UNIVERSIDADE DE CAXIAS DO SUL
Centro de Computação e Tecnologia da Informação
Curso de Bacharelado em Ciência da Computação,

Vinícius Crestani

CLASSIFICAÇÃO DE MENSAGENS DE E-MAIL UTILIZANDO O ALGORITMO *ARTIFICIAL BEE COLONY* (ABC)

Caxias do Sul

2012

Vinícius Crestani

CLASSIFICAÇÃO DE MENSAGENS DE E-MAIL UTILIZANDO O ALGORITMO *ARTIFICIAL BEE COLONY* (ABC)

Trabalho de Conclusão de Curso para obtenção do Grau de Bacharel em Ciência da Computação da Universidade de Caxias do Sul.

Prof. Carine G. Webber
Orientador

Caxias do Sul
2012

RESUMO

Insetos como abelhas, formigas e cupins apresentam um nível de inteligência individual muito baixo, mas coletivamente são capazes de resolver problemas complexos de forma altamente organizada (Lianying e Fengyu, 2006). Buscando inspiração no senso de inteligência coletiva destes insetos, este trabalho tem o objetivo de realizar um estudo sobre algoritmos que baseiam-se nesta metáfora e assim desenvolver um sistema que seja capaz de identificar e classificar mensagens de e-mail. O algoritmo utilizado neste trabalho é o *Artificial Bee Colony* (Karaboga, 2005). Ele baseia-se no comportamento natural das abelhas forrageadoras, abelhas responsáveis por encontrar fontes de alimentos e informar às outras abelhas, que estão na colmeia esperando por estas informações. Para o algoritmo de classificação de mensagens podem ser consideradas como fontes de alimentos as próprias mensagens. Elas serão exploradas pelas abelhas artificiais na tentativa de obter as melhores regras para classificação ou então descobrir a classe a que a mensagem pertence. As mensagens podem ser classificadas em seguras, de *phishing* ou *spam*. Mensagens de *phishing* são aquelas que tentam se passar por empresas reais na tentativa de obter dados pessoais. Outra categoria de mensagens são os *spams* que podem ser classificadas como mensagens de cunho publicitário cujo recebimento não foi solicitado. Para aplicar o algoritmo das abelhas, um sistema de classificação foi desenvolvido a partir de modelos de engenharia de software sobre *datasets* contendo mensagens de e-mail. Após o desenvolvimento do sistema, métricas e técnicas de amostragem foram aplicadas com o objetivo de avaliar e comparar a utilização desta técnica em relação a outras para classificação de mensagens. Os resultados obtidos através da comparação com outros algoritmos podem ser considerados como positivos já que em alguns casos a taxa de mensagens classificadas corretamente foi superior ao algoritmo comparado.

Palavras-chaves: Inteligência de Exames, Algoritmos de Abelhas, *Artificial Bee Colony*, Classificação de Mensagens, Métodos de Amostragem.

ABSTRACT

Insects like bees, ants and termites have a very low level of individual intelligence, but collectively they are able to solve complex problems in a highly organized way (Lianying and Fengyu, 2006). Looking for inspiration in the sense of collective intelligence of these insects, this work has the objective to conduct a study of algorithms that are based on this metaphor and thus develop a system which is capable of identifying and classifying e-mail. The algorithms used in this work will be the *Artificial Bee Colony* (Karaboga, 2005), he is based on the natural behavior of bees foragers, bees responsible for finding sources and inform others bees, that are waiting in the hive for these information. To the message classification algorithm, can be regarded food sources the e-mail messages. They will be explored by artificial bees in attempt to find the best rules to classify messages or identify the class of the message. The messages can be classified as secure, phishing or spam. Phishing messages are those that try to pass by real companies in an attempt to obtain personal data. Another category of messages are the spams that can be classified as advertising messages imprint whose receipt was not requested. To apply the algorithm of bees, a classification system was developed from models of software engineering and applied in a datasets of messages. After the development of the system, metrics and sampling techniques were applied in order to evaluate and compare the use of this technique over the other for the classification of messages. The results obtained by comparison with others algorithm can be considered very positive whereas in some case the tax of messages correctly classified was higher than of compared algorithm.

Key-Words: Swarm Intelligence, Bees Algorithm, *Artificial Bee Colony*, Messages Classification, Sample Methods.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Sigla	Significado em Português	Significado em Inglês
ABC	Algoritmo da Colmeia das Abelhas	Artificial Bee Colony
BA	Algoritmo da Abelhas.	Bees Algorithm.
SCMIE	Sistema de classificação de mensagens baseado em Inteligência de Enxames.	Classification system of messages based on intelligence of swarms.
IIR	Resposta infinita ao impulso	Infinite impulse response

SUMÁRIO

1 INTRODUÇÃO

Classificação é uma forma de análise de dados que utiliza modelos para definir a classe a qual pertence cada informação. Esses modelos são conhecidos como classificadores e têm a função de prever a categoria de uma informação (Han, Kamber e Pei, 2012).

O problema de classificação de dados é um problema de grande importância na computação e se aplica às mais diversas áreas, tais como marketing direcionado, detecção de fraudes em transações de cartões de crédito, na medicina, entre outras. Por exemplo, para categorização de células cancerígenas ou realização de diagnósticos na medicina e na detecção de *spam* e *phishing* em mensagens de e-mail.

A figura 1.1 representa de forma básica o funcionamento da classificação de dados. A partir de um conjunto *x* de atributos, um modelo de classificação irá definir a qual classe de informação *y* essa entrada pertence.

Figura 1.1: Conjunto de atributos fornecidos na entrada gera saída com a classe a qual a informação pertence. (Tan; Steinbach; Kumar, 2006).

O processo de classificação de dados é dividido em duas etapas. A primeira fase do processo é a etapa de treinamento e a segunda é a etapa de classificação. Na etapa de treinamento um classificador é construído através de um conjunto de exemplos. O conjunto de treinamento é formado por dados associados à classe a qual pertence. Por ser conhecida a classe a que pertence cada dado da amostra, esse processo é conhecido como aprendizagem supervisionada. A segunda etapa é conhecida como teste, nela é determinado se o nível de acerto do modelo é aceitável, caso seja, o modelo é utilizado para classificar novos dados (Han, Kamber e Pei, 2012).

O problema da classificação é um problema importante da computação para o qual não existe uma solução ótima. Diversas técnicas conhecidas, tais como as árvores de decisão,

redes bayesianas, rede neurais, entre outras, são aplicadas para se tentar chegar próximo da melhor solução. O fato de nenhuma técnica produzir uma solução ótima evidencia a importância do estudo de novas técnicas.

A técnica de classificação a ser abordada neste trabalho com o objetivo de avaliar e comparar com outras técnicas já conhecidas é o algoritmo ABC (*Artificial Bee Colony*). Esse é um algoritmo baseado no comportamento das abelhas forrageadoras, proposto por Karaboga em 2005 (Karaboga, 2005) para resolver problemas de otimização numérica (Kumbhar e Krishnan, 2011). Além da utilização na otimização de soluções, existem outros trabalhos que demostram com sucesso sua utilização em áreas como *clustering* (Karaboga e Ozturk, 2009), *clustering* difuso (Karaboga e Ozturk, 2010) e classificação (Shukran et al,. 2011).

Partindo da metáfora dos algoritmos baseados em comportamento coletivo da natureza, o objetivo do trabalho é buscar uma solução que possa ser compatível com o problema de classificação de dados, dentro da área de classificação de mensagens de e-mail. Dados do mês de fevereiro de 2012 apresentados no site do Kaspersky Lab, demostram que aproximadamente 80% dos e-mails que circulam na internet são e-mails indesejados (*spam*) ou tentativas de golpes e fraudes (*phishing*) (Kaspersky Lab, 2012).

De acordo com James (2005), podemos usar uma definição geral para *phishing* como sendo o envio de e-mail forjado para um grande número de destinatários que finge vir de uma empresa real com o objetivo de obter dados pessoais, como números de cartão de crédito ou senhas de contas bancárias. A figura 1.2 apresenta uma falsa mensagem enviada em nome do Banco do Brasil informando que o banco adotou um novo sistema de validação e solicita que a vítima crie uma nova senha. Entretanto, o link leva a uma página falsa na quais dados confidenciais do usuário são solicitados.

Já *spam* pode ser definido como e-mail não solicitado. Eles normalmente são enviados a um grande número de pessoas.

O problema de classificação de *phishing* e *spam* consiste em identificar a categoria a qual pertencem os e-mails recebidos e assim separá-los conforme o seu conteúdo, prevenido que o usuário receba e-mails indesejados ou até mesmo que caia em golpes.

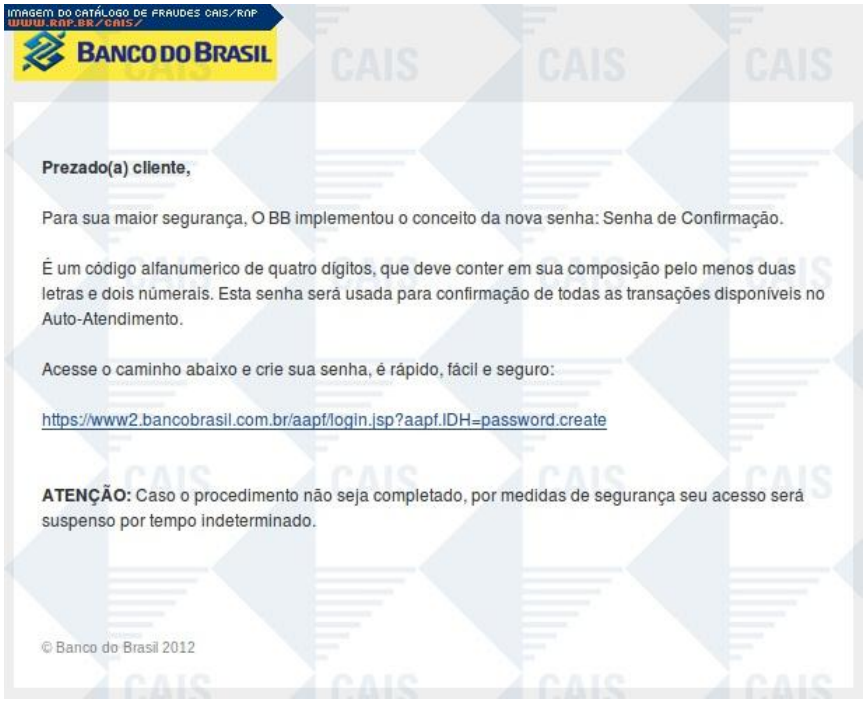


Figura 1.2: Exemplo de falsa mensagem de *phishing* em nome do Banco do Brasil (CAIS, 2012) .

1.1 OBJETIVOS

O objetivo deste trabalho é aplicar uma solução baseada na inteligência coletiva das abelhas para resolver o problema de classificação de dados, mais especificamente o de classificação de mensagens de *phishing* e *spam*. Para isso, foi implementado o algoritmo ABC, além de realizadas comparações com algoritmos já existentes para analisar a qualidade da solução obtida para a classificação dos dados.

Para alcançar o objetivo principal do trabalho os seguintes objetivos específicos foram desenvolvidos:

(a) Estudo sobre o algoritmo ABC (Karaboga, 2005) na sua utilização para classificação de dados.

(b) Definição do projeto para implementação do algoritmo.

(c) Implementação do algoritmo com base no projeto já definido.

(d) Realização de testes utilizando bases de dados públicas de e-mails para verificação quanto ao resultado esperado e a possíveis erros de implementação.

(e) Análise do algoritmo implementado através de comparações de desempenho com outros algoritmos de classificação de dados já existentes.

(f) Apresentação das conclusões a respeito da utilização do algoritmo ABC na classificação de dados.

1.2 ORGANIZAÇÃO DO DOCUMENTO

O capítulo 2 traz uma abordagem sobre a área da Inteligência Artificial conhecida como Inteligência de Enxames, descrevendo aspectos do comportamento das abelhas que estão inseridas neste contexto. Esse capítulo está dividido em seções que explicam desde a inspiração na natureza, a metáfora com a computação, alguns algoritmos baseados nesta técnica até um exemplo real de código implementado.

O capítulo 3 apresenta um projeto com o objetivo de buscar uma solução para o problema de classificação de mensagens de e-mail. O capítulo está dividido em seções que descrevem o processo, sugerem uma modelagem e uma arquitetura para a implementação do código, descrevem os dados de entrada e como o sistema gerado a partir destas diretrizes irá ser avaliado.

O capítulo 4 descreve alguns conceitos importantes como a criação das fontes de alimentos, a descoberta e a avaliação das regras, além de descrever casos de uso que explicam o funcionamento do sistema. Há também uma seção que apresenta os resultados obtidos pelo sistema criado e compara com outros algoritmos de classificação.

O capítulo 5 apresenta as conclusões finais do trabalho desenvolvido. No capítulo são apresentadas seções que descrevem a síntese do trabalho realizado, conclusões a respeito dos resultados e sugestões de melhorias para trabalhos futuros.

2 ALGORITMOS DE ABELHAS

Os algoritmos de abelhas são algoritmos que buscam sua inspiração no comportamento natural das abelhas. Esses algoritmos fazem parte da área de Inteligência Artificial conhecida como Inteligência de Enxames ou Insetos Sociais. Além dos algoritmos de abelhas existem algoritmos baseados também no comportamento das formigas por exemplo (Brownlee, 2011).

As abelhas, formigas e os cupins são insetos que possuem como características principais o fato de individualmente possuírem baixa inteligência e ações não inspecionadas, mas que de maneira global possuem um grande senso de inteligência coletiva dentro da colônia, fazendo com que problemas complexos possam ser resolvidos. Por exemplo, as formigas são capazes de encontrar o caminho mais curto para uma fonte de alimento dentro um grande número de possibilidades de caminhos diferentes (Lianying e Fengyu, 2006). Como característica principal da colônia pode ser citada a auto-organização, ou seja, as atividades não precisam ser controladas por um agente central ou inspecionadas, elas são realizadas independentemente. A chave para o sucesso da colônia é que cada membro individualmente contribui constantemente deixando informações que serão úteis aos outros integrantes da colônia, como no caso das formigas que deixam feromônios ou as abelhas que realizam danças (Lianying e Fengyu apud Bonabeau e Dorigo, 2003; White, 1997).

2.1 INSPIRAÇÃO BIOLÓGICA

Na natureza, abelhas realizam a coleta de néctar em grandes áreas. Dentro da colmeia há abelhas com o papel designado de encontrar as flores que possuem a maior qualidade e quantidade de alimentos. As abelhas comunicam-se umas com as outras através de danças que informam às demais abelhas sobre a direção, distância e a qualidade e o tipo da fonte de alimento encontrada (Karaboga, 2005).

Figura 2.1 é representa o funcionamento da dança das abelhas. Veja que *s* indica o tempo que a dança é realizada para informar a distância da fonte de alimento. O ângulo alfa em que a dança é realizada serve para mostrar às outras abelhas em qual direção a fonte de

alimento está em relação à colmeia (Rouillard, 2012).

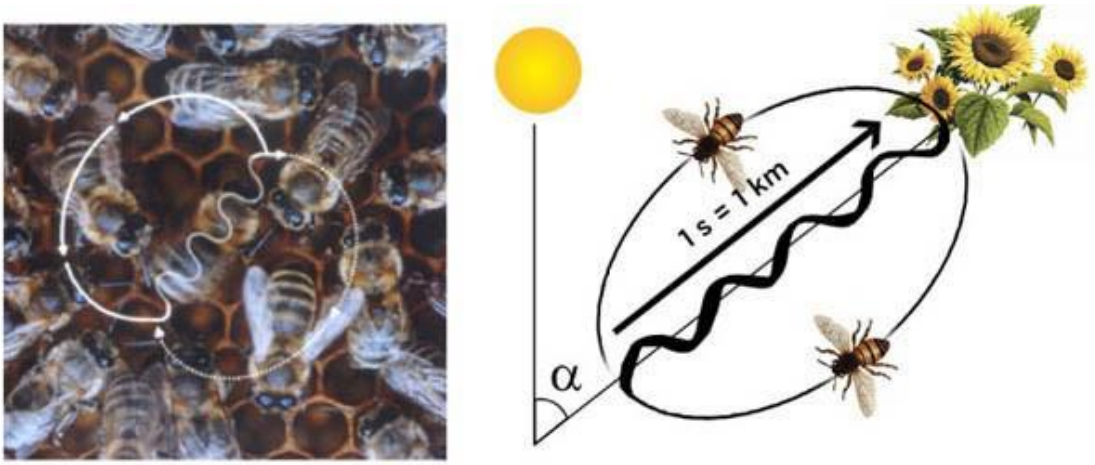


Figura 2.1: Representação do funcionamento da dança das abelhas (Rouillard, 2012).

Dentro da organização da colmeia, as abelhas responsáveis por encontrar fontes de alimentos e repassar estas informações às outras abelhas são chamadas de abelhas forrageadoras elas podem ser divididas em dois grupos: forrageadoras livres e forrageadoras empregadas.

No grupo das abelhas forrageadoras livres existem dois tipos de abelhas. O primeiro tipo engloba as abelhas exploradoras, que irão buscar randomicamente por novas fontes de alimentos próximas a colmeia. As segundas são as abelhas espectadoras que ficam aguardando pela dança das abelhas empregadas, a fim de selecionar uma fonte de alimento que lhe pareça mais aproveitável.

Já as abelhas forrageadoras empregadas são as abelhas que possuem informações sobre fontes de alimentos. Elas são responsáveis também por passar essas informações às outras abelhas da colmeia. As abelhas empregadas irão tornar-se abelhas exploradoras assim que a fonte de alimento se esgotar.

2.2 METÁFORA

Neste trabalho vão ser estudados dois algoritmos que se baseiam no comportamento natural das abelhas forrageadoras, as abelhas responsáveis por encontrar fontes de alimentos e repassar essa informação à colmeia. Este algoritmos já foram utilizados com um bom desempenho em diversas áreas como otimização, classificação e clustering.

