorrendo à média dos valores previstos e à média ponderada por correlação, como duas soluções possíveis. No entanto, tendo como base o sistema desenvolvido, será possível no futuro a concepção, implementação e teste de muitos outros modelos por forma a verificar qual o seu desempenho no âmbito do combate ao *spam*.

1.2 Objectivos

O trabalho a desenvolver tem como objectivo principal a definição e implementação de um sistema de partilha de informação relevante no contexto da filtragem colaborativa de correio electrónico não solicitado. Esta partilha de informação passará pela possibilidade de troca, entre diferentes utilizadores ou comunidades, dos seus filtros de *spam* de forma a possibilitar abordagens colaborativas para aumentar a eficácia dos esquemas usuais de filtragem.

Neste contexto, para atingir o objectivo principal definem-se como objectivos específicos deste trabalho os seguintes pontos:

- Ambientação à área de investigação e às sub-áreas que lhe estão relacionadas – deverá ser realizado um estudo de familiarização com a área onde o trabalho se enquadra e realizada uma análise bibliográfica sobre a mesma e sobre as diferentes sub-áreas relacionadas;
- Estudo e experimentação de tecnologias relevantes para o desenvolvimento – serão estudadas algumas das soluções tecnológicas que permitam o desenvolvimento do sistema pretendido, com especial incidência nos mecanismos de extensão para clientes de correio electrónico;
- Análise e desenho da arquitectura do sistema – deverá ser definida a arquitectura do sistema a desenvolver, especificados os componentes principais da mesma e os tipos de interacção e funcionalidades por ela suportadas;
- Implementação e desenvolvimento da solução – desenvolvimento de um protótipo da solução proposta e a sua integração num cenário real;
- Teste e análise de resultados – testes preliminares ao protótipo que foi desenvolvido e análise de vários resultados da sua utilização.

1.3 Organização

A presente dissertação está organizada em 6 capítulos, sendo estes organizados da seguinte forma:

1. **Introdução:** Capítulo que pretende fazer uma introdução à temática abordada neste trabalho, dando o enquadramento e a motivação sobre o problema. Seguidamente, são apresentados os objectivos do trabalho e, por último, descrita a organização do documento;
2. **O Serviço de Correio Electrónico:** Este capítulo apresentará inicialmente uma descrição do serviço de correio electrónico, sendo por isso especificada a sua arquitectura, os protocolos envolvidos, bem como as suas normas. Posteriormente, abordar-se-á a problemática desta dissertação, o correio não solicitado (*spam*). Aqui, uma descrição geral do problema com especial incidência nas implicações causadas será efectuada, tanto nos utilizadores (quer sejam empresas ou individuais) bem como nos *Internet Service providers* (ISP's). Este capítulo será finalizado com o estado da arte na protecção contra o *spam*, ou seja, serão enumeradas e descritas algumas técnicas existentes *anti-spam*;

3. **Clientes de email e Tecnologias de Desenvolvimento:** Neste capítulo, em primeiro lugar, será efectuada uma breve descrição de alguns clientes de *email* disponíveis para as várias plataformas de sistemas operativos e os prós e os contras da sua utilização neste projecto. De seguida, será especificada uma base teórica no desenvolvimento de extensões (*plug-in's*) para um cliente multi-plataforma e *open-source*, o *Thunderbird*. Aqui, serão descritas as várias linguagens usadas no desenvolvimento da extensão, bem como as funções desempenhadas no cliente *de email*;
4. **Solução Desenvolvida:** No capítulo, será apresentado o desenvolvimento da solução. Assim, descreve-se a arquitectura utilizada na solução. A descrição da arquitectura é ilustrada por imagens e esquemas que explicam os seus detalhes. Depois disso, irá explicar-se o funcionamento de toda a solução, incluindo todos os processos nela executados. Aqui, dar-se-á especial atenção ao método implementado de classificação de um novo *email* e a distribuição dos *training sets*. Finalmente, será dado ênfase às principais decisões tomadas no decorrer do desenvolvimento do projecto e em que medida estas influenciaram a concepção da solução final.
5. **Experiências e Resultados:** Neste capítulo serão apresentados os testes realizados e os resultados obtidos. Em primeiro lugar, serão apresentadas as métricas utilizadas na análise dos resultados. Depois especificar-se-á as configurações utilizadas nos testes. Por último, apresentar-se-ão os resultados dos casos de testes anteriormente especificados, demonstrando a viabilidade e eficácia da solução desenvolvida neste trabalho.
6. **Conclusões:** Este capítulo apresentará uma discussão crítica da globalidade do trabalho desenrolado, bem como dos resultados obtidos. Por fim, será apresentado o trabalho futuro, onde serão pormenorizados alguns aspectos a implementar numa futura continuação deste projecto.

Cada um do capítulos conta com uma pequena introdução, bem como uma conclusão sobre tudo o que foi discutido no mesmo.

No final, serão apresentados os anexos mencionados ao longo deste documento.

2 O Serviço de Correio Electrónico

O Correio Electrónico encontra-se cada vez mais presente no dia-a-dia de todos nós. De entre as suas diversas utilidades, destaca-se a comunicação entre familiares e amigos, i.e. lazer, e a sua utilização como ferramenta de negócios. Esta ferramenta tecnológica tem vindo a ganhar cada vez mais adeptos por ser fácil de manusear, barata, eficaz e, acima de tudo, universal. Por outro lado, cada vez mais surge quem se sirva deste meio para a difusão de publicidade não solicitada e correio maléfico (e.g.: *vírus* e *worms*), tirando proveito das fragilidades do sistema do utilizador para fins ilícitos. A isto, chamamos correio electrónico não solicitado (*spam*).

Neste contexto, de seguida irá ser especificada a estrutura de uma mensagem de correio electrónico e alguns dos protocolos envolvidos no envio e recepção da mesma, acompanhado por um exemplo de um caso real. Por fim, será então dada uma especial atenção à problemática do *spam*, descrevendo-se em que consiste e que soluções existem disponíveis para o combate a este problema.

2.1 Descrição Geral

Uma carta, dita tradicional, envolve duas estruturas físicas distintas: o envelope, com os dados do remetente e do destinatário, e a carta propriamente dita, que contém a mensagem. Todas estas especificações assentam em normas *standard.* Com o correio electrónico, tal como numa carta, acontece o mesmo, ou seja, existe uma estrutura normalizada baseada em protocolos também normalizados. Fisicamente, encontra-se também dividido em duas partes: o *header* e a mensagem, com funções bem definidas. De seguida será feita uma breve referência à estrutura das mensagens e aos protocolos envolvidos no serviço de correio electrónico.

2.1.1 Estrutura de um Email

Como foi dito anteriormente, uma mensagem de *email* é constituída essencialmente por duas partes: o *header* (ou cabeça do *email*) e o corpo da mensagem (ver anexo I para um exemplo de um *email* completo). O *header* contém, por exemplo, informação acerca de quem enviou a mensagem, para quem foi enviada (visto que uma mensagem pode ser enviada para um único destinatário ou para vários), o assunto, a data, *etc.* Esta informação está dividida em vários campos que podem ou não ser visíveis ao utilizador, visto que

nem todos os clientes de *email* possibilitam essa visualização. Estes campos são constituídos por palavras-chave (que indicam os nomes dos campos), seguidas por ":" e a informação desse campo. Existem diferentes tipos, dependendo da necessidade do servidor, do emissor, ou até mesmo do cliente do receptor. Algumas destas palavras-chave, para uma mensagem de texto simples, são obrigatórias, enquanto outras são opcionais. Por exemplo, é comum encontrar as seguintes palavras-chave num *email* [13]:

- *Received*: Indica a rota do *email* até ao destino, o sistema de cada nodo da rota, e o tempo que esteve nesse nodo.
- *Date:* Indica a data e hora à qual a mensagem foi enviada, incluindo a zona-horária, sendo obrigatório o preenchimento por parte do cliente de *email*.
- *From*: O emissor. Composto por duas partes, o nome do emissor, e dentro de "< >", o *email* do emissor. Este campo pode ser alterado pelo utilizador, logo pode não reflectir a verdadeira identidade do utilizador, mas, para que o receptor possa responder ou identificar o emissor, este campo é obrigatório.
- *Sender*: A identidade do emissor. Em alguns sistemas, este campo só será preenchido, se este for diferente do que se encontra no campo From.
- *To*: Uma lista de um ou mais de endereços de *email* para quem é enviado o *email*, sendo obrigatório pelo menos um *email*, visto que o servidor terá que saber a quem entregar.
- *Cc*: Endereço de *email* de todos os que também irão receber uma cópia da mensagem.
- *Subject*: O assunto que a mensagem terá, especificado pela pessoa que enviou.
- *Message-id*: Um identificador único gerado automaticamente. Por vezes, este campo, poderá ser utilizado para controlo de erros, para o caso de várias cópias da mensagem serem recebidas.
- *Reply-to*: Para que endereço de *email* a resposta será enviada (preferencialmente sobre o endereço presente no campo From). Poderá ser modificado pelo utilizador emissor, ou automaticamente inserido por alguns sistemas.

Poderão também ser adicionadas palavras-chave para realizar algumas funções específicas. Neste caso, a sintaxe usada será "X-<nome>:", em que "nome" será a palavra-chave pretendida.

No serviço de correio electrónico poderá também existir a necessidade de enviar ficheiros que poderão ser fotografias, binários, ou algum outro tipo não-textual. Neste contexto foi criado o *MIME* (do inglês *Multipurpose Internet Mail Extensions)*. Com esta extensão torna-se possível a transacção de mensagens que não incluam apenas texto, acrescentando ao *header* alguns campos, como por exemplo:

- *MIME-Version*: Indica a versão de *MIME* utilizada na mensagem.
- *Content-type*: Indica o tipo e o subtipo do conteúdo da mensagem, na forma Tipo/Subtipo, que poderá ser por exemplo text/plain, *multipart/mixed*, *multipart/alternative,* entre muitas outras formas.
- *Content-transfer-encoding:* Define os métodos usados para a transformação dos dados binários, utilizados na mensagem de *email*, em formato de texto (*ASCII*).

2.1.2 Protocolos Envolvidos no envio/recepção

Um protocolo é um método normalizado para o estabelecimento de uma comunicação fim-a-fim, para que as duas partes se entendam e a informação circule da melhor forma e com o mínimo de erros.

Para que exista a transferência de uma mensagem de *email* do receptor para o emissor, terá então que haver um protocolo envolvido. Aqui existem várias possibilidades, dependendo do objectivos em causa e das funcionalidades e vantagens/desvantagens que cada protocolo apresenta ao utilizador. Entre eles podemos destacar o *Internet Message Access Protocol (IMAP)*, o *Post Office Protocol* (*POP*) e o *Simple Mail Transfer Protocol* (*SMTP*). Também o protocolo *HyperText Transfer Protocol* (*HTTP*), apesar de não ser um protocolo puro de transferências de *email's (saindo um pouco do âmbito deste projecto)*, é bastante usado para aceder às caixas de correio e de lá enviar/receber *emails* e outras operações relacionadas. Este serviço é normalmente denominado como *WebMail*, e.g. o serviço *Web* que a Hotmail da *Microsoft* oferece.

O protocolo *IMAP* possibilita o acesso ao *email* e a manipulação de pastas e caixas de correio directamente no servidor remoto. Este é um protocolo cliente/servidor em que o *email* é transferido para o cliente e mantido no servidor. Devido ao facto de todos os *emails* não serem transferidos de uma só vez para o cliente de *email* do utilizador, mas sim só uma mensagem específica (escolhida pelo utilizador), este protocolo não utiliza grande largura de banda, funcionando bem mesmo em ligações de menor velocidade [14].

O protocolo *POP* fornece uma forma simples de acesso à caixa de correio para transferir todos *email*s disponíveis para o cliente de *email* do utilizador, possibilitando assim a leitura da mensagem mesmo quando este se encontra desligado da rede (*offline*) [15]. Neste caso, o utilizador não pode escolher as mensagens a transferir, o que faz com que o número de mensagens a transferir seja superior, tornando a transferências das mensagens mais lentas, podendo incluir mensagens de correio não solicitado (*spam* ou vírus). A única forma de manipulação que o protocolo consegue realizar junto do servidor é a opção de apagar o *email* após

a transferência ou deixar uma cópia no próprio servidor, sendo esta uma opção passível de ser definida pelo utilizador.

Visto que o *IMAP* e o *POP* têm o mesmo propósito, é natural existirem semelhanças entre estes dois. Mas existem também diferenças relevantes para a escolha do protocolo mais apropriado às necessidades do utilizador, tais como, o *IMAP* oferece mais funcionalidades na manipulação de *email* e caixas de correio enquanto o *POP é* mais simples de utilizar e encontra-se implementado pela maioria dos servidores de *email*, entre outras diferenças menos relevantes [16].

Enquanto que os protocolos *IMAP* e o *POP* são dedicados ao acesso remoto a caixas de correio, o protocolo *SMTP* é mais orientado para envio das mensagens. Este protocolo é usado pelos *Mail Transfer Agents (MTAs)* que estabelecem uma transmissão directa para o servidor do receptor ou, se esta não for possível, uma transmissão entre um ou mais servidores *SMTP* até ao destino [17].

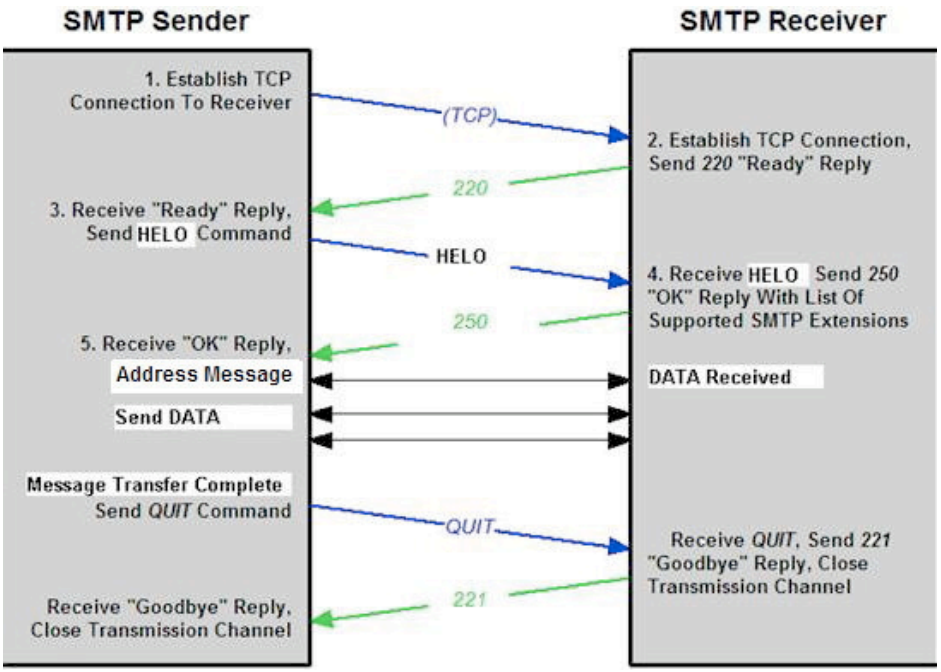

Figura 1. Processo de comunicação através do SMTP

A Figura 1[2] ilustra uma comunicação completa, entre o cliente emissor e o receptor ou servidor intermediário, usando o protocolo SMTP, para efectuar o envio de uma mensagem. A comunicação começa pelo estabelecimento de uma conexão *Transmission Control Protocol* (TCP), sendo seguida de uma mensagem "*Ready*" por parte do receptor. Após as duas partes estarem prontas, o emissor e o receptor enviam várias mensagem antes de começar o envio da mensagem de *email*, para saberem por exemplo, para

[2] http://telecomm.boisestate.edu/itm305l.fall.2008/SMTP.jpg

onde enviar a mensagem. A conexão é finalizada pelo "*QUIT*" do emissor e "*Goodbye*" do receptor, após o *email* estar enviado.

2.1.3 Do envio à recepção de mensagens

Sabendo então o que constitui uma mensagem de correio electrónico e de que modo esta é propagada do emissor para o receptor, segue-se então um exemplo com todos os passos dessa mesma difusão.

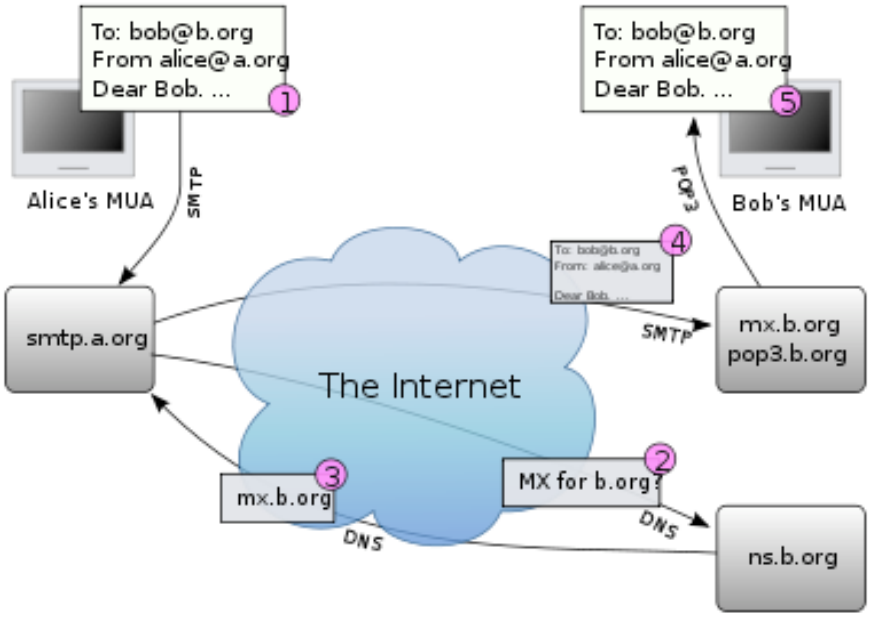

Figura 2. Exemplo de envio e recepção de um *email*

Como ilustrado na Figura 2[3], a Alice vai enviar um *email* com o texto "*Dear bob...*" para o Bob, sendo que cada um possui os endereços alice@a.org e bob@b.org, respectivamente. Para que isso seja possível, o cliente de *email* envia por *SMTP* a mensagem com os campos obrigatórios do *header* preenchidos, em que o campo "*To"* será o endereço de *email* do Bob, o campo "*From*" o endereço de *email* da Alice, e a mensagem aquilo que a Alice pretende transmitir ao Bob (identificado na Figura como 1). O servidor de *SMTP* da Alice recebe o *email*, e obtem o servidor de *email* associado ao endereço do Bob através de interrogações ao servidor de *DNS (Domain Name System)*, como se encontra ilustrado nos passos 2 e 3 da Figura. Após saber qual é o servidor de *email* do Bob, a mensagem é então encaminhada para este, também por *SMTP*,

[3] Imagem retirada de: http://www.answers.com/topic/e-mail

preservando toda a mensagem e *header*, passo 4 da Figura 2. Após a recepção, Bob poderá então aceder a essa mensagem a partir do seu servidor recorrendo ao uso dos protocolos *POP* ou *IMAP*. Supondo que o servidor de *email* de Bob só suporta ligações *POP* para o acesso à sua caixa, o Bob vai proceder à transferência da mensagem para o seu cliente usando esse protocolo, identificando-se junto dele através do recurso ao seu nome de utilizador e palavra-passe (como é ilustrado em 5 na Figura 2). Assim o Bob poderá ler e responder à mensagem que a Alice enviou.

2.2 Correio electrónico não solicitado *(spam)*

Um utilizador que utilize o *email* com elevada frequência, geralmente recebe 2 grupos distintos de mensagens: as que são solicitadas (denominadas *ham*) e as que não são solicitadas (também chamadas *spam*, como já foi referido anteriormente). As mensagens solicitadas são aquelas que o utilizador espera receber, ou até mesmo não esperando receber, são mensagens que o utilizador quererá ler e de importância para ele. Por oposição, as mensagens ditas *spam* ou não solicitadas, são mensagens de publicidade contendo por vezes vírus, *worms,* ou algo maléfico para o sistema, distribuídas em massa, sobre as quais o utilizador não tem controlo relacionado com a sua recepção, i.e., o utilizador não poderá sair da lista de envio do emissor.

O *spam,* geralmente enviado e controlado por pessoas denominadas *spammers*, tem como principais objectivos:

1. A divulgação para venda de produtos por vezes ilícitos, como medicamentos (de prescrição obrigatória) ou drogas;
2. Os actos de *phishing,* i.e.*,* fraude electrónica com o intuito de obter informações pessoais tais como *passwords* ou números de cartões de crédito, fazendo-se passar por empresas de confiança;
3. A intrusão no sistema do utilizador, para que seja roubada informação deste ou, por vezes, usar o sistema como porta para distribuição de mais *spam*.
4. Destruição do sistema do utilizador por via de vírus.

Estas intenções, por parte dos *spammers,* na distribuição do *spam* atingem não só os utilizadores (como alvo primário), mas também os *ISP's*, por serem as entidades responsáveis por controlar os servidores *SMTP's* que vão realizar a entrega das mensagens.

2.2.1 O problema do *spam*

O *spam* está a crescer de dia para dia, afectando todos os que utilizam o serviço de correio electrónico, tornando-se um problema sério. Por um lado, os utilizadores desperdiçam demasiado tempo e largura de banda a transferir e verificar essas mensagens, são contaminados e enganados pelo conteúdo delas e perdem espaço de armazenamento no servidor de *email* (que muitas vezes é limitado). Por sua vez os *ISPs* também são afectados. Estes têm que dispensar espaço de armazenamento e largura de banda, com custos associados, para que as mensagens possam ser guardadas e, sendo estas mensagens do tipo *spam*, estes custos são desnecessários.

Para perceber a gravidade deste problema, cerca de 40% das mensagens de *email* que circularam na Internet durante o ano de 2002 eram *spam* [10], mas este número cresceu significativamente até ao final de 2003, atingindo os 60% [1]. Nos dias de hoje o *spam* corresponde a cerca de 75%/80% de todo o tráfego de mensagens de *email* [9]. Em 2003, estimou-se que foram distribuídas cerca de 76 biliões de mensagens *spam*, significando que foram precisos cerca de 10 *petabytes* de espaço em disco nos servidores de *email* [1], tendo estes despendido cerca de 10 biliões de euros (em aumento de espaço de armazenamento e largura banda) [10, 18, 19]. Mas em 2008, os valores despendidos pelos *ISPs* para os mesmos efeitos, foram estimados na ordem dos 50 biliões de euros [9]. Estes valores reflectem-se sempre no utilizador, visto que é este o destinatário. Em média, um utilizador passou de uma recepção diária na ordem dos 3.7 mensagens de *spam*, em 2001, para 6.2 por dia [18]. Como é natural, a estes números estão associados custos para o utilizador, que por ano na recepção de *spam* atingem os 30 euros [10]. Estes custos são derivados da necessidade de maior largura de banda para a transferência das mensagens e espaço de armazenamento no cliente do utilizador.

O *spam* tem um custo baixo para o *spammer*, visto que são utilizadas técnicas que permitem enviar mensagens em quantidades massivas em reduzido espaço de tempo, custo de envio baixo e indetectáveis. Isto faz com que seja o método preferencial para a distribuição e concretização dos objectivos dos *spammers* (como referido anteriormente). Existem muitas técnicas que os *spammers* usam para enviar *spam* e para evitar certas protecções (na próxima secção serão explicadas algumas das soluções que os *spammers* encontram para evitar cada filtro). Algumas dessas técnicas de envio são [20]:

- *Spam* Directo: Utilização, por parte dos *spammers*, de *ISP's* que não tem politicas de Controlo de *spam*, ou que não aplicam nenhuma penalização pela sua prática.

