
# PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 50/08

# Estudo de Métodos de Aprendizado Online em Sistemas Autônomos

**Sand Luz Corrêa**

**Renato Fontoura de Gusmão Cerqueira**

Departamento de Informática

**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO**

**RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900**

**RIO DE JANEIRO - BRASIL**

Monografias em Ciência da Computação, No. 50/08 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Dezembro, 2008

# Estudo de Métodos de Aprendizado Online em Sistemas Autônomos[1]

**Sand Luz Corrêa e Renato Fontoura de Gusmão Cerqueira**

{scorrea, rcerq}@inf.puc-rio.br

**Abstract.** Autonomic computing systems must manage themselves with little or no human intervention. Clearly, decision making is a critical issue in such system, which must learn how and when to invoke actions based on past experience. Learning methods such as clustering, bayesian network and support vector machines have achieved considerable success in several autonomic computing approaches. However most of these works rely on a batch setting where all of the training data is available in advance. There has been little use of online learning in real-time applications. In this work we survey some approaches designed to support online, incremental learning in autonomic system.

**Keywords:** Autonomic computing, decision making, online learning

**Resumo.** Sistemas autônomos devem se auto-gerenciar com mínima intervenção humana. Neste sentido, tomada de decisão é um aspecto essencial em tais sistemas, os quais devem aprender como e quando invocar ações baseando-se em experiências passadas. Métodos de aprendizado, tais como agrupamento (*clustering*), redes bayesianas e SVM (*support vector machines*), têm sido aplicados em várias abordagens para computação autônomas com considerável sucesso. Contudo, grande parte destes trabalhos se apóiam em cenários *offline*, nos quais os dados de treinamento estão disponíveis antecipadamente. Poucos trabalhos têm explorado aprendizado *online* em aplicações em tempo-real. Neste trabalho, revisamos algumas abordagens projetadas para apoiar o aprendizado *online* e incremental em sistemas autônomos.

**Palavras-chave:** Computação autônoma, tomada de decisão, aprendizado *online*

[1]Trabalho patrocinado pelo Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil. Processo CNPq número 140729/2006-2

**Responsável por publicações:**
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

# 1 Introdução

Sistemas dinamicamente adaptáveis têm atraído a atenção de várias áreas de pesquisas, especialmente computação ubíqua e computação autônoma. Avanços recentes em técnicas de programação e infra-estruturas de desenvolvimento de software permitem que sistemas adaptem seu comportamento corrente, de forma dinâmica, em virtude de necessidades oriundas de seu próprio uso ou condições ambientais [1]. Neste contexto, igualmente importante são as abordagens para tomada de decisão. Estas referem-se aos mecanismos que possibilitam a captura da importância relativa de diferentes ações em um ambiente dinamicamente mutável. Em geral, uma decisão é tomada considerando-se experiências passadas, sendo estas obtidas a partir da análise de um grande volume de dados históricos sobre o sistema em operação. A análise consiste em um processo de aprendizado *offline*, em que modelos do comportamento do sistema são derivados a partir de dados de treinamento disponíveis antecipadamente. De fato, grande parte dos trabalhos que envolvem soluções para computação autônoma ou sistemas auto-gerenciáveis utilizam largamente tal abordagem [2, 3, 4, 5, 6].

Entretanto, uma grande dificuldade de processos de aprendizado *offline* consiste na natureza não incremental do aprendizado. Em muitas aplicações, o monitoramento do ambiente e tomadas de decisão são processos constantes. Isto exige uma abordagem incremental que atualize continuamente o modelo corrente à medida que mais informações se tornem disponíveis. Contudo, grande parte das técnicas de aprendizado *offline* são projetadas para processamento em lote de um grande volume de dados, sendo o conjunto de treinamento lido e processado repetidamente em busca de uma solução ótima. Portanto, tais abordagens se tornam inviáveis para aplicações que necessitam de aprendizado incremental e dinâmico.

Neste trabalho, discutimos algumas abordagens propostas na literatura para resolver este problema. De maneira geral, classificamos tais abordagens em dois grupos: aprendizado incremental e algoritmos *online*. A primeira abordagem refere-se a um conjunto de soluções que adaptam técnicas de aprendizado *offline* para serem aplicadas incrementalmente. Cada passo ou incremento do processo de aprendizado faz uso de estruturas de dados eficientes em termos de consumo de recursos computacionais (especialmente memória). A segunda abordagem refere-se a uma área de pesquisa própria (*Online Algorithms*) e independente da área de aprendizado de máquina. Ambas, porém, compartilham o mesmo objetivo: uma tomada de decisão presente, baseada somente em experiências obtidas no passado.

Este trabalho está organizado da seguinte forma. A seção 2 introduz as principais características associadas ao aprendizado incremental e apresenta dois trabalhos que exploram tais características para tomada de decisão em sistemas autônomos. De maneira geral, estes trabalhos modificam algoritmos clássicos de aprendizado de máquina para adequá-los a processos de aprendizado *online*. O primeiro trabalho baseia-se em um algoritmo de agrupamento incremental e adaptativo. A segunda proposta usa redes bayesianas dinâmicas para introduzir a noção de tempo aos modelos de aprendizado. A seção 3 traz uma introdução à área de algoritmos *online* e algumas propostas que foram adaptadas para sua aplicação em problemas de aprendizado de máquina e tomada de decisão. Finalmente, a seção 4 apresenta a conclusão deste trabalho.

# 2 Aprendizado Incremental

À medida que estudos em aprendizado de máquina são aplicados na resolução de problemas do mundo real, algumas suposições simplistas, que guiaram a área durante anos, são abandonadas. Dentre tais suposições, destaca-se a hipótese (comumente assumida por técnicas de aprendizado de máquina) que considera que todas as observações ambientais usadas em um processo de aprendizado encontram-se disponíveis antecipadamente. O relaxamento de tal suposição cria o conceito de aprendizado incremental, no qual observações são assimiladas à medida que se tornam disponíveis.