Para implementação do algoritmo três componentes são essencialmente considerados: fontes de alimentos, abelhas forrageadoras empregadas e abelhas forrageadoras livres. Além disso, há dois comportamentos básicos a serem seguidos: recrutamento para uma fonte de alimento e o abandono de uma fonte de alimento (Karaboga, 2005).

As fontes de alimentos representam a posição da solução de otimização do problema. A qualidade da fonte é expressa pelo *fitness* da solução.

Nas seções 2.4 e 2.5 são apresentados dois algoritmos distintos que se baseiam no modelo de enxames.

2.3 BEES ALGORITHM

O objetivo do processo do algoritmo denominado Bees Algorithm (BA) é localizar e explorar as melhores posições dentro de um problema de busca (BROWNLEE, 2011).

Exploradores são enviados para encontrar amostras randomicamente dentro do espaço do problema para localizar as melhores posições. As melhores posições são exploradas através de uma busca local, na qual um pequeno número de boas posições são exploradas mais que as outras. As melhores posições são exploradas de forma contínua. Entretanto exploradores continuam a realizar buscas para encontrar outras boas posições.

O BA foi desenvolvido para ser utilizado em problemas de otimização de funções contínuas e combinatórias. A seguir é apresentado um pseudocódigo:

Entrada: $Problema_{tamanho}, Abelhas_{quant}, Posições_{quant}, MelhoresPosições_{quant},$
$TamanhoÁrea_{tamanho}, MelhoresAbelhas_{quant}, OutrasAbelhas_{quant}$

Saída: $Abelha_{melhor}$

1. População = InicializarPopulação($Abelhas_{quant}, Problema_{tamanho}$);
2. **Enquanto** ¬CondiçãoDeParada() **faça**
3. AvaliarPopulação(População);
4. $Abelha_{melhor}$ = PegarMelhorSolução(População);
5. PróximaGeração = 0;
6. $Área_{tamanho}$ = ($TamanhoÁrea_{tamanho} \times ReduçãoÁrea_{fator}$);

7. $Posições_{melhores}$ = SelecionarMelhoresPosições(População, $Posições_{quant}$);
8. **Para cada** $Posição_i$ **em** $Posições_{melhores}$ **faça**
9. $AbelhasRecrutadas_{quant}$ = 0;
10. **Se** $i < MelhoresPosições_{quant}$ **então**
11. $AbelhasRecrutadas_{quant} = MelhoresAbelhas_{quant}$;
12. **Senão**
13. $AbelhasRecrutadas_{quant} = OutrasAbelhas_{quant}$;
14. **Fim**
15. Região = 0;
16. **Para** j **até** $AbelhasRecrutadas_{quant}$ **faça**
17. Região = CriarRegiãoDasAbelhas($Posição_i$, $Área_{tamanho}$);
18. **Fim**
19. PróximaGeração = PegarMelhorSolução(Região);
20. **Fim**
21. $AbelhasRestantes_{quant}$ = ($Abelhas_{quant} - Posições_{quant}$);
22. **Para** j **até** $AbelhasRestantes_{quant}$ **faça**
23. PróximaGeração = CriarAbelhasRandomicamente();
24. **Fim**
25. População = PróximaGeração;
26. **Fim**
27. **Retorna** $Abelha_{melhor}$;

A variável $Área_{tamanho}$ é usada para definir o tamanho da região. Por exemplo em uma função de um problema de otimização, cada dimensão de uma posição terá uma amostra de $x_i \pm (rand() \times Área_{tamanho})$.

A variável $Área_{tamanho}$ é decrementada a cada interação, normalmente por uma constante com valor de 0.95.

O número de melhores posições ($MelhoresPosições_{quant}$) necessariamente precisa ser menor que o de posições ($Posições_{quant}$), e o número de melhores abelhas ($MelhoresAbelhas_{quant}$) é normalmente menor que de outras abelhas ($OutrasAbelhas_{quant}$).

2.4 ARTIFICIAL BEE COLONY

O *Artificial Bee Colony* (ABC) é um algoritmo baseado na Inteligência de Enxames que teve sua utilização proposta por D. Karaboga em 2005. Desde então este algoritmo é muito utilizado para diversas aplicações de otimização como estruturas de proteínas terciárias (Bahamish, Adbullah e Salam, 2009), filtros IRR digital (Karaboga, 2009), redes neurais artificiais (Karaboga e Akay, 2005) entre outras (SHUKRAN, CHUNG, YEH, WAHID, ZAIDI, 2011).

2.4.1 OTIMIZAÇÃO

Para implementação do algoritmo ABC existem duas importantes funções que sustentam o algoritmo:

$$P_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_i} \qquad \text{Eq. (2.1)}$$

$$V_{ij} = X_{ij} + \Phi_{ij}(X_{ij} + X_{kj}) \qquad \text{Eq. (2.2)}$$

Onde P_i é o valor da probabilidade associado com i fontes de alimentos calculados pela Eq.2.1. Uma abelha espectadora irá escolher uma fonte de alimento com base no valor de P_i. Nesta equação fit_i representa a qualidade ou tamanho da fonte de alimento que é medida pelas abelhas trabalhadoras empregadas e SN é o número de fontes de alimentos que é igual ao número de abelhas empregadas.

No algoritmo ABC, as abelhas artificiais necessitam fazer busca para encontrar novas possíveis soluções, A Eq.2.2 representa a estratégia de busca original utilizada pelo ABC como algoritmo de otimização. V_{ij} é o novo candidato a posição de fonte de alimento gerado por esta equação, onde k € [1,2,...,SN] e j € [1,2,...,D] são parâmetros escolhidos randomicamente, mas k deve ser diferente de j. Assim X_{ij} e X_{kj} representam as diferentes posições de fontes antigas de alimentos. A diferença entre essas duas posições é a distância de uma fonte de alimento em relação à outra. SN representa o número de abelhas empregadas e

D é o número de parâmetros de otimização. Φ_{ij} é um número randômico entre [-1,1] e controla a distância de uma posição de fonte de alimento vizinha em todo X_{ij}.

Um dos fatores mais importantes no ABC é o processo de seleção gulosa que determina se uma nova fonte encontrada é melhor do que uma fonte anterior. Se a nova fonte de alimento for considerada melhor que a antiga, esta irá ser substituída da memória pela nova, caso contrário a informação antiga é mantida.

Junto ao parâmetro de controle SN há outros dois parâmetros que são utilizados na definição básica do ABC. O valor de limitação da posição de fonte de alimento e o número máximo de ciclos de buscas. Esses valores que devem ser definidos com cuidado pelo usuário, já que com valores elevados o processo de otimização pode vir a tornar-se lento, em contra partida valores muitos baixos podem não retornar uma boa solução esperada.

A seguir uma representação do pseudocódigo do algoritmo ABC para o problema de otimização:

1. Carrega as amostras de treinamento.
2. Gera a população inicial $i = 1 \ldots SN$.
3. Calcula o fitness (fit_i) da população.
4. Seta o ciclo para 1.
5. **Repita**
6. **Para** cada abelha empregada
   {
   Produzir nova solução de V_i utilizando Eq.2.2
   Calcular valor de fit_i
   Aplicar processo de seleção gulosa
   }
7. Calcular o valor de P_i para soluções (X_i) utilizando Eq.2.1
8. **Para** cada abelha espectadora
   {
   Selecione uma solução X_i com base em P_i
   Produz nova solução V_i
   Calcula o valor de fit_i
   Aplicar processo de seleção gulosa

}

9. **Se** existir alguma solução não verificada
   Então substitua pela nova função que irá ser randomicamente produzida Φ_{ij}
10. Memorize a melhor solução gerada até agora.
11. Ciclo = ciclo + 1
12. **Até** ciclo == número máximo de ciclos.

2.4.2 CLUSTERING

Clustering é o processo de reconhecimento de agrupamentos naturais ou agrupamentos em dados multidimensionais com base em algumas similaridades que são medidas entre os elementos dos grupos (Jain, Murty, Flynn, 1999). A distância da medida geralmente é utilizada para avaliar as similaridades entre os padrões.

O processo de *clustering* separa os objetos em grupos ou classes, esse processo pode ser realizado através de aprendizado não supervisionado ou supervisionado. No processo de aprendizado não supervisionado, o número de classes não precisa ser especificado para os dados de treinamento. Já para o processo *clustering* supervisionado é necessário especificar o que será aprendido, o número de classes. Os *data sets* de treinamento irão conter as informações dizendo a que classe pertencem cada elemento.

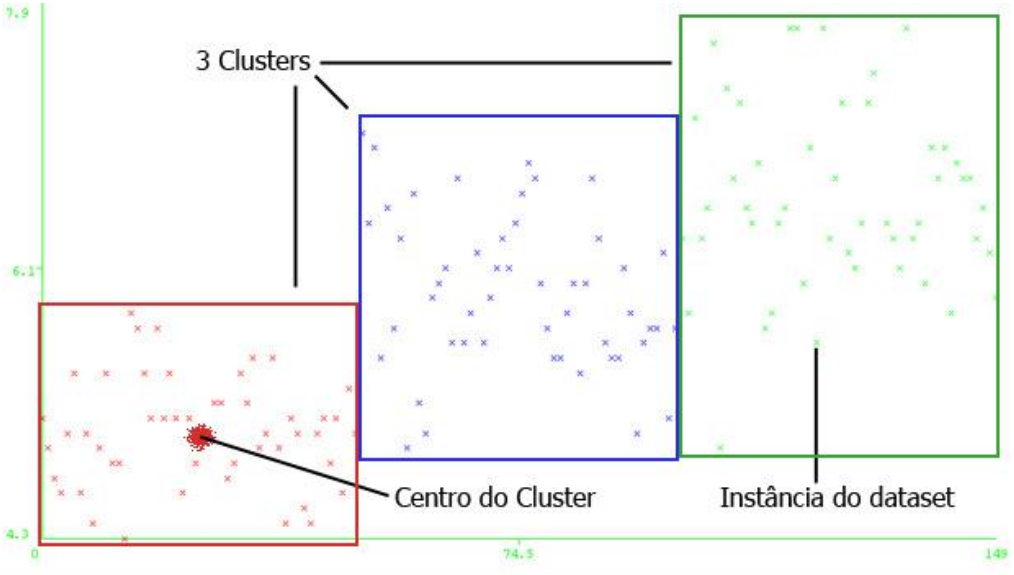


Figura 2.2: Representação de agrupamento de informações através de clusters.

No algoritmo ABC, a primeira metade da colônia é formada por abelhas artificiais empregadas e a outra metade é constituída por abelhas espectadoras. O número de abelhas empregadas ou abelhas espectadoras será igual ao número de soluções, no caso centros do *cluster*, na população. O primeiro passo é gerar uma população inicial distribuída randomicamente $P(C = 0)$ de SN soluções (posições de fontes de alimentos), na qual SN configura o tamanho da população. Para cada solução z_i na qual $i = 1,2, \dots, SN$ representam um vetor D-dimensional. Onde, D é o número do produto de tamanho de entrada e tamanho do *cluster* para cada *data set*. Após a inicialização, a população de posições (soluções) é submetida a repetidos ciclos em que cada grupo de abelhas irá executar o seu papel até a uma determinada condição de parada ser atingida.

As abelhas empregadas fazem alterações em suas memórias sobre a posição (solução) dependendo do local da informação (informação visual) e de testes da quantidade de néctar (valor do *fitness*) da nova fonte (nova solução). Avaliando que a nova posição é melhor da que a posição que já esta na memória, a antiga será esquecida e a nova posição será memorizada.

Após todas as abelhas empregadas completarem o processo de busca, elas irão realizar a dança que sinaliza com informações sobre as fontes de alimentos encontradas e suas localizações que serão compartilhadas com as abelhas espectadoras. As abelhas espectadoras avaliarão as informações vindas de todas as abelhas e farão a escolha de qual fonte de alimento que lhe pareça ser mais atraente. No caso das abelhas empregadas, elas irão verificar se a quantidade de alimentos de uma nova fonte candidata é melhor do que a que já está na memória dela. Se for o caso, a fonte antiga será esquecida e a nova será gravada.

2.4.3 CLASSIFICAÇÃO

O algoritmo ABC para a tarefa de classificação foi proposto por Shukran em 2011 (Shukran et al., 2011). Existem seis componentes principais a serem considerados: (a) formato da regra; (b) função de fitness; (c) estratégia de busca local; (d) descoberta da regra; (e) regra da poda; (f) estratégia de previsão.

(a) Formato da Regra

No método de classificação, cada atributo da regra de classificação contém dois valores: O limite inferior e o limite superior. Este formato pode ser visto na figura 2.3.

Atributo 1		Atributo 2		...	Atributo N	
Limite Inferior	Limite Superior	Limite Inferior	Limite Superior		Limite Inferior	Limite Superior

Figura 2.3: Representação do formato de regra (Shukran et al., 2011).

Onde a atributo 1 até o atributo N representam todos os atributos do *dataset*. Cada atributo possui um limite inferior que representa o mais baixo valor para a regra e um limite superior que representa o maior valor que a regra permite. Existem outros três valores associados à regra de classificação: classe de predição (Classe X), o valor do fitness e porcentagem de cobertura da regra. Estes três valores possuem uma relação maior com a função de fitness e a estratégia de previsão, portanto serão explicados com mais detalhes nas seções seguintes.

(b) Função de Fitness

Para avaliar o valor do fitness, a função utilizada irá calcular o valor aproximado da quantidade de néctar. A representação desta função é definida a seguir na Eq.2.3.

$$\text{Valor do Fitness} = \frac{VP}{VP+FN} \times \frac{VN}{VN+FP} \qquad \text{Eq. (2.3)}$$

Na Eq.2.3 as variáveis VP, FN, FP e VN são os diferentes tipos de informações que respectivamente representam os valores de Verdadeiros Positivos, Falsos Negativos, Falsos Positivos e Verdadeiros Negativos. Para entender melhor o que cada um desses valores significa é importante entender dois conceitos antes.

- Quando o algoritmo verifica o tipo de uma instância, ele irá avaliar cada atributo desta instância. Se o valor de um atributo estiver entre a borda inferior e a borda superior, isto significa que o atributo pode ser coberto pela regra. Se todos os atributos de uma instância podem ser cobertos pela regra, logos esta instância poderá pertencer a regra que a esta cobrindo.
- Se a classe da instância avaliada for igual à classe prevista pela regra, isso demonstra que a instância possui sua classe prevista pela regra.

I. Verdadeiros Positivos (VP): O número de instâncias cobertas pela regra que possuem sua classe prevista pela regra.
II. Falsos Negativos (FN): O número de instâncias não cobertas pela regra, mas que possuem sua classe prevista pela regra.
III. Falsos Positivos (FP): o número de instâncias cobertas pela regra, mas que não possuem a classe prevista pela regra.
IV. Verdadeiros Negativos (VN): o número de instâncias não cobertas pela regra e que não possuem a classe prevista pela regra.

(c) Estratégia de busca local

Quando uma abelha empregada não encontra a informação necessária ou ela atinge o número máximo de ciclos, vai ser necessário que ela busque uma nova fonte através da estratégia de busca local. O algoritmo original ABC para otimização, utiliza a Eq. 2.2 como método de cálculo para troca de local de busca. Com o intuito de adaptar e melhorar o desempenho do algoritmo para classificação, em Shukran et al. (2011) foram realizados testes que comprovam que a Eq.2.4, a equação simplificada é mais eficiente.

$$V_{ij} = X_{kj} \qquad \text{Eq. (2.4)}$$

Onde V_{ij} representa a posição da nova fonte de alimento e X_{kj} representa a fonte anterior. O valor de i e k esta entre 1 e SN, mas k necessita ser um valor diferente de i. Além disso, j é o número de dimensões. Na classificação de dados, as dimensões de um *dataset* são iguais ao número de atributos do *dataset*. $k \in [1,2,\dots,SN]$ e $j \in [1,2,\dots,D]$ são parâmetros escolhidos randomicamente. Quando esta estratégia é colocada em prática para classificação, a Eq.2.4 é modificada da seguinte maneira:

$$V_{ij}\ borda\ inferior = X_{k1j}\ borda\ inferior \qquad \text{Eq. (2.5)}$$

$$V_{ij}\ borda\ superior = X_{k2j}\ borda\ superior \qquad \text{Eq. (2.6)}$$

Onde K_1 e K_2 são dois números randômicos diferentes de i.

(d) Descoberta da Regra

O objetivo da regra de classificação é definir regras que possam identificar uma classe específica de diferentes grupos. Assim, a fase da descoberta da regra é a parte crucial do algoritmo de classificação uma vez que a definição da regra é o resultado desta fase. O fluxograma para descoberta de regra é apresentado na figura 2.4.

