SNIFFER: Um Ambiente Computacional para Gerenciamento de Experimentos de Aprendizado de Máquina Supervisionado

Gustavo E.A.P.A. Batista[1] **, Maria Carolina Monard**[1]

[1]Universidade de São Paulo – USP
Instituto de Ciências Matemáticas e de Computação – ICMC
Caixa Postal 668, 13560-970
São Carlos - SP, Brasil

{gbatista, mcmonard}@icmc.usp.br

Abstract. *Machine Learning – ML – is an inherently empirical research area since it is not possible to determine mathematically whether a ML algorithm will perform better than others. In other words, experimental studies should be carried out in order to understand and explain the differences between learning algorithms. However, experimental evaluation envolves executing a variety of experiments several times, which can be tedious and error-prone if carried out manually. This work presents a computational environment called SNIFFER, which provides support for measuring and comparing ML algorithms for classification tasks. SNIFFER lets the user define his/her own experiments through the organization of the input information in a special tree structure. We consider the setting of experiments through this sort of structure a novel idea.*

Resumo. *O desempenho de algoritmos de Aprendizado de Máquina – AM – necessita ser avaliado experimentalmente, pois não é possível afirmar, somente considerando aspectos da teoria, se um algoritmo terá sempre um desempenho melhor que outro. Assim, análises experimentais são fundamentais em AM. Entretanto, devido à necessidade de repetir a execução dos experimentos, essas análises são muito trabalhosas se forem realizadas manualmente. Neste trabalho apresentamos um ambiente computacional, denominado SNIFFER, para gerenciar experimentos que têm como objetivo medir o desempenho de algoritmos de AM supervisionado. Uma característica do ambiente SNIFFER que consideramos inovadora é que esse ambiente permite ao usuário definir, utilizando uma estrutura de árvore, diversos experimentos com vários algoritmos de aprendizado e conjuntos de dados, os quais são realizados em uma única execução do ambiente.*

1. Introdução

Análises experimentais são fundamentais na área de Aprendizado de Máquina – AM – uma vez que normalmente os métodos utilizados não permitem realizar para um tratamento formal completo. Em outras palavras, para um dado problema não existem instrumentos formais para decidir qual algoritmo de aprendizado é ótimo [Kibler and Langley, 1988]. Mesmo existindo uma vasta literatura em Estatística, Teoria de Aprendizado Computacional e outras áreas afins, a última palavra na decisão de qual método é o melhor para um determinado problema é sempre dada por uma avaliação experimental. Entretanto, mesmo sendo essenciais para o desenvolvimento da área de AM, as avaliações experimentais são normalmente muito trabalhosas e podem se tornar propensas à introdução de erros. Isso ocorre por diversos motivos, entre eles o uso de sistemas de aprendizado de domínio público que utilizam diferentes sintaxes para arquivos de entrada e saída; o uso

de métodos de *reamostragem* que requerem a repetição dos experimentos, a necessidade de ajustar os parâmetros dos métodos em análise ou dos sistemas de aprendizado a serem utilizados, o uso de diversos sistemas de aprendizado e de diversos conjuntos de dados nos experimentos, entre outros. Todas essas variáveis fazem com que um experimento tenha que ser executado um grande número de vezes, gerando uma grande quantidade de resultados que precisam ser analisados e publicados. Manipular esses resultados pode introduzir erros, comprometendo a qualidade da pesquisa.

Neste trabalho é proposto e descrito um ambiente computacional para gerenciar avaliações e comparações experimentais em AM supervisionado, denominado ambiente SNIFFER. O ambiente SNIFFER visa dar suporte ao usuário em todas as tarefas de um experimento de AM que ocorrem a partir do momento que o usuário possui os dados no formato atributo-valor até a análise e comparação de resultados. Grande parte dessas tarefas são realizadas automaticamente, o que reduz o risco de introdução de erros e torna o processo mais ágil. Uma característica do ambiente SNIFFER que consideramos inovadora é que esse ambiente permite ao usuário definir, utilizando uma estrutura de árvore, diversos experimentos com vários algoritmos de aprendizado e conjuntos de dados, os quais são realizados em uma única execução do ambiente.

Este trabalho está organizado da seguinte forma: na Seção 2 é fornecida uma breve introdução sobre AM supervisionado; na Seção 3 são apresentados os principais objetivos do ambiente SNIFFER; na Seção 4 são discutidas as semelhanças e diferenças entre o sistema SNIFFER e alguns trabalhos relacionados; na Seção 5 é descrito o funcionamento do ambiente; na Seção 6 é fornecida uma visão geral da arquitetura e na Seção 7 do projeto do ambiente SNIFFER; por fim, na Seção 8 são apresentadas as conclusões deste trabalho e sugestões de futuras funcionalidades para o ambiente.

2. Aprendizado de Máquina Supervisionado

Aprendizado indutivo é o processo de aprendizado efetuado a partir do raciocínio sobre exemplos fornecidos por um processo externo ao aprendiz. Em AM, o aprendiz é um sistema computacional freqüentemente denotado por *sistema de aprendizado*, *algoritmo de aprendizado*, ou simplesmente *indutor*. Um sistema de aprendizado é um sistema computacional que toma decisões baseado em experiências acumuladas contidas em casos resolvidos com sucesso. O aprendizado indutivo por exemplos pode ser dividido em *aprendizado supervisionado* e *não supervisionado*.

No aprendizado supervisionado é fornecido ao sistema de aprendizado um conjunto de exemplos $E = \{E_1, E_2, \ldots E_N\}$, sendo que cada exemplo $E_i \in E$ possui um rótulo associado. Esse rótulo define a *classe* a qual o exemplo pertence. Um pouco mais formalmente, pode-se dizer que cada exemplo $E_i \in E$ é uma tupla $E_i = (\vec{x_i}, y_i)$ na qual $\vec{x_i}$ é um vetor de valores que representam as características, ou *atributos*, do exemplo E_i, e y_i é o valor da classe desse exemplo. A Tabela 1 mostra a forma geral de um conjunto de exemplos E com N exemplos e M atributos. Essa tabela está no formato *atributo-valor*, o qual é utilizado como entrada pela maioria dos algoritmos de aprendizado. Na forma atributo-valor as colunas ($A_1, \ldots A_M$) da tabela representam os diferentes atributos, e as linhas ($E_1, \ldots E_N$) os diferentes exemplos. O atributo Y é o atributo que assume os valores da classe de cada exemplo E_i.