Em modelos de aprendizado não incremental, todo o conhecimento sobre o ambiente é induzido de forma monolítica e em um único momento. Como conseqüência, tais modelos apresentam as seguintes características: (1) o conhecimento induzido não pode ser atualizado intermitentemente; (2) o algoritmo de aprendizado realiza buscas extensivas sobre os dados a fim de assegurar que a acurácia do preditor tende para uma solução ótima; (3) privilegia-se a acurácia da solução em detrimento à eficiência; (4) normalmente, o conhecimento adquirido não é organizado de forma a facilitar sua recuperação.

Ao contrário, modelos de aprendizado incremental assumem como hipótese fundamental que observações ambientais são obtidas ao longo do processo de aprendizado. Portanto, o ambiente pode mudar ao longo do processo, exigindo-se uma atualização contínua do conhecimento adquirido. Esta necessidade, por sua vez, exige do modelo de aprendizado um compromisso entre acurácia e eficiência. Tais características tornam modelos de aprendizado incremental abordagens promissoras para tomada de decisão em sistemas dinamicamente adaptáveis. Entretanto, poucos trabalhos exploram este tema. A seguir, descrevemos duas propostas que investigam o uso de aprendizado incremental em sistemas autônomos.

## 2.1 MESO

MESO (*Muti-Element Self-Organizing tree*) [7] é um sistema baseado em memória perceptual[2], projetado para apoiar processos de aprendizado incremental e tomada de decisão *online* em sistemas autônomos. Para atingir este objetivo, MESO implementa um algoritmo de agrupamento que envolve duas funções principais: treinamento e classificação. Durante treinamento, padrões são adicionados à memória perceptual, permitindo a construção de um modelo dos dados amostrados. Em geral, padrões compreendem observações relacionadas à qualidade de serviço ou contextos ambientais. Cada amostra do conjunto de treinamento consiste em um par $(x_i, y_i)$, onde $x_i$ é um vetor de padrões e $y_i$ corresponde a uma meta-informação, definida pelo usuário, associada ao padrão. Um exemplo de meta-informação consiste na ação a ser executada, quando o padrão associado é reconhecido.

Na memória perceptual, padrões são organizados em uma estrutura hierárquica que facilita sua recuperação. Na fase de classificação, a estrutura é consultada fornecendo-se o conceito a ser classificado. Este é comparado com os padrões armazenados e a meta-informação associada ao padrão de maior grau de similaridade em relação ao conceito fornecido é retornada.

[2] memoria perceptual refere-se a um tipo de memória de longo prazo, usada para armazenar padrões de estímulos externos

## Esferas Sensitivas

MESO permite a possibilidade de treinamento incremental em sistemas de monitoramento contínuo. Periodicamente, padrões capturados são adicionados ao modelo de aprendizado. Entretanto, para limitar o consumo de recursos computacionais, especialmente memória e processamento, padrões similares são agrupados em pequenos grupos denominados esferas sensitivas. Esferas sensitivas são organizadas em um estrutura hierárquica que agiliza os processos de treinamento e classificação, bem como provê uma significativa compressão de dados. No entanto, para manter a acurácia do classificador, esferas sensitivas crescem incrementalmente.

O algoritmo 1 ilustra o agrupamento usado em MESO. Para cada padrão lido, calcula-se a distância ($d$) entre o padrão e o centro da esfera mais próxima ($w_i$). Se esta distância for menor ou igual a $\delta$ (inicialmente nulo), o padrão é inserido na esfera, cujo centro é recalculado; caso contrário, uma nova esfera é criada contendo o padrão lido. O centro de uma esfera é calculado como a média de todos os padrões que a compõem. Entretanto, ao contrário de algoritmos de agrupamento tradicionais, em MESO, o valor de $\delta$ muda dinamicamente. Efetivamente, $\delta$ representa a sensibilidade do algoritmo em relação à distância entre os padrões.

Uma consideração importante é a relação entre o valor de $\delta$ e seu impacto no funcionamento do sistema. Se $\delta$ é muito pequeno, a taxa de compressão é baixa e muitas esferas serão criadas. Como conseqüência, o tempo para treinamento aumenta consideravelmente. Se $\delta$ é muito grande, poucas esferas são criadas, afetando a acurácia do classificador. Portanto, a função de crescimento para $\delta$ deve permitir um balanceamento entre criação de esferas e crescimento das mesmas. No algoritmo 1, tal função é representada por $grow_\delta$. O parâmetro $f$ é uma função de balanceamento fornecida pelo usuário.

**Algorithm 1** Algoritmo para criação de esferas sensitivas

initialize cluster center, $\delta = 0$, $f$
input pattern $x(t)$
find nearest center, $w_i$
**if** $d(x_i, w_i) \leq \delta$ **then**
  update cluster center
**else**
  create new center $w_j = x(t)$
  $grow_\delta = \frac{(d-\delta)\frac{\delta}{d}f}{1+ln(d-\delta+1)^2}$.
  $\delta = grow_\delta$
**end if**
next pattern

## Estrutura Hierárquica

Para tornar a busca por um padrão mais eficiente, muitos classificadores os organizam em uma estruturas hierárquicas. Em MESO, no entanto, esferas sensitivas, e não padrões individuais, são organizadas em uma estrutura de árvore. Como mostrado na figura 1, esta árvore é construída, inicialmente, atribuindo-se o conjunto de esferas sensitivas ao nó raiz. Um algoritmo recursivo divide cada nó da árvore em subconjuntos de esferas similares e

cada subconjunto se torna um nó filho. Estes também são divididos até que cada nó filho contenha apenas uma esfera. Subconjuntos de esferas similares são criados ao particionar um nó. Para cada nó filho é atribuída uma partição e uma esfera pivô. As demais esferas do nó pai são então distribuídas entre os nós filhos de acordo com a distância em relação às esferas pivô.