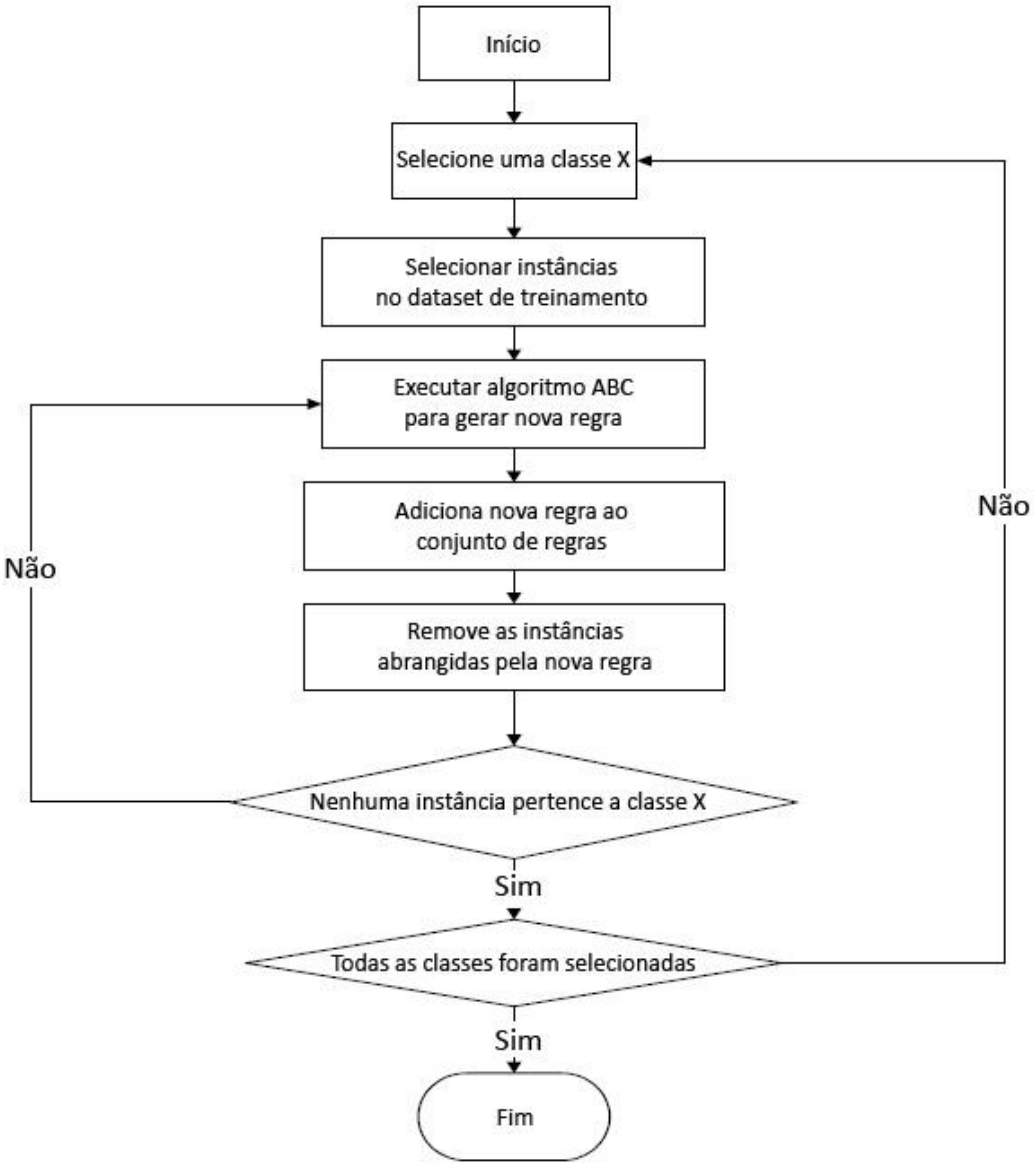


Figura 2.4: Fluxograma da descoberta de regra do algoritmo ABC (Shukran et al., 2011).

No estágio de início é definido a valor de limite inferior e limite superior de cada atributo. O processo é definido pela Eq.2.7 e Eq.2.8 listadas a seguir:

$$\text{Limite inferior} = f - k_1 \times (F_{max} - F_{min}) \qquad \text{Eq.2.7}$$

$$\text{Limite superior} = f + k_2 \times (F_{max} - F_{min}) \qquad \text{Eq.2.8}$$

Para as duas equações, F_{max} e F_{min} são respectivamente, o valor máximo e o valor mínimo de cada atributo. A diferença entre as duas variáveis significa o intervalo do atributo. O valor original é f, k_1e k_2 são valores randômicos entre 0 e 1.

O algoritmo da regra de classificação pode descobrir automaticamente a regra para cada classe. Para cada classe selecionada, o algoritmo vai buscar regras iterativamente até que o conjunto de regras possa cobrir todas as instâncias pertencentes à classe atual. Cada parte da regra única pertencente a uma estrutura de regra e cada conjunto de regras é formado por diversas regras.

(e) Regra da Poda

Depois de todas as classes serem processadas e todos os conjuntos de regras serem gerados, cada regra será colocada no processo de poda. O objetivo principal da regra de poda é remover limitações de regras redundantes que possam ser desnecessariamente inclusas no conjunto de regras. Uma vez que atributos sem relação irão influenciar de forma negativa o resultado da classificação, a regra da poda pode aumentar a precisão dos resultados. O processo é repetido até que todas as regras dentro do conjunto de regras sejam avaliadas.

(f) Estratégia de Previsão

O conjunto de regras podadas será utilizado para prever novos dados de classes desconhecidas. Algumas vezes, o teste dos dados poderá ser coberto por mais de uma regra para diferentes classes. Quando isto acontecer, a estratégia de previsão irá determinar qual classe deverá ser prevista. Existem três passos que devem ser seguidos para realizar a previsão:

(a) Calcular o valor da previsão para todas as regras que foram cobertas pelo teste dos dados.
(b) Acumular estes valores de previsão de acordo com as possíveis classes.
(c) Selecionar a classe com o maior valor de previsão como a classe final.

Após o processo de estratégia de previsão, o núcleo é a função de previsão que irá ser utilizada para calcular o valor de previsão para cada regra. Este valor é definido pela Eq. 2.9 a seguir:

$$previsão = (\alpha \times fitness) + (\beta \times \ porcentagem\ de\ cobertura) \qquad \text{Eq.2.9}$$

Veja que α e β são dois parâmetros associados ponderados com o valor da regra do fitness e com a porcentagem de cobertura da regra respectivamente, $\alpha \in [0,1]$ e $\beta = (1 - \alpha)$. A Eq.2.3 calcula o valor no fitness para cada regra. A porcentagem de cobertura da regra define a proporção de dados cobertos pela regra que tem a classe prevista pela regra (TP). Este cálculo é feito pela Eq.2.10:

$$\text{Porcentagem de cobertura} = \frac{TP}{N} \qquad \text{Eq.2.10}$$

Note que N é o total de dados que pertencem a classes previstas pelas regras.

A figura 2.5 apresenta o processo geral de classificação de dados contendo as principais etapas citadas nesta seção.

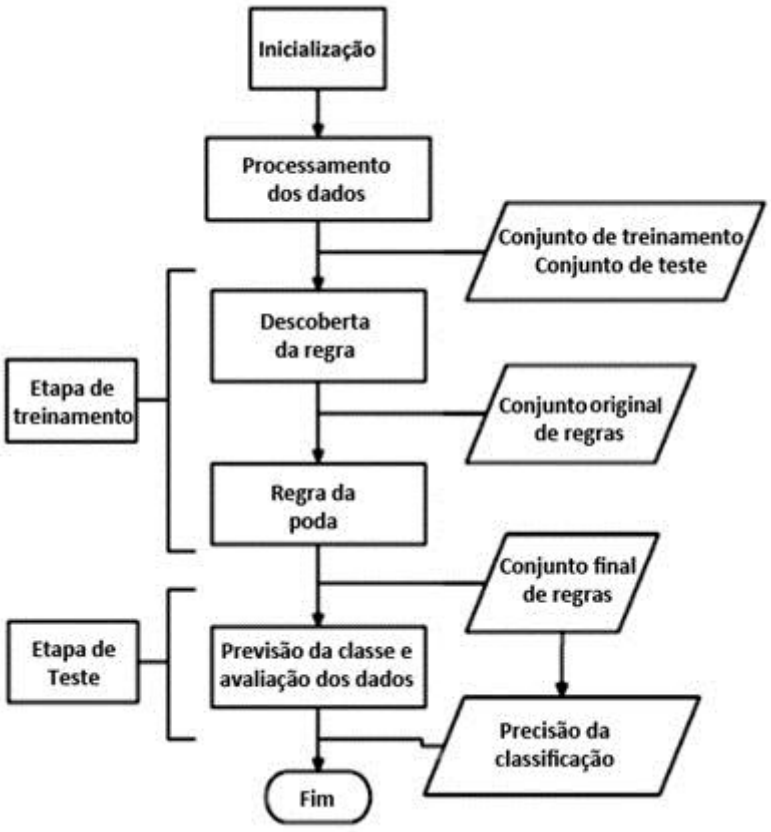


Figura 2.5: Fluxograma de data mining de classificação

2.5 EXEMPLO DE IMPLEMENTAÇÃO

Esta seção explica o funcionamento da implementação do Anexo A, um exemplo em linguagem Java do algoritmo ABC para otimização numérica que foi disponibilizado na internet pelo grupo de pesquisa de sistemas inteligentes através do site da Universidade de Erciyes da Turquia (Artificial Bee Colony (ABC) Algorithm Homepage, 2012).

O código está escrito em linguagem orientada a objeto e possui duas classes:

(a) test.java: Classe que possui o método principal e executa a inicialização do código. A partir desta classe a simulação é disparada.

(b) beeColony.java: Classe que possui todos os métodos que simulam o comportamento das abelhas com o objetivo de resolver o problema de otimização numérica.

Os principais passos do método que inicializa o código são os seguintes.

1. Laço com a quantidade de vezes em que o processo é executado.
2. Inicializar a classe que simula o comportamento das abelhas.
3. Laço com a quantidade de ciclos de forrageamento.

Para compreender o que cada um desses passos representa eles irão ser detalhados a seguir:

1. Laço com a quantidade de vezes em que o processo é executado.

   Este laço faz com que o algoritmo repita todo o processo diversas vezes, no caso deste exemplo, ele será repetido 30 vezes.

   (a) for (run = 0; run < bee.runtime; run++)

   A repetição deste laço faz com que o processo tenha maior robustez refinando a solução.

2. Inicializar a classe que simula o comportamento das abelhas.

   Inicializa as fontes de alimentos. Informações sobre o tamanho e o fitness da fonte são geradas

   (a) bee.initial();

   A abelha artificial irá guardar a informação da melhor fonte de alimento.

   (b) bee.MemorizeBestSource();

3. Laço com a quantidade de ciclos de forrageamento.

   Dentro deste laço são executados os métodos que simulam o comportamento das abelhas.

   (a) for (iter = 0; iter < bee.maxCycle; iter++)

   A quantidade de ciclos de forrageamento, neste caso possui a variável definida em 2500.

   (b) bee.SendEmployedBees();

   A abelha empregada vai ser enviada para analisar e obter informações sobre as fontes de alimentos.

   (c) bee.CalculateProbabilities();

   Através do melhor fitness encontrado, será calculada a probabilidade de escolha de cada fonte de alimento proporcional a sua qualidade. Os valores de probabilidade de cada fonte podem ser calculados usando o valor do fitness da fonte e normalizados pela divisão do maior valor de fitness encontrado. No código em questão a função utilizada é apresenta a seguir: prob[i] = (0.9*(fitness[i]/maxfit))+0.1.

   (d) bee.SendOnlookerBees();

   A abelha espectadora vai produzir uma nova solução e comparar com a solução que ela já conhece, mantendo assim a melhor solução. Se a solução não puder ser melhorada, é incrementada variável *trial*, que controla o número de tentativas de fontes que não puderam ser melhoradas.

   (e) bee.MemorizeBestSource();

   A melhor solução ou fonte de alimento é guardada.

   (f) bee.SendScoutBees();

   Determina as fontes de alimentos que alcançaram o número máximo de

tentativas. Se a fonte atingir este limite é inicializada uma nova fonte.

Além das classes e métodos apresentados, o código possui alguns parâmetros e variáveis de destaque cujos valores de inicialização foram propostos pelo autor da implementação.

1. Parâmetros de controle:
   (a) **int NP = 20:** Número total de abelhas da colônia, incluindo abelhas empregadas e espectadoras
   (b) **int FoodNumber = NP/2:** O número de fontes de alimentos sugeridos foi a metade do número de abelhas.
   (c) **int limit = 100:** Representa o número máximo de tentativas que uma abelha empregada pode tentar fazer para melhorar uma fonte de alimento.
   (d) **int maxCycle = 2500:** Número de ciclos de forrageamento, condição de parada do problema.

2. Variáveis específicas do problema:
   (a) int D = 100: Número de parâmetros do problema a serem otimizados.
   (b) double lb = -5.12: Valor de limite inferior dos parâmetros.
   (c) double ub = 5.12: Valor de limite superior dos parâmetros.
   (d) int runtime = 30: Define a quantidade de vezes que o algoritmo é executado para determinar a robustez do processo.
   (e) double Foods[][] = new double[FoodNumber][D]: Foods é a população de fontes de alimentos. Cada linha da matriz Foods é um vetor contendo D parâmentros à serem otimizados. A quantidade de linhas da matriz Foods é igual ao valor da variável FoodNumber.
   (f) double f[] = new double[FoodNumber]: Vetor contendo os valores objetivos de cada fonte de alimento.
   (g) double fitness[] = new double[FoodNumber]: Vetor contendo os valores de fitness que representa a qualidade de cada fonte de alimento.
   (h) double trial[] = new double[FoodNumber]: Vetor contendo a quantidade de tentativas para cada solução que não pode ser melhorada.

(i) **double prob[] = new double[FoodNumber]:** Vetor que mantém as probabilidades de uma fonte de alimento (solução) a ser escolhida.

(j) **double solution[] = new double[D]:** Nova solução produzida por $V_{ij} = X_{ij} + phi_{ij} \times (X_{kj} - X_{ij})$. j é um parâmetro escolhido randomicamente e k é escolhido randomicamente de uma solução diferente de i.

3 PROJETO DE DESENVOLVIMENTO

Utilizando conceitos de classificação de dados e Inteligência de Enxames estudados nas seções anteriores, este capítulo apresenta uma proposta de desenvolvimento de uma ferramenta de classificação de mensagens baseado no algoritmo ABC (Karaboga, 2005). Para desenvolver este trabalho o ABC foi escolhido por ser um algoritmo já utilizado com sucesso em outras áreas, incluindo classificação de dados, como apresenta o capítulo 2.

Nas próximas seções serão apresentados conceitos e modelos para a construção da ferramenta de classificação que irá ser denominada SCMIE (Sistema de Classificação de Mensagens baseado em Inteligência de Enxames).

3.1 DESCRIÇÃO DO PROBLEMA

O problema estudado se referre à classificação de mensagens de *e-mail*. Estas mensagens vão ser classificadas em uma das três classes seguintes:

(a) ***Phishing:*** Classe que representa os e-mails que possuem alta probabilidade de serem e-mails fraudulentos com o objetivo de obter dados pessoais do usuário.

(b) ***Spam:*** Classe que representa os e-mails que são recebidos de forma não solicitada normalmente contendo material publicitário.

(c) **Seguro:** Classe que representa os e-mails que não se enquadram em nenhuma das categorias citadas anteriormente e que podem ser considerados seguros por não conterem nenhum conteúdo agressivo ao usuário.

Para a criação das regras e validação das mensagens são utilizados atributos cujas eficiências já foram avaliadas e de forma bem sucedida em estudos realizados por FETTE et. Al., (2007) em que são utilizados os seguintes atributos:

(a) **Contém links:** O número de links em um e-mail pode ser um indicativo que este não é seguro.

(b) **URLs formadas por endereço IP:** Empresas reais raramente utilizam links que contenham um endereço IP (ex: http://192.168.0.1/paypal.cgi?fix account). Links formados desta maneira possuem grande de chance de serem do tipo *phishing*.

(c) **Contém javascript:** E-mails com código javascript podem ser utilizados para

esconder informações do usuário. Este atributo pode ser marcado como contém ou não contém código javascript.

(d) Concentração de termos de *spam*: Através de uma lista com termos comuns em e-mail do tipo *spam*, vai ser calculada a porcentagem de palavras no e-mail que se enquadram nesta lista.

(e) Concentração de termos de *phishing*: Através de uma lista com termos comuns em e-mail do tipo *phishing*, vai ser calculada a porcentagem de palavras no e-mail que se enquadram nesta lista.

(f) Concentração de termos de mensagens seguras: Através de uma lista com termos comuns em e-mail considerados seguros, vai ser calculada a porcentagem de palavras no e-mail que se enquadram nesta lista.

Esses atributos no trabalho são denominados como atributos especiais.

3.2 MODELAGEM DA SOLUÇÃO

Um *dataset* formado por uma base de e-mails contendo mensagens de *phishing*, *spam* e seguras já classificadas vai ser dividido em dois conjuntos de dados:

(a) Treinamento: São os dados com os e-mails separados por categorias e que cuja informação a classe a que pertencem é fornecida ao sistema.

(b) Teste: São dados de e-mail cuja informação sobre a classe a que pertencem também é conhecida, mas é utilizada apenas para a avaliação dos resultados.

A partir dos dados de treinamento, o algoritmo das abelhas ABC recebe cada instância com o objetivo de encontrar uma relação entre os atributos da instância e a classe a que pertence e assim definir um conjunto de regras.

Com um conjunto de regras já definidos, estas regras vão ser aplicadas ao *dataset* de teste e assim produzir resultados que posteriormente serão utilizados para poder avaliar e melhorar a performance do algoritmo.

A figura 3.1 representa o processo de treinamento e teste proposto neste trabalho. A parte que se encontra dentro do pontilhado duplo representa a etapa de treinamento na qual os dados de entrada possuem a informação da classe a que pertencem. Nesta etapa o algoritmo ABC é aplicado com o objetivo de obter as regras para classificação. A parte que está dentro do pontilhado simples é a representação da fase de teste. A informação a respeito da classe é

desconhecida, as regras obtidas na etapa de treinamento são aplicadas com o objetivo de classificar os dados para posteriormente serem avaliados.

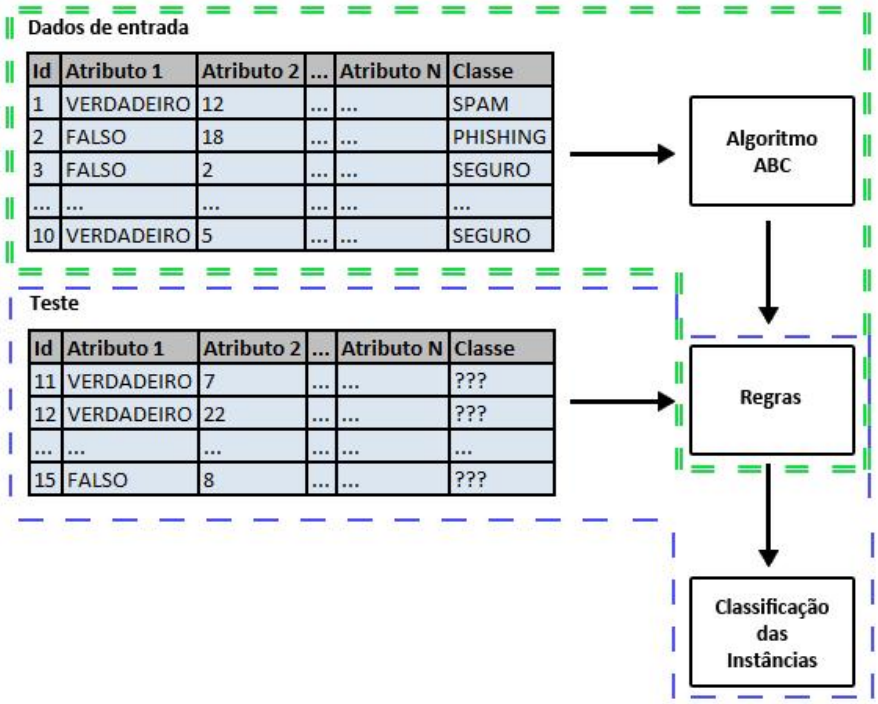


Figura 3.1: Modelo de classificação para o ABC.

