Isaac Caldas Ferreira

Desenvolvimento de um Sistema Anti-Spam de Código Aberto

Itajubá - MG
Junho - 2018

Isaac Caldas Ferreira

Desenvolvimento de um Sistema Anti-Spam de Código Aberto

Universidade Federal de Itajubá – UNIFEI
Instituto de Engenharia de Sistemas e Tecnologias da Informação - IESTI
Programa de Pós-Graduação em Ciência e Tecnologia da Computação - POSCOMP

Orientador: Prof. Dr. Otávio Augusto Salgado Carpinteiro
Coorientador: Prof. Dr. Edmilson Marmo Moreira

Itajubá - MG
Junho - 2018

"E quem amafagafar os mafagafinhos, bom amafagafigador será".

M.A.

AGRADECIMENTOS

Deixo neste capítulo frases, à vocês que ajudaram no desenvolvimento deste trabalho, para mostrar meu agradecimento; mas saibam que ele é muito maior do que eu consegui expressar.

Em momentos difíceis onde nenhuma linha de código parece resolver o *Segmentation Fault*, tive a luz do Espírito Santo para me guiar. Senhor, Meu Deus, obrigado por sempre estar presente em minha vida, por sempre me ajudar a enxergar o caminho e ter paciência – mesmo quando minha vontade é de arremessar meu computador do mais alto prédio.

O apoio que sempre pude contar com, em toda a minha vida. Este que vem sempre de minha mãe e de meu pai. Os quais sempre tiveram as palavras certas para me apoiar, como "calma, logo você acha a solução" e "pare de reclamar e finalize logo seu projeto!". Pai e mãe, muito obrigado.

Meu obrigado aos meus irmãos e namorada que sempre me apoiaram com paciência e café, com distrações necessárias e excelentes campanhas em jogos de vídeo-game. Aos três, meu obrigado.

Sem a mão, e o braço também, do pessoal da Diretoria de Tecnologia da Informação da Unifei este trabalho não seria possível. Agradeço a vocês por disponibilizarem suas horas, dias e meses para me ajudar. Obrigado Pablo, Everaldo, José Renato e todos os outros.

Um grande obrigado ao professor Edmilson Moreira que, com uma paciência absurda, corrigiu, linha a linha, um trabalho de Iniciação Científica, o qual me abriu a cabeça para melhorar minha escrita, em especial a escrita deste.

Meus agradecimentos ao professor Bruno Kuhene por seu apoio fora do horário do expediente, e a cafeteira disponibilizada ao grupo de pesquisa GPESC – sério, café é importante demais.

Não posso deixar de agradecer ao meu "colega de projeto" Marcelo Aragão, que com seu projeto de mestrado, o qual é vinculado a este, apresentou resultados incríveis – vejam seu trabalho "Estudo e Pesquisa em Sistemas Anti-Spam" (ARAGãO, 2018), não darei *spoilers* aqui.

Agraço também aos meus colegas e amigos de república e do grupo GPESC e a todos os outros que não citei, porém que direta ou indiretamente contribuíram para o desenvolvimento deste trabalho – ou com café.

Necessito, também, agradecer ao pastor etíope que descobriu o café.

Por último, deixo meu agradecimento especial àquele que tornou este trabalho possível. Ao meu querido professor e orientador Otávio A. S. Carpinteiro, que logo no início de minha graduação me acolheu como orientado e como um pai me orientou, em nossos projetos e para a vida, durante todos estes anos até este momento, em meu mestrado. Muito obrigado por esta honra. – meu caro leitor, se esta dissertação é legível, e com boa gramática, agradeça ao professor, pois em 2012, ao ler meu primeiro relatório de Iniciação Científica, ele me apontou o caminho, dizendo "seu português é terrível!".

RESUMO

Neste trabalho, apresentamos o desenvolvimento de um Sistema Anti-Spam de Código Aberto (SASCA), em Java. Ao contrário de sistemas *anti-spam* comerciais, o SASCA não faz uso de listas de bloqueio (negras/brancas) e sim de modelos de *machine learning* para classificação de *e-mails*.
Foram realizados diversos experimentos sobre uma base de *e-mails* reais, coletados na Universidade Federal de Itajubá.
Nos experimentos realizados, observou-se que o SASCA obteve desempenho bem próximo ao do sistema *anti-spam* comercial CanIt, em termos de classificação de *e-mails*, mas com desempenho bem melhor, em termos de tempo requerido para classificação.

Palavras-chave: *e-mail*, *spam*, *anti-spam*, modelos de *machine learning*, *parser*, *crawler*.

ABSTRACT

In this work, we present the development of an Open Source Anti-Spam System (SASCA) in Java. Unlike commercial anti-spam systems, SASCA does not make use of block lists (black / white), but rather of m machine learning models for email classification.
Several experiments were carried out on a basis of real e-mails collected at the Federal University of Itajubá.
In the experiments carried out, it was observed that the SASCA performed very close to the CanIt commercial anti-spam system in terms of e-mail classification, but with a much better performance in terms of the time required for classification.

Keywords: e-mail, spam, anti-spam, machine learning models, parser, crawler.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

1 INTRODUÇÃO

Em 1971, o programador Ray Tomlinson desenvolveu uma aplicação que, através da rede de computadores ARPANET — rede que deu origem à Internet —, era capaz de fazer o envio e recebimento de mensagens simples (KARASINSKI, 2009) entre computadores não conectados diretamente entre si. Estas mensagens receberam o nome de *e-mail*, que significa *electronic mail*, ou seja, correio eletrônico.

Conforme o interesse nesta tecnologia aumentava, novas funções foram agregadas e seu protocolo de comunicação — regras que definem o envio e recebimento de mensagens — foi criado. Assim, o *e-mail* passou a ser visto não como uma simples aplicação, mas como, de fato, um correio eletrônico (CROCKER, 2000), tornando-se um serviço da Internet.

Com o crescimento e a popularização do correio eletrônico, diversos servidores de *e-mail* foram criados. Estes possuem a função de receber, manipular e retransmitir um *e-mail* para outro ou outros servidores de *e-mail* (LARRAMO, 2016). Um servidor contém clientes que se conectam a ele utilizando uma conta de *e-mail*, a qual possui um endereço vinculado similar ao de uma caixa postal de um correio *real*.

Os endereços de *e-mail* possuem o formato `nome_do_usuário@domínio`, onde *nome_do_usuário* pode ser qualquer combinação de caracteres alfanuméricos e de alguns símbolos, como o ponto e o sublinhado. O domínio é o servidor de *e-mails* e é descrito por qualquer combinação de caracteres alfanuméricos. Um exemplo de *e-mail* válido é: *joao22@empresa2000.com*.

Atualmente, é possível enviar e receber, através de *e-mail*, grandes mensagens de texto, imagens, vídeos ou qualquer outro tipo de arquivo para qualquer entidade no mundo, seja ela pessoa, empresa, ou instituição. No entanto, da mesma forma que com as cartas nas caixas de correios de nossas casas, *e-mails* podem conter propagandas comerciais, como catálogos de lojas e ofertas de cursos, por exemplo.

O primeiro registro que se tem do uso comercial de *e-mails* foi em 1978, quando Gary Thuerk, gerente de *marketing* da empresa Digital Equipment Corp., enviou para cerca de 400 clientes da ARPANET uma mensagem divulgando um novo

computador da empresa. Este foi o primeiro *e-mail* não-solicitado enviado, o qual, mais tarde, foi denominado *spam* (SMITH, 2007).

1.1 *SPAM*

A denominação do termo *spam* à *e-mails* não-solicitados se deve a uma referência à paródia feita pelo grupo de comédia Monty Python (HISKEY, 2010) em 1970. Nessa paródia, gravada em uma lanchonete, conforme o freguês verifica o cardápio, percebe que todos os itens do cardápio são feitos com SPAM — uma marca de presunto processado e enlatado — e SPAM não é considerado, pelo freguês, como "carne verdadeira". Além disso, durante o decorrer da paródia, a palavra SPAM é repetida cerca de 132 vezes.

Assim surgiu o termo *spam*, escrito em letras minúsculas, para representar *e-mails* que não são *verdadeiros*. Pelo mesmo princípio, *e-mails* válidos foram chamados de *ham* — palavra inglesa que significa "presunto" não-processado.

Em 1998, a palavra *spam* foi adicionada ao dicionário Oxford com o seguinte significado: *spam* são mensagens irrelevantes ou não-solicitadas enviadas pela Internet, geralmente para um grande número de usuários, para fins de publicidade, *phishing*, disseminação de *malware* etc (OXFORD, 2016). Indivíduos que criam e/ou divulgam *spam* são conhecidos pelo termo *spammer*.

Spam está diretamente associado a ataques à segurança da Internet e a do usuário, sendo um dos grandes responsáveis pela propagação de códigos maliciosos, disseminação de golpes e venda ilegal de produtos (CERT, 2012). A seguir, são apresentados alguns dos problemas causados por *spam* aos usuários.

- Perda de mensagens importantes.

  Devido ao grande volume de *spam* recebido, pode-se deixar de ler mensagens importantes, ou lê-las com atraso ou apagá-las por engano.

- Conteúdo impróprio ou ofensivo.

  Grande parte do *spam* é enviada para conjuntos aleatórios de endereços de *e-mail*, mirando um certo público-alvo, que pode ou não estar contido nestes conjuntos. Desta forma, é bastante provável o recebimento de mensagens cujo

conteúdo seja considerado impróprio ou ofensivo, como propagandas sobre medicamentos sexuais, divulgações políticas, ofertas de produtos etc.

- Desperdício de tempo útil.

  Para cada *spam* que se recebe em nas caixas de *e-mail*, é necessário despender um tempo para lê-lo, identificá-lo e removê-lo. Este processo acaba por resultar em desperdício de tempo e perda de produtividade.

- Não recebimento de *e-mails*.

  Há serviços de *e-mail* em que o espaço de armazenagem é limitado. Desta forma, há o risco de encher este espaço com *spams*. Assim, até que o espaço seja liberado, ou seja, *e-mails* sejam removidos, nenhuma nova mensagem será recebida.

- Prejuízos financeiros causados por fraude.

  Spam é amplamente utilizado como veículo para disseminar esquemas fraudulentos, que tentam induzir o usuário ou a acessar páginas clonadas de instituições financeiras ou a instalar programas maliciosos projetados para furtar dados pessoais e financeiros, como senha de conta bancária, por exemplo. Esse tipo de *spam* é conhecido como *phishing* (ou *scam*). O usuário pode sofrer grandes prejuízos financeiros, caso forneça as informações ou execute as instruções solicitadas neste tipo de mensagem.

Independentemente do tipo de acesso à Internet utilizado, é o destinatário do *spam* quem paga pelo envio da mensagem. Os provedores, na tentativa de minimizar os problemas, provisionam mais recursos computacionais e, assim, os custos derivados acabam sendo transferidos e incorporados à mensalidade dos usuários do serviço disponibilizado.

Alguns dos problemas relacionados a *spam*, que provedores, empresas e instituições costumam enfrentar, são listados a seguir.

- Impacto na rede de dados.

  O volume de tráfego gerado pelos *spams* faz com que seja necessário aumentar a capacidade de transferência de dados na Internet. Isto pode implicar em gastos com melhores provedores de Internet ou com infraestrutura.

- Má utilização dos servidores.

  Com os *spams*, surge a necessidade de se utilizar processamento nos servidores de *e-mail* para verificar as mensagens recebidas e tratá-las, caso sejam indesejadas.

- Investimento extra em recursos.

  Os problemas gerados pelos *spams* fazem com que seja necessário aumentar os investimentos para aquisição de equipamentos e de sistemas de filtragem e para contratação de mais técnicos especializados em sua operação.

- Inclusão em listas de bloqueio (*blacklists*).

  Um provedor que tenha usuários envolvidos em casos de envio de *spam* pode ter seu domínio incluído em listas de bloqueio. Isto pode prejudicar o envio de *e-mails* por parte dos demais usuários e resultar em perda de clientes.

De acordo com Schwandt e Kröger (2016), em setembro de 2016, *spam* correspondia a 61,25% do total de *e-mails* que trafegaram pela Internet. Isto representa cerca de 19,7 bilhões de *e-mails* (SORKIN, 2016). A figura 1 apresenta, em termos percentuais, a relação entre *e-mails spam* e o total de *e-mails* enviados nos períodos correspondentes.

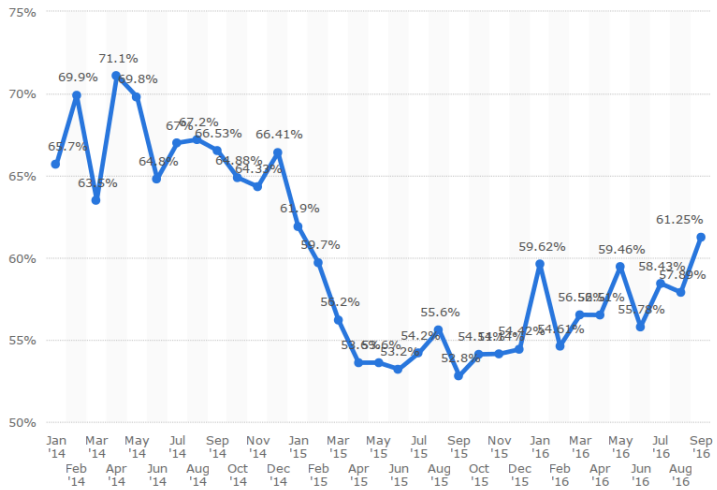

Figura 1 – Taxa de *e-mails spam* enviados entre Janeiro/2014 e Setembro/2016

Schwandt e Kröger (2016)

Devido à necessidade de se combater o *spam*, filtros *anti-spam* (também conhecidos como sistemas *anti-spam*) foram criados.

1.2 *ANTI-SPAM*

Em seus primórdios, a comunicação via *e-mail* era feita através de um protocolo ponto-a-ponto conforme a topologia mostrada na figura 2. Ou seja, para cada transmissão, era necessário conhecer o endereço da máquina destinatária. Devido a esta simplicidade, bloquear um *e-mail* consistia em apenas rejeitar o endereço da máquina do remetente. Esta técnica evoluiu para o que hoje se conhece como *blacklist*, discutida na seção 1.2.1 seguinte.

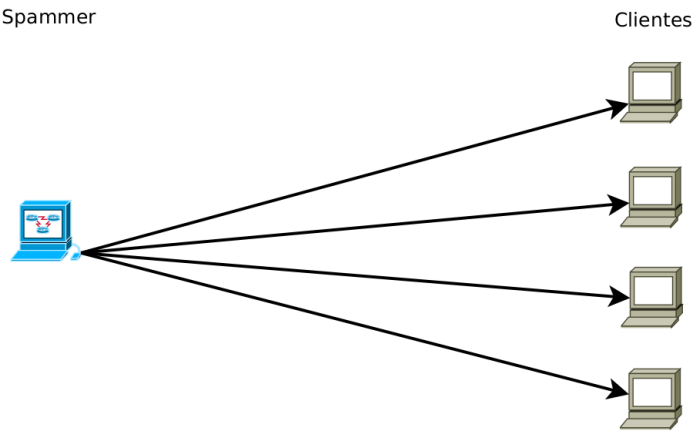

Figura 2 – Primeira topologia do *e-mail*

Sendo facilmente bloqueados, os *spammers* começaram a desenvolver os primeiros *BotNets*. *BotNets* são computadores infectados com um vírus específico que os deixam a mercê de um *spammer*. Isto possibilita aos *spammers* enviar *e-mails* através desta máquina infectada, ocultando a real origem do *spam*, conforme ilustrado na figura 3.

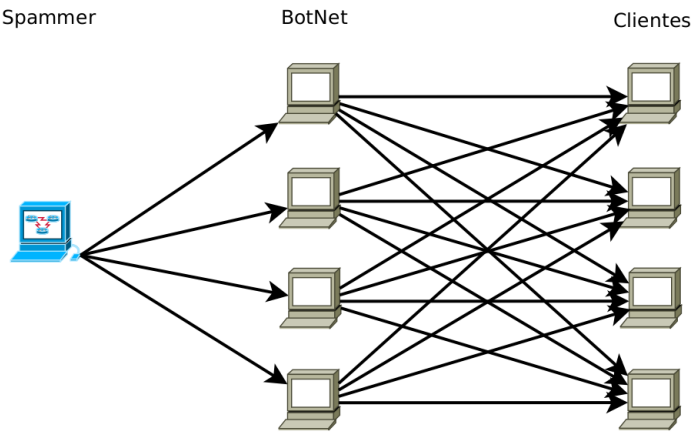

Figura 3 – Propagação de *spam* através de *BotNets*

Conforme visto na seção anterior, após sua criação, o *e-mail* tornou-se cada vez mais indispensável, e, consequentemente, seu uso cresceu exponencialmente. Da mesma forma, os *spams* começaram a se manifestar como *pragas* nas caixas de *e-mail* dos usuários.

Os primeiros *spams*, em geral, continham apenas publicidade, como por exemplo, uma divulgação sobre produtos de uma determinada empresa. Entretanto, com a evolução da computação, surgiram *spams* com conteúdos maliciosos, como por exemplo, vírus que podiam ter a função de extrair dados do usuário, como senhas bancárias, além de outras formas ofensivas, como a criação dos *BotNets* (LUTUS, 2006). Assim, não era mais possível bloquear um usuário, pois cada dia um novo *spam* chegava através de um novo endereço de *e-mail*.

Com necessidade de bloquear estes *e-mails*, preferencialmente antes que eles chegassem ao usuário, aplicações para análise de conteúdo destes, determinando se são *spam* ou não, foram criadas. Estas aplicações são conhecidas pelos nomes *anti-spam*, filtro *anti-spam* e sistema *anti-spam*.

Nas próximas subseções, descreve-se as técnicas mais comuns utilizadas nos filtros *anti-spam*. Na subseção 1.2.1, a técnica que utiliza listas para bloquear ou permitir o recebimento de *e-mail*. Na subseção 1.2.2, a filtragem com base no conteúdo do *e-mail*. Outras técnicas são descritas na subseção 1.2.3.

1.2.1 FILTROS BASEADOS EM LISTAS

Com o auxílio de listas que contêm endereços de *e-mails* e/ou domínios, o *anti-spam* consegue, através de uma análise do endereço do remetente do *e-mail*, determinar se este deve ser classificado como *spam* ou *ham* (SATTERFIELD, 2006). Em outras palavras, com base em listas, visa-se identificar *spams* e *spammers*, bem como *hams* e usuários confiáveis.

As listas podem ser criadas pelos próprios usuários do *e-mail*, pelos servidores e por grandes empresas, que as comercializam. A maior vantagem proporcionada pelas listas comerciais está na quantidade de usuários e domínios que bloqueiam ou liberam, pois são atualizadas diariamente. Seu maior problema, porém, é que algumas empresas obrigam o pagamento de resgate por parte dos usuários ou instituições para retirar seus nomes/domínios da lista. Desta forma, empresas honestas podem ser obrigadas a pagar para serem consideradas honestas. Igualmente, um *spammer* também pode pagar para ser considerado honesto.

Listas com endereços a serem bloqueadas são conhecidas como listas negras (*blacklists*). De forma análoga, listas brancas (*whitelists*) contêm endereços de usuários que foram autenticados como honestos.

Diferentemente das listas negras, que costumam ser comerciais, listas brancas tendem a ser implementadas diretamente nos servidores de *e-mail*, onde um ou mais administradores as gerenciam. Com estas, tem-se duas abordagens. A primeira é a *estrita*, onde apenas os *e-mails* dos usuários que estão na lista são aceitos. A segunda é a *auxiliar*, onde *e-mails* dos usuários que estão na lista são aceitos e os demais *e-mails*, que não estão na lista, podem ser aceitos ou não, dependendo da análise realizada por outros filtros.

Há, igualmente, a possibilidade de se criar uma lista cinza (*greylist*). Esta lista procura tirar proveito do fato de que muitos *spammers* enviam lotes de *spam* através de um domínio. Quando um *e-mail* é recebido de um remetente desconhecido, uma mensagem automática é enviada de volta contendo instruções para *liberá-lo*. Tais instruções são passíveis de serem realizadas apenas por seres humanos, como por exemplo, digitar um código que está embaralhado em uma imagem ou reenviar a mensagem. Caso as instruções sejam realizadas, o endereço do remetente é adicionado à lista branca, marcando-o como válido. Caso contrário, o endereço

permanece na lista cinza, pois está sob avaliação, por um período de tempo até ser descartado ou enviado para a lista negra.

1.2.2 FILTROS BASEADOS EM CONTEÚDO

Filtros baseados em conteúdo são aqueles que analisam o corpo do *e-mail* e procuram identificar características que indiquem tratar-se de *spam* (SATTERFIELD, 2006). Três filtros baseados em conteúdo se destacam, sendo apresentados a seguir.

- Filtro baseado em palavras.

  Este filtro consiste em bloquear qualquer *e-mail* que contenha alguma palavra de um grupo de palavras pré-determinadas.

  Seu uso se deve ao fato de que muitas mensagens de *spam* contêm termos não encontrados frequentemente em comunicações pessoais ou de negócios. Entretanto, se um *e-mail* válido contiver pelo menos uma dessas palavras, pode ser classificado incorretamente.

  Uma das grandes falhas deste tipo de filtro é sua dependência de constante atualização. Um *spammer*, ao descobrir quais palavras são bloqueadas, por exemplo, pode trocá-las por sinônimos ou errar suas grafias, de forma a ainda ser possível ao destinatário reconhecer seus significados.

- Filtro heurístico (baseado em regras).

  Filtros heurísticos atribuem uma pontuação a cada palavra encontrada no *e-mail*. Palavras que frequentemente são encontradas em *e-mails spam*, como por exemplo, “viagra” ou “rolex”, recebem uma pontuação maior. Mensagens com pontuação, igual ou superior a um valor determinado, são classificadas como *spam* e bloqueadas.

  Em geral, este filtro trabalha rápido, o que minimiza o tempo de entrega do *e-mail*. No entanto, encontrar o valor que determina se o *e-mail* é válido ou não pode ser trabalhoso, uma vez que configurado incorretamente pode levar a classificações incorretas.

  Da mesma forma que com o filtro por palavras, os *spammers* podem descobrir quais palavras devem evitar adicionar à mensagem.

- Filtro Bayesiano.

  O filtro Bayesiano é considerado o filtro baseado em conteúdo mais avançado, pois emprega cálculos de probabilidade para classificar *e-mails*.

  O usuário deve *treinar* o filtro, classificando manualmente mensagens *spam* ou *ham* que recebe. Após alguns treinos, o próprio filtro passa a selecionar características (palavras e/ou frases) destes *e-mails* e as adiciona ou a uma lista específica de *spam* ou a outra de *ham*.

  Para entender o comportamento deste filtro, tem-se o exemplo a seguir. Verifica-se que a palavra "valium" aparece 95 vezes na lista de *spam* e 5 vezes na lista de *ham*. Se um determinado *e-mail* contiver uma única palavra "valium", então haverá uma probabilidade de 95% deste *e-mail* ser *spam*. Esta probabilidade é alterada caso o *e-mail* contenha outras palavras. Por exemplo, se contiver uma outra palavra que apareça mais na lista de *ham*, a probabilidade dele ser *spam* diminui.

1.2.3 OUTROS FILTROS

- Filtro baseado em confirmação.

  Ao receber um *e-mail* de um remetente desconhecido, o filtro envia ao remetente um *e-mail* contendo um *desafio*. O desafio consiste em alguma tarefa que apenas um ser humano seria capaz de resolver, como visitar uma página específica e entrar com algum código ou resolver algum quebra-cabeças, pois os *spammers* normalmente contam com sistemas automatizados que não estão preparados para resolver estes desafios. Se o desafio não é realizado em um período de tempo, o *e-mail* é descartado.

  Embora possa prevenir *e-mails* indesejados, este filtro possui alguns problemas. Por exemplo, *e-mails* de *sites* de notícias são enviados através de mecanismos automáticos, que não irão responder ao desafio. Também podem ocorrer mensagens vindas de remetentes, que, embora desconhecidos do filtro, sejam urgentes. Isto levaria a um atraso para a leitura da mensagem ou, igualmente, se o remetente não estiver familiarizado com o filtro, poderá não responder o

desafio a tempo. Este filtro é utilizado geralmente em conjunto com a lista cinza.

- Filtro baseado em colaboração.

  Este filtro utiliza uma abordagem colaborativa, onde diversos usuários, baseados no conteúdo do *e-mail* que receberam, tanto podem denunciar um remetente como sendo *spammer* quanto podem legitimá-lo. O filtro acaba por funcionar como uma junção das listas branca e negra que são compartilhadas por uma comunidade de usuários. Assim, um remetente pode ser listado como transmissor de *spam*, o que causaria seu bloqueio por outros usuários, mas também pode ser listado como um usuário legítimo e, assim, ser liberado.

  Cada notificação é registrada em um banco de dados centralizado, o qual tem seu acesso disponibilizado aos integrantes da comunidade de usuários. A maior vantagem deste sistema é a possibilidade de acabar com uma *epidemia* de *spam* em questão de minutos. Entretanto, da mesma forma, um grupo de *spammers* poderia se unir e adicionar seus próprios *e-mails* como legítimos, o que causaria grandes problemas à comunidade.

- Filtro baseado no DNS.

  Domain name system (DNS) é um serviço de rede provido por servidores responsáveis por traduzir nomes de domínios, por exemplo "empresa2000", para seus correspondentes endereços (CIPOLI, 2016). O filtro procura validar o domínio do remetente do *e-mail*, realizando uma busca nas tabelas dos servidores DNS. Se o domínio for encontrado, o *e-mail* passa a ser considerado válido.

  O maior problema deste filtro é a facilidade que um *spammer* tem de registrar um domínio para si ou enviar *e-mails* através de outros computadores transformados em *BotNets*.

1.3 MÉTRICAS

Existem diversas soluções *anti-spam* comerciais e gratuitas disponíveis no mercado e em comunidades de *software*. Cada uma destas possui alguma vantagem e

desvantagem em relação às demais. Para a aquisição de um *anti-spam*, faz-se uso de métricas para analisar seu desempenho. Algumas métricas são citadas a seguir.

- Acurácia.

  É a relação entre a quantidade de *e-mails* classificados corretamente e incorretamente. Sob esta métrica, são também avaliadas as quantidades de falsos negativos — *spams* classificados como *hams* — e de falsos positivos — *hams* classificados como *spams* — produzidas pelo *anti-spam*. O falso positivo é sempre mais grave que o falso negativo.

- Custo computacional.

  Cada *e-mail* consome um determinado tempo para ser processado por um *anti-spam*. Quanto mais elevado for este tempo, mais investimentos em infraestrutura (*hardware*) serão necessários.

- Custo monetário.

  Quando se fala em *anti-spam* comercial, deve-se levar em conta que estes possuem um custo financeiro para quem o adquire. Este custo pode ser a licença de ativação e/ou uma licença de uso anual.

- Código auditável.

  Qualquer *anti-spam* tem acesso aos conteúdos dos *e-mails* que recebe, o que lhe franqueia acesso a informações que possam ser sigilosas. Se o código do *anti-spam* for auditável, porém, isto significa que os responsáveis pela manutenção dele podem averiguar se alguma informação confidencial está sendo indevidamente utilizada.

- Suporte técnico e manuais.

  Como todo bom sistema de *software*, um *anti-spam* precisa de manuais que orientem o administrador no processo de sua instalação, configuração e operação. Além disso, é desejável o acesso ao suporte técnico do fabricante do *anti-spam*.

1.4 PROPOSTA

Anti-spams comerciais possuem seu código fechado, desta forma não é possível saber o que acontece de fato com um *e-mail* que é processado por ele. A dependência de listas negras, tanto por *anti-spams* comerciais quanto por gratuitos, pode comprometer a confiabilidade deste e, também, aumentar seu o tempo necessário para processar *e-mails*. A proposta deste trabalho é um *anti-spam* absento destas situações.

Portanto, este trabalho visa o desenvolvimento de um Sistema Anti-Spam de Código Aberto, que possua desempenho igual ou superior ao de *anti-spam* comerciais, porém sem depender do uso de listas de bloqueio. Para fins de abreviação, este sistema será referenciado como SASCA.

O SASCA não faz uso de listas de bloqueio. Isto confere-lhe vantagens importantes. Primeira, permite-lhe ser gratuito. Segunda, permite-lhe evitar quaisquer fraudes que possam advir das empresas proprietárias destas listas (OZER, 2008; AARON, 2010; WONG, 2016). Terceira, permite-lhe poupar tempo de classificação por não acessar tais listas na Internet. Quarta, permite-lhe não depender da atualização necessária, constante e muito frequente do conteúdo de tais listas. Por fim, permite-lhe grande escalabilidade quando a Internet migrar para a versão 6 de seu protocolo IP (*Internet Protocol*).

A implementação do SASCA é composta por módulos. Destes, três são os principais para a classificação dos *e-mails*. O primeiro é o módulo Pré-Filtro, que manipula e transforma os *e-mails* com o objetivo de detectar técnicas *spam*. O segundo é o módulo Seleção de Características, que determina quais características são mais relevantes para a classificação dos *e-mails* como *spam* ou *ham*. O terceiro módulo — Classificação — é composto por um modelo de *machine learning* responsável por classificar *e-mails* em *spam* ou em *ham*.

O SASCA é um *anti-spam* para servidor. Desta forma, conta com um módulo MTA (*Mail Transfer Agent*), que lhe possibilita receber *e-mails* advindos da Internet.

O SASCA classifica cada *e-mail* em *ham* ou *spam*. Se classificado como *ham*, o *e-mail* é entregue ao seu destinatário, mas se classificado como *spam*, é retido. Para evitar situações de falso positivo que possam causar prejuízos aos usuários, o

SASCA encaminha, periodicamente, um relatório ao usuário contendo os *e-mails spam* que o usuário recebeu desde o último relatório. Assim, o usuário poderá recuperar *e-mails* classificados incorretamente como *spam*.