A árvore de esferas sensitivas produz um modelo hierárquico dos dados de treinamento. Quanto mais profundo o nível da árvore, maior o grau de similaridade entre as esferas numa mesma partição. Dessa forma, durante uma classificação, o conceito fornecido é comparado com o pivô, a partir da raiz até um nó folha, seguindo o caminho de maior similaridade.

É importante ressaltar que a árvore de MESO pode ser construída incrementalmente, à medida que novos padrões são capturados pelo modelo. Como a árvore manipula esferas (e não padrões individuais) que são criadas controladamente, é possível manter uma estrutura hierárquica de dados comprimidos de forma eficiente. O uso de MESO como ferramenta de apoio à tomada de decisão em softwares adaptáveis foi avaliado através de um estudo de caso envolvendo uma aplicação de *stream* de áudio. Esta deveria adaptar-se diante de diferentes condições de rede, tais como perda de pacote, atraso, disponibilidade de banda. Particularmente, MESO foi utilizado como o módulo de tomada de decisão da aplicação, decidindo como e quando adaptações aconteciam.

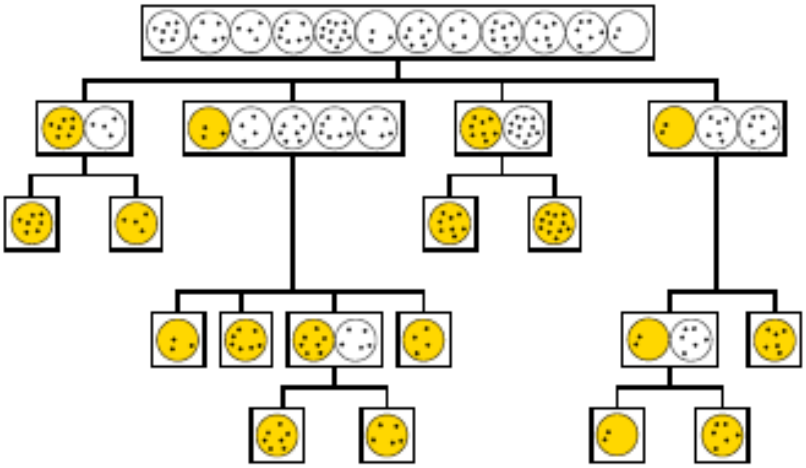

Figura 1: Organização de esferas sensitivas em MESO

## 2.2 Brodie et al.

Em Brodie et al. [8] é proposta uma técnica de mensuração adaptativa, denominadas *probes* ativos, cujo objetivo é prover um mecanismo eficiente de diagnóstico de falhas e localização de problemas em aplicações distribuídas. Para alcançar este objetivo, tal mecanismos se baseia em um processo de aprendizado e inferência *online* que continuamente selecionam um conjunto de *probes* mais representativos para um diagnóstico em um dado momento. Um *probe* consiste em uma sonda ou teste cujo resultado depende de alguns componentes do sistema. Exemplos de *probes* incluem: transações enviadas para um servidor, comandos enviados pela rede, mensagens de correio eletrônico, requisições de serviço,

consulta a um banco de dados, etc.

A seleção ativa de *probes* informativos provê um mecanismo de instrumentação leve e eficiente, já que apenas um conjunto pequeno de sondas são enviadas e processadas. A seleção é implementada por um módulo de tomada de decisão que utiliza uma abordagem de aprendizado incremental e contínuo, onde o diagnóstico é construído e atualizado à medida que novas observações se tornam disponíveis. A seguir, descrevemos como o módulo de tomada de decisão é implementado.

## Seleção Ativa de *Probes*

A figura 2 ilustra a abordagem proposta em Brodie et al. *Probes* são enviados para o sistema monitorado e os resultados são coletados para análise. Se a inferência sobre os dados coletados indicar algum problema, *probes* mais informativos são selecionados e enviados, com base na análise dos resultados dos *probes* anteriores. Este processo é repetido até que o problema seja completamente diagnosticado.