A figura 3.2 apresenta o diagrama de classes do SCMIE. Nele observa-se que a classe InterfaceSCMIE representa a classe que realiza a interação com o usuário além de inicializar a aplicação. Através da interface o usuário poderá selecionar as seguintes opções:

(a) Treinamento, que irá disparar o método btn_TrainingInstances_Click();

(b) Teste, que irá dispara o método de btn_TestInstances_Click();

(c) Testar mensagem, que irá testar um única entrada através do método btn_TestMessage_Click().

Para realizar a comunicação com as classes e métodos que executam a simulação do algoritmo das abelhas para a classificação, esta classe irá possuir uma instância da classe Beehive.

A classe Beehive simula a colmeia, nela irão existir duas instâncias que representam as abelhas. Uma para a abelha forrageadora exploradora que é classe ScoutBee e outra para a abelha forrageadora trabalhadora representada pela classe WorkerBee. Além disso, irá haver coleções de regras do tipo List<Rule> que irão armazenar as regras obtidas pelas abelhas nas fontes de alimentos.

As abelhas irão explorar as fontes de alimentos na tentativa de encontrar ou aplicar regras. Estas regras são representadas pela classe Rule. Dentro desta classe existira uma lista de atributos que definirão o formato da regra. Esses atributos são do tipo lógico.

Para realizar a comunicação com os dados lidos de arquivos, a classe FoodSource possuíra um objeto do tipo Message que é composto por diversas propriedades obtidas através da classe MessageReader, a qual fará a leitura das mensagens e as separarás em listas conforme a sua origem.

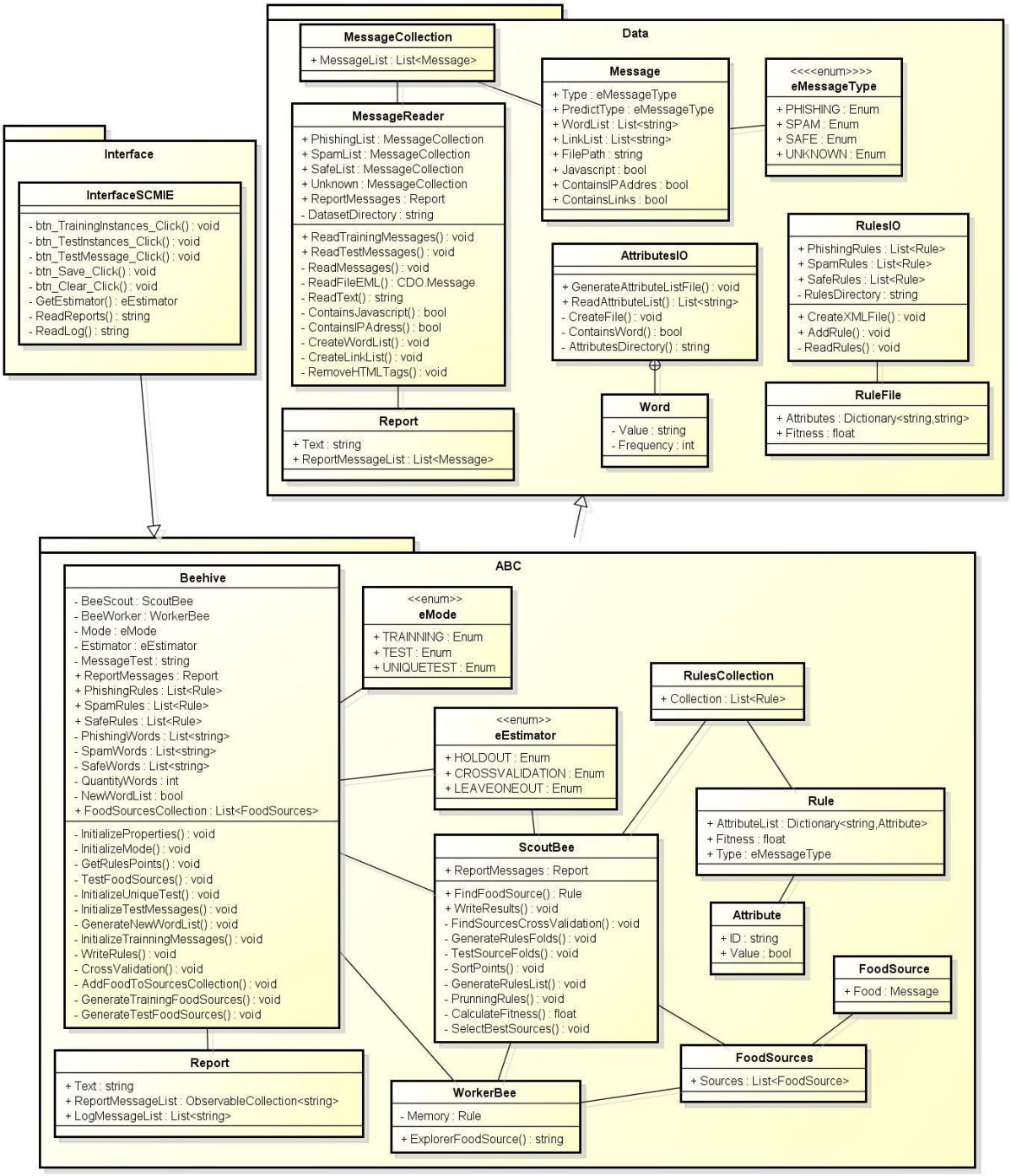


Figura 3.2: Diagrama de classes SCMIE.

A figura 3.3 apresenta o diagrama de atividade para a fase de treinamento da aplicação. Seguindo o fluxo da representação, as atividades são executadas como segue: Primeiro é realizada a leitura das mensagens selecionadas para o treinamento, mensagens esta que é necessário conhecer a classe a que pertencem. Sendo assim, elas serão agrupadas conforme a sua categoria. Se necessário uma nova lista de atributos é gerada.

O próximo passo é definir as fontes de alimentos e suas partições. Em seguida, a abelha exploradora é enviada para obter as regras que são utilizadas para a classificação das mensagens.

Após obter as regras, as partições são testadas para se produzir resultados para avaliação do treinamento. Esse processo é realizado até que todas as partições sejam avaliadas. Quando este processo termina, o conjunto de regras está definido.

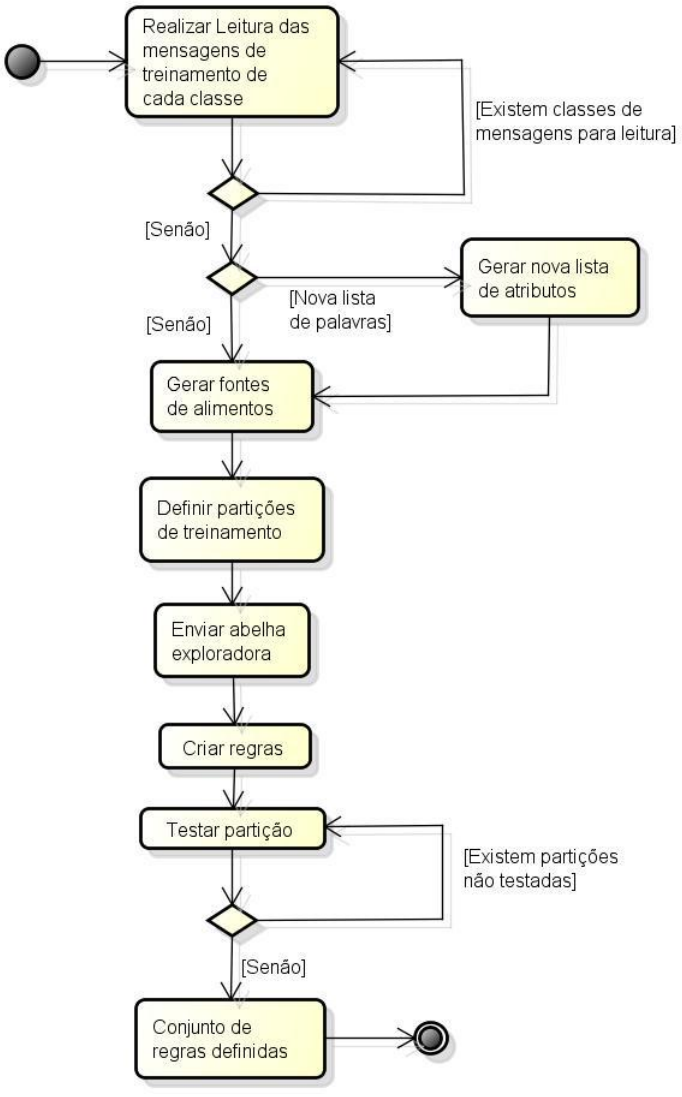


Figura 3.3: Diagrama de atividade para a fase de treinamento.

O diagrama da fase de teste é representado pela figura 3.4. Essa etapa inicia-se pela realização da leitura das regras e em seguida a leitura das mensagens.

Uma mensagem do *dataset* é selecionada para a classificação. Para cada classe de mensagens a abelha trabalhadora é enviada para realizar a classificação da mensagem. A abelha retorna uma pontuação para cada classe e assim define o tipo da mensagem como sendo da categoria que possuir a maior pontuação. Esse processo segue até que todas as mensagens para teste tenha sido selecionadas. Por fim o resultado dos testes é gerado.

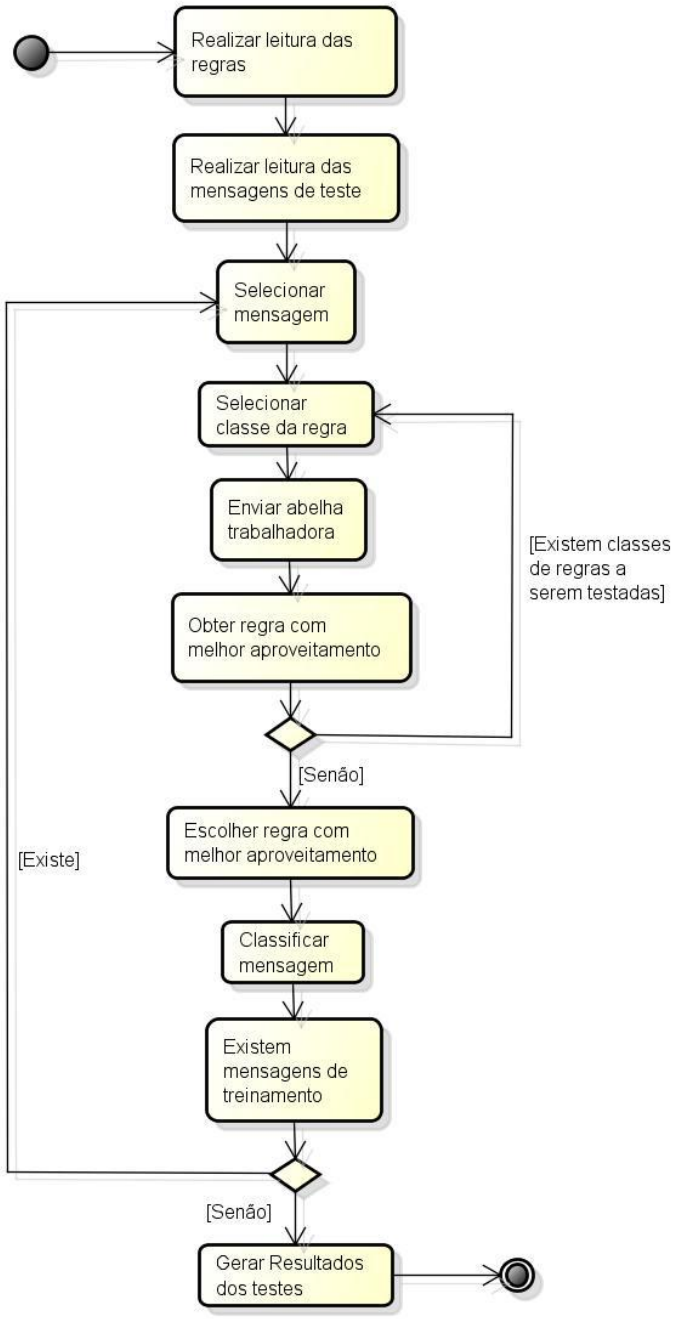


Figura 3.4: Diagrama de atividade para a fase de teste.

Através da interface é possível realizar o teste de uma mensagem única. A figura 3.5 representa este processo. Primeiramente a leitura das regras é realizada e logo após é criado um objeto do tipo mensagem que será a fonte de alimento. Como no processo de teste já apresentado, para cada classe de mensagens a abelha exploradora é enviada a fim de se descobrir de qual classe é obtida a maior pontuação e assim classificar a mensagem selecionada.

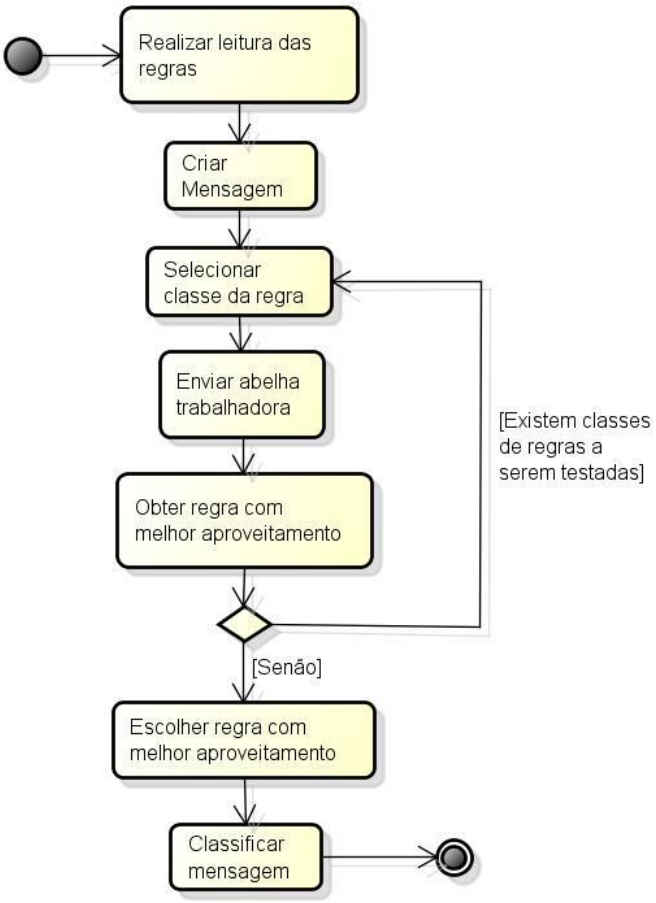


Figura 3.5: Diagrama de atividade para a fase de teste único.

3.3 ARQUITETURA

O projeto de arquitetura está preocupado com a compreensão de como um sistema deve ser organizado e com a estrutura geral desse sistema. O resultado do processo de projeto de arquitetura é um modelo de arquitetura que descreve como o sistema está organizado em um conjunto de componentes de comunicação (SOMMERVILLE 2011). As definições do projeto de arquitetura do sistema irão ser apresentadas nesta seção.

Para este trabalho será utilizado o padrão de arquitetura em camadas. Este padrão permite que os elementos possam ser trabalhados de forma separada e com maior independência. Cada camada é organizada com funcionalidades relacionadas à sua camada e cada uma fornece serviços à sua camada superior.

A figura 3.6 apresenta a organização das três camadas que serão utilizadas no trabalho. Estas camadas serão organizadas da seguinte forma:

(a) **Interface de usuário:** Nesta camada ficará toda a parte de interação do usuário com o software.

(b) **Algoritmo ABC:** Nesta camada estarão todas as classes e métodos que realizam a simulação do algoritmo de abelhas para classificação dos dados.

(c) **Leitura e escrita de dados:** Esta será a camada responsável por fazer a leitura das informações do *dataset*.

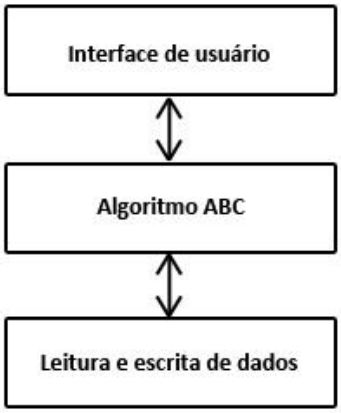


Figura 3.6: Representação do padrão de arquitetura do SCMIE.

3.4 DADOS DE ENTRADA

A base de dados a ser utilizada neste trabalho será composta por conjuntos de e-mails formados por mensagens do tipo *phishing*, *spam* e seguras que serão utilizadas para a etapa de treinamento e teste.

Os dados serão obtidos de bases públicas Nazário (Nazário, 2007) para mensagens de *phishing*. Para mensagens de *spam* e seguras os dados serão utilizados da base Ling-spam (Androutsopoulos, 2000) e também serão utilizadas mensagens da base SpamAssasin (Apache Software, 2003) com mais mensagens do tipo segura.

3.5 AVALIAÇÃO DOS RESULTADOS

Para avaliação do SCMIE serão aplicadas algumas métricas e métodos de amostragem. Para definir estas métricas e suas fórmulas de cálculo primeiramente é necessário entender alguns termos.