O SASCA foi implementado na linguagem Java, de forma a ser independente da plataforma computacional em que execute.

1.5 CONCLUSÃO

O capítulo apresentou uma introdução ao objeto de estudo — o *spam* — e à proposta para combatê-lo — o *anti-spam* SASCA. No próximo capítulo, apresenta-se o conhecimento técnico necessário para seu desenvolvimento.

2 REVISÃO TEÓRICA

Este capítulo aborda, de forma sucinta, a teoria usada no desenvolvimento do SASCA.

No serviço de *e-mails*, há a necessidade de existir um servidor SMTP ou, ao menos, um agente MTA (*Mail Transfer Agent*) responsável pela troca de mensagens entre servidores. No trabalho desenvolvido, usa-se o servidor Postfix, o qual será descrito na seção 2.1.

Para estabelecer uma comunicação com o Postfix, o SASCA também precisa implementar um serviço SMTP. Esta serviço é implementado por meio da biblioteca SubEthaSMTP, a qual será descrita na seção 2.2, juntamente com o protocolo SMTP.

Uma breve descrição sobre *charset* (conjunto de caracteres) será feita na seção 2.3 e servirá como base para a compreensão de certas características dos *e-mails*, apresentados na seção 2.4.

O SASCA foi implementado na linguagem de programação Java. Esta linguagem é descrita na seção 2.5. Na seção 2.6 tem-se a descrição da biblioteca JavaMail, a qual tem tanto a finalidade de manipular o *e-mail* e extrair todas as suas informações quanto a de prover meios convenientes para o acesso à estas informações.

O conteúdo do corpo de um *e-mail* pode vir formatado na linguagem HTML. Para processar este conteúdo formatado, utiliza-se a biblioteca jsoup, a qual tem sua descrição na seção 2.7.

2.1 POSTFIX

O Postfix é um servidor de *e-mails*. Ele inclui um agente de transferência de mensagens (MTA – *Mail Transfer Agent*), responsável pelo envio e recebimento das mensagens (*e-mails*) entre servidores e sistemas locais (DENT, 2003, p. 5). O Postfix envia tanto mensagens através da Internet para outros servidores — que também usam um servidor ou um MTA — quanto envia estas para as caixas postais

de seus destinatários. As caixas postais podem ser acessadas pelos usuários de forma direta ou através de um serviço POP/IMAP.

Na figura 4 tem-se um exemplo sobre o envio de um *e-mail* através da Internet, onde a usuária Heloise (*Sender PC*) envia um *e-mail* para o usuário Abelard (*Recipient PC*), que está em outro domínio.

No exemplo, quando Heloise envia o *e-mail*, ele é entregue ao servidor de *e-mail* Postfix de seu domínio. Este servidor Postfix irá descobrir, com o auxílio do servidor DNS de seu domínio, onde está localizado o servidor de *e-mail* do domínio do Abelard. Com esta informação, o *e-mail* é enviado. Ao receber o *e-mail*, o servidor Postfix do domínio do Abelard verifica quem é o destinatário (no caso, Abelard) e faz a entrega do *e-mail* na caixa postal do Abelard. Através do servidor POP/IMAP, Abelard acessa seu *e-mail*.

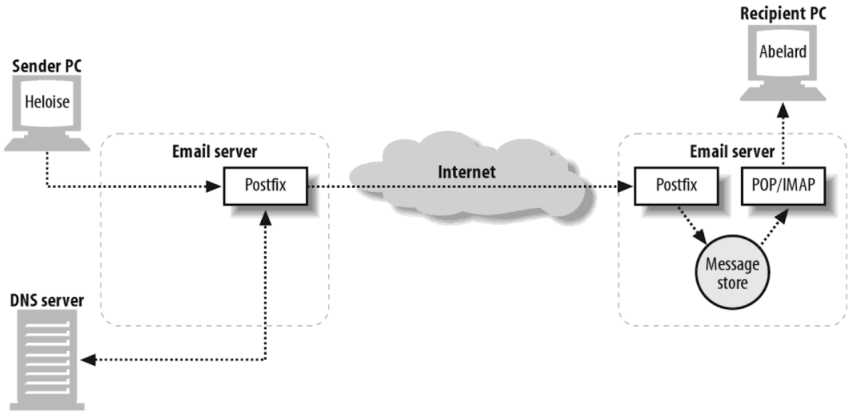

Figura 4 – Exemplo de um envio de *e-mail* através da Internet

Fonte: Dent (2003, p. 6)

Internamente, o Postfix funciona conforme ilustrado na figura 5. O *e-mail* é recebido da Internet pelo processo `smtpd`. Em seguida, é enviado para o processo `cleanup`, o qual, comunicando-se com o processo `trivial-rewrite`, verifica e, se possível, corrige problemas que encontra no cabeçalho do *e-mail*, tal como a formatação do endereço de *e-mail* utilizado. Finalmente, o *e-mail* é enviado para a fila de entrada `incoming` onde, conforme apresentado adiante, outros serviços o enviarão para o armazenamento local ou para o destinatário (VENEMA, 2016).

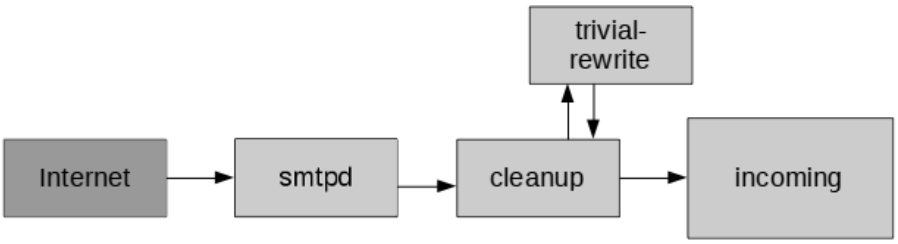

Figura 5 – Processos internos do Postfix

Descreve-se a seguir os processos internos do Postfix.

- `smtpd`. Este processo é responsável por conversar com outro servidor (ou MTA), através do protocolo SMTP, para receber ou enviar *e-mails*. Realiza também verificações e correções (se possível) na integridade destes.

- `cleanup`. Este processo, se possível, corrige problemas no cabeçalho do *e-mail* e padroniza-o. Por exemplo, adiciona o campo `From` ou outro qualquer que esteja ausente. Igualmente, converte endereços de usuários internos (e.g. `usuario@localhost`) para seu correspondente externo (e.g. `usuario@dominio2000.com`).

- `trivial-rewrite`. Este processo complementa o trabalho realizado pelo processo `cleanup`. Uma de suas funções é reescrever os endereços de *e-mail* para o padrão (*standard*). Por exemplo, `site!usuario` é reescrito para `usuario@site`.

- `incoming`. É a fila de entrada, onde ficam armazenados os *e-mails* que estão prontos para serem armazenados nas caixas postais dos usuários finais ou para serem enviados a outros servidores locais.

Quando o *e-mail* chega à fila de entrada, um outro processo, chamado de `qmgr` (*queue manager* ou gerenciador de filas), gerencia sua entrega a seu destino. O processo de entrega do *e-mail* é ilustrado na figura 6.

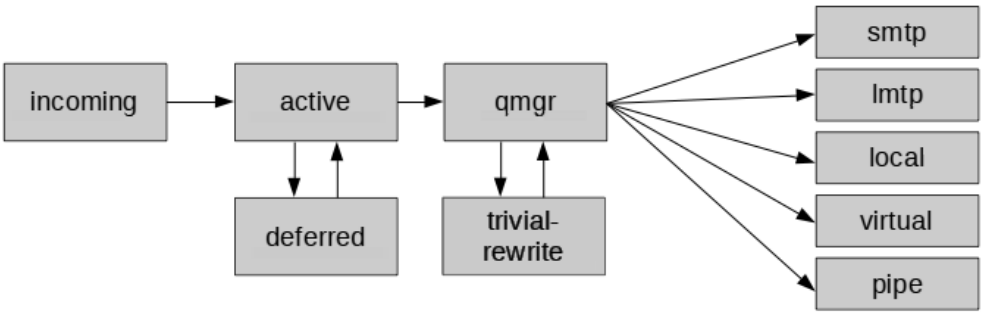

Figura 6 – Processo de entrega do *e-mail*

O processo `qmgr` executa concorrentemente com o processo `trivial-rewrite` que, neste momento, é utilizado para *resolver* os endereços de *e-mail*, isto é, determinar onde o usuário final está localizado. O `qmgr` é utilizado para o roteamento dos *e-mails* através do uso de uma tabela de transporte.

Uma vez determinado para onde o *e-mail* deve ser encaminhado, o processo `qmgr` o envia para um dos processos de transporte — `smtp`, `lmtp`, `local`, `virtual` ou `pipe` — que o encaminhará para seu destino. O processo `qmgr` manipula 3 filas, descritas a seguir.

- `incoming`. Conforme visto anteriormente, os *e-mails* chegam e são armazenados primeiramente nesta fila, antes de serem entregues.
- `deffered`. Esta fila contém os *e-mails* que estão à espera para serem entregues. Caso um ou mais destinatários do *e-mail* não estejam acessíveis em um determinado momento (seu servidor pode estar desligado, por exemplo), o *e-mail* fica armazenado nesta fila até uma nova tentativa de entrega.
- `active`. Esta fila contém os *e-mails* que o processo `qmgr` está tentando entregar. Ela requisita e recebe os *e-mails* das filas `incoming` e `deffered` e, de forma similar a um escalonador de processos, determina quais devem ser entregues.

Para que o Postfix envie os *e-mails* recebidos para o SASCA, precisa-se configurar alguns arquivos do primeiro. Os procedimentos necessários estão descritos na seção 2.1.1 seguinte.

2.1.1 INTERAÇÃO POSTFIX-SASCA VIA SUBETHA SMTP

Conforme apresentado na seção anterior, o servidor Postfix recebe um *e-mail* advindo da Internet, realiza determinados procedimentos internos e depois o envia ou para a caixa postal do destinatário ou para um outro servidor (ou MTA). Neste trabalho, o Postfix o entrega ao SASCA, via um agente MTA acoplado ao SASCA.

Ao acoplar um agente MTA ao SASCA permite-se que este possa estar em um computador diferente do que executa o Postfix. Para a implementação mais rápida do agente, utiliza-se a biblioteca SubEtha SMTP (STEVENS; SCHNITZER, 2015). Houve, também, o estudo de outras bibliotecas, como Apache JAMES e Jsmtpd, mas optou-se pela SubEtha SMTP por esta possuir código aberto, permitindo sua modificação, além da facilidade de uso (no anexo D.1, apresenta-se um exemplo para a criação de um servidor SMTP simples). Após criar o agente, é necessário configurar o Postfix para que este se comunique com aquele.

A figura 7 apresenta o diagrama do fluxo de um *e-mail* processado por algum filtro. Os filtros consistem em um *anti-spam*, como o SASCA, ou um anti-vírus, ou algum outro tipo de filtro.

Após receber um *e-mail*, o Postfix o *desvia* para o SASCA, que o processa. Ao concluir o processamento, se o *e-mail* for classificado como *ham*, será entregue a uma segunda instância do Postfix, denominada `smtpd2` no exemplo, que é encarregada de entregar o *e-mail* ao usuário — conforme será apresentado no capítulo 4. Entretanto, se for classificado como *spam*, o *e-mail* será retido pelo SASCA.

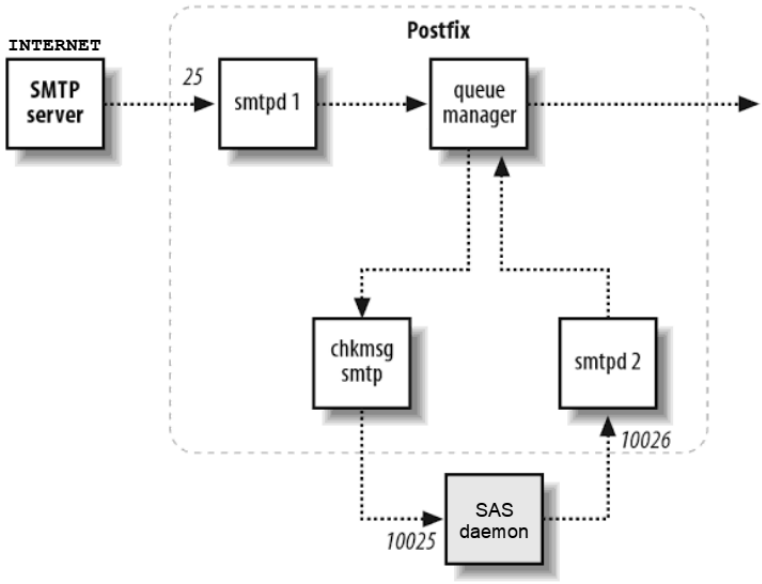

Figura 7 – Diagrama de filtragem de conteúdo

Fonte: Dent (2003, p. 178) – modificado

As configurações que devem ser realizadas no Postfix para que se comunique com o SubEtha SMTP estão descritas no anexo D.2.

2.2 SMTP

O *Simple Mail Transfer Protocol (SMTP)* é um protocolo desenvolvido para a troca de *e-mails* entre servidores (ou MTAs) (DENT, 2003, p. 15-17). Baseia-se em uma sequência de comandos e respostas inteligíveis tanto por computadores quanto por seres humanos. A figura 8 apresenta uma comunicação SMTP com o Postfix através do serviço `telnet`.

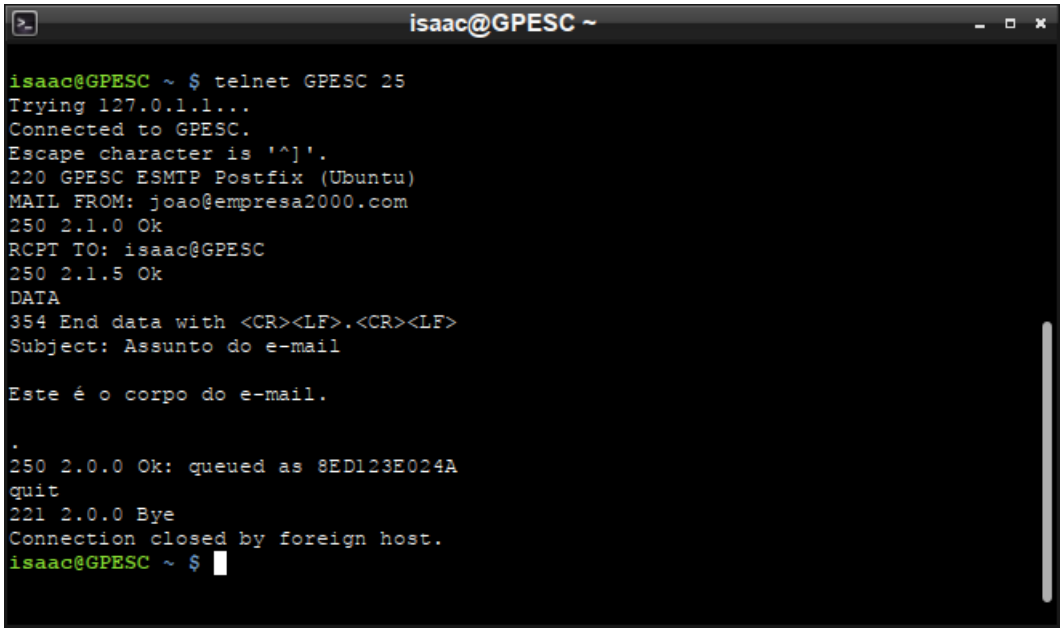

Figura 8 – Troca de mensagens via SMTP

Para a comunicação via `telnet`, precisa-se informar o endereço (ou o nome) do computador onde o servidor de *e-mails* executa bem como o número da porta onde o serviço é disponibilizado. Assim, no exemplo, o nome do computador é GPESC e a porta, 25. Após executar o `telnet`, tem-se a resposta a seguir.

```
Trying 127.0.1.1...
Connected to GPESC.
Escape character is `^]'.
220 GPESC ESMTP Postfix (Ubuntu)
```

Na primeira linha, observa-se que o `telnet` tenta conectar-se com o computador no endereço `127.0.0.1`, que é o endereço local. A segunda linha informa que a conexão foi realizada. A quarta linha apresenta a mensagem de "boas-vindas" do Postfix.

Para enviar um *e-mail* precisa-se definir o remetente e o destinatário. Para isto, usa-se os comandos `MAIL FROM: <e-mail>` e `RCPT TO: <e-mail>`, respectivamente. As linhas seguintes a estes comandos indicam se eles foram executados com sucesso ou não.

No exemplo a seguir, o *e-mail* tem como remetente o usuário `joao@empresa2000.com` e como destinatário o usuário `isaac@GPESC`. Este ultimo é o nome de um usuário da máquina GPESC, conforme podemos ver na primeira linha da figura 8.

```
MAIL FROM: joao@empresa2000.com
250 2.1.0 Ok
RCPT TO: isaac@GPESC
250 2.1.5 Ok
```

O conteúdo (corpo) do *e-mail* é descrito após o comando `DATA`. Algumas diretivas, como `Subject` (assunto), são utilizadas para especificar alguma parte do conteúdo. Ao finalizar a mensagem deve-se digitar um `enter` seguido de um `.` (ponto) e outro `enter`, conforme indicado pela resposta ao comando `DATA`.

A última linha notifica o sucesso do envio da mensagem bem como o identificador desta. Após enviar a mensagem, ela irá para a fila `incoming` do Postfix e, após passar por todas as etapas descritas na seção 2.1, será entregue ao destinatário.

```
DATA
354 End data with <CR><LF>.<CR><LF>
Subject: Assunto do e-mail
Este é o corpo do e-mail.
.
250 2.0.0 Ok: queued as 8ED123E024A
```

Para finalizar, envia-se o comando `QUIT`. Deste recebe-se uma mensagem de "Bye" e, em seguida, a comunicação é encerrada. O *e-mail* será salvo na caixa postal do usuário "isaac", em algum diretório configurado previamente. O usuário poderá acessar sua caixa postal diretamente ou através do serviço POP/IMAP de um cliente de *e-mail*. O *e-mail* enviado é apresentado a seguir.

```
From joao@empresa2000.com Thu Nov 10 17:37:21 2016
Return-Path: <joao@empresa2000.com>
X-Original-To: isaac@GPESC
Delivered-To: isaac@GPESC
```

```
Received: from localhost (localhost [127.0.0.1])
  by GPESC (Postfix) with SMTP id 8ED123E024A
  for <isaac@GPESC>; Thu, 10 Nov 2016 17:35:56 -0200 (BRST)
Subject: Assunto do e-mail
Message-Id: <20161110193609.8ED123E024A@GPESC>
Date: Thu, 10 Nov 2016 17:35:56 -0200 (BRST)
From: joao@empresa2000.com

Este é o corpo do e-mail.
```

2.2.1 GERÊNCIA DA PORTA 25

No exemplo apresentado na figura 8, foi utilizado a porta 25, porta padrão utilizada pelos servidores de *e-mail*. Entretanto, em uma tentativa de reduzir o *spam* no Brasil, diversas empresas estão gerenciando a porta 25 (HOEPERS; STEDING-JESSEN, 2009). A gerência tem o objetivo de diferenciar o envio de um *e-mail* do envio de mensagens entre servidores (KLENSIN; GELLENS, 2011).

Uma das ações implementadas por esta gerência é a configuração da aplicação cliente de *e-mail* para que utilize a porta 587, com autenticação. Transferindo o tráfego de *e-mails* do usuário final para a porta 587, pode-se bloquear o tráfego de saída da porta 25 quando um computador não é um MTA, excetuando-se o caso em que seja explicitamente autorizado. O principal objetivo desta ação é a redução do *spam* enviado por *botnets* (seção 1.2).

2.3 CHARSET

Os caracteres — letras, algarismos e símbolos — são representados no computador por um conjunto de *bits* pré-definido em uma tabela. Este conjunto pré-definido é conhecido como *charset*. A primeira tabela de codificação de caracteres (*chartset*) criada foi a tabela *American Standard Code for Information Interchange (ASCII)*. O anexo C apresenta a tabela ASCII.

Novos *charsets* foram criados com o intuito de incluir outros caracteres, como letras acentuadas e símbolos chineses. Em geral, embora nem sempre, esses novos *charsets* são extensões do ASCII. Por exemplo, o *charset* UTF-8 codifica seus primeiros 128 com a mesma codificação do ASCII. Na tabela ASCII estendida (*Extended* ASCII), porém, a letra `á` (a com acento agudo) é representado pelo *byte* 0xE1 enquanto que na tabela do UTF-8 é representado pelos dois *bytes* 0xC3A1.

Devido a incompatibilidade entre *charsets*, o cabeçalho dos *e-mails* inclui marcações que indicam qual *charset* deve ser utilizado para interpretar os caracteres seguintes. Estas marcações são apresentadas na seção 2.4.

2.4 E-MAIL

O SASCA classifica *e-mails* como *spam* ou *ham* com base na análise de seu conteúdo. Desta forma, para realizar a classificação, deve acessar o corpo de cada *e-mail* que recebe. O *e-mail* seguinte é usado como exemplo nesta seção.

```
1  From: "Joao Chefe" <joao@empresa2000.com>
2  To: "Joao Vintedois" <joao22@empresa2000.com>
3  Date: Sat, 21 Oct 2016 09:34:49 -0300
4  Subject: Um exemplo de e-mail
5  MIME-Version: 1.0
6  Content-Type: multipart/alternative;
7  boundary="----=_NextPart_DC7E1BB5_1105_4DB3_BAE3_2A6208EB099D"
8
9  ------=_NextPart_DC7E1BB5_1105_4DB3_BAE3_2A6208EB099D
10 Content-type: text/plain; charset=iso-8859-1
11 Content-Transfer-Encoding: quoted-printable
12
13 Esta parte =E9 o corpo do e-mail.
14
15 ------=_NextPart_DC7E1BB5_1105_4DB3_BAE3_2A6208EB099D
16 Content-type: text/html; charset=iso-8859-1
17 Content-Transfer-Encoding: quoted-printable
```

```
18
19 <html>
20   <body>
21     <div style=3D"FONT-SIZE: 10pt; FONT-FAMILY: Arial">Esta parte
          =E9 o corpo do e-mail, por=E9m em HTML.</div>
22   </body>
23 </html>
24
25 ------=_NextPart_DC7E1BB5_1105_4DB3_BAE3_2A6208EB099D--
```

Este exemplo é simples. Normalmente, o conteúdo de um *e-mail* é maior e contém mais informações, como registros dos nomes e endereços de servidores pelos quais o *e-mail* passou até alcançar seu destino. Cada atributo do cabeçalho do *e-mail*, o qual está compreendido entre as linhas 1 e 7, é descrito a seguir.

- `From` (linha 1). Este atributo indica, entre os símbolos "<" e ">", o endereço de quem escreve e envia o *e-mail*. Além do endereço, há a opção de informar um nome qualquer, o qual é escrito entre aspas.
- `To` (linha 2). Este atributo informa o endereço do destinatário do *e-mail*. Seu formato é igual ao do atributo `From`.
- `Date` (linha 3). Informa a data e o horário em que foi enviado.
- `Subject` (linha 4). Descreve o assunto do *e-mail*.
- `MIME-Version` (linha 5). Este atributo informa a versão do cabeçalho. Versões diferentes permitem ou não, por exemplo, o uso de acentuação e de anexos no *e-mail*. *Multipurpose Internet Mail Extensions (MIME)* significa, literalmente, Extensões Multipropósitos para Mensagens de Internet.
- `Content-Type` (linha 6). Este atributo especifica o tipo do conteúdo do corpo do *e-mail* (FREED, 2016). No exemplo, o corpo do *e-mail* possui o tipo `multipart`, ou seja, é composto por uma ou mais partes.
- `boundary`. Como o corpo do *e-mail* é composto por múltiplas partes, deve-se especificar, no cabeçalho, onde começa e termina cada parte. No caso, a *string*

"`----=_NextPart_DC7E1BB5_1105_4DB3_BAE3_2A6208EB099D`" é usada para delimitar cada parte do corpo do *e-mail*.

O corpo do *e-mail* vem logo após o cabeçalho. As linhas 9 e 15 delimitam a primeira parte, composta por um conteúdo de texto puro, conforme indicado, na linha 10, pelo atributo `Content-type` com valor `text/plain`. Esta linha também informa que o *charset* utilizado é o `iso-8859-1`.

A linha 11 informa como os caracteres especiais, os acentuados, por exemplo, estão codificados. No caso, estão codificados em ASCII (quoted-printable), o que significa que seu valor hexadecimal irá aparecer após um símbolo de igual (=); conforme pode-se ver na linha 13, onde a letra `é` está codificada como `=E9`. Este valor corresponde à letra `é` na codificação `iso-8859-1`.

Entre as linhas 15 e 25, encontra-se a segunda parte do corpo do *e-mail*. Seu conteúdo é do tipo HTML (`text/html`).

As linhas 8, 12, 14, 18 e 24 estão em branco e são necessárias, pois separam um cabeçalho de outro cabeçalho ou do corpo.

2.5 JAVA

A linguagem de programação Java é multiplataforma, isto é, permite a execução de aplicativos Java em quaisquer plataformas que implementem a *Java Virtual Machine (JVM)* (STÄRK; SCHMID; BÖRGER, 2012). O desempenho de aplicativos em linguagem Java é próximo ou igual ao de aplicativos em linguagem C Gherardi, Brugali e Comotti (2012). O SASCA foi implementado em Java devido ao desempenho desta linguagem bem como às facilidades proporcionadas por suas bibliotecas, que são referência na área.

2.6 JAVAMAIL

O corpo do *e-mail* pode ser composto por texto (`text/plain`) e/ou texto sob o formato HTML (`text/html`). A biblioteca JavaMail é utilizada pelo SASCA para extrair o conteúdo do *e-mail*. JavaMail é disponibilizada pela mantenedora do Java,

a empresa Oracle. Sua utilização foi devida a ela ser uma biblioteca referência na área de *e-mails*.

Internamente, a JavaMail possui classes utilizadas para manipular o cabeçalho, extraindo informações como remetente, destinatário e assunto, bem como para manipular o corpo do *e-mail* (FLATSCHART, 2013, p. 1). Quando um *e-mail* é processado pela JavaMail, um objeto da classe `Message` é criado. Esta classe possui uma referência a um objeto que contém o corpo do *e-mail*.

Como mencionado anteriormente na seção 2.4, o corpo pode ser `Multipart` ou apenas `Part`, uma vez que a classe `Multipart` é filha de `Part`. O objeto `Multipart` contém o corpo em si, ou seja, ou texto (e/ou texto sob formato HTML), ou um arquivo anexo, ou ainda, referências a outras partes. Quando o objeto é da classe `Part`, contém apenas conteúdo textual (texto ou texto sob formato HTML) (FLATSCHART, 2013, p. 19-23). O diagrama da figura 9 ilustra estas classes.

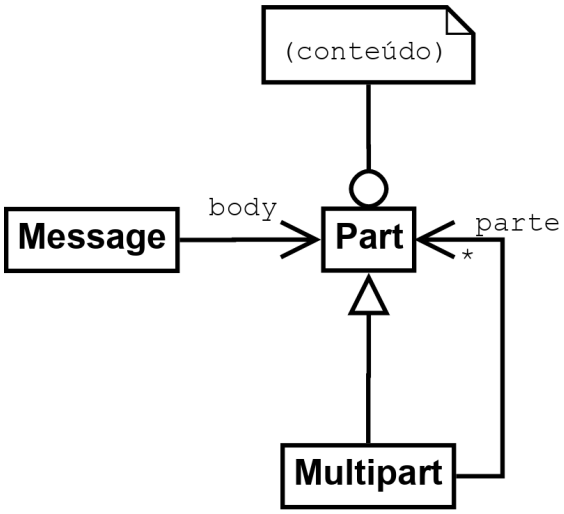

Figura 9 – Diagrama simplificado da JavaMail

O método `isMimeType(String)`, disponível na classe `Part`, é utilizado para identificar o tipo do corpo. Se for `Multipart`, *perguntamos* para o método se o tipo é "`multipart/*`". O mesmo procedimento é feito para identificar texto ou texto sob formato HTML, *perguntando* se o tipo é "`text/plain`" ou "`text/html`",

respectivamente. Se o tipo for processável, utiliza-se o método `getContent` para obter seu conteúdo no formato `String`.

2.7 JSOUP

Com o objetivo de aumentar as possibilidades de informação permitidas no corpo do *e-mail*, foi criada uma extensão do campo MIME para permitir que o corpo do *e-mail* seja em formato HTML. Assim, uma página de Internet, com *links* ou multimídia, pode ser enviada como um *e-mail*.

Uma página descrita na linguagem HTML é composta por *tags*, que possuem o formato `<tag>`, onde `tag` é o nome da *tag*. Determinadas *tags* podem conter sub-*tags*, diferentes ou iguais à *tag*-pai, formando uma hierarquia. Certas *tags* podem conter atributos, como tamanho de texto, cor etc.

A última versão (versão 5) do HTML disponibiliza 104 *tags*, conforme pode ser visto no anexo E. Para realizar o processamento das partes que contenham HTML, tem-se o uso da biblioteca `jsoup` (HOUSTON, 2013).

A `jsoup` foi desenvolvida para ser uma biblioteca para manipular páginas HTML encontradas no mundo real, ou seja, páginas mal-formatadas ou mal-estruturadas, contendo *tags* que não são fechadas explicitamente etc., posto que os navegadores de *web* também as conseguem interpretar e exibir corretamente.

A biblioteca provê uma interface adequada para extrair e manipular o conteúdo de páginas HTML. Por exemplo, possui métodos para percorrer as *tags* na ordem em que elas aparecem, para localizar determinados atributos dentro das *tags* e para limpeza e formatação do código (HEDLEY, 2016).

