# A Biblioteca DOL para Mineração de Dados e Séries Temporais *

**Lucas Fernando Rosada**[1], **Gustavo E.A.P.A Batista**[1,2]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo
Laboratório de Inteligência Computacional – LABIC
Caixa Postal 668, 13560-970 – São Carlos, SP, Brasil

[2]Centro de Estudos Avançados em Segurança de Barragens – CEASB
Parque Tecnológico Itaipu – PTI

lucasrosad@grad.icmc.usp.br, gbatista@icmc.usp.br

***Abstract.*** *This work presents the design, implementation and last developments of the Discover Object Library – DOL. The library main objective is to provide a framework to the development and evaluation of new Data Mining algorithms. DOL acts more directly on the pre and post-processing phases of the Data Mining process and has been employed as basis for the development of several research prototypes.*

***Resumo.*** *Este trabalho apresenta o projeto, implementação e últimos desenvolvimentos da biblioteca Discover Object Library – DOL. Essa biblioteca tem como principal objetivo prover um framework para o desenvolvimento e avaliação de novos algoritmos de Mineração de Dados. A DOL atua mais diretamente nas fases de pré e pós-processamento do processo de Mineração de Dados e já foi utilizada como base para o desenvolvimento de diversos protótipos de pesquisa.*

## 1. Introdução

A pesquisa em Aprendizado de Máquina – AM – requer que os novos métodos desenvolvidos sejam avaliados experimentalmente e comparados com outros métodos publicados na literatura. A existência de diversos sistemas de aprendizado de máquina, bem como a necessidade de rodar os experimentos com uma grande quantidade de conjuntos de dados e possíveis variações de parâmetros dos métodos comparados torna a execução dos experimentos uma tarefa difícil e repetitiva.

*Trabalho desenvolvido com o apoio da Fundação Parque Tecnológico Itaipu – FPTI

Outro fator agravante são os formatos utilizados por cada indutor disponível, o que requer conversão de sintaxe entre os arquivos de dados. Nesse cenário, a criação de uma biblioteca que integre diferentes sistemas de aprendizado e que disponibilize uma interface para criação de novos métodos de AM se faz necessária.

Com esse objetivo foi proposta a biblioteca orientada a objetos *Discover Object Library* – DOL, parte integrante do projeto *Discover* [Prati 2003]. O propósito da biblioteca DOL é promover um *framework* no qual novos métodos de AM possam ser rapidamente implementados [Batista 2003]. Além disso, a biblioteca provê facilidades para a execução dos algoritmos de aprendizado de máquina mais difundidos.

Neste trabalho é apresentado o projeto da biblioteca DOL e as suas principais funcionalidades. Além disso são discutidos os trabalhos em andamento sobre o desenvolvimento dessa biblioteca, incluindo a nova biblioteca *Simplified Discover Object Library* – SDOL – que tem como meta tornar a DOL mais eficiente quanto ao uso de memória principal e processamento.

Este trabalho está organizado da seguinte maneira: na Seção 2 é introduzido o processo de Mineração de Dados, com ênfase nas fases de pré e pós-processamento, principal atuação da DOL; na Seção 3 é apresentada a arquitetura interna da biblioteca, os padrões de projeto utilizados e a sua capacidade de extensão; na Seção 4 é discutida a integração da biblioteca com indutores externos; na Seção 5 são abordados os últimos desenvolvimentos da DOL e as perspectivas futuras para a biblioteca; por fim, na Seção 6 são apresentadas as conclusões deste trabalho.

## 2. Atuação da DOL no Processo de Mineração de Dados

Aprendizado de Máquina tem como objetivo o estudo de técnicas computacionais que permitam a construção de um sistema capaz de adquirir conhecimento. Tal conhecimento é adquirido pelo sistema através de um processo de indução, no qual infere-se logicamente conclusões genéricas a partir de um conjunto de dados específico [Rezende 2003]. Entretanto, a aplicação de sistemas de AM em dados reais apresenta diversas dificuldades, desde a necessidade de pré-processar os dados, com o intuido de melhorar a sua qualidade, até a avaliação e apresentação do conhecimento extraído ao usuário final. Diante dessas dificuldades, foi proposto o processo de Mineração de Dados [Fayyad et al. 1996], o qual pode ser dividido em sete etapas:

1. Identificação do problema e das fontes de dados;

2. Integração dos dados, na qual dados de diferentes fontes são unificados em um só conjunto;
3. Seleção, etapa na qual se seleciona um sub-conjunto relevante à extração de conhecimento;
4. Limpeza dos dados, remoção de inconsistências e dados ruidosos, entre outras atividades;
5. Mineração, execução de algoritmos para extrair os padrões dos dados;
6. Avaliação dos padrões, análise dos padrões realmente válidos, novos e úteis, e;
7. Apresentação do conhecimento, por exemplo, utilizando técnicas de visualização para facilitar o seu entendimento.

Nesse contexto, a biblioteca DOL atua sobretudo na terceira e quinta etapas do processo de Mineração de Dados. O pré-processamento é importante para que sejam passados dados coerentes ao indutor, que executará a busca pelos padrões nos dados. Além disso, é nessa etapa que são realizadas atividades importantes como limpeza, transformação e redução. Essas atividades visam, respectivamente, remover problemas provenientes do processo de coleta, adequar os dados para uso no algoritmo de AM, e, por fim, diminuir quantitativamente ou selecionar qualitativamente os dados, visando diminuir os recursos despendidos durante o processo de mineração [Rezende 2003].

A biblioteca DOL disponibiliza uma série de tratamentos para a execução da etapa de pré-processamento, como filtragem de exemplos através de uma condição lógica, remoção de colunas dentro de modelo atributo-valor, criação de coluna a partir de expressões lógicas, normalização de atributos, além de diversos métodos de tratamento de valores desconhecidos e métodos de amostragem para tratamento do problema de classes desbalanceadas.

Após a mineração dos dados, é necessário avaliar os modelos induzidos. A avaliação é feita de forma diferente para modelos simbólico e não simbólicos. Os modelos não simbólicos, tais como os inferidos por Redes Neurais e Máquinas de Suporte Vetorial, são avaliados segundo as suas capacidades de classificação apenas. Nesses modelos é medida a matriz de confusão em casos de teste e, para problemas que envolvem duas classes, é realizada a análise ROC e calculada a área sob a curva ROC – AUC – [Fawcett 2004]. Para modelos simbólicos, o modelo é convertido para um conjunto de regras, as quais ficam gravadas em um arquivo na sintaxe PBM [Prati et al. 2001]. Para cada regra é calculada a sua matriz de confidência e tal matriz pode ser utilizada para calcular um grande número de medidas de desempenho e de qualidade, por exemplo, a cobertura, o

suporte, precisão, interessabilidade, novidade, entre outras [Lavrač et al. 1999]. A compreensibilidade do conhecimento pelo usuário é altamente importante, sendo possível utilizar a quantidade de regras e a quantidade de condições como medidas para estimar essa qualidade. Um número muito grande de regras ou condições diminui a compreensibilidade de modelo induzido [Rezende 2003].

É possível observar que a biblioteca DOL atua nas etapas de pré e pós-processamento, porém não atua diretamente no processo de indução. De fato, os algoritmos de extração de conhecimento não estão reimplementados diretamente na biblioteca, pois é difícil garantir que uma reimplementação de um algoritmo não irá introduzir diferenças nos resultados, se estes forem comparados com os resultados proporcionados pela implementação original [Batista and Monard 2004]. Quando se trata de pesquisa e comparação de resultados, mesmo pequenas diferenças podem ser indesejadas. A abordagem adotada na biblioteca DOL é diferente da utilizada no sistema Weka [Witten and Frank 2000]. O sistema Weka possui algumas reimplementações de sistemas conhecidos, por exemplo o J48 é uma implementação Java do sistema C4.5. Em [Todorovski and Džeroski 2003] é reportada uma pequena diferença entre os sistemas J48 e C4.5, na média de 0,01% e uma diferença máxima relativa de 4%. A integração da biblioteca com ambientes externos é essencial para a biblioteca contemplar seus requisitos, de forma que há uma interface transparente para o usuário, quando este deseja executar algum dos indutores, e, ao mesmo tempo, a fidelidade dos resultados não é perdida, já que a implementação original dos algoritmos de extração de conhecimento é utilizada.