- *Open Relays: Open* relays são servidores de *email* (geralmente *SMTP*) em que não é necessária autenticação para enviar mensagens. Normalmente são explorados pelos *spammers* para o envio de mensagens *spam* porque mantém o anonimato e são de fácil acesso.
- *Botnets:* Este é o método preferido dos *spammers*. As *Botnets* são um conjunto de máquinas controladas por um nodo central. Esta máquina central, geralmente controlada pelo *spammer*, usa as restantes máquinas para o envio das mensagens. Normalmente, os nodos "filhos" são criados por intermédio de *worms* que se auto-propagam e permitem que o *spammer* as use como *mail relay*. Um exemplo deste *worm* é o "*Bobax"*. Este autopropaga-se por *email* para outras futuras vítimas. Outros *botnets*, muito utilizados pelos *spammers,* são os *"Agobot"* e *"SDBot"* [21].

Para combater este problema, existe um investimento crescente por parte dos *ISPs* e dos utilizadores mais especializados na área informática. Ambas as partes esforçam-se para encontrar formas de tornar ineficazes todos os métodos usados pelos *spammers* e desencorajar a prática de *spam.* Por parte dos *ISP's,* estes tentam encontrar não só uma forma de tornar o *spam* mais dispendioso, mas enquanto não vai sendo possível, tentam filtrar o *spam* antes deste chegar ao utilizador. Por parte do utilizador, já só existe um esforço de filtrar o *spam* que chega do servidor, para que não perca tempo a ver *emails* que não quer, ou mesmo evitar ser infectado por vírus. Esta classificação apresenta dois problemas associados, os denominados falsos positivos (FP) e os falsos negativos (FN). Os falsos positivos são os *email*s que são filtrados como *spam* erradamente, i.e., *ham* que é classificado pelos filtros como *spam.* Este problema faz com que o utilizador não leia uma mensagem se não estiver atento à pasta do filtro de *spam*. Este é o motivo pelo qual o *ISP* não pode apagar nenhuma mensagem classificada pelo filtro, deixando essa tarefa para o utilizador final. Os falsos negativos são o oposto, sendo o *email* classificado pelo filtro como *ham* quando na realidade é *spam.* No caso dessa classificação ocorrer por parte do utilizador isso implicará uma perda de tempo a ver o *email*, ou ainda pior, durante essa visualização o utilizador poderá correr o risco de ser infectado por um vírus. Se essa classificação ocorrer no *ISP,* as consequências não serão tão graves, visto que poderá ser filtrado ainda pelo filtro do utilizador quando este transferir a mensagem. Estima-se que a função de probabilidade do custo de um falso negativo em relação a um falso positivo é dada por [8]:

$$PCF = \frac{p(spam) \times const(FN)}{p(spam) \times \cos t(FN) + p(legitimo) \times \cos t(FP)} \quad (1)$$

Onde cost(FP)> cost(FN).

O factor que mais contribui para o aumento dos falso negativos são as permanentes mudanças de técnicas dos *spammers*. Estes, sempre que descobrem fragilidades nos filtros, mudam os esquemas de envio ou mesmo a estrutura da mesma mensagem. Estas permanentes alterações, fazem com que o filtro tenha que se adaptar, e enquanto isso, vai permitindo que seja possível a errada classificação.

Na próxima secção serão especificadas as técnicas *anti-spam*, incluindo os métodos e os algoritmos usados nos vários filtros existentes, bem como referencia à forma como os *spammers* alteram os seus métodos para que possam ter sucesso nos seus objectivos.

2.2.2 Técnicas *anti-spam*

As técnicas *anti-spam* são vastas e estão em permanente mudança, face às alterações das técnicas dos *spammers*. Existem soluções de diferente natureza que incluem processos judiciais aplicados em alguns países para penalizar quem os envia *spam*, tentativas de aumentar o custo do *spam* para que se torne inviável o seu envio, e até mesmo o recurso a filtros *anti-spam* já referidos na secção anterior.

Em alguns países já se começa a implementar leis para penalizar criminalmente a prática de *spam*. Por exemplo, em Portugal já é considerado crime desde 2004, visando a proibição do uso de *spam* para propostas de marketing directo, excepto se o destinatário der consentimento[4]. Mais cedo, em Julho de 2002, foi aprovada uma directiva de Privacidade e de Comunicação Electrónica pela União Europeia[5] [9], sobre a qual a lei Portuguesa se baseia. Nos Estados Unidos, também existe uma lei similar denominada *"Controlling the assault of non-solicited pornography and marketing act"* (controle do ataque da pornografia e publicidade não solicitada)[6]. Neste país já se está a aplicar com sucesso a lei. Por exemplo, a AOL já identificou vários *spammers*, e já moveu mais de 40 processos civis contra eles na Virgínia, tendo já obtido ordens de restrições, injunções judiciais e até mesmo acordos com os *spammers* [1, 18]. Num outro caso, em 2003 no Estado da Califórnia, foi anunciada a vitória de um processo movido por esse mesmo estado [7]. Estes casos demonstram um sucesso relativo, visto que, em comparação o número de *spam* (e por sua vez, *spammers*), são insignificantes, mas não deixam de ser medidas preventivas para o desincentivo da prática por meio do receio a processos criminais. O grande problema na aplicação da lei é a identificação da real identidade dos *spammers*, visto que eles usam técnicas para preservar o anonimato.

[4] Decreto-Lei nº 7/2004 de 7 de Janeiro, Artigo 22º em http://www.iapmei.pt/iapmei-leg-03.php?lei=2626

[5] Directiva 2002/58/EC

[6] Como noticia o site http://www.computerworld.com.pt/content/view/252/53/

[7] Como noticia tambem o site http://www.computerworld.com.pt/content/view/252/53/

Uma outra técnica seria impor custos acrescidos ao *spam*. Estes custos poderiam ser baseados, por exemplo, em pagamentos para ler/enviar *emails* com quantias simbólicas em dinheiro "virtual", aumentando assim o custo do envio e tornando menos atractivo o envio do *spam* [10]. Um grande problema é a adesão do utilizador tanto ao conceito do pagamento como ao dinheiro virtual. Este problema de adesão do utilizador comum faz com que a aplicação de um sistema baseado em pagamentos monetários seja difícil, ou até mesmo impossível. Um outro exemplo poderia ser a introdução de uma função de custo, que quando aplicada à mensagem durante o seu envio, originaria uma pequena latência de computação. Embora essa latência introduzida seja pequena para o envio de um só *email*, para um grande número de mensagens enviadas em simultâneo, faria com que a latência introduzida fosse maior e, por isso, tornaria ineficaz e não tão atractiva a prática de *spam* [22]. Um grande problema com esta função, seria no envio em massa de *email's* que não fossem *spam* (e.g. a distribuição de um *email* de uma *mailing list*). Esta situação pode ser evitada com o recurso a listas autorizadas de *email* (as chamadas *whitelists*), que ao identificar esse endereço de *email*, permitem o envio do *email* sem nenhuma latência. No entanto, se um novo *email* fosse adicionado à *mailing list*, iria ser introduzida a latência novamente, o que se tornaria novamente problemático. Um outro problema encontra-se na grande capacidade de processamento que os *spammers* possuem. Isto faz com que essa latência, mesmo quando são enviados *email's* em massa, seja reduzida. Aumentando então esse tempo de latência, haveria o problema de utilizadores com fraca capacidade de processamento terem tempos de envio grandes, mesmo enviando uma só mensagem. Este necessário balanceamento, torna a função de custo de difícil implementação, mas não impossível. De momento, já se encontram disponíveis ferramentas que efectuem este aumento de custo eficazmente. Por exemplo, HashCash[8] e CAMRAM[9], já estão disponíveis *on-line* e gratuitas para *download*.

Estas duas técnicas apresentadas têm falhas (algumas delas já referidas) e são difíceis de colocar em prática. Aqui surgem então os filtros como ferramenta alternativa. Estes facilitam a distinção entre o que é *spam* e do que *ham*, tanto por a análise do conteúdo do *email* (métodos heurísticos), como por testes a listas ou repositórios globais de *spam* (métodos colaborativos). Os métodos colaborativos, tipicamente, dependem de uma rede de utilizadores que criam em conjunto uma lista, sob forma de repositório, na qual constam vários endereços/domínios ou assinaturas de mensagens identificadas como sendo *spam* [3, 9, 19, 20] (e.g. as *blacklists* globais). Sempre que um utilizador recebe uma nova mensagem, é calculada a assinatura (em forma de *hash*) para que a privacidade seja assegurada. Esta assinatura é então enviada para um servidor, que procede à comparação desta com todas as existentes na sua lista. Caso a assinatura do *email* se

[8] Disponível em: http://www.hashcash.org/

[9] Disponível em: http://sourceforge.net/projects/camram/

encontre no repositório, será então enviado ao utilizador um aviso de *spam* para essa mesma mensagem [22]. O grande problema subjacente a este método é a grande susceptibilidade a alterações das mensagens de *spam* [3]. Como as assinaturas das mensagens dependem dos caracteres (basta a mudança de um para gerar uma assinatura diferente), os *spammers* inserem caracteres aleatórios nas mensagens enviadas para que altere a sua assinatura fazendo assim com que a sua assinatura não conste da lista sendo, então, identificada correio *ham* e gerando um falso negativo. Em contrapartida, dado o facto deste método utilizar uma rede global de utilizadores, basta um único utilizador identificar uma mensagem como *spam* para que toda a restante rede de utilizadores beneficie dessa *tag*, sendo as próximas mensagens com assinaturas compatíveis identificadas automaticamente. Actualmente existem disponíveis para *download* ferramentas que permitem a implementação deste método colaborativo. Um exemplo é o "*Spamato*"[10] que oferece varias soluções dependentes do sistema operativo e do cliente de *email* que o utilizador possui.

Os filtros que tomam por base métodos heurísticos são, regra geral, locais e procuram determinadas características nas mensagens que permitem distingui-las, i.e., este método assume a possibilidade de encontrar diferenças entre as mensagens *ham* e *spam* [10]. Existem vários métodos heurísticos baseados em diferentes características. Entre eles distinguem-se os baseados na origem da mensagem, os baseados na análise do tráfico e os baseados no conteúdo da mensagem [10].

A classificação baseada na heurística da origem da mensagem ocorre sempre que o cliente de *email* do utilizador começa a transferir a mensagem, usando o *header* para tal. Um exemplo que recorre a este tipo de heurística são as chamadas *blacklist* locais. Uma mensagem quando começa a ser transferida, é-lhe retirada o campo "*FROM:*" comparando-o com cada elemento de uma determinada lista. Caso o endereço conste dessa lista, a mensagem fica automaticamente assinalada como *spam*. Ao utilizador cabe adicionar endereços de *email* das mensagens que são *spam,* à medida que as mensagens conseguem ultrapassar a barreira estabelecida pelo filtro, i.e., o utilizador é o responsável pela manutenção dessa mesma lista. Os *spammers*, para contornar as *blacklists*, usam servidores *SMTP* fidedignos e que ainda não fazem parte das listas estipuladas. Um outro exemplo são as *whitelists* locais*,* que se comportam de forma oposta às *blacklists*. O utilizador, em vez de adicionar os endereços dos *spammers*, adiciona os *email*s da sua confiança, i.e., das mensagens *ham*. O maior problema da utilização deste tipo de listas é a grande taxa de falsos positivos, que como se viu anteriormente, irá provocar custos acrescidos bem como consequências mais nefastas na óptica do utilizador [10].

O filtro baseado na heurística da análise de tráfego é orientado mais para o servidor *SMTP* ou *ISP* do que para o utilizador. Através de análise de registos do tráfego de *SMTP*, identificam, quando a mensagem

[10] Disponível em: http://www.spamato.net/

chega ao servidor ou é enviada, se o emissor da mensagem está a enviar em difusão, para então poder ser classificada essa mesma mensagem [10]. Um problema associado a este método, tem a ver com situações nas quais uma mensagem de *email* é enviada por uma *mailing list*. Por esta ser um método de distribuição em difusão, apesar de não constituir uma ameaça, vai ser reconhecida pelo filtro como sendo uma mensagem de correio não solicitado associada a um potencial perigo de conteúdo para quem recebe.

Por último, a classificação baseada no conteúdo da mensagem, ocorre após recepção total do *email*. A classificação é então efectuada pela identificação das características contidas na mensagem [10], tais como palavras, imagens, anexos ou algum outro conteúdo presente que possa levantar suspeitas, pela utilização de técnicas *data mining*. Deste modo, o filtro é capaz de perceber se *email* é *spam* marcando-o como tal. Para que seja possível a análise e a identificação, o filtro vai ter que aprender a distinguir essas características. Isto é realizado pela correcção, por parte do utilizador, de mensagens que foram classificadas como *spam* ou *ham* erradamente. Quando isto acontece, o filtro procede ao reconhecimento das características dessa mensagem bem como à assimilação das características, para que no futuro, mensagens que apresentem características semelhantes sejam logo identificadas e classificadas correctamente. Isto tudo só é possível pela aplicação de uma função de decisão do estilo [9]:

$$f(m,v) = \begin{cases} cSpam, & se\ decisão\ é\ spam \\ cHam, & caso\ contrário \end{cases} , (2)$$

com *m*, a mensagem a classificar, *v* o vector de parâmetros e *cSpam* e *cHam,* a classificação final da mensagem. O vector de parâmetros *v* pode então ser definido [9]:

$$v = \{ (m_1,y_1), \ldots , (m_n,y_n) \}, y_i \in \{cSpam,cHam\} , (3)$$

com $m_1, m_2,\ldots,m_n$ mensagens que foram usadas para o treino do filtro, e $y_1, y_2,\ldots,y_n$ as características dessas mensagens. Existem vários algoritmos de *data mining* que podem ser utilizados para a implementação da função de classificação e aprendizagem. O mais conhecido e utilizado é o algoritmo *Naive Bayes*[11] (NB). Quando se utiliza este algoritmo, o utilizador terá que treinar o filtro com *ham* e *spam,* até que este comece a distinguir correcta e autonomamente as mensagens que recebe. Geralmente, esta aprendizagem é efectuada através do número de ocorrências de cada palavra da mensagem, definindo assim o conjunto de

[11] Em homenangem a Thomas Bayes (matématico Britanico do sec. XVIII). Este algoritmo foi implementado pela primeira vez num filtro de *email* em 1996.

características da mensagem. Quando um *email* chega, o filtro corre o algoritmo e comparando com o conjunto de treino, classifica a mensagem através da avaliação de algumas palavras pertencentes à mensagem, resultando para cada uma probabilidade de ser *spam* [8-10, 19]. Esta probabilidade é calculada pela seguinte função [3, 10, 12, 23]:

$$\Pr(S \mid W) = \frac{\Pr(W \mid S) \times \Pr(S)}{\Pr(W \mid S) \times \Pr(S) + \Pr(W \mid H) \times \Pr(H)} \quad (4)$$

em que Pr(S|W) é a probabilidade de a mensagem ser *spam* através de uma dada palavra *W*, Pr(S) a probabilidade geral de uma qualquer mensagem ser *spam*, Pr(W|S) a probabilidade de uma palavra *W* estar presente em mensagens *spam*, Pr(H) a probabilidade geral de uma qualquer mensagem ser *ham* e Pr(W|H) a probabilidade de uma palavra *W* aparecer numa mensagem de *ham*. A palavra *W* pertence a um conjunto formado por todas as palavras "interessantes" da mensagem recebida, ou seja [10]:

$$M = \{w_1,\ldots,w_n\} \text{ e } W \in M, \quad (5)$$

em que Pr(w$_i$) ≈ 1 ou Pr(w$_i$) ≈ 0. Por exemplo, a palavra "ff0000" (que é uma *tag html* para especificar a cor vermelha) aparecerá varias vezes nas mensagens de *spam* para dar especial atenção a certos produtos que o *spammer* quer publicitar, logo Pr(ff0000) ≈ 1 sendo então esta uma palavra "interessante". Então, para as *n* palavras "interessantes" da mensagem, a probabilidade total de esta ser *spam* será dada pela seguinte função:

$$\Pr(M) = \frac{\sum_{i=0}^{n} \Pr(W_i)}{n}, \quad (6)$$

com *M* o conjunto de palavras referidas anteriormente e *Pr(w$_i$)* a probabilidade de ser *spam* para a palavra *w$_i$* ∈ *M,* calculada por *(4).* A vantagem deste algoritmo é a possibilidade do utilizador adaptar o filtro à sua maneira através do treino [19]. Por exemplo, para um informático a palavra "*drugs*" terá maior probabilidade no *spam*, enquanto que para um profissional de saúde, terá maior no *ham*. Ou seja, o mesmo filtro será treinado de maneira diferente para a mesma palavra nas situações indicadas. Por outro lado, este filtro é susceptível a técnicas tais como o *"Bayesian poisoning"* (do inglês, envenenamento bayesiano) e à troca de texto por imagens (após o filtro ter sido treinado). O envenenamento bayesiano, usado pelos *spammers*, tenta degradar a eficácia do filtro, mandando vários *emails* de *spam* com palavras legítimas, e com isto, o filtro será

treinado de maneira errada e permitindo assim a má classificação e um maior número de falsos positivos. Uma aplicação do algoritmo Naive Bayes no combate ao *spam*, é o filtro *SpamBayes*[12], que é *open-source* e de acesso livre.

Existem também algumas variantes do algoritmo *Naive Bayes* [12], tais como Multi-Variável*l Bernoulli*[13] NB (MVBNB), o Multi-Variável *Gauss*[14] NB (MVGNB) e o *Bayes* Flexível (BF). A variante MVBNB usa todas as palavras da mensagem para calcular a probabilidade, em vez de um conjunto de palavras "interessantes". Neste caso, é calculado um vector binário (*Bit)* a indicar se cada palavra do treino *V* está presente ou não na mensagem recebida *d*, em que 0 indica que a palavra do treino, W_t, não está presente em *d* e 1 caso contrário [12]. Após o cálculo desse vector será calculada a probabilidade de *d* ser *spam* pela seguinte equação [24]:

$$P(d_i \mid c_j;\theta) = \prod_{t=1}^{|V|}(B_{it}P(w_t \mid c_j\ ;\ \theta) + (1 - B_{it})(1 - P(w_t \mid c_j\ ;\ \theta)))\ ,\quad (7)$$

com $P(W_t \mid cj;\ \Theta)$ sendo a probabilidade de a palavra W_t do conjunto de treino *V* ser *spam*. A variante MVGNB segue uma distribuição normal *g* defendida por [12]:

$$g(x_i\ ;\mu_{i,c},\sigma_{i,c}) = \frac{1}{\sigma_{i,c}\sqrt{2\pi}}\, e^{-\frac{(x_i - \mu_{i,c})^2}{2\sigma_{i,c}{}^2}}\ ,\quad (8)$$

em que $c \in \{Spam, Ham\}$, x_i é a palavra da mensagem recebida e $\mu_{i,c}$, $\sigma_{i,c}$ são estimados pelo treino de *c*.

Seja *x* o conjunto formado por todas as palavras da mensagem, então a probabilidade de ser *c,* usando o algoritmo MVGNB é dada por [12]:

$$p(\vec{x} \mid c) = \prod_{i=1}^{m} g(x_i\ ;\mu_{i,c},\sigma_{i,c})\ ,\quad (9)$$

[12] Disponível em: http://spambayes.sourceforge.net/

[13] Matemático Suíço do sec XVII, no qual deu o nome á Distribuição de Bernoulli.

[14] Físico matemático Alemão do sec XIX.

em que *m* é o número total de palavras presentes na mensagem de *email*. Por fim, o BF também usa uma distribuição normal *g*, com diferentes valores, mas com o mesmo desvio. Então para uma palavra x_i, a probabilidade é definida da seguinte maneira, como se sugere em [12]:

$$p(x_i \mid c) = \frac{1}{L_{i,c}} \cdot \sum_{l=1}^{L_{i,c}} g(x_i ; \mu_{i,c,l}, \sigma_c) \ , \quad (10)$$

em que $L_{i,c}$ é o número de diferentes valores de x_i.

As opções para o combate ao *spam* são as mais variadas. Algumas mais falíveis, outras mais lentas mas com menos erros e ainda outras mais generalistas e usadas somente pelos *ISP's*. A classificação, por ser um método que envolve uma aprendizagem local e personalizada, é de momento uma das opções mais utilizadas para combater o *spam.*

2.3 Conclusões

Este capítulo começou por abordar alguns conceitos genéricos inerentes ao serviço de correio electrónico. Neste contexto foram discutidas questões que vão desde a composição das mensagens até aos protocolos envolvidos na sua difusão. De seguida abordou-se em pormenor a problemática do correio electrónico não solicitado.

Por vezes as mensagens enviadas são para efeitos de publicidade ou maléficos, entrando aqui o conceito de *spam* ou de correio não solicitado. Por ser mau para o utilizador e para o *ISP* (por perda de tempo, dinheiro, largura de banda e espaço de armazenamento), estão a ser criados não só pelos *ISP's*, mas também pelos políticos e utilizadores, métodos que permitem evitar e filtrar todas as mensagens *spam*. Os métodos são variados e vão desde medidas judiciais que visam punir os *spammers* como forma passiva de combate ao *spam,* ao aumento de custos de envio de mensagens tornando o *spam* caro e como tal pouco rentável *e* até filtros que são capazes aprender a distinguir uma mensagem *spam* de uma mensagem *ham* através de modelos de treino específicos. Como nenhum dos métodos até então apresentado é infalível, os *spammers* estão de dia para dia a intensificar o envio e a aperfeiçoar as técnicas utilizadas para contornar as barreiras existentes para o combate ao *spam.* A investigação nesta área não pode parar encontrando-se mesmo em expansão, o que faz com que surjam constantemente métodos alternativos, novos métodos e ferramentas de complementaridade de métodos antigos para tentar por um fim a esta "praga virtual".

3 Clientes de email e Tecnologias de Desenvolvimento

Os clientes de *email* constituem actualmente uma ferramenta essencial que é utilizada pela grande maioria dos intervenientes na rede Internet. Através destas aplicações, o utilizador pode aceder à sua caixa de correio de uma forma simples, intuitiva e, normalmente, através de um ambiente gráfico. Com a natural evolução e com o aumento das exigências dos utilizadores, o cliente de *email* ganhou a possibilidade de, não só aceder à caixa de correio, mas como também guardar contactos de outros utilizadores. Assim, o utilizador poderá manter os seus contactos organizados, e quando quiser enviar uma mensagem de *email* saberá sempre o endereço. Também foram implementados sistemas capazes de aceitar o desenvolvimento de extensões desenvolvidas pelas empresas do ramo, ou até mesmo pelos utilizadores. Estas extensões possibilitam colmatar as diferentes exigências que cada utilizador poderá ter, ou ate mesmo realizar operações automáticas para as quais o cliente de *email* não estava preparado.

Neste contexto, este capítulo começará por analisar alguns dos clientes de correio electrónico existentes, discutindo também as vantagens e desvantagens associadas, quer para os utilizadores quer para o contexto deste projecto. Por fim, serão explicadas algumas das tecnologias usadas neste projecto para o desenvolvimento de extensões para um cliente específico de *email*.