O objetivo do aprendizado supervisionado é induzir um mapeamento geral dos vetores $\vec{x}$ para valores y. Portanto, o sistema de aprendizado deve construir um modelo, $y = f(\vec{x})$, de uma função desconhecida, f, também chamada de *função conceito*[1], que permite predizer valores y para exemplos previamente não vistos. Entretanto, o número de exemplos utilizados para a criação do modelo não é, na maioria dos casos, suficiente para caracterizar completamente essa

[1] *Concept function*.

Tabela 1: Conjunto de exemplos no formato atributo-valor.

	A_1	A_2	$\cdots$	A_M	Y
E_1	x_{11}	x_{12}	$\cdots$	x_{1M}	y_1
E_2	x_{21}	x_{22}	$\cdots$	x_{2M}	y_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
E_N	x_{N1}	x_{N2}	$\ldots$	x_{NM}	y_N

função f. Na realidade, os sistemas de aprendizado são capazes de induzir uma função **h** que aproxima f, ou seja, $\mathbf{h}(\vec{x}) \approx f(\vec{x})$. Nesse caso, **h** é chamada de *hipótese* sobre a função conceito f.

Em aprendizado supervisionado, o atributo classe Y pode ser um atributo qualitativo que assume um conjunto de valores discretos $C = \{C_1, C_2, \ldots C_{Ncl}\}$ ou um atributo quantitativo que assume um conjunto de valores reais. No primeiro caso, a hipótese **h** é denominada *classificador*, e a tarefa de aprendizado é denotada *classificação*. No segundo caso, a hipótese **h** é denominada *regressor*, e a tarefa de aprendizado é denotada *regressão*. Neste trabalho consideramos sistemas de aprendizado supervisionado para tarefas de classificação.

3. Objetivos do Ambiente SNIFFER

Durante a fase de análise de requisitos foram identificadas diversas funcionalidades necessárias para que o ambiente SNIFFER fosse capaz de dar suporte às atividades envolvidas em análises experimentais. Alguns dos principais requisitos identificados são:

Integração com diversos sistemas AM Existem disponíveis em domínio público diversos sistemas de AM. Sistemas como o C4.5 [Quinlan, 1988] são amplamente utilizados pela comunidade, e se tornaram referencias com os quais pesquisadores freqüentemente comparam novos métodos de AM ou modificações a métodos existentes. Na nossa opinião, é interessante que esses sistemas sejam utilizados em suas implementações originais, isso porque re-implementações desses sistemas podem adicionar características indesejáveis ou remover características importantes se comparadas com a versão original. Além disso, consideramos que modificar as implementações desses sistemas não é aconselhável pois essas modificações podem introduzir erros de codificação entre outras conseqüências indesejadas. Também é importante disponibilizar diversos sistemas de AM, de forma que o usuário possa avaliar um novo método de AM de forma mais ampla;

Conversão de sintaxe dos arquivos de entrada Como dito anteriormente, a comunidade de AM freqüentemente utiliza sistemas de aprendizado de domínio público como referência para comparações de desempenho dos novos métodos propostos. Entretanto, como esses sistemas de aprendizado foram desenvolvidos por diferentes pesquisadores, eles não utilizam uma mesma sintaxe para representar os dados de entrada. Mesmo considerando que esses sistemas trabalham sobre dados em um mesmo formato, conhecido como formato atributo-valor, cada sistema de aprendizado utiliza uma sintaxe própria para descrever esses dados. A conversão manual entre as diversas sintaxes existentes utilizando ferramentas não específicas, como editores de texto e planilhas, pode ser altamente trabalhosa e propensa a erros. O ambiente SNIFFER realiza conversões automáticas de sintaxe, para isso foi proposta uma sintaxe padrão para declaração de dados e atributos chamada DSX. Mais detalhes sobre a sintaxe DSX podem ser encontrados em [Batista, 2003];

Aplicação de reamostragem para estimar a taxa de erro verdadeira Com o objetivo de avaliar a precisão do conhecimento induzido, é prática comum em AM a utilização de métodos

de *reamostragem*, tal como o método de validação cruzada k*-fold cross-validation*. Esses métodos consistem em dividir o conjunto de dados em dois conjuntos mutuamente exclusivos: um *conjunto de treinamento* que é fornecido a um sistema de aprendizado para a extração do conhecimento, e um *conjunto de teste* que é utilizado exclusivamente para medir a precisão do conhecimento induzido. A utilização de métodos de reamostragem acarreta em um aumento do trabalho necessário para avaliar experimentalmente um método de AM. Por exemplo, o método mais frequentemente utilizado, *10-fold cross-validation*, exige a criação de 10 pares de conjuntos de treinamento e teste. Outros métodos de reamostragem, como o *bootstrapping* e o *leaving-one-out*, podem exigir ainda mais iterações, podendo ultrapassar uma centena. Dessa maneira, é altamente trabalhoso gerar os conjuntos de treinamento e teste manualmente. Além disso, é aconselhável a utilização de procedimentos computacionais para a implementação dos métodos de reamostragem, uma vez que esses métodos assumem que os exemplos presentes no conjunto de dados a ser reamostrado encontram-se em ordem aleatória. Permitir uma reamostragem manual dos dados que não assegure a suposição de aleatoriedade pode levar a introdução de algum viés na separação dos conjuntos de treinamento e teste, e distorcer as medidas de desempenho;

Conversão de sintaxe dos arquivos de saída de algoritmos de AM simbólico Os diversos sistemas AM simbólico disponíveis não utilizam uma sintaxe padrão para expressar o conhecimento induzido por esses sistemas. O conhecimento adquirido é freqüentemente expresso na forma de uma *árvore de decisão* ou de um conjunto de *regras de decisão*. De forma similar ao que ocorre com os dados de entrada, diferentes sistemas de aprendizado utilizam diferentes sintaxes para representar o conhecimento induzido, o que pode dificultar a análise e integração de bases de conhecimento induzidas por esses sistemas. Para facilitar a análise e a integração de diferentes bases de conhecimento, foi desenvolvida uma sintaxe padrão para regras chamada $\mathcal{PBM}$ [Prati et al., 2001]. Uma vez que o conhecimento está em uma sintaxe comum, é possível integrá-lo de diversas formas como, por exemplo, utilizando *ensembles* [Wolpert, 1992], ou pela composição de classificadores simbólicos em um classificador final também simbólico, como no sistema XRULER [Baranauskas, 2001] e no sistema MCE [Bernardini, 2002].