## 3. A Arquitetura da Biblioteca DOL

A DOL é uma biblioteca orientada a objetos que visa auxiliar na criação, uso e testes de métodos para pré e pós-processamento. Por ser orientada a objetos e por seguir padrões de projeto, a sua extensão é relativamente simples, uma vez entendida sua arquitetura.

Para que a arquitetura da biblioteca fosse de fácil manutenção e entendimento, foi decidido usar padrões de projeto. Padrões de projeto são soluções concisas para problemas recorrentes em orientação a objetos, de maneira que a mesma solução deve ser genérica o suficiente para ser aplicada em mais de um projeto distinto. Dessa forma, o entendimento é mais rápido, uma vez que há vários padrões de projeto altamente difundidos e utilizados, e a solução para o problema é algo que comprovadamente funciona[1]. Sendo utilizados na DOL, há

[1]Um padrão de projeto é aceito se funciona em projetos diferentes [Gamma et al. 1995]

os padrões: *Bridge*, *Strategy*, *Singleton*, *Observer* e *Template*.

Como parte central da biblioteca, há a classe *Core*. A responsabilidade dessa classe é restrita a carregar para a memória principal um conjunto de dados gravado em um arquivo na sintaxe padrão *Discover Standard Syntax*[2] – DSX. Com os dados disponíveis em memória, a classe *Core* fornece acesso a esses dados além de algumas manipulações básicas. Por fim, existem métodos para gravar os dados manipulados em arquivos texto em diversas sintaxes. Entre as manipulações que podem ser realizadas, pode-se listar: os métodos que modificam e adicionam exemplos; removem e adicionam atributos; manipulam o tipo, nome e a posição dos atributos; compõem novos atributos a partir de expressões lógicas ou aritméticas; removem atributos ou ignoram-os logicamente; entre muitos outros.

A partir da classe *Core* podem ser acopladas outras classes que provêm novas funcionalidades. Alguns exemplos dessas classes da biblioteca DOL são:

***Filter*:** permite filtrar temporariamente exemplos de um objeto *Core* a partir de expressões lógicas;

***Resampling*:** dividem os exemplos de um objeto *Cor*e em conjuntos de treinamento e teste. Existem diversas classes *Resampling*, incluindo as que implementam $k$*-fold cross-validation* simples e estratificado;

***Sample*:** realiza amostras aleatórias ou estratificas dos dados em um objeto *Core*;

***BasicStats*:** provê um conjunto de estatísticas descritivas calculadas sobre os dados presentes no objeto *Core*;

***Normalize*:** normaliza atributos numéricos em uma determinada faixa de valores;

***RunInducer*:** executa um indutor utilizando os dados contidos em um objeto *Core*;

***UnderSampling*:** aplica métodos de *under-sampling* para reduzir o número de exemplos da classe majoritária com o objetivo de balancear o conjunto de dados;

***OverSampling*:** aplica métodos de *over-sampling* para aumentar o número de exemplos da classe minoritária com objetivo de balancear o conjunto de dados;

***TreatMissing*:** trata valores desconhecidos utilizando diversas técnicas de imputação;

***Nearest Neighbor*:** calcula os $k$ vizinhos mais próximos de um determinado exemplo.

[2] `http://www.icmc.usp.br/~gbatista/sintaxe_padrao_final.html`.

Em números, a biblioteca DOL é composta por cerca de 115 classes, tendo cerca de 15 classes abstratas. Detalhes sobre o projeto da biblioteca, como diagramas de classe, podem ser encontrados em [Batista and Monard 2003].