3.1 Clientes de email

Actualmente existem muitos clientes de *email,* uns proprietários mas gratuitos, outros que não são gratuitos, e ainda outros *open-source*. A maior parte deles tem uma interface gráfica para maior facilidade do utilizador, mas outros, para os mais experientes, são baseados exclusivamente em ambiente de texto, i.e., são em linha de comandos. Estes são normalmente dedicados à leitura e escrita de mensagens de *email* sendo por isso mais leves, mas mais complicados de operar. Os clientes que se baseiam em ambientes gráficos, distinguem-se pelas funcionalidades que apresentam, pela sua disponibilidade que para uma determinada plataforma, pelo desempenho que cada um possui, mas sobretudo pela interface gráfica. Alguns apresentam funcionalidade básicas, tais como, agenda pessoal para marcar compromissos e outras actividades e lista de contactos (que podem não ser só de endereços de *email*, mas também números de telemóvel e morada). Mas estas não são as únicas características, certos clientes apresentam também bloco de notas, capacidades de ligação a servidores que não sejam de *email* (tais como a servidores de ficheiros ou *web*), ligação e sincronização com dispositivos móveis ou nodos centralizados, bem como muitas funcionalidade que não só ajudam o utilizador nas tarefas diárias, mas também facilitam a sua vida.

Na escolha de um cliente também é necessário terem atenção o sistema operativo em que este opera. Por exemplo existem clientes de *email* só para o sistema operativo da *Microsoft,* enquanto outros só para os da *Apple* e ainda outros que são multi-plataforma. Entre os mais populares, destacam-se o *Outlook* da Microsoft, o *Mail* da *Apple* e o *Thunderbird* da *FireFox*, visto que todos eles possuem características similares e filtros anti-*spam.*

De seguida será descrito, para estes clientes de *email* mais populares, as suas funcionalidades, no que se diferenciam, bem como as suas vantagem e as suas desvantagens.

3.1.1 Outlook

O *Outlook*, proprietário da Microsoft e criado em 1997, é um dos clientes de *email* mais populares, estando disponível para *Microsoft Windows* e para *Mac OS X*, embora seja mais utilizado para *Windows*. É usado por mais de 200 milhões de pessoas todos os dias, não só para a visualização e envio de *email*, mas também na organização da vida e trabalho de cada um. O *Outlook* incorpora várias funcionalidades tais como, marcação de tarefas onde se pode organizar, em forma de lista, os detalhes de uma actividade a realizar, calendário onde se pode agendar os compromissos que o utilizador tenha a uma determinada hora e local, e um organizador de contactos onde se pode guardar os detalhes de contacto (tal como nome, morada, número de telefone ou endereço de *email*) de um outro utilizador [25]. Tudo isto é sincronizável para dispositivos móveis ou servidores que disponham tais serviços compatíveis. Por possuir estas (e muitas outras) funcionalidade, o *Outlook* é considerado por muitos um "canivete suíço" digital [26].

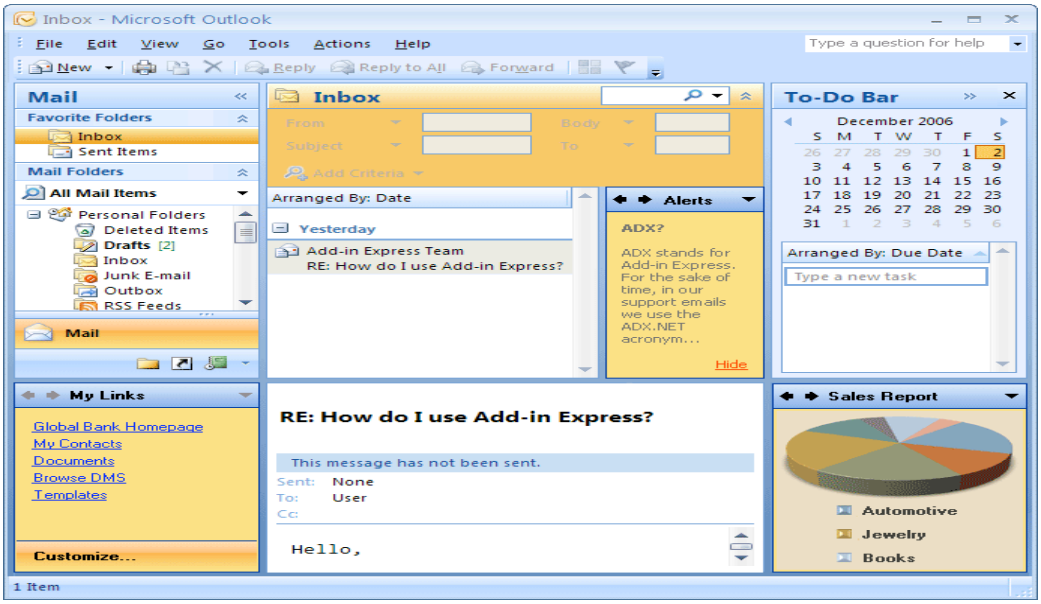

Figura 3. Microsoft Outlook 2007

Na Figura 3, identifica-se a janela principal do *Outlook*, na qual se destacam algumas das funcionalidades enunciadas anteriormente. Este cliente é parte de um conjunto de ferramentas que a mesma companhia comercializa chamado *Microsoft Office*, e já conta com várias versões. Um dos problemas que este cliente apresenta é que, embora muito completo e muito utilizado, não é gratuito. Por esta razão, o cliente apresenta um código fechado, constituindo assim um problema para este trabalho. A *Microsoft* também disponibiliza um cliente de *email* gratuito (embora de código fechado), mas com reduzidas funcionalidades. Este cliente é chamado de *Microsoft Outlook Express* e é distribuído com o sistema operativo do fabricante, apresentando somente funcionalidades básicas, tais como, o acesso à caixa de correio (para leitura e escrita), lista de contactos simples e acesso a grupos de notícias (*newsgroup).* Na distribuição do sistema operativo *Windows Vista*, o *Outlook Express* passou a ser chamado de *Windows Mail*, sendo a principal diferença o ambiente gráfico melhorado. Para além de todas as funcionalidades descritas, as actuais versões disponíveis do Outlook (incluindo o *Express* e o *Mail)* suportam a gestão de várias contas de *email* em simultâneo e os protocolos *POP3* e *IMAP [7]*.

3.1.2 *Mail*

O cliente de *email Mail* (ou *Apple Mail*), desenvolvido pela *Apple Inc.* em 2000 é parte integrante do sistema operativo da mesma companhia, o *Mac OS X,* actualmente na versão 10.6 (ou *Snow Leopard)*.

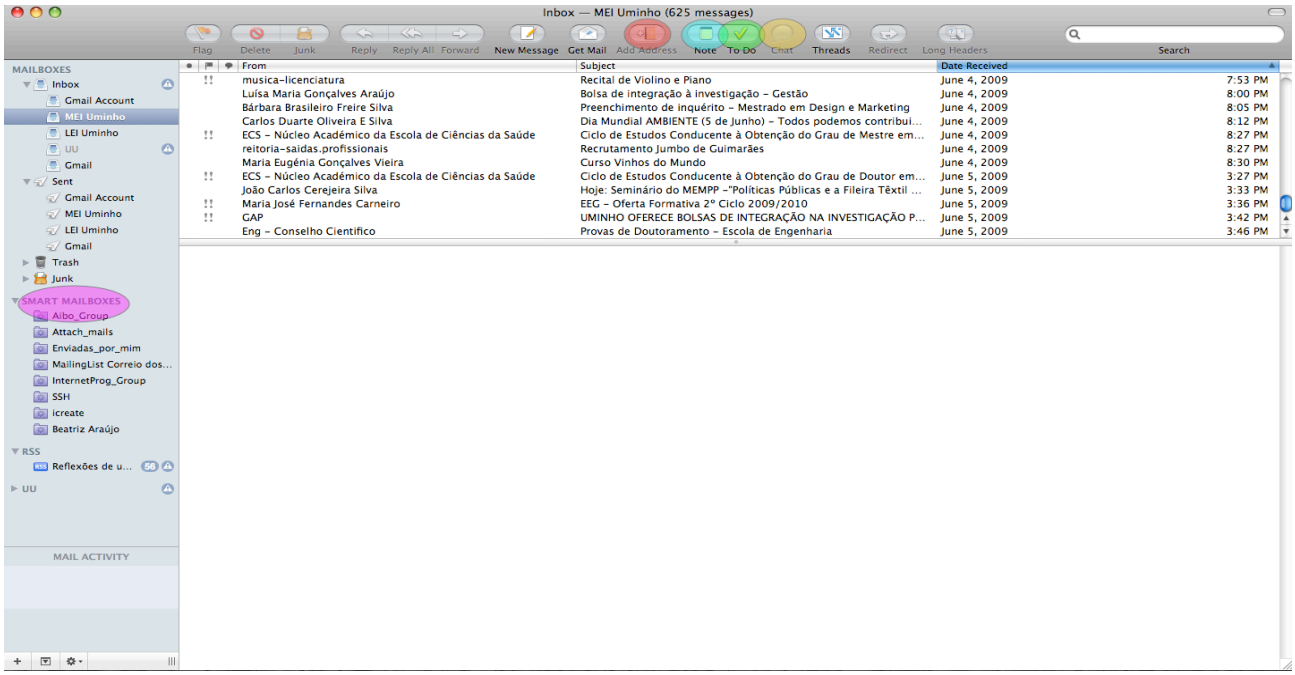

Figura 4. Apple Mail 3.6

A Figura 4 ilustra a última versão do cliente de *email* anteriormente referido. Este integra várias ferramentas do sistema operativo tais como *Address Book* (em português, livro de endereços), assinalado a vermelho, que guarda e ordena todos os contactos do utilizador, incluindo morada, telefone e outras informações dos contactos, *iChat* (na figura a amarelo), programa de conversação que a companhia desenvolveu e *iCal,* agenda pessoal desenvolvida pela *Apple,* que permite a marcação de eventos, reuniões, etc., importantes ao utilizador. Além da inclusão destes programas, o cliente possui também: caixas inteligentes (denominadas por "*Smart Mailboxes*" como representa a Figura 4 em cor-de-rosa), que organizam as mensagens de maneira automática e inteligente mediante um conjunto de regras dadas pelo utilizador; um bloco de notas (também chamado por "*Notes"),* a azul na imagem, que permite ao utilizador tirar notas de algo digitalmente; lista de tarefas, representado na Figura 4 a amarelo, permitindo ao utilizador a anotação de certas tarefas ou afazeres que poderá ter. É também possível realizar a sincronização de toda esta informação para dispositivos móveis, tais como o *iPhone*, ou para servidores especializados, como os da *Yahoo! Contacts*. É possível também com este cliente o uso de várias contas de *email* em simultâneo bem como o uso de protocolos como o IMAP e o POP3.

Tal como o *Outlook*, este cliente é proprietário e, neste caso, comercializado pela *Apple* (embora seja acompanhado com o sistema operativo), sendo por isso de código fechado. Uma outra desvantagem que apresenta é a sua implementação exclusiva em sistemas operativos *Mac OS X*.

3.1.3 *Thunderbird*

O *Thunderbird* (também conhecido por *Mozilla Thunderbird*) é um cliente de *email* gratuito e *open-source* desenvolvido pela *Mozilla Foundation* em 2003. Este cliente evoluiu de uma aplicação desenvolvida pela mesma companhia, o *Mozilla Application Suite* [7].

Actualmente o *Thunderbird* (Figura 5) é suportado pelos mais utilizados sistemas operativos, tais como *Windows, Linux* e *Mac OS X*. Como os restantes clientes de *email*, este também suporta múltiplas contas de *email* bem como os protocolos POP3, IMAP e SMTP. Em termos de segurança, o *Thunderbird* implementa algoritmos de encriptação SSL/TLS (*Secure Sockets Layer/Transport Layer Security*) para os protocolos IMAP e SMTP, no qual se assegura a segurança e integridade dos dados no acesso ou envio de um *email*.

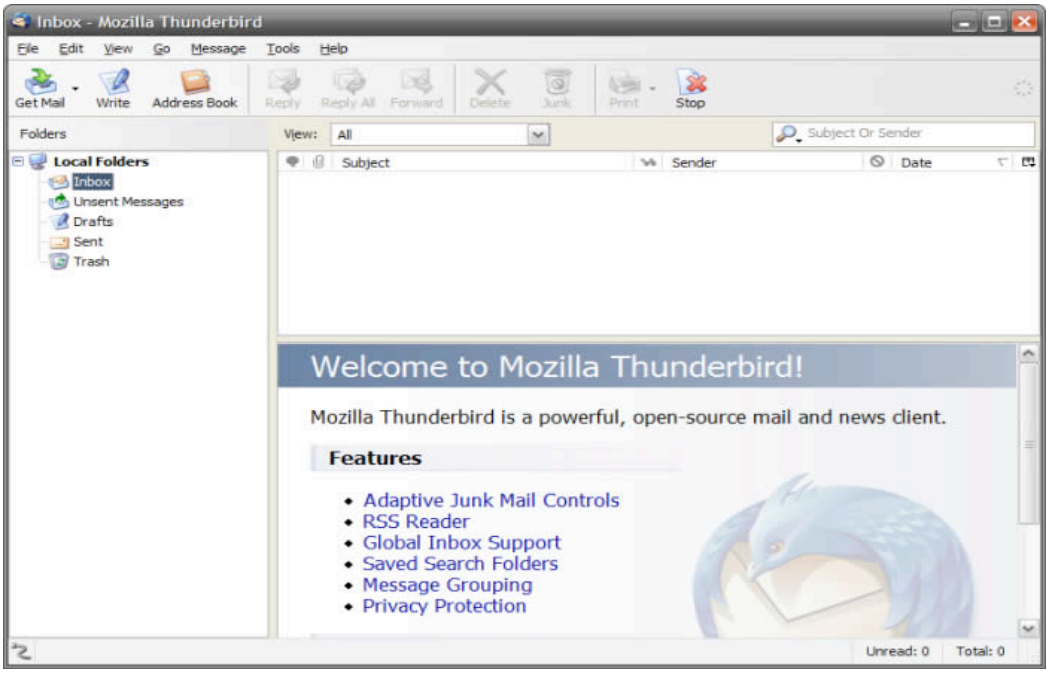

Figura 5. Janela do *Mozilla Thunderbird*

De base o *Thunderbird* vem equipado com um simples e intuitivo leitor/editor de mensagens de *emails*, um filtro de *spam e phishing* (actualmente a utilizar o algoritmo de aprendizagem *Naive Bayes* em conjunto com o uso de *whitelists* baseadas na lista de endereços), leitor de *RSS*[15] e grupo de notícias (*newsgroup*). Este cliente também permite a alteração do ambiente gráfico, usando temas, onde se altera a cor e as imagens dos botões, sendo então adaptável ao gosto de cada utilizador. Em comparação com outros clientes, de base, o *Thunderbird* apresenta poucas ferramentas. Aqui entram as extensões, que expandem o cliente com ferramentas, algoritmos, ou utilidades, que o utilizador necessite [7]. Com esta possibilidade, o cliente que de base é pouco completo, torna-se assim, mediante as necessidades do utilizador, um cliente de *email* personalizado e completo.

A popularidade deste cliente cresce dia para dia, muito pelas suas principais vantagens, i.e., por ser gratuito e ter uma vasta lista de extensões para todas as necessidades do utilizador. Além destas vantagens, este cliente também é *open-source* e está disponível para a grande parte dos sistemas operativos. Por tudo isto, este cliente foi o escolhido para o desenvolvimento deste trabalho.

[15] Sigla para Really Simple Syndication, que sao *web feeds* de actualizaçoes das paginas subcritas

3.2 Tecnologias para desenvolver extensões do *Thunderbird*

Como anteriormente foi referido, uma extensão do *Thunderbird* é um método de expansão das funcionalidades do cliente. Geralmente são feitas por utilizadores que sentem a necessidade de certas funcionalidades inexistentes e as dispõem online em páginas privadas ou na página oficial de extensões do *Thunderbird*[16]. Existe de momento uma vasta gama de extensões tais como dicionários, filtros de conteúdos, etc., que adicionam atalhos em forma de menus e/ou botões na janela principal ou em outra janela especifica (por exemplo, na janela de nova mensagem, se aplicável só a novas mensagens). Para que seja possível a sua distribuição e instalação, as extensões terão que ser compactadas em formato XPI[17]. Este é um ficheiro ZIP, que além de conter os ficheiros de código da extensão desenvolvida, inclui também um conjunto de *scripts* de instalação na raiz do directório. Estes *scripts* de instalação, no caso de versões mais actuais do *Thunderbird*, dividem-se no *chrome manifest* e num ficheiro *Resource Description Framework* (RDF). O primeiro especifica onde encontrar cada componente a instalar (incluindo se está compactado num ficheiro jar ou pasta), enquanto que o RDF é uma especificação de meta data do *World Wide Web Consortium (W3C)* em *Extensible Markup Language (XML)*, no qual indica a informação relativa à extensão, incluindo quem a desenvolveu, a sua identificação, a sua versão e as versões do *Thunderbird* compatíveis. Estes dois ficheiros são obrigatórios, caso contrário, o *Thunderbird* não saberá o que instalar e onde instalar.

Os restantes ficheiros, são responsáveis pela visualização gráfica da extensão, pelo "motor" da extensão, ou por outras acções associadas, incluindo abertura de outros programas. Alguns destes são opcionais, embora raramente deixados de parte. O tipo de ficheiros também será diferente para cada tipo de acção a executar, sendo alguns destes *scripts*, enquanto que outros executáveis.

Como descrito anteriormente, o *Thunderbird* é um cliente que é suportado pela maioria dos sistemas operativos. Isto deve-se ao facto de este ser baseado numa *framework* denominada de plataforma cruzada (ou em inglês Cross-platform), realizada em *C++*. Por esta razão, as extensões desenvolvidas, são igualmente compatíveis com os vários sistemas operativos existentes, excepto quando é utilizada alguma linguagem ou funcionalidade específica de algum. Isto é uma mais-valia para o desenvolvimento deste trabalho, e por isso a principal razão da escolha deste cliente para o desenvolvimento a efectuar neste projecto.

De seguida serão descritas algumas das linguagens de programação utilizadas no desenvolvimento da extensão projectada neste trabalho, bem como o papel de cada uma delas na solução desenvolvida.

---

[16] https://addons.mozilla.org/en-US/thunderbird/

[17] pronunciado como "zippi"

3.2.1 XUL

O XUL, siglas para *XML User Interface*, é uma formatação e vocabulário em XML para definir a apresentação gráfica do *Thunderbird* e suas extensões [7]. Com esta linguagem, é possível adicionar ao *Thunderbird* caixas de texto (também conhecidas pelo seu termo em inglês, *text boxes*), *check boxes, toolbars*, menus, bem como outros elementos relacionados com a visualização gráfica. Após adicionar algum destes elementos, o XUL permite associar uma acção previamente codificada [27]. Também é possível importar componentes já implementados bem como as suas acções associadas. A regularização da interface realizada com esta linguagem é efectuada pelo mesmo motor do *HyperText Markup Language* (*HTML)*, ou seja, tudo o que for codificado em XUL para apresentação gráfica, será transformado em HTML [7]. Por esta razão, esta linguagem tal como o HTML, suporta *Cascading Style Sheets (CSS),* no qual é usado para controlar o *layout* (tal como dimensões ou cores) do componente criado.

Como o XUL é por base XML mas com vocabulário definido, o começo deste tipo de script terá a mesma sintaxe de um ficheiro normal XML, no qual se inicia com a seguinte *tag* (na sua forma mais simples) [27]:

1: *<?xml version="1.0" encoding="UTF-8"?>*

indicando que o documento é realmente XML (pelo começo da tag "<?xml") com a versão "1.0" e que a codificação dos caracteres utilizada é "UTF-8". Na linha seguinte, especificam-se geralmente os ficheiros de estilo CSS (se utilizados) pela seguinte *tag* [7] :

1: <?xml–stylesheet href="chrome://global/skin/" type="text/css"?>

sendo que o começo da *tag* "*xml-stylesheet"* indica que é uma *tag* CSS, seguindo do link do ficheiro (*href*) e no final a indicação do tipo de ficheiro de estilo (usualmente "text/css"). De realçar que o *link* do ficheiro de estilo está no formato *"chrome://<directoria>"*, pois o *Thunderbird* utiliza a palavra reservada "chrome://" no *link* para os ficheiros pertencentes a extensão, atribuindo a estes permissões privilegiadas para a execução. Seguidamente, poderão ser importados ficheiros de extensão à gramática ou DTD (*Document Type Definition*). Após a indicação desta gramática (que é opcional), poderão ser então definidas todas as componentes, localizações e acções.

Por exemplo, pretende-se criar uma janela *find* com simplesmente dois botões: um com o texto "*find*" e um outro com o texto "*cancel*". Isto é possível com o seguinte código XUL[18]:

[18] Exemplo retirado do site: https://developer.mozilla.org/en/XUL_Tutorial/Adding_Buttons

```
1: <?xml version="1.0"?>
2: <?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
3: <window id="findfile-window" title="Find Files"
                    xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
4:  <button id="find-button" label="Find"/>
5: <button id="cancel-button" label="Cancel"/>
6: </window>
```

Este é então um exemplo simples de uma nova janela com a criação de botões. Na linha 3, abre a *tag* de uma nova janela com o título "*Find Files*" e que pertence ao *namespace* definido por "*xmlns*". De seguida, na linha 4 e 5 são então criados os dois botões, em que o primeiro conterá o texto "*Find"* e o segundo "*Cancel*". O exemplo é terminado com o fecho da *tag* responsável pela criação da janela (e que contem toda a hierarquia deste script) *"window"*.

3.2.2 Javascript

O *Javascript* é uma linguagem de *scripting* que foi criada pela *Netscape* em 1995 para que os seus *browsers* fossem capazes de validar formulários e que as páginas criadas interagissem melhor com o utilizador [28]. No decorrer dos anos, esta linguagem disseminou-se não só para outros *browsers,* mas também para outras linguagens e programas, tais como o *Thunderbird* e a inclusão no XUL. O *javascript*, ao invés de outras linguagens que são compiladas, é interpretado. Isto que dizer que o código fonte é executado directamente por um programa que a interpreta chamado interpretador, para que em seguida seja executado pelo processador. Uma das vantagens que esta linguagem apresenta é uma tipagem dinâmica em que os tipos das variáveis não são definidos. Esta característica faz com que as variáveis em *javascript* sejam reutilizáveis, mesmo quando são de tipos diferentes. Segundo [7], o *javascript* é uma linguagem baseada em protótipos. Para o autor, este tipo distingue-se das linguagens baseadas em classes (como o *Java* e *C++*) visto que, para estes últimos, as classes e as instâncias são entidades distintas.

Em termos de suporte de tipos, o *javascript* suporta nativamente grande parte dos tipos primitivos, tais como números, *strings* (cadeia de caracteres), booleanos (valores de verdadeiro e falso), *null* (valores nulos) e o *undefined* (valores que não estão definidos ou indisponíveis). Suporta também nativamente vários tipos de objectos, como por exemplo, funções, *arrays*, data-hora e regExps (i.e., expressões regulares) [7].

Para que seja possível a integração com o XUL, é necessário primeiramente especificar qual o script a carregar por este com a seguinte *tag* XML:

```
1: <script type="text/javascript" src="xxx.js"></script>
```

em que inicialmente se especifica qual é o tipo de *script* (neste caso *javascript*) para que de seguida se especifique a directoria e o *script* a importar, na propriedade *src* da tag (neste caso do exemplo, será na mesma directoria do ficheiro XUL e com o nome "xxx.js"). Também é possível definir o código *javascript* directamente no ficheiro XUL com a seguinte *tag:*

```
1: <script type="text/javascript">
2:         //codigo javascript
3: </script>
```

na qual, a característica *src* do exemplo anterior foi substituída pelo código *javascript* entre a *tag* de abertura e a de terminação. Quando na execução do código XUL aparece a uma destas *tags*, o código *javascript* lá contido é executado. No caso de novas funções, estas são carregadas para que quando chamadas elas possam ser executadas. Com estas funcionalidades extras que o *javascript* confere ao XUL, é possível adicionar acções aos elementos criados na nova extensão do *Thunderbird*.