Recuperação de matrizes de confusão Uma forma bastante comum de avaliar a qualidade do conhecimento induzido é por meio da precisão de classificação, freqüentemente representada por uma *matriz de confusão*. Uma matriz de confusão apresenta o erro cometido pelo classificador para cada um dos valores do atributo classe. Os principais sistemas de aprendizado aceitam conjuntos de treinamento e teste para a criação do modelo e a estimativa da taxa de erro verdadeira, respectivamente. Normalmente, os sistemas de aprendizado gravam a matriz de confusão em um arquivo texto com sintaxe proprietária. O ambiente SNIFFER recupera a matriz de confusão no conjunto de teste, e a armazena em uma estrutura de dados para posterior análise dos resultados;

Cálculo e comparação de medidas de desempenho Recuperadas as matrizes de confusão para cada iteração do método de reamostragem, é necessário calcular estatísticas que forneçam um indicativo do desempenho do sistema de aprendizado no conjunto de dados analisado. O ambiente SNIFFER calcula medidas de desempenho para cada valor do atributo classe e, também, o desempenho geral levando em consideração todos os valores do atributo classe em conjunto. Uma vez que as medidas de desempenho foram calculadas é necessário compará-las para verificar se existem diferenças significativas entre elas, ou seja, se um método superou outro com uma diferença de 95% ou 99% de confiabilidade;

Publicação dos resultados Um outro requisito é automatizar, sempre que possível, a publicação dos resultados. Dessa forma, o ambiente fornece ao usuário uma segurança maior de que os resultados publicados são fiéis aos resultados obtidos nos experimentos. Como, freqüentemente, os resultados dos experimentos envolvem uma grande quantidades de valores numéricos, é

muito comum a introdução de erros durante o processo de confecção de tabelas e gráficos. O ambiente SNIFFER fornece ao usuário relatórios resumidos e detalhados dos resultados obtidos, além de tabular os resultados em um formato que pode ser utilizado para gerar gráficos com o utilitário Gnuplot[2], e tabelas no processador de textos LaTeX[3].

Um exemplo típico de experimento em AM pode ser ilustrado pelos experimentos realizados em [Batista and Monard, 2003] para tratamento de valores desconhecidos nos dados, os quais foram realizados utilizando *10-fold cross-validation*, com 7 proporções de valores desconhecidos (0%, 10%, 20%, 30%, 40%, 50% e 60%) inseridos nos conjuntos de treinamento artificialmente, 8 valores para o parâmetro k do algoritmo K-VIZINHOS MAIS PRÓXIMOS[4] (1, 3, 5, 10, 20, 30, 50, 100), os experimentos executados com 2 sistemas de aprendizado diferentes (C4.5 e CN2) sobre 4 conjuntos de dados diferentes. Portanto, no total, foram realizadas $10 \times 7 \times 8 \times 2 \times 4 = 4480$ execuções dos sistema de aprendizado.

4. Trabalhos Relacionados

Existem diversos ambientes computacionais que compartilham objetivos e características com o ambiente SNIFFER. Alguns dos principais ambientes computacionais em domínio público são: o $\mathcal{MLC}^{++}$[5] [Kohavi et al., 1997]; WEKA[6] [Witten and Frank, 2000]; e YALE[7] [Fischer et al., 2002].

$\mathcal{MLC}^{++}$ apresenta diversas facilidades, tais como interfaces para os principais algoritmos de aprendizado, formato padrão para os dados de entrada, obtenção de estatísticas de desempenho e visualização gráfica das estruturas simbólicas obtidas por alguns algoritmos. Entretanto, essa biblioteca trata os classificadores como "*caixas pretas*", não fornecendo uma visão única dos classificadores simbólicos que podem ser extraídos utilizando a biblioteca. A incorporação de novos aplicativos à $\mathcal{MLC}^{++}$ também não é uma tarefa trivial, já que é necessário recompilar a biblioteca a cada novo aplicativo adicionado, além da dificuldade de adaptação do novo aplicativo aos padrões da $\mathcal{MLC}^{++}$. Além disso, a partir de 1995 a $\mathcal{MLC}^{++}$ passou a ser de propriedade da SGI[8], e a última versão disponibilizada data de 1997.

WEKA, ao contrário da biblioteca $\mathcal{MLC}^{++}$, reimplementa os algoritmos de aprendizado na linguagem Java[9]. Essa abordagem padroniza as interfaces e produz código uniforme, facilitando a inclusão de novos aplicativos. Entretanto, as novas versões dos algoritmos originais, ou até mesmo de novos algoritmos propostos pela comunidade, podem não ser disponibilizados no WEKA, pois exigem a sua conversão em código Java. Além disso, como discutido previamente, a recodificação de algoritmos sempre está sujeita a falhas, as quais podem causar um comportamento anômalo do algoritmo reimplementado que não ocorre no algoritmo original.

O ambiente YALE também está sendo implementado em Java. Esse ambiente utiliza alguns dos algoritmos reimplementados no ambiente WEKA, mas também é capaz de executar alguns algoritmos em suas implementações originais.

É importante ressaltar que os ambientes $\mathcal{MLC}^{++}$, WEKA e YALE possuem objetivos mais abrangentes que o ambiente SNIFFER, uma vez que esses ambientes provêm não somente suporte

[2] `http://www.gnuplot.info`.
[3] `http://www.latex-project.org`.
[4] O algoritmo k-vizinhos mais próximos foi utilizado como método de predição para imputar os valores desconhecidos.
[5] *Machine Learning Library in* C++. `http://www.sgi.com/tech/mlc`.
[6] *Waikato Environment for Knownledge Analysis*. `http://www.cs.waikato.ac.nz/ml/weka`.
[7] *Yet Another Learning Environment*. `http://yale.cs.uni-dortmund.de`.
[8] `http://www.sgi.com`.
[9] `http://java.sun.com`.