A escolha da `jsoup` para manipular o conteúdo HTML de *e-mails* deve-se à dificuldade e ao tempo necessário para desenvolvimento e implementação de um *parser* HTML. A biblioteca possui código aberto, documentado e tutoriais de uso. Foi facilmente integrada ao código do SASCA, conforme descrito no anexo F.

2.8 CONCLUSÃO

O capítulo apresentou o conhecimento técnico necessário para o desenvolvimento do *anti-spam* SASCA. O desenvolvimento é apresentado no capítulo 4. No próximo capítulo, apresenta-se a revisão bibliográfica.

3 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta, de forma sucinta, as principais referências bibliográficas utilizadas no desenvolvimento deste trabalho.

Crocker (2000) apresenta a história do *e-mail*, descrevendo o envio das primeiras mensagens, a criação dos primeiros protocolos até o *e-mail* no formato em que se conhece hoje. Além de concentrar-se nos principais eventos ocorridos, concentra-se, igualmente, nas principais aplicações desenvolvidas.

Wong (2016) questiona o uso das listas negras. Além de exigirem constante atualização, podem bloquear indevidamente domínios legítimos. Estende sua preocupação ao fato de que determinadas empresas, que mantêm listas negras, exigem valores monetários para a remoção de domínios legítimos de suas listas.

Slettnes et al. (2004) diz que é vantajoso filtrar o *e-mail* durante a transação SMTP, em vez de seguir a abordagem convencional de deixar esta tarefa para um sistema *anti-spam*. Lista algumas vantagens. Por exemplo, a possibilidade de interromper a entrega da maioria dos *spams* no início da transação SMTP, antes que o corpo da mensagem seja recebido, economizando-se largura de banda e processamento de CPU. Outras ventagens são a possibilidade de implementar algumas técnicas de filtragem de *spam* que não são possíveis em outro momento futuro como, por exemplo, atrasos na transação SMTP, lista negra e lista cinza (HARRIS, 2003), e a possibilidade de notificar o remetente em caso de falha na entrega (e.g., devido a um endereço de destinatário inválido) sem gerar, diretamente, *spam* colateral.

SpamAssassin (APACHE, 2018) é um *anti-spam* da Apache Foundation. Pode ser integrado a servidores de *e-mail* ou a clientes para filtrar *spam*. Usa um grande conjunto de regras para determinar se o *e-mail* é *spam* ou não. A maioria das regras é baseada em expressões regulares, que são procuradas dentro do conteúdo do *e-mail* e/ou em seu cabeçalho. O SpamAssassin inclui várias técnicas de detecção de *spam*, como filtragem Bayesiana, listas negras baseadas em DNS, listas negras baseadas em URI, listas de permissões baseadas em DNS, *Sender Policy Framework*, dentre outras.

ASSP *Anti Spam SMTP Proxy* (ECKARDT, 2018) é um *anti-spam* que executa como um servidor *proxy*. Possui código aberto e é escrito em Perl. Inclui várias técnicas de detecção de *spam*, como filtragem Bayesiana, validação de mensagem HELO (ou EHLO), lista de permissões, lista negra (e.g., DNSBLs, URIBLs), lista cinza e política de remetente SPF (*Sender Policy Framework*). No servidor ASSP, cada usuário de *e-mail* pode ter sua lista de permissões e lista negra privada, dependendo da configuração definida pelo administrador do ASSP. Os endereços de destino dos *e-mails* enviados pelos usuários são incluídos automaticamente em suas listas de permissões. O principal público-alvo do servidor ASSP são administradores de *e-mail* ou administradores de sistema em instituições relativamente pequenas. Segundo os desenvolvedores, o ASSP deve funcionar bem com menos de 300 endereços de usuários e um volume de mensagens de até cerca de 100.000 ((PATRIK, 2016)) mensagens por dia, embora testes não tenham sido feitos para verificar esses limites.

O qpsmtpd (HANSEN, 2018) é um *daemon* (processo que executa em *background*) SMTP escrito em Perl. Ele contém uma coleção de *plugins*, permitindo que os administradores de *e-mail* realizem a filtragem de *spam* de uma forma mais fácil. A coleção inclui validação de mensagem HELO (ou EHLO), listas negras de DNS e URL, lista cinza, SPF, filtros de *spam* (e.g., SpamAssassin) e antivírus (e.g., AVE, Bitdefender, ClamAV, dentre outros). O *daemon* qpsmtpd foi projetado para ser interposto à frente dos MTAs atuais (e.g., qmail, postfix ou exim). O *e-mail* é recebido pelo qpsmtpd e processado através dos *plugins* configurados, para ser avaliado e classificado como legítimo ou *spam*. Os *e-mails* são descartados ou passados para o MTA.

Barracuda Email Security Gateway (BESG) (BARRACUDA, 2018) é um sistema anti-spam comercializado pela Barracuda Networks. Inclui vários recursos, como bloqueio de *spam* e vírus, proteção de dados, continuidade de *e-mail*, prevenção de DoS *Denial of Service*, criptografia, verificação de vírus de remetente, proteção contra falsificação de remetente, dentre outros. O BESG "limpa" todos os *e-mails* antes de enviá-los ao servidor de *e-mail*, para protegê-lo contra ataques de *malware* e *phishing*. A Barracuda Networks mantém um centro de operações de ameaças que monitora constantemente a Internet para descobrir novas ameaças geradas por *e-mail*.

CanIt-Pro (SKOLL; WHITE, 2018) é um sistema *anti-spam* comercializado pela Roaring Penguin Software. Inclui vários recursos, como proteção contra *spam* e vírus, verificação de *e-mails* de entrada e saída, análise Bayesiana individual por usuário, listas de permissões, listas negras, quarentenas, dentre outros. O CanIt-Pro permite que os usuários finais gerenciem suas próprias configurações e opções. Podem gerenciar suas próprias regras para listas de permissões, listas negras, filtragem de conteúdo, DNSBLs, configurações Bayesianas e assim por diante. No entanto, os administradores podem restringir as regras que os usuários têm acesso. CanIt-Pro fornece relatórios e estatísticas sobre os *e-mails* que processa. Esses incluem estatísticas de correio diário e por hora, os principais vírus, os principais destinatários e assim por diante.

Diferentemente dos *anti-spams* comerciais, o SASCA classifica cada *e-mail* através da análise de seu conteúdo. Para isto, faz uso de modelos de *machine learning*. Portanto, não faz uso de listas negras ou brancas.

4 DESENVOLVIMENTO DO SASCA

O SASCA é um *anti-spam* que utiliza unicamente um modelo de *machine learning* para classificar *e-mails* em *spam* ou *ham*. Não faz uso de listas negras existentes na Internet. O SASCA possui dois modos de operação — Modo de Treinamento e Modo de Execução.

O Modo de Treinamento é usado para treinamento periódico de seu modelo de *machine learning*. Para o treinamento, utiliza-se um conjunto de *e-mails* coletados em um período de tempo, previamente classificados, em *spam* e *ham*, pelos próprios usuários do SASCA. No Modo de Execução, seu modo normal de operação, o SASCA classifica os novos *e-mails* que recebe do Postfix. Modos de Treinamento e Execução são sempre realizados *offline* e *online*, respectivamente.

A figura 10 ilustra a arquitetura modular do SASCA, através de um modelo simplificado de classes UML (*Unified Modeling Language*). Outrossim, apresenta, através de regiões, os módulos Pré-Filtro, Seleção de *Tokens*, Vetorização e Classificação — este último de forma muito simplificada. A figura ilustra o Modo de Treinamento do SASCA. Seu fluxo de execução é representado por um fluxo de setas, que se inicia na classe `Mail`, seguindo, por ordem, para o módulo Pré-Filtro, módulo Seleção de *Tokens*, módulo Vetorização e, por fim, para o módulo Classificação.

A figura 11 ilustra o Modo de Execução de forma similar à figura 10. Diversas classes dos módulos Pré-Filtro, Seleção de *Tokens* e Vetorização foram removidas da figura para torná-la mais clara. Seu fluxo de execução inicia-se na classe `Mail`, seguindo, por ordem, para o módulo SMTP, módulo Pré-Filtro, módulo Seleção de *Tokens*, módulo Vetorização, módulo Classificação e, por último, para o módulo Armazenamento. O módulo Notificação é executado independentemente.

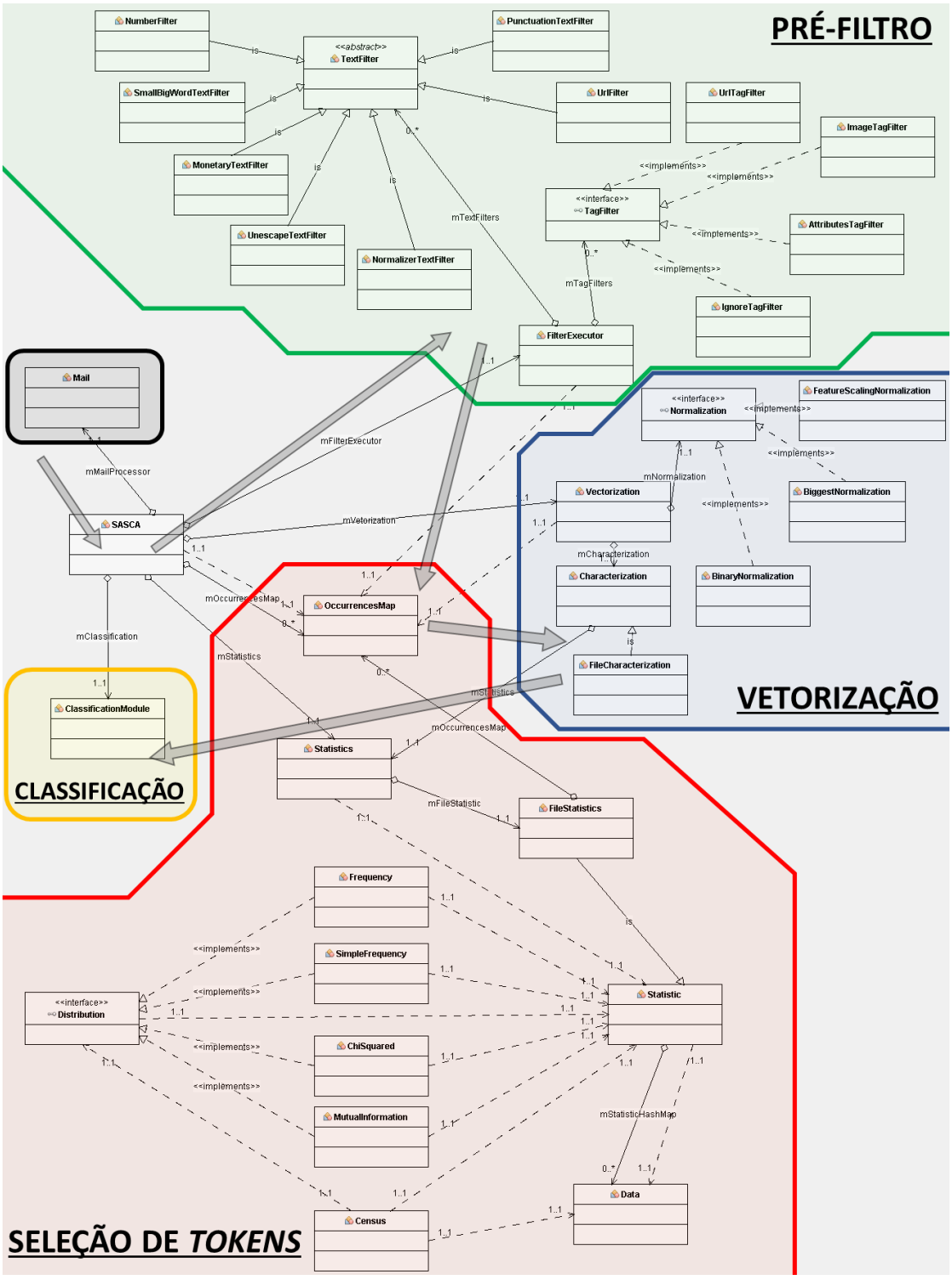

Figura 10 – UML *simplificado* do Sistema Anti-Spam em Modo de Treinamento.

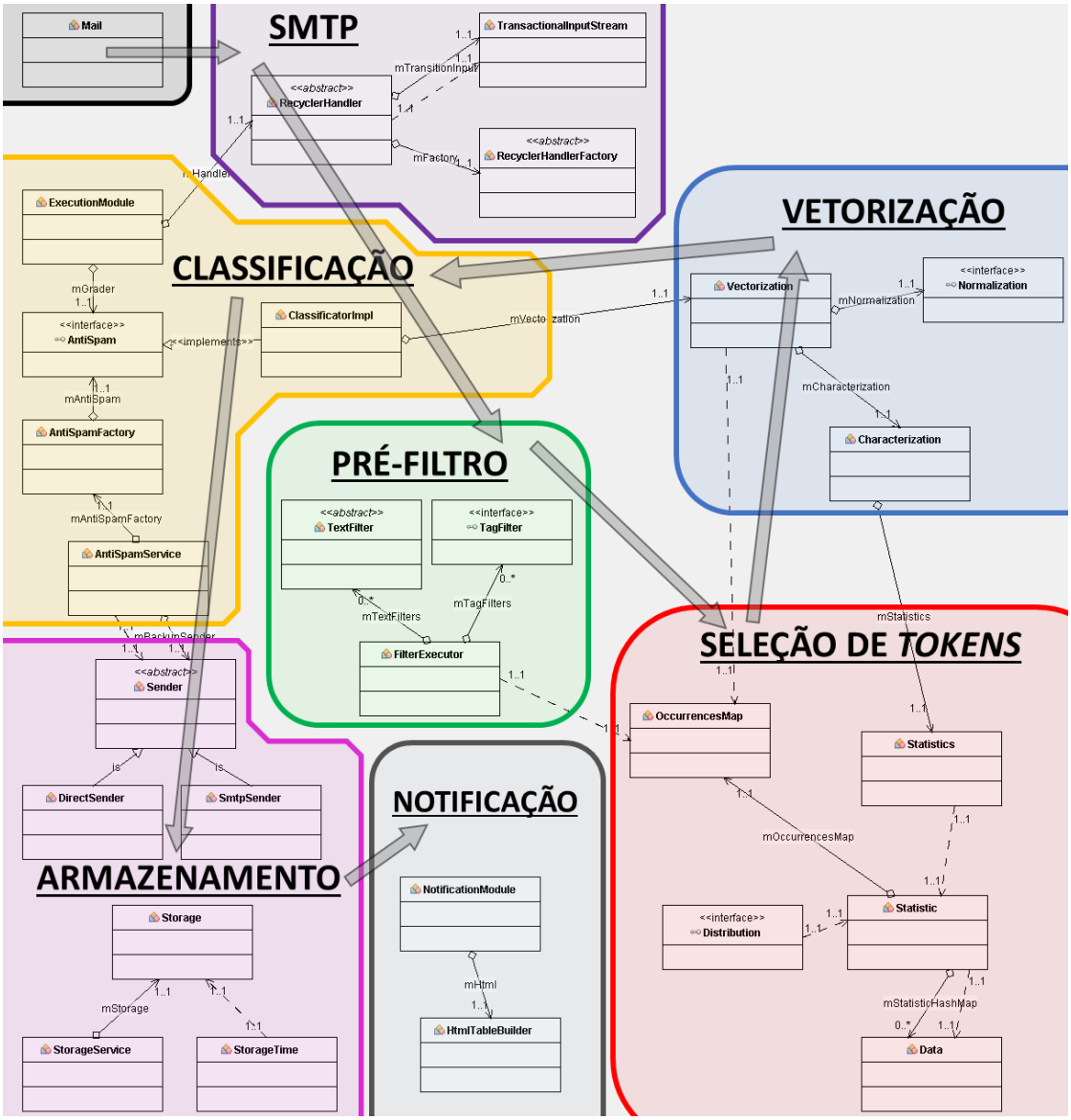

Figura 11 – UML *simplificado* do Sistema Anti-Spam em Modo de Execução.

As figuras 10 e 11 apresentam apenas a *essência* do SASCA. Em verdade, o SASCA possui muito mais classes do que as aqui apresentadas. Inclui, igualmente, ferramentas (apresentadas no capítulo 5) que não fazem parte de seu código principal.

4.1 MÓDULO SMTP

Conforme apresentado na subseção 2.1.1, o SASCA recebe *e-mails* da Internet através do `postfix`. Por sua vez, o `postfix` envia os *e-mails* que recebe ao SASCA, através do Módulo SMTP. Este módulo faz uso da biblioteca `SubEthaSMTP`. O anexo D.1 fornece mais detalhes da comunicação `postfix`-SASCA.

O SASCA também envia *e-mails ham* para o servidor `Zimbra`, que inclui outra instância do `postfix`. O `Zimbra` é responsável por entregar os *e-mails* recebidos nas caixas postais dos usuários.

A implementação do módulo procurou reduzir o tempo de processamento de duas formas. Primeiro, reaproveitando os *buffers* utilizados para armazenar *e-mails*. Segundo, usando `Threads` cuja quantidade pode ser configurada pelo administrador, definindo assim, o número máximo de *e-mails* que podem ser manipulados simultaneamente.

O módulo é configurável. Seus parâmetros, definidos no arquivo `sas.properties`, podem ser configurados pelo administrador. Os parâmetros são descritos a seguir.

- Porta pela qual os *e-mails* são recebidos.

  `POSTFIX_PORT = 1234`

- Endereço do servidor de armazenamento de *spam*. O servidor pode ser tanto o Módulo Armazenamento do SASCA (ver seção 4.6) quanto qualquer outro servidor SMTP (e.g. Zimbra).

  `STORAGE_SPAM_SERVER = localhost`

- Porta do servidor de armazenamento de *spam*.

  `STORAGE_SPAM_PORT = 5001`

- Diretório onde os *spams* serão armazenados.

  `STORAGE_SPAM_FOLDER = /diretório/para/armazenar/spams/`

- Modo de armazenamento dos *spams*. Pode assumir dois valores — `DIRECT` ou `SMTP`. `DIRECT`, para enviar os *spams* para um módulo do SASCA. `SMTP`, para enviar os *spams* para um servidor SMTP (ou MTA).

`STORAGE_SPAM_MODE = DIRECT OU SMTP`

- Parâmetros relativos ao armazenamento dos *e-mails* recebidos da Internet. Possuem uso análogo aos dos quatro anteriores.

  `STORAGE_BACKUP_SERVER`, `STORAGE_BACKUP_PORT`, `STORAGE_BACKUP_FOLDER` e `STORAGE_BACKUP_MODE`

4.2 MÓDULO PRÉ-FILTRO

Anti-Spams baseados em conteúdo, ou seja, aqueles que analisam o conteúdo dos *e-mails* para classificá-los, são frequentemente enganados por *spammers*, com o uso de técnicas que visam tornar legítimos seus *e-mails spam*. Assim, o módulo Pré-Filtro detecta, no corpo do *e-mail*, muitas das técnicas utilizadas pelos *spammers*, marcando-as, se necessário, com marcadores específicos, de forma a aumentar a probabilidade do *e-mail* ser classificado como *spam*.

Os marcadores podem ser adicionados ao *e-mail* segundo duas formas — por substituição e por adição — de acordo com a técnica empregada pelo *spammer*. Na primeira, a técnica encontrada é substituída por sua marca correspondente. Na segunda forma, o marcador é simplesmente adicionado quando a técnica é encontrada. As subseções 4.2.2 e 4.2.1 descrevem as marcações utilizadas.

O módulo Pré-Filtro também "limpa" os *e-mails*, removendo informações irrelevantes para a classificação, como cabeçalhos e outras partes não processáveis, como imagens. Processa somente texto em formato HTML ou texto simples. Caso o *e-mail* não contenha nenhum dos dois, uma mensagem de erro é gravada em um arquivo de registro (*log*).

O módulo Pré-Filtro manipula cada *e-mail* transformando-o em um conjunto de *tokens* (palavras e marcadores). O conjunto é salvo em um arquivo. O módulo foi implementado de forma a ser facilmente expansível, permitindo a inclusão de novos marcadores.

4.2.1 PROCESSAMENTO DE TEXTO EM FORMATO HTML

O módulo Pré-Filtro faz uso da biblioteca `jsoup` para processar texto em formato HTML. A `jsoup` extrai e manipula o código HTML do *e-mail*. Cada *tag* HTML é processada pelos *filtros HTML*. Cada filtro trata uma técnica empregada por *spammers*. Os filtros são ativados previamente, antes da inicialização do SASCA, em um arquivo de configuração, pelo administrador.

Os *filtros HTML* disponíveis no SASCA são descritos a seguir. O nome do filtro no código do SASCA encontra-se entre parênteses.

1. Detector de *tags* ignoradas (`IgnoreTagFilter`).

   O filtro recebe como parâmetro *tags* a serem ignoradas pelo módulo. Assim, quando a *tag* é encontrada, o filtro ignora-a e a substitui pelo marcador `!_ignore_tag`, onde `tag` é o nome da respectiva *tag*. Por exemplo, ao se passar ao filtro, como parâmetro, a *tag* "*script*", ela é ignorada, isto é, não é processada, e todo o conteúdo contido dentro de si é removido. Em seu lugar é inserido o marcador `!_ignore_script`.

2. Detector de *links* (`UrlTagFilter`).

   Um *link*, neste contexto, faz referência a uma página da Internet. *Links* podem estar dentro de *e-mails* legítimos. Entretanto, encontram-se em maior número em *e-mails spam*. *Spammers* usam *links* para desviar o leitor do *e-mail* para, por exemplo, uma página clonada, de forma a coletar suas informações pessoais ou a instalar *malwares* em seu computador.

   O filtro analisa todos os atributos de todas as *tags* em busca do atributo *href*, que é usado para referenciar uma página de Internet. Se encontrá-lo, a *tag* é substituída pelo marcador `!_URL`.

3. Detector de imagens (`ImageTagFilter`).

   O filtro detecta *tags* de imagem. Se for, substitui a *tag* e a imagem pelo marcador `!_IMAGE`.

4. Detector de atributos de *tags* (`AttributesTagFilter`).

Usuários legítimos importam-se, normalmente, com o conteúdo do *e-mail*. *Spammers*, porém, preocupam-se com o estilo do *e-mail*, fazendo uso dos atributos de *tags* HTML. Assim, o filtro extrai todos os atributos de cada *tag* encontrada e, para cada um destes, adiciona o marcador `!_in_atributo`, onde "atributo" refere-se a seu respectivo identificador. Por exemplo, se o filtro encontrar o atributo *style*, adiciona o marcador `!_in_style`.

4.2.2 PROCESSAMENTO DE TEXTO SIMPLES

O módulo Pré-Filtro processa texto simples em duas situações — ou quando o *e-mail* contém apenas texto simples ou logo após o processamento de texto em formato HTML. Os *filtros de texto* são similares aos *filtros HTML*.

Os *filtros de texto* processam os *tokens* segundo uma de três formas de operação — modificando-o, substituindo-o por um marcador ou mantendo-o inalterado. Os *filtros de texto* disponíveis no SASCA são descritos a seguir. O nome do filtro no código do SASCA encontra-se entre parênteses.

1. Valor monetário (`MonetaryTextFilter`).

   O filtro procura por padrões comuns em textos com teor financeiro, ou seja, procura pelos *tokens* `$` e `%`. Se algum destes for encontrado, o *token* é substituído pelo marcador `!_MONETARY`.

2. *Links* (`UrlFilter`).

   O filtro procura por *tokens* "http" e "www". Se encontrá-los, os substitui pelo marcador `!_URL`.

3. Números (`NumberFilter`).

   O filtro verifica se o *token* contém pelo menos um algarismo. Se contiver, substitui o *token* por `!_NUMBER`.

4. Símbolos (`PunctuationTextFilter`).

   O filtro remove símbolos encontrados. Os símbolos removidos são:

   ```
   ! " # $ % & ' * + , - . / : ; < = > ? @ [ ] ^ _ ` {  } ~ |
   ```

Por exemplo, o filtro substitui a palavra `vi@gra` por `vigra` e `dia!` (de "bom dia!") por `dia`.

O filtro deve ser executado após outros filtros que fazem uso destes símbolos para detectar algum padrão. Isto evita, por exemplo, a remoção do símbolo `$` antes que o valor monetário seja processado pelo filtro.

5. Palavras curtas e compridas (`SmallBigWordTextFilter`).

   Se a quantidade de caracteres de uma palavra for menor ou igual a um valor predeterminado, a palavra é substituída pelo marcador `!_SMALL_WORD`. Caso seja maior ou igual a um outro valor predeterminado, é substituída pelo marcador `!_BIG_WORD`. Por padrão, o SASCA considera palavras pequenas as que contêm 3 ou menos caracteres, e grandes as que contêm 20 ou mais caracteres.

6. Normalizador (`NormalizerTextFilter`).

   O filtro substitui cada carácter especial encontrado por seu equivalente não-especial minúsculo. Por exemplo, a letra `é` (`e` com acento agudo) é substituída pela letra `e`. Letras gregas, como a letra alfa, são removidas, bem como outros caracteres especiais que não contenham equivalentes *não-especiais*.

   Como exemplo, tem-se a frase "`T̂ĥïŝ ĩš â f̊ůňķŷ S̈ťŕĭńġ`" que será convertida para "This is a funky String".

7. Correção da codificação dos caracteres (`UnescapeTextFilter`).

   Conforme descrito na seção 2.4, o *e-mail* pode codificar caracteres específicos de outros *charsets* no *charset* ASCII. Por exemplo, a letra `ç` (`c` com cedilha) pode ser codificada como `&ccedil`. A biblioteca JavaMail converte estes caracteres para o *charset* `UTF-8` (padrão do Java). Alguns *e-mails*, porém, podem estar corrompidos, impedindo a codificação.

   O filtro corrige este problema, realizando a correção da codificação dos caracteres corrompidos.

4.3 MÓDULO SELEÇÃO DE *TOKENS*

Em uma abordagem integral, todos os *tokens* — palavras e marcadores — contidos em todos os *e-mails* seriam utilizados para representar cada *e-mail* como um vetor multidimensional em $\Re^n$. Com isto, porém, os *e-mails* seriam representados por vetores de dimensionalidade muito alta, gerando custos de armazenamento e processamento igualmente muito altos. Assim, por ser um *anti-spam* de servidor, o SASCA despenderia muito tempo para classificar cada *e-mail*, algo que deve ser evitado.

O módulo Seleção de *Tokens* tem por objetivo classificar os *tokens* encontrados no conjunto de *e-mails* por ordem de relevância. Assim, os *e-mails* podem ser representados por vetores de dimensionalidade muito mais baixa, onde cada dimensão representa um *token* relevante para a classificação.

O módulo faz uso de métodos estatísticos para classificar os *tokens* por ordem de relevância. Os métodos estatísticos são apresentados nas subseções a seguir.

4.3.1 DISTRIBUIÇÃO DE FREQUÊNCIA

O método distribuição de frequência (*Frequency Distribution* (FD)) leva em consideração a quantidade de vezes em que cada *token* aparece no conjunto de *e-mails*. O método FD é muito simples e rápido. Não pondera, porém, o grau de relevância dos *tokens* para as classes *ham* e *spam*. O método FD é bem descrito na literatura (YAMANE, 1973, p. 6).

4.3.2 INFORMAÇÃO MÚTUA

O método informação mútua (*Mutual Information* (MI)) mede o grau de relevância de cada *token* para as classes *ham* e *spam*. Para isso, faz uso da teoria de probabilidades. O método MI também é bem descrito na literatura (LATHAM; ROUDI, 2009).

4.3.3 QUI-QUADRADO

Tal como o método MI, o método qui-quadrado (*Chi-Squared* (CHI2)) também mede o grau de relevância de cada *token* para as classes *ham* e *spam*. Para isso, faz, igualmente, uso da teoria de probabilidades. O método CHI2 é bem descrito na literatura (YAMANE, 1973, p. 613).

4.4 MÓDULO VETORIZAÇÃO

Como mencionado na seção 4.3, cada *e-mail* é representado como um vetor multidimensional em $\Re^n$, onde cada dimensão representa um *token* encontrado no conjunto de *e-mails*. O módulo Vetorização é, portanto, responsável por converter cada *e-mail* em sua representação vetorial.

A dimensionalidade `n`, ou seja, os `n` *tokens* mais relevantes, selecionados pelo módulo Seleção de *Tokens*, é definida pelo administrador do SASCA. Para ilustrar a operação do módulo Vetorização tem-se o exemplo a seguir.

Supondo que, após processar um conjunto de *e-mails*, o módulo Seleção de *Tokens* selecionou os *tokens* mais relevantes. Supõe-se, igualmente, que o administrador tenha definido a dimensionalidade `n` como sendo igual a cinco, e que os cinco *tokens* selecionados, por ordem de relevância, sejam `emprestimo`, `!_SMALL_WORD`, `!_MONETARY`, `pilula` e `!_BIG_WORD`. O *e-mail* a ser representado é mostrado a seguir.

> Os ovos de páscoa custam R$3,50. Quem se interessar, ligue para 98765-4321.

Antes da representação ser gerada pelo módulo Vetorização, o *e-mail* é processado pelo módulo Pré-Filtro, gerando o *e-mail* a seguir.

> `!_SMALL_WORD` ovos `!_SMALL_WORD` pascoa custam `!_MONETARY` quem `!_SMALL_WORD` interessar ligue para `!_NUMBER`

A figura 12 apresenta a representação do *e-mail*, gerada pelo módulo Vetorização.

Figura 12 – Representação gráfica de um vetor.

O vetor deve ter seus valores normalizados. Para tal, o módulo implementa três algoritmos de normalização, descritos a seguir. O administrador pode escolher qual usar.