A definição dos tipos primitivos em *javascript* é realizada pela palavra reservada *"var"* independentemente de qual o tipo a ser definido, visto que estes são então dinâmicos, mudando só o conteúdo atribuído a essa variável. Os objectos também são definidos da mesma maneira, à excepção das funções, que possuem uma sintaxe diferente. É possível também combinar várias funções numa só variável, sendo o princípio similar às classes do java, havendo a necessidade, nesta situação, de definir uma etiqueta antes da definição de cada função, i.e.:

```
1: var variavel1 = {
2:                 etiqueta1 : defeniçao_da_funçao1 (variavel_a_entrar)
3:                 {
4:                         //código a executar
5:                 },
6:                 (...),
7:                 etiquetaN : defeniçao_da_funçaoN()
8:                 {
9:                   //código a executar
10:                }
11:       };
```

e assim, quando por exemplo se quer executar a *funçao1*, só é necessário chamar a *variavel1* precedendo a *etiqueta1*, i.e., *variavel1.etiqueta1(x).*

Uma função é composta pelos seus atributos e pelo seu corpo. Esta tem um nome o qual deve ser único (tal como as variáveis) para que seja possível a identificação e a execução da mesma, e antecedida pela palavra reservada "*function"*. Os atributos são declarações de variáveis de modo a permitir a passagem e utilização de valores externos ao longo da função em questão. O corpo é um conjunto de instruções definindo

a execução pretendida da função. Em termos de sintaxe, uma função pode ser então definida da seguinte maneira [28]:

```
1: function nome_da_funcao(atributos)
2: {
3:         // corpo da função - código a executar
4: }
```

Uma função além de executar pedaços de código inseridos no corpo da função, pode retornar algum valor por ela calculada.

A titulo ilustrativo, um exemplo completo de uma extensão já com javascript, ou seja um ficheiro XUL com uma acção simples é o seguinte: Pretende-se uma extensão que seja bem-educada e que sempre que o utilizador abra o *Thunderbird* diga simplesmente "ola":

```
1: <?xml version="1.0"?>
2: <window>
3:         <script type="text/javascript">
4:                                 function dizOla()
5:                                 {
6:                                         alert("ola");
7:                                 }
8:                         dizOla();
9:         </script>
10: </windows>
```

O grande problema que o *javascript* tem é que só terá um processo para executar todo o *script* (não sendo possível concorrência de processos). Para que seja possível então uma execução paralela, são então utilizadas outras linguagens de *scripting* a complementar o *javascript*. Estas podem ser por exemplo *Bash/Shell scripting* ou *Python,* que serão explicadas de seguida.

3.2.3 Shell *Scripting*

Tal como o *javascript,* programar em *Shell Scripiting* requer um interpretador, que neste caso denomina-se por *Shell* (dai o nome da linguagem), para que seja possível a execução do código. Esta linguagem baseia-se numa sequência de instruções, que geralmente são comandos do sistema operativo. É possível também definir estruturas de decisão, i.e., cláusulas de condições "*if*", estruturas de repetição, tais como ciclos de *"while"* e *"for",* funções, entre outros. A utilidade deste tipo de script é na automatização de tarefas, que geralmente são realizadas na linha de comandos [29]. Tudo que for possível fazer na linha de comandos, é possível fazer com este tipo de linguagens, usando essas mesmas instruções, na qual a ordem

importa. É frequente encontrar scripts em *Shell scripting* em qualquer sistema operativo. Por exemplo em sistemas operativos *Unix*, o comando *ls* é um *script* que mostra o conteúdo de uma directoria [29].

O interpretador ao ler o conjunto de comandos ira produzir um resultado. Por serem comandos, e estes dependerem do sistema operativo, o *Shell* irá ser diferente para cada um, sendo então um script desenvolvido para *Windows* diferente de um para *Unix*. Por esta razão é que existem diferentes tipos de *Shell Scripting*. Por exemplo, para sistemas operativos *Windows* é usado o *Batch Scripting*, enquanto que para *Unix* é geralmente usado o *Bash Scripting*. A filosofia é semelhante entre os dois, visto que são ambos *Shell Scripiting*, mudando a forma de começar cada *script,* os comandos utilizados (visto que não são compatíveis entre SO's[19]) e a extensão do ficheiro utilizada. O *Batch Script*, foi criado inicialmente para a linha de comandos da *Microsoft*, o *MS-DOS*, no qual possui uma extensão **.bat*, e geralmente começa com a seguinte instrução :

```
1: @ECHO OFF
```

na qual indica ao interpretador para imprimir no ecrã somente o output resultante da execução de qualquer programa ou comando invocado (sem qualquer indicação da linha que está a executar).

Em *Bash script*, sistema criado para sistemas operativos com base *Unix* tais como o *Linux* ou *Mac OS X,* cada ficheiro terá uma extensão **.sh.* Este ficheiro terá que ser iniciado sempre pela seguinte linha:

```
1: #!/bin/bash
```

na qual indica o interpretador dos comandos *bash.*

No contexto do trabalho de desenvolvimento efectuado neste projecto, o *Shell Scripting* é usado em complemento com o *javascript* no *Thunderbird*, já que este não tem processos concorrentes e já é usado na execução do cliente de *email*. Assim sendo é possível, não só executar comandos, mas também lançar programas já compilados, sem que o cliente de *email* fique bloqueado.

3.2.4 *Python*

O *Python* foi desenvolvido em Amesterdão em 1990 por *Guido van Rossum*, mas só se tornou público em 1991 [30]. Esta linguagem é mais uma vez de *scripting*, mas no entanto poderá ser também compilada previamente, i.e., esta linguagem pode ser tanto interpretada como compilada [30]. Inserida nos paradigmas orientados a objectos, imperativos e funcionais, conta com similaridades com as várias linguagens de

[19] SO – Sistemas Operativos

programação mais utilizadas, tais como o *C* e o *Java.* Isto quer dizer que existe o conceito de funções bem como de classes, no mesmo ficheiro de *script,* sendo possível a integração de código de outras linguagens tais como *C++, Java* ou *.Net* [30]. A tipagem usada é forte, i.e., apresenta restrições nas operações quando envolve mistura de tipos de variáveis, e dinâmica, sendo pois possível atribuir vários tipos à mesma variável tal como o *JavaScript*. O *Python*, como é multi-plataforma, foi bem aceite junto da comunidade *open-source* e *Web Developers*, visto que este é aplicado também a paginas *Web*.

Tal como nas linguagens que o originaram, o *Python* apresenta os tipos básicos, tais como *strings*, inteiros, *arrays*, boleanos, entre outros. Também apresenta estruturas de ciclos, tais como o *while* e o *for*, e de condição, tal como a condição *if*. Um aspecto diferente de todas as linguagens mais conhecidas, é a necessidade de identação que o *Python* requer, ou seja, a organização estrutural é muito importante visto que não existe sinais de início e fim, tal como chavetas em *C* e *Java*.

Em *Python* uma classe é definida pela palavra reservada "*class*" seguida do nome da classe e dos parâmetros de início, i.e.:

```
1: class class1(parâmetros):
2: <corpo da classe>
```

Dentro de cada classe, é possível definir não só variáveis globais, mas também funções. Estas funções, são definidas pela palavra reservada *"def"* seguida pelo nome da função e dos seus parâmetros, que podem ser de qualquer tipo, incluindo objectos. Assim, a sintaxe para definir uma função em *Python* será a seguinte:

```
1: def funcao1(parâmetros):
2:         <corpo da funcao>
```

Se a função for *"__init__"* e definida dentro de uma classe, então a mesma será chamada quando a classe à qual pertence for criada, ou seja, será a função construtora que inicia a classe.

Tal como o *Shell Scripting*, o *Python* foi geralmente utilizado neste projecto para lançar processos que vão executar paralelamente à execução normal do *Thunderbird*.

3.2.5 *Java*

O *Java* foi criado pela *Sun Microsystems* na década de 90. Ao contrário da maior parte das linguagens de programação existentes, o *Java*, é compilado inicialmente para um *bytecode*[20] para depois ser executado por uma máquina virtual, a *Java Virtual Machine* (JVM). Esta máquina virtual é um programa que carrega e executa as aplicações em *Java*, previamente convertidas em *bytecode,* similar a um sistema operativo. Por esta razão, existe uma grande independência entre o código desenvolvido e o *software/hardware*, possibilitando então a portabilidade entre sistemas operativos diferentes.

O *Java* é uma linguagem de programação orientada a objectos, baseada nos modelos de *Smalltalk* e *Simula67*, mas apresentando uma sintaxe similar a linguagens tais como *C*, *C++,* mas principalmente, como o *C#.* Por ser orientada a objectos, apresenta uma distinta dependência das classes, sendo estas distintas dos métodos. Ou seja, todo o programa realizado em *Java*, tem que ter obrigatoriamente pelo menos uma classe principal, e dentro dessa classe poderá ter um ou mais métodos. Os métodos, definem a(s) funcionalidade(s) que a classe irá executar, i.e., exercem um papel similar ás funções anteriormente descritas nas outras linguagens de programação.

Além de esta linguagem ser independente do sistema operativo utilizado, apresenta uma outra vantagem, a facilidade da criação de programas distribuídos e multitarefas. As bibliotecas (normalmente conhecidas como *Application Programming Interface* ou API) distribuídas são vastas, geralmente bem documentadas e disponíveis para consulta *online*.

Os tipos disponíveis para a atribuição a variáveis, tais como inteiros, *strings*, boleanos, ou outros, podem ser globais ou pertencentes a cada método. Quando são globais, geralmente são previamente acompanhados por uma palavra reservada, a indicar a globalidade dessa mesma variável. Estas palavras reservadas podem ser *public*, ficando assim o uso directo da variável disponível para fora da classe, *private*, na qual a variável é só possível ser modificada pelos métodos pertencentes a classe que a possui, e *static*, assegurando que essa mesma variável só conterá uma referência na memória. As variáveis pertencentes a métodos só necessitam da declaração do tipo, visto que não são necessárias conter essa palavra reservada.

A construção de uma classe obedece a um conjunto de regras de sintaxe. Inicialmente o nome da classe terá que ser igual ao nome do ficheiro de código fonte (com a primeira letra em maiúscula), terá que conter a indicação da globalização da própria classe (tal como acontece nas variáveis) em conjunto com a palavra reservada "*class"*. Estas regras são obrigatórias para que a compilação e execução do código seja

[20] Ponto intermédio entre o código-fonte e a aplicação final, e produzindo sempre o mesmo resultado independente da plataforma na qual foi feita. Geralmente, o resultado é utilizado por uma maquina virtual.

possível, i.e., a representação de uma classe é feita obrigatoriamente da seguinte maneira, por exemplo, para um ficheiro minhaClasse.java:

```
1: public class MinhaClasse
2: {
3:  //definição da classe e dos seus elementos
4: }
```

na qual a classe é publica e se chama *MinhaClasse.* Na linha 3, é possível então definir não só os métodos pertencentes a classe, mas também variáveis globais.

No *Thunderbird*, o *Java* exerce um papel similar ao *Python*, o que, além de possibilitar a execução de processos paralelos, também aumenta as funcionalidades da extensão já possibilitadas pelo *JavaScript*. Um exemplo destas funcionalidades, é a leitura e escrita em ficheiros, que o *JavaScript* não possui nativamente.

No âmbito do projecto desenvolvido esta linguagem foi também utilizada no desenvolvimento de um servidor para permitir a partilha de ficheiros relativos ao funcionamento da arquitectura proposta.

3.3 Conclusões

Neste capítulo começaram por serem descritos alguns dos populares clientes de correio electrónico existentes. Neste contexto, foram apresentadas algumas características e funcionalidades de uma variedade de clientes de *email*, tais como o *Outlook* da *Microsoft,* o *Mail* da *Apple Computers* e o *Thunderbird* da *Mozilla Foundation,* para os vários sistemas operativos. Foram discutidas também as vantagens e desvantagens de cada um deles, bem como algumas das funcionalidades extras suportadas nativamente. Alguns destes clientes possibilitam a ampliação das funcionalidades extras por meio de extensões, que podem ser mais ou menos difíceis de implementar, dependentes ou não do sistema operativo para que foi desenvolvido ou *open-source*.

De seguida, descreveram-se algumas ferramentas, ou linguagens de programação, utilizadas no desenvolvimento de extensões de um cliente em específico, o *Thunderbird*. A razão pela qual recai a escolha neste cliente, relaciona-se com as vantagens no desenvolvimento para multi-plataformas e por ser *open-source.* Neste contexto, foram descritas algumas das funcionalidades, estrutura básica de funcionamento, papel desempenhado no desenvolvimento do cliente projectado e vantagens e desvantagens que apresentam cada uma das linguagens mencionadas.

As linguagens de programação descritas neste capítulo foram efectivamente utilizadas neste projecto, indo desde as essenciais para o funcionamento da extensão no cliente, o *XUL e o Javascript*, até às que auxiliam nesse mesmo funcionamento, tal como o *Java, Shell Scripting* e o *Python*.

4 Solução Desenvolvida

Com o objectivo de minimizar a quantidade de *spam* nas caixas de correio de cada utilizador bem como a ocorrência de falsos negativos ou falsos positivos, criou-se uma solução que permite não só ao utilizador classificar as suas próprias mensagens, mas também receber contributos de outros utilizadores, sendo este processo realizado através de procedimentos de partilha dos conjuntos de treino dos filtros. Neste sentido, optou-se pelo desenvolvimento de um sistema híbrido baseado na complementaridade e inter-operacionalidade de duas perspectivas distintas de combate a *spam*: uma baseada em processos de filtragem e outra baseada em processos de colaboração. A primeira vai permitir que o utilizador filtre e treine o seu filtro à sua medida, enquanto que a segunda permitirá aos utilizadores a partilha dos seus próprios *bag-of-words* ou *training sets* construídos durante o treino, com outros utilizadores da rede.

Assim, aglomerando estas duas atitudes, possibilitar-se-á ao utilizador uma filtragem ajustada à sua preferência e, em simultâneo, uma maior eficácia na detecção e redução do *spam*. Isto é possível mesmo quando as técnicas dos *spammers* são novas ou alteradas, bastando que alguns dos utilizadores da rede as detectem como tal.

Neste contexto, este capítulo centra-se na descrição da solução que foi desenvolvida e que possibilita o desenvolvimento de processos de filtragem colaborativa de correio não solicitado. Inicialmente a arquitectura da solução desenvolvida será apresentada e explicada, incluindo uma descrição de como se processa a classificação tanto por parte do utilizador, bem como por parte do Plug-in desenvolvido (sendo esta classificação automática). Adicionalmente, descrevem-se os processos da partilha dos filtros, no que se refere ao envio dos filtros dos utilizadores para um servidor dedicado, e ao *download* de um conjunto de filtros do servidor para o cliente do utilizador. No fim do capitulo será realizada uma descrição de todas as decisões tomadas no desenrolar do trabalho elaborado.

4.1 Arquitectura e Funcionamento

O sistema colaborativo desenvolvido para o combate à problemática do *spam* é de seguida apresentado, sendo composto por dois componentes distintos. O primeiro componente é constituído por acções de filtragem, com uma classificação que se baseia no conteúdo das mensagens. No entanto, aqui implementaram-se algumas alterações significativas no que diz respeito ao funcionamento normal de um filtro. Desta forma, na solução desenvolvida, em vez de um só *training set*, recorre-se ao uso de vários

diferentes, e que foram treinados de distintas formas e por vários utilizadores. Assim, a plataforma que foi desenvolvida para um cliente deste sistema terá um *training set* (ou *bag of words*) baseado em conteúdo (que é treinado pelo próprio utilizador), e um conjunto adicional de *training stets* pertencentes a outros utilizadores colaborativos que, sob anonimato, os disponibilizaram para o grupo em questão. Este primeiro componente da arquitectura ocupar-se-á também em elaborar estratégias inteligentes de combinação dos diferentes *training sets* partilhados e colocados ao dispor do utilizador.

A segunda componente do sistema desenvolvido ocupa-se exactamente com a distribuição destes mesmos ficheiros para um servidor que, de forma anónima selecciona quais os ficheiros a distribuir e para quem serão enviados, controlando também as políticas de acessos dos utilizadores. Estas duas componentes formam, no seu todo, uma abordagem baseada em métodos heurísticos, mas no entanto colaborativa, constituindo pois uma abordagem híbrida inovadora. De seguida serão explicados, pormenorizadamente, cada um dos componentes desenvolvidos, com apoio ilustrativo.

4.1.1 Arquitectura da Classificação e Filtragem

Nos dias correntes, o uso dos filtros para a classificação do *spam* está a aumentar exponencialmente. Isto tem-se verificado por inúmeras razões, tais como a possibilidade que o utilizador possui de treinar o seu próprio filtro, à sua maneira e gosto, e também pelo facto de que os algoritmos implementados apresentam uma eficácia bastante boa, comparativamente a outras técnicas anteriormente apresentadas. Esta técnica apresenta contudo algumas falhas que impossibilitam uma filtragem completamente fidedigna. As falhas começam por ser notoriamente visíveis quando o *training set* é criado e o filtro treinado, a partir de um número reduzido de casos de treino. Nesta situação o filtro classificará erradamente o *spam* e o *ham*, levando a inúmeros falsos positivos ou falso negativos. Também se verifica esta situação quando o utilizador treina incorrectamente o filtro. Outro problema que ocorre resulta das rápidas alterações das técnicas de *spam* utilizadas, em que o filtro, por não estar treinado para essas técnicas, obtém uma resposta errada. Por estas razões, as técnicas que se baseiam em filtros requerem, por parte do utilizador, um treino constante e eficiente para que estes não classifiquem erradamente as mensagens.

Por exemplo, são por vezes usadas técnicas como os ataques de dicionário a um utilizador especifico, na qual são enviados mensagens de *spam* numa língua, fazendo que o utilizador treine o seu filtro para mensagens nessa língua especifica. Num momento especifico, o *spammer* envia para esse utilizador a mensagem numa linguagem diferente e, como o filtro foi treinado para conteúdos de diferente língua, não o identifica como *spam*, resultando num falso negativo. Utilizando esta nova arquitectura, além do filtro ser

treinado pelo utilizador, recebe também colaboração da rede, e como os *training sets* que recebe da rede provavelmente foram treinados para tal, mensagens como estas serão detectadas como spam.

Os *trainings sets (*ou *bag-of-Words)* são ficheiros compostos por palavras soltas (ou outras características) e pelas suas frequências, as quais pertencem às mensagens que o utilizador classificou previamente como *ham* ou *spam*. Estes ficheiros, inicialmente vazios, são construídos durante o treino do filtro, integrando palavras contidas nas mensagens utilizadas nesse treino. Dado que é um ficheiro que contem somente uma lista desordenada de palavras e as suas estatísticas, não será possível obter qualquer mensagem apenas através da análise do conteúdo desse mesmo ficheiro.

Assim, cada vez que o filtro é iniciado, o ficheiro é carregado e através do seu conteúdo é formado o modelo de treino, possibilitando a classificação autónoma das mensagens. O modelo de treino é criado pela análise da frequência de ocorrência de cada palavra presente no ficheiro, formando uma base de conhecimento para que quando uma nova mensagem for recebida seja feita a sua comparação e a sua classificação.

A solução aqui apresentada, tem como parte integrante esta técnica de filtragem, mas de forma colaborativa, contribuindo para que os problemas apresentados passem a ser minimizados. Assim, este componente da arquitectura foi desenvolvido em extensões passíveis de serem integradas no cliente *Mozilla Thunderbird.* Adicionalmente, como base de trabalho foi utilizado um filtro já implementado numa extensão para esse cliente, o qual se denomina por *Thunderbayes*[21], que utiliza o algoritmo *Naive Bayes* na classificação das mensagens. Este já apresenta uma filtragem típica que, a partir das propriedades da mensagem, compara-as com o modelo de treino, previamente formado pelo *training set*, e devolve ao cliente de *email* um resultado em percentagem que reflecte se a mensagem é *spam* ou não.

A particularidade do *Thunderbayes* consiste na execução do algoritmo de *Naive Bayes* sob a forma de um *proxy*, fazendo com que o cliente se ligue a essa proxy, em vez da tradicional ligação directa ao servidor de *email*. Este *proxy*, além da classificação também fará a ligação ao servidor de email, sendo então possível o envio e a recepção das mensagens.

Para que os problemas identificados anteriormente fossem minorados, alterou-se substancialmente este proxy, de maneira a que fosse possível aceitar e usar não só o *training set* do utilizador, mas também os vários *training sets* provenientes de outros utilizadores, resultando numa arquitectura nova. A solução desenvolvida confere ao cliente uma nova abordagem no processo de treino. Por tal, e visto que se utiliza a cooperação com outros utilizadores, é possível aumentar a eficácia da filtragem mesmo que o filtro individual

[21] Disponível em https://addons.mozilla.org/en-US/thunderbird/addon/4025

do utilizador não seja antecipadamente treinado. De igual forma, mesmo no caso do filtro individual estar incorrectamente treinado ou aquando da recepção de um novo tipo de mensagem *spam*, será de esperar que esta estratégia consiga uma redução de falsos negativos ou falsos positivos.

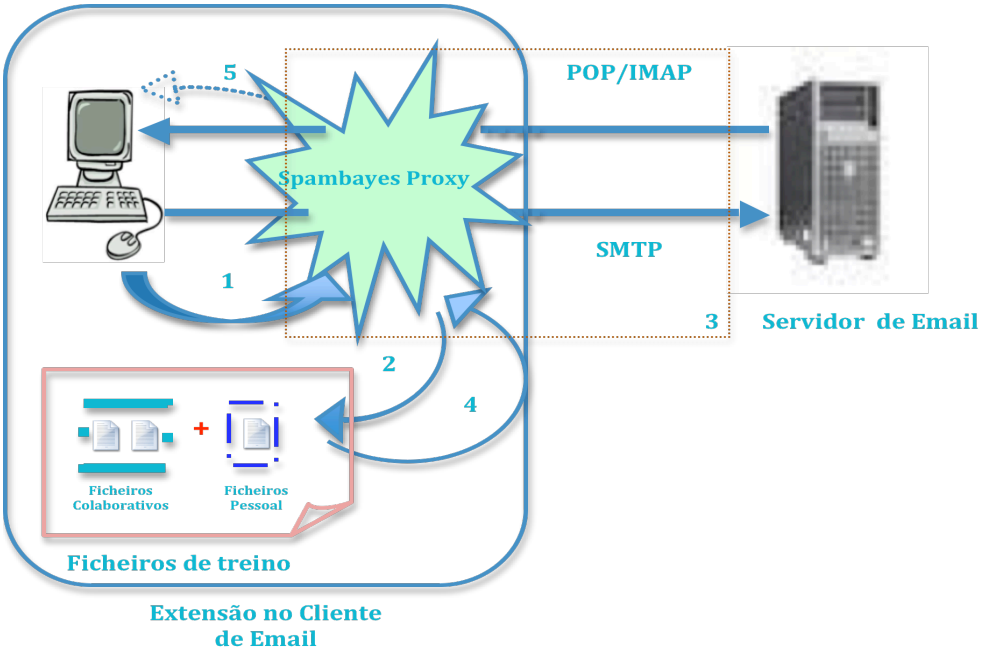

Figura 6. Arquitectura desenvolvida para a classificação de uma mensagem

A Figura 6 ilustra os passos desenvolvidos no treino e na classificação de uma mensagem na sua recepção. Na solução que é apresentada, o utilizador continuará a poder classificar manualmente as suas mensagens como *ham* ou como *spam*, dependo do que este considera como tal, além disso poderá proceder a correcções de classificações erróneas. O cliente, segundo esta classificação vai enviar para o *proxy*, usando o protocolo *HTTP,* o resultado da classificação (0 se for *ham* ou 1 caso contrário) e as características da mensagem, de modo a que o filtro apreenda as preferências do utilizador, como indica a Figura 6 no *passo 1*, construindo assim o seu próprio *training set*.