à realização de experimentos, mas também bibliotecas de classes que implementam diversas funcionalidades importantes em AM. Os ambientes WEKA e YALE provêm também uma interface gráfica de fácil utilização. Por outro lado, o ambiente SNIFFER provê uma estrutura muito mais flexível que permite realizar experimentos que comparam diversos métodos e algoritmos de AM em uma única execução.

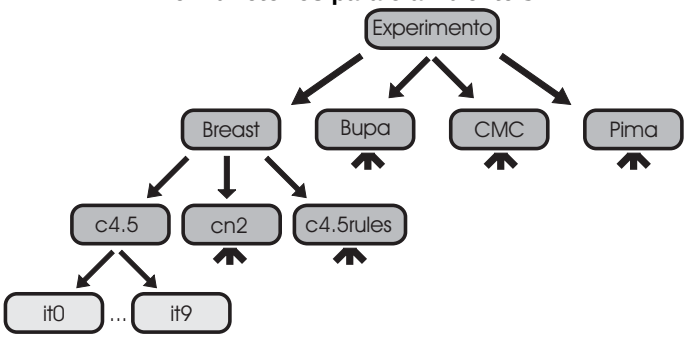
5. O Funcionamento do Ambiente Computacional SNIFFER

O ambiente SNIFFER pode ser executado de duas maneiras diferentes. A primeira, e a mais usual, é na forma de um utilitário que executa e organiza um experimento de AM. Ao final da execução do utilitário, o usuário encontrará um conjunto de relatórios que fornecem visões gerais e detalhadas das execuções dos algoritmos de AM, entre outras informações. Uma segunda maneira de executar o ambiente SNIFFER é utilizar uma API[10] que o ambiente fornece à aplicações clientes. Essa API permite criar novas aplicações que manipulem os resultados obtidos nos experimentos. Por exemplo, por meio da API o usuário pode avaliar os sistemas de aprendizado sob novas medidas de desempenho [Provost and Fawcett, 2001] ou implementar novos testes estatísticos de hipótese [Dietterich, 1997b].

Para organizar um experimento, o ambiente computacional SNIFFER utiliza a estrutura do sistema de arquivos do sistema operacional no qual está sendo executado. Os conjuntos de dados a serem analisados são dispostos em diretórios, sendo que alguns identificadores de diretórios possuem significado especial para o ambiente.

A seguir é utilizado um exemplo para tornar mais simples as explicações sobre o funcionamento do ambiente. Vamos imaginar que desejamos comparar o desempenho de três sistemas de aprendizado: C4.5, C4.5 RULES e CN2 sobre quatro conjuntos de dados amplamente conhecidos pela comunidade de AM: **Breast**, **Bupa**, **CMC** e **Pima** [Blake and Merz, 1998]. Inicialmente, o usuário deve organizar os conjuntos de dados em diretórios. A forma que os diretórios devem ser organizados depende de como o usuário deseja que os testes de hipótese sejam feitos. Por hora, vamos pensar que os conjuntos de dados foram organizados pelo usuário da forma em que são mostrados na Figura 1.

Figura 1: Exemplo de experimento organizado em diretórios para o ambiente SNIFFER.



Existe um diretório raiz para o experimento chamado `Experimento`. Dentro desse diretório, o qual engloba todo o experimento, foram criados quatro diretórios: `Breast`, `Bupa`, `CMC`, e `Pima`, ou seja, um para cada conjunto de dados que será analisado. Dentro de cada diretório dos conjuntos de dados foram criados outros três diretórios: `c4.5`, `cn2` e `c4.5rules` um para cada sistema de aprendizado a ser empregado. Na Figura 1 são mostrados somente os diretórios contidos no diretório `Breast`, os demais foram omitidos por simplificação. For fim, opcionalmente, o usuário pode criar diretórios numerados que seguem o padrão `it0`, `it1`, ... `it(k-1)`, um para cada iteração do método de reamostragem, sendo k o número total de iterações do método, como explicado mais à frente.

[10] *Application Programming Interface*.

Ao ambiente SNIFFER deve ser fornecido um *ponto de entrada*, ou seja, um diretório que seja o raiz para todo o experimento. A partir desse diretório, o ambiente vasculha a árvore de diretório contida no ponto de entrada a procura de diretórios com identificadores especiais. Os identificadores especiais identificam um sistema de aprendizado específico. Atualmente, os identificadores especiais que o ambiente reconhece estão listados na Tabela 2. Entretanto, o ambiente SNIFFER está preparado para acomodar outros sistemas de aprendizado, como descrito na Seção 7. Quando um diretório com identificador especial é encontrado, o ambiente SNIFFER realiza as seguintes tarefas:

Tabela 2: Os identificadores especiais para diretórios utilizados atualmente pelo ambiente SNIFFER.

Identificador	Sistema de Aprendizado
`c4.5`	C4.5 [Quinlan, 1988]
`c4.5rules`	C4.5 RULES [Quinlan, 1987]
`id3`	ID3 [Quinlan, 1986]
`cn2`	CN2 [Clark and Boswell, 1991]
`newid`	NEWID [Boswell, 1990]

1. Caso for solicitado pelo usuário, o ambiente executa um método de reamostragem dividindo os dados em diversos conjuntos de treinamento e teste. Nesse caso, arquivos com extensão `.data` e `.names` precisam estar presentes no diretório com o identificador especial. Como resultado, são criados diversos diretórios com os identificadores `it0`, `it1`, ... `it(k-1)`, um para cada iteração do método de reamostragem, sendo k o número total de iterações do método. Dentro de cada um desses diretórios o ambiente armazena três arquivos com as extensões `.data`, `.test` e `.names`, os quais contém, respectivamente, o conjunto de treinamento e teste dessa iteração, e o arquivo de declaração de atributos relacionado aos conjuntos de treinamento e teste;
2. Por outro lado, o usuário pode não desejar que o ambiente aplique um método de reamostragem e, em vez disso, o usuário pode criar os diretórios com os conjuntos de treinamento e teste na forma `it0`, `it1`, ... `it(k-1)`. Isso é útil em algumas situações como, por exemplo, quando o usuário deseja avaliar dois ou mais sistemas de aprendizado nos mesmos conjuntos de treinamento e teste, com o objetivo de utilizar um teste de hipótese pareado. Nesse caso, o usuário solicita que o ambiente não execute um método de reamostragem, e o ambiente utiliza os arquivos de dados com as extensões `.data`, `.test` e `.names`, que foram armazenados pelo usuário, dentro de cada diretório com o identificador `it`.