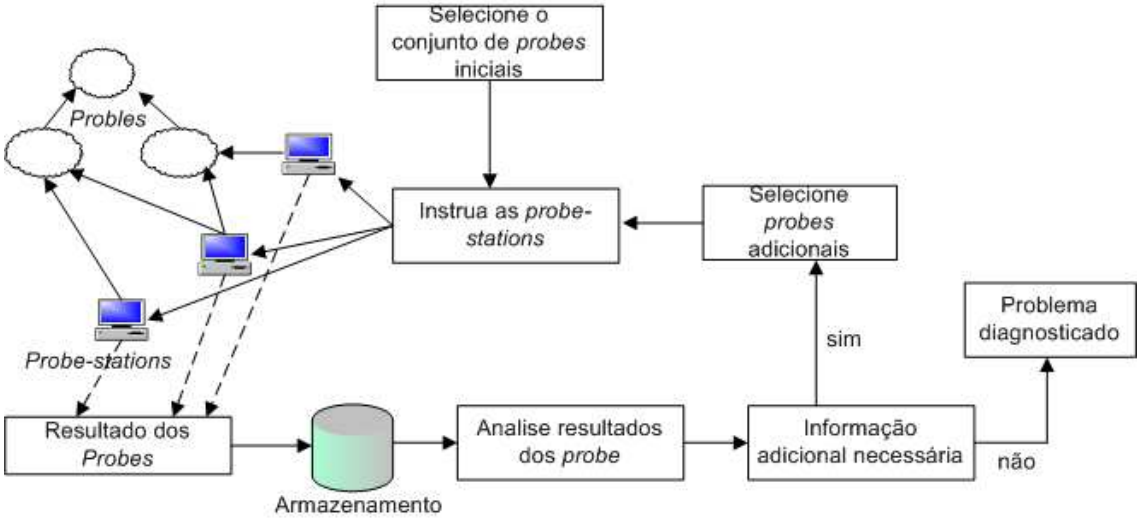


Figura 2: Diagnóstico adaptativo através de *probes* ativos

O algoritmo 2 implementa a seleção ativa de *probes*. Este algoritmo toma como entrada um conjunto de probes $T$ e uma distribuição de probabilidade à priori $P(X)$ sobre os estados do sistema. O algoritmo mantém um conjunto de crença (*belief*) sobre o estado do sistema, $Belief(X) = P(X|T_a)$, onde $T_a$ é o conjunto corrente de *probes* e seus resultados. O algoritmo calcula o próximo *probe* $Y^*$ que maximiza o ganho de informação sobre o estado $X$. Entretanto, diferentemente das abordagens passivas, o algoritmo conhece o resultado dos probes anteriores. Dessa forma, na iteração $k$, o cálculo é condicionado ao conjunto $T_a = \{Y_1^* = y_1^*, ..., Y_{k-1}^* = y_{k-1}^*\}$, enquanto que em abordagens passivas, o cálculo é condicionado a todos os resultados possíveis para o conjunto de *probes* selecionados anteriormente ($T_a = \{Y_1^*, ..., Y_{k-1}^*\}$). Portanto, a seleção ativa de *probes* encontra o $Y$ que maximiza o ganho de informação sobre $X$, dado que $T_a$ ocorreu, ou seja, $I(X; Y|T_a)$ (passo 1 do algoritmo 2).

Após encontrar $Y^*$, o algoritmo espera pelo seu resultado (passo 2). Ele então atualiza o conjunto de *probes* executados (passo 3) e a crença corrente (passo 4). Os passos 1 a

4 são repetidos até que não se possa obter mais informações sobre o estado do sistema e, portanto, melhorar o diagnóstico corrente.

Para o passo 4, atualização de crença, os autores usam um processo de inferência probabilística baseada em redes bayesianas dinâmicas. Redes bayesianas dinâmicas (DBN) estendem redes bayesianas tradicionais (BN), introduzindo-se no modelo a noção de fatias de tempo e especificando-se probabilidades de transição entre fatias consecutivas, ou seja, $P(X^t|X^{t-1})$, onde $X^t = \{x_1^t, ..., x_n^t\}$. Como conseqüência, DBNs podem ser representadas como duas BN interconectadas: a BN que representa a dependência dos estados do sistema no tempo $t$ ($BN_t$, intra-dependência) e a BN que representa a dependência entre $BN_t$ e $BN_{t-1}$ (inter-dependência). A inter-dependência foi modelada assumindo-se que o nó $X_i$ em $NB_t$ é unicamente dependente do nó $X_i$ em $BN_{t-1}$.

Em [8], um pequeno cenário envolvendo diagnóstico de falhas em transações Web foi desenvolvido como prova de conceito para os algoritmos propostos.

**Algorithm 2** Algoritmo para seleção ativa de probes

input: a set of available probes $T$ and a prior distribuition $P(X)$
output: a set $T_a$ of probes and their outcomes and $Belief(X)$
initialize:$Belief(X) = P(X)$, $T_a = \emptyset$
**repeat**
1. $Y^* = argmax_{Y \in T \setminus T_a} I(X; Y|T_a)$
2. execute $Y^*$; it returns $Y^* = y^*$ (0 or 1)
3. update $T_a = T_a \bigcup \{Y^* = y^*\}$
4. update $Belief(X) = P(X|T_a)$

**until** $\neg$ ($\exists Y \in T$ such that $I(X; Y|T_a) > 0$)
return $T_a$, $Belief(X)$

# 3 Algoritmos *Online*

Como mencionado anteriormente, Algoritmos *Online* constitui-se em uma área própria e independente da área de Aprendizado de Máquina. Entretanto, o interesse em comum por problemas de tomada de decisão envolvendo incerteza, motivou o levantamento de uma coleção de problemas em Aprendizado de Máquina que podem ser adequadamente tratados por um *framework* de algoritmos *online* [9]. Dentre tais problemas, destacam-se os que envolvem aprendizado a partir de exemplos.

Em Algoritmos *Online*, o aprendizado a partir de exemplos assume a forma de uma seqüencia consecutiva de rodadas. Em cada rodada, uma pergunta é apresentada ao módulo de aprendizado, o qual deve fornecer uma resposta. Para responder à pergunta, o módulo de aprendizado usa um mecanismo de previsão, denominado hipótese, que funciona como um mapeamento entre conjunto de perguntas e o conjunto de respostas admissíveis. Após prever uma resposta, a resposta correta é apresentada ao módulo de aprendizado. A qualidade da resposta do módulo de aprendizado é aferida por uma função de perda que mede a discrepância entre a resposta dada e a correta. O objetivo final do módulo de aprendizado é minimizar a perda acumulada ao longo de sua execução. Para atingir tal objetivo, o módulo de aprendizado deve atualizar sua hipótese após cada rodada. Dessa forma, cada rodada pode ser dividida em três etapas: primeiramente o algoritmo recebe

uma instância (pergunta); em seguida, o algoritmo prevê um rótulo para a instância (resposta), baseando-se em uma hipótese; finalmente, o rótulo (resposta correta) é apresentado ao algoritmo que, então, atualiza sua hipótese [10].