- Norma binária.

  Esta norma, conforme o nome sugere, tem por objetivo tornar binários os valores do vetor, marcando quais *tokens* do vetor existem ou não no *e-mail*. Portanto, se um valor do vetor for maior ou igual a 1, é substituído por 1. Caso contrário, será mantido em zero.

  A figura 13 apresenta a normalização, pela norma binária, do vetor da figura 12.

Figura 13 – Vetor normalizado pela norma binária.

- Norma da média.

  Os valores do vetor normalizados pela norma da média são dados pela equação 4.1

$$valor = \frac{valor - V_{\min}}{V_{\max} - V_{\min}} \tag{4.1}$$

  onde V_{min} e $V_{máx}$ são, respectivamente, o menor e o maior valor do vetor.

  A figura 14 apresenta a normalização, pela norma da média, do vetor da figura 12.

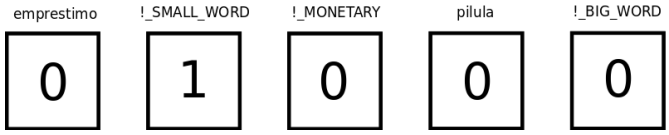

Figura 14 – Vetor exemplo normalizado pela norma da média.

- Norma do maior.

  Os valores do vetor normalizados pela norma do maior são dados pela equação 4.2

$$valor = \frac{valor}{V_{\max}} \tag{4.2}$$

  onde V_{max} é o maior valor do vetor.

  A figura 15 apresenta a normalização, pela norma do maior, do vetor da figura 12.

Figura 15 – Vetor exemplo normalizado pela norma do maior.

Em Modo de Treinamento, os vetores gerados são salvos em dois arquivos — o primeiro contendo *hams* e o segundo *spams*. Estes são utilizados no treinamento do

modelo classificador. Em Modo de Execução, o vetor gerado é enviado diretamente ao modelo classificador. O modelo classificador é descrito a seguir, no módulo Classificação.

4.5 MÓDULO CLASSIFICAÇÃO

O módulo Classificação executa segundo o modo de operação. Em Modo de Treinamento, o módulo treina o modelo classificador, de forma a habilitá-lo a classificar corretamente vetores que representam *e-mails ham* e vetores que representam *e-mails spam*. Em Modo de Classificação, o módulo classifica, nas classes *spam* e *ham*, vetores que representam *e-mails* novos, enviados pelo Postfix.

O módulo é capaz de igualar a quantidade de vetores *spam* e *ham* sempre que o treinamento do modelo classificador assim o exigir. Para igualar a quantidade de vetores em ambas as classes, o módulo duplica, aleatoriamente, vetores na classe que possua a menor quantidade de vetores.

Para o modelo classificador, um modelo de *machine learning*, os vetores são padrões. No Modo de Treinamento, são acrescidas, ao final dos padrões, mais duas dimensões, contendo dois valores que os identificam como *spam* ou *ham*. Nos padrões *ham*, os dois valores são 0 e 1. Nos padrões *spam*, são 1 e 0.

É importante notar que são utilizadas duas unidades para identificar a classe do *e-mail*. Desta forma, produz-se um resultado com maior grau de certeza na classificação. Por exemplo, caso o modelo classificador apresente o resultado `0,3999` e `0,8`, pode-se dizer que o resultado é, de fato, 0 e 1, ou seja, *ham*.

A figura 16 apresenta o padrão da figura 12, caso este seja um padrão *ham*.

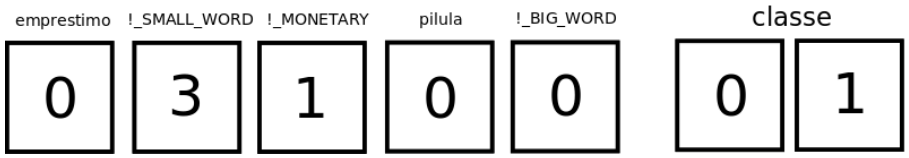

Figura 16 – Padrão de classe *ham*

Ao final do Modo de Treinamento, os valores iniciais e finais dos parâmetros do modelo classificador são salvos em dois arquivos. No Modo de Classificação, os

valores finais dos parâmetros são recuperados do respectivo arquivo, de modo a tornar o modelo apto a classificar corretamente os novos *e-mails*, enviados pelo Postfix.

Quando o modelo não consegue classificar um novo *e-mail*, sua classe é definida como sendo *ham*. Isto é feito para evitar que um possível *e-mail* legítimo seja classificado como *spam*.

O módulo Classificador implementa dois modelos de *machine learning* — *Multi-Layer Perceptron (MLP)* e *Random Forest (RF)*. Os dois modelos são descritos nas duas subseções a seguir.

4.5.1 MODELO MLP

Multi-Layer Perceptron (MLP) é um modelo matemático inspirado na estrutura neuronal do cérebro humano. No cérebro humano, cada neurônio é *ativado* por outros neurônios através de conexões, conhecidas como sinapses. Quando um neurônio recebe ativação de outros neurônios e esta supera seu limiar, ele transmite uma nova ativação aos neurônios seguintes. De forma análoga, no MLP, cada unidade em uma camada `l` recebe ativação das unidades da camada `l-1` e transmite uma nova ativação às unidades da camada `l+1`.

A figura 17 apresenta uma arquitetura típica do MLP. Possui quatro camadas — entrada, escondida 1, escondida 2 e saída — de unidades. As unidades são representadas por círculos na figura. As unidades de uma camada conectam-se às unidades da camada seguinte por meio de conexões, representadas por setas na figura.

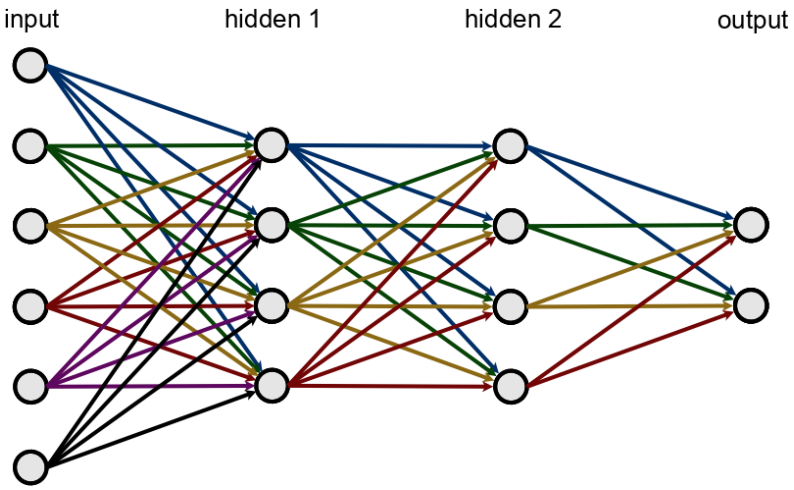

Figura 17 – Arquitetura do MLP

As ativações das unidades da camada de entrada são os valores dos vetores produzidos pelo módulo vetorização (seção 4.4). Cada unidade das camadas seguintes recebe uma ativação net_i que consiste na combinação linear ponderada das ativações das unidades da camada anterior. Portanto, a ativação recebida net_i por uma unidade i de quaisquer das três últimas camadas é dada pela equação 4.3.

$$net_i = \sum_j w_{ij} a_j + bias_i \tag{4.3}$$

onde w_{ij} é o peso da unidade j para a unidade i, a_j é a ativação da unidade j e $bias_i$ é um peso especial que ajusta os valores de net_i.

Como o MLP é um modelo matemático não-linear, a ativação a_i que a unidade i transmite para as unidades da camada seguinte é dada pela equação 4.4.

$$a_i = f(net_i) \tag{4.4}$$

onde $f(x)$ é uma função não-linear diferenciável. Em geral, são usadas as funções sigmóide (LogSig) e tangente sigmóide (TanSig).

Os pesos das conexões são ajustados durante o treinamento do MLP. O treinamento do MLP bem como técnicas para agilizá-lo estão bem descritos na literatura (HAYKIN, 2009).

O SASCA faz uso da biblioteca WEKA (HALL et al., 2009) para implementar o modelo MLP. WEKA é uma biblioteca amplamente conhecida, que implementa muitos modelos de *machine learning*. Tal como o SASCA, possui código aberto.

4.5.2 MODELO RANDOM FOREST

Random Forest (RF) é um modelo de *machine learning* que consiste em um conjunto (*ensemble*) de modelos com arquitetura de árvore de decisão (BREIMAN, 2001). Assim, o modelo RF compõe-se de um conjunto de vários modelos individuais combinados, de forma a produzir um modelo agregado mais poderoso do que qualquer um de seus modelos individuais.

Diferentes modelos podem alcançar um bom desempenho individual. No entanto, tendem a produzir diferentes valores de erros sobre o conjunto de dados. Tipicamente, isso acontece porque cada modelo individual pode se ajustar melhor a uma parte diferente dos dados. Ao combinar diferentes modelos individuais em um conjunto, pode-se balancear seus erros individuais, de forma a reduzir o risco de ajuste parcial sobre os dados.

RFs são amplamente utilizadas na prática e conseguem bons resultados em uma variedade de problemas em várias áreas, tais como (POLAMURI, 2017):

- Comércio bancário: por exemplo, para encontrar os clientes leais e fraudulentos;
- Medicina: por exemplo, para identificar a combinação correta dos componentes para validar um medicamento ou para identificar uma doença com base na análise dos prontuários do paciente;
- Mercado de ações: por exemplo, para identificar o comportamento das ações, bem como a perda esperada ou o lucro, comprando um ativo em particular;
- Comércio eletrônico: por exemplo, para identificar a probabilidade de um cliente gostar dos produtos recomendados com base em tipos similares de clientes.

O treinamento das RFs consiste em sucessivas seleções de características e na criação de árvores de decisão, como mostra o pseudocódigo a seguir.

1. Selecionar aleatoriamente K, de um total de M características, onde $K << M$.
2. Dentre as K características, calcular o nó D, usando o melhor ponto de divisão.
3. Dividir o nó em nós filhos, usando a melhor divisão.
4. Repetir as etapas 1 a 3, até que haja um número pré-determinado L de nós.
5. Repetir N vezes as etapas 1 a 4 para criar uma floresta com N árvores.

RFs classificam amostras por meio de sucessivos testes, contagem de votos e eleição da classe mais votada. O pseudocódigo a seguir e a figura 18 descrevem o processo de classificação das RFs.

1. Para cada amostra, percorrer as N árvore de decisão, de forma a obter N valores de saída.
2. Calcular os votos para cada valor de saída.
3. Considerar a saída mais votada como sendo a saída final, produzida pela RF.

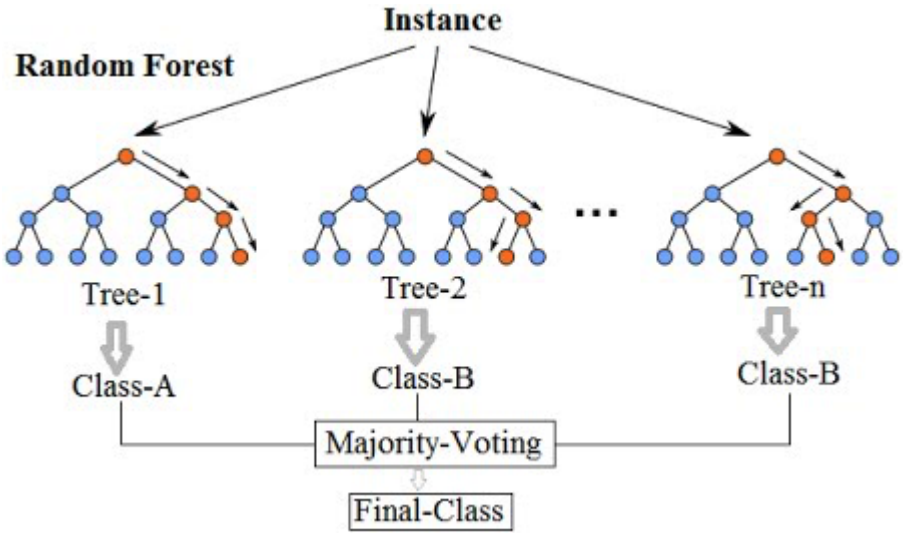

Figura 18 – Exemplo de RF

4.6 MÓDULO ARMAZENAMENTO

A legislação brasileira exige, para fins de auditoria, o armazenamento dos *e-mails* recebidos por um servidor pelo período de cinco anos (FECOMERCIO-SP, 2016). Para fazê-lo, o SASCA conta com um módulo Armazenamento responsável por guardar uma cópia de cada *e-mail* que processa durante o Modo de Execução. Todos os *e-mails* são armazenados em um diretório de *backup* (cópia de segurança).

Adicionalmente, caso o *e-mail* seja *spam*, é guardado também em um diretório próprio para *spams*. O objetivo do armazenamento de *spams* é garantir que nenhum *e-mail* classificado incorretamente, isto é, *ham* classificado como *spam*, seja descartado sem uma consulta prévia ao usuário.

Para garantir que todos os *e-mails* sejam salvos, sem qualquer perda de informação, o módulo leva em consideração o fato de que os diretórios de um sistema operacional possam ter limitações quanto à quantidade de subdiretórios e arquivos que possam conter. O armazenamento pela `Storage` é feito conforme o exemplo a seguir.

Seja um *e-mail* que tenha como destinatário o usuário `joao22` no domínio `empresa2000.com`, ou seja, `joao22@empresa2000.com`. Usando o endereço de *e-mail* como referência, o módulo cria subdiretórios com base em cada letra do nome do usuário dentro de um diretório que possui o nome do domínio. Assim, o *e-mail* será armazenado dentro do diretório `empresa2000/j/o/a/o/2/2/`. O *e-mail* é armazenado em um arquivo cujo nome é `email.eml`, onde `email` é substituído por um conjunto *único* de caracteres, gerados com base na data corrente do sistema, através do método `System.currentTimeMillis()`.

Para cada *e-mail spam* recebido, faz-se necessário que seu destinatário confirme se, de fato, é realmente um *spam*. Para isto, o módulo salva determinadas informações de controle para saber se o destinatário já foi notificado ou não a respeito deste *e-mail*. Caso não tenha sido notificado, o módulo Notificação (apresentado na seção 4.7 seguinte) notificá-lo-a. Caso o destinatário já tenha sido notificado e classificado o *e-mail*, este poderá ser enviado para o retreinamento do modelo classificador.

O módulo Armazenamento pode guardar até bilhões de arquivos de *e-mail* para cada usuário, dependendo das limitações do sistema de arquivos utilizado. Maiores

informações sobre limitações de sistemas de arquivos podem ser encontradas no anexo B.

4.7 MÓDULO NOTIFICAÇÃO

Como todo *anti-spam*, o SASCA pode classificar como *spam e-mails* que sejam legítimos, ou seja, gerando um falso-positivo. Falsos-positivos podem causar sérios incômodos aos usuários caso o *anti-spam* os descarte. Para evitar tais incômodos, o SASCA, por intermédio de seu módulo Notificação, notifica os usuários sobre seus *e-mails* classificados como *spam*.

O módulo de Notificação é executado por um processo separado do SASCA, como por exemplo, por um agendador de tarefas. Em sua execução, o módulo verifica quais usuários receberam *spam*, em um determinado período de tempo, e notifica-lhes por meio de um relatório. A figura 19 apresenta o formato do relatório enviado aos usuários.

Data	Assunto	De	Ação
16/11/2015 19:03:42	Register for the Wolfram Virtual Conference--November 17 and 19	jamie_peterson@wolfram.com	Aceitar · Rejeitar

Figura 19 – Formato do relatório enviado pelo módulo Notificação

O relatório é composto por quatro colunas. A primeira, informa a data/hora em que o *e-mail* foi recebido. A segunda coluna contém o assunto e um *link* para acesso ao conteúdo do *e-mail*. A terceira, informa o remetente. A última coluna possui dois *links* de ação — aceitar ou rejeitar — o *e-mail*. Em versões futuras do SASCA, cogita-se incluir as opções "aceitar sempre", "rejeitar sempre" e outras que possam ser convenientes.

A coluna de ação, através da ação "aceitar", permite ao usuário receber, em sua caixa postal, o *e-mail* barrado. Na coluna de ação, há também a ação "rejeitar". Ao clicar nesta opção, o usuário confirma para o SASCA que, de fato, o *e-mail* é *spam*. Caso o faça, o módulo exclui o *e-mail* da *pasta* de *spams* e o envia a outra unidade de armazenamento, onde são armazenados *e-mails* usados para treinamento do modelo classificador.

O usuário pode, outrossim, optar por ignorar os *e-mails* contidos no relatório. Caso o faça, após um período pré-definido, passam a ser considerados "oficialmente" como *spam*, sendo seus destinos idênticos aos dos *e-mails* clicados na ação "rejeitar".

4.8 CONFIGURAÇÃO E EXECUÇÃO DO SASCA

O SASCA é configurado por meio de parâmetros. Os parâmetros de configuração do modelo MLP estão inseridos no arquivo `neural.properties`. Todos os demais parâmetros de configuração do SASCA estão inseridos no arquivo `main.properties`. Os parâmetros possuem o formato `CHAVE = VALOR`, onde `CHAVE` é o nome do parâmetro e `VALOR` é seu valor.

O SASCA é executado por meio de comandos enviados via terminal. Desta forma, para realizar a execução ou do *anti-spam* SASCA ou de um de seus módulos (de forma individual), é preciso, respectivamente, ou configurar os parâmetros do SASCA ou do módulo, e emitir o comando apropriado para a execução. Os comandos possuem o formato `java -jar SASCA.jar <módulo>`, onde `<módulo>` é o nome do módulo ou, no caso de execução do próprio SASCA, `server`.

O *anti-spam* SASCA é sempre executado em Modo de Execução. Por outro lado, os módulos, individualmente, são executados em Modo de Treinamento. A configuração e execução de cada módulo e do próprio SASCA são descritas nas subseções a seguir.

4.8.1 CONFIGURAÇÃO E EXECUÇÃO DO MÓDULO PRÉ-FILTRO

Para executar o módulo Pré-Filtro em Modo de Treinamento é preciso primeiramente configurar os parâmetros que definem o diretório em que os *e-mails* originais estão e o diretório para onde os *e-mails* filtrados irão. Os parâmetros são descritos a seguir.

- *E-mails* originais.

  ```
  RAW_SPAM_FOLDER = /diretório/para/emails/brutos/spam
  ```

  ```
  RAW_NOT_SPAM_FOLDER = /diretório/para/emails/brutos/ham
  ```

- *E-mails* filtrados.

  `PROCESSED_SPAM_FOLDER = /diretório/para/emails/filtrados/spam`

  `PROCESSED_NOT_SPAM_FOLDER = /diretório/para/emails/filtrados/ham`

Após definir os parâmetros, deve-se executar o comando
`java -jar SAS.jar filter`.

4.8.2 CONFIGURAÇÃO E EXECUÇÃO DO MÓDULO SELEÇÃO DE *TOKENS*

Para executar o módulo Seleção de *Tokens* em Modo de Treinamento é preciso primeiramente configurar os parâmetros que definem o diretório em que os *e-mails* filtrados estão, o arquivo que conterá os *tokens* por ordem de relevância e o método estatístico de seleção de *tokens* a ser usado. Os parâmetros são descritos a seguir.

- *E-mails* filtrados.

  `PROCESSED_SPAM_FOLDER = /diretório/para/emails/filtrados/spam`

  `PROCESSED_NOT_SPAM_FOLDER = /diretório/para/emails/filtrados/ham`

- Arquivo de *tokens*.

  `STATISTICS_FILE = /diretório/para/estatisticas.dat`

- Método estatístico. Pode ser FD (*Frequency Distribution*), MI (*Mutual Information*) ou CHI2 (*Chi-Squared*).

  `STATISTICS_METHOD = FD`

Após definir os parâmetros, deve-se executar o comando
`java -jar SAS.jar statistics`.

4.8.3 CONFIGURAÇÃO E EXECUÇÃO DO MÓDULO VETORIZAÇÃO

Para executar o módulo Vetorização em Modo de Treinamento é preciso primeiramente configurar os parâmetros que definem o diretório em que os *e-mails* filtrados estão, o arquivo que contém os *tokens* por ordem de relevância, o diretório

em que serão armazenados os vetores e sua dimensionalidade. A dimensionalidade não deve ultrapassar a quantidade de *tokens* disponível no arquivo. Os parâmetros são descritos a seguir.

- *E-mails* filtrados.

  `PROCESSED_SPAM_FOLDER = /diretório/para/emails/filtrados/spam`

  `PROCESSED_NOT_SPAM_FOLDER = /diretório/para/emails/filtrados/ham`

- Arquivo de *tokens*.

  `STATISTICS_FILE = /diretório/para/estatisticas.dat`

- Diretório dos vetores.

  `VECTOR_OUT_FOLDER = /diretório/para/vetores/`

- Dimensionalidade dos vetores.

  `VECTOR_LENGTH = 100`

Após definir os parâmetros, deve-se executar o comando
`java -jar SAS.jar vector`

Com a execução do comando, o módulo Vetorização será executado. Os vetores que gerar serão salvos no diretório definido. O diretório contém dois pares de arquivos. O primeiro par é composto pelo arquivo `ham`, que contém todos os vetores da classe *ham* gerados, e o arquivo `ham.link`, que contém o nome do arquivo de *e-mail ham* correspondente a cada vetor no arquivo `ham`. Em maior detalhe, o vetor `n`, no arquivo `ham`, corresponde ao *e-mail* cujo nome é a n-ésima linha do arquivo `ham.link`. O segundo par é o análogo para *spam*.

4.8.4 CONFIGURAÇÃO E EXECUÇÃO DO MÓDULO CLASSIFICAÇÃO (MODO DE TREINAMENTO)

4.8.4.1 MODO DE TREINAMENTO DO MLP

O SASCA conta com duas implementações — interna e externa — do modelo classificador MLP. A implementação interna é denominada MLP-int. A implementação externa, que faz uso da biblioteca WEKA, é denominada MLP-WEKA.

Para executar o MLP-int do módulo Classificação, em Modo de Treinamento, é preciso primeiramente configurar os parâmetros a seguir.

- Diretório dos vetores.

  `VECTOR_OUT_FOLDER = /diretório/com/vetores/`

- Diretório em que são armazenados os melhores pesos iniciais do MLP.

  `NEURAL_FOLDER = /diretório/para/pesos_neurais/`

- Porcentagem de *e-mails* que deve conter o arquivo de vetores de validação. Os demais vetores estarão no arquivo de vetores de treinamento.

  `VALIDATION_PERCENT = 0.30`

- Quantidade de treinamentos a serem realizados.

  `NUMBER_OF_TRAINS = 20`

- Quantidade de `threads` a serem criadas. Define-se, com isto, a quantidade de treinamentos sendo executados em paralelo.

  `NUMBER_OF_ACTIVES_THREADS = 4`

- Quantidade de unidades na primeira e segunda camadas escondidas.

  `DEFAULT_FIRST_HIDDEN_LENGTH = 10`

  `DEFAULT_SECOND_HIDDEN_LENGTH = 10`

- Funções de transferência entre camadas. Os valores possíveis são `TANSIG` (tangente sigmóide), `LOGSIG` (sigmóide logística) e `LINEAR`.

  `DEFAULT_FIRST_HIDDEN_FUNCTION = TANSIG`

  `DEFAULT_SECOND_HIDDEN_FUNCTION = TANSIG`

  `DEFAULT_OUTPUT_FUNCTION = LOGSIG`

- Quantidade de épocas de treinamento.

  `DEFAULT_EPOCHS = 20`

- *Momentum* inicial.

  `DEFAULT_MOMENTUM = 0.9`

- Taxa de aprendizagem inicial.

  `DEFAULT_LEARN_RATE = 0.00001`

- Semente do gerador de pesos iniciais. Os valores possíveis são `0` (zero), para definir uma semente aleatória ou `PRIME`, para definir uma semente aleatória com valor primo, ou `<inteiro>`, para definir um valor inteiro para a semente.

  `DEFAULT_RANDOMIZER_SEED = PRIME`

Após definir os parâmetros, deve-se executar o comando
`java -jar SAS.jar mlp`

Para executar o MLP-WEKA do módulo Classificação, em Modo de Treinamento, é preciso configurar os parâmetros a seguir.

- `CLASSIFICATOR_MODE = WEKA`
- `WEKA_MODEL = MLP`
- `WEKA_PARAMS = -L 0.3 -M 0.2 -N 5000 -V 33 -S 1 -E 20 -H {10, 5}`

Os argumentos do parâmetro `WEKA_PARAMS` são descritos na tabela 1.

Tabela 1 – Argumentos do parâmetro `WEKA_PARAMS` para o modelo MLP — NF: dimensionalidade dos vetores

Parâmetro	Descrição	Valor
-L	taxa de aprendizado do backpropagation	0.3
-M	taxa de momentum do backpropagation	0.2
-N	número máximo de épocas de treinamento	5000
-V	percentual do conjunto de treinamento usado para validação	33
-S	semente do gerador de números aleatórios	1
-E	número consecutivo de erros permitidos para testes de validação antes da rede terminar	20
-H	quantidades de neurônios nas camadas escondidas da rede neural	{(NF + 2) / 2, (NF + 2) / 4}

Após definir os parâmetros, deve-se executar o comando
`java -jar SAS.jar weka-mode`

4.8.4.2 MODO DE TREINAMENTO DO RF

Para executar o RF do módulo Classificação, em Modo de Treinamento, é preciso primeiramente configurar os parâmetros a seguir.

- `CLASSIFICATOR_MODE = WEKA`
- `WEKA_MODEL = RF`
- `WEKA_PARAMS = -I 100 -K 0 -S 1`

Os argumentos do parâmetro `WEKA_PARAMS` são descritos na tabela 2.

Tabela 2 – Argumentos do parâmetro `WEKA_PARAMS` para o modelo RF

Parâmetro	Descrição	Valor
-I	número de árvores que serão construídas	100
-K	número de características consideradas	0
-S	semente do gerador de números aleatórios	1

Após definir os parâmetros, deve-se executar o comando
`java -jar SAS.jar weka-mode`

4.8.5 CONFIGURAÇÃO E EXECUÇÃO DO PRÓPRIO SASCA

Para executar o SASCA em Modo de Execução, ou seja, para executá-lo como um *anti-spam*, que recebe, classifica, armazena e entrega *e-mails* advindos do Postfix (ou outro MTA), é preciso primeiramente configurar os parâmetros que definem o arquivo que contém os pesos finais (após o treinamento) do MLP, o arquivo que contém os *tokens* por ordem de relevância e, opcionalmente, os diretórios de armazenamento de *backups* e de *spams*. Os parâmetros são descritos a seguir.

- Arquivo com os pesos finais do MLP.

  `WEIGHTS_FILE = /diretório/para/pesos_finais.dat`

- Arquivo de *tokens*.

  `STATISTICS_FILE = /diretório/para/estatísticas.dat`

- Diretório de armazenamento de *backups* (opcional).

  `STORE_BACKUP_FOLDER = /diretório/para/backups/`

- Diretório de armazenamento de *spams* (opcional).

  `STORE_SPAM_FOLDER = /diretório/para/spams/`

Após definir os parâmetros, deve-se executar o comando
`java -jar SAS.jar server`

5 FERRAMENTAS AUXILIARES

Este capítulo apresenta as mais importantes ferramentas desenvolvidas. Elas suportam a realização dos experimentos descritos no capítulo 6.

5.1 FERRAMENTA *CRAWLER* PARA OBTER *E-MAILS*

A Universidade Federal de Itajubá (UNIFEI) utiliza o *anti-spam* comercial CanIt. Todos os *e-mails* recebidos pela universidade são classificados e armazenados em um banco de dados pelo CanIt.

A empresa responsável pelo CanIt não disponibiliza informações sobre a arquitetura e o dicionário de seu banco de dados. A única forma, disponibilizada pelo fabricante, para manipular os *e-mails* armazenados no banco de dados, é por meio da ferramenta de gerenciamento do CanIt. A ferramenta é gráfica, composta por páginas HTML, e, por conseguinte, é acessada através de navegadores de Internet. A figura 20 apresenta a página de *login* da ferramenta.

Para obter os *e-mails* do banco de dados do CanIt, portanto, foi criado um *crawler* (do inglês, rastreador) que, simulando a operação do administrador, manipula as páginas HTML da ferramenta de gerenciamento.

A implementação do *crawler* faz uso da biblioteca `HtmlUnit`. A biblioteca implementa todas as funcionalidades de um navegador de Internet, mas sem implementar uma interface gráfica. Algumas das funcionalidades implementadas pela `HtmlUnit` são, por exemplo, a possibilidade de percorrer em *links*, de inserir conteúdo e de executar JavaScript (BOWLER, 2016).

Com o uso do *crawler*, foram capturados 353.151 *e-mails ham* e 509.076 *e-mails spam*. Cada *e-mail* foi armazenado em um arquivo. Em seguida, cada *e-mail* foi processado pelo módulo Pré-Filtro, de forma a transformá-lo em um conjunto de *tokens*. Por fim, todos os *e-mails* originais foram destruídos.

Assim, a base de *e-mails* da UNIFEI utilizada nos experimentos é composta por *e-mails* não-originais, mas sim, representados por conjuntos de *tokens*. A representação por conjuntos de *tokens* não permite a reconstrução do *e-mail*

Figura 20 – Página de *login* da ferramenta de gerenciamento do CanIt

original. Desta forma, preserva-se o anonimato do remetente e destinatários, da rota percorrida, bem como o sigilo do corpo e anexos de cada *e-mail* original.

5.2 FERRAMENTA PARA RECONSTRUIR *E-MAILS*

A realização dos experimentos exige o uso de *e-mails* não-processados. Conforme mencionado na seção anterior (seção 5.1), a base de *e-mails* da UNIFEI é uma base processada. Para que possa ser utilizada nos experimentos, é necessário torná-la não-processada, ou seja, convertê-la novamente em uma base pseudo-original.