Uma vez que o ambiente se certificou de que os dados estão divididos em conjuntos de treinamento e teste, o ambiente passa a executar sobre esses dados o sistema de aprendizado identificado pelo nome do diretório atual – Tabela 2. Para isso, o ambiente acessa cada diretório `it` e realiza as seguintes tarefas:

1. Caso for solicitado pelo usuário, o ambiente SNIFFER converte a sintaxe dos arquivos de dados para a sintaxe do sistema de aprendizado a ser executado;
2. O ambiente executa o sistema de aprendizado selecionado sobre o conjunto de treinamento, e mede o erro de classificação sobre o conjunto de teste;
3. Caso o algoritmo de AM seja simbólico, o ambiente converte as regras que se encontram na sintaxe proprietária do sistema de aprendizado para a sintaxe $\mathcal{PBM}$;
4. Por fim, o ambiente extrai dos arquivos de saída do sistema de aprendizado a matriz de confusão com as classificações incorretas no conjunto de teste. As matrizes de confusão são armazenadas pelo ambiente para a realização de cálculos de desempenho.

O ambiente SNIFFER grava em cada diretório `it` alguns arquivos com o resultado da execução do sistema de aprendizado correspondente em uma única iteração de um método de reamostragem. Esses arquivos possuem como nome o mesmo identificador do arquivo de dados

sobre o qual o sistema de aprendizado foi executado, cada arquivo é diferenciado apenas pela extensão. As extensões utilizadas são:

.out Toda saída direcionada para o dispositivo padrão de saída pelo sistema de aprendizado é redirecionada para esse arquivo. A utilidade desse arquivo pode variar de sistema de aprendizado para sistema de aprendizado. Para alguns sistemas de aprendizado, como o C4.5, essa saída é a forma mais simples de verificar as regras e a precisão do classificador induzido. Outros sistemas de aprendizado, como o CN2, gravam as regras em arquivo texto separado. Deve ser observado que esse arquivo é muito útil para verificar como foi o processo de indução, por exemplo, a geração do classificador e a poda. Esse arquivo pode também ajudar a identificar os erros quando a execução de um sistema de aprendizado falha;

.rules Alguns sistemas de aprendizado, como o CN2 e o NEWID, permitem que as regras induzidas sejam gravadas em arquivos texto. Nesses casos, o ambiente SNIFFER grava essas regras em um arquivo com essa extensão;

.stdrules O ambiente SNIFFER converte as regras na sintaxe proprietária para a sintaxe $\mathcal{PBM}$, e as armazena em um arquivo com a extensão .stdrules.

Todos os resultados de desempenho obtidos com a aplicação de um método de reamostragem são armazenados e identificados com uma *chave*. Uma chave é o caminho composto pelos identificadores dos diretórios desde o ponto de entrada até o identificador do sistema de aprendizado a ser executado. Por exemplo, `./Experimento/Breast/c4.5` é a chave de identificação dos resultados obtidos pela execução do sistema de aprendizado C4.5 no conjunto de dados **Breast**, o qual é o caminho mais à esquerda na Figura 1.

O ambiente SNIFFER percorre toda a árvore de diretórios contida no ponto de entrada. Ao final da busca, podem ser realizadas comparações entre os resultados. Atualmente, o ambiente utiliza o teste-t pareado para *k-fold cross-validation*[11] [Dietterich, 1997a]. Para restringir as comparações, o usuário pode especificar sub-árvores de diretórios nas quais os resultados devem ser comparados. Por exemplo, como pode não fazer sentido comparar resultados obtidos em conjuntos de dados de domínios diferentes, o usuário pode especificar `./Experimento/Breast` para que somente os resultados que estão localizados dentro desse diretório sejam comparados, ou seja, para que somente os resultados obtidos pelos sistemas de aprendizado C4.5, CN2 e C4.5 RULES sejam comparados entre si para o conjunto de dados **Breast**. Para que o ambiente compare os resultados de cada conjunto de dados em separado, o usuário deve especificar uma lista com os diretórios `[./Experimento/Breast, ./Experimento/Bupa, ./Experimento/CMC, ./Experimento/Pima]`.

Ao final do cálculo dos testes de comparação, o ambiente pode tanto gerar relatórios descrevendo os resultados obtidos, quanto ser acessado por meio de sua API para que o usuário recupere determinados dados ou estatísticas de desempenho. Em ambos os casos, as chaves descritas anteriormente são utilizadas para identificar os resultados.