Formalmente, um algoritmo *online* produz uma seqüencia de hipótese $f = (f_1, ..., f_m)$. $f_1$ é uma hipótese inicial arbitrária e $f_i$, para $i > 1$, é a hipótese escolhida após a análise dos $(i-1)$ exemplos anteriores. $l(f_t(x_t), y_t)$ é a função de perda que descreve o erro do algoritmo ao prever $y_t$, baseando-se em $x_t$ e os exemplos anteriores $(x_1, y_1), ..., (x_{t-1}, y_{t-1})$.

Em geral, algoritmos *online* não assumem qualquer suposição estatística sobre a seqüencia de dados, a qual pode ser determinística, estocástica ou mesmo arbitrariamente adaptativa. A maior dificuldade destas abordagens, no entanto, consiste na realimentação contínua de rótulos (ou apresentação da resposta correta), já que, em muitos problemas, não é possível fornecer tal realimentação de forma precisa e em um futuro próximo. A seguir, discutimos alguns trabalhos que utilizam um *framework* de algoritmos *online* para tomada de decisão em sistemas autônomos.

### 3.1 Sehia

Em [11] é apresentado um *framework* baseado em algoritmos *online* para a construção de sistemas autônomos. O *framework* é especialmente projetado para aprendizado a partir de falhas determinísticas e auto-recuperação. A figura 3 apresenta uma visão geral do *framework* proposto.

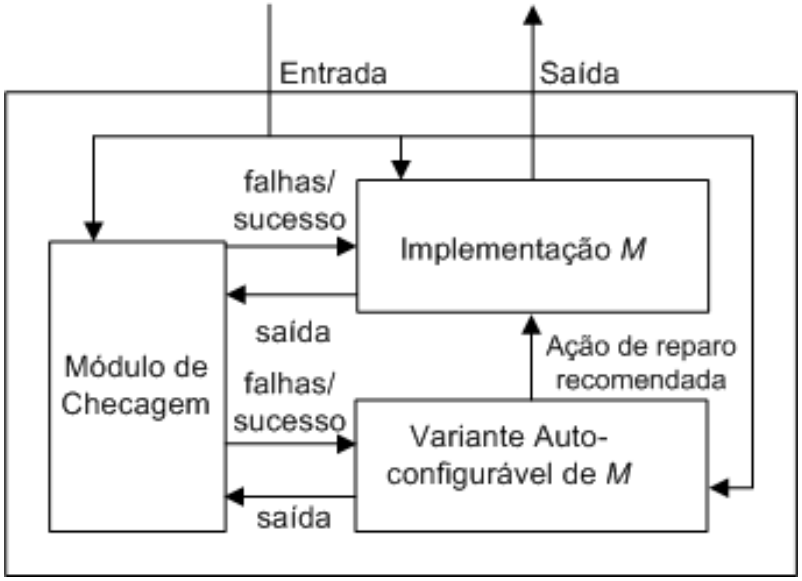


Figura 3: Componentes de um sistema capaz de auto-reparo em Sehia

O módulo de implementação $M$ consiste na versão original do sistema, a qual será implantada e testada. Este módulo recebe dados de entrada diretamente do ambiente em que está inserido e gera saídas. Entretanto, para aplicação do *framework*, assumem-se que o comportamento de $M$ possa ser dividido em rodadas, cada uma das quais começando em um estado válido. Um estado inicial válido, por sua vez, é definido por uma invariante $I_{start}$. Formalmente, um comportamento correto de $M$ é representado por uma seqüencia

de estados e ações, como mostrado abaixo.

$$s_0 a_1 s_1 a_2 ... a_{i_1} s_{i_1} a_{i_1+1} s_{i_1+1} ... a_{i_2} s_{i_2} a_{i_2+1} s_{i_2+1}$$

onde:

- $s_0 \in I$, $I$ é o conjunto de estados iniciais do sistema
- $s_i = \delta(s_{i-1}, a_i)$, $\delta$ é a função de transição que descreve o próximo estado do sistema que resulta da execução da ação $a_i$ no estado corrente $s_{i-1}$
- $\forall_j \in 1, 2, 3, ..., s_{i_j} \in I_{start}$

Uma rodada $t$ é definida como uma seqüencia finita de estados começando em $s_{i_t}$ e terminando em $s_{i_{t+1}}$. Dessa forma, exige-se que o sistema retorne para um estado válido em cada rodada, evitando-se assim a propagação de erros para rodadas posteriores, o que poderia tornar o sistema irrecuperável.

O módulo de checagem verifica se a saída gerada por $M$ é correta. Tal módulo pode ser implementado como uma versão mais simples e formalmente verificada de $M$ ou como uma coleção de monitores que checam continuamente se as invariantes e asserções estão sendo violadas. O resultado da verificação (falha ou sucesso) é retornado para $M$, de forma que o mesmo possa tomar ações corretivas, quando necessário.

O último componente do *framework* consiste em uma variante de $M$ instrumentada de forma a modificar a função de transição de estados $\delta$ de acordo com o modelo de falhas. Em cada rodada, a variante, denotada por $\bar{M}$, executa paralelamente a $M$, processando as mesmas entradas. Entretanto, suas saídas são direcionadas apenas para o módulo de checagem, o qual as verificam. Baseado na resposta do módulo de checagem, $\bar{M}$ decide como e quando modificar a função de transição para a próxima rodada. O algoritmo que $\bar{M}$ usa para tomar esta decisão é denominado estratégia de recuperação.

A estratégia de recuperação deve escolher uma ação de reparo a partir de um espaço possível de ações, definido pelo modelo de falhas. Para explorar as conseqüências de um reparo particular, $\bar{M}$ se auto-configura a cada rodada, mesmo na ausência de falhas. Posteriormente, se $M$ falha, ele se recupera carregando uma ação de reparo recomendada por $\bar{M}$. Dessa forma, $M$ pode se auto-ajustar baseando-se na experiência obtida por $\bar{M}$ em várias rodadas de execução.