A *ferramenta para reconstruir e-mails* faz esta conversão. A conversão dos *e-mails* consiste na substituição de determinados *tokens*, conforme descrito a seguir.

- `!_SMALL_WORD`: é substituído por uma *string* contendo a letra `a`, com tamanho, definido aleatoriamente, de 1 a 3 caracteres.

- `!_BIG_WORD`: é substituído por uma *string* contendo a letra `b`, com tamanho, definido aleatoriamente, de 30 a 39 caracteres.
- `!_MONETARY`: é substituído por uma *string* contendo os caracteres `R$`, seguidos de um número, gerado aleatoriamente, entre 0 e 9999.
- `!_NUMBER`: é substituído por um número, gerado aleatoriamente, entre 0 e 9999.
- `!_URL`: é substituído pelo *link* "http://www.somesite.com.br/link/pagina/qualquer.html".
- `!_IMAGE`: é substituído pela *tag* de imagem `<img src="path/img.png" alt="image" height="42" width="42">`.
- `!_in_<attr>`: Os *tokens* `!_in_<attr>` encontrados em uma sequência, ou seja, sem nenhum outro tipo de *token* entre eles, são substituídos por uma *tag* `p` (paragrafo) contendo a sequência de atributos.

Se o *e-mail* contiver apenas *tokens* de texto, é atribuído o valor `text/plain` a seu atributo `Content-type`. Caso contrário, é atribuído o valor `text/html` (ver seção 2.4).

5.3 FERRAMENTA PARA CORRIGIR BASES DE *E-MAIL*

Bases de *e-mails* são, frequentemente, inconsistentes. Podem possuir dois ou mais *e-mails* similares, ou seja, que se diferenciam uns dos outros por um valor δ de *tokens* muito baixo, classificados em ambas as classes *ham* e *spam*. Isto ocorre, devido a diversos motivos, variando desde classificações feitas por indivíduos distintos até classificações feitas por *anti-spams* em diferentes momentos de tempo.

Assim, bases podem conter *e-mails* muito similares (por exemplo, com $\delta = 3$) ou mesmo *e-mails* iguais (com $\delta = 0$) classificados, de forma inconsistente, em ambas as classes. A *ferramenta para corrigir bases de e-mail* corrige estas bases, tornando-as consistentes. A ferramenta executa os passos a seguir.

1. O usuário define o valor de δ como sendo $\delta = integerNumber$.

2. O usuário define o valor da dimensionalidade dos vetores que representam os *e-mails* como sendo n.

3. Determina-se quais são os *e-mails* similares, ou seja, que se diferenciam uns dos outros por um valor $\delta <= integerNumber$ de *tokens*, obtendo-se diversos conjuntos de *e-mails* similares.

4. Para cada conjunto, verifica-se a classe dominante, ou seja, com a maior quantidade de *e-mails*, atribuindo o valor desta classe para todos os *e-mails* do conjunto. Por exemplo, se um conjunto de *e-mails* similares contiver 17 *e-mails ham* e 54 *e-mails spam*, atribui-se a classe *spam* para todos os 71 *e-mails* do conjunto.

5. Executa-se o Módulo Vetorização para representar os *e-mails* por meio de vetores de dimensionalidade n

6. Executam-se os passos 3 e 4 sobre estes vetores. É importante notar que este passo pode mudar a classe de vetores. Isto significa que, de forma indireta, *e-mails* podem mudar novamente de classe.

5.4 FERRAMENTA PARA REDUZIR A DIMENSIONALIDADE DE VETORES

Os três métodos estatísticos usados pelo módulo Seleção de *Tokens*, apresentados na seção 4.3, classificam, por ordem de relevância, os *tokens* de *e-mails* em uma base. A representação vetorial dos *e-mails* têm por base esta classificação. Em outras palavras, cada dimensão do vetor, que representa um *e-mail*, representa um relevante *token* selecionado.

Os três métodos estatísticos permitem, portanto, que *e-mails* sejam representados por vetores de baixa dimensionalidade, ou seja, permitem que sejam representados por uma quantidade, de *tokens* relevantes, pequena em relação à quantidade de *tokens* existente na base de *e-mails*.

Existem ferramentas, porém, que permitem reduzir ainda mais a dimensionalidade dos vetores de uma base. Ao contrário dos métodos estatísticos, que analisam

tokens, estas ferramentas analisam, diretamente, vetores. De fato, analizam a correlação estatística entre todos os vetores da base, de modo a determinar as dimensões mais relevantes, removendo, dos vetores, as demais.

A *ferramenta para reduzir a dimensionalidade de vetores* reduz a dimensionalidade dos vetores de uma base. A ferramenta faz uso do algoritmo "Multi-Objective Evolutionary Feature Selection" (JIMéNEZ et al., 2017). O algoritmo é implementado pela biblioteca WEKA.

6 EXPERIMENTOS

Este capítulo apresenta os experimentos realizados. Descreve, outrossim, as bases de *e-mail* utilizadas, a metodologia empregada, os resultados alcançados e realiza uma avaliação sobre os resultados e sobre o desempenho obtido pelo SASCA.

6.1 BASES DE *E-MAIL*

Os *e-mails* utilizados nos experimentos são reais. Foram coletados do banco de dados do CanIt, *anti-spam* usado na UNIFEI. Uma ferramenta *crawler* (seção 5.1) foi utilizada para realizar a coleta.

Em um período de três meses, no segundo semestre de 2016, foram coletados 353.151 *e-mails ham* e 509.076 *e-mails spam*. Conforme descrito na seção 5.1, cada *e-mail* foi processado pelo módulo Pré-Filtro, de forma a transformá-lo em um conjunto de *tokens*. Os *e-mails* originais foram todos destruídos.

A base de *e-mails* processados, doravante chamada *base UNIFEI*, é composta, portanto, por *e-mails* reais, porém sob a representação dada por conjuntos de *tokens*. A representação por conjuntos de *tokens* não permite a reconstrução do *e-mail* original, preservando-se, assim, o anonimato do remetente e destinatários, da rota percorrida, bem como o sigilo do corpo e anexos de cada *e-mail* original.

O histograma da figura 21 apresenta os tamanhos dos *e-mails* da base UNIFEI.

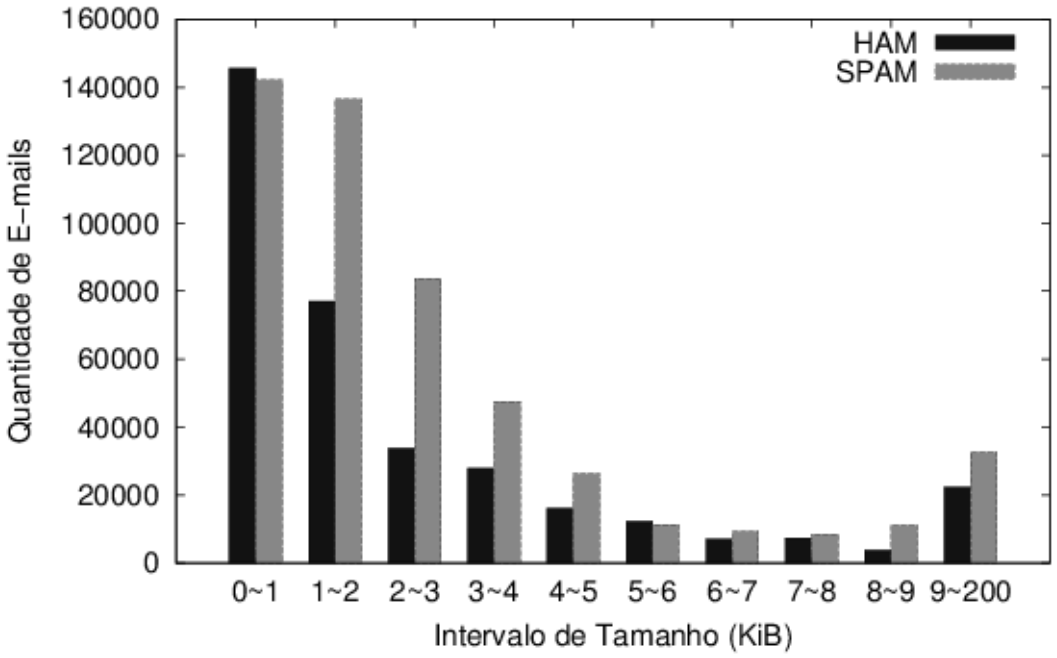

Figura 21 – *E-mails* coletados

Pelo histograma, é possível notar que pode haver arquivos vazios. Um arquivo vazio contém um *e-mail* sem *tokens*. Isto pode ocorrer, por exemplo, caso o *e-mail* original contivesse apenas anexos ou caso possuísse, em seu corpo, somente caracteres inválidos. A maioria dos *e-mails* possui tamanho entre 0 (zero) e 3 KB. Representam cerca de 70% da base de *e-mails*.

Para que os *e-mails* da base UNIFEI pudessem ser visualizados graficamente, o módulo Vetorização foi empregado para convertê-los, através do método estatístico FD, em vetores de dimensão 1024. Vetores iguais foram, então, separados em conjuntos. Verificou-se, então, que um conjunto qualquer poderia conter *e-mails* classificados em ambas as classes *ham* e *spam*. Quando isto ocorria, uma nova classe *ham/spam* foi atribuída a todos os vetores do conjunto. Por fim, os vetores de dimensão 1024 foram transformados em novos vetores de duas dimensões, por meio do método estatístico t-SNE (MAATEN; HINTON, 2008).

A figura 22 apresenta, de forma gráfica, os vetores bidimensionais, ou seja, os *e-mails* da base UNIFEI. Na figura, *e-mails ham* aparecem como pontos azuis,

e-mails spam, como pontos vermelhos e *e-mails ham/spam* aparecem como pontos pretos.

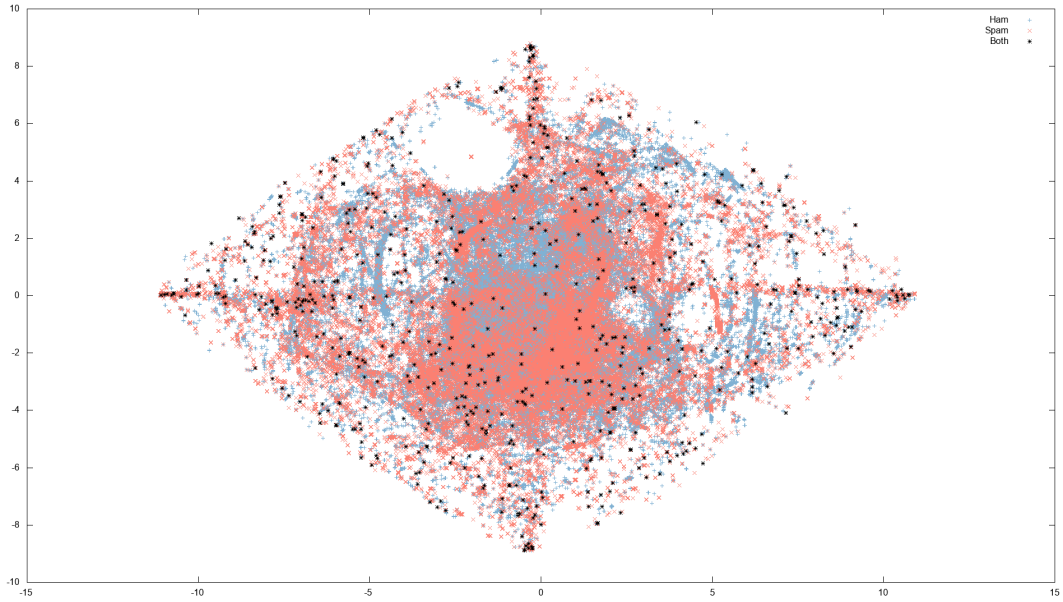

Figura 22 – *E-mails* da base UNIFEI — ham: azul; spam: vermelho; ham/spam: preto

Pela quantidade de pontos pretos, percebe-se que a base UNIFEI é altamente inconsistente. A inconsistência da base é devida unicamente à classificação inconsistente dos *e-mails* feita pelo *anti-spam* CanIt. A classificação inconsistente pode ter ocorrido por diversos motivos. Tais motivos são difíceis de serem discernidos, uma vez que o CanIt, por ser um *anti-spam* comercial, não possui código aberto. Provavelmente, porém, pode-se supor que as classificações inconsistentes foram feitas em diferentes momentos de tempo, durante os quais as listas negras consultadas foram atualizadas, passando a incluir os endereços dos *spammers*.

Uma nova base — UNIFEI-$\delta 0$ — foi criada a partir da base UNIFEI. A ferramenta descrita na seção 5.3 foi utilizada para criá-la. O valor de $\delta = 0$ foi utilizado para a criação da base UNIFEI-$\delta 0$. A nova base é mais consistente que a base UNIFEI, por ter sido criada pela ferramenta.

As duas bases foram usadas nos experimentos. A base UNIFEI possui 353.151 *e-mails ham* e 509.076 *e-mails spam*. A base UNIFEI-$\delta 0$ possui 353.910 *e-mails*

ham e 508.317 *e-mails spam*.

6.2 VETORES

As figuras 23, 24 e 25 apresentam, de forma gráfica, a quantidade de vetores gerados pelos métodos FD, MI e CHI2, respectivamente, sobre a base UNIFEI. As tabelas 3, 4 e 5 apresentam, de forma numérica, esta mesma quantidade. Além disto, apresentam também a quantidade de vetores da base UNIFEI-δ0.

Os vetores foram gerados com as dimensionalidades `8`, `16`, `32`, `64`, `128`, `256`, `512` e `1024`.

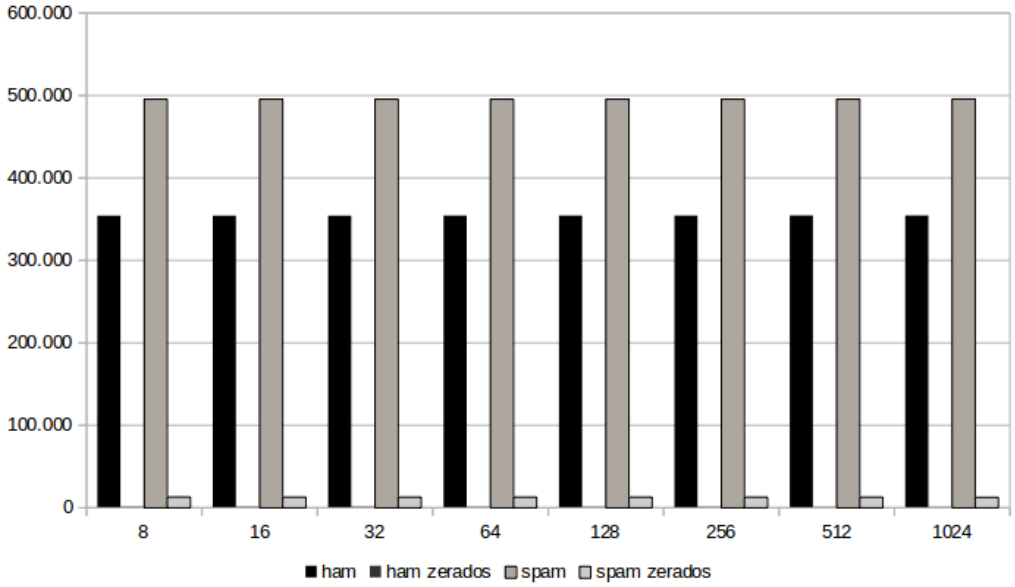

Figura 23 – Vetores gerados pelo método *Frequency Distribution*

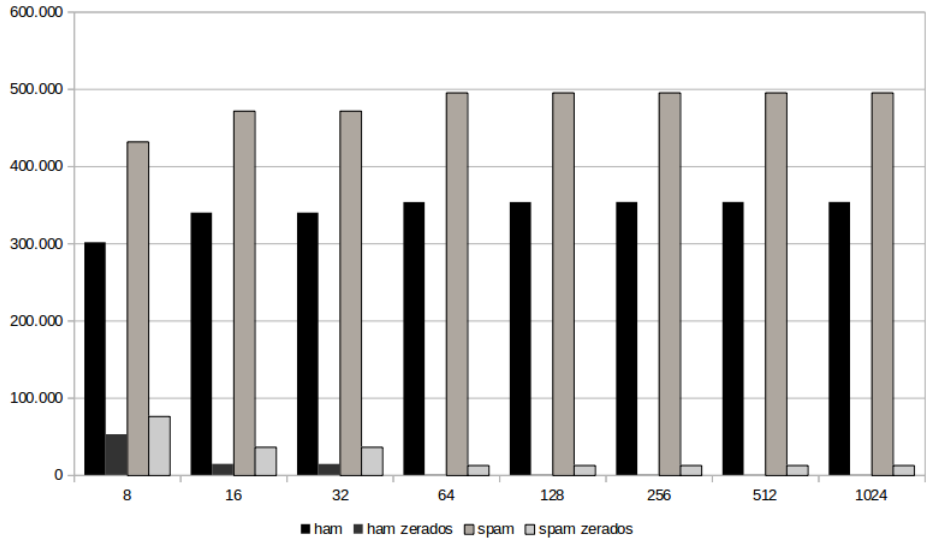

Figura 24 – Vetores gerados pelo método *Mutual Information*

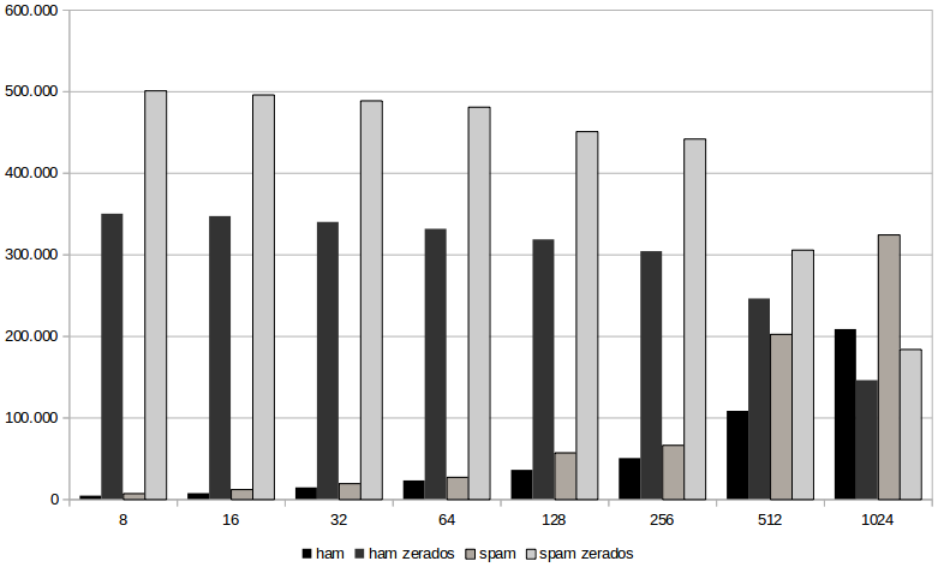

Figura 25 – Vetores gerados pelo método *Squared-CHI*

Tabela 3 – Vetores gerados pelo método *Frequency Distribution*

dimensão	**classe**	**nulos**	**base UNIFEI**	**base UNIFEI-δ0**
8	ham	546	353.365	347.446
	spam	12.831	495.487	501.406
16	ham	506	353.405	349.701
	spam	12.795	495.523	499.227
32	ham	495	353.416	355.024
	spam	12.795	495.523	493.915
64	ham	470	353.441	355.019
	spam	12.785	495.533	493.955
128	ham	463	353.448	355.284
	spam	12.780	495.538	493.702
256	ham	410	353.501	354.859
	spam	12.780	495.538	494.180
512	ham	329	353.582	354.579
	spam	12.752	495.566	494.569
1024	ham	287	353.624	353.868
	spam	12.591	495.727	495.483

Tabela 4 – Vetores gerados pelo método *Mutual Information*

dimensão	**classe**	**nulos**	**base UNIFEI**	**base UNIFEI-δ0**
8	ham	52.677	301.234	102.806
	spam	76.335	431.983	630.411
16	ham	14.269	339.642	323.961
	spam	36.488	471.830	487.511
32	ham	14.175	339.736	319.521
	spam	36.402	471.916	492.131
64	ham	516	353.395	347.165
	spam	12.816	495.502	501.732
128	ham	484	353.427	356.422
	spam	12.816	495.502	492.507
256	ham	443	353.468	354.027
	spam	12.788	495.530	494.971
512	ham	408	353.503	354.041
	spam	12.788	495.530	494.992
1024	ham	408	353.503	353.906
	spam	12.788	495.530	495.127

Tabela 5 – Vetores gerados pelo método *CHI-Squared*

dimensão	**classe**	**nulos**	**base UNIFEI**	**base UNIFEI-δ0**
8	ham	349.792	4.119	3.455
	spam	501.088	7.230	7.894
16	ham	346.814	7.097	5.478
	spam	496.034	12.284	13.903
32	ham	339.675	14.236	14.829
	spam	488.763	19.555	18.962
64	ham	331.185	22.726	23.626
	spam	481.042	27.276	26.376
128	ham	318.241	35.670	27.537
	spam	451.072	57.246	65.379
256	ham	303.849	50.062	45.597
	spam	441.896	66.422	70.887
512	ham	245.856	108.055	95.260
	spam	305.811	202.507	215.302
1024	ham	145.687	208.224	201.855
	spam	183.906	324.412	330.781

O método FD apresenta a menor quantidade de perdas de *e-mails*, ou seja, de vetores nulos, em todas as dimensionalidades. Com vetores de 8 dimensões, há, respectivamente, 546 (0,15%) e 12.831 (2,52%) de vetores *ham* e *spam* nulos. Com vetores de 1024 dimensões, há 287 (0,08%) e 12.591 (2,47%) vetores nulos.

No método MI, com vetores de 8 dimensões, há, respectivamente, 14,88% e 15,02% de vetores *ham* e *spam* nulos. Com vetores de 1024 dimensões, porém, há 0,12% e 2,52% de vetores nulos, percentuais bem próximos aos do método FD. Em verdade, a partir de 64 dimensões, o método MI passa a gerar uma quantidade de vetores nulos similar à do FD.

O método que gera a maior quantidade de vetores nulos é o CHI2. Com vetores de 8 dimensões, há, respectivamente, 98,84% e 98,58% de vetores *ham* e *spam* nulos. Com vetores de 1024 dimensões, há 41,16% e 36,18% de vetores nulos.

6.3 EXPERIMENTO 1

O primeiro experimento teve por objetivo avaliar o tempo computacional médio de execução gasto pelos módulos computacionalmente mais caros do SASCA. Tais

módulos são o módulo SMTP e o módulo Pré-Filtro. Os tempos computacionais médios de treinamento e de classificação dos modelos do módulo Classificação são avaliados na seção 6.5.1. Os demais módulos apresentam tempos médios de execução inferiores a um milissegundo.

Neste experimento, os módulos do SASCA foram executados em um computador com oito *cores* de 2,95 GHz, 8 GB de memória RAM, executando o sistema operacional Ubuntu 16.04. A base de *e-mails* usada, gerada a partir da base UNIFEI, era composta por 36.105 *e-mails*, dos quais 15.122 eram *ham* e 20.983 *spams*. A base foi processada pela ferramenta para reconstruir *e-mails*, descrita na seção 5.2. O tamanho de seus *e-mails* variou de 3 KB a 74 KB.

A execução do experimento seguiu os passos apresentados no diagrama da figura 26.

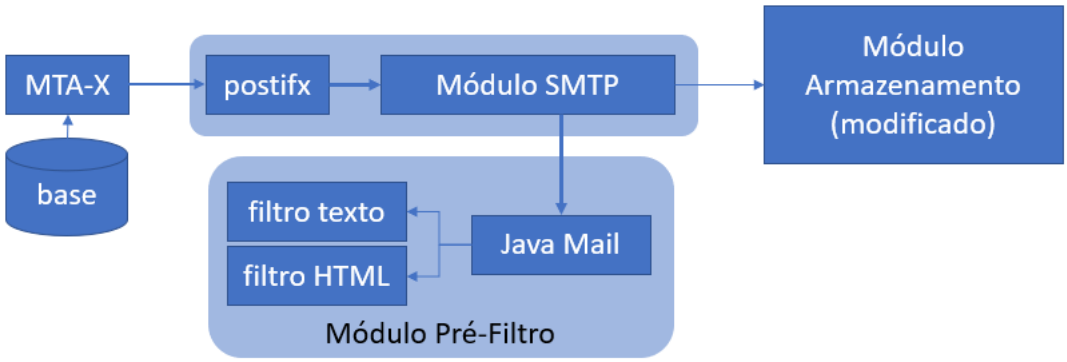

Figura 26 – Diagrama do experimento 1

Os *e-mails*, armazenados no disco rígido do computador, foram enviados por um MTA modificado, denominado MTA-X, para o Postfix. O MTA-X registrou o horário de envio de cada *e-mail*, salvando-o em um arquivo. O Postfix foi configurado para enviar os *e-mails* recebidos para o módulo SMTP do SASCA. O módulo SMTP foi configurado para enviar cada *e-mail* recebido diretamente a um módulo Armazenamento modificado, cuja função era apenas gravar o nome e a hora de chegada do *e-mail* recebido.

Os tempos computacionais médios de execução dos módulos SMTP e Pré-Filtro foram avaliados. A avaliação do módulo Pré-Filtro foi dividida, de forma a avaliar

individualmente seus componentes internos. A avaliação do custo computacional de cada componente é necessária, posto que o custo está diretamente relacionado ao conteúdo do *e-mail* recebido.

A base utilizada no experimento foi a base UNIFEI. Os *e-mails* da base consistem em *e-mails* processados pelo módulo Pré-Filtro. Portanto, a ferramenta descrita na seção 5.2 foi usada para reconstruir os *e-mails*.

Os *e-mails* da base UNIFEI são compostos por texto unicamente ou por ambos, texto e código HTML. Os 22.273 *e-mails* da base, cujos corpos eram compostos por texto e código HTML, foram usados para avaliar o *filtro* HTML do módulo Pré-Filtro. Para a avaliação do filtro de texto, foram usados os demais 13.832 *e-mails* da base, cujos corpos eram compostos por texto unicamente.

Em cada avaliação realizada, os *e-mails* foram enviados, por dez vezes, do MTA-X ao Módulo Armazenamento modificado. Ao final, foi calculado o tempo computacional médio de execução de cada módulo para processar um *e-mail*.

A tabela 6 apresenta os resultados.

Tabela 6 – Resultados do experimento 1

Parte avaliada	Tempo médio (ms)
Módulo SMTP	41,745
JavaMail	0,147
Filtro de HTML	2,410
Filtro de Texto	0,038

Dos resultados apresentados na tabela, observa-se que o maior tempo computacional é o de recebimento do *e-mail*; isto é, o tempo que o *e-mail* leva do momento que chega ao computador, via interface de rede (no caso comum), até o módulo Pré-Filtro. Este tempo tende a ser maior quando o *e-mail* é recebido através da Internet, pois o postifx precisa realizar todos os procedimentos do protocolo SMTP (seção 2.2).

A JavaMail apresenta um tempo computacional médio relativamente pequeno. Seu desempenho está diretamente ligado ao tamanho do cabeçalho do *e-mail*. O cabeçalho de cada *e-mail* é similar ao descrito na seção 2.4, posto que, no experimento, apenas um computador (servidor) envia *e-mails*. No entanto, deve-se ter em mente que os *e-mails* que circulam na Internet apresentam, usualmente,

cabeçalhos maiores, posto que, usualmente, passam por mais de um servidor de *e-mail*.

Da mesma forma que a JavaMail, o *filtro* HTML também tem seu tempo de processamento diretamente relacionado com a complexidade do código HTML presente no *e-mail*. A ferramenta de reconstrução de *e-mails* procura remontar o código HTML, mas a ferramenta não consegue reproduzir o grau de complexidade original; isto é devido ao módulo Pré-Filtro não preservar a hierarquia das *tags* HTML.

6.4 EXPERIMENTO 2

O segundo experimento teve, por objetivo, avaliar os dois *anti-spam* — CanIt e SASCA — sobre uma mesma base de *e-mails*. A figura 27 apresenta a arquitetura empregada no experimento.

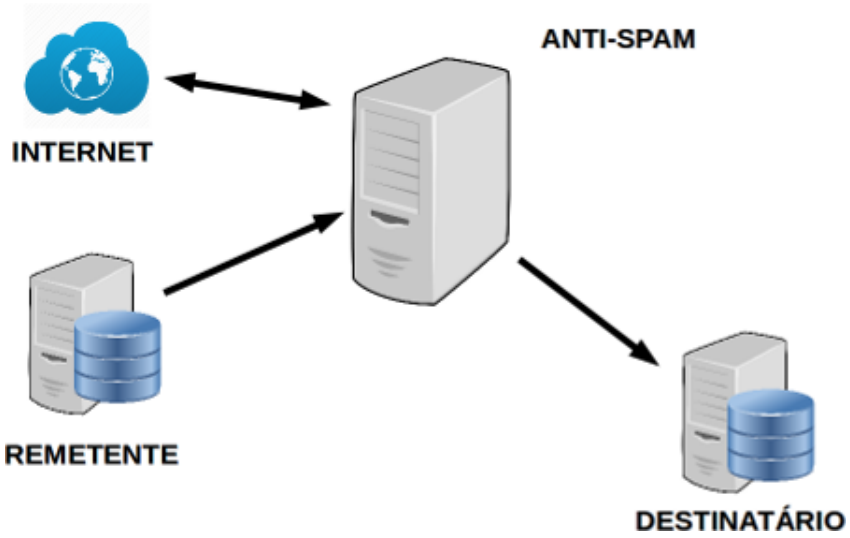

Figura 27 – Arquitetura empregada no experimento

A arquitetura é composta por três computadores. O primeiro, à esquerda, envia *e-mails* utilizando uma versão modificada do MTA-X descrito na seção 6.3. Nesta versão, o MTA-X também registra o horário de envio de cada *e-mail*, mas salva-o no campo *assunto* do *e-mail*, ao invés de salvá-lo em um arquivo. Este computador

possui um processador com dois *cores* de 1,6 GHz, 2 GB de memória RAM e executa o sistema operacional Linux Mint 17.2.