## 4. Interface com Indutores

Como mencionado anteriormente, na DOL não é reimplementado nenhum algoritmo de indução. Tal tarefa é atribuída a programas externos, cuja implementação é a original do autor do método, disponibilizada na *Internet*. Atualmente, estão integrados com a biblioteca DOL os seguintes indutores: *C4.5*, *C4.5 rules*, *CN2*, *Ripper*, *Libsvm*, *SVM Torch*, *Weka*, *JNNS*, além das implementações dos algoritmos *Apriori*, *Redes Neurais* e *Naive Bayes* realizadas por Christian Borgelt[3]. Embora essa abordagem facilite a reprodução, por outros pesquisadores, dos experimentos executados com a DOL, cada um desses sistemas de aprendizado utiliza um formato proprietário para arquivos de entrada e saída, o que cria a necessidade de converter os dados para a sintaxe do indutor.

Dessa forma, além de interagir com esses programas, a DOL possui uma interface para cada um dos algoritmos indutores. Essas interfaces estão implementadas em um conjunto de classes chamadas *RunInducer*. Cada classe *RunInducer* é responsável por executar um indutor específico sobre dados contidos em objetos *Core*. De uma forma resumida, os objetos *RunInducer* realizam os seguintes passos:

1. Objetos *Core* são associados a um objeto *RunInducer* específico. Existem diversas especializações da classe *RunInducer* implementadas, uma para cada indutor integrado à biblioteca DOL. Podem ser associados até três objetos *Core*. O primeiro deve conter os dados de treinamento, o segundo os dados de teste e o terceiro os dados de validação. Se somente o primeiro objeto for associado, então um modelo é criado, mas não há avaliação;
2. Caso haja alguma necessidade imposta pelo indutor, os dados são pré-processados. Por exemplo, para o uso de redes neurais e máquinas de suporte vetorial, os atributos qualitativos são transformados para quantitativos por meio da codificação *um-de-n* (*one-of-n encoding*);
3. Os dados dos objetos *Core* associados são gravados em arquivos na sintaxe requerida pelo indutor;
4. O indutor é executado por meio de uma chamada ao sistema operacional;

[3] `http://www.borgelt.net/software.html`.

5. Se um objeto *Core* com casos de teste foi associado, *RunInducer* analisa a saída do indutor, e extrai informações como matrizes de confusão nos casos de teste;
6. Se o indutor é simbólico, então o modelo gerado pelo indutor é convertido para o formato PBM e gravado em um arquivo em disco;
7. Se o problema analisado possui duas classes, então é realizada uma análise ROC e são calculadas diversas medidas de desempenho incluindo AUC;
8. Os resultados são reportados por meio de um arquivo texto com todas as medidas calculadas.

Atualmente, o projeto Discover disponibiliza o sistema gerenciador de experimentos *Sniffer* [Batista and Monard 2004]. Esse sistema é responsável por organizar grandes experimentos, ou seja, experimentos que envolvem um grande número de conjuntos de dados, métodos avaliados, variações de parâmetros de execução e assim por diante. O sistema *Sniffer* divide um conjunto de dados em conjuntos de treinamento e teste, executa um conjunto de indutores sobre os dados e realiza comparações por meio de testes de estatísticos para identificar diferenças significativas entre os desempenhos dos métodos avaliados. As classes *RunInducer* e o sistema *Sniffer* constituem a base de integração entre a bibliteca DOL e os sistemas indutores desenvolvidos por outros pesquisadores.

## 5. Desenvolvimentos Recentes

A DOL possui uma arquitetura extensível e orientada a objetos, mas para isso deve-se pagar um preço: a perda de desempenho. Uma sobrecarga é gerada devido a diversos fatores, como vários níveis de herança, passagens de mensagens entre as classes, etc. Com isso, métodos que poderiam ser simples, fazem tarefas complexas. Na maior parte das vezes, a arquitetura complexa reflete em maior uso do processador e da memória principal.

Como os experimentos freqüentemente precisam ser executados dezenas de vezes, variando-se pequenos parâmetros de configuração, uma pequena sobrecarga pode gerar um aumento significativo no tempo de processamento computacional. Dessa forma, surgiu a necessidade de melhorar o desempenho da biblioteca.