(a) Tuplas Positivas (P): Tuplas com exemplos da classe de informação que esta sendo testada.

(b) Tuplas Negativas (N): Tuplas com exemplos que pertencem a outras classes diferentes da que esta sendo testada.

(c) Verdadeiros Positivos (VP): Exemplos vindos de tuplas positivas e que foram corretamente classificados.

(d) Verdadeiros Negativos (VN): Exemplos vindos de tuplas negativas e que foram corretamente classificados.

(e) Falsos Positivos (FP): Exemplos vindos de tuplas negativas e que foram incorretamente classificados como positivos.

(f) Falsos Negativos (FN): Exemplos vindos de tuplas positivas e que foram classificados de forma incorreta.

A tabela 3.1 está representando a maneira como os dados serão classificados no caso de avaliação para os exemplos do tipo *phishing*.

Tabela 3.1: Matriz de confusão para a classificação de mensagens.

		Classes Previstas			
		PHISHING	**SPAM**	**SEGURA**	**TOTAL**
Classe atual	**PHISHING**	VP	FN	FN	P
	SPAM	FP	VN	VN	N
	SEGURA	FP	VN	VN	N
	TOTAL	P’	N’	N’	P + N

A partir das informações obtidas através da matriz de confusão, as métricas que serão utilizadas estão listadas na tabela 3.2.

Tabela 3.2: Métricas para avaliação do SCMIE (Han, 2012).

Métrica	Fórmula
Acerto: Taxa de reconhecimento.	$\frac{VP + VN}{P + N}$
Erro: Taxa de classificação incorreta.	$\frac{FP + FN}{P + N}$
Sensibilidade: Taxa de verdadeiros positivos.	$\frac{VP}{P}$
Especificidade: Taxa de verdadeiros negativos.	$\frac{VN}{N}$
Precisão.	$\frac{VP}{VP + FP}$

Para a avaliação dos resultados durante a etapa de treinamento serão utilizados métodos de amostragem. Os métodos destacados neste trabalho são: *holdout*, *leave-one-out* e *cross-validation*.

O estimador *holdout* divide exemplos em uma porcentagem fixa de exemplos p para treinamento e $(1 - p)$ para teste, considerando normalmente $p > 1/2$. Valores típicos são $p = 2/3$ e $(1 - p) = 1/3$, embora não existam fundamentos teóricos sobre estes valores.

Na figura 3.7 é representada graficamente essa proporção de exemplos utilizados para treinamento e para teste.

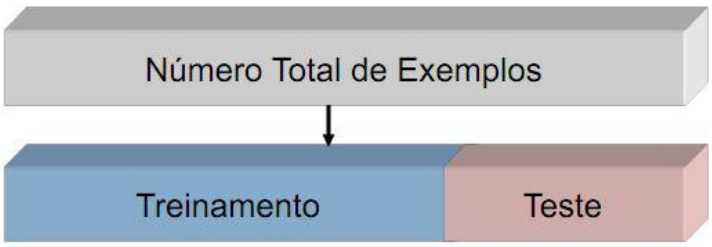


Figura 3.7: *Holdout* (Baranauskas, 2012).

O *cross-validation* é um estimador onde são geradas r-partições para validação cruzada. Os exemplos são aleatoriamente divididos em r partições mutuamente exclusivas de tamanho aproximadamente igual a n/r exemplos. Os exemplos nas (r – 1) partições são usados para treinamento e a hipótese induzida é testada na partição remanescente. Este processo é

repetido r vezes, cada vez considerando uma partição diferente para teste. O erro no *cross-validation* é a média dos erros calculados em cada um das r partições.

A figura 3.8 possui *r* partições, para cada execução uma partição é selecionada para teste, enquanto que todas as outras são utilizadas para treinamento. Cada partição pode ser utilizada somente uma vez para teste, mas para treinamento será utilizada *r* vezes.

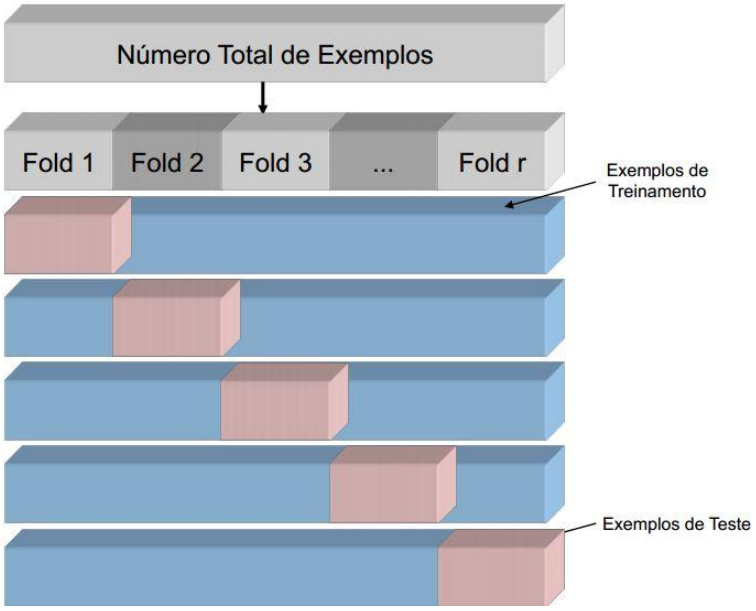


Figura 3.8: *Cross-validation* (Baranauskas, 2012).

O método *leave-one-out* é um caso especial de *cross-validation*. Ele é computacionalmente dispendioso e frequentemente usado em amostras pequenas. Para uma amostra de tamanho *n* uma hipótese é induzida utilizando ($n - 1$) exemplos; a hipótese é testada no único exemplo remanescente. Este processo é repetido *n* vezes, cada vez induzindo uma hipótese deixando de considerar um único exemplo. O erro é a soma dos erros em cada teste divido por *n*.

Este método é representado na figura 3.9 em que para cada exemplo é realizado o treinamento com todas as outras entradas e testado na única entrada selecionada. Esse processo é repetido até que todos os exemplos sejam testados.

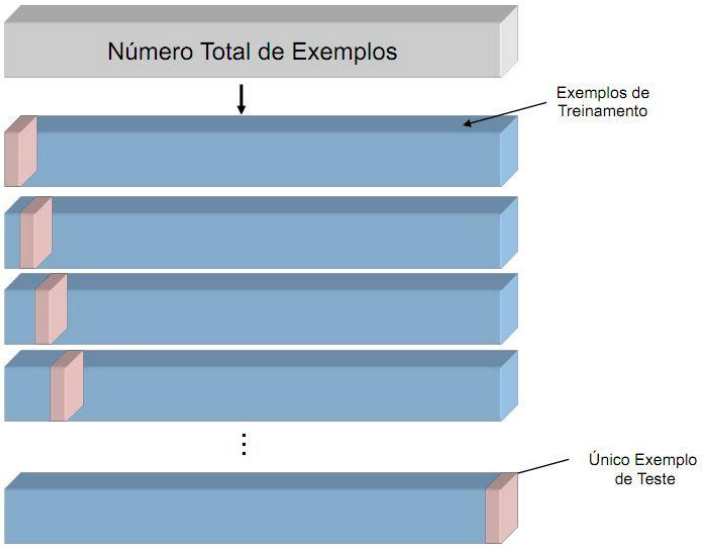


Figura 3.9: *Leave-one-out* (Baranauskas, 2012).

4 DESENVOLVIMENTO DA PROPOSTA DE SOLUÇÃO

O presente capítulo apresenta a implementação do sistema de classificação de mensagens baseado nas técnicas de Inteligência de Enxames (SCMIE). A implementação foi desenvolvida em linguagem orientada a objetos C# utilizando a versão 4 do framework .NET.

Nas seções seguintes são apresentados os principais conceitos implementados e criados à partir da utilização da metáfora das abelhas para resolver problemas de classificação de mensagens.

4.1 CRIAÇÃO DAS FONTES DE ALIMENTO

Os principais componentes que fazem parte da metáfora do algoritmo das abelhas são as abelhas exploradoras, as abelhas trabalhadoras e as fontes de alimentos. Nesta seção será apresentado como a ideia das fontes de alimentos foi abstraída para as mensagens de *e-mail*.

No sistema desenvolvido, o objeto que representa a fonte de alimento é a classe FoodSource. Esta classe possui uma propriedade chamada Food que é do tipo Message. Essa propriedade é definida no construtor do objeto não podendo sofrer alterações posteriormente. A figura 4.1 demonstra através de um diagrama de classe como a estrutura descrita está disposta na implementação.

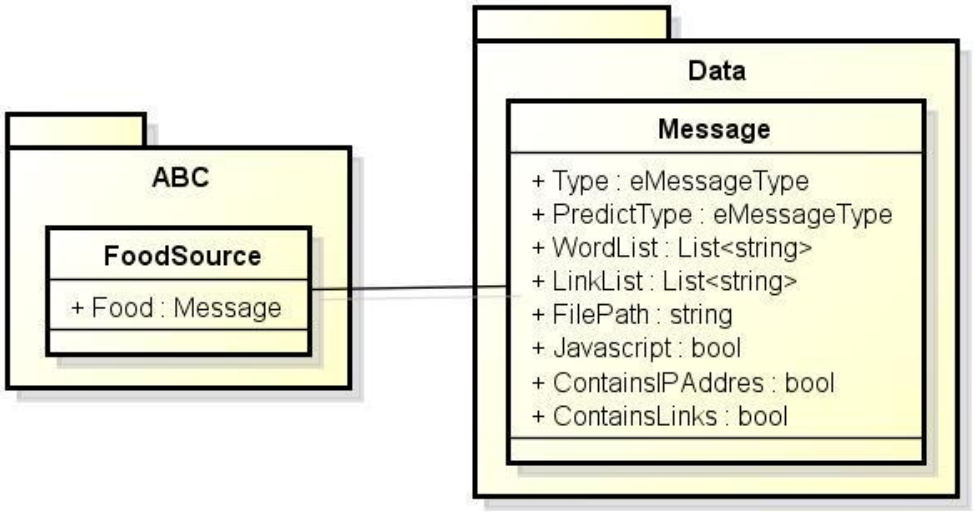


Figura 4.1: Diagrama com as classes que formam uma fonte de alimento.

As informações do objeto Message serão carregadas na etapa de leitura das

mensagens, dados estes que serão utilizados na geração das regras e na classificação das mensagens. Na tabela 4.1 são representadas todas as propriedades que compõem a classe Message, juntamente com o tipo de cada propriedade e a sua função.

Tabela 4.1: Propriedades que compõem o objeto Message.

Nome	**Tipo de Dado**	**Função**
Type	eMessageType	Representar a classe original a que a mensagem pertence.
PredictType	eMessageType	Representar a classe que foi prevista através das regras.
FilePath	string	Armazenar o caminho do arquivo da mensagem.
Javascript	bool	Informação se a mensagem contém código javascript.
ContainsIpAddress	bool	Informação se a mensagem contém links formados por endereço IP.
ContainsLinks	bool	Informação se a mensagem contém links.
WordList	List<string>	Lista formada pelas palavras da mensagem.
LinkList	List<string>	Lista formada pelos *links* que estão presentes na mensagem.

4.2 LISTA DE TERMOS FREQUENTES

Para construção dos atributos que compõem a regra são utilizadas listas contendo os termos mais frequentes para cada uma das classes de mensagens (*phishing*, *spam* e seguras). Esses termos são gerados com base nas mensagens que estão definidas no *dataset* de treinamento. Para a realização do processo de treinamento é necessário que esta lista esteja criada. Se necessário gerar os termos, esse processo será feito antes do treinamento.

Cada lista é composta pelos 300 termos mais frequentes encontrados em cada classe de mensagem. A lista com os termos aparece ordenada pela frequência de cada termo em ordem decrescente. A ordenação dos termos pela frequência possibilita que através da interface do sistema seja possível definir a quantidade de termos que se deseja utilizar para a

criação das regras, ou seja, o usuário pode definir um valor entre 1 e 300 termos.

4.3 DESCOBERTA DAS REGRAS

Uma das etapas mais importantes de todo o processo de classificação é a etapa da descoberta das regras. Para se obter bons resultados na classificação é importante que as regras estejam bem definidas e coesas.

Cada regra é formada por um conjunto de atributos cujos valores podem ser verdadeiros ou falsos. Além disso, utiliza-se também uma variável chamada *fitness* que possui um valor de aproveitamento da regra para o conjunto de mensagens testadas, valor este que pode que variar de 0 a 1, sendo 1 o melhor caso e 0 o pior. Este valor é utilizado para definir a aplicação ou não da regra.

Os atributos que compõem a regra são definidos através dos termos mais frequentes para cada classe de mensagem. Para cada palavra da mensagem usada no treinamento de uma classe, se existir uma palavra igual na lista de termos da classe em questão, vai ser criado um novo atributo com o valor definido como verdadeiro *(TRUE)*. Caso a palavra não esteja incluída na lista de termos da classe, o atributo não é adicionado à regra.

Além da lista de termos, também são utilizados os seguintes atributos:

(a) javascript: verdadeiro *(TRUE)* se a mensagem possuir código javascript, caso contrário falso *(FALSE)*.

(b) links: verdadeiro *(TRUE)* se a mensagem possuir links para conteúdo externo, caso contrário falso *(FALSE)*.

(c) containsIPAdress: verdadeiro *(TRUE)* se algum link possuir um URL formada por um endereço IP, caso contrário falso *(FALSE)*.

A figura 4.2 ilustra como as regras são formadas. No exemplo, as palavras em negrito no trecho da mensagem são aquelas que estão presentes na lista de termos da classe da mensagem. Há também um link encontrado que define como verdadeiro o atributo “links” e também o atributo “containsIPAdress” por esse link possuir um endereço IP. Já o atributo “javascript” não foi classificado com valor verdadeiro pela mensagem por não possuir código javascript embutido nela.

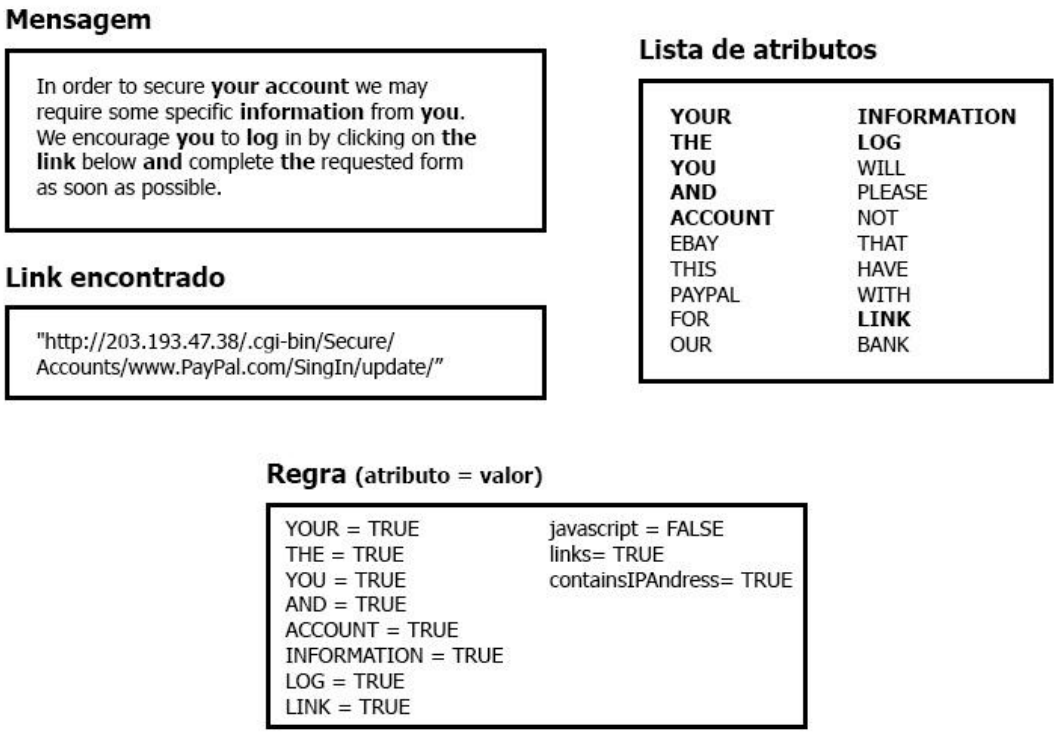


Figura 4.2: Representação da construção de uma regra.

4.4 AVALIAÇÃO DAS REGRAS

A classificação das mensagens se dá através da avaliação das regras que foram criadas nas etapas do treinamento. Essas regras são utilizadas para determinar a qual classe a mensagem que esta sendo avaliada pertence.

Em termo do algoritmo ABC, considera-se que a análise da mensagem é realizada pela abelha trabalhadora. Dentro da colmeia existem as regras que foram trazidas pelas abelhas exploradoras. Cada abelha trabalhadora memoriza uma regra e analisa a fonte de alimento, que no caso é a mensagem que deve ser classificada. A abelha que conseguir o maior índice de aproveitamento da regra que possui na memória, classificará a mensagem de acordo com a classe a que sua regra pertence.

Cada abelha trabalhadora executa o seguinte procedimento para definir o aproveitamento da regra que possui na memória:

(a) Define uma variável para pontuação com valor inicial em zero.

(b) Verifica se o valor de algum dos três atributos especiais definidos (javascript, contém link e link com endereço IP) da regra possuem o mesmo valor que a

mensagem. Em caso positivo a variável da pontuação recebe 2 pontos e em caso negativo não recebe pontuação nenhuma.

(c) Verifica todos os atributos termos da regra que a mensagem contém. Para cada palavra que estiver presente na mensagem, a variável de pontuação recebe 1 ponto, em caso negativo não recebe pontuação nenhuma.

(d) Após analisar todos os atributos é calculada a porcentagem da pontuação obtida.

Uma vez que todas as regras foram avaliadas, então são comparados os aproveitamentos obtidos e assim definida qual regra melhor se encaixa para a classificação da mensagem.