Relativamente aos filtros, o utilizador tem disponível as opções de partilha do seu próprio *training set* e a importação dos *training sets* de outros utilizadores alocados num servidor remoto (este procedimento de partilha será explicado mais à frente neste capítulo). A construção e a importação de *training sets* constitui o *passo 2* ilustrado na Figura 6. Consequentemente, o filtro do cliente de *email* do utilizador vai ter à sua disposição tanto o *training set* pessoal como os ficheiros colaborativos de outros utilizadores, todos podendo assim contribuir para o processo de filtragem. Assim sendo, após a recepção de uma mensagem, por POP ou IMAP, o proxy verificará as características desta (passo 3 da Figura 6), classificando-a. Posteriormente envia

essa mesma mensagem para o cliente. A classificação ocorre quando as características da mensagem são comparadas com o treino resultante do *training set* (passo 4), devolvendo-se uma probabilidade da mensagem ser *spam*. Esta probabilidade é calculada através do uso do algoritmo *Naive Bayes*. Visto que são utilizados diversos *training sets*, sendo que cada um deles retorna diferentes probabilidades, o algoritmo é executado de forma a agregar todas estas probabilidades numa única saída. Assim, para calcular a probabilidade final, o sistema desenvolvido recorre a uma das seguintes funções de combinação (M1 e M2):

- *M1 – Media aritmética*: o resultado final é calculado através da média aritmética das probabilidades geradas por cada *training set* individual, ou seja, cada *training set* é ponderado com o mesmo peso;
- *M2 – Coeficiente de correlação de Pearson:* para cada um dos ficheiros de treino importados pelo utilizador é calculado um peso (ou coeficiente) através da correlação de *Pearson*, que é dada pela seguinte fórmula:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \quad (11)$$

Esta correlação é aplicada ao conjunto de treino do utilizador em causa, ou seja, às mensagens da caixa do correio que anteriormente foram utilizadas para gerar o *training set* individual. Na equação 11, n denota o número total de mensagens consideradas, x_i designa o tipo de mensagem (se 1 é *spam*, caso contrário é um texto legítimo) e y_i representa a probabilidade (valor entre 0 e 1) prevista pelo *training set* a considerar. Esta correlação é calculada para todos os *training sets* que irão ser combinados. O resultado de r varia dentro do intervalo de [-1,1]. Quanto mais próximo o valor de $|r|$ à unidade, maior será a correlação entre as previsões do *training set* a considerar e os valores desejados. Assim, a ideia consiste em utilizar esta métrica para pesar o respectivo *training set*, de acordo com:

$$Rf(M) = \frac{\sum_{i=1}^{F}|r_i| \times \mathrm{Pr}_i(M)}{\sum_{i=1}^{F}|r_i|} \quad (12)$$

em que *Rf* denota o resultado final (probabilidade agregada dos diversos *training sets* a considerar), *M* a mensagem que se quer classificar, *F* o número total de *training sets* considerados, r_i o coeficiente da correlação calculado pela Eq. *(11)* para o *training set i* e $\Pr_i(M)$ a probabilidade prevista pelo *training set i* para a mensagem *M*.

Em cada um destes métodos, o resultado final será calculado e enviado para o cliente em conjunto com a mensagem (passo 5 da Figura 6). Como a *proxy* é local, o envio desse resultado será simples, não envolvendo nenhum protocolo especifico. Por tal, o envio será uma simples actualização da base de dados do cliente, na qual não se guarda apenas a mensagem mas também a informação relacionada com a mesma, tal como a sua classificação. A extensão desenvolvida vai ler o campo da base de dados correspondente e informar o utilizador sobre o resultado da classificação da mensagem.

É no entanto importante nesta fase realçar que, apesar da definição dos dois métodos anteriormente propostos, era objectivo deste trabalho o desenvolvimento de uma plataforma que permitisse o constante estudo e desenvolvimento de diferentes técnicas que combinem os diversos filtros partilhados. Dessa forma, as duas técnicas anteriormente referidas deverão ser vistas no contexto deste trabalho meramente como dois exemplos possíveis de funções de combinação de filtros.

4.1.2 Arquitectura da Partilha dos Filtros

Para que a arquitectura da filtragem anteriormente proposta seja possível será necessário possibilitar a partilha dos *training sets* entre os diversos utilizadores, recorrendo-se neste caso ao seu armazenamento num servidor central. De seguida será descrita a arquitectura implementada para tal efeito.

A Figura 7 refere-se à arquitectura desenvolvida para a partilha desses ficheiros. Aqui são incluídos dois servidores distintos: o Gestor de Repositório (GR), desenvolvido em *Java* que utiliza *sockets* para estabelecer a ligação com o cliente, e o servidor de ficheiros o qual tem implementado os protocolos de transferências de ficheiros *Secure File Transfer Protocol (SFTP)* ou *File Transfer Protocol (FTP)*. Estes dois servidores, que podem ser localizados na mesma máquina ou em máquinas distintas, formam o repositório dos ficheiros que, de forma anónima, gerem as transferências dos diferentes ficheiros partilhados.

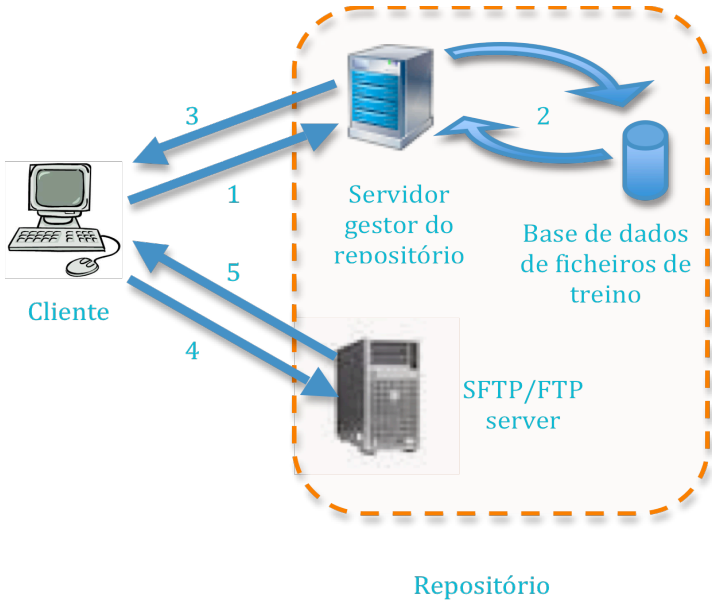

Figura 7. Arquitectura desenvolvida para a partilha de *training sets*

Inicialmente o cliente autentifica-se junto do GR com os seu dados e informa o servidor da acção pretendida. A acção poderá ser *download* ou *upload*, se este quiser transferir a lista de ficheiros do servidor para o cliente ou se quiser transferir o seu ficheiro para o servidor, respectivamente (passo 1 da Figura 7). A sua transacção é efectuada através do recurso à linguagem XML. De seguida, o gestor verificará se o utilizador está correctamente identificado, e em caso afirmativo formula uma resposta mediante o pretendido pelo utilizador. Se o utilizador pretender fazer o *download*, o gestor consultará a base de dados e construirá uma listagem de todos os ficheiros disponíveis para o utilizador. No caso de o utilizador pretender fazer o *upload* do próprio ficheiro o gestor consulta a base de dados para verificar o nome do ficheiro a armazenar.

Como forma de preservar a privacidade cada utilizador terá associado um nome de ficheiro aleatório, não havendo nenhuma correspondência entre o utilizador e esse mesmo nome. O passo 2 da Figura 7 corresponde à ligação à base de dados para verificação e formulação de listagens de ficheiros.

Após a formulação da listagem dos ficheiros a enviar ou a receber, o gestor constrói um ficheiro XML para ser enviado de novo ao cliente (passo 3 da Figura 7). Esta resposta contemplará não só os nomes dos ficheiros a transferir, mas também o endereço do servidor do sistema de ficheiros, os dados de autentificação do servidor e o tipo de protocolo que este utiliza (se é SFTP ou é FTP). Exemplos de sintaxes de pedidos e respostas envolvidas neste processo de comunicação podem ser observadas nos Anexos 2 e 3 para acções de *download* e de *upload*, respectivamente. Com os dados obtidos do gestor do repositório, o cliente liga-se ao servidor indicado, utilizando a autentificação fornecida e indica que ficheiro é que pretende transferir nesse

momento (passo 4 da Figura 7). O servidor verifica a autentificação e se contém o ficheiro pretendido, iniciando a transferência em caso afirmativo, tal como indica no *passo 5* da figura. Estes dois passos são repetidos para todos os elementos da lista de ficheiros a transferir, tal como indicado pelo gestor do repositório. Posteriormente à transferência, um ficheiro de texto é armazenado no cliente, o qual indica os *training sets* que se encontram a partir desse momento disponíveis. Assim, estes ficam aptos para serem utilizados na *proxy* do cliente, podendo a partir desse momento contribuir para a classificação das mensagens que entretanto serão recebidas no cliente.

4.1.3 *Interface* Gráfica e Outras Implementações

A *interface* gráfica é uma parte importante da arquitectura, sendo a partir desta que o utilizador interage com a solução desenvolvida. Assim, a interface foi desenhada de modo a ser fácil e intuitiva, visto que terá que ser acessível para qualquer utilizador, ou seja, desde o menos experiente ao mais experiente. Adicionalmente, no contexto do presente projecto foi também desenvolvida uma página *Web* com o propósito de divulgar e fornecer ajuda acerca de como a extensão desenvolvida funciona.

De seguida serão apresentadas as alterações realizadas no *Thunderbird* bem como a página desenvolvida para cumprir o objectivo proposto.

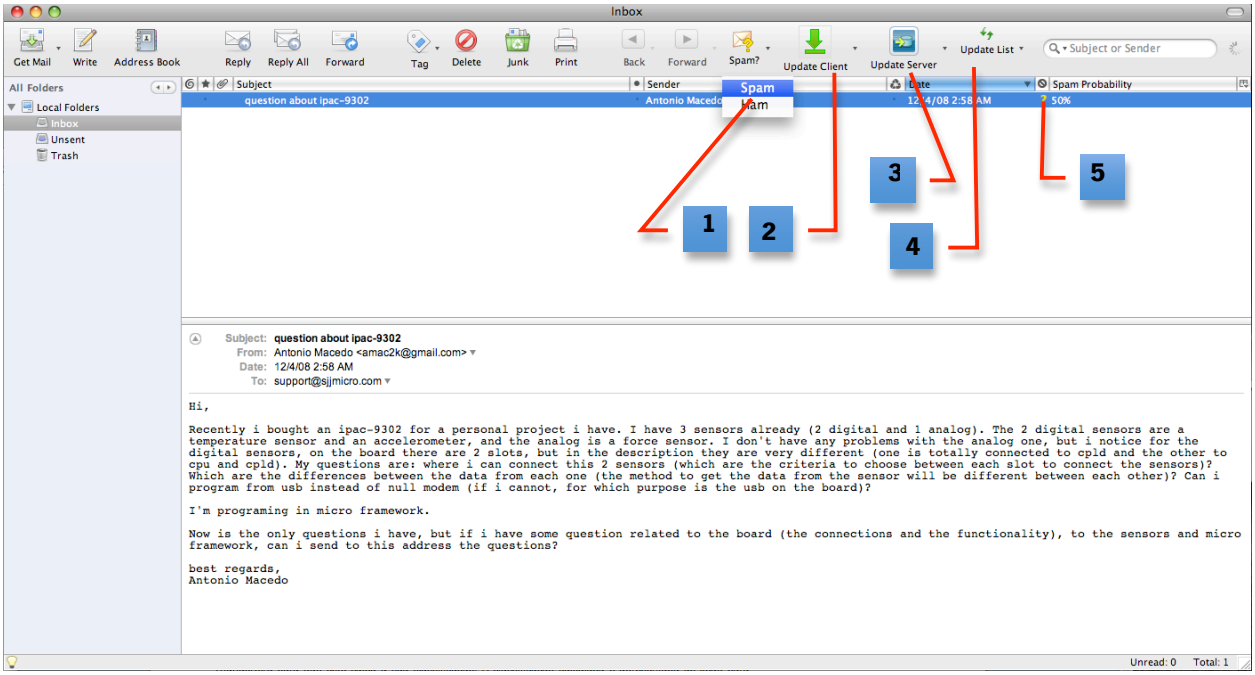

Figura 8. Exemplos de comandos adicionados à interface do *Thunderbird*

A Figura 8 ilustra a janela principal da aplicação do *Thunderbird*. Aqui é possível realizar várias tarefas, como por exemplo enviar ou ler mensagens. Salienta-se ainda as funções adicionadas por intermédio da instalação da extensão desenvolvida, sendo estas numeradas na Figura 8 de 1 a 5.

Na Figura 8 encontram-se indicadas com os números 1 e 5 as funcionalidades já existentes na extensão *ThunderBayes*. O número 1 representa o botão de treino, ou seja, ao seleccionar-se a mensagem que se pretende classificar, o botão é pressionado podendo escolher-se uma das opções existentes: *ham* ou *spam*. A coluna com a indicação do resultado de classificação, em percentagem, encontra-se assinalado como o número 5. A união destas duas funcionalidades faz com que o treino e a avaliação seja possível. Embora estes botões/colunas graficamente não tenham sido modificados, o mecanismo de avaliação foi alterado para suportar a arquitectura desenvolvida.

Outras funcionalidades foram adicionadas ao interface para sustentar a partilha dos *training sets*. No *Thunderbird* são representados sob a forma de botões e numerados na Figura 8 do número 2 ao número 4. O processo de partilha e difusão de filtros implementado no projecto é realizado de duas maneiras: o *download,* que recebe os *training sets* de outros utilizadores disponíveis no repositório de ficheiros (número 2 na Figura 8), e o *upload,* que envia para o repositório de ficheiros o ficheiro do utilizador (número 3 na Figura 8). As duas operações são sempre precedidas de um pedido ao GR, no entanto o pedido não é visível pelo utilizador.

Como muitas vezes o objectivo é partilhar o próprio ficheiro e receber simultaneamente actualizações foi criado um botão que realiza as duas operações em simultâneo (número 4 da Figura 8) de modo a facilitar esta tarefa.

Para que a partilha funcione, o utilizador necessita de configurar o gestor de repositório ao qual tem acesso. Esta tarefa é possível no painel de configurações da extensão e encontra-se indicado na Figura 9.

Neste painel de configuração, o utilizador poderá configurar o *Thunderbayes* para, por exemplo, mover a mensagem para uma pasta especifica após ter sido manualmente classificada. No painel mencionado adicionou-se mais uma configuração, assinalada na Figura 9, para configurar o endereço (*Hostname* na figura) e a porta (*Port* na figura) de ligação com o gestor de repositório de modo a que o cliente possa enviar o pedido para o mesmo. A configuração permite utilizar um nó central e comum para todos os utilizadores da extensão. A criação de nós centrais para cada grupo especifico de utilizadores também é uma possibilidade permitida da configuração. Estas opções são de configuração opcional visto que de base o cliente já se encontra configurado para se conectar a um gestor do repositório geral.

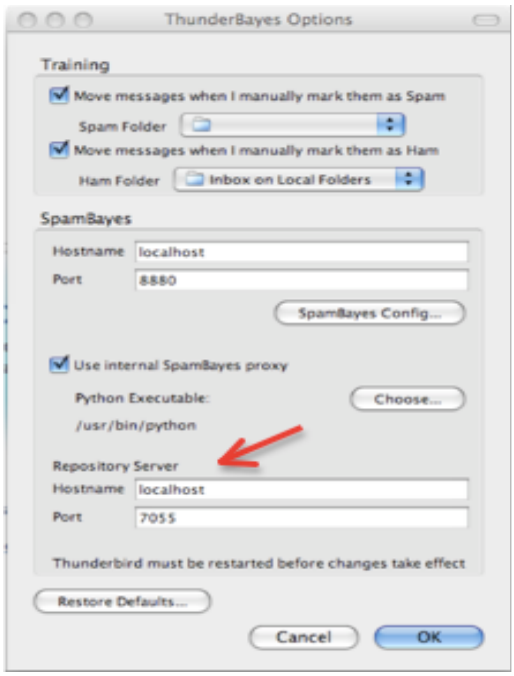

Figura 9. Interface de configuração do *ThunderBayes*

Após o envio e a recepção do pedido, o cliente executa o programa em *Java* para realizar a transferência dos ficheiros. No final uma mensagem de sucesso é observada. O processo não impede o cliente de continuar com a sua execução normal.

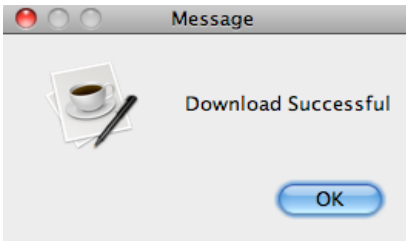

Figura 10. Mensagem de sucesso de *Download* dos ficheiros

Caso a tarefa a efectuar seja de *download*, será apresentada a mensagem de "*Download Successful*" no fim da transferência de todos os ficheiros envolvidos, tal como a Figura 10 indica. Assim, é criado um ficheiro no qual constam os nomes de todos os ficheiros transferidos.

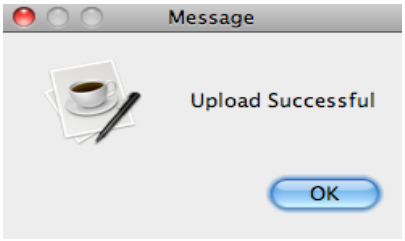

Figura 11. Mensagem de sucesso de *Upload* dos ficheiros

Quando a tarefa é de *upload* do *training set*, será exibida a mensagem de "*Upload Successful* ", tal como indica a Figura 11. Esta mensagem aparece quando o seu ficheiro é enviado para o repositório de ficheiros usando os protocolos indicados pelo GR na resposta do pedido. As mensagens apresentadas nas Figuras 10 e 11 podem aparecer em simultâneo caso o utilizador accione o botão 4 da Figura 8.

Além das configurações do *Thunderbayes*, existe também um painel disponível para configurações do *Spambayes* sob a forma de uma pagina *Web* que funciona localmente.

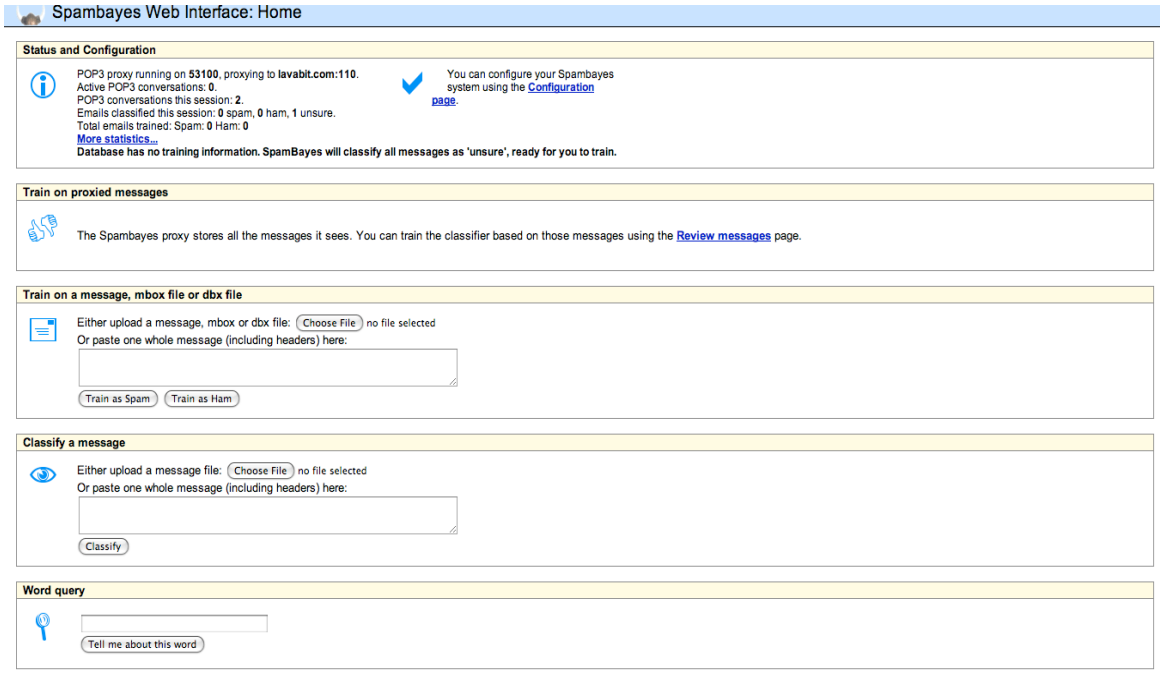

Figura 12. Painel de administração do *Spambayes* (parte integrante do *ThunderBayes*)

A Figura 12 ilustra esta página. Embora todos os campos se mantenham iguais ao original, o funcionamento foi em parte alterado. Não só é possível alterar o modo de funcionamento do *SpamBayes*, i.e., configurar o filtro, como também é possível o seu treino. O treino é feito tanto através da inserção de um ficheiro de *email*, bem como pela inserção de texto (representando uma mensagem). A observação de

estatísticas do treino do filtro e das mensagens recebidas (de *ham* e de *spam*) e avaliar palavras mediante o treino do filtro são outras possibilidades que a página apresenta.

Adicionalmente ao desenvolvimento da arquitectura de partilha de filtros, foi também necessário criar um sistema de suporte e divulgação da extensão desenvolvida por forma a motivar a participação de diversos utilizadores. Para tal criou-se uma página em *PHP*[22], para que de uma forma intuitiva se permita a realização desta função.

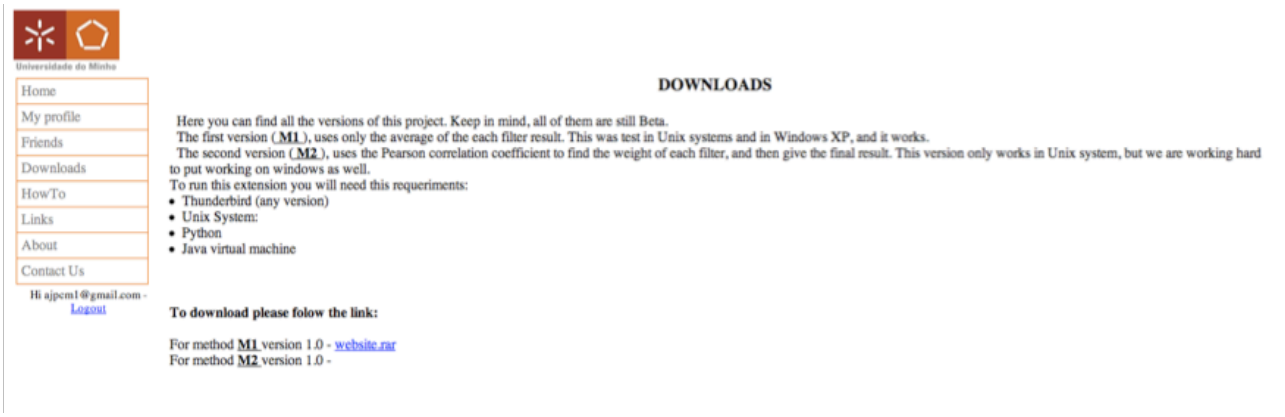

Figura 13. Página Web de apoio e distribuição da arquitectura desenvolvida

A Figura 13 ilustra a página *Web* criada. Nesta página executa-se o *download* da extensão alterada, e também se obtém ajuda sobre o seu funcionamento (em forma de tutoriais). A divulgação da extensão entre os contactos dos participantes, a obtenção de informações sobre como contactar a equipa do projecto para questões ou sugestões, além de outras funcionalidades também estão presentes na página desenvolvida. Para tal, o utilizador terá que se registar, necessitando apenas de introduzir o seu endereço de *email* e uma *password* escolhida pelo mesmo. Além disso, poderá indicar possíveis interessados, introduzindo o endereço destes no momento do registo.