Após o desenvolvimento da biblioteca, pôde-se analisar quais funcionalidades eram mais utilizadas. Ainda, outras funcionalidades eram pouco utilizadas e, dentre essas funcionalidades pôde-se identificar algumas que, se removidas ou restringidas, poderiam levar a um ganho de desempenho da biblioteca DOL. Dessa

forma, decidiu-se, por motivos de compatibilidade, manter o desenvolvimento da DOL e criar uma nova biblioteca, simplificada, chamada *Simplified Discover Object Library* – SDOL.

Algumas funcionalidades que poderiam ser removidas estavam associadas à sintaxe DSX. Dessa forma, visando tornar esse processo de leitura mais eficiente, foi proposta uma sintaxe DSX simplificada. A *Simplified Discover Standard Syntax* – SDSX[4] retirou algumas características pouco utilizadas da DSX, perdendo em flexibilidade, porém visando um ganho no desempenho da interpretação dos arquivos de dados. As funcionalidades retiradas foram:

**Atributos virtuais:** são atributos que devem ser calculados durante a carga dos dados em um objeto *Core*. Eles são definidos por meio de expressões lógicas ou aritméticas que envolvem outros atributos. Tais atributos aumentam o tempo de carga dos dados pois envolvem a utilização de um meta-interpretador da linguagem *Perl*;

**Tipo nominal sem valores:** tipo de dados nominal descreve um atributo que pode assumir apenas um valor dentro de uma lista de valores válidos. Por exemplo, um atributo tamanho poderia assumir apenas os valores *pequeno*, *médio* e *grande*. Originalmente, a sintaxe DSX permitia declarar um atributo nominal sem informar a lista de valores válidos. Dessa forma, todos os valores que ocorriam para esse atributo nos dados eram considerados válidos. Entretanto, quando era necessário verificar quais valores ocorriam para esse atributo, era necessário realizar uma passagem completa sobre os dados. A sintaxe DSX restrigiu a declaração do tipo nominal exigindo que todos os valores que o atributo pode assumir sejam informados.

Através dessa nova sintaxe e de algumas melhorias no interpretador dos arquivos de dados, foi possível auferir um ganho de aproximadamente 8% no tempo de carregamento dos dados para um objeto *Core*. Conseguir esse ganho só foi possível através de um trabalho intenso de *benchmarks*, otimizações no código-fonte e na estrutura de classes. Para medir os ganhos de desempenho foram realizados experimentos nos quais os tempos de processamento da SDOL foram comparados com os tempos de processamento da versão anterior. Os experimentos foram realizandos utilizando alguns conjuntos de dados artificiais, e os experimentos foram repetidos diversas vezes para garantir a confiabilidade dos resultados.

Além das otimizações no *parser* dos arquivos de dados, a classe *Core* está sendo reprojetada para gerar uma classe *SCore*, com menos funcionalidades

[4] http://grad.icmc.usp.br/∼lucasrosad/SintaxeSimpleDSX.html

e mais rápida. Nesse projeto, o modo de armazenamento e acesso dos dados foi modificado e, além disso, os métodos que acessam a estrutura de dados receberam algumas simplificações no código. Em testes preliminares utilizando o algoritmo *k-Vizinhos Mais Próximos*, foi comprovada uma melhoria da ordem de 40% no desempenho geral.

Estão sendo feitas melhorias na qualidade do código, bem como estão sendo criadas dezenas de rotinas de testes automáticos, que visam garantir o funcionamento correto dos módulos após sofrerem manutenção. Basicamente, o processo consiste em escrever um teste de entradas e saídas de um método ou classe antes de sofrer otimizações, pois esse é o comportamento que se espera após a manutenção, mas com velocidade maior. Assim, depois que o método é otimizado, o teste é executado e verifica-se se as saídas condizem com a versão antiga. Esse processo reduz o tempo de desenvolvimento e o aparecimento de erros de codificação na biblioteca.

Tanto a DOL quanto a SDOL estão sendo desenvolvidas utilizando a linguagem *Perl* [Wall et al. 1996], linguagem escolhida por permitir desenvolver protótipos de forma rápida, bem como pela portabilidade e compatibiliade com outras ferramentas já desenvolvidas no laboratório. Além disso, no desenvolvimento da SDOL, a linguagem *Perl* foi mantida para que não houvesse a necessidade de reescrever todo o código já produzido, fato que iria dispender muito tempo e possivelmente introduzir novos defeitos na biblioteca.