O processo do cálculo do aproveitamento de uma regra para uma mensagem é representando na figura 4.3. Pode se verificar como a quantidade de atributos é obtida e como os seus pesos são distribuídos. A mensagem do exemplo alcançou nove pontos de quatorze possíveis da regra. Sendo assim a taxa de compatibilidade foi de 64.28% para a regra aplicada.

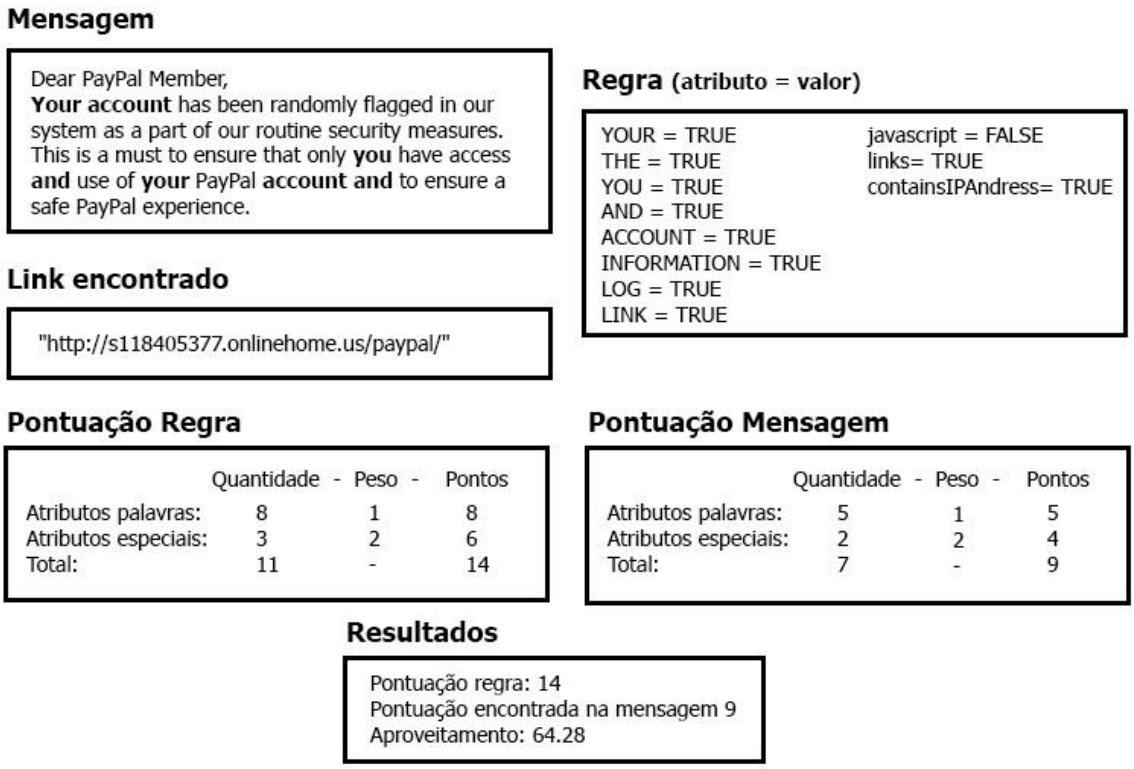


Figura 4.3: Representação da avaliação de uma regra.

4.5 TESTE E ANÁLISE DOS RESULTADOS

Para a criação dos *datasets* de treinamento e teste foram utilizadas mensagens das bases públicas Nazário (Nazário, 2007), Ling-spam (Androutsopoulos,2000) e SpamAssasin (Apache Software, 2003).

As mensagens foram selecionadas aleatoriamente e divididas da seguinte forma: 500 mensagens para cada uma das três classes de mensagens de treinamento e 200 mensagens para cada uma das três classes de teste.

4.5.1 CASO DE USO 1 – TREINAMENTO

Este primeiro cenário descreve o funcionamento da etapa de treinamento. Como resultado desta etapa é esperado que sejam criadas as regras que serão utilizadas para a classificação das mensagens. Ainda como saída destas etapas temos o relatório com os resultados do treinamento que são exibidos na interface do sistema e um relatório completo com a classificação para cada uma das mensagens, este que é gerado em um arquivo de texto.

A figura 4.4 ilustra a interface do sistema de classificação. Nesta figura estão numerados os passos para a realização da tarefa de treinamento que são os seguintes:

1) Botão “Training Data” dispara o início do processo de treinamento.
2) Opções para definição do estimador utilizado para o treinamento.
3) Variável “words atributes” estabelece a quantidade de atributos (termos) que a serem utilizados para o treinamento. O valor varia de 1 a 300.
4) Botão “Generate new list” quando selecionado cria novas listas de termos (atributos) com base nas mensagens que estão no *dataset* de treinamento. Um arquivo de texto para cada classe de mensagem é gerado. Se esta opção estiver marcada o tempo de execução do processo cresce.
5) Exibição do resultado do treinamento.
6) Botão “Save” permite salvar o relatório em um arquivo de texto.

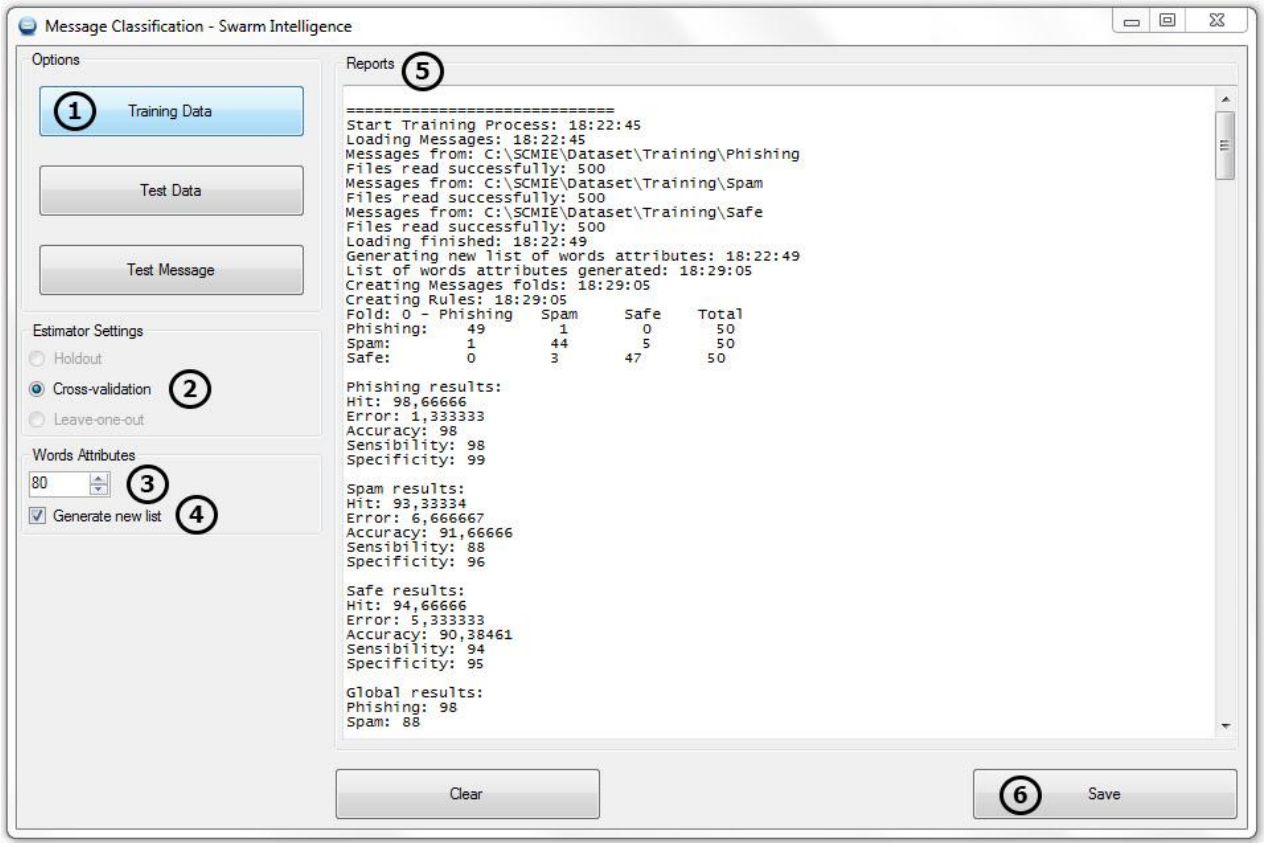


Figura 4.4: Interface do sistema de classificação - treinamento.

4.5.2 CASO DE USO 2 – TESTE DE DATASET

Este cenário descreve os passos para realizar o teste de conjuntos de mensagens. Como resultados são gerados o relatório com os resultados da classificação na interface do sistema, além do relatório completo salvo em um arquivo de texto com o resultado para cada uma das mensagens.

A figura 4.5 enumera os passos para se realizar a avaliação do *dataset* de teste. Os passos a serem seguidos são os seguintes:

1) Botão “Test Data” dispara o início do processo de teste do *dataset*.
2) Exibição do resultado do teste.
3) Botão “Save” permite salvar o relatório em um arquivo de texto.

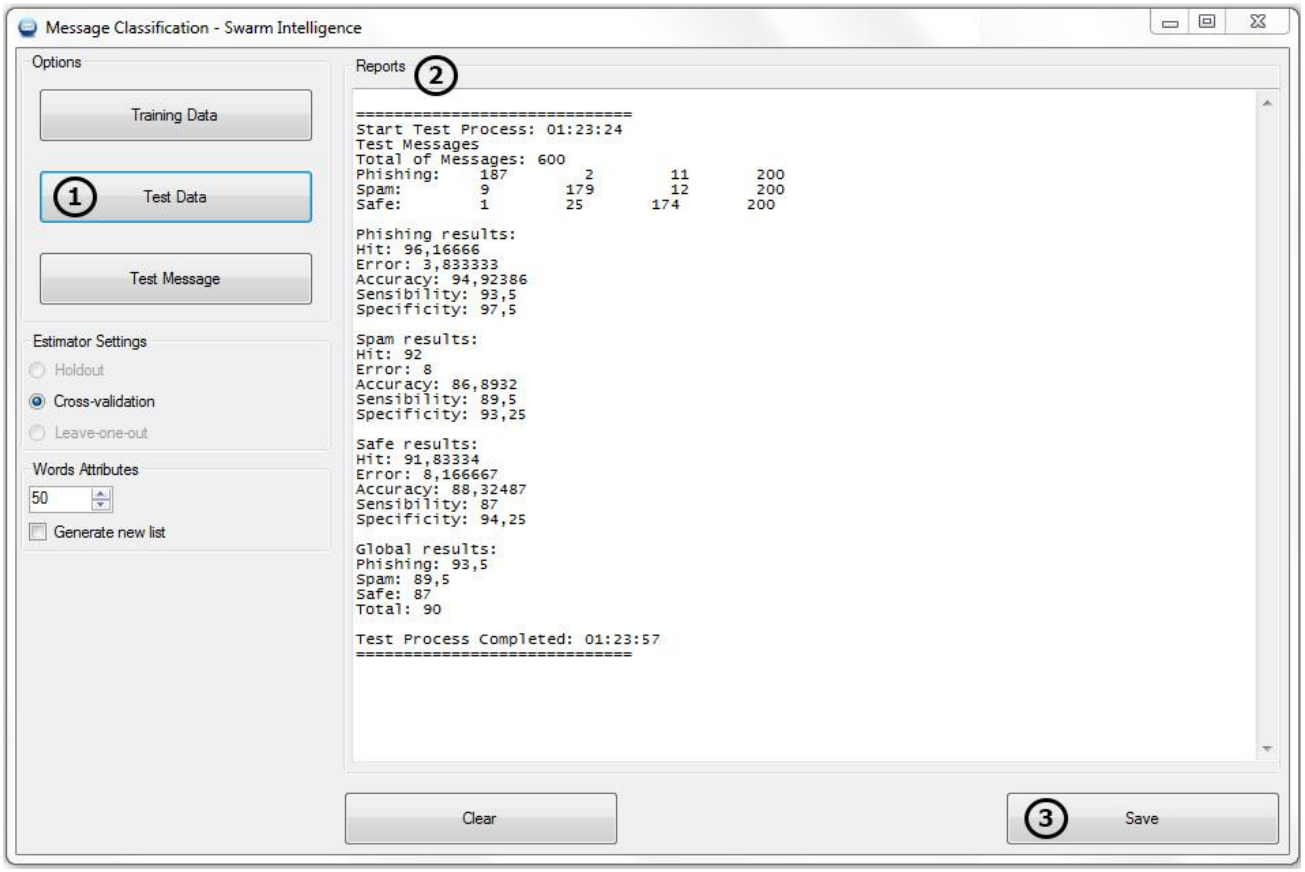


Figura 4.5: Interface do sistema de classificação – teste de *dataset*.

4.5.3 CASO DE USO 3 – TESTE DE MENSAGEM ÚNICA

Este cenário permite que seja feita a classificação de uma única mensagem através da seleção de um arquivo. O resultado desta etapa é classificação da mensagem selecionada.

Os passos para a realização da etapa estão enumerados na figura 4.6 e funcionam da seguinte maneira:

1) Botão “Test Message” abre uma janela de seleção de arquivo para escolher o arquivo que contém a mensagem. Após o arquivo ser selecionado é iniciado o processo de classificação da mensagem.
2) Exibição do resultado da classificação da mensagem.
3) Botão “Save” permite salvar o relatório em um arquivo de texto.

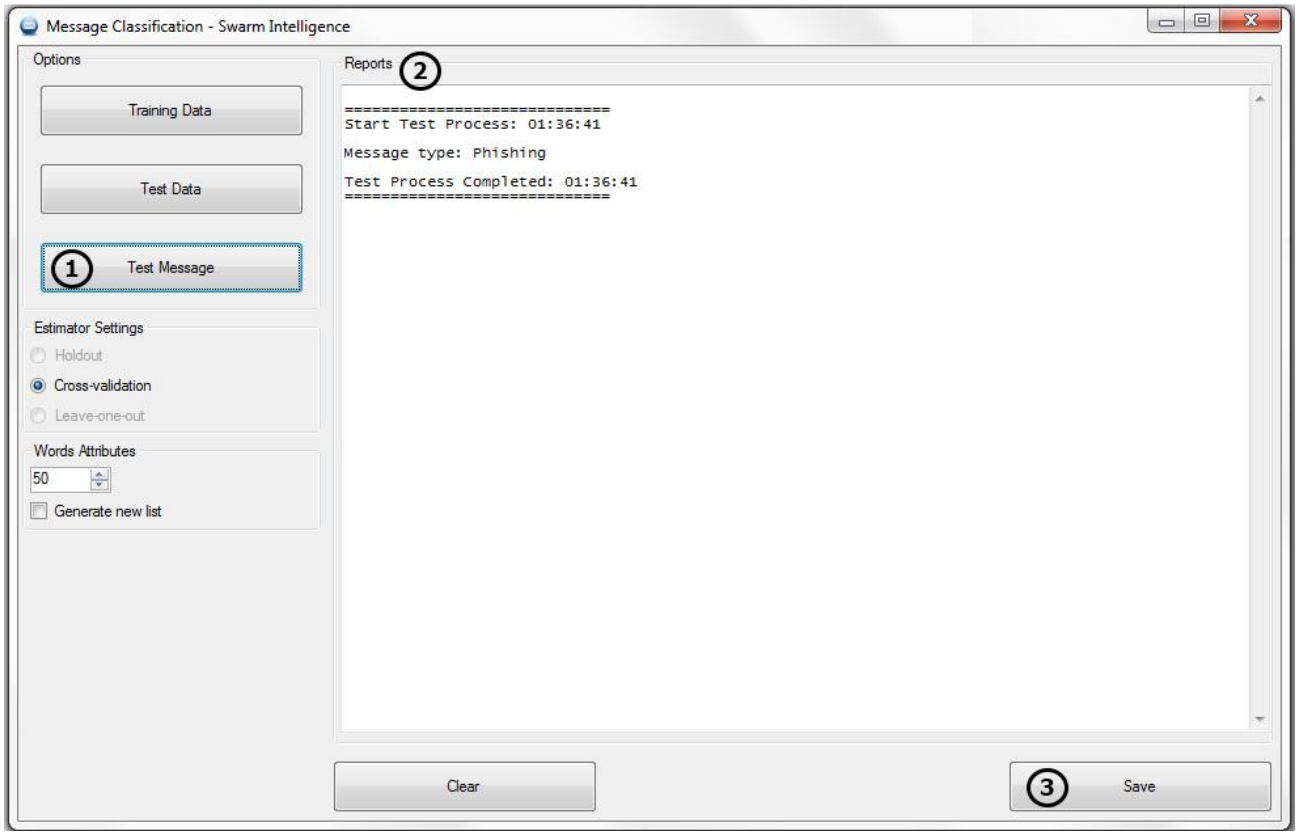


Figura 4.6: Interface do sistema de classificação – teste único.

4.6 ANÁLISE DOS RESULTADOS

Esta seção apresenta os resultados da avaliação do *dataset* de teste com base nas regras produzidas na etapa de treinamento. Para isso foram gerados casos com a quantidade de termos diferentes para comparação e utilizando o estimador *cross-validation*. A configuração que obteve o melhor resultado foi utilizada para comparação com outras ferramentas.