4.2 Exemplo da arquitectura em funcionamento

Após a descrição de como foi projectada a arquitectura desenvolvida e da sua apresentação gráfica, segue-se agora um exemplo abstracto que pretende clarificar e ilustrar o funcionamento prático da mesma.

No exemplo, o João é o utilizador de *email* que utiliza a extensão desenvolvida. A Maria, é a amiga do

[22] Acrónimo recursivo para "PHP: Hypertext Preprocessor"

João que lhe envia uma mensagem e o Spammer-*x*, é o *spammer* que enviou uma mensagem não só para o João, mas também para um conjunto determinado de utilizadores.

O João instala pela primeira vez a extensão desenvolvida e, naquele instante, tem 50 mensagens já recebidas anteriormente. Das 50 mensagens, ele sabe as que são *spam* e as que são *ham*. Então, o João irá treinar o filtro com algumas dessas mensagens, tal como mostra a Tabela 1.

	Total Mensagens	**Usadas no Treino**	**Amostra das Palavras usadas no Treino**
Ham	40	5	Café, fotos, encontro, *download*, registo, tese
Spam	10	10	*Drugs, viagra, sex, scandal, cheap, free,ilegal, celebreties*
Total	**50**	**15**	

Tabela 1. Mensagens que o João tem na sua caixa

O João selecciona primeiramente as 5 mensagens de *ham* e classifica-as como tal. A extensão enviará todo o conteúdo de cada mensagem, incluindo os seus cabeçalhos (*Headers*), por *HTTP* para a *proxy* do filtro, sendo esta sinalizada com a *flag ham*. O filtro, por sua vez, irá observar esse conteúdo, verificando cada uma das características da mensagem através da análise da ocorrência de cada palavra. Além disso, forma um ficheiro com cada palavra associada à respectiva ocorrência, começando assim a formar o *training set*. De seguida, o João selecciona as restantes 10 mensagens que identificou como *spam*, repetindo o processo anterior, mas desta vez enviando a mensagem para a proxy com a flag *spam*. O filtro também vai analisar o conteúdo de cada mensagem e vai adicionar ao ficheiro formado anteriormente as características observadas. Com isto, o filtro estará em condições de formar um modelo de treino capaz de começar a avaliar as novas mensagens recebidas.

Entretanto, o João, que gosta muito de contribuir para a comunidade, partilhou o seu ficheiro *training set* e realizou a actualização dos ficheiros *training set* provenientes do servidor (neste momento não possui nenhum, visto que acabou de instalar a extensão).

Deste modo, a extensão formaliza primeiramente um pedido, em *XML*, junto do GR para realizar a transferência do seu ficheiro (o *upload*), da seguinte forma:

```
<?xml version="1.0" ?>
    <training>
        <type>upload</type>
        <id>1</id>
        <password>at12ksma</password>
        <fileName>/Users/joao/Library/Thunderbird/Profiles/profile1/spambayes_data</fileName>
    </training>
```

Este pedido indica ao GR que o tipo de operação pretendida será de *upload*, identifica-se como o utilizador 1 com a *password* at12ksma (a qual é estática e igual para todos os utilizadores por motivos de teste, estando somente implementada, de momento, a estrutura para suporte à utilização da *password*), e que o ficheiro a fazer upload se situa na directoria local /Users/João/(...)/spambayes_data.

O GR ao receber este pedido, analisa-o e verifica a informação de autenticação do João. Posteriormente, por ser um pedido de *upload* e para que a confidencialidade do João seja preservada, o GR retira da base de dados o nome a dar ao ficheiro que o João irá transferir. Como o servidor de Repositório de Ficheiros (RF) e o GR são diferentes serviços, podendo mesmo estar em computadores diferentes, o GR conterá toda a informação sobre a localização, protocolos de comunicação e acesso ao RF. No caso concreto do RF a que o João irá aceder, o protocolo usado será o SFTP, que está localizado no endereço sftp.uminho.pt, utiliza a porta 21 para estabelecer ligação e requer como *username* tese e *password* 1234 para a autentificação do acesso.

Após saber toda a informação de acesso e de processamento do ficheiro a receber, o gestor formula a resposta ao pedido de *upload*, também em XML, contendo a seguinte informação:

```
<?xml version="1.0" ?>
    <ftp>
        <link>sftp.uminho.pt</link>
        <port>21</port>
        <serverType>scp</serverType>
        <type>upload</type>
        <username>tese</username>
        <pwd>1234</pwd>
        <path>/User/SFTPRep/</path>
        <filename>hammie.pck</filename>
        <endFilename>1.pck</endFilename>
        <originalName>/Users/joao/Library/Thunderbird/Profiles/profile1/spambayes_data</originalName>
    </ftp>
```

Aqui, indicar-se-á à extensão o endereço e a porta do repositório, o tipo de protocolo utilizado para a comunicação com o RF, o tipo de pedido a que está a responder, o *username* e a *password* que o RF necessita para realizar o acesso as directorias. A directoria a aceder no RF para transferir o ficheiro, o nome original do ficheiro a transferir e o seu pseudónimo (nome a dar no RF) e a directoria onde o ficheiro se situa

localmente no cliente também serão mencionados à extensão. A extensão ao receber esta resposta irá invocar um programa em *Java*, que executará o *upload*, utilizando para tal todos os campos fornecidos pela resposta, incluindo o protocolo que a mesma especificou. Com esta transferência, a comunidade beneficiará da classificação do João, o que contribuirá para uma melhor classificação.

Posteriormente a enviar toda a informação, a extensão prepara um novo pedido ao GR, contudo desta vez será para efectuar a transferência dos ficheiros de outros utilizadores (*download)*, presentes no RF. O novo pedido, também codificado em XML, contém os mesmos dados de identificação do utilizador e acção pretendida, tal como no pedido de upload. Este será descrito da seguida forma:

```
<?xml version=\"1.0\" ?>
        <training>
                <type>download</type>
                <id>1</id>
                <password>at12ksma</password>

        <fileName>/Users/joao/Library/Thunderbird/Profiles/profile1/spambayes_data/</fileName>
          </training>
```

Como aconteceu no pedido de envio de ficheiro, aqui indica-se ao GR o tipo de operação pretendida (*download)*, a sua identidade e a sua directoria, de modo a possibilitar a transferência dos ficheiros.

O GR, ao receber este pedido, irá formular uma resposta idêntica ao pedido de upload (realizada anteriormente), mas desta vez consultará a base de dados para criação de uma lista de ficheiros, incluindo-a na sua resposta, à qual o utilizador poderá ter aceso para transferência. Então, o pedido é criado, com os dados do RF anteriormente descritos e a lista de ficheiros acabada de formar. No caso do João, a lista será composta pelos ficheiros *2.pck, 3.pck e 4.pck*, que são todos os ficheiros presentes no RF. Com esta informação o GR é capaz de formalizar a seguinte resposta:

```
<?xml version="1.0" ?>
        <ftp>
                 <link>sftp.uminho.pt</link>
                <port>21</port>
                <serverType>scp</serverType>
                <type>download</type>
                <username>tese</username>
                <pwd>1234</pwd>
                <path>/User/SFTPRep/</path>
                <files>
                        <filename>2.pck</filename>
                            <filename>3.pck</filename>
                           <filename>4.pck</filename>
                </files>
                  <originalName>/Users/joao/Library/Thunderbird/Profiles/profile1/spambayes_data</originalName>
          </ftp>
```

Da mesma forma que na resposta do pedido de *upload*, esta indica o endereço, a porta e o tipo de protocolo que o RF terá, o tipo de acção pretendida, o *username* e a *password* do RF para efectuar o acesso, a directoria na qual se situam os ficheiros a descarregar, a listagem com os nomes desses ficheiros e por fim, a directoria para onde transferir os ficheiros no cliente. A extensão, ao receber estes dados, invoca o programa em *Java* (anteriormente referido), que descodifica a resposta XML e transfere esses ficheiros. O programa conclui criando um ficheiro que contém todos os nomes dos ficheiros acabados de transferir.

No final da execução deste programa, caso o João esteja a utilizar a extensão com o método M2, esta encontra as 5 mensagens de *ham* e as 10 de *spam*, classificadas pelo próprio. Assim, é chamado uma *script* que, dadno como *input* esses *emails* e o ficheiro criado pelo programa em *Java*, encontra para cada uma das entradas do ficheiro o seu respectivo peso em relação às mensagens classificadas pelo utilizador (a classificação é certa, sendo *ham* igual a 0 e *spam* igual a 1). Os pesos são calculados através do coeficiente de correlação de Pearson explicado anteriormente. Para cada ficheiro, o resultado dos pesos calculados (em valor absoluto) são os apresentados na Tabela 2. É, então, gravado numa base de dados o peso de cada um dos ficheiro para futura consulta, no momento da utilização na classificação de mensagens.

Ficheiro	Peso (valor absoluto)	Comentários
hammie.pck	1	*Training-set* do utilizador.
2.pck	0.5	Existe uma correlação média[23] entre o resultado da classificação usando este ficheiro e a classificação dada pelo utilizador.
3.pck	0.1	Existe uma fraca correlação entre o resultado esperado e o resultado dado usando este ficheiro. A classificação usada por este ficheiro não será muito acertada para o João, levando na maioria das vezes a FN ou a FP. Por esta razão devera ser dada menos importância ao resultado proveniente deste ficheiro.
4.pck	0.6	Existe uma correlação média (sendo maior do que a correlação de 2.pck) entre os dois resultados, sendo então mais indicado para o João, logo será dada maior importância.

Tabela 2. Tabela de pesos de cada *Training Set*

Após este passo, a actualização do servidor e do cliente é realizada com sucesso, e este último está pronto a classificar as novas mensagens com os mais recentes modelos de treinos, importados de outros utilizadores, sempre com a personalização do João.

[23] com $|r| \geq 0.7$ correlação forte, $0.3 < |r| < 0.7$ correlação média e $0 \leq |r| < 0.3$ correlação fraca

Como o João para receber mensagens tem que se ligar por *POP* (único protocolo disponibilizado pelo servidor), e visto que agora tem uma proxy entre o servidor de *email* e a sua caixa de correio, a extensão irá alterar a ligação de *fetch* de *email* para a proxy, sendo esta que estabelece a conexão para o servidor do João.

Imaginemos agora que, a amiga de longa data do João, a Maria, envia a seguinte mensagem para combinar um café com o mesmo:

From: *Maria@friends.com*
To: *João@myemail.com*
Subject: *Café amanha?*
(...)
Message:

Olá,

Então João esta tudo bem contigo? Por aqui está tudo óptimo. Ontem estive a ver as fotos dos nossos tempos de infância e apercebi-me que já não estou contigo á muito tempo.

Que dizes amanha a tarde irmos tomar um café juntos, actualizar as noticias e matar saudades?

Beijos,

Maria

Ao mesmo tempo o *Spammer-X*, envia também a seguinte mensagem, a fazer publicidade a venda de drogas tais como "Viagra":

From: *spammerx@maskemail.com*
To: *João@myemail.com*
Subject: *Best buy for you!!*
(...)
Message:

Dear costumer,

We have some excellent deal for you...CHEAP DRUGS!

Do you know Viagra and Valium are hard to get and you will need prescription?

Not any more... Now you can get Viagra for $2 and Valium $3 per pill.

Go to drugs.com and you will see this and much more!

Cheers!!

A *proxy* irá realizar o *fetch* destas duas mensagens de *email* e, para cada uma, irá observar as suas características principais, compara-las com os modelos de treino gerados através de cada *training set*

importado, e obter, para cada um destes, uma classificação (através da utilização do algoritmo *Naive Bayes*). De seguida, cada classificação será multiplicada pelo seu respectivo peso (calculado anteriormente, e referido na Tabela 2), de forma a atribuir uma confiança à pontuação gerada por esse *training set*, sendo por fim, somados cada um desses valores e divididos pelo somatório dos pesos de cada *training set* disponível na base de dados. Por tal, quando o filtro recebe o *email* da Maria extrairá as características dessa mensagem e obterá, para cada um dos modelos de treino gerados pelos *training sets* disponíveis (incluindo o seu próprio), uma classificação, que depois é multiplicada pelos pesos da tabela anterior, originando o presente resultado (ver Tabela 3).

	Peso absoluto (P)	**Classificação (C)**	**Total (P*C)**	**Comentários**
hammie.pck	1	0.01	0.01	O João treinou previamente o filtro como *ham* com palavras que aparecem na mensagem, tais como fotos, encontro e café. Não aparece nenhuma palavra na mensagem que terá sido utilizada no treino de *spam.* Estas contribuíram para uma classificação baixa.
2.pck	0.5	0.3	0.15	Embora uma classificação não tão baixa como a do anterior e não com tanta certeza, a utilização deste *training-set* não deu indicação de *spam*. Este valor poderá ter surgido devido a algum conteúdo da mensagem ter sido classificado como *spam*, mas a sua maioria como *ham.*
3.pck	0.1	0.7	0.07	Embora não tenha muito peso na classificação, este valor indica que esta mensagem é *spam*, o que neste caso poderia levar a um FP. Esta classificação errónea poderá ter sido provocada pela classificação como *spam* de *email*s com conteúdos semelhantes ao desta mensagem
4.pck	0.6	0.2	0.12	Com mais certezas que o 2.pck, esta classificação indica que a mensagem será *ham*. As razões enumeradas anteriormente continuam a ser válidas para este caso.
Somatório dos Totais (ST)			0.35	
Somatório dos Pesos (SP)			2.2	
Classificação Final (ST/SP)			**0.15**	

Tabela 3. Classificação da mensagem da Maria

Como indicado na Tabela 3, a classificação total é de 0.15. Esta classificação é baixa, logo dando ao João uma indicação forte que a mensagem é segura (cerca de 15% de certeza que é *spam*, i.e, 85% de certeza que é *ham*). Por sua vez, a proxy envia essa mensagem ao cliente e actualiza a base de dados do

Thunderbird para que este saiba a sua classificação. A classificação calculada é apresentada ao João para que ele saiba agir com segurança.

Ao receber a mensagem do *Spammer-X*, o filtro irá proceder de igual forma, gerando os resultados da Tabela 4.

	Peso absoluto (P)	**Classificação (C)**	**Total (P*C)**	**Comentários**
hammie.pck	1	0.99	0.99	A maioria das palavras que o João usou para treinar o filtro como *spam*, aparecem no *email*, logo terá uma classificação elevada.
2.pck	0.5	0.99	0.495	As palavras contidas na mensagem são geralmente indicadoras de *spam*, e quando se classifica bem o filtro, a classificação será elevada, tal como aconteceu nesta situação.
3.pck	0.1	0.8	0.08	Embora a classificação seja dada como *spam*, esta classificação não é muito elevada. Isto acontece quando as palavras que contem a mensagem sejam usadas também em mensagens *ham*, para o utilizador em questão.
4.pck	0.6	0.95	0.576	Aqui aconteceu certamente o que foi descrito em 2.pck.
Somatório dos Totais (ST)			2.141	
Somatório dos Pesos (SP)			2.2	
Classificação Final (ST/SP)			**0.973**	

Tabela 4. Classificação da mensagem do Spammer-X

Ao observar a Tabela 4 apercebemo-nos que a classificação final da mensagem é de 0.973. A classificação indica ao João que a mensagem recebida é com grande probabilidade *spam* (cerca de 97.3% de certeza que é *spam*). Esta classificação é enviada ao cliente, tal como aconteceu com a mensagem da Maria, e este irá apresenta-la ao João para que decida se quer ver ou apagar a mensagem.

A qualquer instante, o João poderá classificar uma nova mensagem de *email*, ou então reclassificar alguma mensagem que se enganou a classificar. Também poderá, a qualquer instante e sempre que achar necessário, actualizar as listas de *training sets*.

Com este exemplo prático, demonstrou-se passo a passo os fundamentos da arquitectura desenvolvida e o funcionamento da aplicação elaborada neste trabalho.

4.3 Decisões de Desenvolvimento

Algumas decisões tiveram que ser tomadas tanto na estruturação da arquitectura, bem como no seu desenvolvimento. Estas recaíram na escolha do cliente de *email* a utilizar, na possibilidade da utilização de algum filtro já implementado, nos protocolos de comunicação para o envio e recepção dos ficheiros e nas linguagens de programação a serem utilizadas para implementar todos os módulos e funcionalidades desenvolvidas no projecto.

Como referido anteriormente existem muitos clientes de *email*, cada um com as suas vantagens e desvantagens. A escolha do cliente teria que atender a alguns requisitos, tais como ser de código aberto para uma maior facilidade de interacção e implementação, utilizar extensões como possibilidade de expansão de funcionalidades, ser implementado para uma grande variedade de plataformas (multi-plataformal) e utilizado por um vasto número de utilizadores. Observando as características dos principais clientes de *email* enunciadas no Cap. 3, é fácil excluir o cliente da *Apple*, o *Mail*, visto que somente é suportado pelo sistema operativo da própria empresa e não é de código aberto. Assim, restam dois clientes, o *Outlook e o Thunderbird*. O primeiro, embora utilizado por um grande número de pessoas (sendo até ao momento o cliente de *email* mais utilizado), apenas é suportado por dois sistemas operativos (o da *Microsoft e do da Apple*) e não é de código aberto. O segundo, não sendo tão utilizado como o *Outlook*, é no entanto gratuito, de código aberto e multi-plataformal. Tendo em consideração estas vantagens e desvantagens, decidiu-se então utilizar por base o *Thunderbird* no desenvolvimento desta arquitectura.

Após a escolha do cliente para a realização deste trabalho, foi necessário definir a técnica de filtragem a utilizar na solução a desenvolver. A técnica de filtragem utilizando métodos heurísticos foi a seleccionada dado que é a técnica que mais se adapta às preferências de cada utilizador. Os *training sets* (ou *bag-of-words*) são criados por meio de treino, o que possibilita a sua partilha. Muitos são os algoritmos existentes que utilizam esta técnica, sendo actualmente o melhor e o mais utilizado o *Naive Bayes*, algoritmo explicado anteriormente no Cap. 3. Este algoritmo, para além de apresentar as vantagens referidas, também possibilita a criação do *training set* referido.

Existem vários filtros de código aberto disponíveis que implementam este algoritmo, inclusive o próprio *Thunderbird* possui um. Uma vez que o propósito deste trabalho não é criar um filtro de raiz mas sim alterar uma técnica de modo a tentar melhorar o desempenho, decidiu-se utilizar um dos filtros já existentes.

O filtro incluído no *Thunderbird* seria o ideal visto que não necessita da instalação de programas extras, pois este já se encontra implementado e pronto a utilizar. O grande problema na sua utilização está na impossibilidade de ser alterado por meio de extensões, porque a única maneira de ser modificado é no seu código fonte, o que dificulta a distribuição do mesmo. A solução foi encontrada através da utilização de uma extensão já existente, o *ThunderBayes*. Este integra um outro programa (funcionando como *proxy*) chamado *SpamBayes*, o qual usa o *Naive Bayes* para obter a classificação. Além disso, o *ThunderBayes* permite de uma forma fácil alterar o seu código, sendo apenas necessário a distribuição dessa mesma extensão. Uma outra vantagem na utilização do *ThunderBayes* centra-se no facto de já conter todo o *software*, não necessitando para o seu funcionamento a instalação de programas extras.

A arquitectura proposta possui duas partes distintas, a filtragem e a colaboração por meio de partilha de filtros. Em relação à segunda componente foi necessário escolher os protocolos de comunicação entre o cliente de *email* e o Gestor do Repositório e entre o Cliente e o Repositório de Ficheiros. Inicialmente, para assegurar um bom funcionamento e segurança foi criado o Gestor de Repositório utilizando *sockets*, como explicado anteriormente, devido ao facto de possibilitar um processo de comunicação mais livre, o que tornou mais fácil implementar os comandos e/ou as acções. Por fim, para efectuar o envio/recepção dos ficheiros entre o cliente e o Repositório utilizaram-se dois protocolos bem conhecidos e de larga utilização para transferência de ficheiros: o *FTP* e o *SFTP,* permitindo este último a transferência de dados de forma encriptada, fazendo pois com que os ficheiros dos utilizadores não sejam tão expostos a possíveis intercepções por terceiros.

Seguidamente foram tomadas algumas decisões relacionadas com o processo de implementação e desenvolvimento da arquitectura, as quais debruçaram-se sobretudo na escolha das linguagens de programação a utilizar para as várias funcionalidades implementadas neste projecto. Enquanto que o *XUL* é uma linguagem obrigatória para o desenvolvimento de extensões para o *Thunderbird* as restantes não o são. Para interagir com o *XUL* é possível usar qualquer linguagem de *script*, como acontece no desenvolvimento de paginas *HTML*. O *Thunderbird* utiliza o *JavaScript* para efectuar esta interacção, implementando várias funcionalidades extras, as quais são úteis para o desenvolvimento da arquitectura. Além do mais, toda a documentação e tutoriais utilizam-na. Por estas razões, a escolha prendeu-se pela utilização desta linguagem para interagir com o *XUL*.

Uma vez que o filtro já estava implementado (sendo unicamente modificado para realizar as tarefas propostas pela arquitectura) recorrendo à linguagem de programação *Python,* esta linguagem foi mantida e utilizada para as modificações pretendidas no processo de filtragem. Para a partilha de filtros utilizaram-se duas linguagens de programação distintas, o *XML* e o *Java*. A primeira, utilizada nas mensagens entre o GR e

o cliente para efectuar o pedidos, foi escolhida por ser uma linguagem de anotação largamente implementada nas *APIs* de outras linguagens (para que a mensagem seja interpretada). Já a segunda foi utilizada no desenvolvimento do GR e na implementação da comunicação entre o cliente e o RF. No que se refere à utilização de Java no desenvolvimento do GR, isto deveu-se à simplicidade da interpretação de *XML a partir do Java*. Relativamente à utilização de Java na implementação dos protocolos de comunicação com RF, este facto foi motivado pela já existência de *APIs* para realizar as tarefas de transferência. Para além destes argumentos, é ainda de referir que o *Java* é independente do sistema operativo em utilização, o que torna a extensão mais independente do sistema operativo que o utilizador possui.

4.4 Conclusões

Este capítulo centrou-se na descrição de um sistema colaborativo de combate ao *spam* que foi desenvolvido neste projecto. Desta forma, a arquitectura aqui apresentada propõe uma solução baseada em filtragem, sendo no entanto distribuída e assumindo um cariz colaborativo, tirando pois partido da crescente colaboração e organização em redes sociais dos diversos intervenientes na rede Internet. Neste contexto, foram descritos em detalhe os diferentes componentes de uma arquitectura colaborativa para detecção de *spam*.

As questões de privacidade foram também levadas em conta na solução desenvolvida, visto que os filtros utilizados são recursos sensíveis para o utilizador. É por esta razão que a solução proposta mantém o anonimato no nome dos ficheiros distribuídos e assenta num nó central autónomo e que se quer de confiança.

Por fim foram também apresentados exemplos a ilustrar todo o processo de funcionamento da arquitectura descrita bem como enumeradas e descritas as principais decisões tomadas tanto na estruturação da arquitectura, bem como no processo da sua implementação.