O segundo computador, ao centro, executa um dos dois *anti-spam*. Possui um processador com oito *cores* de 2,95 GHz e 4 GB de memória RAM. O CanIt é distribuído em um pacote. Ao instalar o pacote, instala-se uma versão personalizada do sistema operacional Debian 6, versão esta que contém o CanIt. Portanto, o CanIt executa sobre o Debian 6. O SASCA também executa sobre o Debian 6, instalado em outra partição do disco rígido. Desta forma, garante-se uma comparação justa entre os dois *anti-spam*, posto que ambos executam sobre uma mesma plataforma computacional.

O terceiro computador, à direita, executa o módulo de Armazenamento modificado, descrito na seção 6.3. O módulo recebe os *e-mails* classificados e registra, em uma mesma linha de um arquivo, tanto o horário em que foram recebidos quanto o horário em que foram enviados, horário este contido no campo *assunto* dos *e-mails*. A configuração de *hardware* deste computador é igual à do segundo. Executa, porém, o sistema operacional Ubuntu 16.04.

O terceiro computador executa, igualmente, um servidor NTP (*Network Time Protocol*). Os outros dois computadores sincronizam, através do servidor NTP, seus horários com o horário do terceiro computador. Assim, o servidor mantém iguais os horários dos três computadores, garantindo que os tempos de envio e recebimento dos *e-mails* sejam registrados com a maior precisão possível, dentro de uma faixa de erro de, aproximadamente, um milissegundo (DELAWARE, 2014).

Os computadores estão conectados através de um roteador, formando uma rede local. Esta rede está conectada à Internet, posto que o CanIt depende desta conexão para validar sua licença e para consultar listas negras.

A base utilizada no experimento foi, igualmente, a base UNIFEI. Como os *e-mails* da base consistem em *e-mails* processados pelo módulo Pré-Filtro, a ferramenta descrita na seção 5.2 foi usada para reconstruí-los.

Os *e-mails* foram agrupados em 6 conjuntos, de acordo com seus tamanhos. *E-mails* com tamanho entre `1K-2K` foram para o primeiro conjunto. *E-mails* com tamanho entre `2K-3K`, `3K-4K`, `4K-5K`, `10K-20K` e `20K-30K` foram, respectivamente, para o segundo, terceiro, quarto, quinto e sexto conjuntos. Através dos quatro primeiros conjuntos, avalia-se o tempo médio de processamento de *e-mails* cujos

tamanhos variam no intervalo de `1K`. Através dos dois últimos conjuntos, avalia-se o tempo médio de processamento de *e-mails* cujos tamanhos variam no intervalo de `10K`. Os demais *e-mails* da base não foram utilizados pois, com aqueles dos seis conjuntos, já foi possível avaliar os tempos médios de processamento dos dois *anti-spams*.

O MTA-X, executado no primeiro computador, fez uso de *threads*, de forma a permitir-lhe o envio simultâneo de *e-mails*. Quatro modos de envio foram avaliados. No primeiro modo, com o uso de apenas um *thread*, os *e-mails* foram enviados individualmente, um por vez. No segundo modo, com o uso de dois *threads*, os *e-mails* foram enviados, simultaneamente, de dois em dois. No terceiro modo, com o uso de quatro *threads*, os *e-mails* foram enviados, simultaneamente, de quatro em quatro. No quarto modo, com o uso de oito *threads*, os *e-mails* foram enviados, simultaneamente, de oito em oito. Os quatro modos de envio foram denominados modos T1, T2, T4 e T8, por fazerem uso de um, dois, quatro e oito *threads*, respectivamente.

O experimento produz, como resultado, o tempo gasto, por cada *anti-spam*, para receber, classificar e entregar os *e-mails* ao módulo Armazenamento modificado. Desta forma, pode-se avaliar não somente o impacto causado pelo tamanho do *e-mail* mas também quão rápido cada *anti-spam* processa os *e-mails*.

As figuras 28 e 29 apresentam, de forma gráfica, o tempo médio gasto, respectivamente, pelo CanIt e pelo SASCA, para receber, classificar e entregar um *e-mail* ao módulo Armazenamento modificado. As tabelas 7 e 8 apresentam, de forma numérica, estes mesmos tempos.

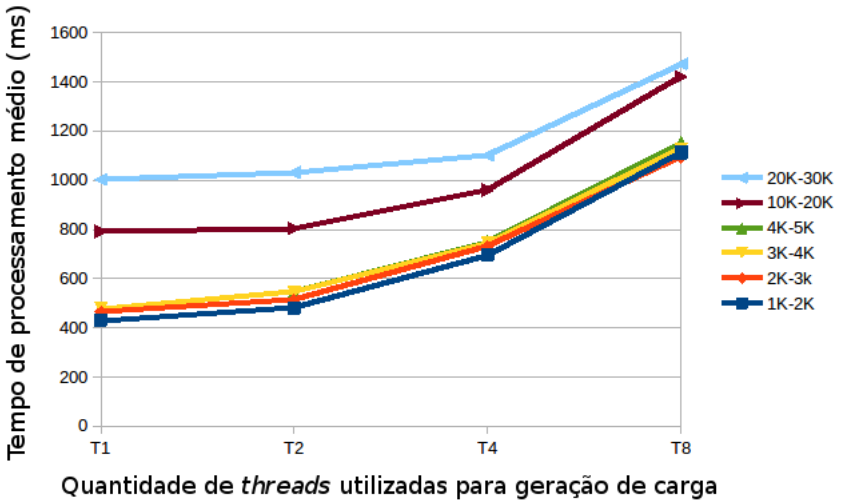

Figura 28 – Tempo médio gasto pelo CanIt para processar um *e-mail*

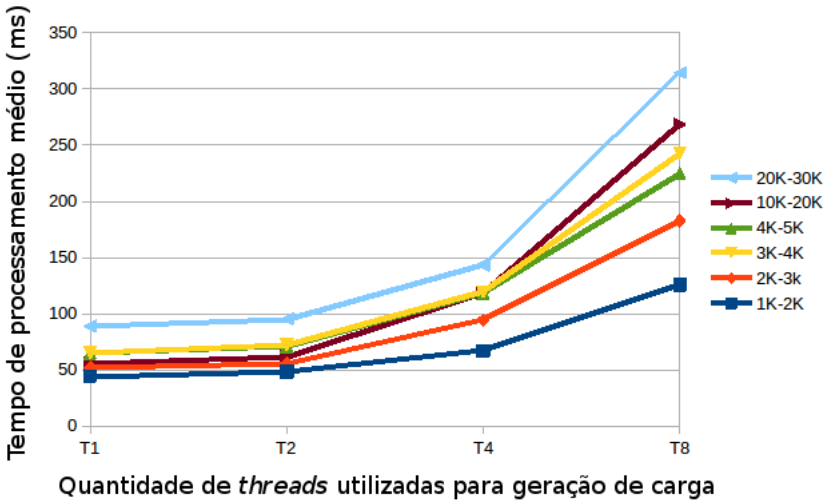

Figura 29 – Tempo médio gasto pelo SASCA para processar um *e-mail*

Tabela 7 – Tempo médio gasto pelo CanIt para processar um *e-mail*

	1K-2K	**2K-3K**	**3K-4K**	**4K-5K**	**10K-20K**	**20K-30K**
T1	430,16	464,29	479,40	477,99	791,79	1002,51
T2	480,59	512,65	545,63	545,21	804,39	1030,75
T4	696,43	730,72	742,14	747,52	960,75	1101,44
T8	1113,07	1095,01	1126,25	1152,39	1419,81	1473,50

Tabela 8 – Tempo médio gasto pelo SASCA para processar um *e-mail*

	1K-2K	**2K-3K**	**3K-4K**	**4K-5K**	**10K-20K**	**20K-30K**
T1	44,15	52,14	65,18	65,17	55,60	89,03
T2	48,31	55,99	72,46	71,26	61,73	95,46
T4	67,04	94,38	119,45	118,23	119,02	143,47
T8	125,85	182,88	242,16	224,97	268,48	314,30

Dos resultados apresentados nas figuras 28 e 29 e nas tabelas 7 e 8, pode-se observar que os dois *anti-spams* comportam-se da mesma forma com o aumento do número de *threads*. Quantos mais *e-mails* forem recebidos simultaneamente, maior será o tempo médio para seus processamentos. Isto ocorre devido ao fato de que alguns recursos, necessários para o processamento dos *e-mails*, não são compartilháveis. Por exemplo, as portas *socket* de recebimento dos *e-mails*, nos dois *anti-spam*, não são compartilháveis. Assim, apesar da quantidade de *threads* de que dispõe, o MTA-X só consegue enviar *e-mails* individualmente, um por vez, posto que, quando um de seus *threads* envia um *e-mail*, os demais precisarão aguardar a conclusão do envio para poderem disputar pela chance de alocar a porta *socket* de recebimento. Cogita-se, em uma versão futura, estudar as possibilidades de modificar o Postfix do SASCA para que possa fazer uso de *threads*.

Observa-se, outrossim, que o SASCA apresenta tempos médios de processamento até dez vezes menores que os do CanIt. Por exemplo, com o conjunto `1K-2K`, usando-se um *thread*, o SASCA gasta um tempo médio de 44,15 ms e o Canit, de 430,16 ms. Este comportamento permanece em todos os demais resultados.

Dos resultados, observa-se que, com o aumento do tamanho do *e-mail*, o tempo médio de processamento aumenta sensivelmente no CanIt. O mesmo comportamento, porém, não ocorre no SASCA. No CanIt, por exemplo, ao comparar-se o conjunto `1K-2K` com o `10K-20K`, usando-se um *thread* para o envio do *e-mail*, há um aumento de 361,63 ms no tempo médio de processamento. No SASCA, o aumento é de 11,45 ms.

O gráfico da figura 30 compara os tempos médios de processamento, gastos pelos dois *anti-spam*, sobre o conjunto `1K-2K`. As tabelas 9 e 10 comparam os tempos médios de processamento sobre todos os conjuntos.

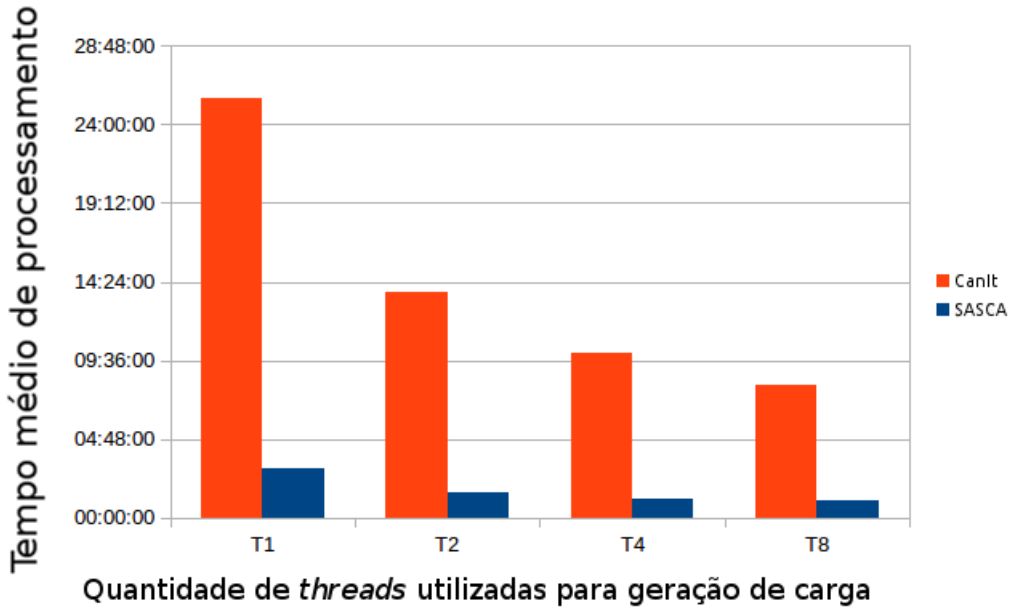

Figura 30 – Tempos médios de processamento, gastos pelos dois *anti-spam*, sobre o conjunto `1K-2K`

Tabela 9 – Tempos médios de processamento gastos pelo CanIt

	1K-2K	2K-3K	3K-4K	4K-5K	10K-20K	20K-30K
T1	25:37:03	14:25:37	09:41:07	09:33:33	07:57:52	02:32:30
T2	13:47:42	08:01:48	05:31:54	05:31:01	04:02:32	01:17:20
T4	10:00:52	05:46:24	03:47:49	03:47:26	02:24:45	00:40:21
T8	08:04:35	04:26:46	02:53:47	02:54:59	01:47:08	00:24:43

Tabela 10 – Tempos médios de processamento gastos pelo SASCA

	1K-2K	2K-3K	3K-4K	4K-5K	10K-20K	20K-30K
T1	03:02:47	02:01:25	01:31:36	01:32:28	00:56:26	00:15:05
T2	01:34:45	01:05:58	00:53:18	00:52:34	00:30:35	00:07:57
T4	01:09:04	00:54:15	00:44:55	00:44:51	00:25:00	00:05:22
T8	01:04:29	00:53:12	00:45:25	00:43:59	00:25:11	00:05:09

Os tempos médios do SASCA chegam a ser 22 horas e 34 minutos menores, usando-se um *thread* para envio de *e-mails*, e 7 horas menores, usando-se oito *threads*.

Devido ao fato do CanIt não possuir código aberto, não é possível determinar quais motivos o levam a gastar mais tempo para processar os *e-mails*.

Dos resultados, pode-se observar, igualmente, que, embora o tempo médio de processamento de cada *e-mail* aumente conforme o aumento no número de *threads*, o tempo médio de processamento do conjunto de *e-mails* diminui. Isto ocorre devido ao fato de que quando um *e-mail* está sendo processado pelo *anti-spam*, outro pode estar sendo recebido e outro ainda pode estar sendo enviado para o módulo Armazenamento modificado. A redução do tempo médio de processamento, porém, não é linear, pois, com o aumento no número de *threads*, observa-se, outrossim, o aumento na demanda simultânea pelos recursos não-compartilháveis.

6.5 AVALIAÇÃO DOS MODELOS CLASSIFICADORES

Dois modelos — MLP-WEKA e RF — foram avaliados. O modelo MLP-interno não foi incluído na avaliação por ter apresentado resultados de classificação inferiores aos do MLP-WEKA.

O experimento 3, descrito na seção 6.5.1, teve, por objetivo, avaliar os dois modelos classificadores — MLP-WEKA e RF — sobre a base UNIFEI. Por sua vez, o experimento 4, descrito na seção 6.5.2, teve, por objetivo, avaliar os dois modelos classificadores sobre a base UNIFEI-δ0. A acurácia nas classificações e os tempos de treinamento e de classificação dos modelos foram medidos.

Vários conjuntos de vetores foram criados a partir da base UNIFEI. Para criá-los, seguiu-se uma metodologia composta por dois passos. Primeiro, foram criados, através dos três métodos estatísticos de seleção de *tokens* — FD, MI e CHI2 —, conjuntos de vetores com oito distintas dimensionalidades — 8, 16, 32, 64, 128, 256, 512 e 1024. Segundo, as dimensionalidades dos vetores dos conjuntos foram reduzidas, através do uso da ferramenta descrita na seção 5.4. Os dois modelos foram avaliados sobre estes últimos conjuntos de vetores. A avaliação teve, por base, métricas de precisão, revocação, tempos de treinamento e de classificação.

Para o cálculo das métricas de precisão e de revocação, são necessárias as definições a seguir:

- N_{HAM} é a quantidade total de *hams* no conjunto de vetores testado;

- N_{SPAM} é a quantidade total de *spams* no conjunto de vetores testado;
- $n_{H \to H}$ é a quantidade de *hams* classificados corretamente;
- $n_{H \to S}$ é a quantidade de *hams* classificados como *spam*;
- $n_{S \to S}$ é a quantidade de *spams* classificados corretamente;
- $n_{S \to H}$ é a quantidade de *spams* classificados como *ham*.

As métricas de precisão medem a exatidão da classificação, dada pela quantidade de falsos-positivos obtida. Três métricas de precisão — P(HAM), P(SPAM) e P(GERAL) — foram usadas para avaliar os dois modelos. São calculadas pelas equações (6.1), (6.2) e (6.3).

$$P(HAM) = \frac{n_{H \to H}}{n_{H \to H} + n_{S \to H}} \tag{6.1}$$

$$P(SPAM) = \frac{n_{S \to S}}{n_{S \to S} + n_{H \to S}} \tag{6.2}$$

$$P(GERAL) = \frac{N_{HAM} * P(HAM) + N_{SPAM} * P(SPAM)}{N_{HAM} + N_{SPAM}} \tag{6.3}$$

As métricas de revocação medem a completude da classificação, dada pela quantidade de falsos-negativos obtida. Três métricas de revocação — R(HAM), R(SPAM) e R(GERAL) — foram usadas para avaliar os dois modelos. São calculadas pelas equações (6.4), (6.5) e (6.6).

$$R(HAM) = \frac{n_{H \to H}}{n_{H \to H} + n_{H \to S}} \tag{6.4}$$

$$R(SPAM) = \frac{n_{S \to S}}{n_{S \to S} + n_{S \to H}} \tag{6.5}$$

$$R(GERAL) = \frac{N_{HAM} * R(HAM) + N_{SPAM} * R(SPAM)}{N_{HAM} + N_{SPAM}} \tag{6.6}$$

Nos experimentos 3 e 4, cada resultado, bem como seu intervalo de confiança (GARDNER; ALTMAN, 1986), foi calculado a partir da média de dez execuções dos modelos. Os intervalos de confiança foram calculados com 5% de nível de confiança, ou seja, são válidos com 95% de certeza.

6.5.1 EXPERIMENTO 3

O objetivo deste experimento é avaliar os dois modelos classificadores — MLP-WEKA e RF — sobre a base UNIFEI. A avaliação utiliza, como métricas, a acurácia, em precisão e revocação, nas classificações e os tempos de treinamento e de classificação.

A metodologia empregada para a execução do experimento compõe-se de dois passos. Primeiro, escolhe-se o conjunto a ser testado. Por exemplo, o conjunto de vetores, de dimensionalidade 8, obtido por meio da execução do método estatístico de seleção de *tokens* FD sobre a base UNIFEI. Segundo, seleciona-se os vetores do conjunto que serão usados no treinamento e no teste do modelo.

Na avaliação do MLP-WEKA, 40% dos vetores do conjunto são usados no treinamento, 20% na validação do treinamento e os 40% restantes são usados no teste. Os conjuntos de treinamento e de validação não incluem vetores nulos. Todos os vetores nulos, porém, são incluídos no conjunto de teste. Na avaliação da RF, 50% dos vetores do conjunto, não incluindo vetores nulos, são usados no treinamento e os 50% restantes, incluindo vetores nulos, são usados no teste.

As tabelas 3, 4 e 5 da seção 6.2 apresentam a quantidade de vetores *ham* e *spam*, não nulos, de cada conjunto. A quantidade de vetores nulos em cada conjunto está descrita nas tabelas 3, 4 e 5 da seção 6.2.

O experimento foi realizado em um computador com processador de 4 *cores* de 3.20 GHz, 32GB de memória, executando o sistema operacional Linux Mint 18.3.

Tabela 11 – Quantidade de vetores em cada conjunto da base UNIFEI

Dimensão	Classe	FD	MI	CHI2
8	ham	353.365	301.234	4.119
	spam	495.487	431.983	7.230
16	ham	353.405	339.642	7.097
	spam	495.523	471.830	12.284
32	ham	353.416	339.736	14.236
	spam	495.523	471.916	19.555
64	ham	353.441	353.395	22.726
	spam	495.533	495.502	27.276
128	ham	353.448	353.427	35.670
	spam	495.538	495.502	57.246
256	ham	353.501	353.468	50.062
	spam	495.538	495.530	66.422
512	ham	353.582	353.503	108.055
	spam	495.566	495.530	202.507
1024	ham	353.624	353.503	208.224
	spam	495.727	495.530	324.412

As tabelas 12, 13, 14 e 15 apresentam, respectivamente, a precisão nas classificações, a revocação nas classificações, o tempo de treinamento e o tempo de classificação do modelo MLP-WEKA. De forma análoga, as tabelas 16, 17, 18 e 19 apresentam os valores obtidos pelo modelo RF. Nestas oito tabelas, NF e NF' indicam, respectivamente, a dimensionalidade dos vetores dos conjuntos antes e após a redução realizada pela ferramenta descrita na seção 5.4.

Tabela 12 – Precisão na classificação do modelo MLP-WEKA

			Precisão (MLP-WEKA)		
Método	NF	NF'	P(HAM)	P(SPAM)	P(GERAL)
CHI2	8	8	40,96 ± 0,00	96,08 ± 0,49	73,51 ± 0,29
CHI2	16	10	56,44 ± 17,83	79,80 ± 10,24	70,24 ± 1,32
CHI2	32	20	56,05 ± 17,39	77,18 ± 8,94	68,53 ± 1,89
CHI2	64	48	45,72 ± 10,87	81,45 ± 5,63	66,84 ± 1,19
CHI2	128	86	60,71 ± 18,34	74,84 ± 9,59	69,08 ± 1,83
CHI2	256	34	40,55 ± 0,01	71,03 ± 6,37	58,69 ± 3,79
CHI2	512	45	50,91 ± 14,43	63,59 ± 1,61	58,37 ± 4,99
CHI2	1024	17	53,35 ± 12,67	90,59 ± 12,60	74,52 ± 1,70
FD	8	5	46,88 ± 17,68	64,96 ± 5,69	55,93 ± 11,64
FD	16	3	58,68 ± 0,15	75,80 ± 0,17	67,25 ± 0,05
FD	32	6	60,41 ± 2,78	76,06 ± 4,38	68,24 ± 1,00
FD	64	22	71,30 ± 1,09	81,99 ± 2,14	76,65 ± 1,15
FD	128	65	75,48 ± 1,54	86,12 ± 2,00	80,81 ± 0,38
FD	256	51	75,00 ± 1,22	81,52 ± 0,84	78,26 ± 0,38
FD	512	101	75,39 ± 1,01	83,98 ± 1,21	79,69 ± 0,44
FD	1024	105	71,79 ± 5,48	80,53 ± 4,80	76,16 ± 0,74
MI	8	4	56,09 ± 1,28	76,23 ± 2,22	66,44 ± 0,53
MI	16	7	57,68 ± 0,13	77,99 ± 0,66	68,05 ± 0,39
MI	32	5	58,05 ± 0,12	77,10 ± 0,55	67,78 ± 0,30
MI	64	9	63,30 ± 0,73	81,42 ± 2,22	72,36 ± 0,77
MI	128	18	67,98 ± 5,83	81,58 ± 6,34	74,78 ± 0,41
MI	256	31	71,30 ± 0,94	84,95 ± 1,85	78,13 ± 0,50
MI	512	125	74,69 ± 1,50	85,59 ± 2,04	80,15 ± 0,37
MI	1024	208	75,43 ± 2,03	88,10 ± 2,45	81,78 ± 0,48

Tabela 13 – Revocação na classificação do modelo MLP-WEKA

			Revocação (MLP-WEKA)		
Método	NF	NF'	R(HAM)	R(SPAM)	R(GERAL)
CHI2	8	8	99,99 ± 0,00	0,09 ± 0,00	40,99 ± 0,00
CHI2	16	10	70,02 ± 34,50	30,09 ± 34,51	46,44 ± 6,26
CHI2	32	20	70,02 ± 34,48	30,11 ± 34,49	46,44 ± 6,26
CHI2	64	48	89,97 ± 22,56	10,18 ± 22,57	42,80 ± 4,12
CHI2	128	86	60,11 ± 36,78	40,16 ± 36,83	48,30 ± 6,79
CHI2	256	34	99,52 ± 0,12	0,72 ± 0,14	40,73 ± 0,03
CHI2	512	45	77,89 ± 28,62	23,16 ± 28,91	45,67 ± 5,25
CHI2	1024	17	80,19 ± 29,72	25,71 ± 27,95	49,22 ± 3,07
FD	8	5	64,00 ± 24,19	54,86 ± 17,15	59,42 ± 3,55
FD	16	3	87,72 ± 0,30	38,37 ± 0,60	63,01 ± 0,15
FD	32	6	84,41 ± 9,71	43,22 ± 10,05	63,79 ± 0,69
FD	64	22	85,36 ± 2,45	65,67 ± 2,29	75,50 ± 0,99
FD	128	65	88,27 ± 2,38	71,22 ± 3,17	79,73 ± 0,57
FD	256	51	83,53 ± 1,43	72,14 ± 2,27	77,83 ± 0,56
FD	512	101	86,17 ± 1,54	71,89 ± 1,92	79,02 ± 0,47
FD	1024	105	82,23 ± 10,12	65,31 ± 8,42	73,76 ± 0,97
MI	8	4	88,15 ± 4,39	34,32 ± 5,83	60,49 ± 0,86
MI	16	7	88,93 ± 0,44	37,52 ± 0,29	62,67 ± 0,21
MI	32	5	87,80 ± 0,51	39,27 ± 0,44	63,00 ± 0,16
MI	64	9	88,65 ± 2,62	48,53 ± 3,03	68,58 ± 0,29
MI	128	18	83,74 ± 14,64	56,84 ± 10,41	70,28 ± 2,26
MI	256	31	88,38 ± 1,95	64,43 ± 2,45	76,39 ± 0,37
MI	512	125	87,93 ± 2,50	70,11 ± 3,13	79,00 ± 0,36
MI	1024	208	90,14 ± 3,00	70,42 ± 3,95	80,26 ± 0,76

Tabela 14 – Tempo de treinamento do modelo MLP-WEKA

Método	NF	NF'	Tempo de treinamento (MLP-WEKA)
CHI2	8	8	00:00:00.173 ± 00:00:00.142
CHI2	16	10	00:00:04.203 ± 00:00:02.715
CHI2	32	20	00:00:07.150 ± 00:00:06.049
CHI2	64	48	00:00:43.870 ± 00:00:29.221
CHI2	128	86	00:01:34.197 ± 00:00:30.596
CHI2	256	34	00:04:00.943 ± 00:02:56.879
CHI2	512	45	00:03:51.106 ± 00:01:27.896
CHI2	1024	17	00:14:20.381 ± 00:15:28.216
FD	8	5	00:00:57.230 ± 00:00:19.496
FD	16	3	00:07:36.711 ± 00:07:08.330
FD	32	6	00:01:39.661 ± 00:01:20.227
FD	64	22	00:05:53.577 ± 00:01:48.272
FD	128	65	00:19:09.338 ± 00:03:33.137
FD	256	51	00:15:13.454 ± 00:03:37.718
FD	512	101	00:59:25.592 ± 00:21:14.650
FD	1024	105	00:38:45.791 ± 00:02:27.502
MI	8	4	00:12:45.004 ± 00:07:36.121
MI	16	7	00:02:00.906 ± 00:01:08.805
MI	32	5	00:01:59.880 ± 00:01:03.761
MI	64	9	00:03:39.381 ± 00:01:42.241
MI	128	18	00:05:11.680 ± 00:01:27.164
MI	256	31	00:11:24.748 ± 00:02:38.636
MI	512	125	00:50:57.477 ± 00:02:52.817
MI	1024	208	02:11:27.315 ± 00:08:38.977

Tabela 15 – Tempo de classificação do modelo MLP-WEKA

Método	NF	NF'	Tempo de classificação (MLP-WEKA)
CHI2	8	8	00:00:00.990 ± 00:00:00.005
CHI2	16	10	00:00:01.206 ± 00:00:00.023
CHI2	32	20	00:00:02.281 ± 00:00:00.035
CHI2	64	48	00:00:08.344 ± 00:00:00.250
CHI2	128	86	00:00:23.218 ± 00:00:00.046
CHI2	256	34	00:00:04.971 ± 00:00:00.017
CHI2	512	45	00:00:07.140 ± 00:00:00.034
CHI2	1024	17	00:00:01.469 ± 00:00:00.022
FD	8	5	00:00:00.438 ± 00:00:00.014
FD	16	3	00:00:00.383 ± 00:00:00.022
FD	32	6	00:00:00.539 ± 00:00:00.017
FD	64	22	00:00:01.631 ± 00:00:00.033
FD	128	65	00:00:08.263 ± 00:00:00.076
FD	256	51	00:00:05.505 ± 00:00:00.070
FD	512	101	00:00:18.106 ± 00:00:00.108
FD	1024	105	00:00:19.570 ± 00:00:00.071
MI	8	4	00:00:00.444 ± 00:00:00.029
MI	16	7	00:00:00.565 ± 00:00:00.018
MI	32	5	00:00:00.454 ± 00:00:00.016
MI	64	9	00:00:00.625 ± 00:00:00.023
MI	128	18	00:00:01.292 ± 00:00:00.028
MI	256	31	00:00:02.579 ± 00:00:00.030
MI	512	125	00:00:27.160 ± 00:00:00.017
MI	1024	208	00:01:11.920 ± 00:00:00.065