## O Problema do Bandido Multi-armado

No *framework* proposto, o problema a ser resolvido consiste na escolha de uma ação de reparo, baseando-se na história de desempenho de ações candidatas no passado. A escolha da ação de reparo a ser executada é implementada por um algoritmo de aprendizado *online*, sendo a escolha tratada como um processo de aprendizado por erros. Cada rodada corresponde a uma tentativa na qual o aprendizado prossegue. A solução utilizada baseia-se em um algoritmo *online* clássico denominado o problema do bandido multi-armado (*the multi-armed bandit problem*). Neste algoritmo, um jogador deve escolher, em cada rodada, uma máquina dentre $m$ candidatas. No final de cada rodada, o jogador recebe uma recompensa de acordo com a escolha feita. Recompensas podem ser não positivas e não se assumem suposições estatísticas sobre elas.

Adaptando-se o algoritmo original para resolver o problema do *framework* proposto, o jogador corresponde ao módulo $M$ e as máquinas correspondem às ações de reparo. Se a ação escolhida evita a presença de erros durante a rodada, o módulo recebe uma recompensa de 1; caso contrário 0. Este problema ilustra o compromisso entre exploração de alternativas e aproveitamento das melhores escolhas. Em geral, exploração é desejável, porém, em demasia, pode levar a um baixo aproveitamento das melhores alternativas. O *framework* proposto trata tal dilema, paralelizando-se operações de exploração, atribuídas a $\bar{M}$, e de aproveitamento, realizada por $M$.

A estratégia de seleção de ações de reparo utilizada pelo *framework* é ilustrada no algoritmo 3. A idéia básica do algoritmo consiste em usar pesos para monitorar o desempenho de $m$ ações de reparo em uma seqüencia de rodadas. Um peso alto para uma ação indica uma história de bom desempenho. No início de cada rodada $t$, o algoritmo sorteia uma ação $i_t$ de acordo com uma distribuição de probabilidade $p_1(t), p_2(t), ..., p_m(t)$ sobre as ações. Esta distribuição é uma mistura de normal e uma distribuição que atribui a cada ação uma probabilidade associada ao seu peso. A distribuição baseada em peso facilita o aproveitamento das melhores alternativas, enquanto que a distribuição uniforme assegura a exploração do espaço de ações. Se a ação escolhida $i_t$, provoca uma falha no sistema, gera-se um custo de 1 unidade para a ação $i_t$ e seu peso é decrementado por um fator exponencial.

**Algorithm 3** Algoritmo para seleção de ações de reparo

parameter: $\gamma \in (0, 1)$
$t = 1$
**loop**
  **for** $i = 1$ to $m$ **do**
    $p_i(t) = (1 - \gamma)\frac{w_i(t)}{\sum_{j=1}^{m} w_j(t)} + \frac{\gamma}{m}$
  **end for**
  Draw action $i_t$ from the set {1,2,...,m} randomly according to the probabilities $p_1(t), p_2(t), ..., p_m(t)$
  **if** $i_t$ failed **then**
    $C_{i_t} = 1$
  **else**
    $C_{i_t} = 0$
  **end if**
  **for** $j = 1$ to $m$ **do**
    **if** $j = i_t$ **then**
      $\hat{C}_j(t) = C_j(t)/p_j(t)$
    **else**
      $\hat{C}_j(t) = 0$
    **end if**
    $w_j(t + 1) = w_j(t).exp(-\frac{\gamma}{m}\hat{C}_j(t))$
  **end for**
  $t = t + 1$
**end loop**

O *framework* proposto foi validado através de um estudo de caso envolvendo monito-

ramento de dados trafegados numa rede. O monitor é responsável por detectar tráfego malicioso e eliminar os pacotes afetados. Duas implementações do monitor foram fornecidas, sendo a mais otimizada correspondendo ao módulo $M$ e a versão mais simples ao módulo de checagem. Também foi fornecido uma versão adaptativa de $M$, correspondendo ao módulo de reparo $\bar{M}$.

### 3.2 Aprendizado *Online* Usando Kernels

Métodos de aprendizado baseados em *kernel* [12] consistem em uma classe de algoritmos para reconhecimento de padrões, cujos representantes mais comuns são as Máquinas de Vetores Suporte (SVM). Nestes algoritmos, dados de entrada são mapeados implicitamente para um novo espaço de características, onde uma hipótese mais apropriada pode ser encontrada. O mapeamento para o novo espaço é feito através de funções *kernel*, usando-se um produto escalar. Métodos baseados em *kernel* (KM) consistem em uma classe de algoritmos para reconhecimento de padrões, cujos representantes mais comuns são as máquinas de vetores-suporte (SVM). Tais algoritmos tratam o problema de reconhecimento de padrões, mapeando-se os dados de entrada ou conjunto de treinamento para um novo espaço de maior dimensão, denominado espaço de características (*feature space*). Neste espaço, vários métodos podem ser usados para descobrir correlações entre os dados. Funções *kernel* são usadas para permitir a manipulação do espaço de características, sem, no entanto, computar explicitamente as coordenadas dos dados no espaço. Ao contrário, dados são manipulados computando-se o produto escalar entre as imagens de todos os pares de dados no espaço de características. Esta operação exige um esforço computacional menor que o cálculo explícito de coordenadas.

Métodos baseados em *kernel* têm sido utilizados com sucesso em muitos cenários de aprendizado *offline*. Entretanto, poucos trabalhos estendem estes métodos para cenários de aprendizado *online*. Dentre tais trabalhos, destacamos [].

Em especial, SVM têm sido utilizados com sucesso em muitos cenários de aprendizado *offline*, apresentando bons resultados de desempenho em tarefas de classificação e regressão. O bom resultado motivou o uso de tais métodos também em cenários de aprendizado *online*, como demonstrado a seguir.