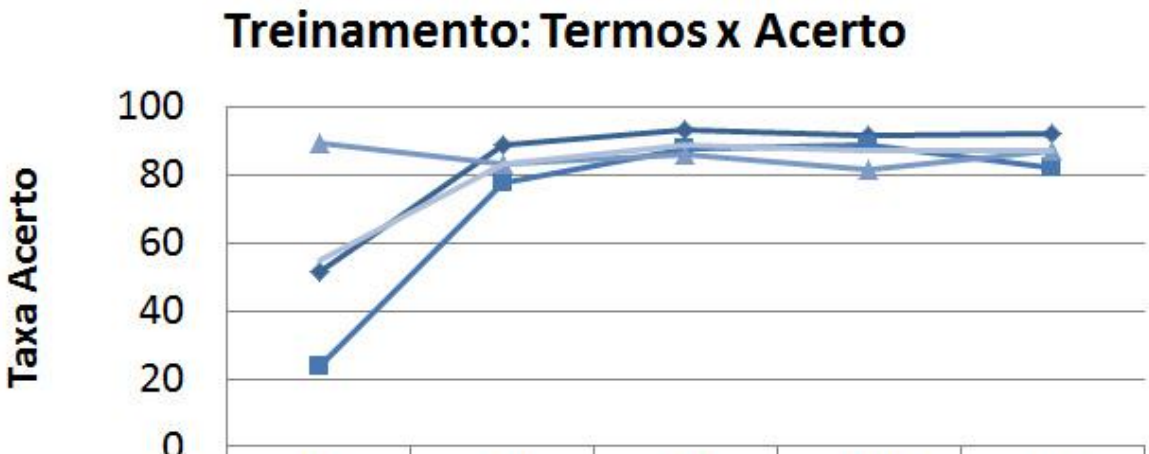
O gráfico da figura 4.7 apresenta os resultados obtidos na etapa de treinamento. Ele faz relação entre a quantidade de termos (eixo x) com a taxa de acerto (eixo y). O valor de acerto para cada uma das classes *phishing*, *spam* e segura é o valor médio da taxa de acerto de todas as partições de treinamento.

Para encontrar os resultado exibidos no gráfico foi gerado o processo de treinamento 5 vezes. Em cada uma das execuções a opção “words atributes” da figura 4.4 foi configurada respectivamente com os seguintes valores: 20, 50, 80, 100 e 150.

Para cada um desses valores testado a taxa de mensagens corretamente classificadas

pode ser vista na tabela junto ao gráfico da figura 4.7.

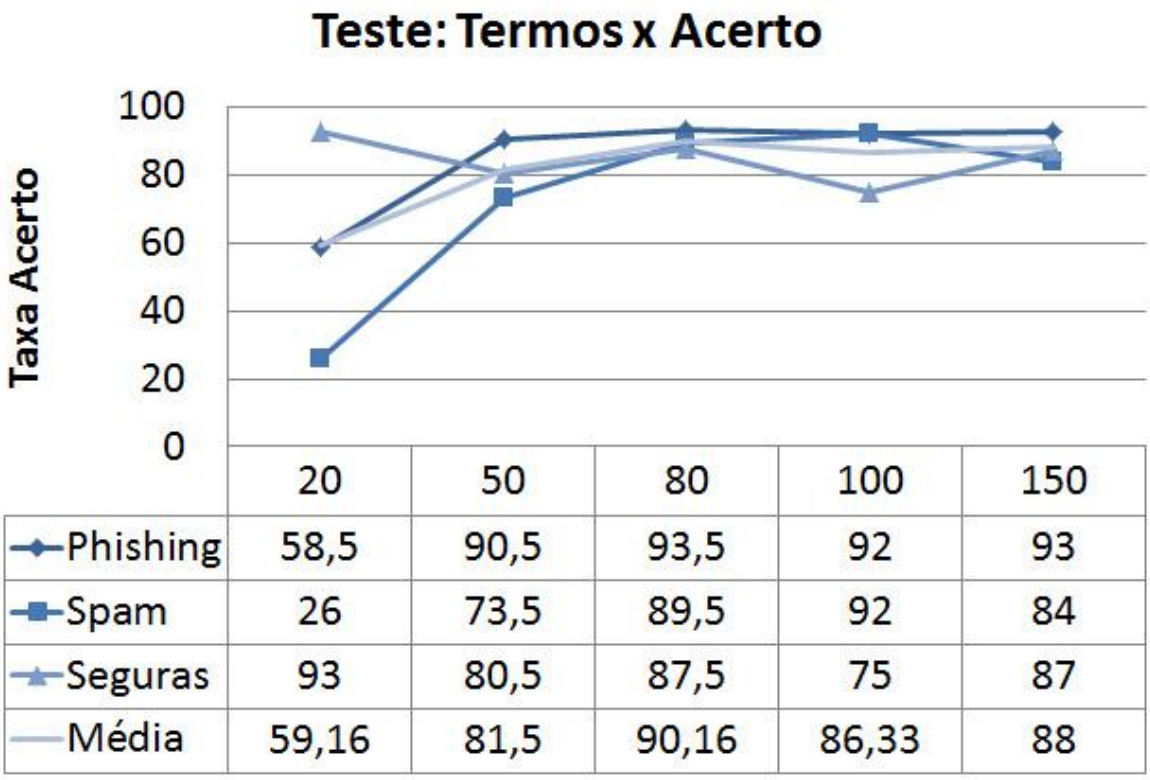
O melhor resultado obtido foi no caso em que foram utilizados 80 termos, onde a taxa de acerto das mensagens de *phishing* ficou em 93,4%, *spam* 87,6% e seguras 86% o que da uma média de 89% de mensagens corretamente classificadas.



	20	50	80	100	150
Phishing	51,4	88,6	93,4	91,4	92,4
Spam	23,6	77,6	87,6	88,6	82
Seguras	89,4	83,4	86	81,4	87,2
Média	54,8	83,2	89	87,13	87,2

Figura 4.7: Gráfico do treinamento - Quantidade termos x Taxa de acerto.

A figura 4.8 apresenta o gráfico da relação da quantidade de termos utilizados e a sua taxa de acerto para a etapa de teste. Observando-se os gráficos das figuras 4.7 e 4.8 é possível perceber que nos dois casos há um padrão entre a taxa de acerto e a quantidade de termos. Pode ser verificado que nos dois exemplos a taxa de acerto cresce até a utilização de 80 termos, e a partir de então ela começa a sofrer uma diminuição. Assim como no gráfico de treinamento o melhor resultado também foi obtido utilizando-se 80 termos. Por este motivo a configuração com 80 termos foi escolhida para realizar as comparações com outros algoritmos.



	20	50	80	100	150
Phishing	58,5	90,5	93,5	92	93
Spam	26	73,5	89,5	92	84
Seguras	93	80,5	87,5	75	87
Média	59,16	81,5	90,16	86,33	88

Figura 4.8: Gráfico do teste - Quantidade termos x Taxa de acerto.

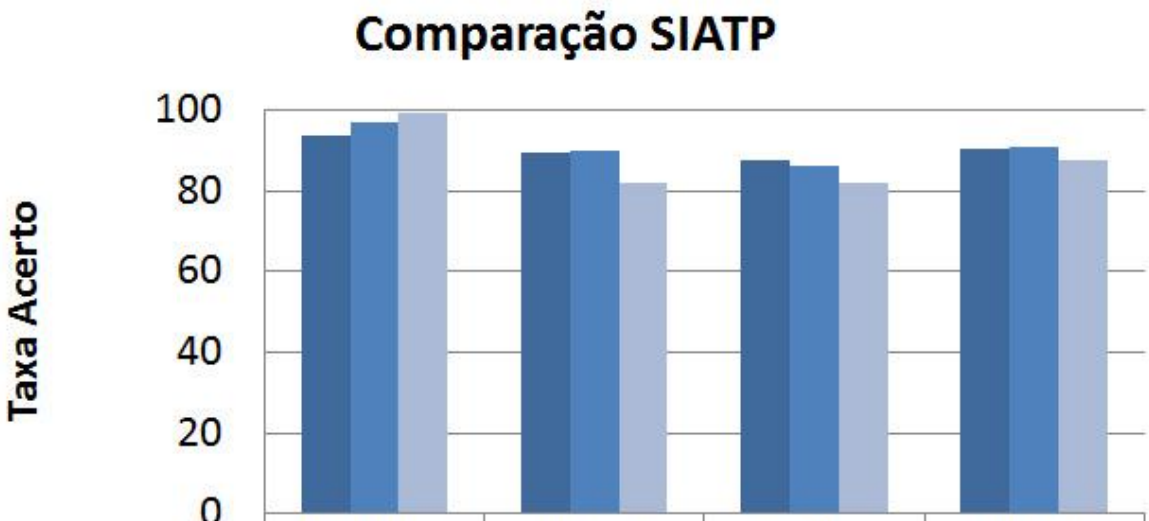
Utilizando a configuração de 80 termos, os resultados obtidos na etapa de teste podem ser visualizados na tabela 4.2 através da matriz de confusão criada. Para o teste foram utilizadas 600 mensagens divididas igualmente entre as categorias de *phishing*, *spam* e seguras. A taxa de acerto para a classe de *phishing* foi 93,5%, para *spam* 89,5% e segura 87,5%. A média da taxa de acerto destes valores ficou em 90,16%.

Tabela 4.2: Matriz de confusão dos resultados do teste.

	Classes Previstas			
	PHISHING	**SPAM**	**SEGURA**	**TOTAL**
PHISHING	187	2	11	200
SPAM	9	179	12	200
SEGURA	1	24	175	200
TOTAL	197	205	198	600

A figura 4.9 compara o desempenho do SCMIE com outro sistema de classificação de mensagem que utiliza metodologia imunológica baseada na Teoria do Perigo. O sistema

imunológico utilizado para comparação é denominado SIATP (Sistema Imunológico Artificial Baseado na Teoria do Perigo) e foi desenvolvido por Gomes (2010). A comparação do sistema é feita ainda com uma versão do SIATP integrada (SIATP int.) com servidor de e-mail (Guimarães, 2012). O gráfico realiza a comparação da taxa de acerto do SCMIE para cada uma das categorias com resultados obtidos nos trabalho de Gomes (2010) e Guimarães (2012).



	Phishing	Spam	Safe	Média
SCMIE	93,5	89,5	87,5	90,16
SIATP	97	90	86	91
SIATP (int.)	99,05	81,73	81,76	87,51

Figura 4.9: Comparação do ABC com SIATP.

Para efeito de comparação também é utilizado o trabalho de Hepp (2010) que realiza uma análise dos principais algoritmos de classificação de dados para a classificação de mensagens de *phishing*.

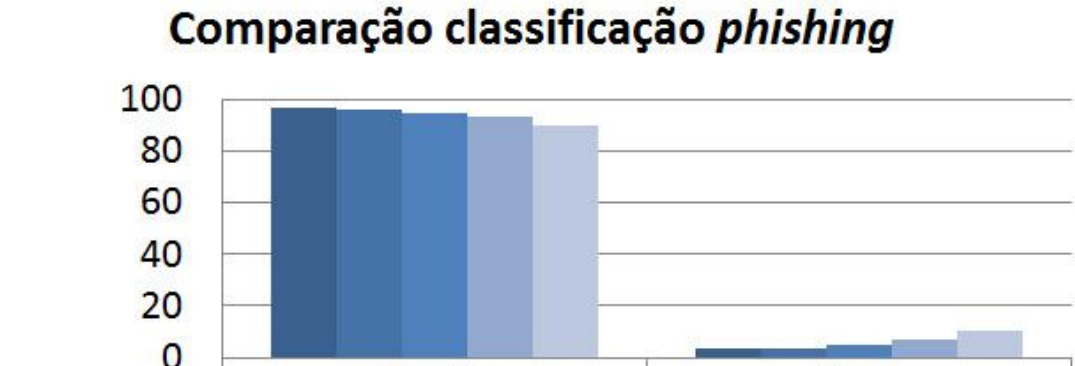
A fim de se realizar a comparação com o trabalho de Hepp, as mensagens de *spam* são desconsideradas, o que resultou nos dados exibidos na matriz de confusão da tabela 4.3 com somente os valores de *phishing* e segura. Do total de 198 mensagens de *phishing*, 187 foram classificadas corretamente o que gerou uma taxa de acerto 96,79%. As mensagens de *phishing* classificadas como seguras foram 11 o que gerou uma taxa de erro de 3,2%.

Tabela 4.3: Matriz de confusão desconsiderando resultados de *spam*.

	Classes Previstas		
	PHISHING	**SEGURA**	**TOTAL**
PHISHING	187	11	198
SEGURA	1	175	176
TOTAL	188	186	374

Hepp analisou os algoritmos *Multilayer Perceptron* (MP), J48, *BayesNet* e E-M para o problema de classificação de mensagens. Para cada um destes algoritmos Hepp utilizou um mesmo *dataset* formado por 514 mensagens obtidas de bases de dados públicas e selecionadas de forma aleatória. Das 514 mensagens 122 foram representadas por mensagens de *phishing*, e 392 mensagens consideradas seguras. Foram ainda aplicadas configurações específicas dos algoritmos que estão definidas em seu trabalho.

A figura 4.10 ilustra os resultados da comparação do SCMIE com os algoritmos citados através da taxa de acerto de mensagens corretamente classificadas e da taxa de erro, formada por mensagens classificadas de forma errada. A taxa de mensagens do SCMIE classificadas corretamente teve um resultado satisfatório, sendo levemente melhor que o MP que havia obtido o melhor resultado de mensagens classificadas de forma correta.



	Acerto	Erro
SCMIE	96,79	3,2
MP	96,5	3,5
J48	94,94	5,06
BayesNet	93,19	6,81
E-M	89,69	10,31

Figura 4.10: Comparação SCMIE com outros algoritmos.

5 CONCLUSÃO

Este capítulo apresenta as conclusões do trabalho de forma em que é apresentada uma síntese do trabalho, considerações sobre os resultados obtidos e possíveis melhorias futuras.

5.1 SINTESE DO TRABALHO

O trabalho foi feito baseado no estudo de técnicas de Inteligência Artificial inspiradas no comportamento natural de insetos, como as abelhas que possuem pouca inteligência individualmente, mas que coletivamente são capazes de resolver problemas complexos.

Através do levantamento bibliográfico foi possível perceber que existem diversas técnicas baseadas no comportamento das abelhas e que estas técnicas obtiveram resultados satisfatórios sendo utilizadas para resolver problemas de diferentes naturezas como otimização, *clustering*, classificações de dados, entre muitos outros problemas.

Dentre estas áreas pode-se destacar o trabalho de Shukran (et al., 2011) que utiliza o algoritmo das abelhas para classificação de dados. Este trabalho foi umas das principais referências utilizadas para a construção da solução, mostrando-se uma técnica eficiente e versátil que pode ser aplicada para a classificação de outros tipos de dados.

O objetivo deste trabalho foi aplicar uma solução baseada no comportamento natural das abelhas para a classificação de mensagens de *phishing*, *spam* e mensagens seguras.

Para compreender melhor a forma como os algoritmos de abelhas podem ser aplicados para o problema de classificação de mensagens, foi realizado um estudo sobre como as abelhas comportam-se na natureza para encontrar as melhores fontes de alimentos e passar essa informação a colmeia. Inspirado neste comportamento biológico das abelhas foram apresentando alguns algoritmos construídos a partir desta metáfora que são capazes de resolver problemas como otimização, *clustering* e classificação de dados.

Para a criação do projeto de desenvolvimento foi definido o problema da classificação de mensagens e utilizando a metáfora das abelhas foi criada a modelagem do problema juntamente com a sua arquitetura. Além disso, foram determinados os dados de entrada e como a avaliação dos resultados é feita.

Após os conceitos serem estudados e a modelagem ser definida, foi realizada a implementação do ABC para se classificar de mensagens de *phishing*, *spam* e mensagens seguras. Como resultado da implementação é esperado que o sistema retorne resultados de classificação de mensagens com base em regras que ele mesmo cria. Os resultados do sistema são usados para se fazer a análise do desempenho da ferramenta.

5.2 RESULTADOS

Para realizar a avaliação do desempenho do SCMIE foram utilizados como forma de comparação resultados apresentados através de outros estudos feitos em relação à classificação de mensagens.

Os dados utilizados para comparação foram gerados a partir de um *dataset* contendo mensagens de *phishing*, *spam* e seguras. Esses resultados foram processados através da etapa de teste do sistema desenvolvido.

Considerando-se que esta é uma primeira versão do sistema e a técnica do algoritmo das abelhas não é tão popularmente utilizada para classificação de dados, os resultados podem ser considerados como altamente satisfatórios, já que o índice das mensagens classificadas corretamente foram resultados próximos a outros algoritmos de classificação, sendo inclusive superiores na em alguns casos.

Através da modelagem e do desenvolvimento do sistema foi possível perceber que a utilização do algoritmo das abelhas pode ser adaptado a outros áreas de classificação de dados assim como foi feito neste trabalho. Em contrapartida, a utilização de uma quantidade de dados muito grande para o treinamento pode gerar um custo computacional muito elevado, o que poderia vir a inviabilizar a utilização do processo.

Observando os pontos positivos e negativos que foram destacados, fica claro que o sistema tem muito a evoluir ainda através de melhorias que possam proporcionar o aumento do seu desempenho e da sua estabilidade.

5.3 TRABALHOS FUTUROS

Durante o desenvolvimento do trabalho forma identificadas algumas melhorias que

podem vir a contribuir para uma melhora do desempenho do sistema de classificação. As melhorias identificadas são as seguintes:

(a) Implementação dos estimadores Leave-one-out e Holdout. Para este trabalho foi utilizado o estimador Cross-validation por possuir um embasamento teórico melhor em relação ou Holdout e por ser teoricamente mais rápido que o Leave-one-out.

(b) Criação de novos atributos especiais para as regras. Os atributos definidos com especiais são os atributos “javascript”, “links” e “containsIPAdress”. Outros atributos como, por exemplo, se a mensagem possui imagens ou código html poderiam contribuir para regras mais completas.

(c) Avaliação das regras. Quando uma regra é aplicada a uma mensagem, para verificar o nível de compatibilidade da regra com a mensagem cada atributo possui um peso. Este peso poderia ser definido através da interface do sistema com o usuário.

(d) Lista com palavras de parada. Implentação de uma lista com palavras de parada para eliminar da lista de termos preposições e artigos.

6 REFERÊNCIAS

ANDROUTSOPOULOS, I. Ling-spam. Dispon__vel em: < http://csmining.org/index.php/ling-spam-datasets.html/> Acesso em: setembro 2010.

ANTISPAM.BR. **O que é spam?** Disponível em: <http://www.antispam.br/conceito/>. Acesso em 19 mar. 2012.

APACHE SOFTWARE. Spamassassin public mail corpus. Dispon__vel em: <http://spamassassin.apache.org/publiccorpus/> Acesso em: junho de 2012.