5 Experiências e Resultados

Após o desenvolvimento da arquitectura anteriormente explicada, mas antes de se colocar o sistema desenvolvido sob utilização real, procedeu-se a dois testes laboratoriais preliminares que incidiram sobre questões relacionadas com as estratégias de filtragem. A primeira experiência utiliza um conjunto de mensagens de uma mesma conta de *email* real (contendo aproximadamente 5000 mensagens) recebidas ao longo de vários anos. Já a segunda experiência utiliza um repositório público (Enron), composto por vários utilizadores, sendo desta forma mais realista do que a primeira experiência para o teste do sistema em causa.

Neste contexto, inicialmente serão descritas as métricas e o tipo de gráficos que permitiram avaliar os resultados das experiências realizadas. Posteriormente explicar-se-á como foram configuradas as experiências que utilizaram um conjunto de mensagens de uma conta de *email* real, sendo os resultados dessa experiência posteriormente apresentados. Esta primeira experiência tem também o objectivo de explicar os processo de aprendizagem e colaboração inerentes ao sistema desenvolvido.

O capítulo termina com a descrição e apresentação dos resultados da segunda experiência realizada. Esta revela-se de maior importância pois nesse caso foram efectivamente utilizadas diferentes caixas de correio electrónico, pertencentes a diferentes indivíduos e com um maior número de mensagens, sendo pois uma experiência mais realista para aferir a viabilidade do sistema e das técnicas propostas.

5.1 Curvas ROC

A curva *Receiver Operating Characteristics* (ROC) é uma representação gráfica que mede o desempenho de classificadores binários (i.e. cuja saída tem duas classes possíveis), sendo por isso comummente utilizada para comparar e seleccionar filtros [31, 32]. Desde à muito tempo que se usufrui das curvas ROC em diversas áreas cientificas, exemplo disso é a área da teoria dos sinais ou as áreas médicas [33, 34]. Nos dias correntes, existe um aumento da utilização destas curvas, visto que estudos comprovam que a métrica convencional de precisão (i.e. taxa global de acertos) não é a mais apropriada [35, 36].

Para calcular uma curva ROC, basta ter um classificador cuja saída seja do tipo probabilística, ou seja, que retorne uma probabilidade para uma dada classe $p \in [0,1]$. Fixa-se então um ponto de decisão D, sendo que se assume a classe como positiva se $p>D$, caso contrário a classe prevista é negativa. Mapeando-se depois o conjunto de valores previstos (do tipo Positivo ou Negativo) com o conjunto de valores desejados,

é possível criar uma matriz 2x2, designada de matriz de confusão (Tabela 5), a qual possibilita a computação das fórmulas que geram o gráfico da curva ROC [31].

		Valores Previstos pelo Classificador	
		P	**N**
Valores Desejados	**P**	1 Verdadeiros Positivos (VP)	3 Falsos Negativos (FN)
	N	2 Falsos Positivos (FP)	4 Verdadeiros Negativos (VN)

Tabela 5: Matriz de Confusão

Quando se tem um classificador e um exemplo de teste, pode acontecer uma de quatro situações:

- Ambos os valores, do exemplo (desejado) e previsto pelo classificador, são positivos, obtendo-se um Verdadeiro Positivo (VP) – quadrado 1 da Tabela 5;
- O valor desejado é negativo, enquanto que o previsto pelo classificador é positivo, pelo que se obtêm um Falso Positivo (FP) – quadrado 2 da Tabela 5;
- O valor do exemplo é positivo, enquanto que o previsto pelo classificador é negativo, obtendo-se um Falso Negativo (FN) – quadrado 3 da Tabela 5;
- Ambos os valores são negativos, obtendo-se um Verdadeiro Negativo (VN) – quadrado 4 da Tabela 5.

Com base nestas classificações da matriz, é possível construir a curva ROC pela aplicação das seguintes fórmulas:

$$TFP = \frac{FP}{TN}, \quad (13)$$

$$TVP = \frac{VP}{TP} \text{ , (14)}$$

sendo TFP a taxa de falsos positivos e TVP a taxa de verdadeiros positivos. Os valores de *TN* e *TP* são calculados através das equações:

$$TN = VN + FP \text{ , (15)}$$

$$TP = VP + FN \text{ , (16)}$$

Assim, a curva ROC descreve a relação entre a TVP (representado na Figura 14 como "Rácio de Verdadeiros Positivos", eixo dos *y*) e a TFP (representado na Figura 14 como "Rácio de Falsos Positivos", eixo dos *x*), i.e., entre os benefícios e os custos.

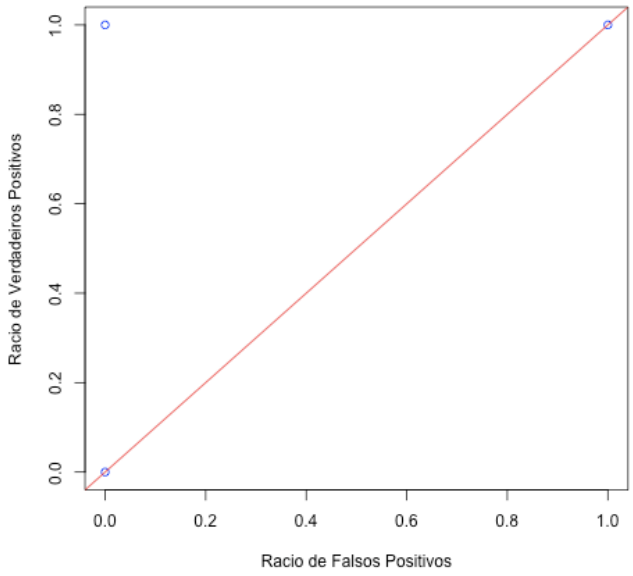

Figura 14. Pontos Importantes no gráfico ROC

Observando o gráfico da figura 14 conclui-se que alguns pontos assinalados na curva ROC são de grande importância [31]. Estes são:

- O ponto (0,0), que representa um ponto de corte (decisão) de *D* >1. Como todas previsões estão abaixo do valor de *D*, o classificador prevê sempre a classe como negativa, pelo que não origina falsos positivos;

- No outro extremo, o ponto (1,1) representa o local onde $D < 0$, sendo que o classificador prevê sempre exemplos positivos;
- Já o ponto (0,1) é a classificação óptima, dado que qualquer amostra negativa do teste será classificada como tal, ocorrendo o mesmo no caso contrário.

Na realidade, é raro que uma curva ROC passe pelo ponto perfeito (0,1), sendo então que uma curva é melhor se a sua curvatura se aproximar deste ponto.

5.2 Métricas

De forma a possibilitar a avaliação das experiências realizadas utilizaram-se duas métricas de avaliação de filtros, a *Area Under the Curve* (AUC) e a TPR@FPR.

A primeira métrica corresponde à área sob a curva (*Area Under the Curve – AUC)* da *Receiver Operating Characteristic – ROC*, sendo calculada com a seguinte expressão:

$$AUC = \int_0^1 ROC \; dD \;, \quad (17)$$

Quanto maior for o valor de AUC, melhor é a capacidade de discriminação do modelo estatístico [37]. Um classificador aleatório tem uma AUC=0.5, enquanto que o modelo perfeito corresponde a uma AUC de 1.0. Esta métrica permite assim comparar curvas ROC, sendo uma das mais utilizadas [38, 39].

A segunda métrica devolve a TPR[24] (do português TVP) para uma dada FPR[25] (do português TFP) fixa. Uma vez que na detecção de *spam*, os FP têm um custo superior, i.e. é mais perigoso classificar uma mensagem legítima como *spam*, normalmente utilizam-se valores baixos de FPR. Nas experiências efectuadas optou-se por um limite de erro de t=0.01 (1%) [40].

[24] TPR – *True Positive Rate*

[25] FPR – *False Positive Rate*

5.3 Configuração da experiência da Conta de *Email* Real

As experiências realizadas decorreram num ambiente laboratorial e tiveram em consideração um conjunto de mensagens previamente recolhidas da conta de *email* do autor deste trabalho, contendo 5000 mensagens manualmente classificadas em *spam* e *ham*. Acima de tudo, com estas experiências pretende-se demonstrar que a arquitectura proposta neste trabalho realmente funciona. As mensagens foram divididas, de modo aleatório, por três caixas de correio, com o objectivo de simular três utilizadores distintos. A Tabela 6 indica as informações relativas ao resultado desta divisão, i.e., as informações das mensagens que constituem cada caixa de correio utilizadas nesta experiência.

Caixa	Data da Primeira Mensagem	Data da Última Mensagem	Número de Mensagens
1	29/06/2006	06/07/2009	500
2	27/02/2006	05/07/2009	500
3	06/02/2006	05/07/2009	500
		Total	1500

Tabela 6: Informação das mensagens de cada caixa

Sendo este um teste de laboratório decorrido localmente, toda a experiência utilizou a mesma máquina para tal. Simulou-se também todas as trocas das mensagens entre o servidor POP/SMTP e o cliente de *email* e entre este e o GR/RF. O sistema operativo da máquina usada para estes testes foi o *Mac OS X - 10.5* ("*Leopard*") da *Apple*, com a versão desenvolvida para este sistema operativo do *Thunderbird 2.0* como cliente de *email.*

Como foram desenvolvidos dois métodos diferentes (denominados e descritos no capitulo anterior como M1 e M2), estes serão comparados com os resultados do filtro individual (utilizado pelo *ThunderBayes* na sua forma original). Para esta comparação será necessário obter-se dados estatísticos, de modo a avaliar a possível melhoria conseguida com esta arquitectura e a sua viabilidade, dependendo do método utilizado. Nesse sentido, em cada caixa de correio foi necessário executar a mesma experiência, com os três filtros.

Assim, dividiram-se as mensagens de cada caixa de correio de modo a treinar o filtro. O conjunto de treino (*training set)* foi criado e usado em todos os testes da mesma caixa de correio e utilizado na simulação da partilha dos filtros para os métodos M1 e M2 de forma a permitir as experiências realizadas na caixa em questão. Por essa razão dividiu-se cada caixa de correio (total de 500 mensagens) nas primeiras 334 mensagens para treino e restantes 166 mensagens, tal como indica a Tabela 7.

Caixa	Treino[26]	Teste[27]	Número de *Spam*[28]	Número de *Ham*[29]
1	334	166	223	111
2	334	166	230	104
3	334	166	227	107
Total	1002	498	680	322

Tabela 7: Divisão e classificação das mensagens

Inicialmente para a caixa de correio 1, utilizou-se o filtro individual com o *training set* obtido no passo anterior. Assim, reenviaram-se as 166 mensagens de teste obtendo para cada mensagem uma probabilidade final. Esta probabilidade foi gravada num ficheiro para posterior análise. O mesmo foi efectuado para cada uma das restantes caixas de correio.

Seguidamente, trocou-se a extensão original pela extensão que contem o método M1, procedendo-se ao *download* dos *training sets* contidos no servidor, sendo estes os ficheiros gerados pela classificação das restantes duas caixas de correio. Reenviou-se novamente as mensagens de teste da caixa de correio 1, utilizando também o *training set* resultante do treino. Consequentemente, uma probabilidade para cada *email* foi obtida, inicialmente de cada ficheiro importado e do próprio e, de seguida, calculou-se a probabilidade final, como explicado no capitulo 4. Estas probabilidades são guardadas num ficheiro, que é constituído não só por estas probabilidades, mas também pelo número de *email* correspondente. Estes passos foram repetidos para cada caixa de correio, mas os *training sets* foram trocados pelos seus correspondentes.

Por fim, a extensão que continha o M1 foi alterada para a extensão que envolvia o método M2. Repetiram-se os passos anteriores, mas desta vez geraram-se também os pesos de cada ficheiro importado no momento do *download* dos mesmos. Estes pesos, bem como as probabilidades calculadas durante a classificação utilizando os ficheiros importados, o próprio ficheiro, e a probabilidade final calculada são guardados para futura análise de resultados.

[26] Número de mensagens usadas no treino (escolhidas aleatoriamente das 500 correspondentes)

[27] Número de mensagens usadas para realizar as experiencias (escolhidas aleatoriamente das 500 correspondentes)

[28] Mensagens manualmente classificadas como *spam* a partir das mensagens de treino correspondentes

[29] Mensagens manualmente classificadas como *ham* a partir das mensagens de treino correspondentes.

5.4 Resultados da Conta de *Email* Real

Sabendo as probabilidades de cada filtro, em relação a cada mensagem de teste recebida, para cada caixa de correio simulada nesta experiência e a certeza dada para essa mesma mensagem, foi possível obter os resultados estatísticos para cada uma das métricas apresentadas. Assim, também foi possível construir o gráfico da curva ROC. Neste gráfico foram comparados os três métodos usados pelo sistema para cada uma das caixas de correio.

De seguida, serão apresentadas as curvas ROC geradas em R[30] correspondentes a cada uma das caixas de correio simuladas nesta experiência, e respectivos métodos de filtragem.

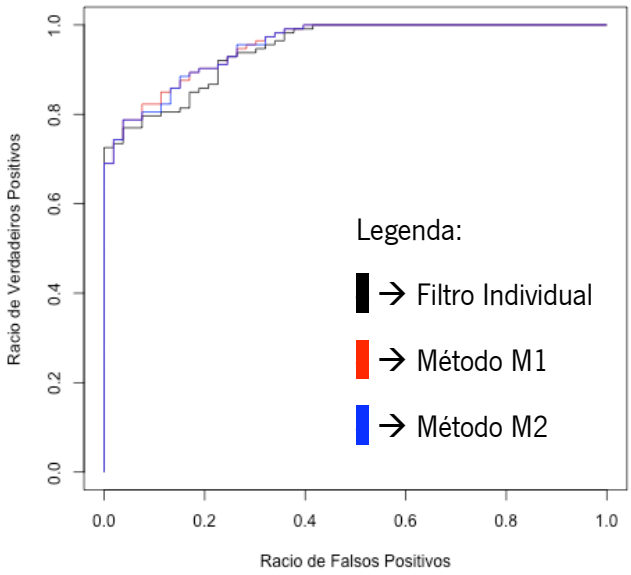

Figura 15. Curvas ROC para a caixa de correio 1

Observando o gráfico da Figura 15, relativo à caixa de correio 1, é possível verificar que em relação ao filtro individual, os métodos M1 ou M2 apresentam, regra geral, melhores resultados. A melhoria dos métodos que a arquitectura utiliza é nitidamente melhor quando a TFP (assinalado no gráfico como "Rácio de Falsos Positivos") é compreendida aproximadamente entre os valores 0.1 e 0.25. É neste conjunto de valores que as curvas de M1 e de M2 se aproximam mais dos valores óptimos da TVP (assinalado no gráfico como "Rácio de verdadeiros Positivos"), i.e., é neste intervalo que estes métodos se aproximam mais do ponto de coordenadas (0,1).

[30] *Software* e linguagem de programação utilizada para cálculos estatísticos (ver www.r-project.org).

	AUC		
Caixa	**Ind.**	**M1**	**M2**
1	94.62%	**95.44**%	95.34%

Tabela 8: Valores de AUC para caixa de correio 1

Após a análise deste gráfico, observa-se que os métodos M1 e M2 obtêm melhores resultados comparativamente ao filtro individual para a caixa de correio 1. No entanto, sobre M1 e M2 não se podem tirar certezas absolutas sobre qual o melhor, dado que as curvas se igualam na maioria dos pontos. Existem porém, alguns pontos nos quais M1 se destaca, embora não se consiga retirar nenhuma conclusão do mesmo. Para tal, é necessário comparar a área de cada curva, utilizando a métrica AUC. Os valores presentes na Tabela 8 indicam que M1 apresenta melhores resultados, embora sem significado estatístico visto que a diferença entre M1 e M2 é de 0.1%.

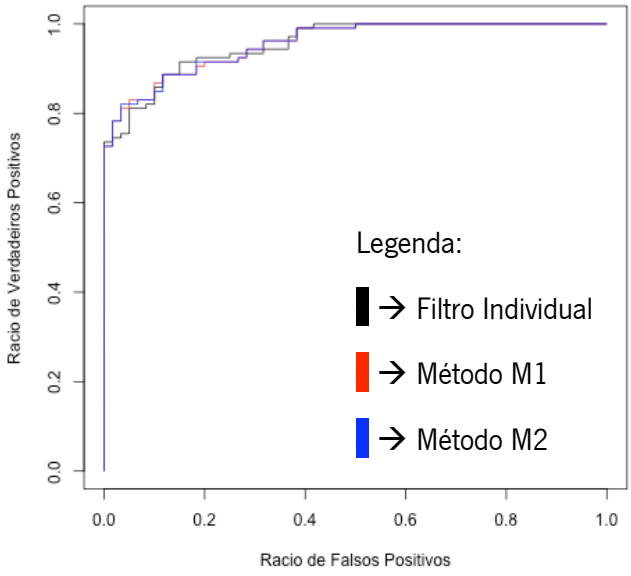

Figura 16. Curvas ROC para a caixa de correio 2

Já no gráfico da Figura 16 da curva ROC referente à caixa de correio 2, numa primeira análise, as curvas são em geral muito próximas. Isto quer dizer que os métodos M1 e M2 têm resultados aceitáveis, mas não foram melhores que os do filtro individual. Numa análise mais cuidada, verifica-se que em certos pontos,

os métodos M1 e M2 são melhores que o filtro individual. Por exemplo, em pontos compreendidos entre 0.02 e 0.06 no eixo da TFP (assinalado no gráfico como "Rácio de Falsos Positivos").

Mesmo havendo uma melhoria nestes pontos, não é possível concluir com qual dos métodos, M1 ou M2, se obtêm um melhor desempenho. A impossibilidade de uma conclusão deve-se ao facto de as curvas se igualarem na maioria dos pontos.

	AUC		
Caixa	**Ind.**	**M1**	**M2**
2	95.60%	**95.64**%	95.63%

Tabela 9: Valores de AUC para a caixa de correio 2

Pela análise da Figura 16, é difícil concluir com convicção sobre qual dos filtros teve melhor desempenho. Na verdade, é necessário observar os valores da métrica AUC que se encontram expostos na Tabela 9. É de salientar que a diferença entre os três métodos é bastante reduzida, contudo existe uma diferença. A diferença não possui significado estatístico mas, indica que o método M1 apresenta um desempenho ligeiramente superior ao dos restantes.

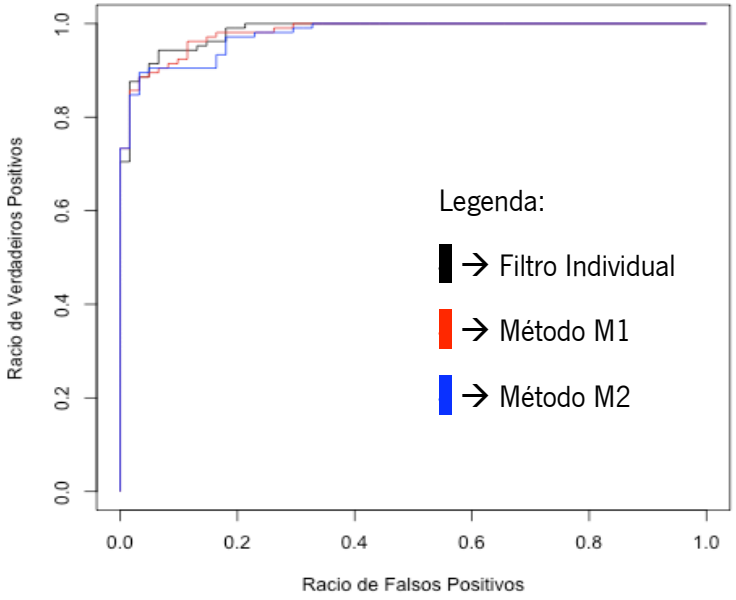

Figura 17. Curvas ROC para a caixa de correio 3

Contrariamente aos gráficos referentes às caixas de correio 1 e 2 que demonstraram alguns resultados animadores, relativamente ao desempenho da arquitectura desenvolvida, o gráfico para a caixa de correio 3 não apresenta resultados tão positivos. A Figura 17 permite verificar as curvas geradas na caixa de correio 3, que no global apresentou um desempenho melhor quando se utilizou o filtro individual. No entanto, observando o gráfico ao pormenor é possível perceber que os métodos M1 e M2 possuem um desempenho superior em duas zonas distintas.

A razão que leva a que os resultados não sejam tão favoráveis deve-se provavelmente ao facto de se terem gerado as três caixas de correio a partir de um mesmo utilizador. Assim sendo, os *training sets* importados são muito idênticos ao *training set* pessoal. Este facto que faz com que a classificação obtida pelos métodos M1 ou M2 seja semelhante, ou em alguns casos até pior, ao que foi obtido pelo filtro individual.

De seguida são apresentados os valores das métricas correspondentes aos testes efectuados em cada caixa de correio simulada. Os valores das métricas encontram-se na Tabela 10, nesta indica-se a média da métrica em relação ao filtro utilizado.

	AUC			**TPR@FPR com Threshold t = 0,01**		
Caixa	**Ind.**	**M1**	**M2**	**Ind.**	**M1**	**M2**
1	94.62%	95.44%	95.34%	72.57%	69.03%	69.03%
2	95.60%	95.64%	95.63%	73.58%	72.64%	72.64%
3	98.38%	98.16%	97.64%	70.48%	73.33%	73.33%
Média	**96.19%**	**96.41%**	**96.20%**	**72.21%**	**71.67%**	**71.67%**

Tabela 10. Resultados obtidos para a experiência da "conta de *email* real"

Ao se analisar a Tabela 10 observa-se que, na métrica AUC, as médias dos métodos M1 e M2 são ligeiramente melhores em comparação com a média do filtro individual, visto que a diferença entre eles é diminuta. Em contrapartida, os valores da métrica TPR@FPR não apontam para uma melhoria no desempenho, uma vez que M1 e M2 são piores que o filtro individual. Neste caso, para t=0.01, M1 e M2 obtiveram valores exactamente iguais.

Como a AUC mede a área ROC, ou seja, a capacidade de discriminação, uma melhoria nos seus valores contribuiu para uma melhor capacidade de discriminação. Por este facto, é possível afirmar que

mesmo com resultados pouco satisfatórios, os métodos que compõe a arquitectura proposta (M1 ou M2) obtiveram uma melhor capacidade de discriminação comparativamente ao filtro individual. O problema centra-se na obtenção de uma melhor TPR em relação a uma FPR fixa de 1% por parte do filtro individual.

Os resultados não foram muito animadores, embora este conjunto de experiências não seja completamente realista: as três caixas de correio pertencem ao mesmo utilizador; por outro lado, foram simuladas poucas caixas de correio, tendo estas poucas mensagens para teste ou para treino.

Na próxima secção será novamente testada a arquitectura desenvolvida através de um repositório público. Este, por sua vez, contém mais caixas de correio (até 5) e que são efectivamente pertencentes a diferentes utilizadores.

5.5 Experiências e Resultados do Repositório *Enron*

Face ao observado na experiência relatada na secção anterior, foi decidido alargar os testes a cenários que envolvessem caixas de correio electrónico de diferentes utilizadores e um maior volume de mensagens, para esse efeito recorreu-se à utilização de um repositório público. Como se irá referir, os resultados conseguidos nesta experiência são mais encorajadores, sendo agora mais visível o ganho da utilização do sistema proposto face ao modelo tradicional de filtragem.