SVM é um classificador binário cuja idéia básica é construir uma função *kernel* não linear para mapear dados do espaço de entrada para um espaço de dimensão maior. Neste espaço, generaliza-se um hiperplano que provê a separação ótima entre as duas classes analisadas. Formalmente, dado um conjunto de treinamento $T = \{(x_i, y_i) : x_i \in R^D, y_i \in \{-1, 1\}, i = 1, ..., m\}$, onde $x_i$ é um vetor $D$-dimensional de reais e $y_i$ indica a classe a que $x_i$ pertence, um classificador SVM é representado pela função:

$$F(x) = \sum_i a_i y_i x.x_i + b \tag{1}$$

Tipicamente, os coeficientes $a_i$ da equação 1 possuem valores não nulos apenas para um pequeno subconjunto do conjunto de treinamento. Tal subconjunto é conhecido como conjunto suporte e seus elementos são denominados vetores suporte.

Em Wang et al. [13], um algoritmo SVM é adaptado para aprendizado *online*, visando a construção de filtros de *spam*. A predição do algoritmo em cada rodada é determinada pelo classificador SVM. Entretanto, para tornar o algoritmo mais eficiente e, portanto, mais adequado para uso *online*, a função de decisão foi restrita ao conjunto suporte, sendo o

restante do conjunto de treinamento desprezado. Como apenas os vetores suporte possuem coeficientes não nulos, os demais dados do conjunto de treinamento podem ser desprezados sem grande perda de precisão na classificação. O algoritmo 4 ilustra a construção da função de decisão para um filtro de *spam*.

**Algorithm 4** SVM restrito ao conjunto suporte

```
initialize: Seed SVM classifier with a few examples of each class (spam and ham)
train an initial SVM filters
for i = 1 to m do
  classify x_i
  query the true label of x_i
  if filter's prediction is wrong then
    retrain SVM filters based on SV_i + x_i
  else
    discard x_i
  end if
end for
```

Entretanto, um grande problema da adaptação proposta por Wang et al é o fato de que, para um grande conjunto de dados, algoritmos SVMs produzem um grande número de vetores suporte. Como conseqüência, ambos os requisitos de tempo e espaço do classificador, os quais aumentam linearmente com o número de vetores produzidos, são fortemente comprometidos. Para aplicações *online*, nas quais os dados e, conseqüentemente, o conjunto de treinamento, aumentam continuamente, o número de vetores produzidos também aumentará proporcionalmente.

Para resolver este problema, alguns trabalhos propõem limitar o número de vetores suporte criados e mantidos pelo classificador, já que, intuitivamente, espera-se que o conjunto suporte se torne saturado em algum momento. Uma vez que um número suficiente de vetores suporte foram produzidos, é possível determinar unicamente um hiperplano separador e, portanto, futuras adições ao conjunto suporte se tornam redundantes.

Ao limitar o tamanho do conjunto suporte de um classificador SVM, uma política de substituição de vetores suporte deve ser criada para tratar erros de predição, quando o conjunto já atingiu seu limite. Kivinen et al. [14] propõe o descarte dos vetores que estão a mais tempo na memória. Enquanto esta solução pode ser interessantes para entradas que seguem uma distribuição variante no tempo, ela pode ser arriscada para as demais situações. Uma solução baseada no ganho de informação é proposta em Agarwal et al. [15]. Nesta proposta, o ganho de informação é medido pela distância entre os vetores suporte e é denominada *Span*.

Suponha um conjunto de vetores suporte (SV) de $j$ elementos $\{(x_1, y_1), ..., (x_j, y_j)\}$ associado ao conjunto de treinamento $T$. Seja $a^0 = \{a_1^0, ..., a_j^0\}$ o conjunto de coeficientes de SV. Para qualquer SV $x_p$, define-se *Span* de $x_p$, $S_p$, como :

$$S_p^2 = min_{x \in \wedge_p} \| \ x - x_p \ \|^2 \tag{2}$$

$\wedge$ representa uma combinação linear dos outros SVs. Define-se *S-span* como o maior $S_p$ sobre todos os SVs, ou seja, $S = max_{p \in N_j} S_p$. A noção de *Span*, é então usada para determinar o vetor suporte a ser substituído, como ilustrado no algoritmo 5.

**Algorithm 5** SVM com número de vetores suporte limitado

Let the current set of SVs in the memory be $X_{old} = \{(x_1, y_1), ..., (x_n, y_n)$. Assume the memory is full; otherwise, one can keep adding the incoming SVs until it is full. Let $(x, y)$ be an incoming data point.
Compute $F(x) = \sum_{i=1}^{n} a_i y_i x.x_i + b$
**if** $F(x) = y$ **then**
  discard $(x, y)$
**else**
  Compute $S_{old}$, the *S-span* of $X_{old}$
  **for** $i = 1$ to $n$ **do**
    Compute $X^i_{new} = \{(x_1, y_1), ..., (x_{i-1}, y_{i-1}), (x_{i+1}, y_{i+1}), ..., (x_n, y_n), (x, y)\}$
    Compute $S_i$, the *Span* of $X^i_{new}$
  **end for**
  Find the least *S-span* among $S_i \ \forall \ i = 1, ..., n$. Let this be $S_p$; the corresponding support vector removed is $x_p$.
  **if** $S_p < S_{old}$ **then**
    remove $x_p$ from the set
  **else**
    discard $(x, y)$
  **end if**
**end if**

O algoritmo apresentado em Agarwal et al. foi testado em dois experimentos: um conjunto de dados apresentando distribuição variante no tempo e um conjunto apresentando distribuição estática. Em ambos os casos, o algoritmo conseguiu uma redução máxima de aproximadamente 70% do número de vetores suportes gerados, sem afetar, de forma significativa, a precisão do classificador.