Tabela 16 – Precisão na classificação do modelo RF

			Precisão (RF)		
Método	NF	NF'	P(HAM)	P(SPAM)	P(GERAL)
CHI2	8	8	40,96 ± 0,00	97,10 ± 0,39	74,12 ± 0,23
CHI2	16	10	40,96 ± 0,00	88,57 ± 0,70	69,08 ± 0,41
CHI2	32	20	40,95 ± 0,00	84,99 ± 0,75	66,97 ± 0,45
CHI2	64	48	40,92 ± 0,00	84,19 ± 0,71	66,50 ± 0,42
CHI2	128	86	40,85 ± 0,00	85,70 ± 0,39	67,40 ± 0,23
CHI2	256	34	40,56 ± 0,00	68,39 ± 0,35	57,12 ± 0,21
CHI2	512	45	41,35 ± 0,01	64,88 ± 0,10	55,20 ± 0,06
CHI2	1024	17	44,98 ± 0,02	98,94 ± 0,05	75,66 ± 0,03
FD	8	5	90,94 ± 0,06	88,91 ± 0,05	89,92 ± 0,03
FD	16	3	63,39 ± 0,02	92,63 ± 0,08	78,03 ± 0,04
FD	32	6	80,13 ± 0,05	90,26 ± 0,11	85,20 ± 0,04
FD	64	22	91,03 ± 0,04	96,75 ± 0,03	93,89 ± 0,02
FD	128	65	91,78 ± 0,03	97,41 ± 0,02	94,60 ± 0,01
FD	256	51	88,99 ± 0,06	96,69 ± 0,07	92,85 ± 0,02
FD	512	101	90,66 ± 0,03	97,12 ± 0,02	93,89 ± 0,02
FD	1024	105	85,81 ± 0,04	97,03 ± 0,04	91,43 ± 0,02
MI	8	4	62,93 ± 0,43	71,81 ± 0,24	67,49 ± 0,33
MI	16	7	60,46 ± 0,02	94,21 ± 0,10	77,70 ± 0,06
MI	32	5	60,58 ± 0,03	94,90 ± 0,07	78,12 ± 0,04
MI	64	9	74,98 ± 0,07	90,05 ± 0,12	82,52 ± 0,04
MI	128	18	70,40 ± 0,04	95,38 ± 0,03	82,90 ± 0,02
MI	256	31	82,03 ± 0,07	94,71 ± 0,06	88,38 ± 0,03
MI	512	125	88,73 ± 0,02	97,12 ± 0,03	92,93 ± 0,02
MI	1024	208	90,44 ± 0,04	96,94 ± 0,04	93,70 ± 0,02

Tabela 17 – Revocação na classificação do modelo RF

			Revocação (RF)		
Método	NF	NF'	P(HAM)	P(SPAM)	P(GERAL)
CHI2	8	8	100,00 ± 0,00	0,09 ± 0,00	40,99 ± 0,00
CHI2	16	10	99,98 ± 0,00	0,13 ± 0,00	41,00 ± 0,00
CHI2	32	20	99,96 ± 0,00	0,17 ± 0,00	41,01 ± 0,00
CHI2	64	48	99,94 ± 0,00	0,21 ± 0,00	40,98 ± 0,00
CHI2	128	86	99,93 ± 0,00	0,28 ± 0,00	40,94 ± 0,00
CHI2	256	34	99,46 ± 0,01	0,79 ± 0,01	40,75 ± 0,00
CHI2	512	45	96,88 ± 0,01	4,02 ± 0,01	42,21 ± 0,00
CHI2	1024	17	99,90 ± 0,00	7,26 ± 0,02	47,23 ± 0,02
FD	8	5	88,60 ± 0,06	91,20 ± 0,07	89,90 ± 0,03
FD	16	3	96,45 ± 0,05	44,43 ± 0,06	70,41 ± 0,02
FD	32	6	91,63 ± 0,11	77,33 ± 0,09	84,47 ± 0,02
FD	64	22	96,95 ± 0,03	90,47 ± 0,04	93,71 ± 0,02
FD	128	65	97,57 ± 0,02	91,29 ± 0,02	94,42 ± 0,01
FD	256	51	96,98 ± 0,07	88,04 ± 0,08	92,50 ± 0,02
FD	512	101	97,32 ± 0,02	90,00 ± 0,03	93,66 ± 0,02
FD	1024	105	97,42 ± 0,03	83,94 ± 0,05	90,67 ± 0,03
MI	8	4	76,12 ± 0,04	57,57 ± 0,76	66,59 ± 0,38
MI	16	7	97,50 ± 0,05	38,94 ± 0,05	67,58 ± 0,03
MI	32	5	97,80 ± 0,03	39,11 ± 0,05	67,81 ± 0,02
MI	64	9	92,34 ± 0,11	69,21 ± 0,13	80,77 ± 0,03
MI	128	18	97,13 ± 0,02	59,21 ± 0,05	78,16 ± 0,03
MI	256	31	95,57 ± 0,06	79,13 ± 0,08	87,34 ± 0,03
MI	512	125	97,40 ± 0,03	87,67 ± 0,04	92,52 ± 0,02
MI	1024	208	97,16 ± 0,04	89,77 ± 0,06	93,46 ± 0,02

Tabela 18 – Tempo de treinamento do modelo RF

Método	NF	NF'	Tempo de treinamento (RF)
CHI2	8	8	00:00:00.028 ± 00:00:00.019
CHI2	16	10	00:00:00.071 ± 00:00:00.004
CHI2	32	20	00:00:00.165 ± 00:00:00.003
CHI2	64	48	00:00:00.531 ± 00:00:00.002
CHI2	128	86	00:00:01.693 ± 00:00:00.017
CHI2	256	34	00:00:01.058 ± 00:00:00.039
CHI2	512	45	00:00:07.205 ± 00:00:00.085
CHI2	1024	17	00:00:16.567 ± 00:00:00.538
FD	8	5	00:00:30.808 ± 00:00:00.333
FD	16	3	00:00:29.647 ± 00:00:00.528
FD	32	6	00:00:36.723 ± 00:00:00.455
FD	64	22	00:01:09.291 ± 00:00:00.677
FD	128	65	00:02:39.321 ± 00:00:00.865
FD	256	51	00:03:12.799 ± 00:00:01.835
FD	512	101	00:04:29.471 ± 00:00:01.929
FD	1024	105	00:07:26.325 ± 00:00:02.502
MI	8	4	00:00:17.138 ± 00:00:00.728
MI	16	7	00:00:34.198 ± 00:00:00.842
MI	32	5	00:00:18.094 ± 00:00:00.845
MI	64	9	00:00:52.094 ± 00:00:00.871
MI	128	18	00:01:14.909 ± 00:00:01.037
MI	256	31	00:02:35.688 ± 00:00:01.113
MI	512	125	00:08:00.978 ± 00:00:02.845
MI	1024	208	00:11:46.847 ± 00:00:03.511

Tabela 19 – Tempo de classificação do modelo RF

Método	NF	NF'	Tempo de classificação (RF)
CHI2	8	8	00:00:05.342 ± 00:00:00.285
CHI2	16	10	00:00:09.753 ± 00:00:00.234
CHI2	32	20	00:00:30.825 ± 00:00:00.644
CHI2	64	48	00:01:13.794 ± 00:00:07.132
CHI2	128	86	00:02:19.578 ± 00:00:14.123
CHI2	256	34	00:01:09.041 ± 00:00:01.815
CHI2	512	45	00:01:22.547 ± 00:00:03.037
CHI2	1024	17	00:00:26.439 ± 00:00:02.393
FD	8	5	00:00:42.935 ± 00:00:03.355
FD	16	3	00:00:27.233 ± 00:00:01.045
FD	32	6	00:00:46.636 ± 00:00:03.520
FD	64	22	00:00:50.898 ± 00:00:01.471
FD	128	65	00:00:58.227 ± 00:00:02.559
FD	256	51	00:01:25.623 ± 00:00:03.702
FD	512	101	00:01:26.540 ± 00:00:01.037
FD	1024	105	00:01:57.638 ± 00:00:02.656
MI	8	4	00:00:19.674 ± 00:00:01.366
MI	16	7	00:00:29.684 ± 00:00:02.517
MI	32	5	00:00:19.125 ± 00:00:00.904
MI	64	9	00:00:44.231 ± 00:00:04.190
MI	128	18	00:00:42.425 ± 00:00:01.413
MI	256	31	00:01:05.949 ± 00:00:02.598
MI	512	125	00:01:49.900 ± 00:00:02.148
MI	1024	208	00:02:20.178 ± 00:00:02.078

Da análise dos resultados do modelo MLP-WEKA, pode-se concluir o seguinte:

- Conjuntos criados com o método CHI2 apresentaram os piores resultados de precisão e revocação. Estes resultados, de certa forma, eram previstos, uma vez que estes conjuntos possuem uma quantidade muito grande (e.g., 98,84%) de vetores nulos;
- O melhor resultado — 81,78% de precisão e 80,26% de revocação — foi obtido com o conjunto criado pelo método MI, com vetores de dimensionalidade 1024 reduzida para 208. O segundo melhor resultado — 80,81% de precisão e 79,73% de revocação — foi obtido com o conjunto criado pelo método FD, com vetores de dimensionalidade 128 reduzida para 65;

- Em geral, os resultados — precisão e revocação — obtidos com conjuntos criados pelos métodos FD e MI são próximos.

Da análise dos resultados do modelo RF, pode-se concluir o seguinte:

- Conjuntos criados com o método CHI2 apresentaram os piores resultados de precisão e revocação, devido ao fato de possuírem quantidades muito grandes de vetores nulos;
- O melhor resultado — 94,60% de precisão e 94,42% de revocação — foi obtido com o conjunto criado pelo método FD, com vetores de dimensionalidade 128 reduzida para 65. O segundo melhor resultado — 93,89% de precisão e 93,71% de revocação — foi obtido com o conjunto criado pelo método FD, com vetores de dimensionalidade 64 reduzida para 22;
- Em geral, os resultados — precisão e revocação — obtidos com conjuntos criados pelos métodos FD e MI são próximos.

Da comparação dos resultados do modelo MLP-WEKA com os do modelo RF, pode-se concluir o seguinte:

- O maior tempo de treinamento da RF — 11 minutos e 46 segundos — foi com um conjunto criado com o método MI, com vetores de dimensionalidade 1024. Este tempo, porém, é consideravelmente baixo, se comparado com o maior tempo de treinamento do MLP-WEKA — 2 horas, 11 minutos e 27 segundos. A implementação do modelo RF no WEKA é *multithread* e, portanto, quanto mais *cores* o processador do computador possuir, mais rápido ele irá executar. A implementação do MLP-WEKA não é *multithread*;
- O MLP-WEKA apresentou os melhores tempos de classificação. Seu pior resultado foi de 1 minuto e 11 segundos. O pior tempo de classificação da RF, por sua vez, foi de 2 minutos e 20 segundos;
- Portanto, conclui-se que o melhor modelo classificador é o RF.

6.5.2 EXPERIMENTO 4

O objetivo deste experimento é avaliar os dois modelos classificadores — MLP-WEKA e RF — sobre a base UNIFEI-δ0. A avaliação utiliza, como métricas, a acurácia, em precisão e revocação, nas classificações e os tempos de treinamento e de classificação.

A metodologia empregada para a execução do experimento, para a criação dos conjuntos de vetores da base UNIFEI-δ0 e para a seleção dos vetores dos conjuntos de treinamento, validação e teste é a mesma empregada no experimento 3 (seção 6.5.1).

As tabelas 3, 4 e 5 da seção 6.2 apresentam a quantidade de vetores *ham* e *spam*, não nulos, de cada conjunto. A quantidade de vetores nulos em cada conjunto está descrita nas tabelas 3, 4 e 5 da seção 6.2.

Tabela 20 – Quantidade de vetores em cada conjunto da base UNIFEI-δ0

Dimensão	Classe	FD	MI	CHI2
8	ham	347,446	102,806	3,455
	spam	501,406	630,411	7,894
16	ham	349,701	323,961	5,478
	spam	499,227	487,511	13,903
32	ham	355,024	319,521	14,829
	spam	493,915	492,131	18,962
64	ham	355,019	347,165	23,626
	spam	493,955	501,732	26,376
128	ham	355,284	356,422	27,537
	spam	493,702	492,507	65,379
256	ham	354,859	354,027	45,597
	spam	494,180	494,971	70,887
512	ham	354,579	354,041	95,260
	spam	494,569	494,992	215,302
1024	ham	353,868	353,906	201,855
	spam	495,483	495,127	330,781

O experimento foi realizado no mesmo computador usado no experimento 3 (seção 6.5.1), ou seja, em um computador com processador de 4 *cores* de 3.20 GHz, 32GB de memória, executando o sistema operacional Linux Mint 18.3.

As tabelas 21, 22, 23 e 24 apresentam, respectivamente, a precisão nas classificações, a revocação nas classificações, o tempo de treinamento e o tempo de classificação do modelo MLP-WEKA. De forma análoga, as tabelas 25, 26, 27 e 28 apresentam os valores obtidos pelo modelo RF. Nestas oito tabelas, NF e NF' indicam, respectivamente, a dimensionalidade dos vetores dos conjuntos antes e após a redução realizada pela ferramenta descrita na seção 5.4.

Tabela 21 – Precisão na classificação do modelo MLP-WEKA

			Precisão (MLP-WEKA)		
Método	NF	NF'	P(HAM)	P(SPAM)	P(GERAL)
CHI2	8	4	99,85 ± 0,05	59,07 ± 0,00	75,88 ± 0,02
CHI2	16	16	41,65 ± 0,00	99,18 ± 0,05	75,40 ± 0,03
CHI2	32	28	99,55 ± 0,10	59,46 ± 0,00	75,98 ± 0,04
CHI2	64	42	41,65 ± 0,00	98,86 ± 0,08	75,39 ± 0,05
CHI2	128	69	98,68 ± 0,07	59,92 ± 0,03	76,29 ± 0,05
CHI2	256	116	64,85 ± 20,15	81,81 ± 13,18	74,74 ± 0,74
CHI2	512	121	59,80 ± 13,57	87,15 ± 11,07	74,38 ± 0,47
CHI2	1024	152	69,08 ± 14,32	77,60 ± 12,76	73,52 ± 0,20
FD	8	5	60,36 ± 0,34	73,94 ± 2,20	67,21 ± 0,97
FD	16	3	51,97 ± 13,06	76,30 ± 6,58	64,25 ± 9,76
FD	32	8	64,24 ± 4,88	73,70 ± 5,04	69,03 ± 1,37
FD	64	11	65,15 ± 1,06	78,28 ± 1,05	71,80 ± 0,82
FD	128	24	75,31 ± 1,41	81,39 ± 1,11	78,39 ± 0,64
FD	256	60	73,74 ± 0,66	85,35 ± 1,38	79,62 ± 0,38
FD	512	79	75,24 ± 2,69	83,16 ± 1,78	79,25 ± 0,84
FD	1024	77	71,64 ± 1,51	83,78 ± 1,71	77,78 ± 0,35
MI	8	3	80,57 ± 3,37	77,63 ± 3,80	79,12 ± 0,21
MI	16	3	69,87 ± 0,05	79,75 ± 0,07	74,93 ± 0,04
MI	32	7	76,78 ± 2,84	77,94 ± 0,23	77,37 ± 1,35
MI	64	10	66,97 ± 6,19	78,34 ± 6,00	72,70 ± 0,35
MI	128	30	70,28 ± 0,63	84,25 ± 2,44	77,36 ± 1,09
MI	256	49	74,34 ± 3,88	81,48 ± 3,76	77,96 ± 0,62
MI	512	118	73,06 ± 1,19	85,89 ± 1,57	79,56 ± 0,34
MI	1024	155	77,22 ± 2,90	85,63 ± 1,70	81,48 ± 0,71

Tabela 22 – Revocação na classificação do modelo MLP-WEKA

			Revocação (MLP-WEKA)		
Método	NF	NF'	R(HAM)	R(SPAM)	R(GERAL)
CHI2	8	4	99,85 ± 0,05	59,07 ± 0,00	75,88 ± 0,02
CHI2	16	16	99,98 ± 0,00	1,32 ± 0,01	42,09 ± 0,00
CHI2	32	28	2,70 ± 0,01	99,99 ± 0,00	59,90 ± 0,00
CHI2	64	42	99,96 ± 0,00	2,57 ± 0,01	42,53 ± 0,00
CHI2	128	69	8,56 ± 0,12	99,92 ± 0,00	61,34 ± 0,05
CHI2	256	116	63,13 ± 33,74	43,41 ± 34,75	51,63 ± 6,22
CHI2	512	121	82,61 ± 24,62	32,68 ± 25,13	56,00 ± 1,89
CHI2	1024	152	65,92 ± 29,35	52,90 ± 28,14	59,14 ± 0,67
FD	8	5	83,12 ± 2,93	46,28 ± 2,61	64,55 ± 0,26
FD	16	3	81,52 ± 20,51	41,48 ± 14,72	61,31 ± 2,72
FD	32	8	76,92 ± 11,79	54,74 ± 15,86	65,69 ± 3,55
FD	64	11	83,96 ± 1,23	56,19 ± 2,39	69,89 ± 0,99
FD	128	24	82,67 ± 1,75	73,48 ± 2,36	78,02 ± 0,72
FD	256	60	87,64 ± 1,63	69,56 ± 1,64	78,48 ± 0,11
FD	512	79	84,67 ± 2,69	72,43 ± 4,92	78,47 ± 1,36
FD	1024	77	86,63 ± 2,59	66,33 ± 3,15	76,36 ± 0,45
MI	8	3	76,47 ± 6,60	80,13 ± 5,75	78,28 ± 0,52
MI	16	3	82,41 ± 0,07	66,11 ± 0,06	74,06 ± 0,03
MI	32	7	77,18 ± 1,24	77,29 ± 3,59	77,24 ± 1,25
MI	64	10	80,45 ± 16,01	56,73 ± 10,53	68,49 ± 2,65
MI	128	30	87,47 ± 2,96	63,97 ± 2,01	75,55 ± 0,68
MI	256	49	82,35 ± 7,36	71,07 ± 6,28	76,64 ± 1,09
MI	512	118	88,38 ± 1,84	68,13 ± 2,58	78,12 ± 0,49
MI	1024	155	86,92 ± 2,42	74,49 ± 5,31	80,63 ± 1,58

Tabela 23 – Tempo de treinamento do modelo MLP-WEKA

Método	NF	NF'	Tempo de treinamento (MLP-WEKA)
CHI2	8	4	00:00:02.311 ± 00:00:00.621
CHI2	16	16	00:00:05.887 ± 00:00:04.722
CHI2	32	28	00:00:26.725 ± 00:00:16.402
CHI2	64	42	00:01:31.297 ± 00:00:29.129
CHI2	128	69	00:05:23.780 ± 00:04:34.159
CHI2	256	116	00:09:56.157 ± 00:01:00.782
CHI2	512	121	00:53:22.908 ± 00:46:01.910
CHI2	1024	152	00:59:18.567 ± 00:09:41.915
FD	8	5	00:00:52.779 ± 00:00:37.226
FD	16	3	00:01:06.080 ± 00:00:30.760
FD	32	8	00:01:29.640 ± 00:00:45.851
FD	64	11	00:02:52.848 ± 00:01:11.867
FD	128	24	00:08:22.381 ± 00:02:44.154
FD	256	60	00:21:29.634 ± 00:05:45.896
FD	512	79	00:27:28.849 ± 00:04:23.578
FD	1024	77	00:26:09.283 ± 00:03:28.547
MI	8	3	00:00:58.559 ± 00:00:33.555
MI	16	3	00:00:43.803 ± 00:00:21.798
MI	32	7	00:01:06.504 ± 00:00:17.764
MI	64	10	00:02:05.908 ± 00:00:50.601
MI	128	30	00:13:21.549 ± 00:03:25.748
MI	256	49	00:17:22.840 ± 00:02:22.171
MI	512	118	00:54:48.178 ± 00:14:06.396
MI	1024	155	01:43:32.567 ± 00:09:52.741

Tabela 24 – Tempo de classificação do modelo MLP-WEKA

Método	NF	NF'	Tempo de classificação (MLP-WEKA)
CHI2	8	4	00:00:00.658 ± 00:00:00.010
CHI2	16	16	00:00:01.836 ± 00:00:00.024
CHI2	32	28	00:00:03.734 ± 00:00:00.055
CHI2	64	42	00:00:06.858 ± 00:00:00.075
CHI2	128	69	00:00:15.036 ± 00:00:00.175
CHI2	256	116	00:00:37.625 ± 00:00:00.114
CHI2	512	121	00:00:37.977 ± 00:00:00.190
CHI2	1024	152	00:00:50.665 ± 00:00:00.062
FD	8	5	00:00:00.453 ± 00:00:00.031
FD	16	3	00:00:00.379 ± 00:00:00.001
FD	32	8	00:00:00.598 ± 00:00:00.019
FD	64	11	00:00:00.742 ± 00:00:00.018
FD	128	24	00:00:01.782 ± 00:00:00.022
FD	256	60	00:00:07.205 ± 00:00:00.096
FD	512	79	00:00:11.703 ± 00:00:00.146
FD	1024	77	00:00:11.120 ± 00:00:00.189
MI	8	3	00:00:00.572 ± 00:00:00.045
MI	16	3	00:00:00.383 ± 00:00:00.035
MI	32	7	00:00:00.545 ± 00:00:00.027
MI	64	10	00:00:00.762 ± 00:00:00.037
MI	128	30	00:00:02.500 ± 00:00:00.039
MI	256	49	00:00:05.198 ± 00:00:00.190
MI	512	118	00:00:24.309 ± 00:00:00.070
MI	1024	155	00:00:40.818 ± 00:00:00.661

Tabela 25 – Precisão na classificação do modelo RF

			Precisão (RF)		
Método	NF	NF'	P(HAM)	P(SPAM)	P(GERAL)
CHI2	8	4	99,90 ± 0,03	59,07 ± 0,00	75,90 ± 0,01
CHI2	16	16	99,90 ± 0,04	59,17 ± 0,00	76,00 ± 0,02
CHI2	32	28	99,88 ± 0,04	59,46 ± 0,00	76,11 ± 0,02
CHI2	64	42	41,66 ± 0,00	99,27 ± 0,05	75,63 ± 0,03
CHI2	128	69	98,91 ± 0,02	59,97 ± 0,00	76,41 ± 0,01
CHI2	256	116	98,32 ± 0,05	60,54 ± 0,00	76,27 ± 0,02
CHI2	512	121	50,98 ± 0,01	95,84 ± 0,06	74,89 ± 0,03
CHI2	1024	152	54,17 ± 0,02	92,26 ± 0,06	74,06 ± 0,04
FD	8	5	97,79 ± 0,01	98,47 ± 0,02	98,13 ± 0,01
FD	16	3	70,30 ± 0,07	89,24 ± 0,17	79,86 ± 0,07
FD	32	8	88,07 ± 0,07	94,22 ± 0,10	91,18 ± 0,02
FD	64	11	75,85 ± 0,03	94,17 ± 0,10	85,13 ± 0,04
FD	128	24	86,40 ± 0,02	97,53 ± 0,04	92,04 ± 0,02
FD	256	60	90,66 ± 0,03	98,36 ± 0,03	94,56 ± 0,01
FD	512	79	90,68 ± 0,03	98,48 ± 0,02	94,64 ± 0,02
FD	1024	77	89,14 ± 0,05	97,20 ± 0,07	93,22 ± 0,02
MI	8	3	96,77 ± 0,02	72,16 ± 0,03	84,63 ± 0,02
MI	16	3	94,42 ± 0,05	89,18 ± 0,03	91,74 ± 0,02
MI	32	7	97,01 ± 0,02	90,89 ± 0,02	93,89 ± 0,01
MI	64	10	68,00 ± 0,05	96,78 ± 0,04	82,51 ± 0,02
MI	128	30	85,74 ± 0,07	94,75 ± 0,12	90,31 ± 0,04
MI	256	49	86,10 ± 0,04	97,23 ± 0,04	91,73 ± 0,01
MI	512	118	90,38 ± 0,03	98,09 ± 0,02	94,28 ± 0,01
MI	1024	155	89,90 ± 0,03	98,50 ± 0,03	94,25 ± 0,01

Tabela 26 – Revocação na classificação do modelo RF

			Revocação (RF)		
Método	NF	NF'	R(HAM)	R(SPAM)	R(GERAL)
CHI2	8	4	1,16 ± 0,01	100,00 ± 0,00	59,27 ± 0,00
CHI2	16	16	2,05 ± 0,01	100,00 ± 0,00	59,52 ± 0,00
CHI2	32	28	2,70 ± 0,01	100,00 ± 0,00	59,91 ± 0,00
CHI2	64	42	99,97 ± 0,00	2,58 ± 0,01	42,54 ± 0,00
CHI2	128	69	8,72 ± 0,02	99,93 ± 0,00	61,42 ± 0,00
CHI2	256	116	8,77 ± 0,02	99,89 ± 0,00	61,94 ± 0,00
CHI2	512	121	99,19 ± 0,01	16,40 ± 0,02	55,07 ± 0,01
CHI2	1024	152	97,78 ± 0,02	24,27 ± 0,04	59,40 ± 0,02
FD	8	5	98,45 ± 0,02	97,81 ± 0,01	98,13 ± 0,01
FD	16	3	92,41 ± 0,15	61,70 ± 0,15	76,91 ± 0,04
FD	32	8	94,49 ± 0,11	87,52 ± 0,09	90,96 ± 0,01
FD	64	11	95,52 ± 0,08	70,37 ± 0,06	82,78 ± 0,03
FD	128	24	97,79 ± 0,03	85,00 ± 0,04	91,31 ± 0,02
FD	256	60	98,46 ± 0,03	90,12 ± 0,03	94,23 ± 0,01
FD	512	79	98,57 ± 0,02	90,14 ± 0,03	94,30 ± 0,02
FD	1024	77	97,39 ± 0,06	88,42 ± 0,06	92,85 ± 0,01
MI	8	3	63,25 ± 0,05	97,83 ± 0,01	80,31 ± 0,03
MI	16	3	87,91 ± 0,04	95,05 ± 0,05	91,56 ± 0,02
MI	32	7	89,83 ± 0,03	97,34 ± 0,02	93,66 ± 0,01
MI	64	10	98,15 ± 0,03	54,60 ± 0,14	76,19 ± 0,05
MI	128	30	95,18 ± 0,12	84,60 ± 0,11	89,82 ± 0,03
MI	256	49	97,53 ± 0,03	84,65 ± 0,04	91,01 ± 0,01
MI	512	118	98,21 ± 0,02	89,81 ± 0,03	93,95 ± 0,01
MI	1024	155	98,60 ± 0,03	89,20 ± 0,03	93,84 ± 0,01

Tabela 27 – Tempo de treinamento do modelo RF

Método	NF	NF'	Tempo de treinamento (RF)
CHI2	8	4	00:00:00.067 ± 00:00:00.045
CHI2	16	16	00:00:00.232 ± 00:00:00.002
CHI2	32	28	00:00:00.640 ± 00:00:00.014
CHI2	64	42	00:00:02.126 ± 00:00:00.015
CHI2	128	69	00:00:11.972 ± 00:00:00.052
CHI2	256	116	00:00:28.367 ± 00:00:00.166
CHI2	512	121	00:04:00.610 ± 00:00:02.150
CHI2	1024	152	00:13:14.865 ± 00:00:07.704
FD	8	5	00:00:34.150 ± 00:00:00.283
FD	16	3	00:00:29.915 ± 00:00:00.372
FD	32	8	00:00:49.848 ± 00:00:00.486
FD	64	11	00:01:24.405 ± 00:00:02.222
FD	128	24	00:01:58.231 ± 00:00:01.367
FD	256	60	00:04:14.807 ± 00:00:02.097
FD	512	79	00:05:30.383 ± 00:00:01.690
FD	1024	77	00:05:19.756 ± 00:00:01.987
MI	8	3	00:00:18.330 ± 00:00:00.179
MI	16	3	00:00:26.374 ± 00:00:00.191
MI	32	7	00:00:45.323 ± 00:00:00.277
MI	64	10	00:01:23.869 ± 00:00:02.673
MI	128	30	00:02:52.458 ± 00:00:00.980
MI	256	49	00:04:03.581 ± 00:00:02.118
MI	512	118	00:07:41.898 ± 00:00:02.527
MI	1024	155	00:13:30.913 ± 00:00:10.112

Tabela 28 – Tempo de classificação do modelo RF

Método	NF	NF'	Tempo de classificação (RF)
CHI2	8	4	00:00:03.233 ± 00:00:00.058
CHI2	16	16	00:00:16.063 ± 00:00:00.462
CHI2	32	28	00:00:38.277 ± 00:00:02.966
CHI2	64	42	00:01:01.993 ± 00:00:05.301
CHI2	128	69	00:01:41.650 ± 00:00:01.394
CHI2	256	116	00:03:58.305 ± 00:00:18.075
CHI2	512	121	00:04:41.128 ± 00:00:20.914
CHI2	1024	152	00:05:26.634 ± 00:00:23.013
FD	8	5	00:00:38.227 ± 00:00:03.353
FD	16	3	00:00:30.882 ± 00:00:03.135
FD	32	8	00:00:46.834 ± 00:00:06.070
FD	64	11	00:01:01.692 ± 00:00:07.017
FD	128	24	00:01:00.205 ± 00:00:05.707
FD	256	60	00:01:27.565 ± 00:00:05.863
FD	512	79	00:01:28.888 ± 00:00:02.975
FD	1024	77	00:01:29.501 ± 00:00:02.371
MI	8	3	00:00:10.758 ± 00:00:00.829
MI	16	3	00:00:27.742 ± 00:00:03.525
MI	32	7	00:00:41.540 ± 00:00:03.053
MI	64	10	00:00:50.739 ± 00:00:02.594
MI	128	30	00:01:22.403 ± 00:00:04.564
MI	256	49	00:01:31.318 ± 00:00:08.911
MI	512	118	00:01:40.479 ± 00:00:01.903
MI	1024	155	00:02:19.988 ± 00:00:09.063

Da análise dos resultados do modelo MLP-WEKA, pode-se concluir o seguinte:

- Conjuntos criados com o método CHI2 apresentaram os piores resultados de precisão e revocação, devido ao fato de possuírem quantidades muito grandes de vetores nulos;
- O melhor resultado — 81,48% de precisão e 80,63% de revocação — foi obtido com o conjunto criado pelo método MI, com vetores de dimensionalidade 1024 reduzida para 155. O segundo melhor resultado — 79,59% de precisão e 78,12% de revocação — foi obtido com o conjunto criado pelo método FD, com vetores de dimensionalidade 512 reduzida para 118;

- Conjuntos criados com o método MI apresentaram resultados, de precisão e revocação, acima de 75%, em sua maioria. Por sua vez, conjuntos criados com o método FD apresentaram resultados, de precisão e revocação, acima de 61%, em sua maioria.