## 6. Conclusão

Apesar do projeto ainda estar em fase de desenvolvimento, os resultados obtidos até o momento com a SDOL são bastante promissores e apontam para uma futura implementação mais eficiente. Como a biblioteca DOL é utilizada como base para o desenvolvimento de diversos sistemas, um melhor desempenho da nova biblioteca deve ser refletido nesses sistemas. Pode-se citar como exemplos de sistemas construídos sobre a DOL os ambientes *NNRules* [Milaré 2003] para a extração de conhecimento simbólico a partir de Redes Neurais; o algoritmo *Co-Training* [Matsubara 2004] para aprendizado semi-supervisionado; o ambiente GAESC [Bernardini 2006] para evolução de classificadores utilizando algoritmos genéticos e o ambiente ECLE [Pila 2007] para a indução de regras com propriedades específicas.

Os resultados experimentais apresentados ainda são preliminares. É necessário executar um conjunto maior de experimentos, e avaliar os ganhos de desempenho em diferentes situações, incluindo conjuntos de dados de tamanhos

diferentes e diversos tipos de tarefas de processamento de dados. Existem ainda outros melhoramentos a serem feitos os quais são bastante requisitados pela comunidade, como um instalador semi-automático, documentação das diversas classes e arquitetura da biblioteca, além de exemplos que demonstram como utilizar diversos recursos.

Por fim, as biblioteca DOL e SDOL podem ser utilizadas no processamento de dados temporais ou sequenciais. É objetivo desta pesquisa investigar alguns métodos de pré-processamento de dados temporais e sequenciais que serão integrados à biblioteca. Esses métodos devem realizar tarefas comuns como a remoção de ruídos e a normalização de dados.

**Agradecimentos:** Os autores agradecem ao revisor anônimo pelas sugestões realizadas sobre a versão preliminar deste artigo.

## Referências

Batista, G. E. and Monard, M. C. (2004). Sniffer: um Ambiente Computacional para Gerenciamento de Experimentos de Aprendizado de Máquina Supervisionado. In *I WorkComp Sul*, pages 13–24.

Batista, G. E. A. P. A. (2003). Pré-processamento de Dados em Aprendizado de Máquina Supervisionado. Tese de Doutorado, ICMC-USP.

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.

Bernardini, F. C. (2006). Combinação de Classificadores Simbólicos Utilizando Medidas de Regras de Conhecimento e Algoritmos Genéticos. Tese de Doutorado, ICMC-USP.

Fawcett, T. (2004). Roc graphs: Notes and practical considerations for researchers. Technical report, HP Labs Tech Report HPL-2003-4.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). Knowledge Discovery and Data Mining: Towards a Unifying Framework. In *Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 82–88.

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

Lavrač, N., Flach, P. A., and Zupan, B. (1999). Rule Evaluation Measures: A Unifying View. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming*, pages 174–185.

Matsubara, E. T. (2004). O Algoritmo de Aprendizado Semi-supervisionado Co-training e sua Aplicação na Rotulação de Documentos. Dissertação de Mestrado, ICMC-USP.

Milaré, C. R. (2003). Extração de Conhecimento de Redes Neurais Artificiais utilizando sistemas de Aprendizado Simbólico e Algoritmos Genéticos. Tese de Doutorado, ICMC-USP.

Pila, A. D. (2007). Computação Evolutiva para a Construção de Regras de Conhecimento com Propriedades Específicas . Tese de Doutorado, ICMC-USP.

Prati, R. C. (2003). O *Framework* de Integração do Sistema DISCOVER. Dissertação de Mestrado, ICMC-USP.

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.

Rezende, S. O. (2003). *Sistemas Inteligentes: Fundamentos e Aplicações*. Editora Manole, Barueri, SP, Brasil.

Todorovski, L. and Džeroski, S. (2003). Combining Classifiers with Meta Decision Trees. *Machine Learning*, 50:223–249.

Wall, L., Christiansen, T., and Schwartz, R. L. (1996). *Programming Perl*. O'Reilly & Associates, 2 edition.

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