O ambiente SNIFFER gera quatro relatórios de resultados, são eles:

Summary Esse relatório apresenta um resumo das estatísticas calculadas pelo ambiente. São apresentados os erros médio e os desvios padrão das k iterações do método de reamostragem. Os erros médio e os desvios padrão são calculados tanto para cada valor do atributo classe individualmente quanto para todos os valores possíveis do atributo classe;

Detailed Esse relatório é gerado para cada execução de um método de reamostragem. Ele apresenta, para cada iteração do método de reamostragem, a matriz de confusão e as taxas de erro para cada valor do atributo classe em separado, e para todos os valores em conjunto. Ao final,

[11] *k-fold cross-validation paired t-test.*

esse relatório apresenta um resumo de todas as iterações, além da taxa de erro média e do desvio padrão para todas as interações;

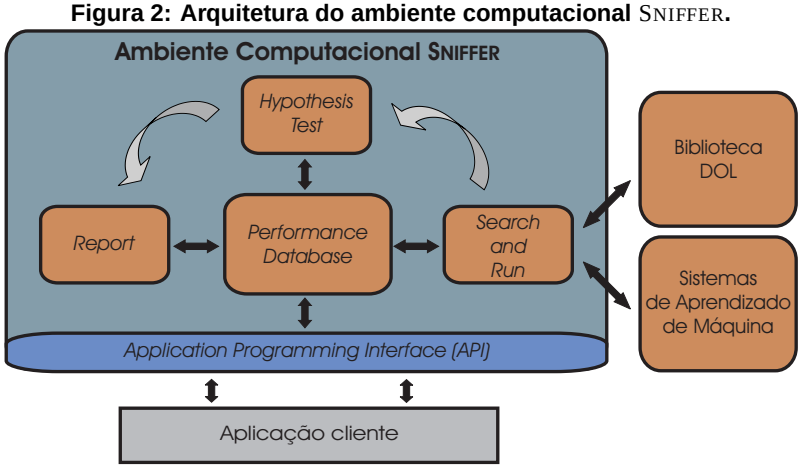
HypothesisTest Os resultados dos testes de hipótese comparando dois ou mais resultados obtidos com a aplicação de um método de reamostragem são apresentados nesse relatório. Os resultados apresentados respeitam as restrições de comparação impostas pelo usuário. Os testes de hipótese são realizados para cada valor do atributo classe individualmente e para todos os valores em conjunto. Ainda, o relatório indica quando um resultado é estatisticamente significativo (com 95% de confiança) ou altamente significativo (com 99% de confiança);

GnuPlot O último relatório gerado apresenta os mesmos resultados do relatório `Summary`, mas em um formato mais simples. Esse relatório está no mesmo formato de dados utilizado pelo utilitário de geração de gráficos GnuPlot™, mas também pode ser utilizado por outros utilitários, bem como para a geração de tabelas no processador de textos LaTeX.

6. A Arquitetura do Ambiente Computacional SNIFFER

O objetivo desta seção é apresentar uma descrição da arquitetura do ambiente SNIFFER. O ambiente utiliza uma biblioteca de classes chamada DISCOVER OBJECT LIBRARY – DOL [Batista, 2003]. A biblioteca DOL fornece um conjunto de métodos para pré-processamento de dados, dentre eles a aplicação de métodos de reamostragem e a conversão da sintaxe DSX para a sintaxe utilizada por diversos sistemas de aprendizado. Na Figura 2 é apresentada uma representação gráfica da arquitetura do ambiente. O ambiente conta com quatro módulos principais, são eles:

Figura 2: Arquitetura do ambiente computacional SNIFFER.



SearchandRun Esse módulo vasculha a árvore de diretórios contida no ponto de entrada a procura de diretórios com identificadores especiais. Quando um diretório com identificador especial é encontrado, esse módulo realiza todas as tarefas necessárias para executar o sistema de aprendizado, executa o sistema de aprendizado e extrai a matriz de confusão no conjunto de teste. As matrizes de confusão extraídas são armazenadas e gerenciadas por outro módulo chamado PerformanceDatabase. Esse módulo é responsável por armazenar e permitir consultas a todos os dados e estatísticas obtidas pelo ambiente SNIFFER. O módulo SearchandRun faz a interface com os sistemas de aprendizado. Além disso, esse módulo utiliza a biblioteca DOL para dividir o conjunto de dados em conjuntos de treinamento e teste, bem como para realizar conversões entre sintaxes. Ao final da busca, o módulo SearchandRun ativa o módulo HypothesisTest, o qual realiza os testes de hipótese com o objetivo de identificar diferenças significativas entre os resultados;

HypothesisTest Esse módulo aplica os testes de hipótese sobre os resultados obtidos com a finalidade de identificar diferenças significativas. Como descrito anteriormente, esse módulo utiliza atualmente o teste-t pareado para *k-fold cross-validation*. Outros testes podem ser implementados utilizando a API do ambiente SNIFFER, ou por meio da criação de uma classe específica para o novo teste de hipótese, como discutido na Seção 7. Os resultados dos testes de hipótese são armazenados e gerenciados pelo módulo PerformanceDatabase;

PerformanceDatabase O módulo PerformanceDatabase implementa um banco de dados com dados e estatísticas sobre o desempenho dos sistemas de aprendizado executados. Os dados que esse módulo armazena vêm dos módulos SearchandRun e HypothesisTest. O módulo Report acessa os dados armazenados nesse banco de dados para organizar os relatórios fornecidos ao usuário. Ainda, a API do ambiente SNIFFER fornece ao usuário acesso aos dados armazenados no módulo PerformanceDatabase.

Report O módulo Report organiza as informações presentes no módulo PerformanceDatabase em relatórios gravados em arquivos texto. Esses relatórios fornecem ao usuário visões gerais e detalhadas do desempenho dos sistemas de aprendizado, e preparam os dados para serem utilizados por outros aplicativos.

7. O Projeto do Ambiente Computacional SNIFFER

O objetivo desta seção é fornecer uma visão geral de como o ambiente SNIFFER foi projetado e implementado. De uma forma geral, o ambiente foi projetado tendo em vista possíveis extensões que podem ser adicionadas a ele. Nesse caso, alguns padrões de projeto [Gamma et al., 1995] ajudam a melhorar o projeto do ambiente, tornando o projeto flexível o bastante para que novas funcionalidades sejam adicionadas ao ambiente, sem que exista a necessidade de realizar grandes modificações no projeto original. Algumas modificações com as quais o ambiente SNIFFER terá que lidar no futuro são:

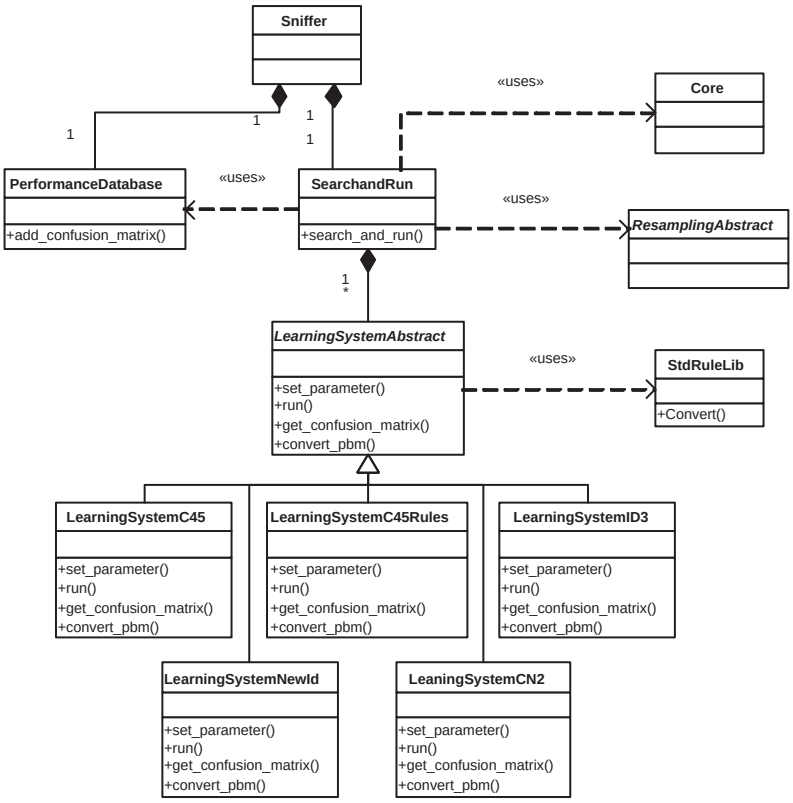
Novos sistemas de aprendizado Pode-se desejar adicionar um sistema de aprendizado supervisionado que ainda não foi integrado ao ambiente SNIFFER. Nesse caso, o ambiente SNIFFER utiliza classes que "envolvem" os sistemas de aprendizado fazendo com que esses sistemas tenham o mesmo funcionamento do ponto de vista externo. Adicionar um novo sistema de aprendizado significa criar uma dessas classes específica para o novo sistema de aprendizado;

Novos métodos de reamostragem Diferentes métodos de reamostragem podem ser empregados em conjuntos de dados com características distintas. Por exemplo, o método *k-fold cross-validation* pode ser utilizado em conjuntos de dados de tamanho médio com excelente precisão. Para conjuntos de dados menores (abaixo de 200 exemplos), o método *bootstraping* [Weiss and Kulikowski, 1991] é mais recomendado. O ambiente SNIFFER utiliza a biblioteca DOL para aplicar métodos de reamostragem nos conjuntos de dados. Para adicionar novos métodos de reamostragem ao ambiente SNIFFER basta que o método de reamostragem seja adicionado a biblioteca DOL;

Novos testes de hipótese A comunidade de AM ainda não chegou a um consenso sobre quais testes de hipóteses devem ser utilizados para comparar o desempenho de dois sistemas de aprendizado [Dietterich, 1997b]. Novos testes de hipótese podem ser integrados ao ambiente SNIFFER de duas maneiras diferentes: ou por meio da API que o ambiente proporciona; ou por meio da criação de uma classe específica para esse fim.

Na Figura 3 é apresentado o projeto do módulo SearchandRun. Esse módulo possui uma classe principal também chamada `SearchandRun`. Essa classe percorre a árvore de diretórios em busca de diretórios com identificadores especiais, como explicado na Seção 5. A classe `SearchandRun` utiliza as classes `Core` e `ResamplingAbstract` de biblioteca DOL

Figura 3: Projeto do módulo SearchandRun **do ambiente computacional** SNIFFER**.**



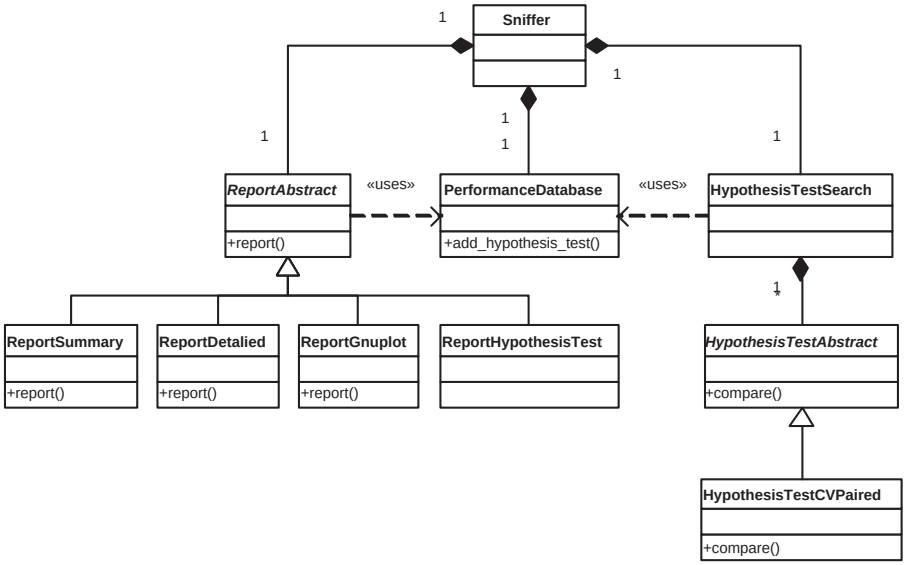
para converter entre as sintaxes dos sistemas de aprendizado e dividir os dados em conjuntos de treinamento e teste, respectivamente.

A classe `SearchandRun` utiliza a classe `LearningSystemAbstract` para realizar a interface entre o módulo e os diversos sistemas de aprendizado. A classe `LearningSystemAbstract` define métodos para ajustar os parâmetros do sistema de aprendizado, executar esse sistema, obter a matriz de confusão no conjunto de teste e converter as regras induzidas por algoritmos de AM simbólico para, o formato $\mathcal{PBM}$. Para realizar a conversão para o formato $\mathcal{PBM}$, a classe `LearningSystemAbstract` utiliza a classe `StdRuleLib`. As sub-classes da classe `LearningSystemAbstract` implementam os métodos descritos anteriormente para cada sistema de aprendizado suportado pelo ambiente SNIFFER.