Da análise dos resultados do modelo RF, pode-se concluir o seguinte:

- Conjuntos criados com o método CHI2 apresentaram os piores resultados de precisão e revocação, devido ao fato de possuírem quantidades muito grandes de vetores nulos;
- O melhor resultado — 98,13% de precisão e 98,13% de revocação — foi obtido com o conjunto criado pelo método FD, com vetores de dimensionalidade 8 reduzida para 5. O segundo melhor resultado — 94,64% de precisão e 94,13% de revocação — foi obtido com o conjunto criado pelo método FD, com vetores de dimensionalidade 512 reduzida para 79;
- Em geral, os resultados — precisão e revocação — obtidos com conjuntos criados pelos métodos FD e MI são próximos.

Da comparação dos resultados do modelo MLP-WEKA com os do modelo RF, pode-se concluir o seguinte:

- O maior tempo de treinamento da RF — 13 minutos e 30 segundos — foi com um conjunto criado com o método MI, com vetores de dimensionalidade 1024. Este tempo, porém, é consideravelmente baixo, se comparado com o maior tempo de treinamento do MLP-WEKA — 1 hora, 43 minutos e 32 segundos;
- O MLP-WEKA apresentou os melhores tempos de classificação. Seu pior resultado foi de 40 segundos. O pior tempo de classificação da RF, por sua vez, foi de 2 minutos e 19 segundos;
- Portanto, conclui-se que o melhor modelo classificador é o RF.

Por fim, da comparação dos resultados obtidos neste quarto experimento com os resultados obtidos no terceiro experimento, pode-se concluir a importância do

tratamento dos dados. Com a redução da inconsistência da base UNIFEI, obtem-se uma melhoria — de 94,68% para 98,13% — na melhor precisão, bem como uma melhoria — de 94,42% para 98,13% na melhor revocação.

7 CONCLUSÃO

A dissertação propõe um novo *anti-spam* — SASCA. O SASCA possui arquitetura modular. Usa modelos de *machine learning* para a classificação dos *e-mails*, não fazendo, assim, uso de listas negras e brancas.

O módulo Pré-Filtro detecta técnicas usadas por *spammers*. Utiliza dois filtros — filtro HTML e filtro de texto — para detectá-las e torná-las visíveis, por meio de marcadores, aos módulos seguintes.

O módulo Seleção de *Tokens* utiliza métodos estatísticos para selecionar os *tokens* — palavras, *tags* HTML e marcadores — mais relavantes para a classificação dos *e-mails*. O módulo Classificador faz uso de dois modelos de *machine learning* — MLP e RF — para classificar os *e-mails* nas classes *ham* e *spam*. O módulo Armazenamento armazena todos os *e-mails* recebidos em arquivos de forma a poderem ser auditados.

O SASCA foi implementado na linguagem Java. Sua implementação segue os padrões de projeto apresentados por Yener e Theedom (2014). Seu código é claro, simples e de fácil manutenção. Para se criar um novo filtro para o módulo Pré-Filtro, por exemplo, basta apenas implementar uma interface e inserir o novo código na lista de filtros do módulo. O uso abundante de bibliotecas, além de simplificar o código, reduz a possibilidade de erros, pois são implementadas por programadores que estudam e dominam o assunto.

A implementação do SASCA foi feita de forma a aumentar seu desempenho em execução. Em seu código, por exemplo, ao invés de listas, usa vetores (*arrays*) preferência com tamanho fixo para evitar a criação de classes iteradoras que, futuramente, seriam excluídas pelo coletor de lixo da máquina virtual Java.

Todo o seu código foi escrito em língua inglesa, inclusive comentários e documentação (JavaDocs) dos métodos para permitir sua internacionalização. Igualmente, todas as mensagens produzidas pelo SASCA estão em um arquivo. Isto facilita a tradução das mensagens para outros idiomas.

A implementação de uma interface visual para sua configuração é possível, devido ao fato do SASCA utilizar arquivos para armazená-la. Assim, a interface precisaria

apenas exibir, ao administrador, as opções disponíveis de configuração, com base nos comentários de cada parâmetro nos arquivos. Após salvar a configuração nos arquivos, a própria interface pode permitir, ao administrador, a execução do SASCA ou de qualquer um de seus módulos, individualmente. O uso de arquivos também permite ao SASCA ser executado como um serviço, ou seja, como um processo *daemon*.

O SASCA foi comparado ao *anti-spam* CanIt, utilizado, atualmente, na UNIFEI. Através dos resultados dos experimentos, observou-se que o tempo médio de processamento de cada *e-mail* gasto pelo SASCA chega a ser quase dez vezes menor que o do CanIt. Os dois modelos de *machine learning* do módulo Classificador do SASCA foram capazes de aprender a base de *e-mails* classificada pelo CanIt. Mesmo sendo esta base inconsistente, devido às classificações inconsistentes do CanIt, os dois modelos conseguiram classificar seus *e-mails* corretamente, sobretudo o modelo RF.

7.1 SUGESTÕES PARA TRABALHOS FUTUROS

Pretende-se instalar o SASCA na UNIFEI. Inicialmente, executando-o em paralelo com o CanIt e, depois, sem o CanIt.

Pretende-se criar um manual de instalação e de uso. Pretende-se, outrossim, adicionar o projeto SASCA ao GitHub, como um projeto de código aberto.

Cogita-se, em uma versão futura, modificar o Postfix do SASCA de forma a torná-lo *multithread*.

Pretende-se criar uma interface gráfica para a configuração e execução do SASCA. Por fim, pretende-se disponibilizar outros modelos de *machine learning* no módulo Classificador do SASCA, de forma a dar ao administrador mais opções de acordo com os tipos e idiomas de *e-mails* que seu domínio recebe.

A LICENÇA DE USO

O código do *anti-spam* desenvolvido e suas ferramentas estão licenciados sobre a GNU GENERAL PUBLIC LICENSE versão 3, de 29 de junho de 2007.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License form ore details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

B LIMITAÇÃO DOS SISTEMAS DE ARQUIVOS

Neste anexo são apresentadas as limitações dos principais sistemas de arquivos do Windows e do Linux (ISW, 2009).

- FAT32
  - Quantidade máxim de arquivos: 268.173.300.
  - Número máximo de arquivos por diretório: 2^{16}-1 (65.535).
  - Tamanho máximo de um arquivo: 2 GiB - 1 sem LFS (*Large File Support*), 4 GiB - 1 com.
- NTFS
  - Número máximo de arquivos: 2^{32}-1 (4.294.967.295)
  - Tamanho máximo de um arquivo
    - Implementação: 2^{44} - 2^{6} *bytes* (16 TiB - 64 KiB)
    - Teórico: 2^{64} - 2^{6} *bytes* (16 EiB - 64 KiB)
  - Tamanho máximo da unidade de armazenamento (HD)
    - Implementação: 2^{32} - 1 *clusters* (256 TiB - 64 KiB)
    - Teórico: 2^{64} – 1 *clusters* (16 EiB - 64 KiB)
- ext2
  - Quantidade máxima de arquivos: 10^{18}
  - Quantidade máxima de arquivos por diretório: 1,3 x 10^{20} – há problemas de performance quando esta quantidade passa de 10.000
  - Tamanho máximo de um arquivo
    - 16 GiB para bloco com tamanho de 1 KiB

        - 256 GiB para bloco com tamanho de 2 KiB
        - 2 TiB para bloco com tamanho de 4 KiB
        - 2 TiB para bloco com tamanho de 8 KiB
    - Tamanho máximo da unidade de armazenamento (HD)
        - 4 TiB para bloco com tamanho de 1 KiB
        - 8 TiB para bloco com tamanho de 2 KiB
        - 16 TiB para bloco com tamanho de 4 KiB
        - 32 TiB para bloco com tamanho de 8 KiB

- ext3
    - Quantidade máxima de arquivos: é o menor valor entre a divisão do tamanho do HD por 2^{13} e o número de blocos
    - Tamanho máximo de um arquivo: o mesmo de ext2
    - Tamanho máximo da unidade de armazenamento (HD): o mesmo de ext2

- ext4
    - Quantidade máxima de arquivos: 2^{32} - 1 (4.294.967.295)
    - Quantidade máxima de arquivos por diretório: ilimitada
    - Tamanho máximo de um arquivo: 2^{44} - 1 bytes (16 TiB - 1)
    - Tamanho máximo da unidade de armazenamento (HD): 2^{48} - 1 bytes (256 TiB - 1)

C TABELA ASCII

A figura 31 contém a primeira tabela ASCII criada, com apenas 7 *bits* para representar todos os caracteres necessários em sua época de criação.

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Figura 31 – Tabela ASCII

Fonte:
https://codehs.gitbooks.io/apjava/content/Methods/strings-and-characters.html

D POSTFIX E SUBETHA SMTP

Neste anexo tem-se duas subseções que contêm a descrição de como implementar um simples servidor STMP para enviar e receber *e-mails* com o Postfix através da filtragem de conteúdo. A primeira subseção (D.1) descreve os procedimentos para criar um servidor SMTP para receber os *e-mails* e a segunda subseção (D.2) tem-se a descrição de como configurar o Postfix para enviar os *e-mails* para o SubEtha SMTP.

D.1 SUBETHA SMTP

Para criar um servidor SMTP para receber *e-mails* com o SubEtha SMTP, precisa-se instanciar uma classe `SMTPServer` que é o servidor. E para o tratamento destes é feito através de um implementação da interface `MessageHandler`.

Durante a criação do objeto `SMTPServer`, deve-se passar como parâmetro uma fábrica que é encarregada de instanciar a nossa implementação da `MessageHandler`, esta fábrica é a classe `MessageHandlerFactory`.

No trecho de código a seguir tem-se a primeira implementação: o `MyHandler`. Este implementa a interface `MessageHandler` que possui 3 métodos. Ao receber um *e-mail* o servidor irá chamar primeiramente o método `from` passando como parâmetro uma `String` que indica quem é remetente; em seguida o método `recipient` é chamado e, da mesma forma que o anterior, informa para quem é o *e-mail* (o destinatário). Por último o método *data* é executado e contém o conteúdo do *e-mail*, isto é, seu cabeçalho e corpo.

```
1 public static class MyHandler implements MessageHandler {
2   public void from(String from) throws RejectException {
3     ...
4   }
5   public void recipient(String recipient) throws RejectException {
6     ...
7   }
```

```
8   public void data(InputStream data) throws RejectException,
        TooMuchDataException, IOException {
9     ...
10  }
11 }
```

```
1  public static class MyHandler implements MessageHandler {
2    public void from(String from) throws RejectException {
3      ...
4    }
5    public void recipient(String recipient) throws RejectException {
6      ...
7    }
8    public void data(InputStream data) throws RejectException,
         TooMuchDataException, IOException {
9      ...
10   }
11 }
```

Uma vez definido como cada *e-mail* recebido será tratado, deve-se instanciar a fábrica e criar o servidor, conforme pode ser visto no trecho de código seguinte.

No código da fábrica tem-se apenas um método que é utilizado para instanciar um novo objeto para o tratamento das mensagens. Isto é feito para que se possa ter controle sobre quantos *e-mails* serão tratados por vez. Por exemplo, se o objetivo é tratar no máximo 10 *e-mails*, coloca-se um contador, se esta quantidade for atingida então *bloqueia-se* a execução do método até que seja encerrado o processamento de um ou mais *e-mails*.

É importante ressaltar a linha 3 onde é configurado a porta pela qual se irá receber os *e-mails*. Esta deve ser a mesma configurada no Postfix – conforme a subseção D.2 seguinte, foi usada a porta 10025.

```
1  public static void main() {
2    SMTPServer smtpServer = new SMTPServer(new MyFactory());
3    smtpServer.setPort(10025);
4    smtpServer.start();
```

```
5  }
6
7  public static class MyFactory implements MessageHandlerFactory {
8    public MessageHandler create(MessageContext c) {
9      return new MsgHandler(c);
10   }
11 }
```

D.2 FILTRAGEM DE CONTEÚDO DO POSTFIX

Para permitir que o Postfix envie e receba os *e-mails* do SAS, via SubEtha SMTP, deve-se modificar o arquivo `master.cf`. Essas modificações foram feitas com base em Dent (2003, p. 177-181) e Walter (2016), porém modificadas para servir à proposta deste trabalho.

Nas próximas linhas a seguir tem-se as configurações que devem ser inseridas ao final do arquivo. Estas definem que *e-mails* recebido pelo serviço `smtpd` (linha 1) deve ser encaminhado para um outro serviço que esteja executando localmente na porta 10025, definido na linha 2 por `127.0.0.1:10025`.

Na linha 3 definimos a quantidade máxima de *e-mails* que devem ser enviados por vez para serem processados. E a linha 4 configura uma otimização de velocidade, a qual está disponível a partir da versão 2.7 do Postfix.

```
1 smtp inet n - n - 20 smtpd
2   -o smtpd_proxy_filter=127.0.0.1:10025
3   -o smtpd_client_connection_count_limit=10
4   -o smtpd_proxy_options=speed_adjust # Postfix 2.7 ou maior
```

Após definir como o *e-mail* será enviado para o SubEtha SMTP, deve-se configurar como ele será recebido de volta. Neste caso, será usado o mesmo Postfix para receber estes *e-mail*, porém há a possibilidade de que um outro Postfix seja configurado – ou ainda um outro servidor, que não seja o Postfix.

Assim como as configurações anteriores, as seguintes também devem ser inseridas ao final do arquivo. Na linha 1 tem-se que o *e-mail* será recebido pelo serviço `smtpd` através do IP local e da porta 10026. As linhas seguintes configuram as restrições

de recebimento deste *e-mail* e as redes de encaminhamento para este (linhas 2 e 10).

```
1  127.0.0.1:10026 inet n - n - - smtpd
2    -o smtpd_authorized_xforward_hosts=127.0.0.0/8
3    -o smtpd_client_restrictions=
4    -o smtpd_helo_restrictions=
5    -o smtpd_sender_restrictions=
6    # Para Postfix 2.10 e superior, devemos inserir os seguintes
7    -o smtpd_relay_restrictions=
8    -o smtpd_recipient_restrictions=permit_mynetworks,reject
9    -o smtpd_data_restrictions=
10   -o mynetworks=127.0.0.0/8
11   -o receive_override_options=no_unknown_recipient_checks
```

E TAGS DO HTML5

Na tabela E a seguir, tem-se todas as *tags* do HTML 5 que estão disponíveis para a construção de páginas *web*.

<!DOCTYPE>	<a>	<abbr>	<address>	<area>
<article>	<aside>	<audio>	<b>	<base>
<bdi>	<bdo>	<blockquote>	<body>	
<button>	<canvas>	<caption>	<cite>	<code>
<col>	<colgroup>	<command>	<datalist>	<dd>
<del>	<details>	<dfn>		<dl>
<dt>		<embed>	<fieldset>	<figcaption>
<figure>	<footer>	<form>	<h1> - <h6>	<head>
<header>	<hgroup>	<hr>	<html>	<i>
<iframe>	<img>	<input>	<ins>	<kbd>
<keygen>	<label>	<legend>		<link>
<map>	<mark>	<menu>	<meta>	<meter>
<nav>	<noscript>	<object>		<optgroup>
<option>	<output>	<p>	<param>	<pre>
<progress>	<q>	<rp>	<rt>	<ruby>
<s>	<samp>	<script>	<section>	<select>
<small>	<source>			<style>
<sub>	<summary>	<sup>	<table>	<tbody>
<td>	<textarea>	<tfoot>	<th>	<thead>
<time>	<title>	<tr>	<track>	<u>
	<var>	<video>	<wbr>	

Tabela 29 – *Tags* disponíveis no HTML 5

F EXEMPLO COM JSOUP

Para realizar o processamento do HTML pela `jsoup`, deve-se passar como parâmetro uma `String` que contém todo o HTML, em seguida enviar o código para a biblioteca que, então, irá realizar o processamento.

No trecho de código a seguir tem-se a `String html` que representa um código HTML qualquer. Ao executar o comando `Jsoup.parse(html)`, recebe-se um objeto do tipo `Document` com o qual será realizado o processamento das *tags*.

```
1  String html = "<html><head><title>First parse</title></head>"
2      + "<body><p>Parsed HTML into a doc.</p></body></html>";
3  Document doc = Jsoup.parse(html);
```

Para gerar o objeto `Document`, a `jsoup` realiza a interpretação de todo o conteúdo do HTML e remonta todo este no formato de objetos, desta forma pode-se navegar pelas *tags* conforme apresentado no trecho de código a seguir.

Como pode-se observar, deve-se *pedir* para a `jsoup` todos as *tags*; o que é feito na linha 1, através do método `select` – o método `body` pega o corpo do HTML, isto é, todo o conteúdo da *tag* `body` –. A navegação pelos elementos é feita através da iteração apresentada nas linhas 2-5.

```
1  Elements elements = document.body().select("*");
2  for (Element element : elements) {
3    Tag tag = element.tag();
4    Attributes attributes = element.attributes();
5  }
```

Um objeto `Element` representa uma *tag*; para obter seu nome deve-se executar o método `tag`, conforme pode ser visto na linha 3. Quando for necessário processar um ou mais de seus atributos usa-se o método `attributes` para obtê-los, conforme pode ser visto na linha 4.

G ALTERAÇÃO DO PROTOCOLO SMTP NO POSTFIX

Para adicionar as modificações propostas, são necessários 3 passos: adicionar o código criado junto ao código-fonte do Postfix, modificar o arquivo de compilação Makefile.in e alterar 2 fontes para realizar a chamada da função que faz a consulta ao banco de dados e processa a resposta.

G.1 O CÓDIGO

Crie o arquivo `bw_check_list.c` com o seguinte código.

```
1  #include<mysql/mysql.h>
2  #include<stdio.h>
3  #include<string.h>
4
5  static int check_spam(const char *list, const char *senderAddr,
       const char *userAddr) {
6      char sql[] = "SELECT User FROM `";
7      strcat(sql, list);
8      strcat(sql, "` WHERE addr = '");
9      strcat(sql, senderAddr);
10     strcat(sql, "' AND user = '");
11     strcat(sql, userAddr);
12     strcat(sql, "'");
13
14     // conecta ao banco
15     MYSQL DBCon;
16     MYSQL_RES *result;
17     MYSQL_ROW dados;
18
```

```
19      mysql_init(&DBCon);
20
21      // Conecta com o banco de dados
22      mysql_real_connect(&DBCon, "127.0.0.1", "user", "password",
            "database", 0, NULL, 0);
23
24      // Executa a consulta
25      mysql_query(&DBCon, sql);
26
27      // Recebe os dados da consulta
28      result = mysql_store_result(&DBCon);
29
30      // 1: tem na lista, 0: nao tem
31      int res = result ? 1 : 0;
32
33      // Limpa da memoria
34      mysql_free_result(result);
35
36      // Fecha a conexao
37      mysql_close(&DBCon);
38
39      return res;
40  }
```

G.2 ARQUIVO *MAKE*

Para ser compilado devidamente, o Postfix precisa ter o arquivo `Makefile.in` que está localizado na pasta `src/smtpd` tenha um parâmetro de compilação adicionado. Este parâmetro permitirá a biblioteca `mysql` compile junto ao código.

Para isto, deve-se substituir as linhas abaixo,

$(PROG): $(OBJS) $(LIBS)

$(CC) $(CFLAGS) $(SHLIB_RPATH) -o $@ $(OBJS) $(LIBS) $(SYSLIBS)

por

$(PROG): $(OBJS) $(LIBS)
$(CC) $(CFLAGS) $(SHLIB_RPATH) -o $@ $(OBJS) $(LIBS) $(SYSLIBS) ‘mysql_config –cflags –libs‘

G.3 LISTA NEGRA

No arquivo `smtpd_check.c`, localizado na pasta `src/smtpd`, altera-se a função `reject_unknown_address` que realiza a rejeição. Ao final desta, antes do último `return`, adiciona-se as seguintes linhas.

```
1     if (check_spam(account_to_check, addr) == 1)
2     return (SMTPD_CHECK_DUNNO);
```

E no início deste arquivo deve-se inserir a chamada ao arquivo que contém a função `check_spam`. Aqui supõe-se que este arquivo esteja no mesmo diretório.

```
1   #include "bw_check_list.c"
```

REFERÊNCIAS

AARON. *Emailreg.org is a scam*. 2010. Disponível em: <http://joystickjunkie.blogspot.com.br/2010/04/emailregorg-is-scam.html>. Acesso em: 01/10/2016.

APACHE. *The Apache Spam Assassin Project*. 2018. Disponível em: <https://spamassassin.apache.org/>. Acesso em: 12/04/2017.

ARAGãO, M. V. C. *Estudo e Pesquisa em Sistemas Anti-Spam*. 2018. Disponível em: <https://github.com/marcelovca90/anti-spam-weka-cli>. Acesso em: 05/01/2018.

BARRACUDA. *Barracuda Email Security Gateway*. 2018. Disponível em: <https://www.barracuda.com/products/emailsecuritygateway>. Acesso em: 12/02/2018.

BOWLER, M. *HtmlUnit - Welcome to HtmlUnit*. 2016. Disponível em: <http://htmlunit.sourceforge.net/>. Acesso em: 22/12/2016.

BREIMAN, L. Random forests. *Machine learning*, Springer, v. 45, n. 1, p. 5–32, 2001. ISSN 1098-6596.

CERT. *Cartilha de Segurança – Spam*. 2012. Disponível em: <http://cartilha.cert.br/spam>. Acesso em: 13/09/2016.

CIPOLI, P. *O que é DNS*. 2016. Disponível em: <http://canaltech.com.br/o-que-e/internet/o-que-e-dns/>. Acesso em: 27/09/2016.

CROCKER, D. *E-mail history*. 2000. Disponível em: <http://www.livinginternet.com/e/ei.htm>. Acesso em: 13/09/2016.

DELAWARE, U. of. *Clock Discipline Algorithm*. 2014. Disponível em: <https://www.eecis.udel.edu/~mills/ntp/html/discipline.html>. Acesso em: 12/02/2018.

DENT, K. D. *Postfix: the definitive guide*. [S.l.]: "O'Reilly Media, Inc.", 2003.

ECKARDT, T. *Anti Spam SMTP Proxy*. 2018. Disponível em: <http://www.thockar.com/assp-home/>. Acesso em: 12/02/2018.

FECOMERCIO-SP. *Legislação*. 2016. Disponível em: <http://www.fecomercio.com.br/noticia/saiba-por-quanto-tempo-guardar-os-documentos-da-empresa>. Acesso em: 12/02/2018.

FLATSCHART, F. *JavaMail – API Design Specification*. [S.l.]: Brasport, 2013.

FREED, N. *Media Types*. 2016. Disponível em: <http://www.iana.org/assignments/media-types/media-types.xhtml>. Acesso em: 21/10/2016.

GARDNER, M. J.; ALTMAN, D. G. Confidence intervals rather than p values: estimation rather than hypothesis testing. *Br Med J (Clin Res Ed)*, British Medical Journal Publishing Group, v. 292, n. 6522, p. 746–750, 1986.

GHERARDI, L.; BRUGALI, D.; COMOTTI, D. A java vs. c++ performance evaluation: a 3d modeling benchmark. In: SPRINGER. *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. [S.l.], 2012. p. 161–172.

HALL, M. et al. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, ACM, v. 11, n. 1, p. 10–18, 2009.

HANSEN, A. B. *qpsmtpd*. 2018. Disponível em: <http://smtpd.github.io/qpsmtpd/>. Acesso em: 12/02/2018.

HARRIS, E. The next step in the spam control war: Greylisting. *http://projects. puremagic. com/greylisting/whitepaper. html*, 2003.

HAYKIN, S. *Neural Networks and Learning Machines*. [S.l.]: Pearson, 2009.

HEDLEY, J. *jsoup – Java HTML Parser*. 2016. Disponível em: <https://jsoup.org/>. Acesso em: 28/11/2016.

HISKEY, D. *How the word "spam" came to mean "junk message"*. 2010. Disponível em: <http://www.todayifoundout.com/index.php/2010/09/how-the-word-spam-came-to-mean-junk-message>. Acesso em: 13/09/2016.

HOEPERS, C.; STEDING-JESSEN, K. Gerência de porta 25: Motivaçao, importância da adoçao para o combate ao spam e discussoes no brasil e no mundo. 2009.

HOUSTON, P. *Instant jsoup How-to*. [S.l.]: Packt Publishing Ltd, 2013.

ISW. *How many files can I put in a directory?* 2009. Disponível em: <http://stackoverflow.com/a/466596>. Acesso em: 18/10/2016.

JIMéNEZ, F. et al. Multi-objective evolutionary feature selection for online sales forecasting. *Neurocomputing*, v. 234, n. Supplement C, p. 75 – 92, 2017. ISSN 0925-2312. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0925231216315612>. Acesso em: 29/11/2017.

KARASINSKI, E. *História do e-mail*. 2009. Disponível em: <http://www.tecmundo.com.br/web/2763-a-historia-do-email.htm>. Acesso em: 13/09/2016.

KLENSIN, J.; GELLENS, R. *RFC 6409: Message Submission for Mail*. 2011. Disponível em: <https://tools.ietf.org/html/rfc6409>. Acesso em: 21/09/2017.

LARRAMO, M. *What is a Mail Server and How Does it Work?* 2016. Disponível em: <http://www.samlogic.net/articles/mail-server.htm>. Acesso em: 07/12/2016.

LATHAM, P. E.; ROUDI, Y. Mutual information. *Scholarpedia*, v. 4, n. 1, p. 1658, 2009. Revision #122173.

LUTUS, P. *The Anti-Spam Home Page*. 2006. Disponível em: <http://arachnoid.com/lutusp/antispam.html>. Acesso em: 14/09/2016.

MAATEN, L. v. d.; HINTON, G. Visualizing data using t-sne. *Journal of machine learning research*, v. 9, n. Nov, p. 2579–2605, 2008.

OXFORD. *Definition of spam*. 2016. Disponível em: <http://www.oxforddictionaries.com/definition/english/spam>. Acesso em: 13/09/2016.

OZER, Z. *Biggest. Spam Scam. Ever.* 2008. Disponível em: <http://zacharyozer.blogspot.com.br/2008/10/worst-engineers-ever.html>. Acesso em: 01/10/2016.

PATRIK. *100,000 spams per week no more*. 2016. Disponível em: <https://sourceforge.net/p/assp/wiki/ASSP_Success_Stories/>. Acesso em: 08/02/2018.

POLAMURI, S. *How the random forest algorithm works in machine learning*. 2017. 1–14 p. Disponível em: <http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/>.

SATTERFIELD, B. *Ten Spam-Filtering Methods Explained*. 2006. Disponível em: <https://www.techsoupcanada.ca/en/learning_center/10_sfm_explained>. Acesso em: 26/09/2016.

SCHWANDT, F.; KRöGER, T. *Spam e-mail traffic share*. 2016. Disponível em: <https://www.statista.com/statistics/420391/spam-email-traffic-share/>. Acesso em: 16/12/2016.

SKOLL, D.; WHITE, B. *CanIt-Pro*. 2018. Disponível em: <https://www.roaringpenguin.com/products/canit-pro>. Acesso em: 12/02/2018.

SLETTNES, T. et al. *Spam Filtering for Mail Exchangers*. [S.l.]: Version, 2004.

SMITH, G. *Unsung innovators: Gary Thuerk, the father of spam*. 2007. Disponível em: <http://www.computerworld.com/article/2539767/cybercrime-hacking/unsung-innovators--gary-thuerk--the-father-of-spam.html>. Acesso em: 13/09/2016.

SORKIN, D. E. *Spam Statistics and Facts*. 2016. Disponível em: <http://www.spamlaws.com/spam-stats.html>. Acesso em: 13/09/2016.

STÄRK, R. F.; SCHMID, J.; BÖRGER, E. *Java and the Java virtual machine: definition, verification, validation*. [S.l.]: Springer Science & Business Media, 2012.

STEVENS, J.; SCHNITZER, J. *SubEtha SMTP - GitHub*. 2015. Disponível em: <https://github.com/voodoodyne/subethasmtp>. Acesso em: 21/11/2016.

VENEMA, W. Z. *The Postfix*. 2016. Disponível em: <http://www.postfix.org/>. Acesso em: 10/11/2016.

WALTER, S. T. *ClamSMTP: Using with Postfix*. 2016. Disponível em: <http://thewalter.net/stef/software/clamsmtp/postfix.html>. Acesso em: 29/11/2016.

WONG, J. *Barracuda Central (vs) EmailReg.org*. 2016. Disponível em: <https://thejame.wordpress.com/2016/04/27/barracuda-central-vs-emailreg-org/>. Acesso em: 01/10/2016.

YAMANE, T. Statistics: An introductory analysis. Harper & Row New York, NY, 1973.

YENER, M.; THEEDOM, A. *Professional Java EE design patterns*. [S.l.]: John Wiley & Sons, 2014.