Os resultados aqui apresentados foram obtidos pelo repositório público *Enron [41]*, usando as mensagens de *ham* dos cinco utilizadores com maiores caixas de correio (*Kam, Far, Bec, Lok e Kit*). De realçar que se pode assumir que estes utilizadores possuem perfis de exposição à Internet relativamente semelhantes, dado que pertencem todos à mesma empresa. Quanto ao spam, este foi colectado do repositório público de *Bruce Guenter* (http://untroubled.org/spam), que é baseado em armadilhas de *spam* (falsos endereços de email que são divulgados na Internet). As mensagens de *ham* e *spam* foram misturadas de um modo realista, de acordo com a sua data de recepção. Por sua vez, o algoritmo de mistura utilizou um parâmetro de selecção de P=0.5, o que faz com que quaisquer dois utilizadores tenham cerca de 50% do mesmo *spam* em comum.

Para cada uma das métricas (AUC e TPR@FPR com t=0.01) também foram testados os métodos: *(i)* Filtro Individual (**Ind.**), *(ii)* Método 1 (**M1**), e *(iii)* Método 2 (**M2**).

A titulo de exemplo, a Figura 18 representa a curva *ROC* para o utilizador *Bec* e usa os vários métodos acima referidos. O gráfico demonstra que os classificadores que pertencem aos métodos M1 e M2, correspondentes aos exemplos implementados na arquitectura proposta neste trabalho, obtiveram um melhor desempenho em relação ao classificador do filtro individual. Neste caso, os métodos M1 e M2 têm

performances similares, pois as curvaturas dos dois métodos são mais próximas do ponto (0,1) do gráfico em relação à curvatura resultante do filtro individual.

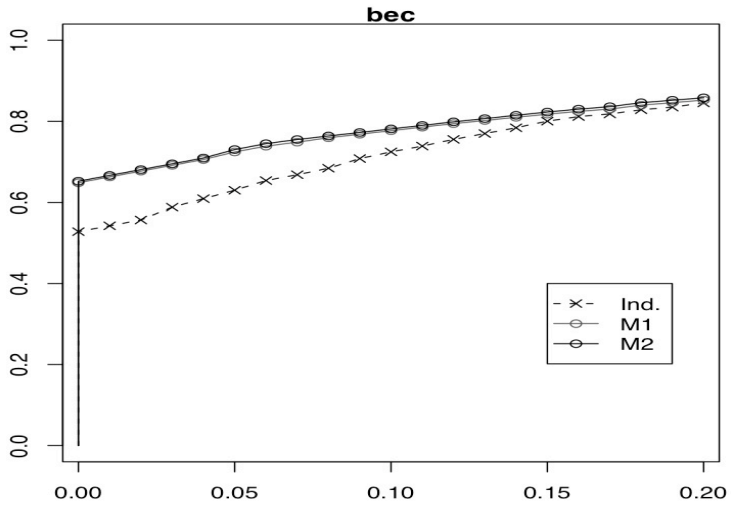

Figura 18. Curvas ROC para o utilizador Bec do *Enron*

Já a Tabela 11 descreve um sumário dos resultados obtidos (valores médios de diversas simulações, em %) nos testes para cada uma das métricas e, dentro destas, para cada uma das técnicas utilizadas.

	AUC			**TPR@FPR com Threshold t = 0,01**		
	Ind.	**M1**	**M2**	**Ind.**	**M1**	**M2**
Kam	62.1%	**94.5%**	**94.5%**	1.5%	67.8%	**67.9%**
Far	**93.5%**	92.2%	92.9%	19.9%	66.5%	**66.8%**
Bec	91.5%	92.1%	**92.2%**	54.2%	66.3%	**66.7%**
Lok	91.4%	**93.8%**	93.7%	**78.0%**	72.7%	73.1%
Kit	74.6%	**94.5%**	94.3%	18.9%	**71.7%**	71.1%
Média	**82.6%**	**93.4%**	**93.5%**	**34.5%**	**69.0%**	**69.1%**

Tabela 11. Resultados obtidos para a experiência da utilizando o *Dataset*

Através da análise dos resultados conclui-se que tanto o método M1 como o método M2 apresentam melhores resultados do que o filtro individual. Embora sem significado estatístico, para o repositório estudado, o método M2 é ligeiramente melhor. Isto poderá ser explicado pela proximidade entre os perfis de utilização do serviço de correio electrónico dos vários utilizadores considerados. Ou seja, não existe nenhum filtro de um

utilizador específico que seja extremamente diferente dos utilizados pelos outros utilizadores. Se este fosse o caso, o método que efectua ponderação (M2) deveria apresentar, de uma forma notória, melhores resultados do que a simples média. De qualquer modo, ambos os métodos apresentam resultados encorajadores em comparação com o modelo individual. Em particular, existe uma diferença bastante significativa em termos da métrica TPR@FPR na comparação entre os métodos M1 e M2 e o filtro individual (aproximadamente 34% para o filtro individual e 69% para M1 e M2), o que é deveras relevante para a problemática específica da detecção de *spam*.

5.6 Conclusões

Este capítulo focou-se nos testes preliminares efectuados à arquitectura desenvolvida. Neste contexto, executaram-se duas experiências diferentes, alterando o conjunto de mensagens utilizadas e o número de participantes. Na primeira experiência, o conjunto de mensagens eram todas provenientes de apenas uma conta de *email*, simulando-se os utilizadores pela divisão em três caixas de correio de 500 mensagens cada. A segunda experiência decorreu através de um *dataset* público, o *Enron,* no qual a proveniência das mensagens *ham* era de 5 utilizadores diferentes, pertencentes à mesma empresa.

Os resultados obtidos podem-se considerar positivos, o que demonstrou a possibilidade de sucesso da arquitectura desenvolvida. Os resultados provenientes das mensagens disponíveis na conta de *email* não foram os melhores comparando com os do filtro individual, pois as mensagens das caixas de correio simuladas são todas provenientes da mesma conta de *email*. Outra razão para estes resultados deve-se ao número reduzido de utilizadores simulados, i.e., apenas se simulou três caixas de correio. Estes resultados foram claramente superados pelos obtidos nos testes com o *dataset Enron* visto que aqui existem mais utilizadores para simular a partilha dos seus *training sets*. Outro factor a influenciar foi o facto das mensagens contidas no *dataset* serem variadas e cada utilizador ter o seu próprio género de mensagens (como acontece na realidade).

Observando estes resultados espera-se que quantos mais utilizadores partilharem os seus filtros, mais exacta deverá ser a classificação obtida pelo sistema. Tendo isto em consideração, poder-se-á então concluir que a utilização arquitectura desenvolvida poderá efectivamente trazer melhorias aos métodos de combate ao spam existentes. Adicionalmente, convém realçar que os métodos implementados (M1 e M2) são meramente ilustrativos. De facto, através da plataforma desenvolvida será possível testar novas estratégias colaborativas de combinação de filtros, havendo pois a possibilidade de futuros desenvolvimentos deste projecto.

6 Conclusões

Após ter sido introduzida a problemática do *spam* e as suas técnicas de combate, explicou-se detalhadamente a proposta de uma nova abordagem colaborativa de filtragem de correio não solicitado e os resultados preliminares obtidos pela utilização dessa mesma solução. Neste capitulo serão agora mencionadas algumas conclusões relacionadas com o trabalho desenvolvido. Desta forma, inicialmente será efectuado um resumo, sendo este sucedido por uma breve discussão do trabalho executado. O capitulo terminará com uma descrição de todas as ideias para uma futura ampliação deste trabalho.

6.1 Resumo

Ao longo do documento descreveu-se toda a problemática do *spam*, tendo sido apresentadas diferentes técnicas de combate ao mesmo. Estas técnicas são as mais diversas, algumas delas apoiadas juridicamente, outras pela introdução de custos e outras pela utilização de técnicas de filtragem. O trabalho assentou precisamente neste ultimo ponto. Um novo sistema que visa o combate ao *spam* foi apresentado neste projecto, juntando duas das principais técnicas utilizadas, a colaborativa e a heurística do conteúdo da mensagem.

A implementação da arquitectura foi estudada por forma a se obter um sistema final o mais funcional possível e apto a ser utilizado pelo máximo de número de utilizadores possível. Por tal, usou-se o *Thunderbird* como cliente de *email*, sendo a arquitectura implementada através de uma extensão para este. De forma a facilitar a tarefa de implementação aproveitou-se uma extensão já existente (o *ThunderBayes*) como ponto de partida para o desenvolvimento do sistema do cliente, dado que a extensão já implementa a filtragem heurística recorrendo a um algoritmo denominado *Naive Bayes*.

A interface criada foi também pensada para ser fácil de utilizar, de modo a que mesmo o utilizador mais inexperiente não tenha dificuldades para interagir com o sistema. Para tornar isto possível, empregou-se um conjunto de botões, adicionáveis ao painel principal do *Thunderbird* para permitir os processos de treino e a partilha de filtros.

No contexto do trabalho desenvolvido foram realizados dois testes preliminares para verificar o desempenho e a viabilidade da arquitectura desenvolvida. O primeiro recorreu a uma listagem de mensagens de uma caixa de correio, sendo esta dividida aleatoriamente em três caixas de correio simuladas. O segundo

usou um repositório público conhecido, o *Enron*. Neste último teste foram utilizados cinco utilizadores diferentes que pertencem ao repositório. Os resultados obtidos, e em particular os obtidos na segunda experiência, levam a afirmar que a arquitectura tem de facto viabilidade. Adicionalmente, novos métodos colaborativos poderão no futuro ser implementados e testados utilizando a arquitectura desenvolvida neste trabalho.

Como resultado de todo o trabalho desenvolvido neste projecto, e que foi documentado nesta dissertação, foi publicado um artigo científico numa conferencia nacional na temática das Redes de Computadores [42].

6.2 Discussão

Na realização da arquitectura proposta foi utilizada e alterada uma extensão já existente, o *ThunderBayes*, que utiliza o algoritmo *Naive Bayes* na classificação das mensagens. Como é óbvio poderia ter sido implementado um filtro de raiz, mas visto que o propósito do trabalho foi uma prova de conceito de uma arquitectura nova que surgiu de técnicas já existentes optou-se pela utilização desta extensão. Adicionalmente, uma outra razão para essa opção deveu-se ao facto do tempo que seria necessário aplicar para desenvolver uma solução de raiz. Dessa forma a opção de alterar e adaptar um extensão já existente para o cliente seleccionado foi aquela que se afigurou mais proveitosa.

O tempo foi também uma condicionante neste trabalho, assim a implementação teve de ser simplificada. Por tal, remeteram-se para trabalho futuro outros métodos de colaboração e ideias que teriam sido proveitosas para esta solução. Pela mesma razão, os testes realizados foram simples, sendo só possível constatar até que ponto a arquitectura tem bons desempenhos em laboratório. Outros testes, mais realistas, seriam vitais para a real medição dos benefícios e desempenho que a solução terá num ambiente de utilização real.

Os resultados obtidos foram animadores, e por isso deram uma indicação sobre o possível bom desempenho que a solução terá. Como foi explicado, os resultados melhoraram quando foi considerado um cenário com um número maior de utilizadores na simulação de partilha de filtros. Em suma, a indicação é que quanto maior for a rede de utilizadores a partilhar os filtros, melhor tenderão a ser os resultados obtidos. Por tal, seria necessário ampliar os testes para situações mais realistas, com mensagens mais vastas e com mais utilizadores na simulação de forma a se ter uma maior certeza sobre este facto, mas por limitações temporais remeteu-se este ponto para trabalho futuro.

6.3 Trabalho futuro

Existem muitas ideias possíveis para trabalho futuro de forma a alargar e a melhorar a solução aqui desenvolvida. Assim, alguns destes aspectos podem melhorar a segurança, a proliferação da extensão desenvolvida, a confiança nos resultados devolvidos ao cliente e a estabilidade da extensão desenvolvida. Algumas das possibilidades de expansão e melhoramento da solução podem ser as seguintes:

- Implementar um sistema de autentificação da extensão junto do sistema desenvolvido por forma a possibilitar uma maior segurança na partilha de ficheiros. Este sistema deve contar com um meio de registo para que seja possível a um novo utilizador utilizar a extensão desenvolvida. Uma possibilidade para este registo é a utilização da pagina Web desenvolvida para a distribuição da extensão. É necessário também implementar na extensão um mecanismo que permita ao utilizador introduzir os seus dados de autentificação;
- Desenvolvimento do repositório de *training sets* de forma a ser possível a categorização dos mesmos. Dessa forma seria possível obter-se colecções de *training sets* pertencentes a utilizadores com perfis semelhantes (e.g. profissão, assuntos de interesse, *etc.*);
- Alargar a implementação da arquitectura proposta a outros clientes de *email* menos utilizados ou que não sejam de código aberto, de modo a abranger utilizadores que não usam o *Thunderbird* como cliente de *email*;
- Implementação de um método simbiótico (M3) para a avaliação da importância a atribuir aos ficheiros importados para o cálculo da probabilidade final. Este novo método, implementado com o algoritmo *Naive Bayes*, possibilitaria avaliar a importância dos *training sets* importados de forma inteligente;
- Realização de experiências mais fiáveis, sendo estas o mais reais possíveis, i.e., a realização de testes com caixas de correios e utilizadores reais. Os testes não deverão ser só laboratoriais. Caso não seja possível, os testes devem ser realizados com um número maior de caixas de correio distribuídas por vários sistemas.

7 Referência Bibliográficas

[1] J. C. Sipior, B. T. Ward, and P. G. Bonner, "*Should Spam Be On The Menu?*", *Communications of the Association for Computing Machinery (ACM*). vol. 47 ACM, pp. 59-64, June 2004.

[2] Y. Zhou, M. S. Mulekar, and P. Nerellapalli, "*Adaptive Spam Filtering Using Dynamic Feature Space*", *17th IEEE International Conference on Tools with Artificial Intelligence* ICTAI'05, November 2005.

[3] F. Fdez-Riverola, E. Iglesias, F. Diaz, J. Mendez, and J. Corchado, "*SpamHunting: An Instance-Based Reasoning System for Spam Labeling and Filtering*", *Decision Support Systems Journal*. vol. 43(3), pp. 722-736, 2007.

[4] D. Miyamoto, H. Hazeyama, and Y. Kadobayashi, "*Detecting Methods of Virus Email Based on Mail Header and Encoding Anomaly*", *International Conference on Neural Information Processing (ICONIP),* November 2008.

[5] A. Corporation, "*The Changing Face of Exchange Email Management*", September 2005.

[6] R. Segal, "*Combining Global and Personal Anti-Spam Filtering*", *Proceedings of the Fourth Conference on Email and Anti-Spam*, August 2007.

[7] A. C. Hubmann-Haidvogel, "*ThreadVis for Thunderbird: A Thread Visualisation Extension for the Mozilla Thunderbird Email Client*", September 2008.

[8] TomFawcett, "*"Invivo" Spam FItering: A Challenge Problem for Data Mining*", *Knowledege Discovery and Data Mining (KDD) Explorations,* vol. 5, pp. 140-148, December 2003.

[9] E. Blanzieri and A. Bryl, "*A Survey of Learning-Based Techniques of Email Spam Filtering*", *Technical Report # DIT-06-056,* January 2008.

[10] F. D. Garcia, J.H. Hoepman, and J. Nieuwenhuizen, "*Spam Filter Analysis*", *Proceedings of 19th International Federation for Information Processing (IFIP) International Information Security Conference* WCC2004-SEC, February 2004.

[11] S. Hershkop and S. Stolfo, "*Combining Email Models for False Positive Reduction*", *Proc. Of The Eleventh ACM SIGKDD - International Conference on Knowledge Discovery and Data Mining* ACM, August 2005.

[12] V. Metsis, I. Androutsopoulos, and G. Paliouras, "*Spam Filtering with Naive Bayes - which Naive Bayes?*", *Third Conference on Email and Anti-Spam* CEAS 2006, July 2006.

[13] G. Klyne and J. Palme, "*Registration of Mail and MIME Header Fields*", *RFC4021*, March 2005.

[14] T. Gray, "*Message Access Paradigms and Protocols*", *Networks & Distributed Computing* Washington: University of Washington, September 1995.

[15] J. Myers and M. Rose, "*Post Office Protocol - Version 3*", *RFC1939*, May 1996.

[16] T. Gray, "*Comparing Two Approaches to Remote Mailbox Access: IMAP vs. POP*", *Networks & Distributed Computing* Washington: University of Washington, September 1995.

[17] J. B. Postel, "*Simple Mail Transfer Protocol*", *RFC821*, August 1982.

[18] A. K. Systems, "*Anti-Spam White Paper*", *Aladdin Knowledge Systems,* 2003.

[19] A. Limited, "*Whitepaper: Spam A Corporate Concern*", *AltoHiway Limited,* January 2004.

[20] A. Ramachandran and N. Feamster, "*Understanding the Network-Level Behavior of Spammers*", Special Interest Group on Data Communications (*SIGCOMM'06)*, September 2006.

[21] P. Barford and V. Yegneswaran, "*An Inside Look at Botnets*", *Computer Sciences Department* Madison: University of Wisconsin, 2006.

[22] S. Hird, "*Technical Solutions for Controlling Spam*", *Australian UNIX and Open Systems User Group* (*AUUG2002)*, September 2002.

[23] D. Madigan, "*Statistics and The War on Spam*", Rutgers University, 2004.

[24] A. McCallum and K. Nigam, "*A Comparison of Event Models for Naive Bayes Text Classification*", *Association for the Advancement of Artificial Intelligence (AAAI) Technical Report WS-98-05*, 1998.

[25] B. Dyszel, "*Outlook 2007 for Dummies*", Wiley Publishing, Inc., *2007*.

[26] P. G. Aitken, "*Outlook 2007 Bible*. Indianapolis", Wiley Publishing, Inc., 2007.

[27] K. C. Feldt, "*Programming Firefox*", O'Reilly Media, Inc., 2007.

[28] D. Goodman and M. Morrison, "*JavaScript Bible"*, 5th Edition ed., Wiley Publishing, Inc., 2004.

[29] E. Foster-Johnson, J. C. Welch, and M. Anderson, "*Beginning Shell Scripting*", Wiley Publishing, Inc., 2005.

[30] M. Lutz, "*Programming Python",* 3rd Edition ed., O'Reilly Media, Inc., 2006.

[31] D. A. O. Gomes, "*Detecção de Intrusão em Redes de Computadores Utilizando Classificadores One-Class*", *Engenharia da Computação* Universidade de Pernambuco, 2006.

[32] T. Fawcett, "*An Introduction to ROC Analysis.*", *Institute for the Study of Learning and Expertise, 2005. Pattern Recognition Letters,* vol. 27, pp. 861-874, 2006.

[33] J. P. Egan, "*Signal Detection Theory and ROC Analysis*", *Series in Cognition and Perception, Academic Press,* 1975.

[34] J. A. Swets, R. M. Dawes and J. Monaham, "*Better Decisions Through Science*", *Scientific American* vol. 283, pp. 82-87, 2000.

[35] F. Provost and T. Fawcett, "*Analysis and Visualization of Classifier Performance: Comparison Under Imprecise Class and Cost Distributions*.", *Third International Conference on Knowledege Discovery and Data Mining (KDD-97), AAAI Press, Menlo Park, CA,* pp. 43-48, 1997.

[36] F. Provost, T. Fawcett, and R. Kohavi, "*The Case Against Accuracy Estimation for Comparing Induction Algorithms*", *Shavlik, J. (Ed.), Proc. ICML-98. Morgan Kaufmann, San Francisco, CA,* pp. 445-453, 1998.

[37] A. M. Silva, "*Utilização de Redes Neurais Artificiais para Classificação de Spam*", Março de 2009.

[38] A. P. Bradley, "*The Use of Area Under the ROC Curve in the Evaluation of Machine Leaning Algorithms*", *Pattern Recognition Letters,* vol. 30, pp. 1145-1159, 1997.

[39] J. A. Hanley and B. J. Mcneil, "*The Meaning and Use of the Area Under a Teceiver Operating Characteristic (ROC) Curve*," *Radiology,* vol. 143, pp. 29-36, 1982.

[40] M. Chang, W. Yih, and C. Meek, "*Partitioned Logistic Re-gression for Spam Filtering*", *14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* pp. 97-105, 2008.

[41] R. Beckermann, A. McCallum, and G. Huang, "*Automatic Categorization of Email into Folders: Benchmark Experiments on Enron and SRI Corpora*", *University of Massachusetts Amherst,* 2004.

[42] A. Machado, P. Cortez, P. Sousa, "*Filtragem Colaborativa de Correio Electrónico Não Solicitado*", *In Actas da 9ª Conferência sobre Redes de Computadores, Oeiras, Portugal, Outubro de 2009.*

ANEXO 1 – Exemplo De Email

From - Mon Jul 6 11:10:24 2009

X-UIDL: 00000dc547ef7315

Return-path: <slaterrn@iowatelecom.net>

Received: from mx1.dc.iol.pt ([172.16.54.1]) by mstore1.iol.pt (Sun Java System Messaging Server 6.2-4.03 (built Sep 22 2005)) with ESMTP id <0JSJ002KD99KU0F0@mstore1.iol.pt>; Tue, 04 Dec 2007 16:17:44 +0000 (GMT)

Received: from mx1.anubis.local (ste [10.1.2.2]) by mx1.dc.iol.pt (Postfix) with ESMTP id DB3D347E7F; Tue, 04 Dec 2007 16:17:41 +0000 (WET)

Received: from repsolypf.com (unknown [189.24.180.18]) by mx1.anubis.local (Postfix) with ESMTP id A96D823D70; Tue, 04 Dec 2007 16:17:36 +0000 (WET)

Date: Tue, 04 Dec 2007 08:36:47 -0700

From: "Rolando K. Slater" <slaterrn@iowatelecom.net>

Subject: #1 PharmacyOnNet sells Phentermin, Ambien, Codeine, ValiuXanassss.... 5fzqxc1mghk2

ANEXO 2 – Pedido/Resposta XML de Download Entre o Cliente e o Gestor de Repositório

Pedido

<?xml version="1.0" ?>

<training>

<type>download</type>

<id>Identificação do utilizador***</id>***

<password>Password do Utilizador***</password>***

<fileName>Directoria para onde transferir os ficheiro***<fileName>***

</training>

Resposta

<?xml version="1.0" ?>

<ftp>

<link>Endereço do Repositório ***</link>***

<port>Porta de comunicação ***</port>***

<serverType>scp</serverType>

<type>download</type>

<username>*Username* de acesso ao repositório***</username>***

<pwd> *Password* de acesso ao repositório ***</pwd>***

<path>Directoria de onde transferir os ficheiros do repositório***</path>***

<files>

<filename>Nome do primeiro ficheiro a transferir do repositório

</filename>

<filename>Nome do segundo ficheiro a transferir do repositório

</filename>

(...)

</files>

<originalName>Directoria para onde transferir os ficheiro no utilizador***</originalName>***

</ftp>

ANEXO 3 – Pedido/Resposta XML de Upload Entre o Cliente e o Gestor de Repositório

Pedido	***<?xml version="1.0" ?>*** ***<training>*** ***<type>upload</type>*** ***<id>***Identificação do utilizador***</id>*** ***<password>***Password do Utilizador***</password>*** ***<fileName>***Directoria do ficheiro *a transferir* ***</training>***
Resposta	***<?xml version="1.0" ?>*** ***<ftp>*** ***<link>***Endereço do Repositório ***</link>*** ***<port>***Porta de comunicação ***</port>*** ***<serverType>scp</serverType>*** ***<type>upload</type>*** ***<username>****Username* de acesso ao repositório***</username>*** ***<pwd>*** *Password* de acesso ao repositório ***</pwd>*** ***<path>***Directoria para onde transferir o ficheiro***</path>*** ***<filename>***Nome do ficheiro a transferir***</filename>*** ***<endFilename>***Nome do ficheiro no repositório</***endFilename>*** ***<originalName>***Directoria do ficheiro a transferir no utilizador***</originalName>*** ***</ftp>***