Artificial Bee Colony (ABC) Algorithm Homepage. **Artificial Bee Colony (ABC) Algorithm**. Disponível em: < http://mf.erciyes.edu.tr/abc/>. Acesso em 22 mai. 2012.

BAHAMISH, H. A. A.; ABDULLAH, R.; SALAM, R. A. **Protein Tertiary Structure Prediction Using Artificial Bee Colony Algorithm. Paper Apresented at the Modelling & Simulation** – AMS ´09. Third Asia International Conference on Modelling & Simulation, 2009.

BARANAUSKAS, J. A. **Métodos de Amostragem e Avaliação de Algoritmos** – Notas de aula. Dísponivel em: <http://professor.ufabc.edu.br/~ronaldo.prati/MachineLearning/AM-I-Metodos-Amostragem.pdf> Departamento de Física e Matemática – USP Universidade de São Paulo, 2012.

BONABEAU, Eric; DORIGO, Marco. **Swarm Intelligence: From Natural to Artificial Systems** – www.cs.virginia.edu/~evans/bio/slides/presentation.pdf, 2003.

BROWNLEE, Jason. **Clever Algorithms - Nature-Inspired Programming Recipes** – 1ª. Ed. LuLu, 2011.

CAIS, Centro de Atendimento a Incidentes de Segurança **– Catálogo de Fraudes – Autoatendimento Banco do Brasil.** Disponível em: <http://www.rnp.br/cais/fraudes.php?id=17001&ano=&mes=&pag=1&busca=&tag_extend=&tag=>. Acesso em 09 abr. 2012.

CERVO, Amado Luiz; BERVIAN, Pedro Alcino; DA SILVA, Roberto. **Metodologia Científica** – 6. Ed. São Paulo: Pearson, 2007.

GOMES, Camila. **Metodologia Imunológica Baseada na Teoria do Perigo para o Desenvolvimento de Sistema de Diagnóstico** – Trabalho de conclusão de curso, Universidade de Caxias do Sul, 2010.

GUIMARÃES, Bruno B. **Relatório de bolsa de iniciação científica** – Universidade de Caxias do Sul, 2012.

HAN, Jiawei; KAMBER, Micheline; PEI, Jian. **Data Mining Concepts and Techniques** – Morgan Kaufmann publications, USA, 2012.

HEPP, Felipe S. **Aprendizagem Automática de *Phishing* por Mineração de Dados** – Trabalho de conclusão de curso, Universidade de Caxias do Sul, 2010.

JAIN, A.K.; MURTY, M.N.; FLYNN, P.J. **Data Clustering: A Review** – ACM Comput.Surveys 31 (3) (1999) 264-323.

JAMES, Lance. **Phishing Exposed** – Rockland, MA, USA: Syngress, 2005.

KARABOGA, D. **An Idea Based on Honey Bee Swarm for Numerical Optimization.** (TECHNICAL REPORT-TR06), 2005.

KARABOGA, D.; AKAY, B.; OZTURK, C. **Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks Modeling Decisions for Artificial Intelligence** – (pp- 318.329), 2007.

KARABOGA, Dervis; OZTURK, Celal. **A Novel Clustering approach: Artificial Bee Colony (ABC) Algorithm** – Elsevier B.V. 2009.

KARABOGA, Dervis; OZTURK, Celal. **Fuzzy Clustering With Artificial Bee Colony Algorithm** – Scientific Research and Eassays Vol. 5(14), pp. 1899-1902, 2010.

KARABOGA, N. **A New Design Method Based on Artificial Bee Colony Algorithm for Digital IIR Filters.** – Journal of the Franklin Institute, 346(4), 328-348.

Kaspersky Lab. Disponível em: <http://www.kaspersky.com>. Acesso em 26 mar. 2012.

KUMBHAR, Pravin Yallappa; KRISHNAN, Shoba. **Use of Artificial Bee Colony (ABC) Algorithm in Artificial Neural Network Synthesis** – International Journal of Advanced Engineering Sciences and Technologies, Vol No. 11, Issue No. 1, 162 – 171, 2011.

LIANYING, Zhou; FENGYU, Liu. **A Swarm-Intelligence-Based Intrusion Detection Technique** – IJCSNS, Vol. 6, No. 7B, 2006.

NAZARIO, J. Phishing corpus. Dispon__vel em: <http://monkey.org/~jose/blog/viewpage.php?page=phishing_corpus> Acesso em: junho 2012.

ROUILLARD, Meghan. **Polarization Sensitivity: A Strong and Weak Sense** – Larouche Pac. Disponível em: <http://larouchepac.com/node/17209>. Acesso em 14 abr. 2012.

SHUKRAN, Mohd Afizi Mohd; CHUNG, Yuk Ying; YEH, Wei-Chang; WAHID, Noorhaniza; ZAIDI, Ahmad Mujahid Ahmad. **Artificial Bee Colony based Data Mining Algorithms for Classification Tasks** – Published by Canadian Center of Science and Education, 2011.

SOMMERVILLE, Ian. **Engenharia de Software** – 9. Ed. São Paulo: Pearson, 2011.

Spam report: February 2012. Disponível em: <http://www.securelist.com/en/analysis/204792224/Spam_report_February_2012>. Acesso em 26 mar. 2012.

TAN, Pang-Ning; STEINBACH, Michael; KUMAR, Vipin. **Introduction to Data Mining** – Addison-Wesley, 2005.

WHITE, Tony. **Swarm Intelligence: A Gentle Introduction with application** – www.sce.carleton.ca/researchers/tony/index.html, 1997.

ANEXO A

```
public class test {
        static beeColony bee = new beeColony();

        public static void main(String[] args) {
                int iter = 0;
                int run = 0;
                int j = 0;
                double mean = 0;
                // srand(time(NULL));
                for (run = 0; run < bee.runtime; run++) {
                        bee.initial();
                        bee.MemorizeBestSource();
                        for (iter = 0; iter < bee.maxCycle; iter++) {
                                bee.SendEmployedBees();
                                bee.CalculateProbabilities();
                                bee.SendOnlookerBees();
                                bee.MemorizeBestSource();
                                bee.SendScoutBees();
                        }
                        for (j = 0; j < bee.D; j++) {
                                //                      System.out.println("GlobalParam[%d]:
%f\n",j+1,GlobalParams[j]);
                                System.out.println("GlobalParam[" + (j + 1) + "]:"
                                                + bee.GlobalParams[j]);
                        }
                        // System.out.println("%d. run: %e \n",run+1,GlobalMin);
                        System.out.println((run + 1) + ".run:" + bee.GlobalMin);
                        bee.GlobalMins[run] = bee.GlobalMin;
                        mean = mean + bee.GlobalMin;
```

```
                }
                mean = mean / bee.runtime;
                // System.out.println("Means of %d runs: %e\n",runtime,mean);
                System.out.println("Means  of " + bee.runtime + "runs: " + mean);
        }
    }


import java.lang.Math;

public  class beeColony {

    /* Control Parameters of ABC algorithm*/
    int NP=20; /* The number of colony size (employed bees+onlooker bees)*/
    int FoodNumber = NP/2; /*The number of food sources equals the half of the colony size*/
    int limit = 100;  /*A food source which could not be improved through "limit" trials is abandoned by its employed bee*/
    int maxCycle = 2500; /*The number of cycles for foraging {a stopping criteria}*/

    /* Problem specific variables*/
    int D = 100; /*The number of parameters of the problem to be optimized*/
    double lb = -5.12; /*lower bound of the parameters. */
    double ub = 5.12; /*upper bound of the parameters. lb and ub can be defined as arrays for the problems of which parameters have different bounds*/

    int runtime = 30;  /*Algorithm can be run many times in order to see its robustness*/

    int dizi1[]=new int[10];
    double Foods[][]=new double[FoodNumber][D];       /*Foods is the population of food sources. Each row of Foods matrix is a vector holding D parameters to be optimized. The number of rows of Foods matrix equals to the FoodNumber*/
    double f[]=new double[FoodNumber];       /*f is a vector holding objective function values associated with food sources */
    double fitness[]=new double[FoodNumber];      /*fitness is a vector holding fitness (quality) values associated with food sources*/
    double trial[]=new double[FoodNumber];       /*trial is a vector holding trial numbers through which solutions can not be improved*/
    double prob[]=new double[FoodNumber];         /*prob is a vector holding probabilities of food sources (solutions) to be chosen*/
    double solution[]=new double[D];           /*New solution (neighbour) produced by v_{ij}=x_{ij}+\phi_{ij}*(x_{kj}-x_{ij}) j is a randomly chosen parameter and k is a randomlu chosen solution different from i*/

    double ObjValSol;            /*Objective function value of new solution*/
```

```
        double FitnessSol;              /*Fitness value of new solution*/
        int neighbour, param2change;                    /*param2change corrresponds to j, neighbour
corresponds to k in equation v_{ij}=x_{ij}+\phi_{ij}*(x_{kj}-x_{ij})*/

        double GlobalMin;                       /*Optimum solution obtained by ABC algorithm*/
        double GlobalParams[]=new double[D];                    /*Parameters of the optimum
solution*/
        double GlobalMins[]=new double[runtime];
              /*GlobalMins holds the GlobalMin of each run in multiple runs*/
        double r; /*a random number in the range [0,1)*/

        /*a function pointer returning double and taking a D-dimensional array as argument */
        /*If your function takes additional arguments then change function pointer definition
and lines calling "...=function(solution);" in the code*/

//      typedef double (*FunctionCallback)(double sol[D]);

        /*benchmark functions */

//      double sphere(double sol[D]);
//      double Rosenbrock(double sol[D]);
//      double Griewank(double sol[D]);
//      double Rastrigin(double sol[D]);

        /*Write your own objective function name instead of sphere*/
//      FunctionCallback function = &sphere;

        /*Fitness function*/
        double CalculateFitness(double fun)
         {
                double result=0;
                if(fun>=0)
                {
                        result=1/(fun+1);
                }
                else
                {

                        result=1+Math.abs(fun);
                }
                return result;
         }

        /*The best food source is memorized*/
        void MemorizeBestSource()
        {
          int i,j;
```

```
            for(i=0;i<FoodNumber;i++)
            {
            if (f[i]<GlobalMin)
                        {
          GlobalMin=f[i];
          for(j=0;j<D;j++)
            GlobalParams[j]=Foods[i][j];
          }
            }
 }
```

/*Variables are initialized in the range [lb,ub]. If each parameter has different range, use arrays lb[j], ub[j] instead of lb and ub */

/* Counters of food sources are also initialized in this function*/

```
void init(int index)
{
  int j;
  for (j=0;j<D;j++)
                        {
       r = (   (double)Math.random()*32767 / ((double)32767+(double)(1)) );
       Foods[index][j]=r*(ub-lb)+lb;
                        solution[j]=Foods[index][j];
                        }
            f[index]=calculateFunction(solution);
            fitness[index]=CalculateFitness(f[index]);
            trial[index]=0;
}

/*All food sources are initialized */
void initial()
{
            int i;
            for(i=0;i<FoodNumber;i++)
            {
            init(i);
            }
            GlobalMin=f[0];
    for(i=0;i<D;i++)
    GlobalParams[i]=Foods[0][i];
}

void SendEmployedBees()
{
 int i,j;
 /*Employed Bee Phase*/
  for (i=0;i<FoodNumber;i++)
```

```
            {
            /*The parameter to be changed is determined randomly*/
            r = ((double) Math.random()*32767 / ((double)(32767)+(double)(1)) );
            param2change=(int)(r*D);

            /*A randomly chosen solution is used in producing a mutant solution of the
solution i*/
            r = (   (double)Math.random()*32767 / ((double)(32767)+(double)(1)) );
            neighbour=(int)(r*FoodNumber);

            /*Randomly selected solution must be different from the solution i*/
           // while(neighbour==i)
           // {
           // r = (   (double)Math.random()*32767 / ((double)(32767)+(double)(1)) );
           // neighbour=(int)(r*FoodNumber);
           // }
            for(j=0;j<D;j++)
            solution[j]=Foods[i][j];

            /*v_{ij}=x_{ij}+\phi_{ij}*(x_{kj}-x_{ij}) */
            r = (   (double)Math.random()*32767 / ((double)(32767)+(double)(1)) );
            solution[param2change]=Foods[i][param2change]+(Foods[i][param2change]-
Foods[neighbour][param2change])*(r-0.5)*2;

            /*if generated parameter value is out of boundaries, it is shifted onto the
boundaries*/
            if (solution[param2change]<lb)
              solution[param2change]=lb;
            if (solution[param2change]>ub)
              solution[param2change]=ub;
            ObjValSol=calculateFunction(solution);
            FitnessSol=CalculateFitness(ObjValSol);

            /*a greedy selection is applied between the current solution i and its mutant*/
            if (FitnessSol>fitness[i])
            {

            /*If the mutant solution is better than the current solution i, replace the solution
with the mutant and reset the trial counter of solution i*/
            trial[i]=0;
            for(j=0;j<D;j++)
            Foods[i][j]=solution[j];
            f[i]=ObjValSol;
            fitness[i]=FitnessSol;
            }
            else
            {   /*if the solution i can not be improved, increase its trial counter*/
               trial[i]=trial[i]+1;
```

```
            }

            }
            /*end of employed bee phase*/
        }

        /* A food source is chosen with the probability which is proportioal to its quality*/
        /*Different schemes can be used to calculate the probability values*/
        /*For example prob(i)=fitness(i)/sum(fitness)*/
        /*or in a way used in the metot below prob(i)=a*fitness(i)/max(fitness)+b*/
        /*probability values are calculated by using fitness values and normalized by dividing
maximum fitness value*/
        void CalculateProbabilities()
        {
            int i;
            double maxfit;
            maxfit=fitness[0];
          for (i=1;i<FoodNumber;i++)
                {
                  if (fitness[i]>maxfit)
                  maxfit=fitness[i];
                }

         for (i=0;i<FoodNumber;i++)
                {
                 prob[i]=(0.9*(fitness[i]/maxfit))+0.1;
                }
        }

        void SendOnlookerBees()
        {

          int i,j,t;
          i=0;
          t=0;
          /*onlooker Bee Phase*/
          while(t<FoodNumber)
                {

                r = (   (double)Math.random()*32767 / ((double)(32767)+(double)(1)) );
                if(r<prob[i]) /*choose a food source depending on its probability to be chosen*/
                {
                t++;

                /*The parameter to be changed is determined randomly*/
                r = ((double)Math.random()*32767 / ((double)(32767)+(double)(1)) );
                param2change=(int)(r*D);
```

```
            /*A randomly chosen solution is used in producing a mutant solution of the
solution i*/
            r = (   (double)Math.random()*32767 / ((double)(32767)+(double)(1)) );
            neighbour=(int)(r*FoodNumber);

            /*Randomly selected solution must be different from the solution i*/
            while(neighbour == i)
            {
               //System.out.println(Math.random()*32767+"  "+32767);
            r = (   (double)Math.random()*32767 / ((double)(32767)+(double)(1)) );
            neighbour=(int)(r*FoodNumber);
            }
            for(j=0;j<D;j++)
            solution[j]=Foods[i][j];

            /*v_{ij}=x_{ij}+\phi_{ij}*(x_{kj}-x_{ij}) */
            r = (   (double)Math.random()*32767 / ((double)(32767)+(double)(1)) );
            solution[param2change]=Foods[i][param2change]+(Foods[i][param2change]-
Foods[neighbour][param2change])*(r-0.5)*2;

            /*if generated parameter value is out of boundaries, it is shifted onto the
boundaries*/
            if (solution[param2change]<lb)
              solution[param2change]=lb;
            if (solution[param2change]>ub)
              solution[param2change]=ub;
            ObjValSol=calculateFunction(solution);
            FitnessSol=CalculateFitness(ObjValSol);

            /*a greedy selection is applied between the current solution i and its mutant*/
            if (FitnessSol>fitness[i])
            {
            /*If the mutant solution is better than the current solution i, replace the solution
with the mutant and reset the trial counter of solution i*/
            trial[i]=0;
            for(j=0;j<D;j++)
            Foods[i][j]=solution[j];
            f[i]=ObjValSol;
            fitness[i]=FitnessSol;
            }
            else
            {   /*if the solution i can not be improved, increase its trial counter*/
               trial[i]=trial[i]+1;
            }
            } /*if */
            i++;
            if (i==FoodNumber)
            i=0;
```

```
            }/*while*/

            /*end of onlooker bee phase     */
        }

        /*determine the food sources whose trial counter exceeds the "limit" value. In Basic
ABC, only one scout is allowed to occur in each cycle*/
        void SendScoutBees()
        {
        int maxtrialindex,i;
        maxtrialindex=0;
        for (i=1;i<FoodNumber;i++)
              {
              if (trial[i]>trial[maxtrialindex])
              maxtrialindex=i;
              }
        if(trial[maxtrialindex]>=limit)
        {
                init(maxtrialindex);
        }
        }


double calculateFunction(double sol[])
{
return Rastrigin (sol);
}
        double sphere(double sol[])
        {
        int j;
        double top=0;
        for(j=0;j<D;j++)
        {
        top=top+sol[j]*sol[j];
        }
        return top;
        }

        double Rosenbrock(double sol[])
        {
        int j;
        double top=0;
        for(j=0;j<D-1;j++)
        {
        top=top+100*Math.pow((sol[j+1]-
Math.pow((sol[j]),(double)2)),(double)2)+Math.pow((sol[j]-1),(double)2);
        }
        return top;
```

```
	}

	double Griewank(double sol[])
	{
		int j;
		double top1,top2,top;
		top=0;
		top1=0;
		top2=1;
		for(j=0;j<D;j++)
		{
			top1=top1+Math.pow((sol[j]),(double)2);

top2=top2*Math.cos((((sol[j])/Math.sqrt((double)(j+1)))*Math.PI)/180);

		}
		top=(1/(double)4000)*top1-top2+1;
		return top;
	}

	double Rastrigin(double sol[])
	{
		int j;
		double top=0;

		for(j=0;j<D;j++)
		{
			top=top+(Math.pow(sol[j],(double)2)-
10*Math.cos(2*Math.PI*sol[j])+10);
		}
		return top;
	}
}
```