A classe `SearchandRun` utiliza a classe `PerformanceDatabase` para armazenar as matrizes de confusão obtidas nas execuções dos sistemas de aprendizado. A classe `Sniffer` faz a interface entre os módulos internos do ambiente SNIFFER e a aplicação externa. A classe `Sniffer` também provê a API para que classes externas ao ambiente possam estender as suas funcionalidades.

Na Figura 4 é apresentado o projeto dos módulos Report e HypothesisTest. A classe `HypothesisTestSearch` procura por resultados a serem comparados respeitando as limitações impostas pelo usuário. Quando dois resultados a serem comparados são encontrados, essa classe utiliza a classe `HypothesisTestAbstract` para compará-los. A classe `HypothesisTestAbstract` implementa a estrutura comum entre diversos testes de hipótese, e as sub-classes de `HypothesisTestAbstract` implementam testes de hipótese específicos. Atualmente, o teste-t pareado para *k-fold cross-validation* está implementado pela classe `HypothesisTestCVPaired`.

Figura 4: Projeto dos módulos Report **e** HypothesisTest **do ambiente computacional** SNIFFER**.**



A classe `ReportAbstract` cria a estrutura para a implementação de relatórios. Atualmente, quatro classes derivadas da classe `ReportAbstract` implementam os relatórios disponibilizados pelo ambiente: `Summary`, `Detailed`, `GnuPlot` e `HypothesisTest`.

8. Conclusão

Neste trabalho foram apresentadas descrições a respeito do funcionamento, da arquitetura e do projeto do ambiente SNIFFER. O ambiente SNIFFER tem como principal objetivo gerenciar experimentos em AM, os quais são geralmente muito trabalhosos e propensos à introdução de erros. O ambiente utiliza a árvore de diretórios do sistema operacional para organizar os experimentos, essa estrutura se mostrou na prática bastante flexível e funcional, sendo uma das características do ambiente que consideramos inovadora.

Como trabalhos futuros, acreditamos que a geração de relatórios com a complexidade sintática do conhecimento induzido para os algoritmos de AM simbólico é uma funcionalidade relevante para os usuários que desejam avaliar a qualidade do conhecimento induzido não apenas pela sua precisão de classificação, mas também pela compreensibilidade do conhecimento. Outras funcionalidades desejáveis são a geração de gráficos ROC [Provost and Fawcett, 2001] e relatórios utilizando a área sob a curva ROC (AUC) como medida de desempenho, e a utilização de outros testes de hipótese, como por exemplo, o teste de McNemar [Dietterich, 1997b].

Agradecimentos. Os autores agradecem à CAPES e à FAPESP pelo auxílio parcial nesta pesquisa.

Referências

Baranauskas, J. A. (2001). Extração Automática de Conhecimento por Múltiplos Indutores. Tese de Doutorado, ICMC-USP, `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-08102001-112806`.

Batista, G. E. A. P. A. (2003). Pré-processamento de Dados em Aprendizado de Máquina Supervisionado. Tese de Doutorado, ICMC-USP, `http://www.icmc.usp.br/~gbatista`.

Batista, G. E. A. P. A. and Monard, M. C. (2003). An Analysis of Four Missing Data Treatment Methods for Supervised Learning. *Applied Artificial Intelligence*, 17(5):519–533. `http://www.icmc.usp.br/~gbatista`.

Bernardini, F. C. (2002). Combinação de Classificadores Simbólicos para Melhorar o Poder Preditivo e Descritivo de *Ensembles*. Dissertação de Mestrado, ICMC-USP.

Blake, C. and Merz, C. (1998). UCI Repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Boswell, T. (1990). Manual for NewId version 4.1. Technical Report TI/P2154/RAB/4/2.3, The Turing Institute.

Clark, P. and Boswell, R. (1991). Rule Induction with CN2: Some Recent Improvements. In Kodratoff, Y., editor, *Fifth European Conference (EWSL 91)*, pages 151–163. Springer-Verlag.

Dietterich, T. G. (1997a). Limitations on Inductive Learning (Extended Abstract). Technical report, Oregon State University. `ftp://ftp.cs.orst.edu/pub/tgd/papers`.

Dietterich, T. G. (1997b). Statistical Tests for Comparing Supervised Classification Learning Algorithms. Technical report, Oregon State University. `ftp://ftp.cs.orst.edu/pub/tgd/papers/stats.ps.gz`.

Fischer, S., Klinkenberg, R., Mierswa, I., and Ritthoff, O. (2002). Yale: Yet Another Learning Environment — Tutorial. Technical Report CI-136/02, Collaborative Research Center 531, University of Dortmund.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Resusable Object-Oriented Software*. Addison Wesley.

Kibler, D. and Langley, P. (1988). Machine Learning as an Experimental Science. *Machine Learning*, 3(1):5–8.

Kohavi, R., Sommerfield, D., and Dougherty, J. (1997). Data Mining Using $\mathcal{MLC}^{++}$: A Machine Learning Library in C++. *International Journal on Artificial Intelligence Tools*, 6(4):537–566.

Prati, R. C., Baranauskas, J. A., and Monard, M. C. (2001). Extração de Informações Padronizadas para a Avaliação de Regras Induzidas por Algoritmos de Aprendizado de Máquina Simbólico. Technical Report 145, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_145.ps.zip`.

Provost, F. J. and Fawcett, T. (2001). Robust Classification for Imprecise Environments. *Machine Learning*, 42(3):203–231.

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, 1:81–106. Reprinted in Shavlik and Dieterich (eds.) Readings in Machine Learning.

Quinlan, J. R. (1987). Generating Production Rules from Decision Trees. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 304–307, Italy.

Quinlan, J. R. (1988). *C4.5 Programs for Machine Learning*. Morgan Kaufmann, CA.

Weiss, S. M. and Kulikowski, C. A. (1991). *Computer Systems that Learn*. Morgan Kaufmann, San Mateo, CA.

Witten, I. H. and Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.

Wolpert, D. H. (1992). Stacked Generalization. *Neural Networks*, 5:241–259.