# 4 Conclusão

A realização completa da computação autônoma apoia-se na noção de softwares capazes de adaptar dinamicamente seu comportamento em virtude de novas condições de uso ou características do ambiente. Tais adaptações exigem tomadas de decisão em tempo real a fim de prevenir-se de prejuízos ou perda de serviço. Neste contexto, métodos de aprendizado *online*, incremental e dinâmico tornam-se requisitos essenciais. Neste trabalho, apresentamos algumas propostas com este objetivo. De maneira geral, os trabalhos apresentados envolvem dois grupos de soluções: aprendizado incremental e algoritmos *online*.

O primeiro grupo de soluções apresenta como vantagem o fato de se baseiarem em métodos tradicionais de aprendizado *offline* e, portanto, um conjunto de algoritmos bem conhecidos para o problema de aprendizado em máquina. Porém, tais soluções, mesmo adaptadas para cenários *online*, apresentam restrições, já que não foram originalmente projetadas para este propósito. Soluções baseadas em algoritmos *online*, por sua vez, são inerentemente incrementais e dinâmicas, porém sofrem com a restrições de aplicabilidade, já que, em muitos problemas, não é possível fornecer realimentação de conhecimento de forma precisa e em um futuro próximo. A despeito de tais dificuldades, os trabalhos

aqui apresentados representam os primeiros passos em direção a sistemas autônomos mais maduros e melhor preparados para a solução de problemas do mundo real.

## Referências


[1] MCKINLEY, P. K.; SADJADI, S. M.; KASTEN, E. P. ; CHENG, B. H.. **Composing Adaptive Software**. Computer, 37(7):56–64, 2004.

[2] BARHAM, P.; DONNELLY, A.; ISAACS, R. ; MORTIER, R.. **Using Magpie for Request Extraction and Workload Modelling**. In: OSDI'04: PROCEEDINGS OF THE 6TH CONFERENCE ON SYMPOSIUM ON OPEARTING SYSTEMS DESIGN & IMPLEMENTATION, p. 18–18, Berkeley, CA, USA, 2004. USENIX Association.

[3] COHEN, I.; GOLDSZMIDT, M.; KELLY, T.; SYMONS, J. ; CHASE, J. S.. **Correlating Instrumentation Data to System States: a Building Block for Automated Diagnosis and Control**. In: OSDI'04: PROCEEDINGS OF THE 6TH CONFERENCE ON SYMPOSIUM ON OPEARTING SYSTEMS DESIGN & IMPLEMENTATION, p. 231–244, Berkeley, CA, USA, 2004. USENIX Association.

[4] CHEN, M.; KICIMAN, E.; BREWER, E. ; FOX, A.. **Pinpoint: Problem Determination in Large Dynamic Internet Services**. In: PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, p. 595–604, Washington, DC, USA, 2002. IEEE Computer Society.

[5] ZHANG, Q.; CHERKASOVA, L.; MATHEWES, G.; GREENE, W. ; SMIRNI, E.. **R-Capriccio: A Capacity Planning and Anomaly Detection Tool for Enterprise Services with Live Workloads**. In: PROCEEDINGS OF THE 8TH INTERNATIONAL MIDDLEWARE CONFERENCE, p. 244–265. Springer Berlin / Heidelberg, 2007.

[6] KUMAR, V.; COOPER, B.; EISENHAUER, G. ; SCHWAN, K.. **iManage: Policy-Driven Self-Management for Enterprise-Scale System**. In: PROCEEDINGS OF THE 8TH INTERNATIONAL MIDDLEWARE CONFERENCE, p. 287–307. Springer Berlin / Heidelberg, 2007.

[7] KASTEN, E. P.; MCKINLEY, P. K.. **MESO: Supporting Online Decision Making in Autonomic Computing Systems**. IEEE Trans. on Knowl. and Data Eng., 19(4):485–499, 2007.

[8] RISH, I.; BRODIE, M.; SHENG, M.; ODINTSOVA, N.; BEYGELZIMER, A.; GRABARNIK, G. ; HERNANDEZ, K.. **Adaptive Diagnosis in Distributed Systems**. IEEE Transactions on Neural Networks, 16(5):1088–1109, Sept. 2005.

[9] BLUM, A.. **On-line Algorithms in Machine Learning**. In: IN PROCEEDINGS OF THE WORKSHOP ON ON-LINE ALGORITHMS, DAGSTUHL, p. 306–325. Springer, 1996.

[10] SHALEV-SHWARTZ, S.. **Online Learning: Theory, Algorithms, and Applications**. PhD thesis, Hebrew University, 2007.

[11] SESHIA, S. A.. **Autonomic Reactive Systems via Online Learning**. In: ICAC '07: PROCEEDINGS OF THE FOURTH INTERNATIONAL CONFERENCE ON AUTONOMIC COMPUTING, p. 30, Washington, DC, USA, 2007. IEEE Computer Society.

[12] SCHOLKÖPF, B.; SMOLA, A. J.. **Learning with Kernels**. The MIT Press, Cambridge, MA, USA, 2002.

[13] WANG, Q.; GUAN, Y. ; WANG, X.. **SVM-Based Spam Filter with Active and Online Learning**. In: PROCEEDINGS OF THE FIFTEENTH TEXT RETRIEVAL CONFERENCE (TREC 2006), 2006.

[14] KIVINEN, J.; SMOLA, A. J. ; WILLIAMSON, R. C.. **Online Learning with Kernels**. IEEE Transactions on Signal Processing, 52(8):2165–2176, Aug. 2004.

[15] AGARWAL, S.; SARADHI, V. V. ; KARNICK, H.. **Kernel-based online machine learning and support vector reduction**. Neurocomput., 71(7-9):1230–1237, 2008.